Universidad Zaragoza
1542

Trabajo Fin de Grado

Implementación de efectos digitales de audio en tiempo real utilizando un microprocesador

Implementation of real-time digital audio effects using a microprocessor

Autor

Luis Llorente Muro

Director

Isidro Urriza Parroqué

Departamento de Ingeniería Electrónica y Comunicaciones

Escuela de Ingeniería y Arquitectura

2021-2022

En este futuro brillante no puedes olvidar tu pasado.

BOB MARLEY,

Cantante y compositor jamaicano.

A mis padres, a mi bredda y a Isidro.

Muchas gracias.

RESUMEN

La historia de los DSPs (Digital Signal Processors), que comenzaron como un simple tema de investigación a mediados de los 60, es un hito que ha revolucionado industria y sociedad por igual. Los conocimientos avanzados en la teoría del Procesado Digital de Señales, junto con el desarrollo tecnológico que habían experimentado los Circuitos Integrados, dieron lugar a los primeros microprocesadores, mejorando el rendimiento notablemente, reduciendo el coste, tamaño y disipación de energía.

Este trabajo se centra en desarrollar una plataforma software que permite implementar en un microprocesador (cortex-M) algoritmos de procesado de audio en tiempo real.

En primer lugar, se han implementado y ajustado los distintos algoritmos de procesado de audio con la ayuda de MATLAB® y Simulink® para, posteriormente, implementarlos en coma fija en C. A continuación, se ha desarrollado una plataforma software escalable que reúne dichos algoritmos y que permite al usuario comunicarse con el microprocesador desde el Personal Computer (PC) a través de un puerto serie. De este modo, el microprocesador controla los distintos efectos que el usuario demande a través del sistema de desarrollo FM4-176L-S6E2CC-ETH.

ABSTRACT

DSPs history, which began as a simple research topic in the mid-1960s, is a landmark that has revolutionized industry and society alike. Advanced knowledge in Digital Signal Processing theory, together with the technological development that Integrated Circuits had undergone, gave rise to the first microprocessors, significantly improving performance, reducing cost, size, and energy dissipation.

This work focuses on developing a software platform that allows real-time audio processing algorithms to be implemented in a microprocessor (cortex-M).

In the first place, the different audio processing algorithms have been implemented and adjusted with the help of MATLAB® y Simulink® to subsequently implement them in fixed point in C. Next, a scalable software platform that brings together these algorithms has been developed. This platform also allows the user to communicate with the microprocessor from the Personal Computer (PC) through a serial port. In this way, the microprocessor controls the different effects that the user requests through the FM4-176L-S6E2CC-ETH device.

ÍNDICE DE CONTENIDOS

LISTA DE ABREVIATURAS Y ACRÓNIMOS

DSP	Digital Signal Processor
PC	Personal Computer
GUI	Graphical User Interface
ADC	Analog-Digital Converter
DAC	Digital-Analog Converter
RTA	Real-time Analyzer
VSTi	Virtual Studio Technology Instrument
GCC	GNU Compiler Collection

1. INTRODUCCIÓN

1.1. Contexto actual

En las cuatro últimas décadas, con el auge de la música electrónica, a inicios de los años 60, el campo del audio y los efectos digitales han experimentado un notable desarrollo. La mayoría de los dispositivos utilizados en este sector, como sintetizadores, pedaleras para guitarra o mesas de mezclas, tienen en común la utilización de técnicas de procesado digital de señal en tiempo real. Para ello, requieren de una frecuencia de muestreo relativamente elevada, de microprocesadores extremadamente rápidos y eficientes, de código optimizado, y de grandes cantidades de memoria.

Este trabajo se ha centrado en el desarrollo de una librería de efectos digitales de audio en C [1], que puede ejecutarse en un microprocesador, siguiendo una metodología que permite añadir más efectos con facilidad, si se desea en el futuro.

Los efectos analógicos son los más populares entre los más entendidos, debido a que logran un sonido más puro y de mayor calidad. Esto sucede, por ejemplo, porque no se pierde precisión en el proceso de conversión analógico-digital [2] y digital-analógico (Figura 1) o en las operaciones en coma fija realizadas en el código.

Figura 1. Diagrama de bloques de un sistema digital genérico.

Sin embargo, para la gente que no ha desarrollado un oído entrenado es imperceptible la diferencia entre un efecto analógico y uno digital. De esta forma, una pedalera multiefectos digital sería mucho más práctica a la hora de añadir efectos nuevos, o realizar pequeñas variaciones en el código original que permitan enfatizar ciertos parámetros del efecto.

1.2. Efectos de sonido

Los efectos de audio modifican el sonido original de un instrumento en base a una serie de parámetros controlados por el usuario mediante una GUI (Figura 2), para dotarlo de características adicionales a las que éste poseía inicialmente, que enriquezcan la variedad de sonidos que puede producir.

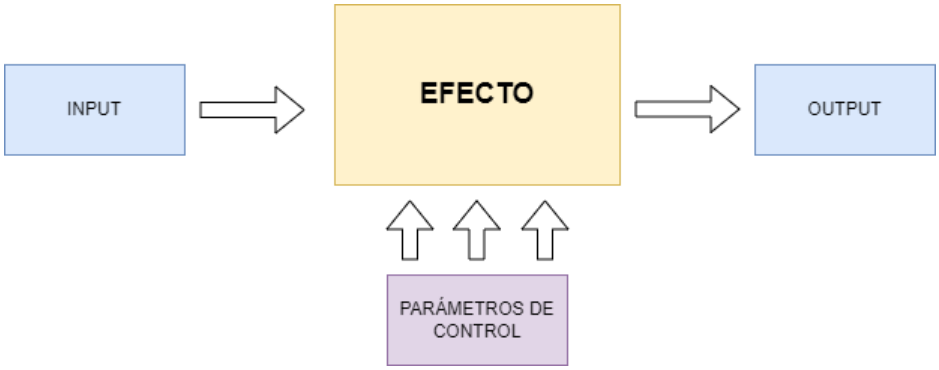
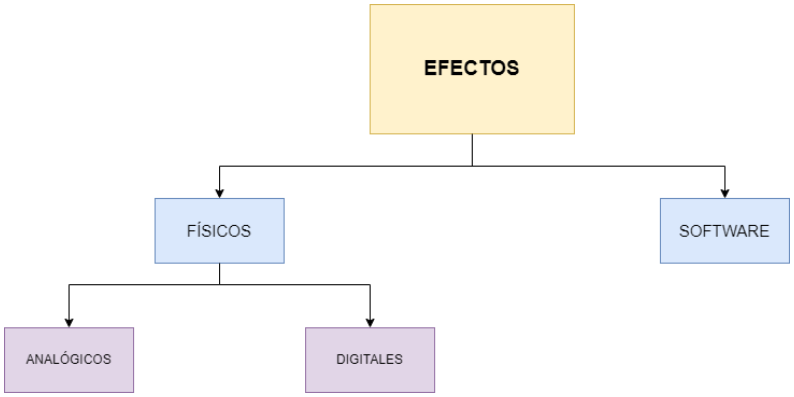


Figura 2. Diagrama de bloques de la aplicación de un efecto de audio.

Normalmente, la mayoría de los efectos se suelen aplicar en tiempo real, bien sea en conciertos en vivo o en el proceso de grabación de un disco. En concreto, en la fase de mezcla y masterización en el proceso de postproducción.

Los efectos se pueden dividir en dos grandes subgrupos. Efectos físicos, a su vez divididos en analógicos y digitales, y efectos software (Esquema 1). Este trabajo se ha centrado en la implementación de efectos digitales.



Esquema 1. Clasificación de efectos de audio según el tipo.

Los efectos en formato físico se suelen aplicar en tiempo real, generalmente en instrumentos como la guitarra eléctrica o el bajo. Los usuarios los encuentran disponibles como pedales individuales (Figura

3, izquierda) o como pedaleras multiefectos (Figura 3, derecha), que contienen una agrupación de pedales individuales.

Figura 3. Pedal individual de guitarra (izquierda) [3] y pedalera multiefectos analógica (derecha) [4].

A su vez, los efectos en físicos se subdividen en analógicos y digitales. El primer grupo está compuesto por circuitería electrónica analógica o por elementos electromecánicos.

Por otro lado, los efectos digitales, como su nombre indica, están compuestos por circuitería digital. Este tipo de efectos se suelen comercializar en formato multiefectos (Figura 4), permitiendo así una gran variedad de efectos distintos en el mismo dispositivo (*Flanger*, *Chorus*, *Wah-wah*…)

Figura 4. Pedalera multiefectos digital [5].

Por último, los efectos software han experimentado un desarrollo exponencial en la última década. Antiguamente era indispensable acudir a un estudio de grabación si se querían obtener unos resultados lo más profesionales posibles. En la actualidad, basta con tener un PC, una tarjeta de sonido potente y herramientas de edición de audio como *Cubase*® [6] o *Pro-Tools*® [7]. Este tipo de software suele llevar incorporado un paquete de efectos con los que el PC modifica la señal digital según las necesidades del usuario. Además, junto con estas herramientas se pueden utilizar *Real-time Analyzer* (RTAs) [8] (Figura 5, izquierda) o *Virtual Studio Technology Instrument* (VSTi) [9] (Figura 5,

derecha), efectos musicales en formato software, para lograr un sonido profesional inimaginable hace unos pocos años.

Figura 5. Ejemplos gráficos de un RTA (izquierda) [10] y de un VSTi (derecha) [11].

1.3. Planteamiento y objetivos

El objetivo de este trabajo es desarrollar una plataforma *software* que permita implementar en un microprocesador (cortex-M) algoritmos de procesado de audio en tiempo real en C (Figura 6). Además, dicha plataforma deberá ser escalable, permitiendo añadir nuevos efectos con facilidad.

Los distintos algoritmos se implementan en coma fija, obteniendo un ahorro en la simplicidad del microprocesador, además de un ahorro de energía, a cambio de aumentar el tiempo de desarrollo.

Por último, se deberá crear un *testbench* que permita verificar el correcto funcionamiento de dicha plataforma.

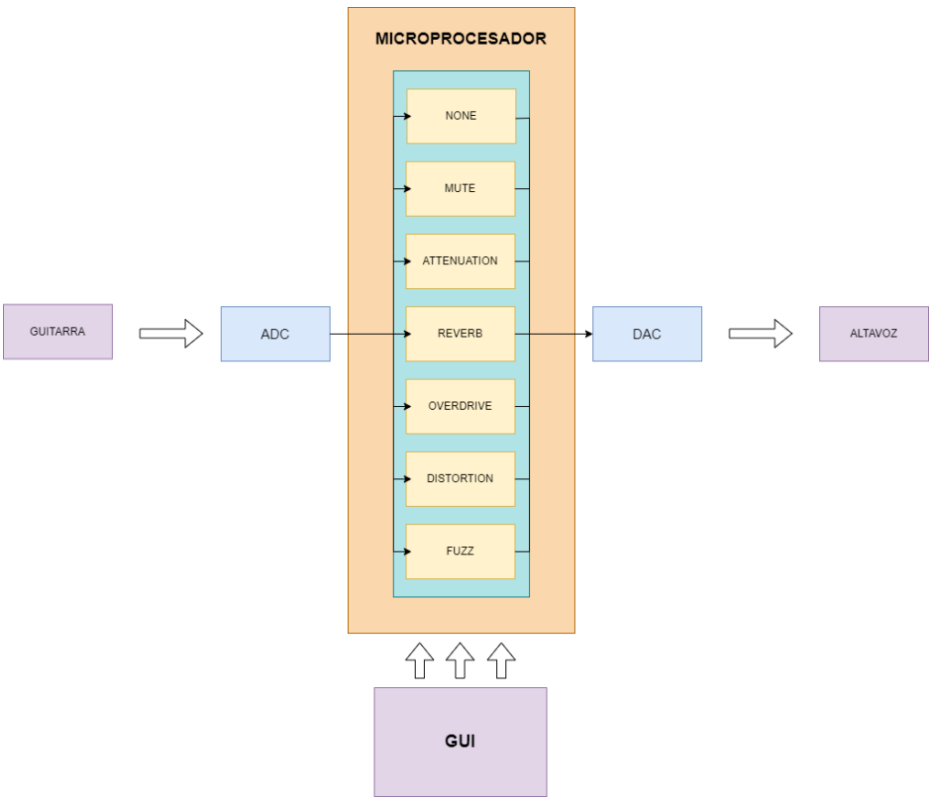


Figura 6. Diagrama de bloques de los efectos implementados en este trabajo.

1.4. Alcance

El sistema estará compuesto por los siguientes bloques: el microprocesador cortex-M4, el software ejecutado por el microprocesador y una GUI para establecer los distintos parámetros de los distintos efectos desde un PC.

Además, tendrá las siguientes características:

- La programación del microprocesador se realizará en lenguaje C, y comprenderá la gestión de las comunicaciones con el PC, así como de las distintas interrupciones.
- La interfaz gráfica de usuario se implementará en MATLAB®, y será la encargada de gestionar los distintos parámetros de los distintos efectos que el usuario demande.
- Se documentará el proceso de desarrollo del sistema, así como las características y funciones más relevantes.

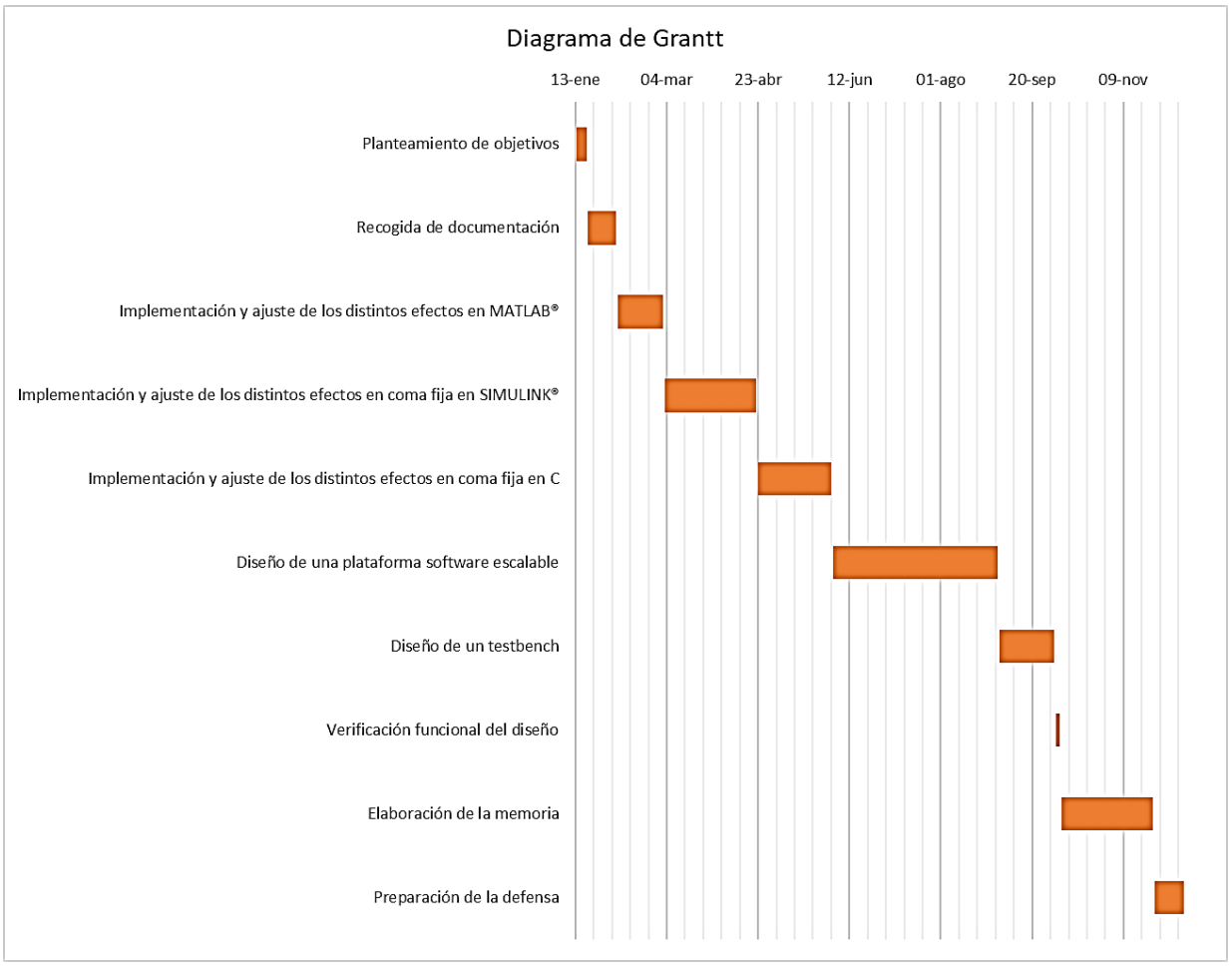
Por último, el correcto funcionamiento del sistema será verificado mediante simulación e implementación experimental.

1.5. Metodología

La metodología llevada a cabo a lo largo de este trabajo puede estructurarse en seis bloques principales:

a. Implementación y ajuste de los distintos efectos en MATLAB®,
b. implementación y ajuste de los distintos efectos en coma fija en Simulink®,
c. implementación y ajuste de los distintos efectos en coma fija en C, utilizando GNU Compiler Collection (GCC) [12],
d. diseño de una plataforma software ejecutable en el microprocesador, utilizando GCC,
e. implementación de un *testbench* en ARM Keil µVision® IDE [13]
f. y verificación funcional del diseño con el sistema de desarrollo FM4-176L-S6E2CC-ETH [14].

1.6. Planificación



Esquema 2. Planificación de este trabajo.

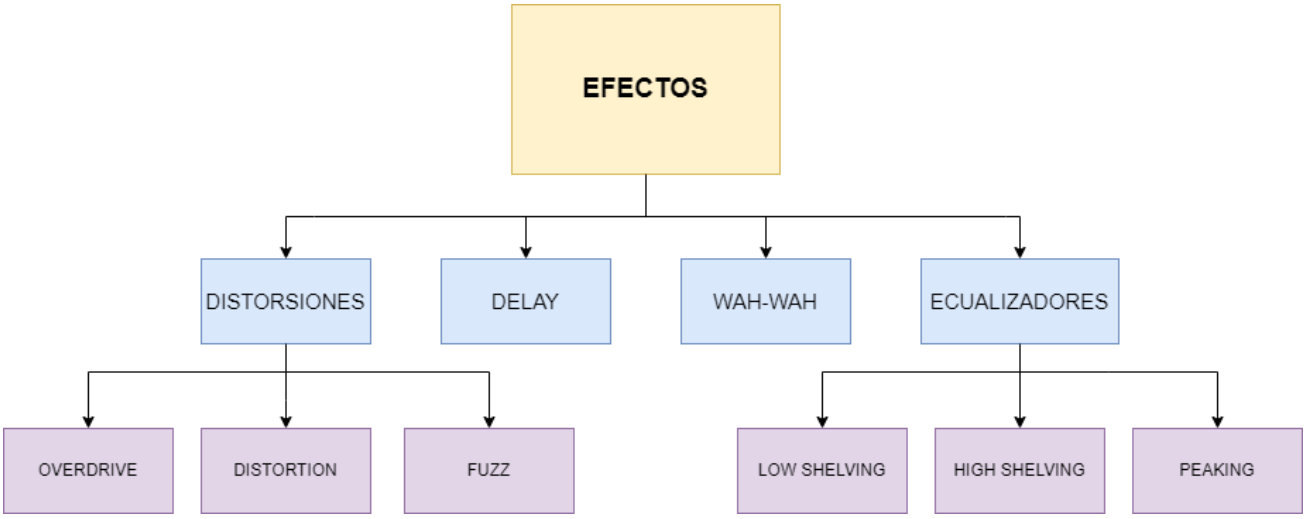
1.7. Estructura de la memoria

La presente memoria pretende exponer de manera clara y concisa el trabajo desarrollado, así como los resultados y las conclusiones alcanzadas a lo largo de este trabajo.

En el primer capítulo se introducen las características principales de los efectos de sonido y su relación con este proyecto, así como la motivación y los objetivos pretendidos, junto con una pequeña descripción de las tareas realizadas. En el segundo, se presenta una descripción detallada de la totalidad de los efectos implementados. En el tercer capítulo se presenta la implementación en coma fija de los distintos efectos utilizando la metodología descrita previamente. En el cuarto, se describe la librería de efectos implementada y las instrucciones para añadir nuevos efectos. En el quinto capítulo se detalla el diseño de un *testbench* que permita verificar funcionalmente el correcto comportamiento del diseño. En el sexto capítulo se explican las conclusiones obtenidas y las posibles líneas de investigación futuras. Por último, adjuntos al documento principal, se incluyen varios anexos con los distintos códigos y esquemas utilizados a lo largo de este trabajo.

2. DESCRIPCIÓN DE LOS DISTINTOS EFECTOS EN MATLAB®

En este capítulo se describen en profundidad los distintos efectos implementados en este trabajo, agrupados como se muestra en el Esquema 3.



Esquema 3. Clasificación de los distintos efectos implementados en este trabajo.

2.1. Distorsiones

La distorsión, uno de los efectos más extendidos entre los guitarristas, se produce al aumentar la ganancia de la señal de entrada hasta el punto en el que se satura a su máximo valor (Figura 7).

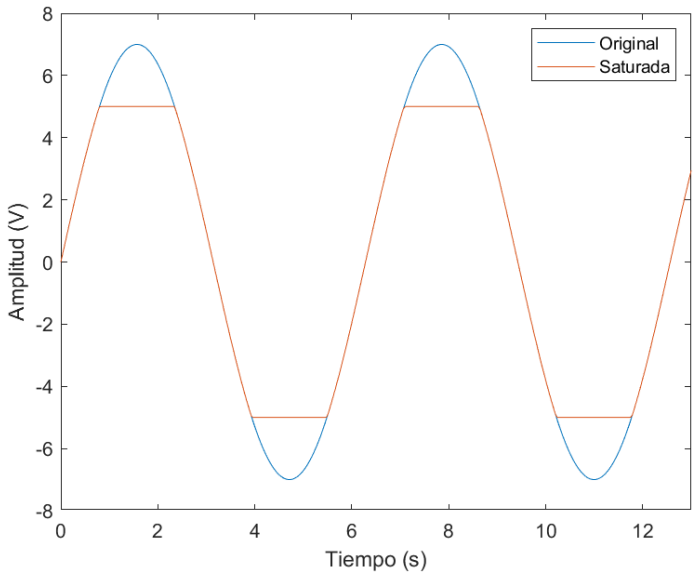


Figura 7. Comparativa entre la señal original y la señal saturada a 5 V.

Existen dos tipos de saturación, básicamente. *Soft Clipping*, donde se produce una transición más suave desde la parte normal de la sinusoide a la parte saturada, y *Hard Clipping*, donde la transición es mucho más brusca (Figura 8) [15].

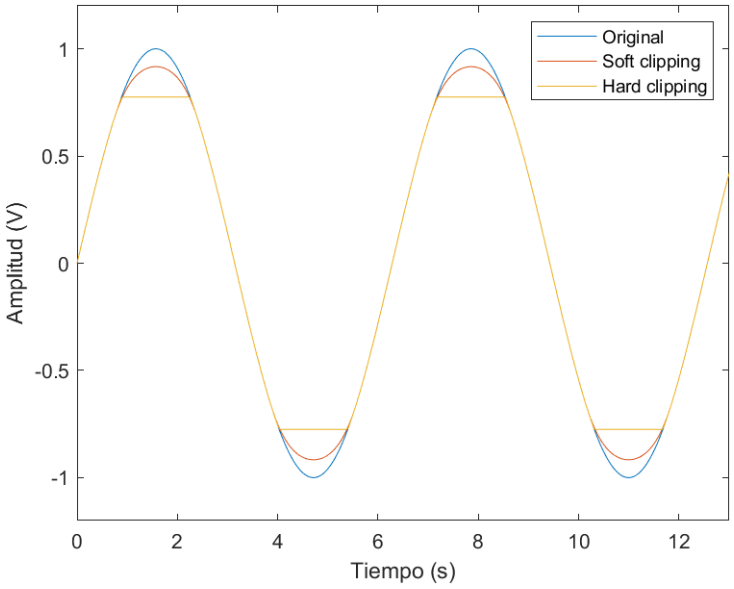


Figura 8. Comparativa entre *Soft Clipping* y *Hard Clipping*.

Por otro lado, la saturación puede ser simétrica o asimétrica (Figura 9) [16].

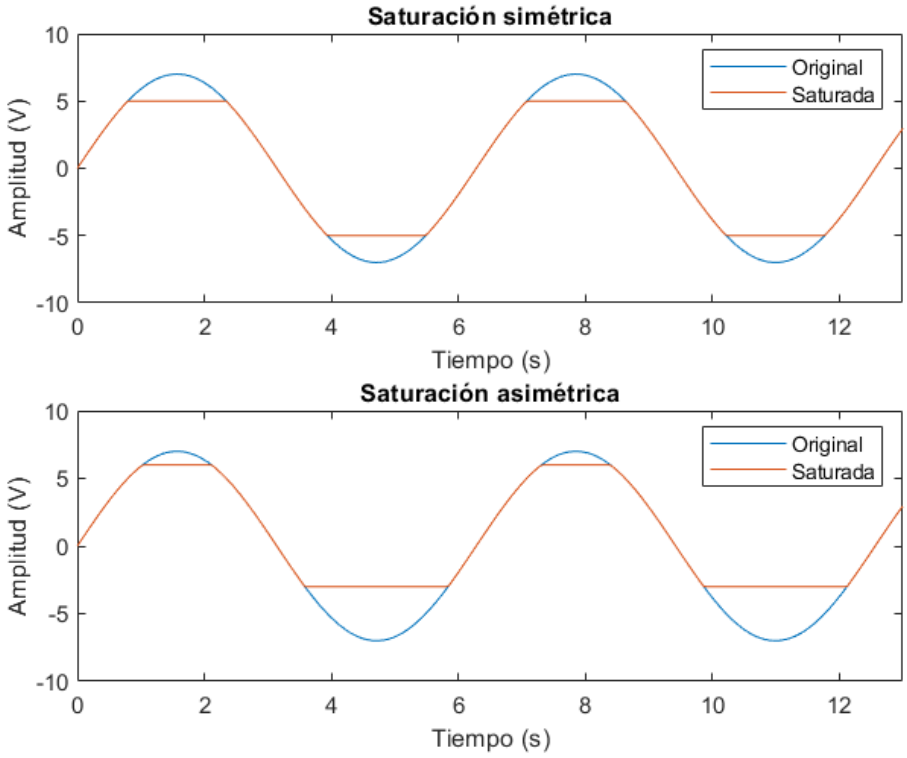


Figura 9. Comparativa entre saturación simétrica y asimétrica.

Las distorsiones más popularizadas son el *Overdrive*, *Distortion* y *Fuzz* [17].

2.1.1. *Overdrive*

En las distorsiones *overdrive* se hace uso de una saturación suave simétrica (*Symmetrical Soft Clipping*), descrita bajo la siguiente ecuación (Ecuación 1) [18].

$$f(x) = \begin{cases} 2x & 0 \leq x \leq \frac{1}{3} \\ \frac{3-(2-3x)^2}{3} & \frac{1}{3} \leq x \leq \frac{2}{3} \\ 1 & \frac{2}{3} < x < 1 \end{cases}$$

Ecuación 1. Ecuación para conseguir el efecto *overdrive*.

Por comodidad, se ha operado con la expresión dada para los valores comprendidos entre $^1/_2$ y $^1/_3$, obteniendo la Ecuación 2.

$$f(x) = \begin{cases} 2x & 0 \leq x \leq \frac{1}{3} \\ -\frac{1}{3} - 3x^2 + 4x & \frac{1}{3} \leq x \leq \frac{2}{3} \\ 1 & \frac{2}{3} < x < 1 \end{cases}$$

Ecuación 2. Ecuación simplificada para conseguir el efecto *overdrive*.

El *overdrive*, a diferencia del resto de efectos que se comentan a lo largo de este trabajo, no tiene ningún parámetro de entrada que pueda controlar el usuario. Simplemente se hace uso de una saturación suave simétrica cuya curva característica se observa en la Figura 10.

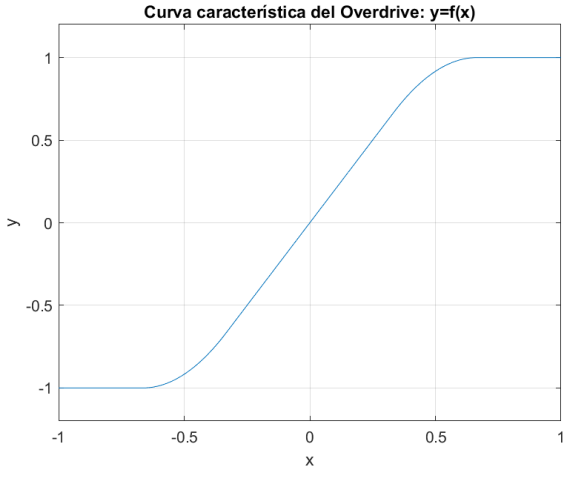


Figura 10. Curva característica del *overdrive*.

Hasta el umbral, 1/3 en este caso, simplemente se multiplica por dos la señal de entrada, manteniéndose la curva característica en su región lineal. Entre 1/3 y 2/3, la curva característica produce una

compresión suave, descrita por el término central de la Ecuación 1. Por último, para valores por encima de 2/3, la salida se fija a 1.

2.1.2. *Distortion*

Para el efecto *distortion*, se puede recurrir a la siguiente expresión no lineal (Ecuación 3) [19].

$$f(x) = sgn(x)\ (1 - e^{-|ax|})$$

Ecuación 3. Ecuación de la distorsión exponencial.

La curva característica, un poco más brusca que la del *Overdrive*, se ilustra en la Figura 11.

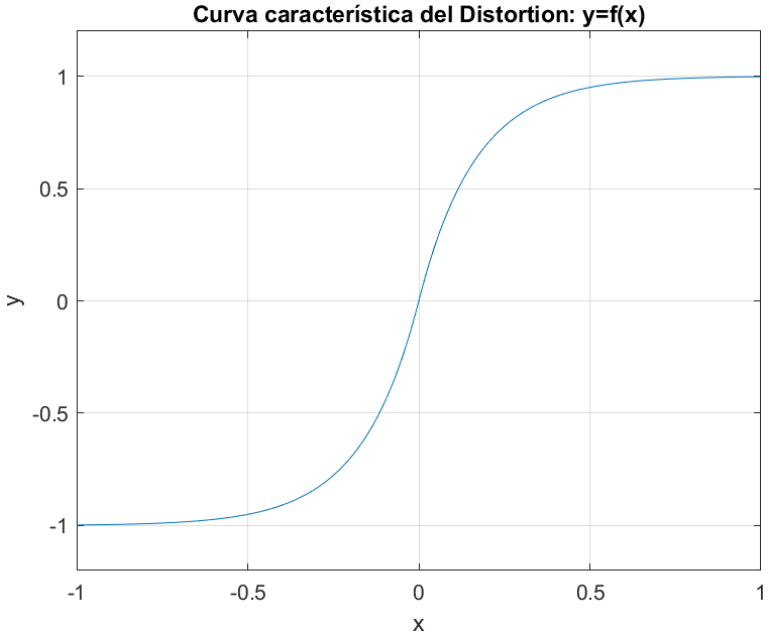


Figura 11. Curva característica de la distorsión exponencial.

2.1.3. *Fuzz*

Una de las representaciones más extendidas donde se hace uso de una saturación brusca es el efecto *Fuzz Face* [20], utilizado por Jimi Hendrix.

Dicho efecto se basa en la expresión no lineal detallada en la Ecuación 4 [21].

$$f(x) = \frac{x}{|x|}\ (1 - e^{\frac{ax^2}{|x|}})$$

Ecuación 4. Ecuación del efecto *Fuzz*.

En la Figura 12. Curva característica de la distorsión *Fuzz*. se representa la curva característica del *Fuzz*, muy similar a la del *Distortion*, pero reflejada.

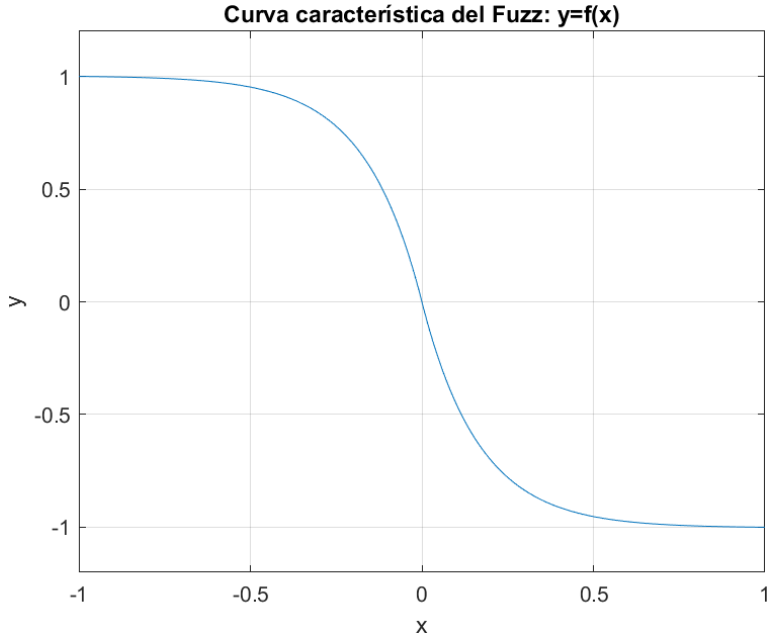


Figura 12. Curva característica de la distorsión *Fuzz*.

2.2. Ecualizadores

A diferencia de los filtros paso bajo, paso alto, paso banda y banda eliminada que atenúan el espectro de audio por encima o por debajo de una determinada frecuencia de corte, los ecualizadores dan forma al espectro de audio aumentando ciertas bandas de frecuencia, pero dejando otras intactas [22]. Dentro de los distintos tipos de ecualizadores existentes, este trabajo se centra en ecualizadores de primer orden (*low shelving* y *high shelving*), y en un ecualizador de segundo orden (*peaking*).

2.2.1. *Low shelving/High shelving*

Este tipo de ecualizadores de primer orden se construyen utilizando un paso todo de primer orden [23], cuya función de transferencia se describe en la Ecuación 5, donde $C_{B/C}$ son los parámetros de la frecuencia de corte para aumentar (C_B) o cortar (C_C).

$$A(z) = \frac{z^{-1} + C_{B/C}}{1 + z^{-1}C_{B/C}}$$

Ecuación 5. Función de transferencia del paso todo empleado.

Para un *low shelving*, dichos parámetros se pueden calcular empleando la Ecuación 6 (a) [23], mientras que para un *high shelving* se calculan con la Ecuación 6 (b). En estas expresiones, $\frac{f_c}{f_s}$ corresponde a la mitad de la frecuencia de corte normalizada, parámetro que junto con la ganancia es controlado por el usuario. Por otro lado, V_0 es la ganancia pasada de dB a lineal (Ecuación 6 (c)).

$$C_B = \frac{\tan(\frac{\pi f_c}{f_s} - 1)}{\tan(\frac{\pi f_c}{f_s} + 1)} \qquad C_B = \frac{\tan(\frac{\pi f_c}{f_s} - 1)}{\tan(\frac{\pi f_c}{f_s} + 1)} \qquad V_0 = 10^{G/20}$$

$$C_C = \frac{\tan(\frac{\pi f_c}{f_s} - V_0)}{\tan(\frac{\pi f_c}{f_s} + V_0)} \qquad C_C = \frac{V_0 \tan(\frac{\pi f_c}{f_s} - 1)}{V_0 \tan(\frac{\pi f_c}{f_s} + 1)}$$

(a) **(b)** **(c)**

Ecuación 6. (a) parámetros *boost/cut* del *low shleving*, (b) parámetros *boost/cut* del *high shelving* y (c) conversión de la ganancia (dB) a lineal.

Una vez aclarados estos detalles, tenemos el filtro *low shelving/high shelving* en la Figura 13, cuya función de transferencia se detalla en la Ecuación 7, donde $H_0 = V_0 - 1$.

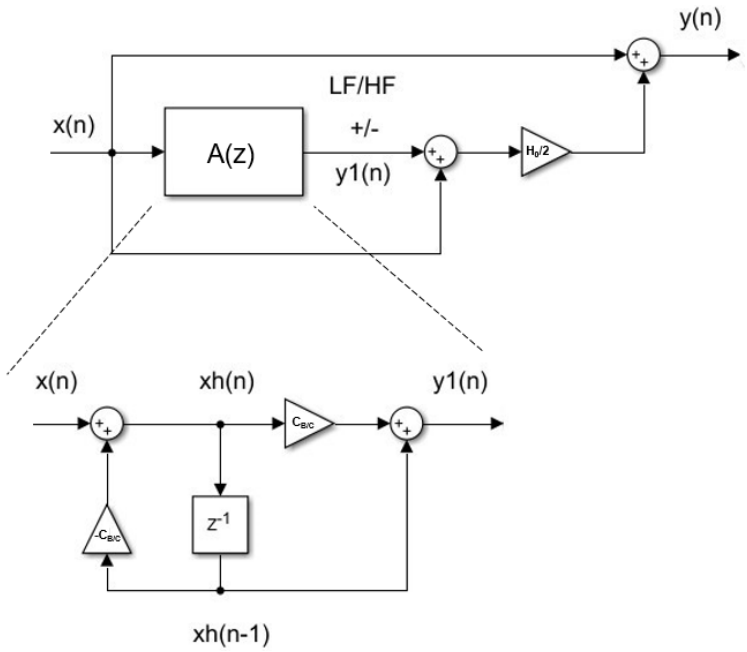


Figura 13. Diagrama de bloques del *low shelving/highs helving*.

$$H(z) = 1 + \frac{H_0}{2}[1 \pm A(z)]$$

Ecuación 7. Función de transferencia del *low shelving/high shelving* (+/-).

2.2.2. *Peaking*

De forma similar a los ecualizadores de primer orden, los ecualizadores de segundo orden [23] se construyen utilizando un paso todo de segundo orden, cuya función de transferencia se describe en la Ecuación 8.

$$A_2(z) = \frac{-C_{B/C} + d(1 - C_{B/C})z^{-1} + z^{-2}}{1 + d(1 - C_{B/C})z^{-1} - z^{-2}\, C_{B/C}}$$

Ecuación 8. Función de transferencia del *peaking*.

En este caso, $C_{B/C}$ se calcula con la

Ecuación **9**, donde $\frac{f_b}{f_s}$ corresponde a la mitad del ancho de banda normalizado, parámetro controlado por el usuario.

$$C_B = \frac{\tan\left(\frac{\pi f_b}{f_s} - 1\right)}{\tan\left(\frac{\pi f_b}{f_s} + 1\right)} \qquad C_C = \frac{\tan\left(\frac{\pi f_b}{f_s} - V_0\right)}{\tan\left(\frac{\pi f_b}{f_s} + V_0\right)}$$

Ecuación 9. (a) Parámetros *boost* del *peaking*, (b) parámetros *cut* del *peaking*.

Con estas ideas presentes, se describe el filtro *peaking* en la Figura 14, donde $d = -\cos\left(\frac{2\pi f_c}{f_s}\right)$, cuya función de transferencia se detalla en la Ecuación 10.

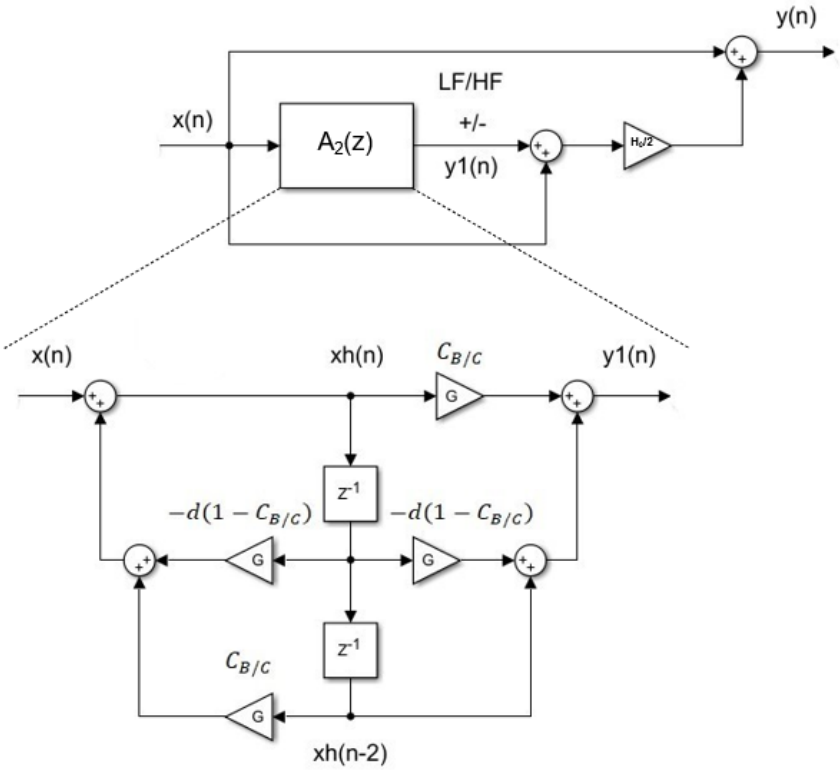


Figura 14. Diagrama de bloques del *peaking*.

$$H(z) = 1 + \frac{H_0}{2}[1 - A_2(z)]$$

Ecuación 10. Función de transferencia del *peaking*.

2.3. Wah-wah

En este tipo de efectos, el sonido se produce por un barrido en frecuencia muy característico que simula el sonido "*wah wah*", de ahí el nombre de este tipo de efectos. Este barrido en frecuencia se define como la oscilación de la frecuencia central sobre un filtro paso banda [24]. En general, este tipo de filtros se pueden clasificar en dos subgrupos: *Automatic-wah* y *manual-wah* [25].

En los *wah-wah* automáticos la frecuencia central oscila a una tasa constante establecida por el usuario [24], lo que reduce la flexibilidad de barrido que ofrece un *wah-wah* manual. Los *wah-wah* manuales se controlan típicamente con un pedal, donde se permite el movimiento de balanceo de la punta del pie al talón y viceversa. En este trabajo, se ha implementado un *wah-wah* automático, utilizando el filtro variable mostrado en la Figura 15 [25], donde $Q_1 = \frac{1}{\mathrm{Q}}$ y $F_1 = 2\sin(\frac{\pi f_c}{f_s})$, con f_c la frecuencia central y f_s la frecuencia de muestreo. Lógicamente, tanto f_c como Q, ancho de banda del filtro paso banda, son parámetros controlados por el usuario.

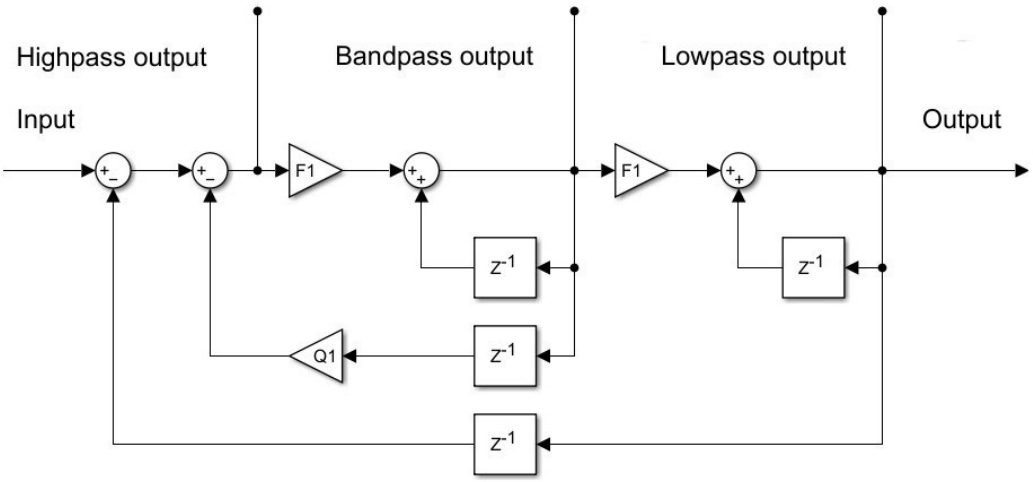


Figura 15. Diagrama de bloques del *wah-wah* manual.

2.4. Delay

En este efecto la señal de entrada se repite, como mucho, una o dos veces, a un nivel de volumen muy similar al original [27]. Para ello, se hace uso de una línea de retardo multiplicada por una ganancia atenuadora que se superpone a la señal de entrada del sistema, tal y como se muestra en la Figura 16.

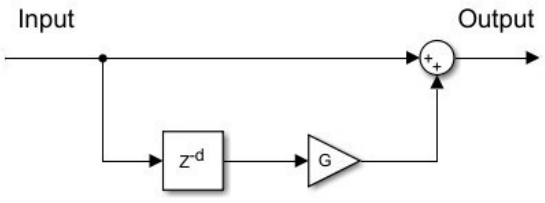


Figura 16. Diagrama de bloques del *delay*.

3. IMPLEMENTACIÓN DE LOS DISTINTOS EFECTOS EN COMA FIJA

En este proyecto se han procesado señales de audio de 16 bits, muestreadas a 48 kHz, en formato *stream*, es decir, muestra a muestra, obteniendo así las restricciones del diseño.

Por este motivo, se ha limitado a 16 bits el número de bits utilizados tanto en los coeficientes como en los parámetros de los distintos algoritmos. De esta forma, y al tratarse de una implementación en un microprocesador de 32 bits, el resultado de las multiplicaciones será de 32 bits, aunque Simulink® permita trabajar con números de 64 bits.

3.1. Implementación en MATLAB®

Una vez estudiadas las características principales de los distintos efectos, el siguiente paso es implementarlos en coma fija en C. Para ello, tal y como se viene comentando a lo largo de este trabajo, se ha partido de una primera implementación en MATLAB®, utilizando reales, que será el modelo de referencia con el que se compararán los distintos efectos implementados tanto en Simulink®, donde se ha utilizado el *Fixed-Point Toolbox*, como en C.

En la Figura 17, se observa una comparativa entre la señal original y la señal procesada, para el caso del efecto *overdrive*.

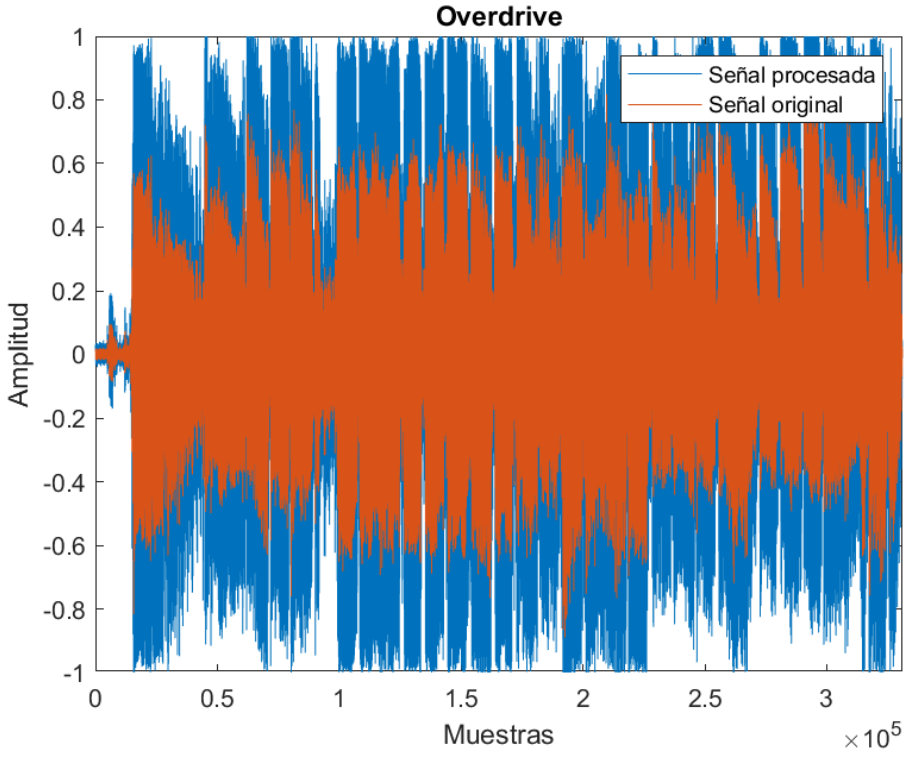


Figura 17. Comparativa entre la señal original y la señal procesada con el efecto *overdrive*.

Conviene destacar que, tanto el código de la totalidad de los efectos creados como el código utilizado para obtener las distintas gráficas comparativas entre señal original y procesada, se recogen en los correspondientes Anexos.

Además, para comprobar en tiempo real cómo suenan los distintos efectos y cómo afecta la variación de sus respectivos parámetros, de una forma rápida e intuitiva, se ha implementado una GUI en MATLAB® App Designer (Figura 18), con la ayuda de algunos tutoriales disponibles en MathWorks® [28,29].

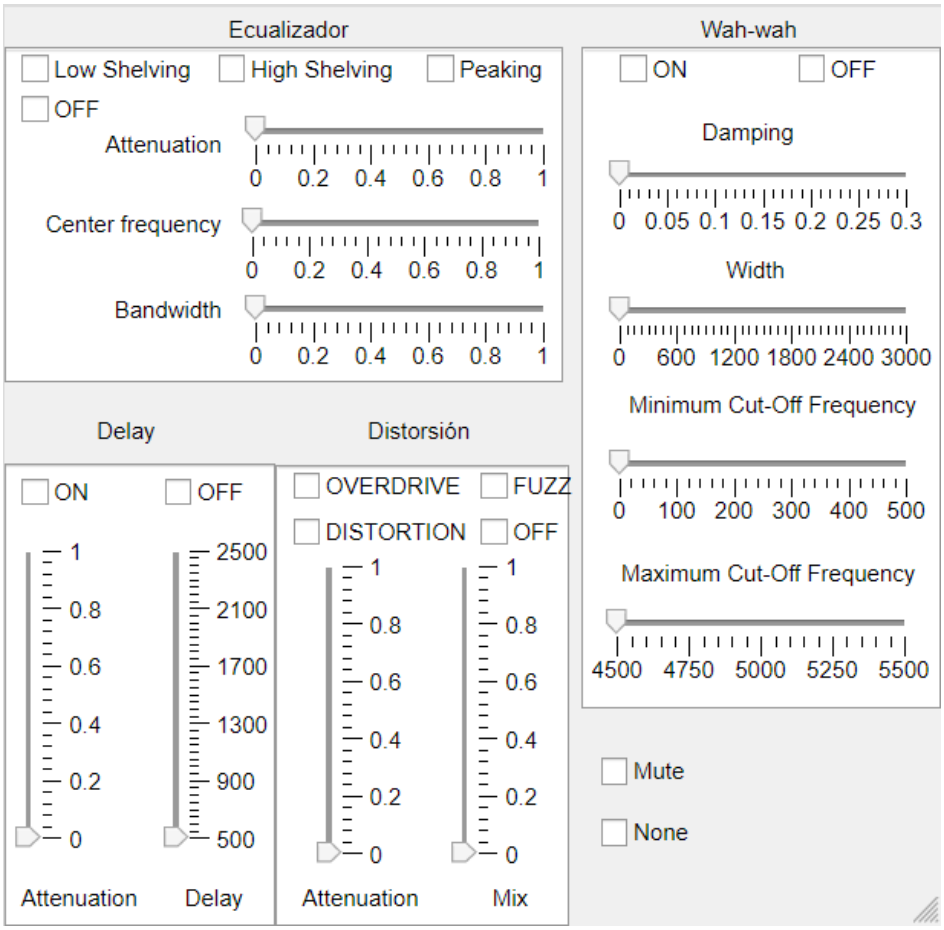


Figura 18. GUI en MATLAB®.

El código utilizado para crear esta GUI se recoge en los correspondientes Anexos.

3.2. Implementación en Simulink®

En este apartado se han trasladado los efectos implementados y ajustados previamente en MATLAB® a un sistema de bloques de alto nivel en Simulink®, que facilita su posterior implementación en C.

En un primer lugar se han implementado los efectos en formato *double*. Tras comprobar gráficamente que la señal de salida es idéntica a la obtenida con el código en MATLAB®, se ha procedido a implementarlos en coma fija.

Al trabajar en coma fija, se decide el formato de cada nodo intermedio, es decir, el número de bits y la escala. Por ejemplo, en un formato de <16,15> se tienen 16 bits de profundidad y una escala de 15, siendo $2^{-15} = 0.000030517578125$. De esta forma, se pueden representar valores desde -1 a 0.999969482421875.

Entonces, en este apartado, se ha determinado el formato de cada nodo ajustando mediante desplazamientos las operaciones aritméticas. Por ejemplo, si se tiene una señal en formato <16,15> multiplicada por un coeficiente en formato <16,13> el formato del nodo de salida será de <32,28> (Figura 19). Si se quiere volver a multiplicar por otro coeficiente, hay que realizar su correspondiente desplazamiento para volver a tenerla en 16 bits.

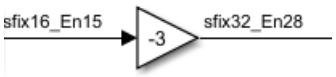


Figura 19. Producto en coma fija.

En este proceso se produce un error de truncado que, junto con los errores de cuantización de los coeficientes, hace que se pierda precisión, tal y como se puede observar en la Figura 20, donde se muestra una gráfica comparativa entre la señal procesada en MATLAB® y la señal procesada en Simulink® en coma fija para el efecto *distortion*.

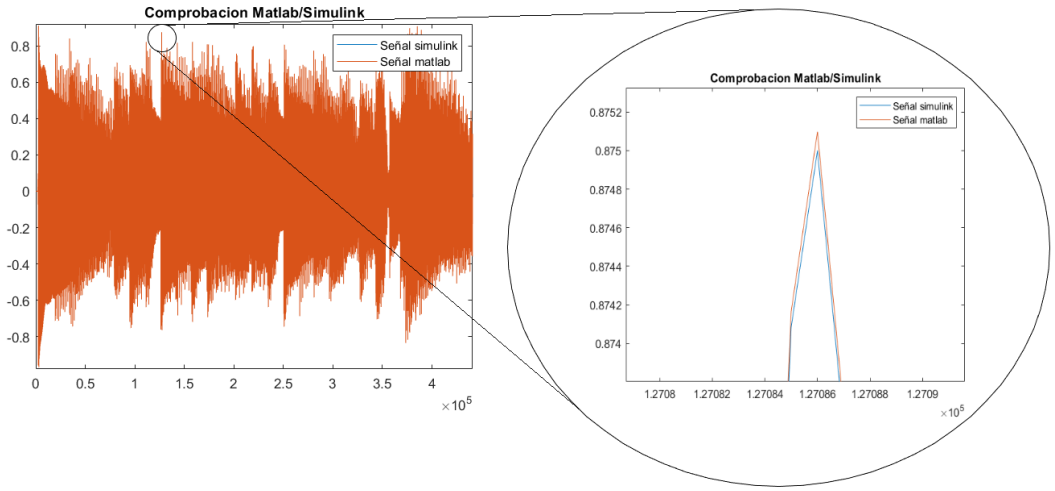


Figura 20. Comparación gráfica entre el efecto en MATLAB® y en coma fija en Simulink®.

Conviene destacar que los respectivos diagramas de bloques utilizados en Simulink®, se recogen en los correspondientes Anexos.

3.3. Implementación en C

En este bloque se han trasladado los efectos implementados y ajustados previamente en coma fija en Simulink® a C. Además, debido a su simplicidad, se han implementado directamente desde cero los efectos *none, mute* y atenuación.

En primer lugar, se desarrollan los distintos algoritmos en el PC para, posteriormente, trasladar ese mismo código al microprocesador. Para ello, se hace uso de los tipos definidos en la librería <stdint.h>. Por otro lado, al trabajar con números en coma fija, las operaciones aritméticas que se utilizan son enteros. Por ejemplo, si se quiere multiplicar 0.75 (formato <16,15>) por 0.25 (formato <16,16>), dará como resultado el producto en formato <32,31>.

$$0.75 \cdot 2^{15} = 24576$$

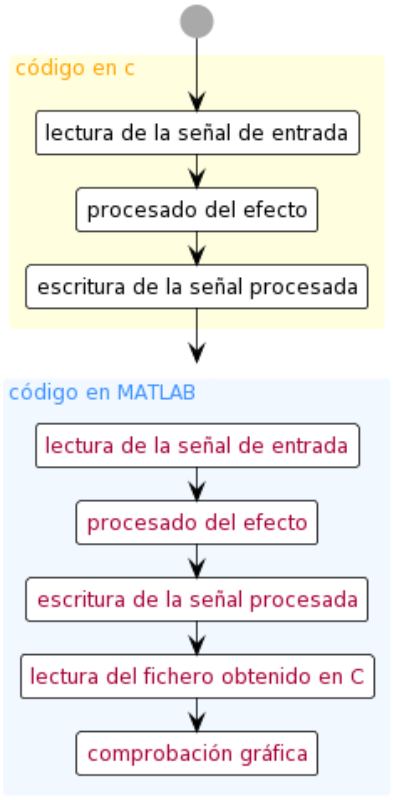
$$0.25 \cdot 2^{16} = 16384$$

Si se multiplican estos dos números enteros y se divide por la escala, se obtiene el valor esperado.

$$\frac{24576 \cdot 16384}{2^{31}} = 0.1875 = 0.75 \cdot 0.25$$

Otro detalle a tener en cuenta es que, tal y como se ha visto en la DESCRIPCIÓN DE LOS DISTINTOS EFECTOS EN MATLAB®, algunos efectos requieren de funciones matemáticas como la exponencial, el seno o el coseno. Para ello, se utilizan una serie de LUTs optimizadas [30].

Para verificar el correcto funcionamiento de los distintos efectos se vuelca el efecto procesado en C en un fichero de texto, para, posteriormente leerlo desde MATLAB® y compararlo gráficamente con el efecto procesado en MATLAB® (Esquema 4).



Esquema 4. Comprobación de efectos en C.

En la Figura 21, se muestra una gráfica comparativa entre la señal procesada en MATLAB® y la señal procesada en C para el efecto *distortion*.

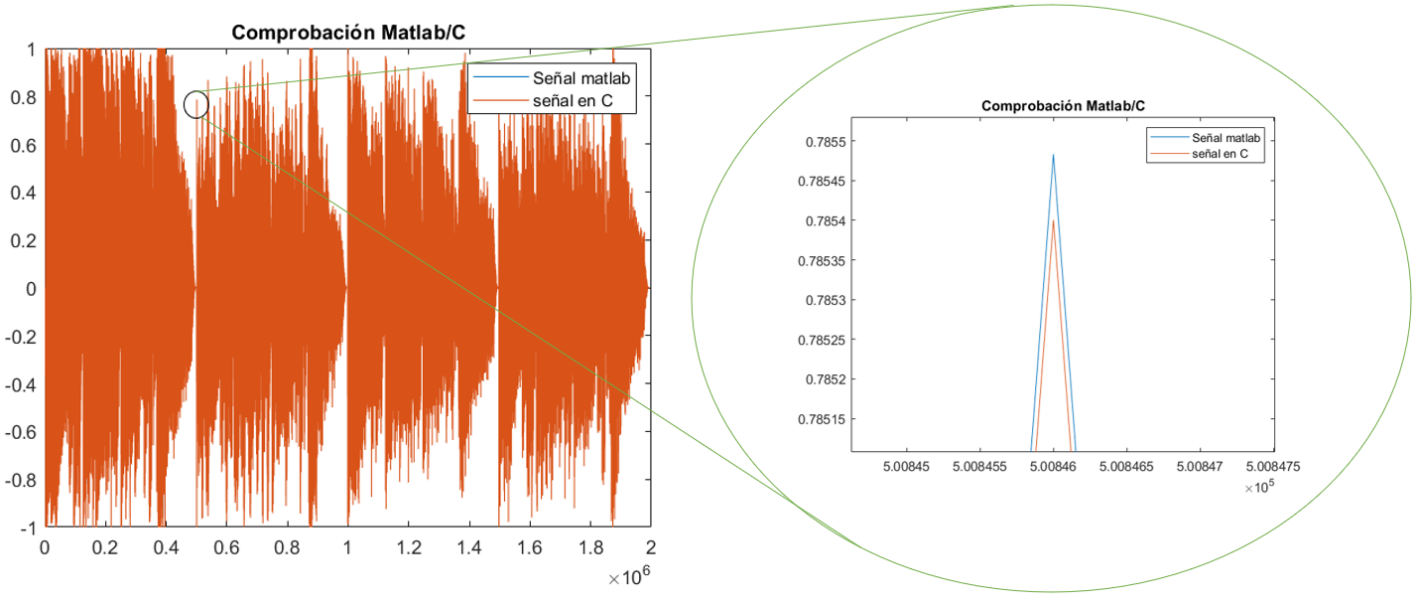


Figura 21. Comparación gráfica entre el efecto en MATLAB® y en C.

El código de la totalidad de los efectos creados en C, así como el de las LUTs optimizadas se recogen en los correspondientes Anexos.

4. LIBRERÍA DE EFECTOS EN COMA FIJA

4.1. Interfaz

En este apartado se ha diseñado una plataforma software donde se reúnen todos los efectos implementados a lo largo de este trabajo. Para ello, se ha desarrollado la clase mostrada en la Figura 22.

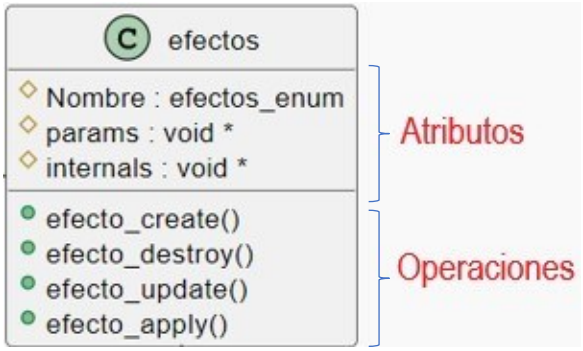


Figura 22. Atributos y operaciones de la clase efectos.

Dicha clase contiene tres atributos; el nombre, los parámetros del efecto y los elementos internos (ID, *delays*, constantes…); y las funciones que implementan las operaciones aplicadas a éstos.

La primera función, *efecto_create* (Código 1) crea el efecto, es decir, crea todas las estructuras de datos con los campos nombre, parámetros e *internals*. Para ello, se utilizan los recursos que ofrece C para reservar memoria de forma dinámica como, por ejemplo, la funcion *malloc*. La segunda función, *efecto_destroy* (Código 2), común a todos los efectos, libera el espacio de memoria reservado al crear el efecto, una vez se ha dejado de utilizar. En cuanto a la función *efecto_update* (Código 3), se encarga de actualizar el efecto con los parámetros del efecto que demande el usuario. Por último, la función *efecto_apply* (Código 4) se encarga de aplicar el efecto a la señal de entrada.

```
efecto_t *efecto_none_create() {
    static uint8_t id=0;
    efecto_noneinternals_t *noneinternals;
    efecto_t * efecto_ptr = malloc(sizeof(efecto_t));
    efecto_ptr->efecto = EFECTO_NONE;
    efecto_ptr->internals = malloc(sizeof(efecto_noneinternals_t));
    noneinternals=efecto_ptr->internals;
    noneinternals->id = id++;
    efecto_ptr->params = NULL;
    return efecto_ptr;
}
```

Código 1. Función *create* del efecto *none*.

```
void efecto_destroy(efecto_t *efecto)
{
    if (efecto != NULL)
    {
        if (efecto->internals != NULL)
        {
            free(efecto->internals);
        }
        efecto->internals = NULL;
        if (efecto->params != NULL)
        {
            free(efecto->params);
        }
        efecto->internals = NULL;
        free(efecto);
    }
}
```

Código 2. Función destroy.

```
void efecto_distortion_update(efecto_t * efecto, efecto_distortionparams_t * params)
{
    efecto_distortionparams_t *distortion_params = efecto->params;
    distortion_params=efecto->params;
    distortion_params->gain = params->gain;
    distortion_params->mix = params->mix;
}
```

Código 3. Función *update* del efecto *distortion*.

```
int16_t efecto_none_apply(efecto_t * efecto, int16_t sample) {
    return sample;
}
```

Código 4. Función apply del efecto *none*.

4.2. Diseño de la librería

Una vez estudiadas las características principales de la clase efectos, se crean nuevas subclases heredadas de ésta.

En primer lugar, en la Figura 23 se muestran las distintas subclases que no tienen ni parámetros ni *internals*, a excepción del identificador.

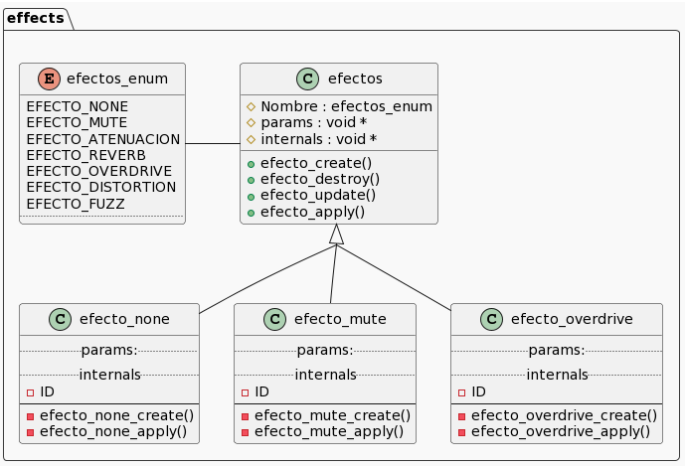


Figura 23. Subclases sin parámetros ni *internals*, excepción del identificador.

Estas subclases son las más fáciles de diseñar. Al no tener parámetros, no necesitan una función *update*. Simplemente, efecto_create y efecto_apply, donde se encuentra el algoritmo del efecto (Código 5).

```
typedef struct efecto_none_internals_str
{
    uint8_t id;
} efecto_noneinternals_t;

efecto_t *efecto_none_create() {
    static uint8_t id=0;
    efecto_noneinternals_t *noneinternals;
    efecto_t * efecto_ptr = malloc(sizeof(efecto_t));
    efecto_ptr->efecto = EFECTO_NONE;
    efecto_ptr->internals = malloc(sizeof(efecto_noneinternals_t));
    noneinternals=efecto_ptr->internals;
    noneinternals->id = id++;
    efecto_ptr->params = NULL;
    return efecto_ptr;
}

int16_t efecto_none_apply(efecto_t * efecto, int16_t sample) {
    return sample;
}
```

Código 5. Subclase efecto_none.

El siguiente grupo de subclases, mostradas en la Figura 24, sí que tienen parámetros, aunque, al igual que sucedía con las subclases anteriores, tampoco tienen *internals*, a excepción del identificador.

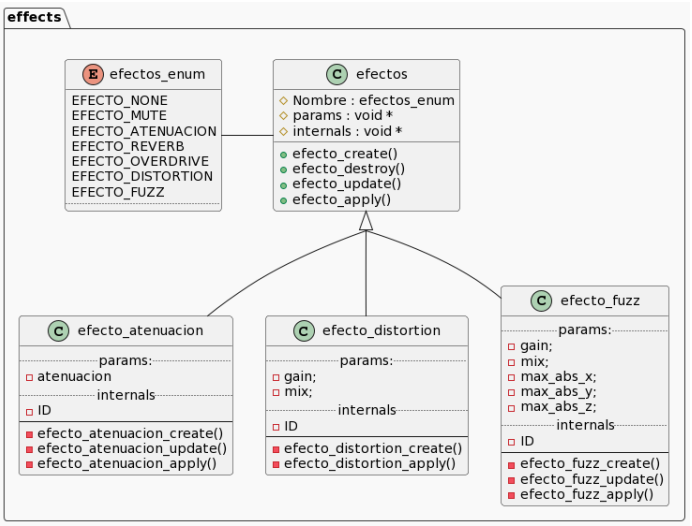


Figura 24. Subclases con parámetros, pero sin *internals* a excepción del identificador.

Como este subgrupo sí que cuenta con parámetros, necesitan una función *update* (Código 6).

```
void efecto_distortion_update(efecto_t * efecto, efecto_distortionparams_t * params) {
    efecto_distortionparams_t *distortion_params = efecto->params;
    distortion_params=efecto->params;
    distortion_params->gain = params->gain;
    distortion_params->mix = params->mix;
}
```

Código 6. Función *update* del efecto *distortion*.

Por último, la clase efecto_reverb tiene tanto parámetros como *internals* (Figura 25).

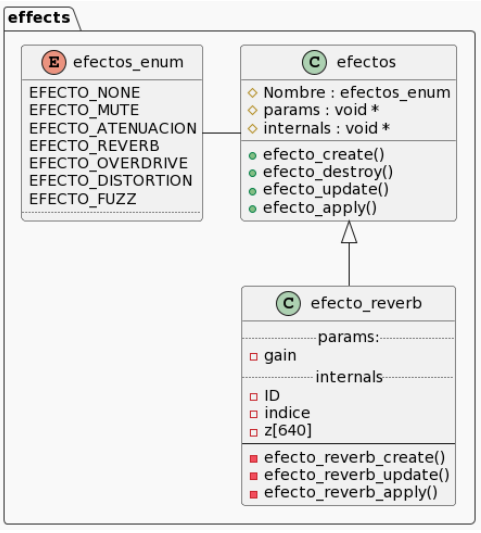


Figura 25. Subclases tanto con parámetros como con *internals*.

En los correspondientes Anexos se recoge el código en C del resto de los efectos implementados en esta arquitectura *software*.

4.3. Instrucciones para añadir nuevos efectos

Por último, como se viene comentando, lo más importante de esta arquitectura, cuya jerarquía se define en la Figura 26, es su escalabilidad, permitiendo al usuario añadir nuevos efectos con facilidad.

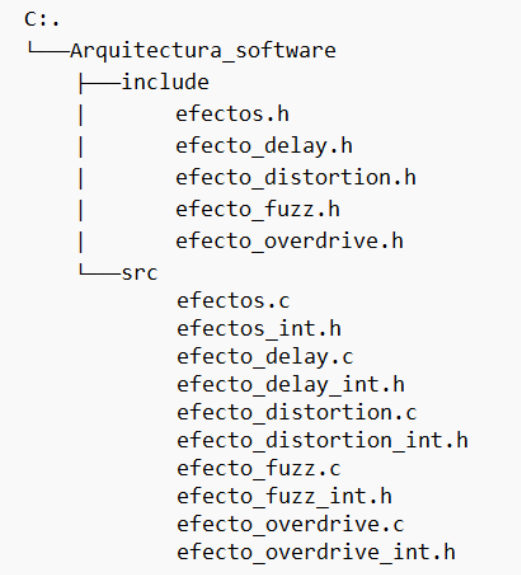


Figura 26. Árbol de directorios de la carpeta "Arquitectura_software".

Cabe destacar la importancia de diferenciar la parte pública, accesible por el usuario, de la privada. Se ha tratado de esconder y encapsular todo aquello que el usuario no tiene por qué conocer, imposibilitando cualquier tipo de modificación malintencionada, garantizando así la integridad de los distintos archivos.

De cara a un usuario final, sólo es necesario que conozca el contenido de la carpeta *include*. En ésta se incluyen todos los .h que tienen información pública. En concreto, existe un .h genérico para todos los efectos (efectos.h) y un .h para cada uno de los efectos que se han implementado. Por otro lado, en la carpeta *src* se encuentra el .c genérico para todos los efectos, así como el .c y .h privado para cada uno de los efectos.

Para añadir nuevos efectos, en primer lugar, se debe modificar el fichero include/efectos.h (Código 7).

```
#include <stdint.h>
#include "efecto_overdrive.h"
#include "efecto_distortion.h"                    // <<---- Incluir el efecto_NOMBRE.h
#include "efecto_fuzz.h"

typedef enum efectos_enu { EFECTO_OVERDRIVE,
                           EFECTO_DISTORTION,     // <<---- Incluir el nombre del efecto
                           EFECTO_FUZZ,
                           } efectos_enu_t;
```

Código 7. Extracto del fichero efectos.h.

Acto seguido, se debe modificar el fichero src/efectos_int.h (Código 8).

```
#include "efectos.h"
#include "efecto_overdrive_int.h"
#include "efecto_distortion_int.h"              // <<---- Incluir el efecto_NOMBRE_int.h
#include "efecto_fuzz_int.h"
```

Código 8. Extracto del fichero src/efectos_int.h.

Por último, se deben modificar las funciones de src/efectos.c

5. VERIFICACIÓN FUNCIONAL DEL DISEÑO

Una vez implementados los distintos efectos en C, se ha desarrollado un *testbench* para comprobar el funcionamiento de la arquitectura *software* con el sistema de desarrollo FM4-176L-S6E2CC-ETH.

Dicho sistema cuenta con un microprocesador ARM Cortex M4F [31] (Figura 27, azul) y con un códec de audio (Figura 27, naranja) con conversores ADCs y DACs estéreo de 24 bits, cuya entrada digital de audio puede tener de 16 a 32 bits de profundidad y frecuencias de muestreo de 8 a 96 kHz [32], por lo que está pensado para trabajar con audio de alta resolución [33].

A continuación, se detalla el esquema de interconexión empleado.

En primer lugar, se conecta la placa con el ordenador a través de un cable USB con terminación Micro USB (Figura 27, rosa). Acto seguido, se conecta la salida de audio del ordenador con la entrada de audio de la placa (Figura 27, amarillo) utilizando un cable *jack*. Por último, se conectan unos auriculares con la salida de audio de la placa (Figura 27, morado).

Figura 27. Periféricos de la placa FM4-176L-S6E2CC-ETH.

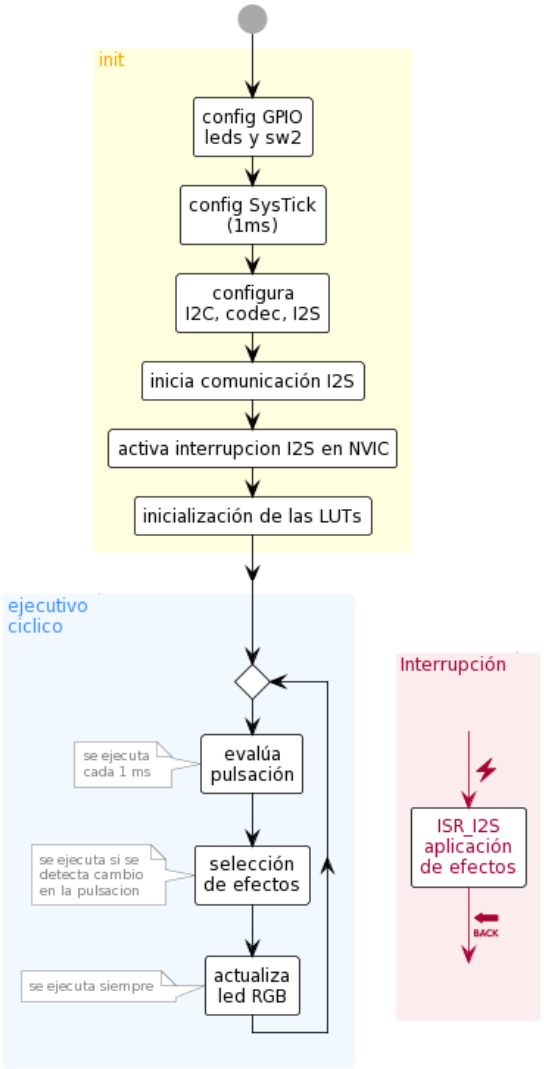
Se cambia de efecto cada vez que el usuario hace una pulsación corta en el *user button* (Figura 27, rojo), y, una vez seleccionado el efecto deseado, se debe hacer una pulsación larga para que se aplique dicho efecto a la señal de entrada.

Por otro lado, para facilitar al usuario la identificación del efecto que está seleccionando, se enciende el LED RGB (Figura 27, verde) con el color que corresponda según la Tabla 1.

Tabla 1. Relación de colores y efectos.

Color	Efecto
Off	*None*
Rojo	*Fuzz*
Verde	*Mute*
Azul	*Overdrive*
Amarillo	Atenuación
Magenta	*Distortion*
Cian	*Delay*

Para implementar este sistema, se ha partido de un sistema desarrollado previamente en las prácticas de la asignatura de Sistemas Electrónicos con Microprocesadores, modificándolo ligeramente para que se adapte a las necesidades de este trabajo (Esquema 5).



Esquema 5. Esquema del funcionamiento del sistema.

Por último, se ha grabado un vídeo que respalda las comprobaciones experimentales del diseño [34].

6. CONCLUSIONES Y LÍNEAS DE INVESTIGACIÓN FUTURAS

6.1. Conclusiones

En este trabajo se ha diseñado, implementado y verificado una librería de efectos de audio. Para cumplir con los objetivos del proyecto se han realizado las siguientes tareas:

- Se ha comprobado gráfica y auditivamente con MATLAB® la correcta implementación de los distintos algoritmos de procesado de audio en C.
- Se ha conseguido diseñar una plataforma *software* escalable, permitiendo añadir nuevos efectos con facilidad.
- Se ha verificado funcionalmente el diseño mediante el sistema de desarrollo FM4-176L-S6E2CC-ETH.

6.2. Líneas de investigación futuras

Con la realización del presente proyecto se ha demostrado la viabilidad de implementar una librería de efectos en el microprocesador. Sin embargo, el prototipo implementado dista mucho de ser un sistema válido para una aplicación real en la industria musical.

Para poder llegar a implementar un sistema válido para una aplicación real, los siguientes pasos de la investigación podrían ser los siguientes:

- Implementación de nuevos efectos: Al haber desarrollado una plataforma *software* escalable en este trabajo, se podrían añadir otros efectos muy conocidos en la industria musical como es el caso del *flanger*, *chorus* o *phaser*, o algunos de los efectos que se han llegado a implementar y a ajustar en Simulink®, pero que, por falta de tiempo, no se han llegado a implementar en C.
- Implementación de un módulo Bluetooth®: Con el fin de facilitar al usuario la selección de los distintos efectos y sus respectivos parámetros, de una forma más intuitiva y rápida, se podría añadir un módulo Bluetooth® que permitiese al usuario utilizar la GUI implementada en MATLAB® para comunicarse con el microprocesador desde cualquier dispositivo electrónico.

7. REFERENCIAS

[1] L. Llorente, "Implementation of real-time digital audio effects using a microprocessor" December, 2022 [Online]. Available: https://github.com/luisllorentemuro/Implementation-of-real-time-digital-audio-effects-using-a-microprocessor, [Accessed: Nov. 20, 2022].

[2] S. W. Smith, "ADC and DAC", in *The scientist and engineer's guide to digital signal processing*, 2nd ed., San Diego: California Technical Pub, 1997, pp. 35-66.

[3] "Boss Flanger BF-3 - Pedal de guitarra flanger." [Online]. Available: https://www.pronorte.es/p/boss-flanger-bf-3, [Accessed: Nov. 20, 2022].

[4] "modelo 3d Guitarra pedalera - TurboSquid 771408." [Online]. Available https://www.turbosquid.com/es/3d-models/guitar-pedalboard-max/771408, [Accessed: Nov. 20, 2022].

[5] "Mooer GE300 - Pedalera multiefectos guitarra." [Online]. Available https://www.pronorte.es/p/mooer-ge300, [Accessed: Nov. 20, 2022].

[6] Steinberg Media Technologies GmbH, *Cubase.* [Propietary Software]. Hamburg, DE, 2022.

[7] Avid Technology, Inc, *Pro-Tools*. [Propietary Software]. Massachusetts, US, 2022

[8] C. Truesdell, "Pro Tools", in Mastering digital audio production : the professional music workflow with Mac OS X, 2nd ed., Wiley, 2007, pp. 129-130

[9] S. Gibbs, C. Arapis, C. Breiteneder, V. Lalioti, S. Mostafawy and J. Speier, "Virtual studios: an overview," *IEEE MultiMedia*, vol. 5, no. 1, March, pp. 18-35, 1998.

[10] "Amazon.com: AudioControl DM-RTA Analizador en tiempo real y herramienta de prueba múltiple : Industrial y Científico." [Online]. Available https://www.amazon.com/-/es/AudioControl-DM-RTA-Analizador-herramienta-múltiple/dp/B07NPS82GC, [Accessed: Nov. 20, 2022].

[11] "VSTi llegada DF - SoftPlug." [Online]. https://softplug.com/es/product/adventus-df/, [accessed Nov. 20, 2022].

[12] R. Stallman, "Using and porting the gnu compiler collection," Free Softw. Found., 1989, [Online]. Available: http://gcc.gnu.org/onlinedocs/gcc-2.95.3/gcc.html, [Accessed: Nov. 25, 2022].

[13] R. Environment, "µVision ® IDE," 2018. [Online]. Available https://www2.keil.com/mdk5/uvision/, [Accessed Nov. 20, 2022].

[14] "FM4-176L-S6E2CC-ETH QUICK START GUIDE", [Online]. Available: http://www.cypress.com/fm4-176l-s6e2cc-eth, [Accessed: Nov. 20, 2022].

[15] J. D. Reiss and A. McPherson, "Overdrive, Distortion and Fuzz" in *Audio Effects: Theory, Implementation and Application*, London: CRC Press, 2014, pp. 169-170.

[16] J. D. Reiss and A. McPherson, "Overdrive, Distortion and Fuzz" in *Audio Effects: Theory, Implementation and Application*, London: CRC Press, 2014, pp. 171-172.

[17] U. Zölzer, Ed., "Nonlinear processing" in *DAFX: Digital Audio Effects*, 2nd ed., Wiley, 2011, pp. 124-130.

[18] M. Schetzen, *The Volterra and Wiener Theories of Nonlinear Systems*, Krieger Publishing, 1980.

[19] R. Bendiksen, "Digitale Lydeffekter", M.S. thesis, Norwegian University of Science and Technology, Trondheim, Norway, 1997.

[20] R. G. Keen. The technology of the fuzz face, 1998. [Online]. Aviable: www.geofex.com. [Accessed Aug. 20, 2022].

[21] D. Marshall and K. Sidorov. Class Lecture, Topic: Digital Audio Effects, School of Computer Science & Informatics, Cardiff University, Welsh, UK. [Online], Available: https://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/PDF/07_Audio_Effects.pdf [Accessed: Nov. 20, 2022].

[22] U. Zölzer, Ed., "Filters and delays" in *DAFX: Digital Audio Effects*, 2nd ed., Chichester, West Sussex, U.K : Wiley, 2011, pp. 61.

[23] U. Zölzer, Ed., "Filters and delays" in *DAFX: Digital Audio Effects*, 2nd ed., Chichester, West Sussex, U.K : Wiley, 2011, pp. 61-66.

[24] U. Zölzer, Ed., "Filters and delays," in *DAFX: Digital Audio Effects*, 2nd ed. Chichester, West Sussex, U.K : Wiley , 2011, pp. 67 - 68.

[25] P. Burrows, "Real-Time Wah GUI Implementation in MATLAB (Guitar Demo)", December, 2014 [Online]. Available: https://www.youtube.com/watch?v=WRH_n2Wz49o [Accessed: Nov. 20, 2022].

[26] U. Zölzer, Ed., "Filters and delays," in *DAFX: Digital Audio Effects*, 2nd ed. Chichester, West Sussex, U.K : Wiley, 2011, pp. 48 - 51.

[27] A. J. Czubak, "Guitar Effects Processor Using DSP", M.S. Thesis, Bradley University, Peoria, IL, 2014.

[28] "Creating apps with graphical user interfaces in MATLAB" [Online]. Available: http://www.mathworks.com/discovery/matlab-gui.html?refresh=true [Accessed: Nov. 20, 2022].

[29] M. Fig, "41 complete GUI examples" July, 2009 [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/24861-41-complete-gui-examples [Accessed: Nov. 20, 2022].

[30] "Fixed Point implementation of C exponent function. The Coding Forums." [Online]. Available: https://www.thecodingforums.com/threads/fixed-point-implementation-of-c-exponent-function.717009/ [Accessed: Nov. 20, 2022].

[31] Arm Ltd.®, "Arm Cortex-M4," datasheet, Feb. 2020 [Revised Nov. 2022]

[32] Wolfson Microelectronics plc®, "Portable Internet Audio CODEC with Headphone Driver and Programmable Sample Rates", WM8731 / WM8731L datasheet, Feb. 2005 [Revised Nov. 2022]

[33] Sony, "Sound Quality Comparison of Hi-Res Audio vs. CD vs. MP3 | Sony US." [Online]. Available: https://electronics.sony.com/hi-res-audio-mp3-cd-sound-quality-comparison [Accessed: Nov. 20, 2022].

[34] L. Llorente, "Implementation of real time digital audio effects using a microprocessor", November, 2022 [Online]. Available: https://youtu.be/V3GtaTH-AZY, [Accessed: Nov. 20, 2022].