Universidad Católica Argentina
Facultad de Ingeniería y Ciencias Agrarias


Trabajo Final de Ing. Electrónica

# Plataforma Abierta para el Procesamiento de Audio e Implementación de Efectos en Alta Calidad

**Alumnos:**

Gonzalo Giulio 15-122300-0

Santiago Savalli 15-142258-2

**Tutor:**

Ing. Ricardo Vecchio

Ing. Norberto Heyaca

**- 2021 -**

# Índice

# 1 - Introducción

## 1.1 - Alcance y Objetivos del Proyecto

Este trabajo final plantea el diseño de una pedalera digital multi-efecto que funcione en tiempo real pensada en torno a un modelo de plataforma abierta. Para lograr esto, se establecieron cuatro objetivos distintos.

Primero, se debe desarrollar una interfaz de audio digital de alta calidad y accesibilidad. Esto implica contar con un sistema de conversión digital-analógica y analógica-digital capaz de interactuar con un procesador de alta velocidad. Debe contar con conectores Jack a fin de ser compatible con aplicaciones musicales. Se propone, para dicho sistema, obtener una calidad de audio de resolución extremo a extremo superior a la calidad de CD (16 bits a 44,1 KHz).

Como segundo punto, sobre el programa que corre en la CPU se pretende ofrecer al usuario la posibilidad de seleccionar entre una variedad de efectos musicales equivalente a los de mayor demanda en el mercado. Del mismo modo, se añade el requerimiento de que éstos tengan la capacidad de ajustarse dinámicamente mientras se ejecuta el programa minimizando el impacto en la carga del procesador.

En tercer lugar, se busca que la pedalera sea lo suficientemente intuitiva como para que un usuario sin conocimientos de programación pueda interactuar con las funcionalidades presentes. Esto requiere el diseño de una interfaz gráfica que actúe como interprete entre el usuario y el programa.

Por último, es importante para este trabajo la adopción de un esquema de plataforma abierta. De esta manera, se dejan las puertas abiertas a que un usuario con experiencia extienda el código con sus propias ideas de efectos u optimizaciones. Además de la publicación del código final, queda implícita la abstinencia del uso de cualquier tipo de hardware o código protegido por licencia.

En términos económicos, se planea mantener un presupuesto acotado para lograr un costo reducido en comparación al resto de los productos similares que se pueden encontrar en el mercado.

## 1.2 - Conceptos Básicos

Una pedalera digital multi-efecto se trata de un dispositivo que se conecta entre el instrumento (generalmente una guitarra eléctrica) y el parlante para poder transformar el sonido de acuerdo al gusto del intérprete. Estas transformaciones consisten en el procesamiento de la señal de audio que ingresa al dispositivo y tienen el fin de otorgarle características distintas al timbre propio del instrumento. Si bien este trabajo hace foco en esto último, las pedaleras no se limitan a esta función, sino que muchas veces también pueden llegar a sintetizar ritmos de baterías, actuar como afinador, entre otros.

Si bien este tipo de pedaleras actúan de forma digital, se encuentran preparadas para recibir señales analógicas en la entrada y transmitir señales analógicas a la salida. Esto se debe a que los instrumentos convencionales generan una señal analógica y los parlantes deben recibir una excitación analógica para poder traducir el voltaje a presión sonora. Por esto, la entrada de dicho dispositivo debe poseer un conversor analógico/digital y la salida debe poseer un conversor digital/analógico.

El hecho de que se produzca una conversión inmediatamente implica un deterioro en la calidad del audio. Sin embargo, conforme más avanzado sea este proceso, más se puede mitigar este deterioro. Los parámetros que indican cuán fiel será la conversión son la profundidad de bits y la frecuencia de muestreo. Como referencia, el estándar para calidad de CD es 16 bits a 44,1kHz.

Todo el procesamiento se lleva a cabo en la CPU de la pedalera, la cual ejecuta las instrucciones que previamente se le cargaron mediante un código desarrollado por el programador.

Las señales de audio contienen la información codificada en su respuesta en frecuencia. Esto significa que los algoritmos que realiza el procesador para lograr los efectos se suelen basar en filtros digitales. Para esto, se aprovechan herramientas como la transformada de Laplace, la transformada Z y la transformada bilineal.

## 1.3 - Estructura del Trabajo

- Primero, en el capítulo 2, se detalla una breve historia del rol de los efectos en la música explicando cómo se llega al estado del arte de las pedaleras multi-efecto.
- A continuación, durante los capítulos 3 al 7, se habla a cerca de la elección del hardware, la estructura de la interfaz de audio, el código desarrollado y la interfaz gráfica de usuario.
- Los capítulos 8 al 12 comienzan con una explicación teórica de cada efecto en particular. Luego continúa con una elección de método y termina con un ejemplo de implementación en código. Se concluyó mantener este formato para tener una conexión directa entre la teoría y la práctica a lo largo de cada efecto.
- Por último, en los capítulos 13 y 14, se explica cómo se llega a un producto finalizado y se desarrollan las conclusiones correspondientes al proyecto.

# 2 - Contexto Histórico y Actualidad

## 2.1 - Comienzos

Históricamente, los instrumentos musicales fueron diseñados para producir sonidos muy particulares. Las limitadas variaciones en el timbre y las características del sonido surgieron como producto de la técnica y expresividad del artista.

A partir de la década del 1930, la tecnología llegó a ser lo suficientemente desarrollada como para facilitarle al músico herramientas externas destinadas a controlar las particularidades del sonido de su instrumento. La primera innovación fue por parte del órgano Hammond, éste incluía nueve barras deslizantes denominadas “drawbars” que podían ajustar la intensidad de los armónicos o sub-armónicos de la nota fundamental mediante ruedas fónicas. Cada una de estas barras contaba con un total de nueve posiciones (desde cero hasta ocho) yendo desde intensidad cero hasta máxima intensidad. Una combinación de estas nueve barras en nueve posiciones posibles definidas se pasó a denominar preset.

Muchos artistas comenzaron a experimentar con estas configuraciones y cada uno buscaba desarrollar los preset que mejor realicen el sonido proyectado para su música. También surgió una especie de competencia donde algunos músicos comenzaron a guardar sus configuraciones como un secreto de profesión ya que el timbre pasaba a ser parte de su marca y no querían que los demás lo copien.

Mientras tanto, alrededor de esta época, los guitarristas quienes solían ser opacados por instrumentos con presencias mayores como los de viento metálicos (saxofón, trompeta, entre otros) comenzaron a tener una revolución llevada a cabo con la aparición de las guitarras eléctricas y los amplificadores. El instrumento comenzó a ser relevante dentro de varios conjuntos y los fabricantes no desaprovecharon la oportunidad de intervenir en la señal para lograr herramientas configurables de forma externa.

## 2.2 - Pedales de Guitarra

La primera de ellas en ser fabricado en masa fue el trémolo de DeArmond en 1948. La electrónica no se encontraba lo suficientemente desarrollada, así que su funcionamiento se basaba en un líquido electrolítico (este debía ser rellenado por los usuarios generalmente con un limpiavidrios de la marca Windex) contenido dentro de un frasco metálico que era agitado mediante un motor. La agitación del líquido producía que la señal de la guitarra alterne entre ser enviada al amplificador y ser dirigida a tierra de manera cuasi periódica. El control del efecto se facilitaba al usuario mediante dos perillas que manejaban la excitación del motor.

Una variante muy innovadora de este modelo surgió poco después cuando las perillas se remplazaron por un único control implementado mediante un pedal. Esto permitía a los músicos hacer configuraciones en el efecto mientras tocaban la guitarra. De aquí se deriva el nombre que se le dio a estos dispositivos: pedales de efectos.

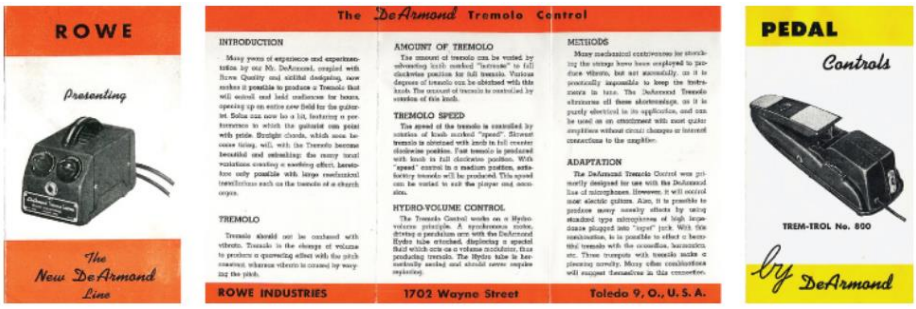


*Figura 2.2.1 – Folleto informativo de con ambas versiones del trémolo DeArmond.*

Durante la década de 1950, hubo avances significativos en el área de la electrónica. Éstos facilitaron la aparición de configuraciones externas dentro de los mismos amplificadores de guitarra como las perillas de ganancias y las de echo. Sin embargo, no se abandonó por completo la idea de intervenir entre la guitarra y el amplificador ya que comenzaron a aparecer equipos valvulares con efectos como distorsión, delay o trémolo. Estos equipos rápidamente se volvieron muy engorrosos de utilizar debido a su peso e inconveniencia.


*Figura 2.2.2 - Equipo de efectos Echoplex con tecnología valvular.*

La tecnología de transistores mejoró de forma eficiente y, en 1962, apareció el primer pedal transistorizado en el mercado: el Maestro Fuzz Tone. Tuvo un comienzo relativamente lento hasta que la banda Rolling Stones lo usó para su hit "(I Can't Get No) Satisfaction" en 1965. Este salto de popularidad desató una reacción en cadena que culminó con una especie de perfección de estos pedales de efecto. Comenzó a estandarizarse un interruptor accionable con el pie (comúnmente un 3PDT) que permitía hacer que la señal sea procesada por el efecto o se apague el pedal y se haga un bypass de todo el circuito. Los transistores comenzaron a ser más pequeños y accesibles resultando en cajas más livianas y baratas.


*Figura 2.2.3 – Distintas versiones del pedal transistorizado phase 90 de MXR a lo largo de los años.*

Este fenómeno, a su vez, impulso a los fabricantes de efectos a crear pedales cada vez más creativos. Se comenzaron a simular varios efectos que antes los hacia un productor musical en la grabación como el chorus, el flanger y los ecualizadores. Incluso se desarrollaron efectos

experimentales posibilitados solamente por el ambiente de desarrollo como el ringmod y los octavadores.

Los pedales de efectos se convirtieron en un estándar para muchos músicos, quienes buscaban tener acceso a cada vez más de ellos para poder inspirar su creatividad y desarrollar su sonido. Aún estaba presente la limitación de tener uno o dos efectos por dispositivo así que, para tener la posibilidad de usar varios efectos a la vez, se popularizó el uso de las "pedalboard". Se trata de tablas diseñadas para ajustar un número de pedales con el fin de facilitar su transporte, protección, alimentación, conexión en serie y uso.

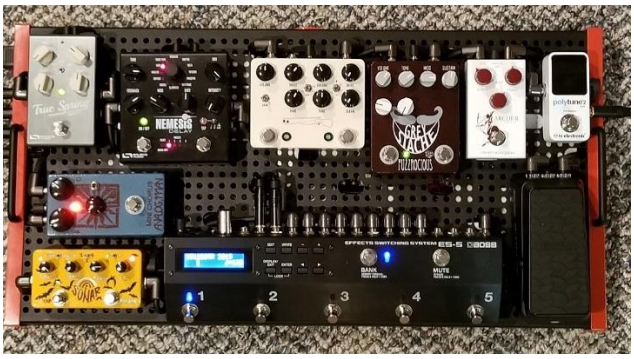

*Figura 2.2.4 – Ejemplo de pedalboard. Nótese la facilidad que brinda para el manejo de cables y la presentación de pedales.*

Mientras tanto, la tecnología digital fue avanzando a grandes pasos. En el año 1983 apareció en el mercado el primer pedal digital por parte de la BOSS llamado Digital Delay DD-2. No sólo imitaba su diseño, sino que tenía la calidad necesaria para poder competir con sus contrapartes analógicos.


*Figura 2.2.5 – Pedal de delay digital DD-2 de BOSS.*

Siguiendo a este, comenzaron a desarrollarse replicas digitales para prácticamente todos los efectos comerciales. Si bien la tecnología digital se volvía cada vez más barata y más potente, siempre existió una apreciación por los sonidos analógicos que hizo que, hasta el día de hoy, ambos continúen en constante competencia. Sin embargo, una gran ventaja que aprovechó la tecnología digital para poder establecerse más en el mercado fue el nivel de integración.

## 2.3 - Pedaleras Digitales Multi-efecto

Los pedales digitales pudieron introducir varios efectos en el mismo dispositivo de una manera mucho más eficiente en espacio y en costo. Tanto es así que lo que antes se lograba con muchos

pedales analógicos costosos conectados en serie en una "pedalboard" se pudo condensar en un producto único que se denominó pedalera digital multi-efecto.

El concepto es sencillo; se trata de un dispositivo que aprovecha la digitalización de la señal junto con un procesador potente y algoritmos eficientes para brindarle una gran cantidad de personalización al sonido del instrumento.

Actualmente en el mercado existe una muy amplia variedad de estos productos. Como es de pensar, los equipos de alta gama se caracterizan por tener una presentación acabada, utilizar procesadores más rápidos y lograr sonidos más atractivos. También, a medida que aumenta el precio se pueden encontrar funcionalidades adicionales como interfaces de usuario interactivas, controles programables (como pedales de expresión, perillas e interruptores 3PDT), variedad de conectores de entrada/salida (Jack, XLR, MIDI, etc.), sinterización de ritmos, capacidad de grabación y reproducción, entre otros. A continuación, una breve exposición de algunos modelos populares con el fin de dar más detalle a las alternativas disponibles y al estado del arte.

El Zoom G1X se encuentra como una opción atractiva para principiantes debido a su bajo costo y alta versatilidad. Cuenta con más de 70 efectos, los cuales se pueden conectar en serie hasta cinco veces en simultáneo, patrones sintetizables de batería y 30 segundos para grabación de bucles. Si bien la calidad del empaquetado no es excelente, cuenta con un pedal de expresión, cuatro perillas y dos botones multiuso. Su interfaz de usuario tampoco es de lo mejor que se puede encontrar en el mercado, pero cumple bien su función con su pantalla minimalista.


*Figura 2.3.1 – Pedalera digital multi-efecto Zoom G1X Four.*

El Boss Pocket GT es un modelo diseñado con el fin de maximizar la eficiencia de tamaño con sus pequeñas dimensiones. Toma varios compromisos con esta decisión como la ausencia de un pedal de expresión o interruptores robustos accionables con el pie, pero compensa gran parte de ello gracias el acercamiento no tradicional que tiene con su interfaz gráfica. Ésta se encuentra implementada mediante una aplicación para smartphone que se conecta vía bluetooth para que el usuario pueda ajustar parámetros virtuales desde su celular.


*Figura 2.3.2 – Pedalera digital multi-efecto Boss Pocket GT.*

Un diseño más tradicional de alta gama sería el de la pedalera Boss ME-80. Cuenta con un gran acabado y una interfaz gráfica heredada del mundo analógico. El mismo consiste en una gran cantidad perrillas con funciones fijas etiquetadas sobre el producto y un display minimalista 7 segmentos con dos dígitos.


*Figura 2.3.3 - Pedalera digital multi-efecto Boss ME-80.*

La pedalera digital Pedalboard de Headrush se puede describir a grandes rasgos como un equivalente al ME-80 de Boss con una interfaz de usuario más moderna. Cada interruptor tiene un display con función programable y también posee una pantalla LED a color con la información de todos los efectos siendo utilizados. Esta última se puede manipular de manera táctil o mediante las perillas configurables ubicadas a su lado.


*Figura 2.3.4 - Pedalera digital multi-efecto Headrush Pedalboard.*

# 3 - Kit de Desarrollo

## 3.1 - Requerimientos

Para poder llevar a cabo este proyecto, es necesario contar con hardware capaz de cumplir ciertos requisitos. Partiendo por la CPU, según los estándares planteados previamente, la velocidad de procesamiento debe ser lo suficientemente rápida como para poder manipular muestras de audio de 16 bits a, por lo menos, 44,1KHz. Además, debe contar con una FPU para realizar operaciones con números decimales de forma óptima.

Con respecto a los periféricos, es necesario contar con una cantidad de memoria RAM y flash capaz de almacenar tanto el código generado como los assets utilizados en la interfaz de usuario. Se estimó, como mínimo, 192KB de RAM y 512 KB de flash. Los estrictos tiempos demandados por la aplicación exigen una manera eficiente de utilizar los ciclos de la CPU, por lo tanto, es necesario contar con canales DMA para realizar la transmisión de memoria entre la RAM y los otros periféricos. Por otra parte, es indispensable un conversor analógico-digital y digital analógico capaz de interactuar con el protocolo de audio I2S.

Los conectores requeridos para la aplicación son Jack 1/4 TS (mono) utilizados en la industria de la música.

Por último, para que el proceso de desarrollo sea ágil es necesario contar con un depurador de código.

## 3.2 - Proceso de Elección

Teniendo en cuenta los requerimientos mencionados de procesador, la primera alternativa explorada fue la de utilizar un dsPIC de Microchip dada su capacidad de realizar operaciones aritméticas a nivel de hardware. Fue eventualmente descartado ya que, si bien su arquitectura de 16 bit cumplía con los requisitos, en la comparativa, se descubrieron opciones más atractivas de 32 bit con un presupuesto similar y dentro del mercado nacional. De hecho, esta investigación de los productos disponibles llevó a concluir que, en lo subsiguiente, solamente se consideraran procesadores de este último tipo.

Investigando la oferta local, una opción disponible que se hizo interesante fue el kit de desarrollo de la empresa STM denominado STM32F407VG. Tiene un microcontrolador CortexM4 con las siguientes especificaciones:

- Arquitectura de 32-bit
- Velocidad de núcleo de 168MHz
- FPU de precisión simple
- Conjunto completo de instrucciones DSP
- 1MB de flash y 192KB de RAM
- 2 I2S full-duplex
- ADC/DAC internos de 12 bit
- 16 DMA streams
- 12 timers de 16 bit y dos timer de 32 bit

La categoría del kit se llama discovery, es un modelo enfocado a poder desarrollar aplicaciones con distintos periféricos externos de manera sencilla. En general estos integrados suelen ser

acelerómetros, micrófonos, interfaz SD, conector camera, interfaz LCD, interfaz SPDIF, interfaz ethernet, entradas/salidas de audio, entre otros.

Otro factor que influyó en la decisión de trabajar sobre esta plataforma fue que la placa incluía un DAC externo de la empresa Cirrus Logic llamado CS43L22, capaz de operar con muestras de 24 bit a una velocidad de muestreo máxima de 96KHz. Si bien el ADC no cumplía con el requisito, la primera versión del proyecto fue realizada con este kit el cual permitió avanzar en el entendimiento de la arquitectura Cortex, el desarrollo de código en la plataforma de STM y comunicación con dispositivos de audio externos (I2C, I2S, DMA).

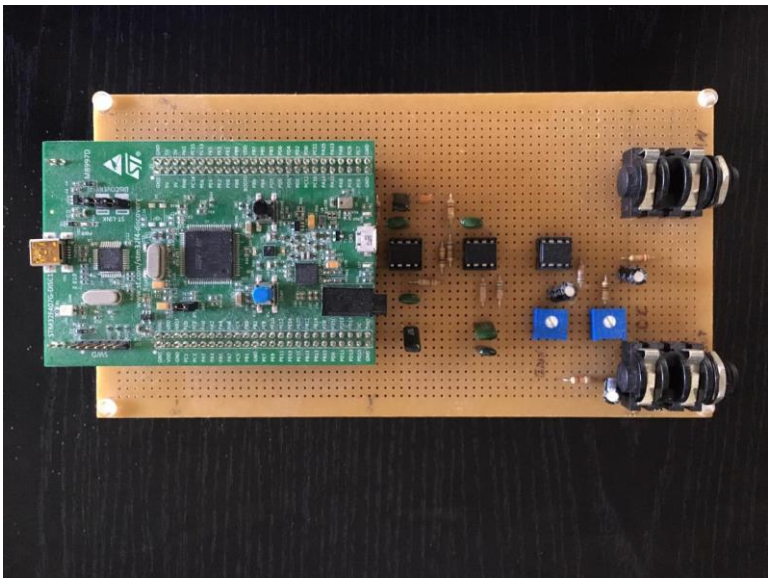

*Figura 3.2.1 – Prototipo del proyecto realizado sobre STM32F407VG.*

Una vez finalizada esta primera versión del proyecto, el próximo paso fue el de alcanzar los 24 bit de resolución tanto en el ADC como en el DAC. Para lograrlo se consideraron dos opciones: comprar un ADC externo o un audio códec (ADC/DAC). Para la primera los candidatos fueron el CS5340, el CS5343 y el ADS1252. En la segunda, el PCM3060 de Texas Instruments surgió como una buena alternativa.

Sin embargo, se encontraron dificultades en el mercado local motivo por el cual se inició una búsqueda en la oferta internacional la cual concluyó en la adquisición de la placa STM32F746G Discovery de STM. Se trata de otro kit de la misma categoría que contiene un audio códec integrado, entradas/salidas de audio y un lector de tarjetas SD. Además, posee una pantalla LCD táctil que permitió que se considerara la implementación de la interfaz usuario de manera gráfica, hecho que amplió la proyección del trabajo.

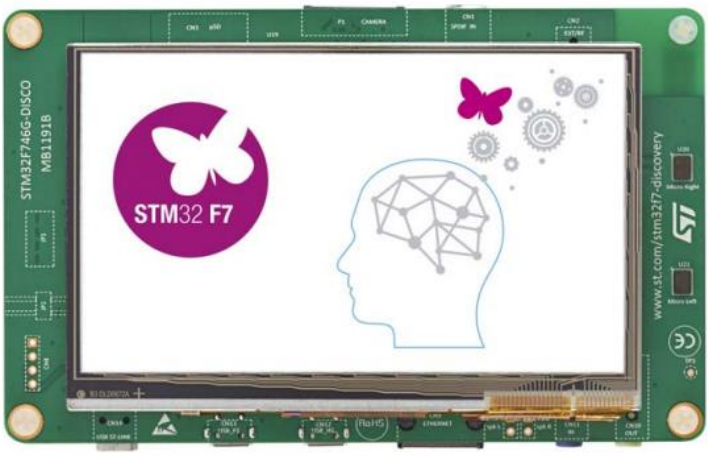


*Figura 3.2.2 – Vista inferior del kit de desarrollo STM32F746G Discovery.*


*Figura 3.2.3 - Vista superior del kit de desarrollo STM32F746G Discovery.*

## 3.3 - Especificaciones

El kit de desarrollo cuenta con varias facilidades, entre ellas:

- Microcontrolador Cortex M7 de 32 bit con 1 MB de memoria flash y 340 KB de memoria RAM embebidos.
- Chip programador/debugger ST-LINK on-board accessible mediante mini USB.
- Capacidad de correr los sistemas operativos embebidos habilitados por Mbed Enabled.
- Dos puertos micro USB para comunicación virtual, dispositivos de almacenamiento masivo y debug. Ambos soportan ser dispositivos y host, uno soporta Full Speed (12 Mbps) mientras que el otro soporta Full Speed y High Speed (480 Mbps).
- TFT-LCD (Thin Film Transistor-Liquid Crystal Display) de 4,3 pulgadas y 480x272 píxeles con sensor de tacto capacitivo.

- Conector DCMI de cámara para aplicaciones de video, IoT y de automatización de hogar.
- Codec de Audio WM8994 conectado por la interfaz SAI. Cuenta con dos salidas estéreo para parlantes, una entrada y una salida de audio (line in y line out) con jack de 3,5 mm, dos entradas a través de micrófonos MEMS (microelectro-mechanical systems) y una entrada de audio digital SPDIF RCA.
- Dos botones (uno de reset y otro de usuario).
- 128 Mb de memoria flash Quad-SPI NOR y 128 Mb de SDRAM (con 64 Mb accesibles).
- Conector de tarjeta microSD.
- Conector EEPROM I2C para dongle de RF.
- Conector Ethernet RJ45 de 10/100 Mbit.
- Pin headers de Arduino Uno V3 destinados a compatibilidad con placas shield.
- Cinco fuentes de alimentación totales entre los 3 USB, el conector VIN de Arduino y el conector externo lateral de 5V.

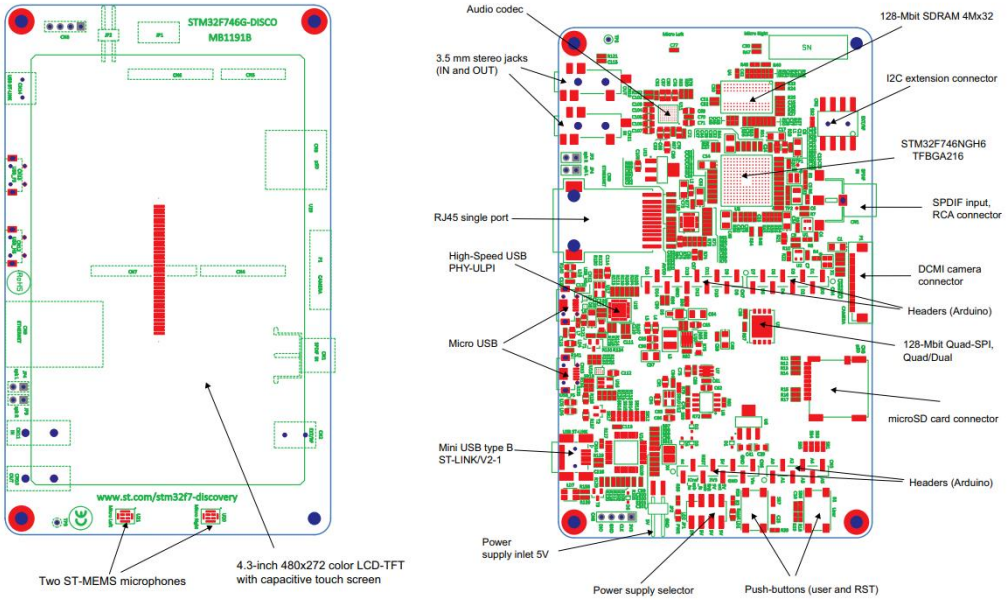


*Figura 3.3.1 - Vista superior e inferior (respectivamente) del kit de desarrollo STM32F746G Discovery con detalle de componentes.*

# 4 - Conversión A/D – D/A

## 4.1 - Conversores Convencionales

Las señales de audio son, por naturaleza, analógicas; se producen de manera analógica (en la mayoría de los casos) y se propagan de manera analógica. Sin embargo, ya que se desea procesar estas señales mediante un DSP, se precisa de un equivalente digital de estas señales analógicas. También es muy importante que esta señal digital sea lo más fidedigna posible respecto a la señal analógica original, de este modo, es posible asegurar una buena definición en el audio.

Existen diversos métodos para implementar un sistema de conversión analógica a digital y viceversa. El más común se muestra en la Figura 4.1.1.

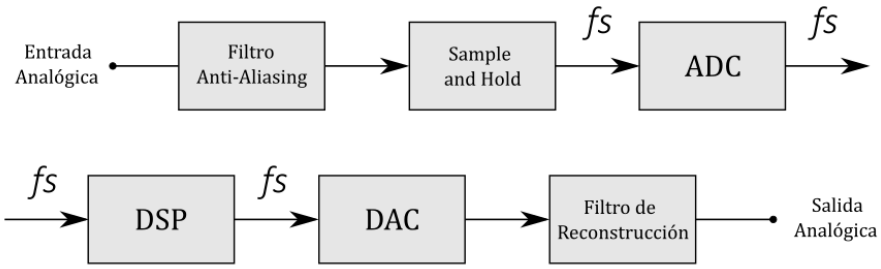


*Figura 4.1.1 - Diagrama en bloques de una conversión A/D - D/A.*

Las señales analógicas presentan imagen y dominio continuo o, lo que es lo mismo en este caso, amplitud y tiempo. Para lograr una señal digital que se puede procesar en un DSP, se deben discretizar ambos.

El tiempo se discretiza en intervalos periódicos dictados por un clock en un proceso denominado muestreo. La tasa a la cual se producen estos intervalos se denomina frecuencia de muestreo (fs). Más adelante se discutirá la importancia de este parámetro de diseño.

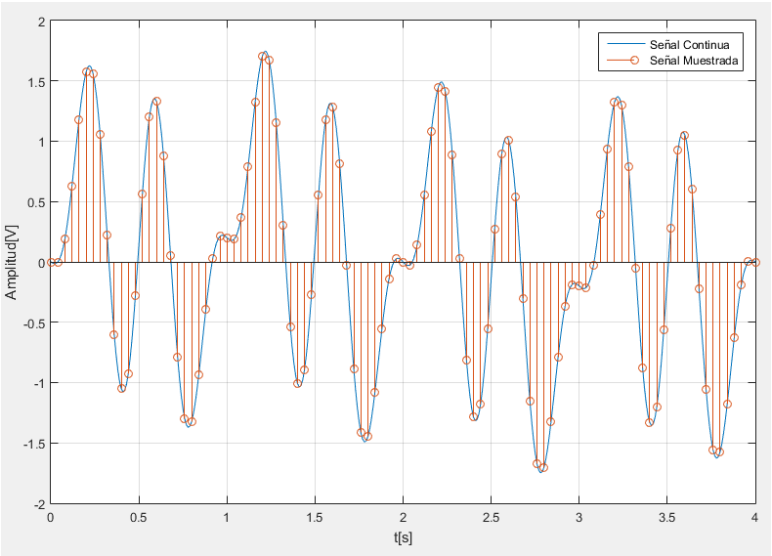


*Figura 4.1.2 - Muestreo de una señal analógica a 25Hz.*

El proceso que digitaliza la amplitud se denomina cuantización y consiste en una operación no lineal que mapea valores continuos a un conjunto predeterminado de valores discretos. Por último, se codifica la cuantización asignado una palabra de bits a cada uno de estos valores

discretos para poder procesarse adecuadamente. Hay varias maneras de codificar una cuantización, pero la más conocida es la modulación PCM (modulación por pulsos codificados). Se trata una técnica que asigna niveles distintos a cada palabra de bits buscando representar de la mejor manera posible el valor de la señal en el tiempo.

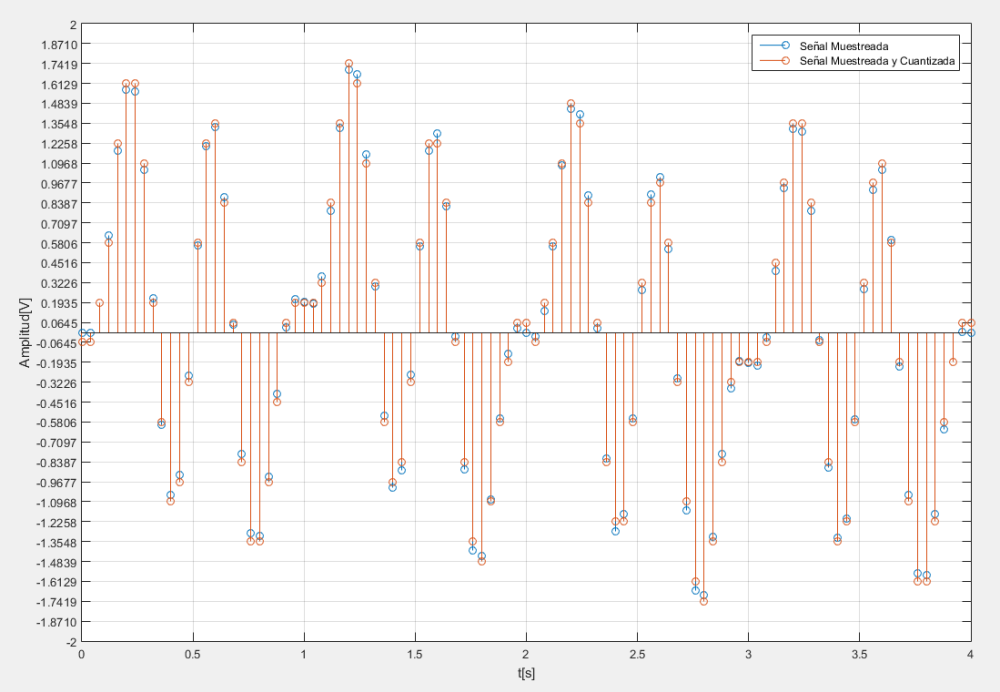


*Figura 4.1.3 - Cuantización de una señal muestreada mediante modulación PCM (32 niveles o 5 bits linealmente distribuidos entre -2V y 2V).*

Las discrepancias entre la señal muestreada y la señal muestreada y cuantizada empeoran la performance del sistema y producen lo que se denomina ruido de cuantización. Si se trabaja con modulación PCM, aumentar la cantidad de bits siempre reduce esta fuente de ruido. También, para cierto tipo de señales, puede ser beneficioso distribuir los niveles de cuantización de una manera no lineal. Esto permite aumentar el nivel de detalle en ciertos umbrales mientras que, en otros, se pierde resolución.

El muestreo, la cuantización y la codificación ocurren entre los bloques Sample and Hold y ADC. Primero, el bloque Sample and Hold recibe el clock de muestreo y retiene el voltaje de la entrada analógica mientras que el ADC realiza la conversión, esto se repite periódicamente con un período de Ts segundos (Ts = 1/fs). El ADC es el encargado de devolver la señal muestreada, cuantizada y codificada. A lo largo de los años se diseñaron varios sistemas de ADC como por el ejemplo el ADC SAR (Registro de Aproximaciones Sucesivas) o el ADC integrador de doble rampa. Sin embargo, el funcionamiento interno de estos está más allá del alcance del trabajo.

Una vez que se tiene una señal PCM, se está en condiciones de transmitir las muestras al DSP; aquí es donde ocurre el procesamiento digital. Para poder escuchar los resultados, la señal procesada debe convertirse nuevamente a una señal analógica. Aquí es donde comienza el proceso inverso: la conversión digital a analógica.

El DAC toma como entrada el flujo de bits transmitido por el DPS y logra una señal analógica a su salida. Se puede realizar completamente con componentes pasivos como en la famosa red de resistencias R-2R.

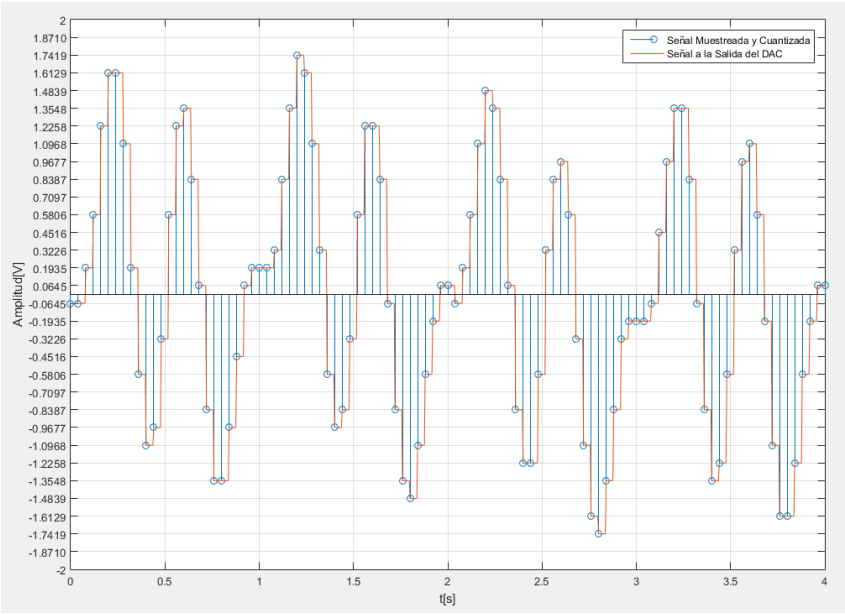


*Figura 4.1.4 - Conversión de una señal muestreada y cuantificada a una señal analógica mediante un DAC.*

Si bien la señal ya es analógica, presenta componentes de alta frecuencia no deseados que introducen una distorsión. Para resolver esto, se coloca un filtro pasa bajos de reconstrucción que permite separar las componentes de frecuencia de interés en la señal de audio.

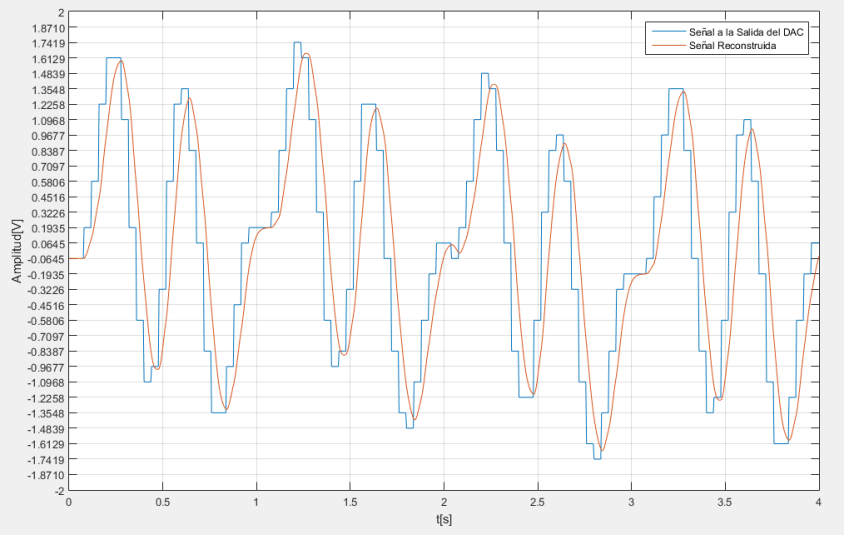


*Figura 4.1.5 - Reconstrucción de la señal de un DAC mediante un filtro pasa bajos de cuarto orden.*

En la Figura 4.1.6 se pueden apreciar las distorsiones introducidas a lo largo de este sistema. Gran parte de ellas se deben al ruido de cuantización y a la respuesta en fase del filtro de reconstrucción. Esta última, sin embargo, no afecta de manera significativa la percepción humana de una señal de audio, por lo cual, generalmente no es tomada en cuenta. Hay otro tipo de fenómeno que no se puede apreciar en este caso, pero también puede conspirar contra la conversión A/D – D/A: el aliasing.

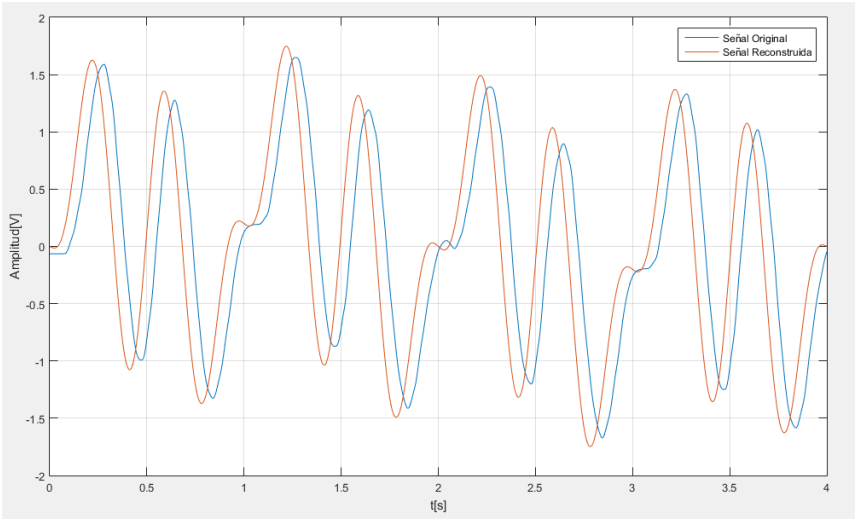


*Figura 4.1.6 - Comparación entre una señal y su copia reconstruida luego de una conversión A/D – D/A suponiendo que no se realiza ningún procesamiento digital en el DSP.*

Para poder entender el aliasing y la necesidad de utilizar un filtro anti-aliasing antes del bloque Sample and Hold, hay que analizar lo que le ocurre a la señal en el dominio de la frecuencia.

La señal muestreada se puede definir como la señal original multiplicada por un tren periódico de impulsos con período Ts:

$$x_s(t) = x(t) \cdot p(t) = x(t) \cdot \sum_{k=-\infty}^{k=+\infty} \delta(t - kT_s)$$

Se puede demostrar que la transformada de Fourier de esta expresión es la siguiente:

$$X_s(f) = f_s \cdot \sum_{k=-\infty}^{k=+\infty} X\big(2\pi(f - kf_s)\big) \qquad \text{Ec. 4.1.1}$$

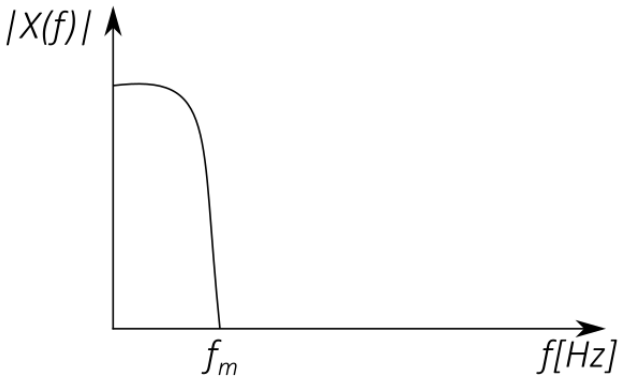


*Figura 4.1.7 - Espectro de frecuencias de una señal analógica genérica a ser muestreada según la Ec. 4.1.1.*

Como se puede ver en la Figura 4.1.8, esto implica que las componentes de frecuencias de la señal de entrada se repiten con un período de fs Hz en doble banda lateral una vez que se produce el muestreo. Todo este agregado no contiene información y debe filtrarse cuando la señal se reconstruya para evitar distorsiones. Debido a esto, es muy importante tener algún tipo de ancho de banda entre la frecuencia máxima de la información de la señal analógica (fm) y el comienzo de la primera banda lateral izquierda (fs-fm). En un caso extremo y suponiendo que

se puede construir un filtro ideal, este ancho puede llegar a ser nulo, de allí se deduce la condición de Nyquist para el muestreo: la frecuencia de muestreo debe ser por lo menos dos veces mayor a la frecuencia máxima de la señal a muestrear de tal manera de no perder información en el proceso.

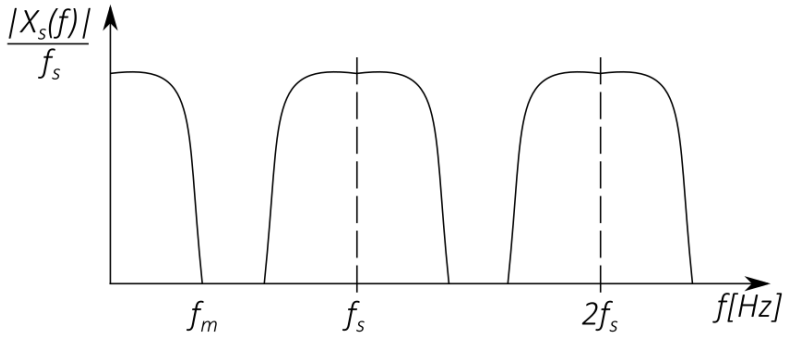


*Figura 4.1.8 - Espectro de frecuencias de una señal analógica muestreada por un tren de impulsos.*

La tal llamada pérdida de información es introducida por un fenómeno denominado aliasing y puede ocurrir de dos maneras. Primero, si se utiliza una frecuencia de muestreo demasiado baja para la aplicación o, lo que es lo mismo, si no se cumple con el criterio de Nyquist. Tal como se puede ver en la Figura 4.1.9, se comienzan a solapar las repeticiones espectrales hasta el punto en que empiezan a influir dentro de la banda de interés.

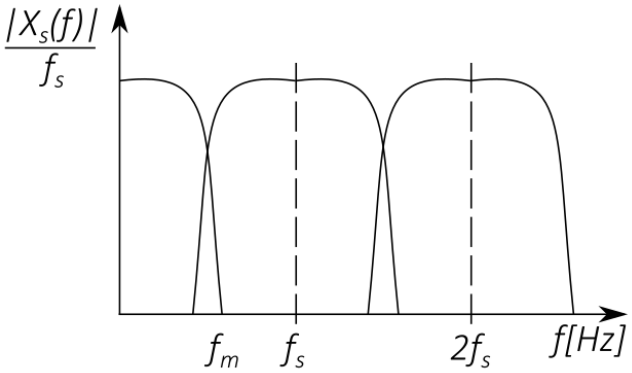


*Figura 4.1.9 - Espectro de frecuencias de una señal muestreada por debajo de la frecuencia de Nyquist.*

La otra alternativa no ocurre debido a una pobre elección de una frecuencia de muestreo, sino a la naturaleza de la propia señal a medir. Hasta aquí, se supuso que esta señal contiene la información de interés de manera pura. Sin embargo, en la mayoría de los casos, y especialmente en las señales de audio, esto no ocurre. Suelen aparecer ruidos fuera de banda introducidos tanto por la transmisión como por harmónicos producidos en el mismo instrumento. Éstos pueden producir aliasing ya que las bandas laterales izquierdas también pueden reflejarlos dentro de la banda de interés.

El filtro anti-aliasing, entonces, busca reducir de la mejor forma posible todo el ruido fuera de banda de tal manera de tener solamente componentes de señal hasta fm antes de realizar el muestreo.

Como conclusión, este esquema de conversión tradicional tiene varias cabidas para mejoras. Primero, la única opción para reducir el ruido de cuantización es la de incrementar la cantidad de bits del ADC, lo cual aumenta la complejidad y el costo de los integrados, especialmente si se desea utilizar una alta frecuencia de muestreo. Para un ADC SAR, se estima que por encima de

los 16 bits de resolución ya no vale la pena agregar más bits ya que la densidad espectral del ruido de cuantización suele estar por encima del valor de los bits adicionales. El segundo problema tiene que ver con los filtros; para realizar procesamiento digital en una señal analógica se precisan dos filtros analógicos (uno de anti-aliasing y otro de reconstrucción). No solamente eso, sino que estos filtros suelen ser excesivamente complejos porque no hay mucho ancho de banda entre fm y fs/2. En aplicaciones de audio, es común utilizar filtros Butterworth o Chebyshev de sexto u octavo orden y el problema aquí es que estos circuitos analógicos pueden ser tan malos como los problemas que tratan de prevenir.

## 4.2 - Sistemas Multi-tasa y Conversión Delta-Sigma

La solución que ofrecen los sistemas multi-tasa (multirate) a los esquemas de conversión tradicionales viene por dos partes: el sobre-muestreo (oversampling) y la conformación del ruido (noise shaping). Su esquema se muestra en la Figura 4.2.1.

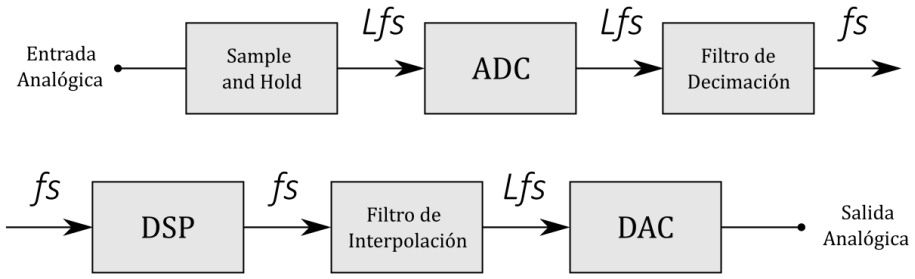


*Figura 4.2.1 - Diagrama en bloques de una conversión A/D - D/A en un sistema multi-tasa con un factor de sobre-muestreo L.*

El sobre-muestreo, como su nombre lo indica, consiste en tomar muestras de una señal analógica muy por encima de la frecuencia de Nyquist. Como se puede apreciar en la Figura 4.2.2, esto logra un ancho de banda interesante entre la señal original y la primera réplica espectral. Gracias a este fenómeno, se puede reducir significativamente o quitar por completo el filtrado analógico tanto en la entrada como en la salida ya que las componentes no deseadas suelen estar fuera de la banda audible. Incluso si se precisa un filtro analógico, se puede implementar con sencillez mediante un filtro de bajo orden de manera integrada.


*Figura 4.2.2 - Espectro de frecuencias de una señal sobremuestreada en un factor de L veces.*

Otra gran ventaja de aplicar sobre-muestreo es la de la reducción del ruido de cuantización. Para entender esto, resulta útil analizar lo que ocurre con la densidad espectral de potencia del ruido de cuantización. Primero, se puede obtener un modelo linealizado de un cuantificador mediante una fuente de ruido aditiva con la siguiente función de densidad de probabilidad uniforme:

$$f_Q(q) = \begin{cases} \frac{1}{\Delta}, -\frac{\Delta}{2} < q < \frac{\Delta}{2} \\ 0, \ otro\ q \end{cases}$$

Donde Q es la variable aleatoria que representa la diferencia entre la señal original y la señal cuantizada y Δ es la diferencia entre dos niveles de cuantización cualesquiera (suponiendo una distribución lineal de dichos niveles).

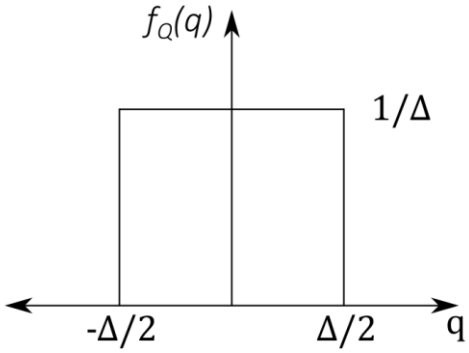


*Figura 4.2.3 - Función de densidad de probabilidad del error de cuantización en un modelo linealizado.*

La cantidad total de potencia de ruido de cuantización debe ser igual a la varianza de Q:

$$\mu_Q = E[Q] = \int_{-\infty}^{+\infty} q \cdot f_Q(q)\, dq = 0 \qquad \text{Ec. 4.2.1}$$

$$\sigma_Q^2 = E\left[(Q-\mu_Q)^2\right] = E[Q^2] = \int_{-\infty}^{+\infty} q^2 \cdot f_Q(q)\, dq = \frac{\Delta^2}{12} \qquad \text{Ec. 4.2.2}$$

Una primera observación que se puede realizar es que este resultado no depende de la frecuencia de muestreo. Sin embargo, en esta configuración, esta potencia se distribuye uniformemente en un ancho de banda igual a la mitad de la frecuencia de muestreo. Esto significa que, si bien el sobre-muestreo no puede reducir la cantidad total de potencia de ruido de cuantización, sí puede lograr una disminución de la potencia total dentro de la banda de interés (como se puede ver en la Figura 4.2.4).

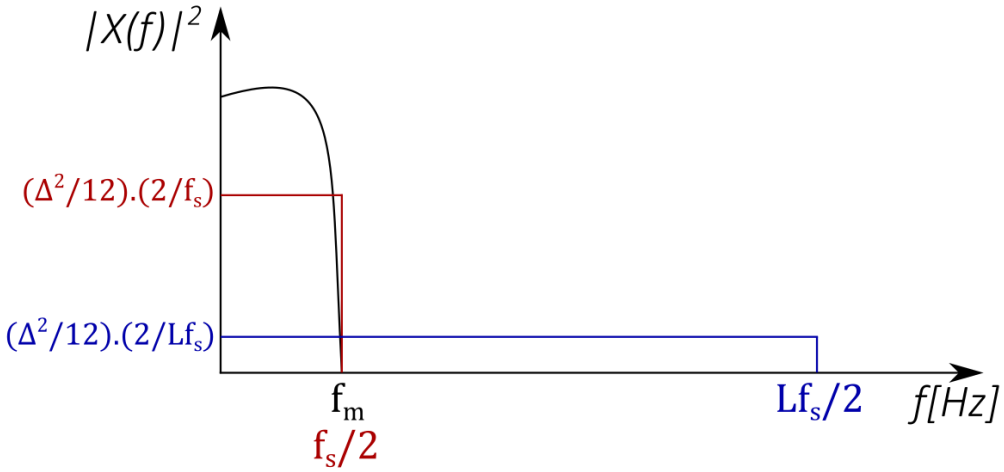


*Figura 4.2.4 - Comparación de la densidad espectral del ruido de cuantización para un sistema con muestreo a la frecuencia de Nyquist contra un sistema de sobre-muestreo por un factor L.*

Esta técnica de dejar la potencia del ruido de cuantización fuera de banda se puede explotar aún más si se implementa un sistema de conformación del ruido. La idea básica es la de crear una

función de transferencia que deforme esta característica de ruido de tal manera que se reduzcan las componentes dentro de banda a costas de incrementar las componentes fuera de banda.

El conversor más conocido que puede implementar sobre-muestreo y conformación de ruido en alta performance y bajo costo es el conversor Delta-Sigma. La Figura 4.2.5 propone un diagrama en bloques de este sistema.

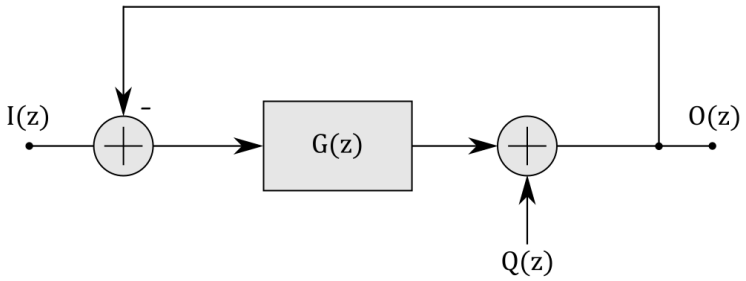


*Figura 4.2.5 - Diagrama en bloques equivalente de un conversor Delta-Sigma de primer orden con cuantificador linealizado.*

Si bien los conversores analógicos/digitales no son lineales por el hecho de tener un bloque cuantificador, en este caso también se puede aproximar que se trata de un generador de ruido con una función de densidad de probabilidad uniforme. De esta manera, se pueden calcular las funciones de transferencia consiguiendo resultados que se adecuan bastante bien a la realidad.

El nombre Delta-Sigma se elabora de un conversor pariente llamado conversor Delta. Éste último solamente calcula la diferencia entre la entrada y la salida y devuelve un uno binario si la entrada es menor a la salida y un cero binario en la situación inversa. Allí, "Delta" hace referencia a la operación de resta. En este caso, se agrega el término "Sigma" ya que el bloque G(z) no tiene transferencia unitaria y actúa como integrador de tal manera que:

$$G(z) = \frac{z^{-1}}{1 - z^{-1}}$$

Se define la función de transferencia del ruido NTF(z) como la relación entre la salida O(z) y el ruido Q(z) al pasivar la entrada I(z). Ésta se puede calcular como:

$$NTF(z) = \frac{O(z)}{Q(z)}\Big|_{I(z)=0} = \frac{1}{1 + G(z)} = 1 - z^{-1} \qquad \text{Ec. 4.2.3}$$

De manera similar, se define la función de transferencia de la señal STF(z) como la relación entre la salida O(z) y la entrada I(z) pasivando el generador de ruido Q(z). La expresión correspondiente es la siguiente:

$$STF(z) = \frac{O(z)}{I(z)}\Big|_{Q(z)=0} = \frac{G(z)}{1 + G(z)} = z^{-1} \qquad \text{Ec. 4.2.4}$$

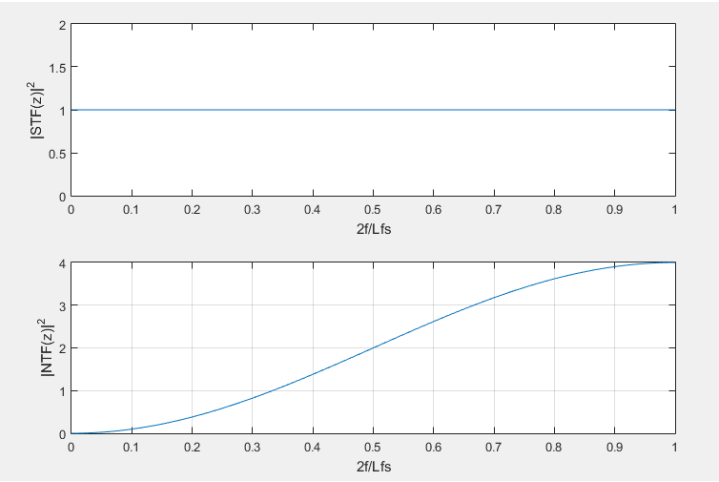


*Figura 4.2.6 - Respuesta de potencia de las funciones de transferencia de la señal y del ruido en un conversor Delta-Sigma de primer orden.*

En la Figura 4.2.6 se ilustra el objetivo del diseño: la señal no es modificada en amplitud, pero la densidad espectral de potencia del ruido se logra reducir dentro de la banda de interés mientras que la mayor cantidad de la potencia total queda fuera de banda.

Debido a que las frecuencias de sobre-muestreo son tan altas, el cuantificador de este conversor es lo más sencillo que puede llegar ser; devuelve un uno lógico si su entrada está por arriba de cero y devuelve un cero lógico si su entrada está por debajo de cero. En otras palabras, se trata de un cuantificador de un solo bit. Esto no es un gran problema ya que con todas las medidas que se toman para reducir el ruido de cuantización, la resolución equivalente es mucho mayor a un bit.

La señal de flujo de bits a la salida de un conversor Delta-Sigma se denomina PDM (modulación por densidad de pulsos). En crudos términos, es una señal de ceros y unos a muy alta velocidad. A diferencia de la modulación PCM donde se representa el valor temporal de la señal, en PDM, el promedio instantáneo del flujo de bits representa el promedio instantáneo de la señal. Un ejemplo se muestra en la Figura 4.2.7.

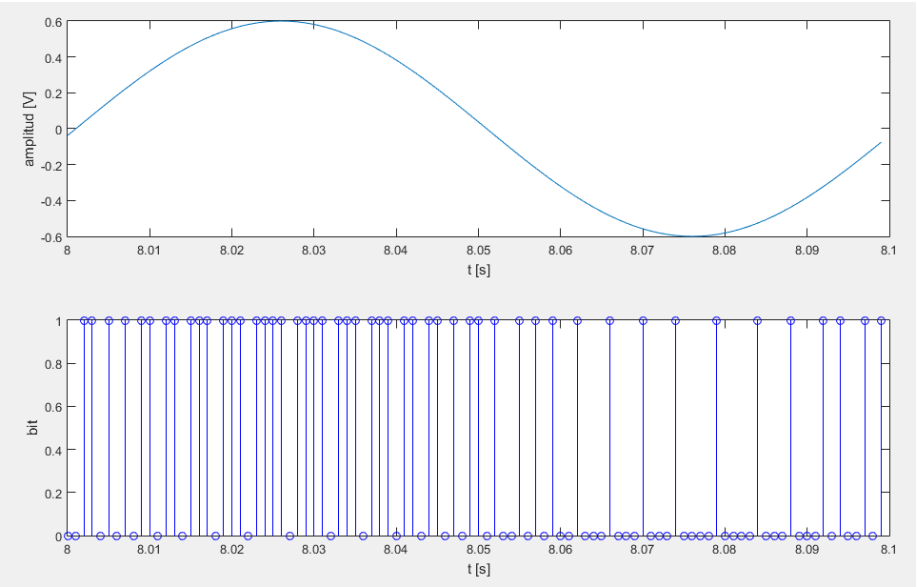


*Figura 4.2.7 - Representación de una señal sinusoidal en PDM con un conversor Delta-Sigma de primer orden y un factor de sobre-muestreo de 50 veces.*

A medida que se aumenta el orden del conversor, se reduce el ruido de cuantización dentro de banda gracias a la característica de conformación de ruido. La Figura 4.2.8 muestra una comparativa de la función de transferencia de ruido para distintos órdenes del modulador.

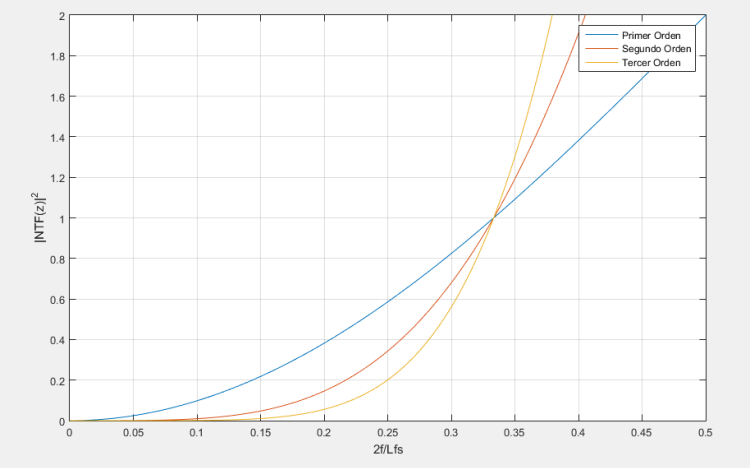


*Figura 4.2.8 - Comparación de la función de transferencia de ruido para conversores Delta-Sigma de distintos órdenes.*

Una manera de lograr mayores órdenes en el diseño es la de agregar más lazos con integradores antes de la etapa cuantificadora como se puede ver en la Figura 4.2.9. Estas arquitecturas permiten llegar a resoluciones equivalente de 24 bits (como se utiliza en este trabajo) o incluso más.

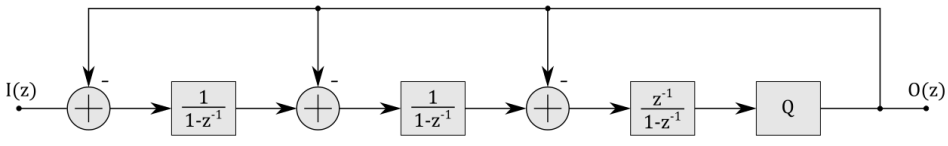


*Figura 4.2.9 - Modulador Delta-Sigma de tercer orden.*

Para poder hacer procesamiento digital sobre una señal PDM, primero se debe hacer algún tipo de traducción para obtener la señal PCM equivalente a la tasa de muestro normal del sistema; de este trabajo se encarga el filtro de decimación. Además de esto, también se busca remover el ruido de cuantización fuera de banda y el aliasing que pueda llegar a quedar mediante filtrado. Este filtro se implementa totalmente de manera digital.

Si una señal tiene un factor de sobre-muestreo de 50 veces, se deberán quitar 49 muestras por cada 50. Las muestras se van decimando progresivamente a medida que se recorre una serie de filtros pasa bajos. La muestra restante queda en codificación PCM gracias a que estos filtros devuelven la media móvil de lo que van procesando lo cual devuelve el valor de la señal en el tiempo.

Hacer la conversión digital/analógica en estos sistemas se reduce al filtro de interpolación. Éste toma como entrada la señal PCM procesada por el DSP y devuelve la señal PDM equivalente agregando muestras para llegar a la misma tasa de sobre-muestreo que se llegó en la conversión analógica/digital. Para pasar una señal PDM a una señal analógica solamente hace falta utilizar un filtro pasa bajos de orden reducido.

# 5 - Flujo de Audio

## 5.1 - Códec

La señal a procesar ingresa a la placa mediante el conector de audio analógico (jack) de 3.5mm color violeta en el kit de desarrollo. Este conector posee tres contactos, uno referencia a tierra y los otros dos llevan el canal izquierdo y derecho respectivamente. Si bien en la mayoría de los casos esta señal es tipo mono por naturaleza, también esta soportado el caso de señales de audio estéreo en la entrada.

Inmediatamente, se utilizan capacitores de 1uF en serie al camino de ambas señales para proteger a la placa eliminando cualquier componente de continua que puedan llegar a tener éstas. De aquí, entran directamente en los pines IN1LN e IN1RN del códec de audio wm8994 de Wolfson Electronics integrado en el kit.

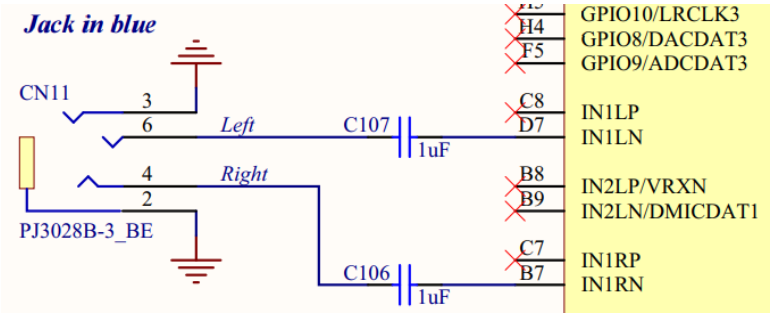


*Figura 5.1.1 - Conexión de la entrada de línea al códec de audio integrado en el kit de desarrollo.*

El propósito del códec (Codificador/Decodificador) de audio es el de manipular la señal de entrada (y luego la de salida) de tal manera que el DSP quede liberado para realizar el procesamiento digital. Si bien su función principal es la de hacer las conversiones analógico/digital y digital/analógico, este integrado también brinda muchas facilidades adicionales que son muy útiles para trabajar con señales de audio. En la Figura 5.1.2, se puede apreciar todos los bloques importantes que componen a este chip y como están conectados.

La programación del códec se realiza mediante la escritura de sus registros. Los registros tienen 16 bits y se identifican tanto por su número de registro como por un código hexadecimal único (este último es el que se debe invocar en el código). Dentro de cada uno de estos registros, se encuentran las configuraciones, las cuales se identifican vagamente a través de un código de texto que levemente representa lo que hace (por ejemplo: IN1LN_TO_IN1L es una configuración de un bit dentro del registro R40(28h) que conecta la entrada IN1LN al amplificador IN1L).

Mediante este proceso de escritura, se puede llegar a la configuración deseada del chip según la aplicación. Para explicar la configuración utilizada en este trabajo, se estudiará cada etapa por separado.

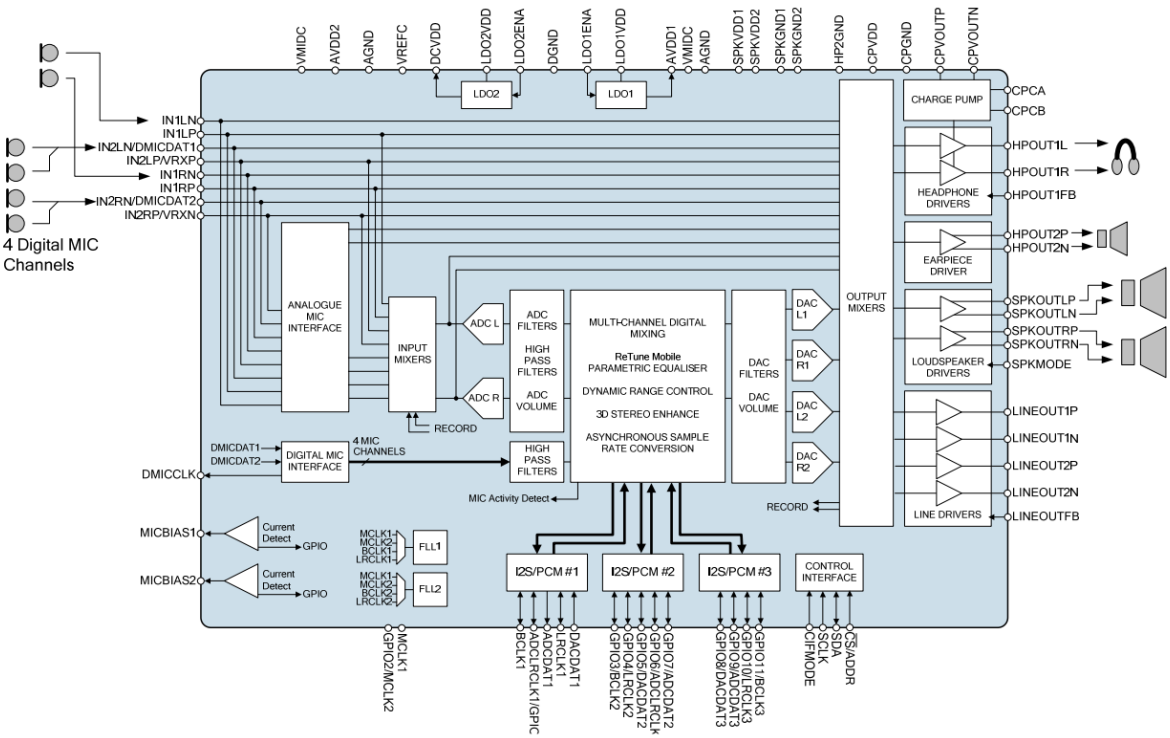


*Figura 5.1.2 - Diagrama en bloques del códec de audio wm8994.*

Primero se encuentran lo que la hoja de datos denomina caminos de entrada analógicos e incluye todo lo que ocurre desde el ingreso de la señal analógica hasta la conversión analógico digital por parte de los ADC. Las entradas IN1LN e IN1RN se conectan a la interfaz analógica de PGA (amplificadores de ganancia programable) mediante las configuraciones IN1LN_TO_IN1L e IN1RN_TO_IN1R respectivamente. Los PGA se habilitan con IN1L_ENA e IN1R_ENA. También se deben deshabilitar el mute y seleccionar la ganancia (desde -16,5 dB hasta 30 dB con pasos de 1,5 dB) con IN1L_MUTE / IN1L_VOL [4:0] y IN1R_MUTE / IN1R_VOL [4:0] pero estos deben ser los últimos pasos para inicializar el códec.

Una sencilla inspección de la Figura 5.1.3 revela que estos PGA soportan entrada balanceada (diferencial) y no balanceada (single). La conexión utilizada en este caso es la última así que las señales negativas se conectan en la entrada inversora del amplificador mientras que la entrada no inversora se aprovecha para una tensión interna denominada VMID. Ésta se configura mediante VMID_SEL [1:0] y se calcula con un divisor resistivo de dos resistencias de 40 kΩ entre AVDD1 y tierra. Esto provee una tensión de AVDD1/2 que se suma como valor de continua a las salidas de los PGA para permitir la máxima excursión simétrica de las señales de audio cuando entren la interfaz digital.

Para funcionar correctamente, el códec precisa utilizar esta tensión a través de un buffer que se habilita con VMID_BUF_ENA. Además de una tensión de offset, los circuitos analógicos del chip necesitan una corriente de polarización habilitada mediante BIAS_ENA.

A la salida de los PGA, se encuentran los mezcladores de entrada MIXINL y MIXINR. Ya que solamente se dispone de un ADC en el canal izquierdo y un ADC en el canal derecho para las cinco fuentes de audio posible, estos mezcladores permiten sumar las señales de entrada y combinarlas para lograr una sola señal en cada ADC. Los mezcladores de habilitan a través de MIXINL_ENA y MIXINR_ENA mientras que el camino de conexión con los amplificadores se establece con IN1L_TO_MIXINL e IN1R_TO_MIXINR. Cuando se trata de una entrada que viene

desde los PGA, se tiene una opción de amplificar 30 dB a la señal antes de realizar la suma. Esto se puede configurar mediante los controles IN1L_MIXINL_VOL e IN1R_MIXINR_VOL.

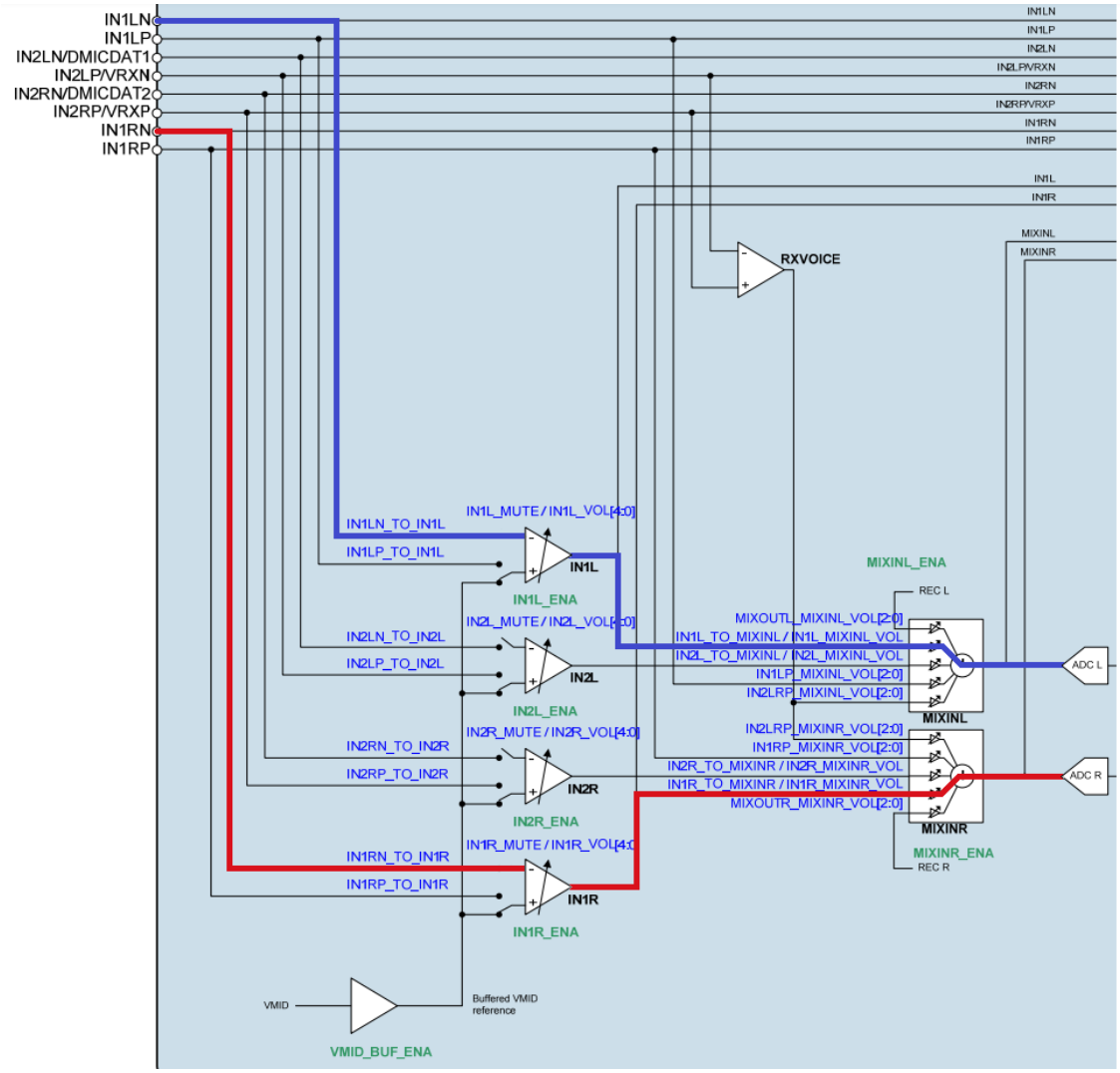


*Figura 5.1.3 - Caminos de entradas analógicas del códec wm8994. Las configuraciones utilizadas de canal izquierdo y derecho se encuentran resaltadas en azul y rojo respectivamente.*

Como fase final del camino analógico, los mezcladores pasan la señal hacia los ADC quienes realizan una conversión delta-sigma de 24 bits (como se explicó en el capítulo 4) y un filtro de decimación para dar una señal digital PCM a la siguiente fase del códec.

Los ADC también constituyen la primera parte del camino de entrada digital (ver Figura 5.1.4) y se habilitan con ADCL_ENA y ADCR_ENA. Las muestras digitales se transportan mediante un contenedor regulado por una interfaz de audio. AIF1 y AIF2 son las dos opciones de interfaces digitales para utilizar. La AIF1 es obligatoria para las señales que salen directamente de los ADC, por lo tanto, en este proyecto se trabajó con ésta.

AIF1 genera un contenedor TDM con dos ranuras temporales; ambas ranuras comienzan con un canal izquierdo y terminan con un canal derecho. La cantidad de bits por canal se selecciona mediante AIF1_WL [1:0] y su formato se establece con AIF1_FMT [1:0]. En este caso, se utilizan palabras de 24 bits en formato I2S. La frecuencia de muestreo del ADC está vinculada con la frecuencia de la interfaz que se escribe con el control AIF1_SR.

La salida del ADCL se inserta en AIF1, ranura temporal 0, canal izquierdo con ADC1L_TO_AIF1ADC1L, de manera similar, la salida del ADCR se inserta en AIF, ranura temporal 0, canal derecho con ADC1R_TO_AIF1ADC1R. La ranura 1 se transporta vacía ya que ésta solo puede llevar información de micrófonos digitales.

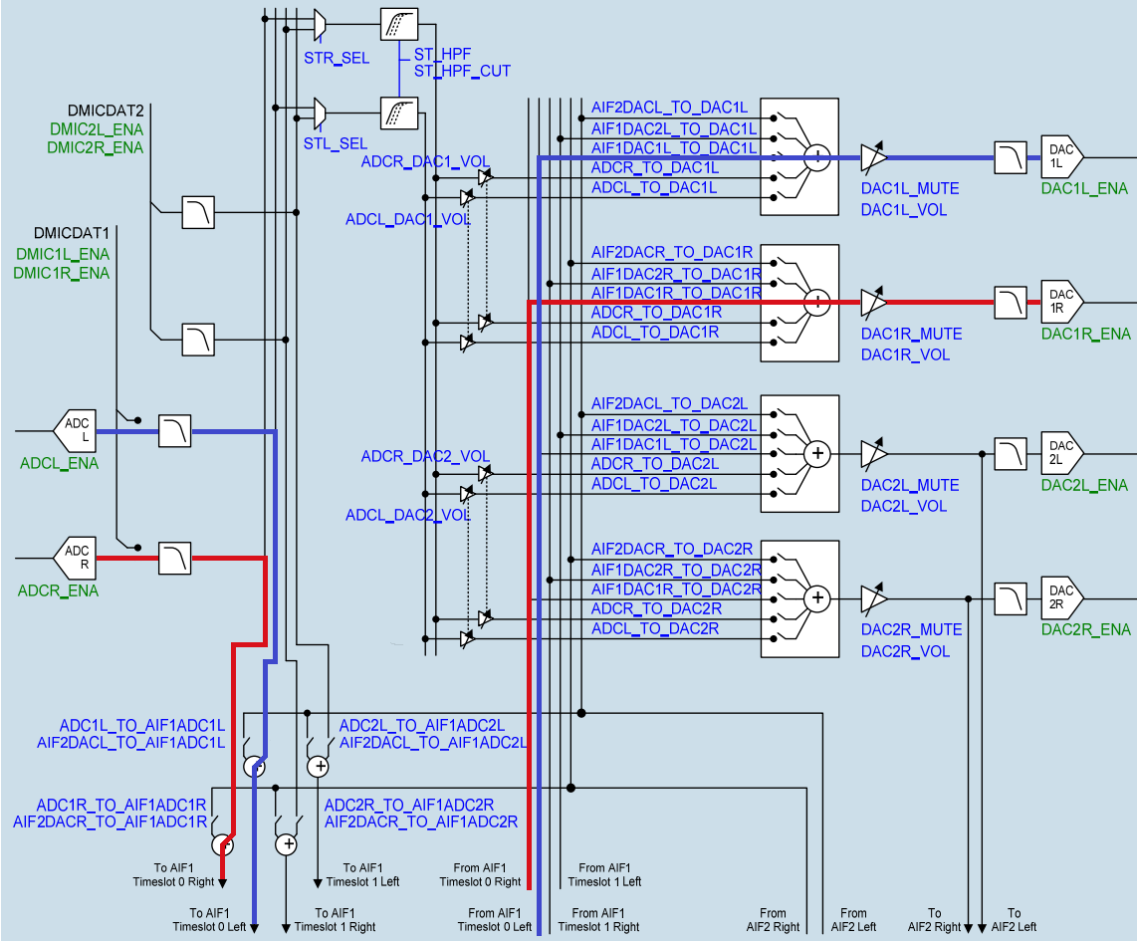


*Figura 5.1.4 - Caminos de entradas y salidas digitales del códec wm8994. Las configuraciones utilizadas de canal izquierdo y derecho se encuentran resaltadas en azul y rojo respectivamente.*

Una vez que se multiplexan los canales digitales, se pasa a la etapa de DRC (control de rango dinámico). Hay un bloque DRC en el camino de entradas digitales y otro en camino de salidas digitales, sin embargo, solamente se puede habilitar uno de estos simultáneamente en la misma ranura temporal (a través de AIF1ADC1L_DRC_ENA / AIF1ADC1R_DRC_ENA y AIF1DAC1_DRC_ENA respectivamente). Ya que se habilitaron los DRC en el camino de entradas digitales, hizo falta activar los filtros pasa altos correspondientes mediante AIF1ADC1L_HPF y AIF1ADC1R _HPF para asegurar que se trabaje sin valor de continua. Adicionalmente, la frecuencia de corte de estos filtros se configuró en AIF1ADC1_HPF_CUT [1:0] para tener un modo de alta fidelidad (lo más baja posible para no afectar la respuesta de graves).

Los DRC tienen una gran cantidad de opciones que se encuentran entre los registros 440h y 444h. La función principal es la de realizar una compresión parametrizada de envolvente para evitar cambios drásticos en el volumen de la señal. Se puede elegir entre activar un modo anti saturación (protección contra desbordes en las muestras) y un modo de liberación rápida (protección contra picos de señal producidos de forma esporádica). También se puede habilitar una compuerta de ruido (noise gate) para minimizar ruidos estáticos con alta atenuación a las

señales de bajo nivel. Finalmente, se puede realizar una detección de nivel de envolvente y vincularlo con un pin GPIO para generar una interrupción externa (generalmente utilizado con micrófonos).

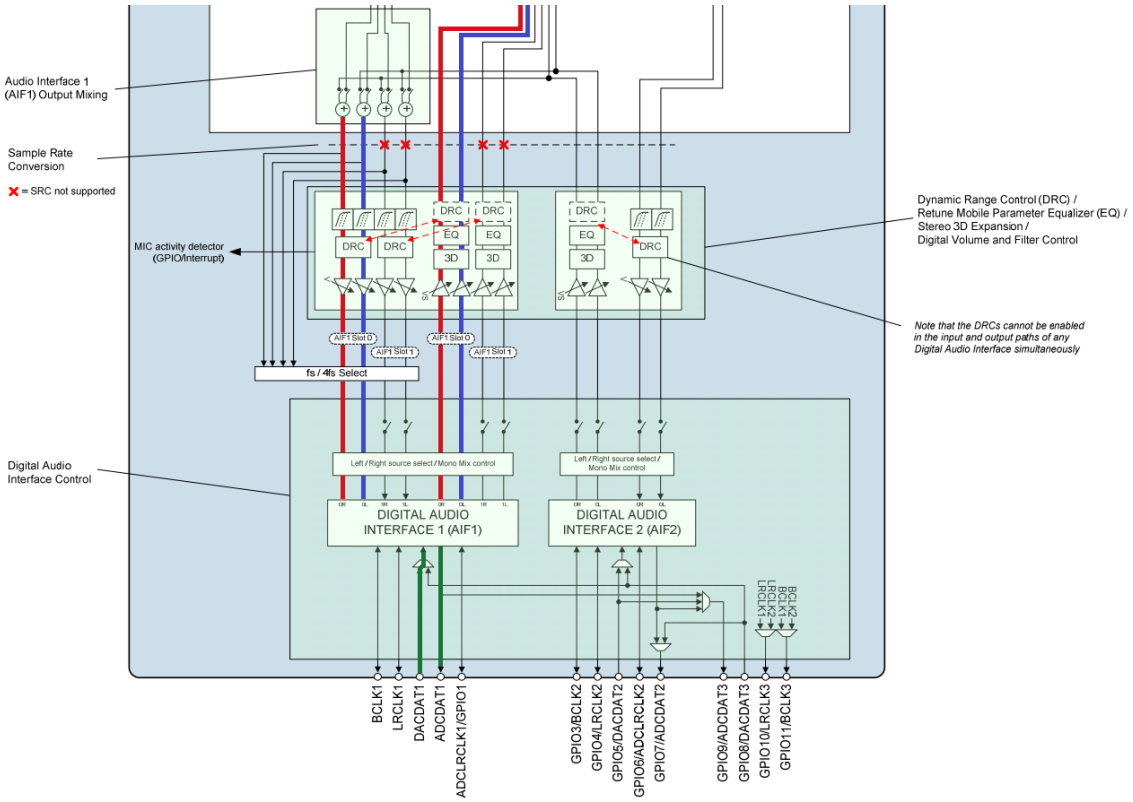


*Figura 5.1.5 - Caminos de mezcla digital del códec wm8994. Las configuraciones utilizadas de canal izquierdo y derecho se encuentran resaltadas en azul y rojo respectivamente. Los resaltes en verde indican la combinación de ambos canales por multiplexación temporal.*

A continuación de los DRC hay una etapa de control de volumen digital. En este caso, se utiliza el valor por defecto de 0 dB de ganancia ya que no se precisa un ajuste en este instante. De todas maneras, la ganancia se puede controlar por separado en ambos canales (desde -71,625 dB hasta +17,625 dB en pasos de 0,375 dB) con los controles AIF1ADC1L_VOL [7:0] y AIF1ADC1R_VOL [7:0]. Alternativamente, si se desea ajustar los volúmenes de los dos canales de manera conjunta, se puede escribir un 1 en el bit AIF1ADC1_VU.

La siguiente configuración disponible consiste en un modo de salida ultrasónica que permite cuadruplicar la frecuencia de muestro. Esto resulta útil en situaciones muy específicas como cuando se desea transportar señales con contenido de alta frecuencia de un micrófono ultrasónico. Por lo tanto, en este trabajo no se aprovecha de esta funcionalidad.

Otra etapa que se mantiene desactivada es la de mezcla mono. Ésta permite obtener la suma del canal izquierdo y derecho sobre ambos canales con una atenuación de 6 dB para evitar saturaciones. En la interfaz AIF1, ranura temporal 0, se activa mediante el bit AIF1DAC1_MONO. También, en esta misma sección se puede programar un filtro de deénfasis apropiado para entradas en formato CD con el control AIF1DAC1_DEEMP [1:0].

Finalmente, antes de enviar las muestras al procesador, se encuentra la interfaz AIF1. Como ya se comentó, aquí se combinan ambos canales para formar una trama en formato I2S. Está configurada para interactuar en la comunicación en modo esclavo mediante AIF1_MSTR. Si bien

ambas ranuras de tiempo se configuran previamente para interactuar con los ADC, además es preciso habilitarlas con AIF1ADC1L_ENA y AIF1ADC1R_ENA. Los canales también se pueden intercambiar con AIF1ADCL_SRC y AIF1ADCR_SRC, pero no es necesario en este caso.

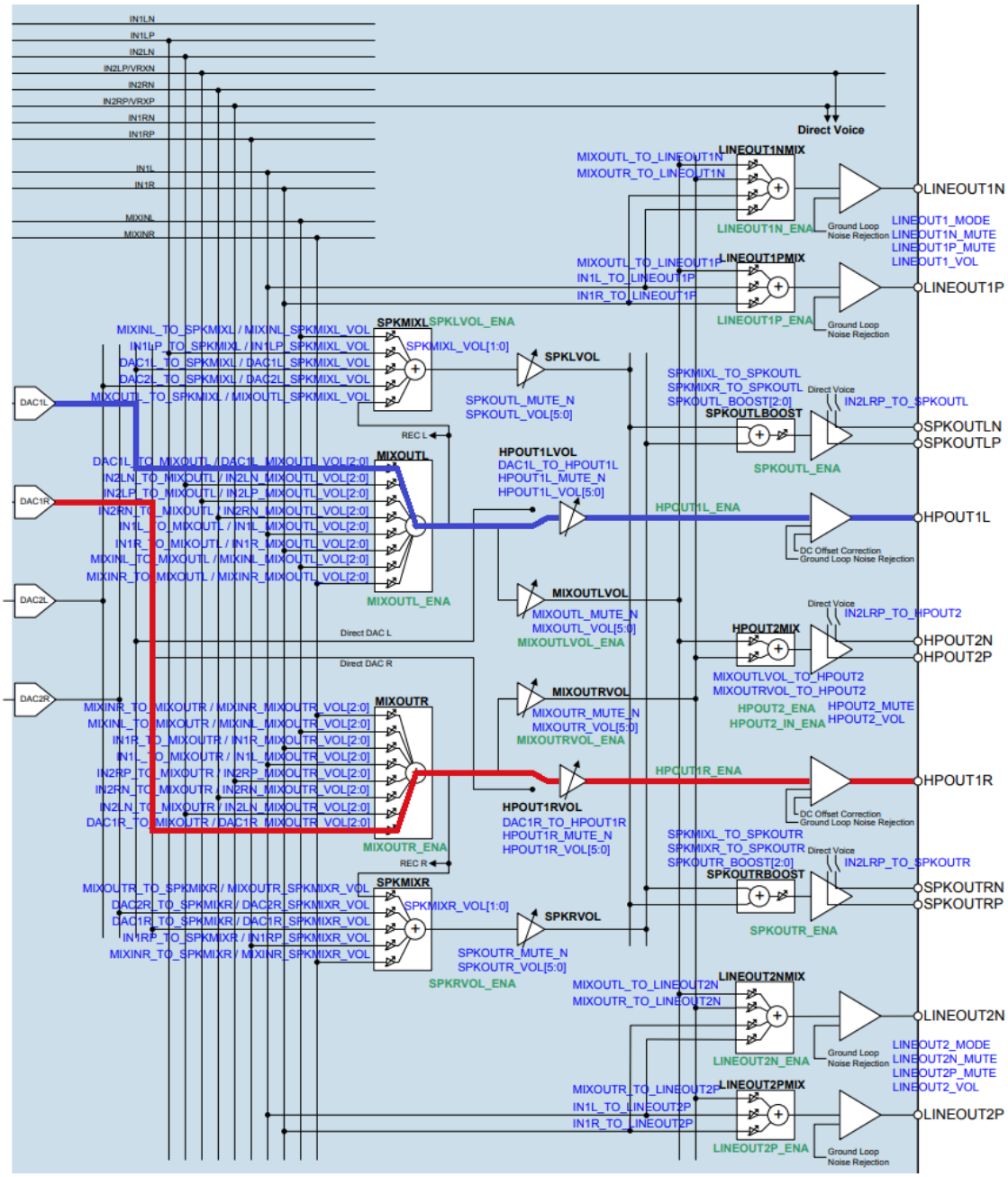


*Figura 5.1.6 - Caminos de salidas analógicas del códec wm8994. Las configuraciones utilizadas de canal izquierdo y derecho se encuentran resaltadas en azul y rojo respectivamente.*

Otras configuraciones importantes dentro de la interfaz AIF1 son las de compresión/expansión a través de AIF1DAC_COMP. Consiste en una cuantización no lineal que asigna más resolución a las amplitudes más pequeña a costas de una pérdida de resolución en amplitudes más grandes. Se puede realizar mediante el estándar europeo (A-law) o el estándar americano/japonés (u-law) con la ayuda del bit AIF1DAC_COMPMODE. Sin embargo, en este trabajo tampoco se pone en uso principalmente porque este modo opera con una codificación de punto flotante de 8 bit la cual empeoraría mucho la resolución deseada de 24 bit.

Una vez armada la trama, se envía por el pin ADCDAT1. En caso de que no se desee utilizar un procesador externo se puede activar una loopback entre ADCDAT1 y DACDAT1 (el pin encargado de recibir el audio) mediante el control AIF1_LOOPBACK.

El camino de salida digital es bastante simétrico al de entrada y comienza cuando la información llega al códec de audio mediante el pin DACDAT1. AIF1 desarma la trama y la convierte en los dos canales (izquierdo y derecho) para que las etapas posteriores los procesen. Luego del mezclador mono, se encuentra otro control de volumen digital idéntico al que se encontraba en el camino de entrada con la única excepción de que tiene un control de mute que se debe deshabilitar al final de la inicialización con AIF1DAC1_MUTE. Adicionalmente, este comando de unmute se puede realizar en una transición suave con el parámetro AIF1DAC1_UNMUTE_RAMP.

Además de los DRC, también existe un ecualizador parametrizable de cinco bandas y un simulador de sonido 3D en serie al camino de la señal. Se busca tener un sonido alterado en lo menos posible por el códec así que ninguna de estas funciones se activa.

Como se puede ver en la Figura 5.1.4, los canales se encaminan hacia los conversores DAC1L y DAC1R. AIF1DAC1L_TO_DAC1L y AIF1DAC1R_TO_DAC1R son los controles que se deben intervenir para lograr esta conexión. Antes de llegar a los DAC, se encuentra otro conjunto de controles digitales de volumen (de -71,625 dB hasta 0 dB con pasos de 0,375 dB) configurables mediante DAC1L_VOL [7:0] y DAC1R_VOL [7:0] (se puede escribir un 1 en DAC1_VU para actualizar ambos volúmenes en simultaneo). Éstos también tienen un mute que se debe deshabilitar con DAC1L_MUTE y DAC1R_MUTE al final de la secuencia de inicialización.

Luego de estos controles de volumen, se encuentran finalmente los dos DAC, los cuales se activan con los parámetros DAC1L_ENA y DAC1R_ENA. Su operación consiste en un sobre-muestreo por interpolación de 24 bit para lograr una señal analógica alta calidad.

En la Figura 5.1.6, se ve que las señales que salen de los DAC pueden ir a los mezcladores SPKMIX para salidas de parlantes o a los mezcladores MIXOUT que sirven, entre otras cosas, para salidas de auriculares y salidas de línea. En el kit de desarrollo, el jack de audio de 3.5mm color verde está conectado a la salida de auriculares así que resulta de interés realizar las configuraciones pertinentes para este periférico.

Primero, se deben configurar las salidas de los DAC a MIXOUTR y MIXOUTL con los controles DAC1L_TO_MIXOUTL y DAC1R_TO_MIXOUTR. Estos mezcladores permiten programar individualmente la ganancia de todas sus entradas, DAC1L_MIXOUTL_VOL [2:0] y DAC1R_MIXOUTR_VOL [2:0] permiten hacer este ajuste a las señales provenientes de los DAC1 (desde -21 dB hasta 0 dB en pasos de 3dB).

Seguido de esto, es necesario realizar una secuencia de activación de la etapa de salida de los auriculares. Primero, se deben activar estas etapas de salida mediante HPOUT1L_ENA y HPOUT1R_ENA. Luego, debe haber un tiempo muerto de 20us para habilitar HPOUT1L_DLY y HPOUT1R_DLY, los cuales ponen en marcha una intermedia de configuración.

Una vez en esta fase, se debe inicializar el multiplicador de tensión (charge pump) a través de CP_ENA. Este dispositivo toma como entrada 1,8V conectador al pin CPVDD del códec y devuelve dos tensiones que se pueden medir en los pines CPVOUTP y CPVOUTN. El objetivo de estos rieles de tensión es el de controlar la alimentación de los auriculares evitando capacitores de acople. Además, se puede configurar el multiplicador de manera que se ajuste dinámicamente a los cambios en la señal habilitando CP_DYN_PWR y seleccionando la interfaz AIF1, ranura 0, en

CP_DYN_SRC_SEL. Esto logra realizar una optimización de consumo de potencia en tiempo real en la etapa de salida. Se trata de una invención propietaria de Wolfson y conforma lo que ellos autodenominaron etapa de salida “clase W”.

Adicionalmente, se debe configurar un DC servo que trabaja en conjunto con el multiplicador de tensión. Su función se reduce a mantener activamente el valor de continua de la salida dentro de 1mV de tierra. Tener un valor de continua en salidas sin capacitor de acople genera corrientes estáticas (que representan un consumo indeseado) y sonidos de molestos “pop” en el encendido y apagado. La habilitación se realiza a través de DCS_ENA_CHAN_0 y DCS_ENA_CHAN_1.

Los controles DCS_TRIG_ENA_STARTUP_0 y DCS_TRIG_ENA_STARTUP_1 se deben escribir en 1 antes de que haya audio circulando por la etapa de salida para que el DC servo realice las mediciones necesarias en cada canal. Aproximadamente, se deben asegurar 86 ms de silencio hasta que se complete este proceso. Para proceder de manera segura con el audio, se pueden leer continuamente estos registros hasta que vuelvan a tener un valor de 0.

El último paso de inicialización de los auriculares es el de escribir un 1 en HPOUT1L_OUTP, HPOUT1L_RMV_SHORT, HPOUT1R_OUTP y HPOUT1R_RMV_SHORT. Después de esto sólo hacen falta encender y configurar los PGA de salida HPOUT1LVOL y HPOUT1RVOL. Los caminos de conexión se establecen con DAC1L_TO_HPOUT1L y DAC1R_TO_HPOUT1R y, las ganancias (desde -57 dB hasta +6 dB en pasos de 1dB), con HPOUT1L_VOL [5:0] y HPOUT1R_VOL [5:0]. Finalmente se deben deshabilitar los mutes de cada canal con HPOUT1L_MUTE_N y HPOUT1R_MUTE_N.

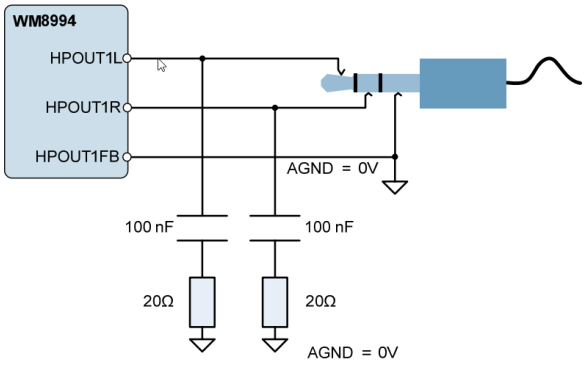


*Figura 5.1.7 - Red de conexión zobel empleada en el kit de desarrollo STM32F746G Discovery.*

Físicamente, las salidas de auriculares salen por los pines HPOUT1L y HPOUT1R. Se recomienda utilizar la red de conexión de la Figura 5.1.7 para maximizar la performance de audio en todas las aplicaciones. Este circuito atenúa posibles oscilaciones de alta frecuencia causadas por, entre otras, efecto inductivo en los auriculares o cables/caminos impresos largos (de alta capacidad).

## 5.2 - SAI

La transmisión de muestras de audio entre el códec de audio y el procesador se realiza mediante un protocolo de comunicación serial síncrono denominado I2S (IC to IC Sound). Opera en semidúplex y puede ocupar tanto el rol de maestro como el de esclavo. Su funcionamiento a grandes rasgos se puede resumir mediante las acciones de tres pines:

- CK: clock serial que se transmite en caso de ser maestro y se recibe en caso de ser esclavo.

- WS: selector de palabra, informa al receptor sobre cuál canal trata la información recibida en el pin SD.
- SD: datos seriales, contiene las muestras de audio de canal derecho e izquierdo multiplexadas en el tiempo.

También, en el caso de que el I2S sea configurado como maestro, se puede utilizar el pin MCK como clock de referencia para para dispositivos de audio externos. Tiene una frecuencia igual a 256 veces la frecuencia de muestreo del audio.

En este trabajo es preciso operar en dúplex para que el audio pueda ser recibido y transmitido por el procesador simultáneamente, debido a esto, hizo falta declarar dos instancias del protocolo semidúplex en ambos sentidos como se puede apreciar en la Figura 5.2.1. Entre las tres entidades que están involucradas en la comunicación, solamente una de ellas debe estar configurada como maestro. En este caso, ese rol lo ocupa la instancia de transmisión de I2S esto significa que las señales de CK y WS son impuestas por él.

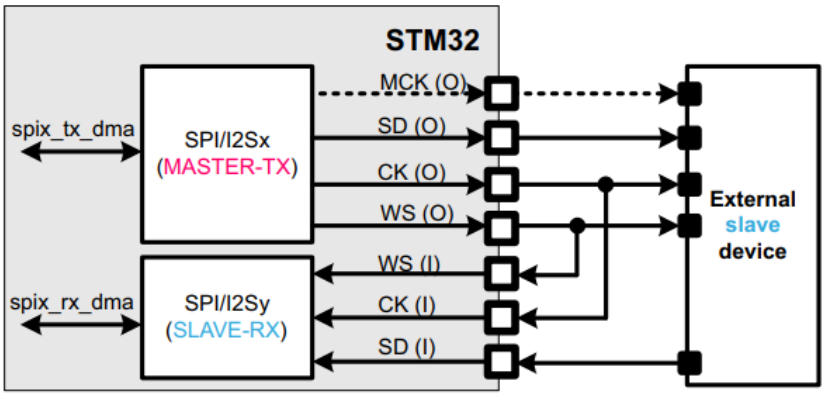


*Figura 5.2.1 - Configuración de comunicación dúplex entre el procesador y el dispositivo de audio externo con dos instancias de protocolo I2S.*

I2S soporta cuatro formatos de transmisión distintos de transmisión de datos. Siempre se debe enviar el primero el bit más significativo del canal izquierdo por el pin SD mientras que un flanco del clock indica transmisión y el otro indica recepción (puede adoptar cualquier polaridad definida por software). Las principales diferencias entre cada uno de estos es la señalización mediante el pin WS y la justificación de los datos cuando el tamaño de la trama es mayor al tamaño de los datos.

En este caso, se trabajó con el estándar I2S de Philips. Aquí, la señal WS en estado 0 indica canal izquierdo en SD y viceversa. Es importante destacar que el flanco de WS se produce siempre un ciclo de clock antes de cada MSB (bit más significativo). Este fenómeno se puede ver fácilmente en la Figura 5.2.2.

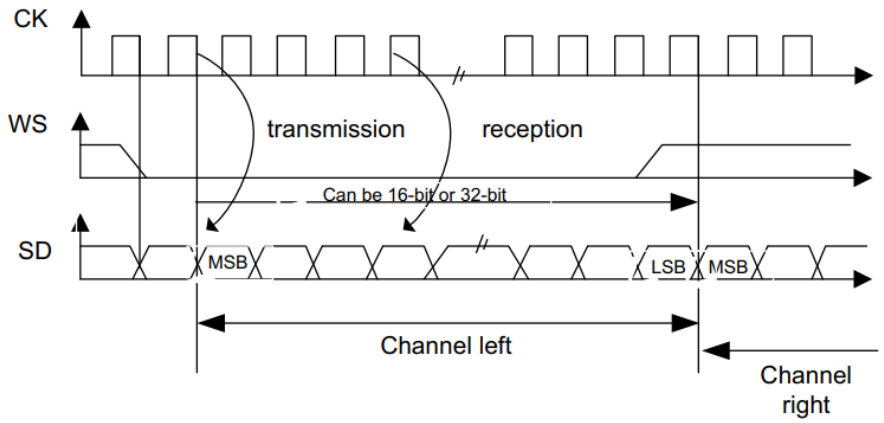


*Figura 5.2.2 - Estándar de Philips para el protocolo I2S.*

La trama se puede configurar para darle 16 bit o 32 bit a cada canal. El tamaño de los datos siempre debe ser menor o igual al tamaño del slot asignado para asegurar un funcionamiento correcto. Cuando los datos son más pequeños, el bit más significativo se transmite primero y los bits posteriores al bit menos significativo se fuerzan a un valor de 0.

Si bien el protocolo I2S puede funcionar de manera independiente, en este trabajo se lo utiliza encapsulado dentro de la interfaz SAI (Interfaz Serial de Audio) para poder tener mayor flexibilidad. Las tramas SAI (ver Figura 5.2.3) consisten en una cantidad de slot programables y cada uno de éstos se puede habilitar o deshabilitar. El tamaño de cada slot puede ser de 16 bit, 32 bit o se puede hacer igual al tamaño de los datos contenidos en el slot. El tamaño de trama se recomienda configurarlo a una potencia de 2 entre 8 y 256. Todos los ciclos de clock en los cuales no haya nada para transmitir se puede optar entre escribir ceros o poner la salida en modo de alta impedancia (del lado de la recepción simplemente se realiza un descarte).

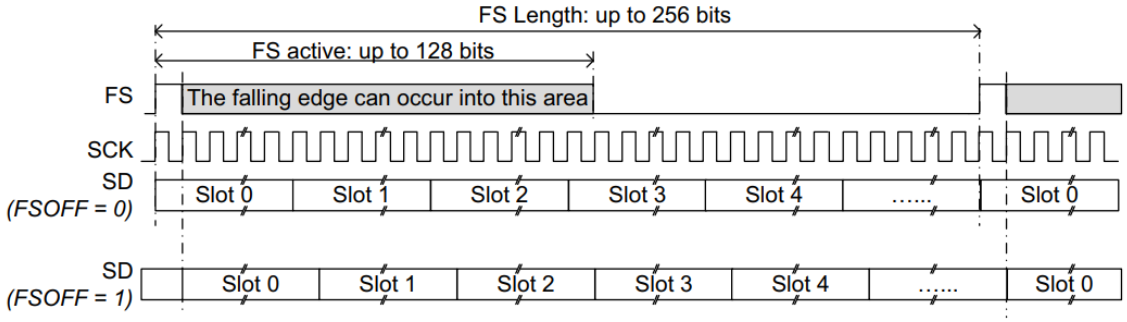


*Figura 5.2.3 - Ejemplo de trama SAI genérica.*

Otra opción que brinda flexibilidad a la SAI es la posibilidad de programar la acción del pin FS (sincronización de trama) para que actúe como WS en el protocolo que sea que se encapsula. Se puede elegir en qué estado comienza al principio de la trama y se debe determinar un período de actividad en el cual puede ocurrir el flanco ascendente/descendente. Si se define que el FS identifica canal izquierdo y derecho, la cantidad de slots debe ser par y el flanco sucede cuando ocurren la mitad de los slots. Finalmente, se puede establecer un offset en las transiciones de FS para replicar lo que ocurre con el WS en el estándar I2S de Philips.

La configuración de trama utilizada en este trabajo consiste en 4 slots de 24 bit cada uno. Esto se hace para tener compatibilidad con la trama armada por la interfaz AIF1 del códec de audio.

Debido a que no se utiliza la ranura 1 del AIF1, solamente se activan los slots 0 y 2. Con todos los slots hay un total de 96 bit, el tamaño de la trama es de 128 ya que debe ser potencia de 2 y mayor a este total. Tanto los slots apagados como los bits al final de la trama se fuerzan a cero.

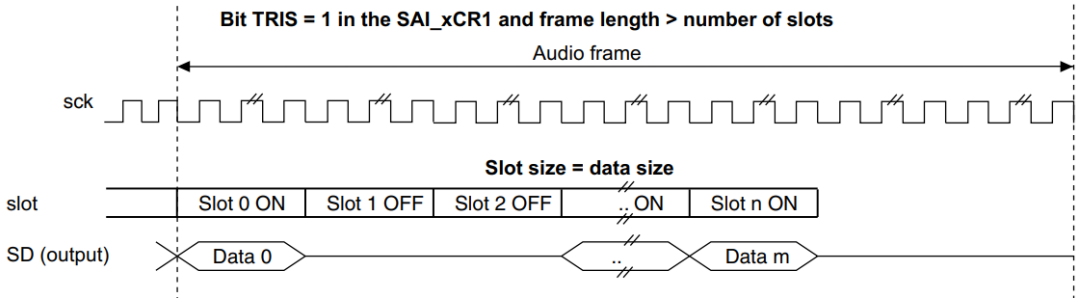


*Figura 5.2.4 - Ejemplo de trama SAI con tamaño total mayor a tamaño de slots.*

El pin FS se ajustó de tal manera que sea compatible con el estándar I2S de Philips. Se habilitó el offset y la identificación de canal, el valor de inicio es 0 (canal izquierdo) y el tiempo activo es de 64 bits (mitad de la trama) para que se pueda realizar el flanco positivo al comienzo del segundo slot.

## 5.3 - DMA

Una vez que se encuentra resuelta la transmisión de datos a nivel lógico, hace falta una entidad que sea capaz de regular las transacciones de tramas de audio. Hay tres posibilidades provistas por la interfaz SAI. La primera se basa en un sondeo (o polling), la segunda se basa en interrupciones y la tercera, utilizada en este trabajo, se basa en DMA.

DMA (Acceso Directo a la Memoria) es un controlador integrado en el procesador Cortex M7 destinado a administrar transferencias entre memoria y periféricos sin necesidad de acción de la CPU. De esta manera, el DSP queda liberado para realizar otras operaciones.

En total, se cuentan con dos controladores (DMA1 y DMA2), cada uno posee un total de ocho streams (flujos) y cada stream puede tener hasta ocho canales (solicitudes). Dentro de DMA2, se utilizan el stream 4, canal 3 y el stream 7, canal 0 para transmisión y recepción respectivamente ya que éstos se encuentran mapeados a la instancia de SAI correspondiente.

Las transacciones de DMA consisten en tres pasos. Primero se carga la información ubicada en el registro del periférico o en la memoria en un registro especial de DMA. Luego, se utiliza ese registro para guardar los datos en la memoria o en el registro del periférico según corresponda. Finalmente, se realiza un decremento en otro registro de DMA que contiene la cantidad de transacciones a realizar.

Los registros del periférico son fijos, lo que significa que la información siempre se debe cargar o guardar en la misma dirección a lo largo de la transmisión. Sin embargo, del lado opuesto, la memoria guarda los datos en un buffer de punteros consecutivos (que se puede administrar como un arreglo). Esto implica que cada vez que se realiza una transacción, se debe realizar un incremento en el puntero a memoria.

La transmisión es continua y el buffer es finito así que cada vez que ocurre un desborde en la posición del puntero, se debe resetear el mismo al comienzo de las direcciones. Esto se configura desde DMA habilitando la opción de administración de puntero bajo un buffer circular. Esto

también significa que una vez que se llama a las funciones de play y record de DMA, los bloques se audio se comenzarán a recibir y transmitir de manera continua según la frecuencia de muestreo. Como el fuljo de audio es la parte más importante en el producto, la prioridad de interrupción del DMA es la máxima.

Existe la opción de utilizar un buffer de tipo FIFO como intermediario de la transacción DMA en el caso de que el tamaño de bits de escritura sea distinto al de lectura. Sin embargo, no se aprovecha ya que, tanto desde el lado de la memoria como del lado de los periféricos, los datos se leen en base a palabras de 32 bits (los ocho bits más significativos se fuerzan a 0).

Esto significa que todas las posiciones del buffer en memoria son de 32 bits. Esto conlleva un problema ya que el códec interpreta que se manejará una variable entera de 24 bits y, por lo tanto, utiliza complemento a la base para representar número negativos. Si las muestras se leen en una variable de 32 bits con ocho ceros en las posiciones más significativos, se interpretará que todos los números son positivos (lo cual distorsiona los valores negativos). El método que se utiliza para interpretar correctamente esta discrepancia es el de correr la variable de 32 bits 8 bits hacia la izquierda (insertando ceros desde la derecha) se castea la variable a entero y se vuelven a correr 8 bits a la derecha. Esto asegura que se inserten bits en 0 si el valor era positivo y bits en 1 si el valor era negativo de manera de conservar el signo.

El tamaño del buffer de audio es un importante parámetro de diseño. Mientras más pequeño sea, mayor va a ser la frecuencia de interrupciones al microprocesador las cuales generarán excesivas secuencias de iniciación. Sin embargo, llega un punto en el que el tamaño del buffer es tan grande que se introduce un retardo entrada/salida perceptible al oído. Lo más conveniente siempre suele ser usar el buffer más grande que pueda asegurar un margen razonable (resulta de un ajuste empírico) ante este último umbral. De esta manera se pueden mitigar ambos problemas.

Este buffer se divide en dos bloques para poder aprovechar al máximo el tiempo de procesamiento. Gracias a DMA, se tiene la facilidad de poder realizar procesamiento sobre un bloque mientras que se realizan transacciones (recepción y transmisión) con el otro. Esto se aprovecha para alternar entre ambos bloques de manera continua, logrando un incremento en el tiempo disponible para realizar operaciones (ver Figura 5.3.1).

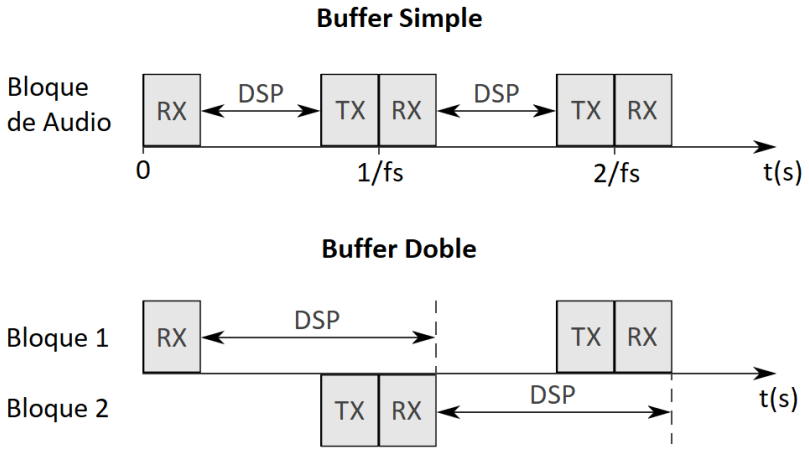


*Figura 5.3.1 – Comparación entre utilización de un buffer simple contra un buffer doble. Nótese el tiempo de procesamiento ganado en el segundo modo.*

Las funciones de callback de DMA se encargan de coordinar este último proceso. Existe una callback que interrumpe cuando se termina la mitad de la transferencia donde se realiza el procesamiento del primer bloque y una callback que interrumpe cuando se termina la transferencia completa donde se realiza el procesamiento del segundo bloque.

# 6 - Estructura del Código

## 6.1 - Entorno de Desarrollo Integrado

En etapas anteriores del proyecto, se utilizó el IDE de STM basado en Eclipse denominado SW4STM32 (System Workbench for STM32). La decisión se tomó debido a que, además de ser recomendado por STM, es gratuito y de código abierto. También, se aprovechó de la herramienta STM32CubeMX para mejorar el entendimiento sobre la configuración de periféricos del microcontrolador mediante su interfaz de usuario.

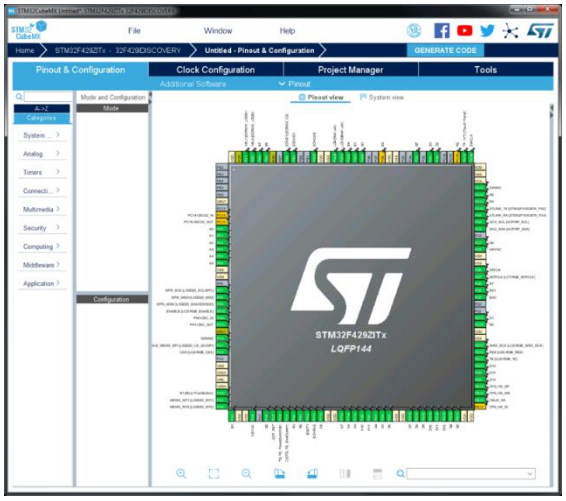


*Figura 6.1.1 - Herramienta de configuración y generación de proyectos STM32CubeMX.*

Finalmente, se adoptó el IDE uVision de Keil debido a que ofrece una mejor interfaz de depuración y de configuración del compilador. Además, presenta mejoras de estabilidad y navegación a lo largo del código en comparativa con el SW4STM32, el cual se encontraba en una instancia de desarrollo. Se resalta que la versión instalada de uVision corresponde a la arquitectura ARM, por lo tanto, ofrece amplio soporte a una gama de microcontroladores y/o kits de desarrollo.

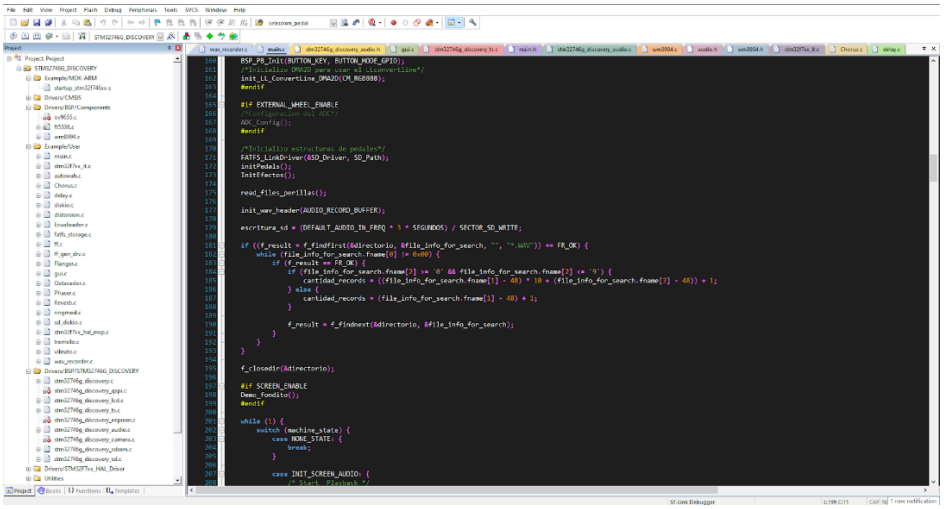

*Figura 6.1.2 – IDE uVision de Keil.*

Si bien este software ajusta sus configuraciones según la placa seleccionada para programar, se realizaron algunas pequeñas modificaciones. Se seleccionó el debugger ST-Link como

predeterminado para simplificar el proceso de depuración y se eligió el nivel 2 de optimización para el compilador. Por último, se configuró el IDE para tener acceso a las librerías CMSIS de matemática rápida en los casos en que fuera necesario.

## 6.2 - Diagrama de Flujo

El programa se basa en un esquema de máquina de estado. Esto significa que se tiene una variable la cual representa la tarea actual que se encuentra ejecutando el código. Un resumen de cada uno se ilustra en la Figura 6.2.1.

| | | |
|---|---|---|
| 0 | None State | No opera |
| 1 | Init Screen Audio | Ocurre una vez al inicializar |
| 2 | Buffer Offset Half | Procesa la primera mitad del buffer |
| 3 | Buffer Offset Full | Procesa la segunda mitad del buffer |
| 4 | Screen Refresh | Actualiza los componentes de la pantalla |

*Figura 6.2.1 – Resumen de estados del código.*

Se comienza en un estado de inicialización donde se configuran los periféricos y/o circuitería externa utilizados por el microcontrolador (detallado en la sección 6.3). Además de esto, se declaran las variables necesarias.

Inmediatamente se pasa al estado “none” o de no operación. Aquí, la CPU puede ser interrumpida por dos eventos: llegada de bloques de audio mediante DMA o refresco periódico de la pantalla LCD mediante el TIM3.

Al llegar una interrupción por parte de DMA, la máquina puede pasar al estado de “half buffer” o “full buffer” dependiendo de qué mitad del bloque se haya llenado. En ambos casos, primero se copia el contenido correspondiente a un buffer auxiliar de procesamiento. Seguido de esto, se recorre un arreglo en donde se dispone de los estados de los efectos (prendido o apagado) y se realizan las operaciones necesarias según aquellos que se encuentren encendidos. Por último, se vuelcan los resultados en el buffer correspondiente de salida (ver Figura 6.2.2) y se vuelve al estado “none”.

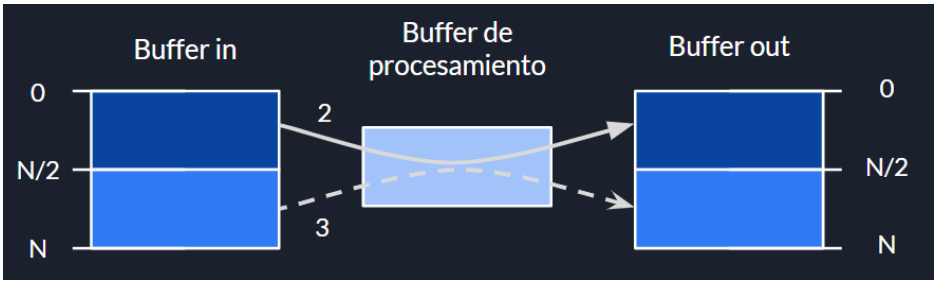


*Figura 6.2.2 – Interacción entre los buffer de entrada y salida cuando se realiza una interrupción por parte de DMA.*

Por último, en el caso de refresco de la pantalla, se pasa al estado “screen refresh”. El objetivo de éste es el de evaluar en qué contexto se encuentra la interfaz de usuario y ejecutar las acciones correspondientes para actualizar la información disponible en el LCD (ver Capítulo 7). En otras palabras, es responsable tanto de dibujar los elementos gráficos (menúes, pedales individuales, perillas, LEDs de encendido/apagado, etc) como de llevar a cabo la lógica que relaciona la pantalla con el resto del programa. Por ejemplo: si se presiona sobre el switch 3PDT, se debe encender el LED correspondiente y se debe cambiar el estado del efecto dentro del

programa para que el mismo sea procesado posteriormente. Cabe destacar que mientras se detecta interacción del usuario con la pantalla, se deshabilita el procesamiento de audio con el fin de mitigar ruidos no deseados durante las transiciones. Al igual que la interrupción de DMA, ésta también finaliza volviendo al estado “none”.

Este proceso de transición de estados se resume en la Figura 6.2.3 y la Figura 6.2.4.

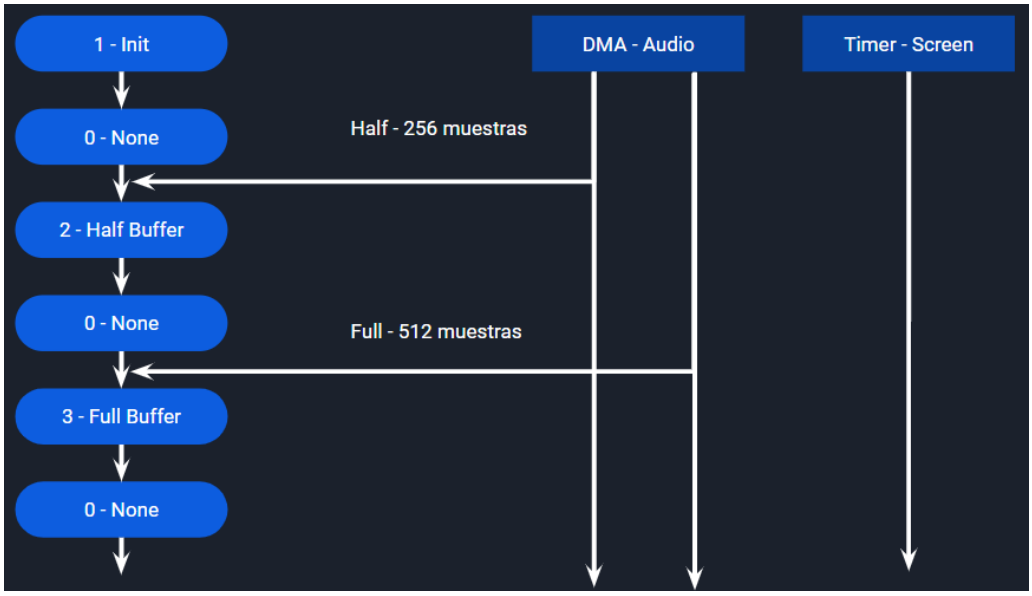


*Figura 6.2.3 – Diagrama de transición de estados ante una interrupción DMA.*

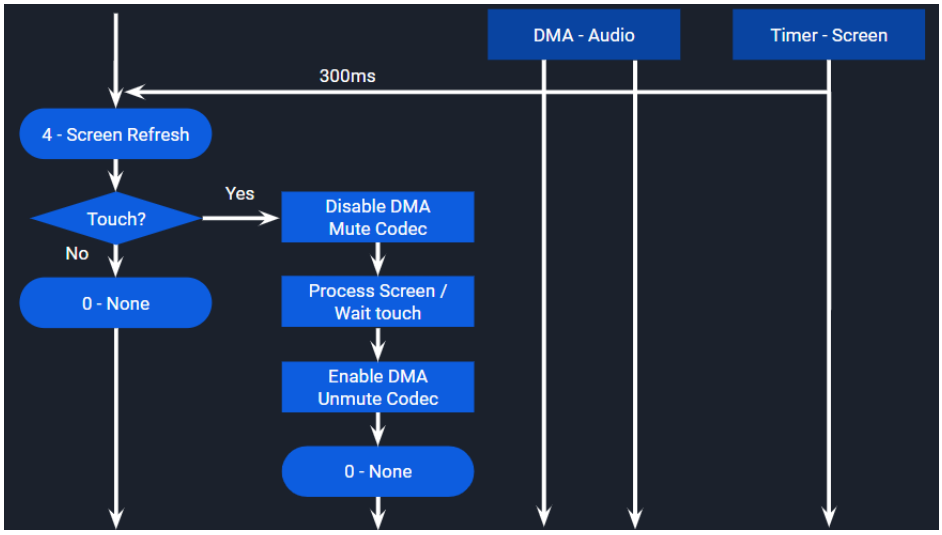


*Figura 6.2.4 – Diagrama de transición de estados ante una interrupción de TIM3.*

## 6.3 - Librerías Utilizadas

La implementación del código desarrollado requiere de una cantidad amplia de librerías para interactuar con las opciones de la placa STM32F746G. Gran parte de éstas son provistas en la página de STM y tienen como autor a el equipo de aplicación de la empresa MCD Electronik.

En general, en este proyecto las librerías actúan como controladores (o drivers) para los diferentes periféricos utilizados en el programa. Sirven como una capa de abstracción destinada a la programación de componentes y características mediante una lista de funciones definidas

dentro de ella. STM las denomina API (application programming interfaces) y las categoriza aproximadamente en tres tipos. Las primeras son las API de bajo nivel que son las más cercanas al hardware y están disponibles para una cantidad limitada de periféricos. Luego, se encuentran las API de alto nivel que son BSP (board support package) y HAL (hardware abstraction layer). La diferencia más importante entre estas dos es que la primera está desarrollada de manera relativa a la configuración de hardware de el kit de desarrollo utilizado mientras que la segunda es más genérica y se puede implementar de igual manera en cualquier kit stm32f7.

La totalidad de librerías utilizadas en este proyecto se puede categorizar según la prestación que brindan al mismo:

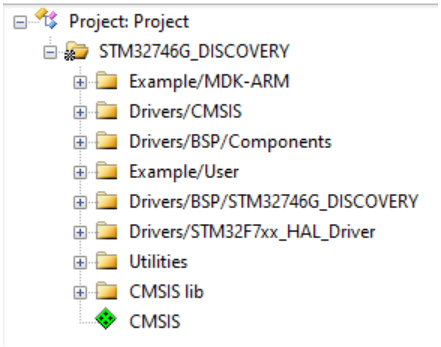


*Figura 6.3.1 – Carpetas de directorios utilizados en el proyecto.*

En términos de audio, tal como se explicó en el Capítulo 5, los periféricos utilizados son: el códec de audio wm8994 que se encarga de la conversión A/D y D/A, el SAI que emplea los protocolos de comunicación y control I2S e I2C respectivamente y el DMA que se ocupa de la transferencia entre memoria y periféricos para las muestras de audio.

Para el códec, se dispone de una librería en el directorio de componentes responsable de escribir los registros pertinentes para la configuración deseada del mismo (ver sección 5.1). Las librerías de SAI, I2S, I2C y DMA se encuentran dentro del directorio de drivers de HAL, cada una de ellas en archivos separados. Finalmente, todas son agrupadas mediante una librería BSP (llamada desde el main) que se encarga de configurar todos los componentes previamente mencionados para poner en funcionamiento a la interfaz de audio.

Para la interfaz de usuario, en primer lugar, se acude a la librería BSP TS (touchscreen) para todo lo relacionado a la detección de touch y la determinación de la coordenadas de contacto. Por otra parte, se accede al BSP LCD que, a su vez, remite a la librería LTDC de HAL para instrucciones básicas del display como la selección de capa y el dibujo de fondos. El timer 3 utilizado para las interrupciones de refresco de pantalla tiene su librería correspondiente en la HAL TIM.

Por último, se desarrollo una librería específica de este proyecto denominada gui.c en la carpeta de usuario (ver Figura 6.3.2) destinada a manejar las interacciones de la interfaz gráfica. Es responsable de la lógica detrás de los botones, las perillas y los pedales (ver Capítulo 7). También hace uso del periférico DMA2D cuya librería se encuentra en la carpeta HAL.

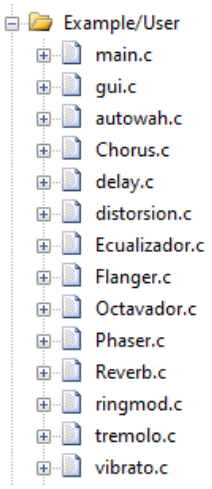


*Figura 6.3.2 – Archivos de desarrollo propio en la carpeta de usuario.*

Además de las librerías destinadas al audio y a la pantalla, se implementan otros controladores para la extensión de memoria disponible. Uno de ellos es el SDRAM, se trata de un chip dedicado en la placa con 128 Mbit de memoria adicional. Tiene su propia librería BSP y HAL para establecer su funcionamiento y según se programó en main.h, almacena buffers para la transferencia de imágenes a la pantalla, los buffers de audio y las perillas/sliders presentes en la interfaz gráfica. El otro controlador amplía la memoria flash mediante una tarjeta SDHC (que también tiene librerías BSP y HAL). Aquí se guardan los fondos de los menúes/pedales y las perillas que más tarde se vuelcan sobre la memoria SDRAM. En la memoria flash integrada se tienen los distintos tipos de led apagados y prendidos para los menúes/pedales y algunas perillas especiales selectoras de onda.

Más allá de los archivos desarrollados en este proyecto, cabe destacar algunas modificaciones importantes realizadas sobre las librerías ya existentes. Entre ellas, se adaptó el código de la BSP audio para poder manejar muestras de 24 bits a 48 KHz según lo detallado en el Capítulo 5. Además, se realizó un ajuste sobre BSP TS para poder deshabilitar el audio cuando se detecta contacto con la pantalla y rehabilitarlo en el caso contrario. Finalmente, se modificó el archivo de configuración de arranque aumentando el heap size a 0x4000 a fin de reservar memoria suficiente para las variables declaradas posteriormente.

# 7 - Interfaz Gráfica de Usuario

## 7.1 - Funcionamiento a Alto Nivel

En este proyecto, se aprovechó la pantalla FTF-LCD facilitada por el kit de desarrollo para crear una interfaz gráfica intuitiva de tal forma de brindarle a un usuario sin conocimiento del código la posibilidad de utilizar el proyecto dentro de ciertos límites.


*Figura 7.1.1 – Interfaz de usuario diseñada para la pantalla del kit de desarrollo. Menú 1 a la izquierda, menú 2 a la derecha.*

Como se puede apreciar en la Figura 7.1.1, esta interfaz consiste en dos pantallas de menú con seis pedales cada una (ofreciendo un total de doce efectos distintos). El programa comienza en la primera pantalla (menú 1) y se pueden presionar las flechas laterales para cambiar del menú 1 al menú 2 y viceversa. Cada uno de los pedales tiene dos tipos de interacciones; si se presiona la parte inferior de su modelo (donde se encuentra el interruptor push 3PDT) se puede prender o apagar el pedal, si se presiona la parte superior se accede a una nueva pantalla propia del pedal. Estas pantallas se refieren internamente como pantallas de pedales individuales, hay una por cada pedal y cada una tiene opciones similares.

Primero, se puede prender o apagar el pedal apretando sobre el botón push 3PDT, esto acciona el prendido o apagado (respectivamente) de un led de referencia en la imagen. Luego se encuentran los controles de los efectos que pueden ser tanto perillas como deslizadores dependiendo del caso. También se puede navegar desde este menú a las pantallas individuales de los pedales adyacentes mediante las flechas laterales. Finalmente, el botón de home vuelve al menú correspondiente (si el pedal es de los primeros seis, retorna al menú 1, si se trata de la segunda tanda de seis, regresa al menú 2).

Los parámetros de los efectos se encuentran inicializados de tal manera de tener un preset que represente de forma adecuada de cada efecto. El usuario puede variar los parámetros dentro de los rangos configurados a ajustar, pero solamente se mantienen siempre y cuando la placa se encuentre prendida. Apagar y prender la placa o apretar el botón de reset ubicado en la parte inferior resulta en un retorno a los ajustes por defecto.

En la sección 13.1, se explicará en detalle los parámetros disponibles para cada efecto.

## 7.2 - Funcionamiento a Bajo Nivel

En el código, se dispone de una serie de estructuras de datos diseñadas para facilitar la lógica de la interfaz gráfica (ver Figura 7.2.1). “PedalElement” representa todas las cosas que conforman a un pedal dentro del código, hay un pedal por cada efecto. Dentro de ella se encuentra primero “PerillaElement”, la cual se encarga de cargar la cantidad de perillas o

deslizadores que hacen falta dentro del pedal. Cada una de estas perillas o deslizadores está representada unívocamente por un "GUIElement" (habrá tantos "GUIElement" definidos en cada "PerillaElement" como ajustes tenga el efecto correspondiente). Otra estructura que se desprende de "PedalElement" es "PushElement", esta última se encarga de todo lo relacionado con el prendido y apagado de los pedales y leds. Se encuentra conformada por un estado (resultado de cuatro banderas multiplexadas en la misma variable) y dos "LinkElement" correspondientes a el botón 3PDT de menú y al botón 3PDT del pedal individual. Una estructura "LinkElement" se desprende directamente de "PedalElement" para representar el botón que, desde el menú, entra a la pantalla del pedal individual. "MenuElement" contiene dos "LinkElement" que se encargan de regular las flechas de navegación laterales en las pantallas de pedales individuales. Finalmente, el último integrante de la estructura "PedalElement" es una referencia a la función del efecto en cuestión dentro de su correspondiente archivo ".c".

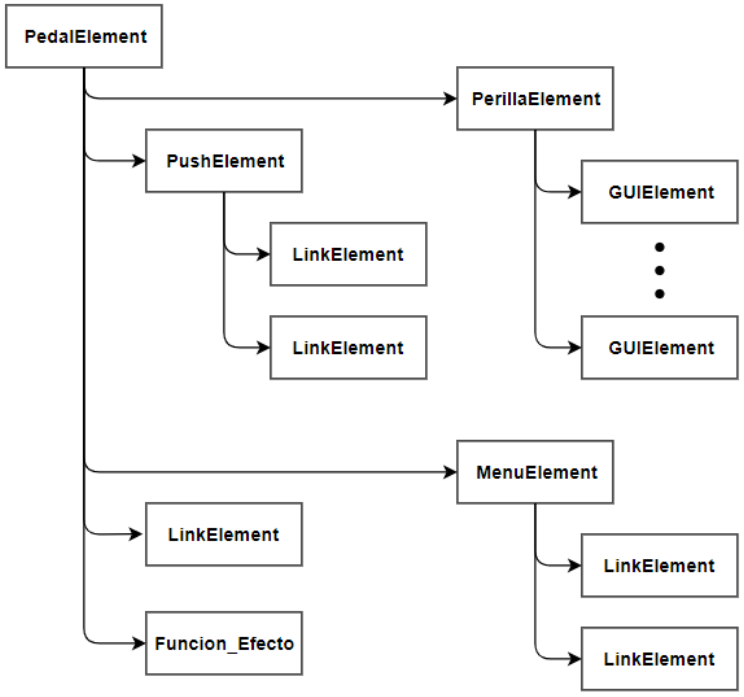


*Figura 7.2.1 – Diagrama de la estructura PedalElement.*

Como se puede observar, todo desemboca en estructuras "GUIElement" o "LinkElement". Estas dos son las que se encargan de la mayoría del trabajo a través de sus funciones de control. Las estructuras "LinkElement" definen un botón físico en la pantalla; contienen información de área (en coordenadas de píxeles x e y) y las instrucciones necesarias para administrar ese botón en la forma de una función "handle" y una función "callback". "GUIElement" ajusta varios parámetros como la inicialización, la sensibilidad, la posición y la tira de imágenes correspondientes al deslizador o perilla. Además de una función "handle" y una función "callback", también dispone de una función "render".

Las funciones "handle" se llaman periódicamente dentro del ciclo principal del programa cuando se detecta un toque de pantalla y se encargan de verificar si la posición del toque afecta al elemento en cuestión (hacen una operación condicional con las coordenadas x e y). Adicionalmente, las "handle" evalúan si es necesario llamar a la función "callback" de la misma estructura (a veces se debe esperar a que el botón se suelte y, en otros casos, se debe llamar inmediatamente). Cuando se llaman dentro de una estructura "GUIElement" también calculan

el cambio de la posición de la perilla o deslizador en base al movimiento del toque del usuario y de la sensibilidad. Es importante destacar que solamente se debe llamar a las “handle” que entran en juego dentro de la pantalla proyectada en el momento. De otra manera, se podrían accionar botones que no se encuentran dentro de la pantalla.

Las “callback” generalmente describen qué es lo que se debe realizar una vez que se acciona el botón. En el caso, de los botones push 3PDT, se debe cambiar el estado del pedal y se debe dibujar el led correspondiente (ya sea en el menú o en la pantalla de pedal individual). Cuando se trata de un vínculo a otra pantalla, se debe cargar la imagen de la nueva pantalla y se deben a volver a dibujar todos los elementos que cambiaron de estado (la posición de las perillas y el estado de los leds). Por último, las “callback” de las perillas o deslizadores remiten a una función dentro del “.c” del efecto donde se encuentra la actualización a la variable controlada por el elemento.

Cuando se trata de un “GUIElement”, los “render” también se llaman periódicamente (después de los “handle”) y se encargan de graficar las perillas o deslizadores en la pantalla. Dibujar perillas es bastante distinto a dibujar una pantalla (que se maneja simplemente en la callback) ya que se encuentran almacenadas de otra manera. Las pantallas son pesadas (480x272 píxeles) pero no es un gran problema si tardan del orden de un segundo en cargar, debido a esto, se encuentran almacenadas en una tarjeta SD insertada en el kit con un código numérico para invocarlas con facilidad. Por otro lado, las perillas son mucho más livianas (52x52 píxeles) así que se pueden guardar en la memoria SDRAM para que la operación de búsqueda sea más eficiente y la imagen se actualice mucho más rápido. Sin embargo, se debe guardar una imagen por cada posición de la perilla, esto presenta un compromiso ya que mientras más imágenes, más grado de detalle al girar la perilla, pero mayor cantidad de memoria SDRAM ocupada y viceversa.

*Figura 7.2.2 – Ejemplo de spritesheet correspondiente a una perilla con seis posiciones posibles.*

Estas imágenes se encuentran guardadas en formato “bitmap” (cada píxel contiene información de rojo, azul, verde, transparencia y no hay ningún tipo de compresión) ordenadas en una disposición vertical. Internamente, se hace referencia a estos documentos como “spritesheets”. La “handle” le proporciona a la “render” la información necesaria para calcular cual de todas las perillas se deben dibujar en cada momento. Esta última debe hacer un llamado al protocolo DMA2D cada vez que se desea hacer una transferencia desde la memoria a la pantalla LCD. Si bien las perillas son rectangulares en el “spritesheet”, éstas se encuentran rodeadas de transparencia y se dibujan sobre la capa superior del LCD, lo que significa que el fondo de la imagen no se pisa con un rectángulo, sino que se muestra como si la perilla estuviese apoyada encima.

Afuera de la estructura “PedalElement” solamente existen dos “LinkElement” destinados a las flechas laterales de transición entre el menú 1 y el menú 2.

En ejecución, se genera un arreglo global de doce posiciones de estructuras “PedalElement” (una para cada pedal) junto con los dos “LinkElement” de las flechas del menú y se cargan por completo en una función de inicialización.

## 7.3 - Diseño Gráfico

Las imágenes proyectadas sobre la pantalla fueron generadas por computadora mediante programas de edición. El fondo negro es consistente alrededor de todas las imágenes cargadas en la memoria SD. Los pedales son parte de diseños comerciales, pero están editados para servir mejor a las funciones que les corresponden en el programa. En general, todos tuvieron un cambio de estilo de perillas para poder evitar una saturación innecesaria de la memoria SDRAM con modelos distintos. También se realizaron retoques a los nombres de los pedales y de los ajustes para facilitarle al usuario la función de cada elemento. En algunos casos, también se agregaron o quitaron perillas acordes a la cantidad que hacía falta para realizar configuraciones. Ya que las imágenes de pedales solamente incluían el led en un estado (prendido o apagado), se hicieron ajustes en los colores para obtener una imagen de led prendido y de led apagado para cada pedal.

*Figura 7.3.1 – Ejemplo led prendido/apagado.*

Como se verá más adelante en la sección 13.1, los pedales se apoyaron sobre el fondo negro agregando una sombra para dar una ilusión de espacio. Más tarde, se agregaron las flechas laterales y el botón de home donde corresponden. Finalmente, se guardaron la imagen en formato “bitmap”, resolución 480x272 píxeles, se nombraron de acuerdo el código numérico y se cargaron en la memoria SD.

Las perillas y deslizadores se diseñaron basados en una posición de referencia. Las perillas se crearon en un entorno de diseño 3d y se generó el “spritesheet” mediante un script que toma fotos a medida que se rota la pieza mientras las dispone de manera vertical. Los deslizadores se realizaron manualmente en un programa de edición cambiando de lugar la pieza central de extremo a extremo. Una vez generado el “spritesheet” en formato “bitmap”, se introdujo el archivo en un conversor de imágenes para obtener un arreglo de “uint8_t” listo para ser cargado en la memoria SD y más tarde ser volcado la SDRAM.

# 8 - Principios Teóricos sobre Filtros Digitales

## 8.1 - FIR vs IIR

En el mundo digital, los filtros tienen un potencial mucho mayor que en el mundo analógico. Esto se debe a que un filtro analógico, a medida que aumenta su complejidad, requiere más componentes, más tamaño físico y más trabajo para ensamblarse. En un filtro digital, el aumento de la complejidad implica solamente más memoria y más velocidad de procesamiento. Dos cosas que son cada vez menos costosas en los tiempos actuales. De todas maneras, ambos tienen sus limitaciones, el analógico está limitado por el ruido y el digital por la precisión de la conversión y del redondeo que le corresponda al tipo de variable utilizada al momento de realizar operaciones.

Existen dos categorías de filtros que se diferencian usualmente: los filtros FIR y los filtros IIR. Ambos llamados así por el tipo de respuesta que presentan a un impulso a la entrada (FIR: respuesta impulsiva finita, IIR: respuesta impulsiva infinita).

Un filtro FIR es bastante simple de diseñar a partir de la respuesta impulsiva del sistema ya que su comportamiento se puede reducir en un arreglo de coeficientes. Para calcular la salida en base a cualquier entrada basta con hacer el producto de la entrada con los coeficientes miembro a miembro:

$$y[n] = \sum_{i=0}^{N-1} b_i \cdot x[n-i] = b_0 \cdot x[n] + b_1 \cdot x[n-1] + \cdots + b_{N-1} \cdot x[n-(N-1)]$$

Esta cuenta produce una suma ponderada de la entrada actual con entradas anteriores. Nótese que el subíndice i no puede tomar valores negativos ya que, de otro modo, el filtro no sería causal. La causalidad es una característica intrínseca de cualquier filtro en tiempo real debido a que no se pueden hacer cuentas utilizando entradas posteriores por el simple hecho de que son desconocidas al momento de realizar el cálculo.

La transformada z es una herramienta poderosa de análisis de los filtros en tiempo discreto. Si se recuerda la propiedad de desplazamiento en el tiempo:

$$Z\{x[n-k]\} = z^{-k} \cdot X(z)$$

Es sencillo escribir la expresión general de un filtro FIR en el dominio z:

$$H(z) = \sum_{i=0}^{N-1} b_i \cdot z^{-i}$$

De la misma manera que en la transformación de Laplace, la respuesta en frecuencia se evalúa recorriendo el eje imaginario positivo, en la transformación z, la respuesta en frecuencia se evalúa recorriendo la circunferencia unitaria centrada en el origen. Esto se debe a la relación que hay entre la variable de Laplace s y z:

$$z = e^{sT} \qquad \text{Ec. 8.1.1}$$

Donde T es la inversa de la frecuencia de muestreo. Aprovechando esta información, finalmente se puede denotar la respuesta en frecuencia de un filtro FIR de la siguiente manera:

$$H(e^{j\omega T}) = \sum_{i=0}^{N-1} b_i \cdot e^{-ji\omega T}$$

También es posible aproximar la implementación de un filtro IIR como FIR mediante un truncamiento. Este consiste en igualar a cero todos los coeficientes después del coeficiente N-1. Si una gran proporción de la energía de la respuesta impulsiva se encuentra dentro de los N coeficientes restantes, la aproximación será aceptable. Esta herramienta es de gran interés en contextos similares a los de este trabajo ya que ayuda a expresar filtros básicos de primer y segundo orden (ambos de los cuales tienen una respuesta impulsiva infinita) de manera finita.

Los filtros FIR tiene extensas aplicaciones en campos como las comunicaciones digitales. Sin embargo, para los fines de este trabajo, los filtros FIR tienen una gran desventaja con respecto a los IIR: requieren N multiplicaciones y N-1 sumas en cada ciclo. Considerando que los filtros más comunes requieren del orden de las decenas de coeficientes para representarlos correctamente de manera aproximada, esta alternativa no se hace tan atractiva como la que se explicará a continuación.

## 8.2 - Diseño de filtros IIR

Los filtros IIR son un poco más elaborados en términos de su diseño, pero requieren mucho menos cómputo que un filtro FIR truncado en un caso general. La gran diferencia radica en que incorporan a su cálculo las salidas previas para calcular la salida actual. Debido a esto, se les ha dado el nombre de filtros recursivos.

La idea esencial de un filtro IIR se puede explicar rápidamente con el ejemplo de un pasa bajos de primer orden:

$$y[n] = d \cdot y[n-1] + (1-d) \cdot x[n]\,, 0 \leq d \leq 1$$

Se dice que es de primer orden en este caso ya que se considera una sola instancia previa de la salida en la su expresión, si se tuviese dependencia de instancias anteriores, sea tanto de la entrada o de la salida, el orden corresponde a la más previa de estas instancias.

También aparece un parámetro d, el cual tiene que estar relacionado de alguna manera a la respuesta en frecuencia del filtro. Si vamos a los casos extremos, cuando d se fija en 0 estamos en la presencia de un sistema identidad. Por otro lado, cuando d vale 1, la entrada se ignora por completo y la salida es igual a la salida anterior en un bucle. El primer caso se corresponde con un pasa bajos con ancho de banda infinito y, el segundo, con uno de ancho de banda nulo. Cualquier valor de d intermedio da el resto de las posibilidades.

Sería entonces de interés relacionar el parámetro d con el ancho de banda del filtro de alguna manera. Se puede transformar ambos lados de la ecuación para hallar la función de transferencia en variable z:

$$H(z) = \frac{1-d}{1-dz^{-1}}$$

A continuación, se podría evaluar la expresión en el dominio de la frecuencia recorriendo la circunferencia unitaria:

$$H\left(e^{j\omega t}\right) = \frac{1-d}{1-de^{-j\omega t}}$$

Finalmente se podrían obtener conclusiones de la relación de d con el ancho de banda mediante el módulo de la transferencia. Sin embargo, este proceso es engorroso y sería mucho más fácil de llegar a un método que tan solo involucre la inspección de la función de transferencia. Aquí es donde el mundo digital recurre al mundo analógico; se puede aprovechar la gran cantidad de teoría desarrollada a lo largo de los años para funciones de transferencia de variable continua.

El procedimiento es entonces el inverso, se comienza con la función de transferencia en variable s de Laplace. Desde aquí se busca una manera de traducirlo a variable discreta con ayuda de alguna transformación auxiliar. Finalmente, se calculan los coeficientes correspondientes a las entradas y salidas previas para implementar el filtro de manera digital.

Nuevamente resulta más conveniente dar un ejemplo de este proceso para explicarlo en detalle. Un filtro pasa bajos de segundo orden con ganancia unitaria en continua tiene la siguiente expresión:

$$H(s) = \frac{{\omega_c}^2}{s^2 + \frac{\omega_c}{Q} \cdot s + {\omega_c}^2} \qquad \text{Ec. 8.2.1}$$

Donde ωc es la frecuencia de corte o de -3dB en radianes por segundo (en este caso) y Q es el factor de calidad calculado como la relación entre la frecuencia de corte y el ancho de banda ωb:

$$\omega_c = 2\pi f_c \, , Q = \frac{\omega_c}{\omega_b} = \frac{f_c}{f_b} \qquad \text{Ec. 8.2.2}$$

Ahora se busca hallar la correspondiente transferencia en variable discreta H(z), con lo cual, lo más correcto sería recurrir a la relación entre las variables s y z de la Ec. 8.1.1. Sin embargo, el resultado no sería una razón de polinomios, con lo cual, no se podría hacer una traducción sencilla al momento de calcular los coeficientes.

Para solucionar este problema, se inventó la transformada bilineal. Este método aproxima la relación entre s y z a una relación de polinomios de primer orden (de allí el nombre bilineal). Se puede deducir su expresión fácilmente a partir de la Ec. 8.1.1.

$$z = e^{sT} = \frac{e^{\frac{sT}{2}}}{e^{-\frac{sT}{2}}} \approx \frac{1 + \frac{sT}{2}}{1 - \frac{sT}{2}} \Longrightarrow s \approx \frac{2}{T}\frac{z-1}{z+1} = \frac{2}{T}\frac{1 - z^{-1}}{1 + z^{-1}}$$

La función exponencial se divide en otras dos exponenciales y, luego, se utilizan los primeros dos términos de la aproximación por serie de Taylor de cada una para tener una ecuación lineal en ambos lados de la división. Esta aproximación mejora conforme disminuye el módulo del exponente, gracias a esto, mientras mayor sea la frecuencia de muestreo, menor será el error.

Esta transformación garantiza que el plano izquierdo se mapea a dentro de circunferencia unitaria, que el eje jω se mapea sobre la circunferencia unitaria y que el plano derecho se mapea en toda la zona exterior a esta. A pesar de esto, hay un precio a pagar por utilizarla, se produce una distorsión en el mapeo de las frecuencias. Para entender este fenómeno, puede evaluarse la transformación sobre la circunferencia unitaria:

$$s_a = j\omega_a = \frac{2}{T}\frac{1 - e^{-j\omega_d T}}{1 + e^{-j\omega_d T}} = \frac{2}{T}\frac{e^{j\frac{\omega_d T}{2}} - e^{-j\frac{\omega_d T}{2}}}{e^{j\frac{\omega_d T}{2}} + e^{-j\frac{\omega_d T}{2}}}$$

Donde $\omega_a$ es la frecuencia analógica y $\omega_d$ es la frecuencia digital.

Si se recuerda le expresión exponencial compleja de la tangente trigonométrica, se puede simplificar esta relación:

$$\tan(x) = \frac{e^{jx} - e^{-jx}}{j \cdot (e^{jx} + e^{-jx})}$$

$$\omega_a = \frac{2}{T}\tan\left(\frac{\omega_d T}{2}\right) \qquad \text{Ec. 8.2.3}$$

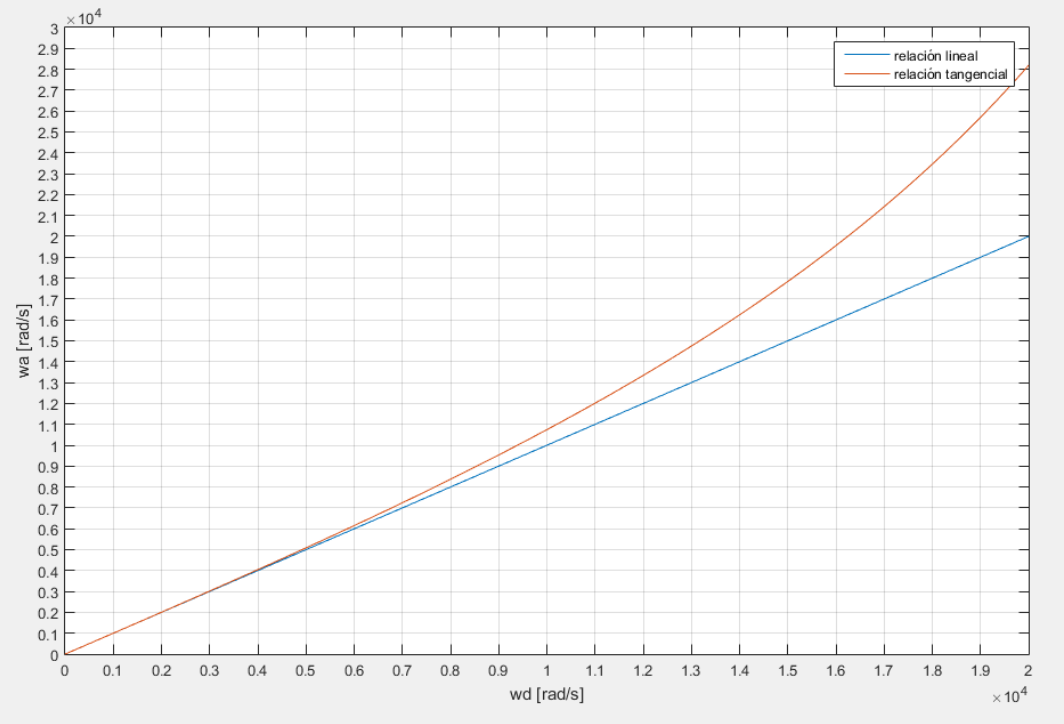


*Figura 8.2.1 – Comparación entre una relación lineal y la relación de la Ec. 8.2.3 suponiendo una frecuencia de muestreo de 44,1 KHz.*

A pesar de que no hay una manera obvia de mitigar completamente este efecto conservando la simplicidad de la transformada bilineal, se puede reemplazar esta expresión en la frecuencia de corte analógica. De esta forma, se garantiza que las frecuencias alrededor de la frecuencia de corte son las que menos distorsión sufren debido al mapeo.

Entonces, si se realizan ambos reemplazos en la Ec. 8.2.1, se obtiene:

$$H(s) = \frac{\left[\left(\frac{2}{T}\right)\tan\left(\frac{\omega_d T}{2}\right)\right]^2}{\left[\left(\frac{2}{T}\right)\cdot\frac{1 - z^{-1}}{1 + z^{-1}}\right]^2 + \frac{1}{Q}\left(\frac{2}{T}\right)\tan\left(\frac{\omega_d T}{2}\right)\cdot\left(\frac{2}{T}\right)\cdot\frac{1 - z^{-1}}{1 + z^{-1}} + \left[\left(\frac{2}{T}\right)\tan\left(\frac{\omega_d T}{2}\right)\right]^2}$$

Suponiendo:

$$K = \tan\left(\frac{\omega_d T}{2}\right) = \tan\left(\frac{\pi f_c}{f_s}\right) \qquad \text{Ec. 8.2.4}$$

Se llega al siguiente resultado:

$$H(z) = \frac{K^2 + (2K^2)z^{-1} + K^2 z^{-2}}{\left(1 + \frac{K}{Q} + K^2\right) + (2K^2 - 2)z^{-1} + \left(1 - \frac{K}{Q} + K^2\right)z^{-2}}$$

Finalmente, hay que calcular los coeficientes del filtro digital correspondiente. Este proceso también se puede hacer bastante sistemático; sólo hace falta utilizar la expresión general de una transferencia discreta bicuadrática y obtener los coeficientes a partir de ella:

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \qquad \text{Ec. 8.2.5}$$

Utilizando la propiedad de desplazamiento en el tiempo, la ecuación en diferencia resulta:

$$y[n] = b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] - a_1 y[n-1] - a_2 y[n-2] \qquad \text{Ec. 8.2.6}$$

La Figura 8.2.2 demuestra el diagrama en bloques de este tipo de ecuaciones. Si se intercambia el orden de los sistemas en cascada (operación que no altera el sistema en su totalidad gracias a que ambos bloques cumplen con la propiedad de ser lineales e invariante temporales), se puede obtener una pequeña simplificación que permite ahorrar dos bloques de retardo temporal. Esto se ilustra en la Figura 8.2.3.

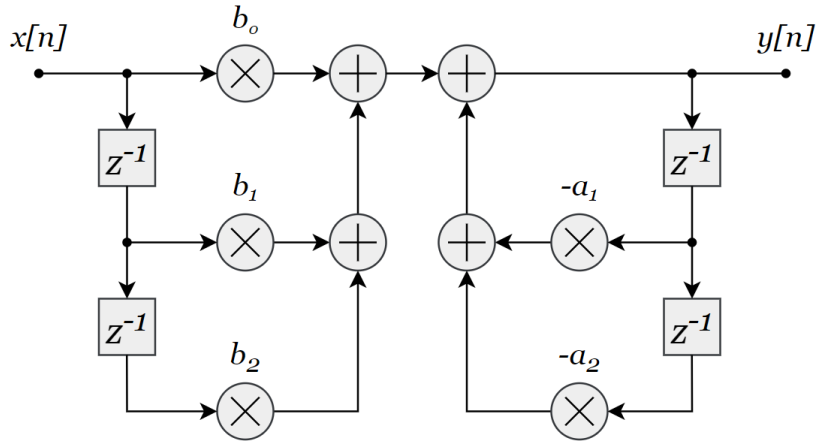


*Figura 8.2.2 – Diagrama en bloques de un filtro recursivo de segundo orden.*

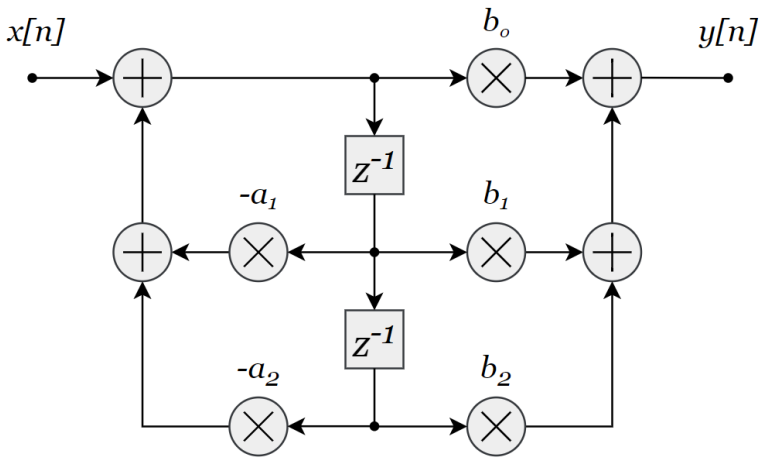


*Figura 8.2.3 – Diagrama en bloques simplificado de un filtro recursivo de segundo orden.*

Nótese que, para no cometer errores a la hora de implementar estos filtros, es preciso normalizar la transferencia en la Ec. 8.2.5 dividiendo por el término independiente del denominador.

Si se repite este proceso para los filtros más comunes de segundo orden, se obtienen los coeficientes resumidos en la Tabla 8.2.1.

| Filtros | $b_0$ | $b_1$ | $b_2$ | $a_1$ | $a_2$ |
|---|---|---|---|---|---|
| **Pasa Bajos** | $\frac{K^2Q}{K^2Q+K+Q}$ | $\frac{2K^2Q}{K^2Q+K+Q}$ | $\frac{K^2Q}{K^2Q+K+Q}$ | $\frac{2(K^2-1)Q}{K^2Q+K+Q}$ | $\frac{K^2Q-K+Q}{K^2Q+K+Q}$ |
| **Pasa Altos** | $\frac{Q}{K^2Q+K+Q}$ | $\frac{-2Q}{K^2Q+K+Q}$ | $\frac{Q}{K^2Q+K+Q}$ | $\frac{2(K^2-1)Q}{K^2Q+K+Q}$ | $\frac{K^2Q-K+Q}{K^2Q+K+Q}$ |
| **Pasa Banda** | $\frac{K}{K^2Q+K+Q}$ | $0$ | $\frac{-K}{K^2Q+K+Q}$ | $\frac{2(K^2-1)Q}{K^2Q+K+Q}$ | $\frac{K^2Q-K+Q}{K^2Q+K+Q}$ |
| **Rechaza Banda** | $\frac{(K^2+1)Q}{K^2Q+K+Q}$ | $\frac{2(K^2-1)Q}{K^2Q+K+Q}$ | $\frac{(K^2+1)Q}{K^2Q+K+Q}$ | $\frac{2(K^2-1)Q}{K^2Q+K+Q}$ | $\frac{K^2Q-K+Q}{K^2Q+K+Q}$ |
| **Pasa Todo** | $\frac{K^2Q-K+Q}{K^2Q+K+Q}$ | $\frac{2(K^2-1)Q}{K^2Q+K+Q}$ | $1$ | $\frac{2(K^2-1)Q}{K^2Q+K+Q}$ | $\frac{K^2Q-K+Q}{K^2Q+K+Q}$ |

*Tabla 8.2.1 – Coeficientes de la Ec. 8.2.6 para los filtros más comunes de segundo orden.*

## 8.3 - Filtros Sintonizables

Con esta información ya es suficiente para llevar a cabo cualquiera de estos cinco filtros elementales de segundo orden en forma digital. Sin embargo, este acercamiento todavía tiene cabida para mejoras. Aquí se deben calcular todos los coeficientes por separado y, además, estos presentan expresiones de funciones trigonométricas al cuadrado que son innecesariamente complejas de evaluar. Esto se torna en un problema, especialmente en estas aplicaciones, ya que es de interés variar los parámetros del filtro de manera continua en algunos efectos. Con esta motivación se desarrollaron los filtros sintonizables.

Se suele partir de un filtro pasa todo sintonizable al cual se le realizan operaciones básicas (como sumas o multiplicaciones por escalares) que permiten transformar a éste en otros filtros de interés. Una observación importante es que este tipo de operaciones siempre mantienen los polos de la función de transferencia, pero pueden manipular la ubicación de los ceros (si se

quiere hacer un cambio en los polos, se deberá partir de otra función de transferencia como se verá a continuación). Para ilustrar este concepto, se tomará como primer ejemplo el caso de un pasa todos de primer orden.

Su función de transferencia en tiempo continuo para una ganancia unitaria es:

$$H_{AP}(s) = \frac{s - \omega_o}{s + \omega_o} \qquad \text{Ec. 8.3.1}$$

Aplicando la transformación bilineal se puede llegar a lo siguiente:

$$H_{AP}(z) = \frac{(1-K) - (1+K)z^{-1}}{(1+K) - (1-K)z^{-1}} = \frac{\frac{1-K}{1+K} - z^{-1}}{1 - \frac{1-K}{1+K}z^{-1}}$$

Para reducir un poco la expresión, se puede definir un nuevo parámetro c que depende de frecuencia de corte:

$$c = \frac{1-K}{1+K} = \frac{1 - \tan\left(\frac{\pi f_c}{f_s}\right)}{1 + \tan\left(\frac{\pi f_c}{f_s}\right)} \qquad \text{Ec. 8.3.2}$$

Finalmente, la función de transferencia del filtro pasa todo de primer orden sintonizable sería:

$$H_{AP}(z) = \frac{c - z^{-1}}{1 - cz^{-1}} \qquad \text{Ec. 8.3.3}$$

Se puede apreciar la ventaja de que solamente hay un coeficiente que calcular comparado con el caso general de tres coeficientes para un filtro genérico de primer orden.

Mucho otros filtros sintonizables se pueden lograr haciendo transformaciones básicas a esta expresión. Para derivar ejemplos, resulta útil remitirse a la función de transferencia pasa todos en variable de Laplace de la Ec. 8.3.1.

Para el caso de un filtro pasa bajos, se puede deducir lo siguiente en variable continua:

$$H_{LP}(s) = \frac{\omega_o}{s + \omega_o} = \frac{1 - H_{AP}(s)}{2}$$

Lo cual también debe ser válido en variable discreta:

$$H_{LP}(z) = \frac{1 - H_{AP}(z)}{2} \qquad \text{Ec. 8.3.4}$$

El mismo proceso se puede llevar a cabo con otro tipo de filtros como es el ejemplo del pasa altos:

$$H_{HP}(z) = \frac{1 + H_{AP}(z)}{2} \qquad \text{Ec. 8.3.5}$$

Resultados como estos se pueden aprovechar para construir una implementación de los filtros basada en diagrama en bloque de la Figura 8.3.1. Según el signo que se utilice en la suma, se

puede obtener un filtro o el otro. Nótese que ahora los filtros solo tienen un parámetro para calcular cada vez que se desea ajustar la frecuencia de corte (Ec. 8.3.2), lo cual los hace bastantes eficientes para aplicaciones en tiempo real.

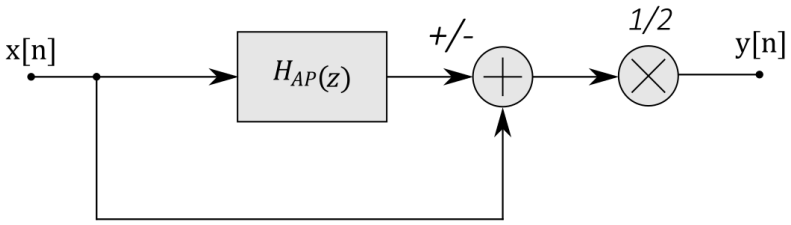


*Figura 8.3.1 – Diagrama en bloques para filtros pasa bajos y pasa altos sintonizables.*

En el caso de los filtros de segundo orden se puede seguir un proceso similar. Primero, es útil establecer la función de transferencia de un filtro pasa todos en variable z consultando a la Tabla 8.2.1:

$$H_{AP2}(z) = \frac{\frac{K^2Q - K + Q}{K^2Q + K + Q} + \frac{2(K^2 - 1)Q}{K^2Q + K + Q}z^{-1} + z^{-2}}{1 + \frac{2(K^2 - 1)Q}{K^2Q + K + Q}z^{-1} + \frac{K^2Q - K + Q}{K^2Q + K + Q}z^{-2}} \qquad \text{Ec. 8.3.6}$$

Una gran ventaja que se puede apreciar por simple inspección es que $b_0 = a_2$, $b_1 = a_1$ y $b_2 = 1$, con lo cual hay que calcular solamente dos tipos de coeficientes en lugar de los cinco coeficientes que necesita un filtro genérico de segundo orden. Al realizar algunas observaciones y aproximaciones, se puede incluso simplificar su cómputo.

Primero, se puede asilar y desarrollar el coeficiente $b_0$ con la Ec. 8.2.2 y la Ec. 8.2.4:

$$b_0 = \frac{K^2Q - K + Q}{K^2Q + K + Q} = \frac{\tan^2\left(\frac{\pi f_c}{f_s}\right)\frac{f_c}{f_b} - \tan\left(\frac{\pi f_c}{f_s}\right) + \frac{f_c}{f_b}}{\tan^2\left(\frac{\pi f_c}{f_s}\right)\frac{f_c}{f_b} + \tan\left(\frac{\pi f_c}{f_s}\right) + \frac{f_c}{f_b}} = \frac{\tan^2\left(\frac{\pi f_c}{f_s}\right) - \tan\left(\frac{\pi f_c}{f_s}\right)\frac{f_b}{f_c} + 1}{\tan^2\left(\frac{\pi f_c}{f_s}\right) + \tan\left(\frac{\pi f_c}{f_s}\right)\frac{f_b}{f_c} + 1}$$

Luego, se utiliza una aproximación de serie de Taylor, suponiendo que el argumento de la tangente es relativamente pequeño (generalmente se cumple gracias a las altas frecuencias de muestreo utilizadas):

$$\tan(x) \approx x, |x| \ll 1$$

La tangente al cuadrado se desprecia en la suma y, en el segundo término, la constante que multiplica a la tangente entra dentro del argumento:

$$\frac{\tan^2\left(\frac{\pi f_c}{f_s}\right) - \tan\left(\frac{\pi f_c}{f_s}\right)\frac{f_b}{f_c} + 1}{\tan^2\left(\frac{\pi f_c}{f_s}\right) + \tan\left(\frac{\pi f_c}{f_s}\right)\frac{f_b}{f_c} + 1} \approx \frac{1 - \tan\left(\frac{\pi f_c}{f_s}\frac{f_b}{f_c}\right)}{1 + \tan\left(\frac{\pi f_c}{f_s}\frac{f_b}{f_c}\right)} = \frac{1 - \tan\left(\frac{\pi f_b}{f_s}\right)}{1 + \tan\left(\frac{\pi f_b}{f_s}\right)}$$

Este es el primer parámetro del filtro pasa todo sintonizable, denominado “c”. Si bien es muy similar al caso de primer orden, en esta ocasión, solamente tiene una dependencia con del ancho de banda deseado:

$$c = \frac{1 - \tan\left(\frac{\pi f_b}{f_s}\right)}{1 + \tan\left(\frac{\pi f_b}{f_s}\right)} \approx b_0 = a_2 \qquad \text{Ec. 8.3.7}$$

Finalmente, falta simplificar la expresión de $b_1 = a_1$. Se puede expresar este coeficiente en función de c con la siguiente observación:

$$b_1 \frac{K^2 + 1}{K^2 - 1} - 1 = b_0 \approx c \Longrightarrow b_1 \approx (1 + c)\frac{K^2 - 1}{K^2 + 1}$$

El segundo factor se puede reducir mediante algunas relaciones trigonométricas:

$$\frac{K^2 - 1}{K^2 + 1} = \frac{\tan^2\left(\frac{\pi f_c}{f_s}\right) - 1}{\tan^2\left(\frac{\pi f_c}{f_s}\right) + 1}$$

$$\cos(2\alpha) = \cos^2(\alpha) - \text{sen}^2(\alpha) \Longrightarrow \tan^2(\alpha) = 1 - \frac{\cos(2\alpha)}{\cos^2(\alpha)}$$

Reemplazando:

$$\frac{\tan^2\left(\frac{\pi f_c}{f_s}\right) - 1}{\tan^2\left(\frac{\pi f_c}{f_s}\right) + 1} = \frac{-\frac{\cos\left(2\frac{\pi f_c}{f_s}\right)}{\cos^2\left(\frac{\pi f_c}{f_s}\right)}}{2 - \frac{\cos\left(2\frac{\pi f_c}{f_s}\right)}{\cos^2\left(\frac{\pi f_c}{f_s}\right)}} = \frac{-\cos\left(2\frac{\pi f_c}{f_s}\right)}{2\cos^2\left(\frac{\pi f_c}{f_s}\right) - \cos\left(2\frac{\pi f_c}{f_s}\right)}$$

Gracias a otra relación trigonométrica, se puede simplificar el denominador:

$$\cos^2(\alpha) = \frac{\cos(2\alpha) + 1}{2} \Longrightarrow 2\cos^2(\alpha) - \cos(2\alpha) = 1$$

El resultado es el último parámetro del filtro sintonizable. Éste depende solamente de la frecuencia de corte deseada y se suele llamar d:

$$d = \frac{K^2 - 1}{K^2 + 1} = -\cos\left(\frac{2\pi f_c}{f_s}\right) \qquad \text{Ec. 8.3.8}$$

Mediante estos dos parámetros, se puede reescribir la transferencia del filtro pasa todo en la Ec. 8.3.6 para tener un pasa todo sintonizable:

$$H_{AP2}(z) \approx \frac{c + (1 + c)dz^{-1} + z^{-2}}{1 + (1 + c)dz^{-1} + cz^{-2}} \qquad \text{Ec. 8.3.9}$$

Donde c y d se definen según la Ec. 8.3.7 y la Ec. 8.3.8.

Un proceso muy similar al empleado en el filtro pasa todo de primer orden se puede utilizar para obtener otro tipo de filtros partiendo de este resultado. En este caso se presenta el ejemplo de los filtros pasa banda y rechaza banda para completar todos los casos.

$$H_{BP}(z) = \frac{1 - H_{AP2}(z)}{2} \qquad \text{Ec. 8.3.10}$$

$$H_{BR}(z) = \frac{1 + H_{AP2}(z)}{2} \qquad \text{Ec. 8.3.11}$$

Estos también se pueden implementar mediante el diagrama en bloques de la Figura 8.3.2

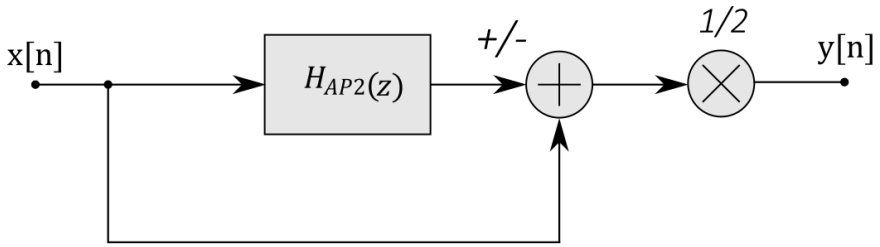


*Figura 8.3.2 – Diagrama en bloques para filtros pasa banda y rechaza banda sintonizables.*

# 9 - Efectos Basados en Filtros

## 9.1 - Ecualizador

Muchas veces cuando se trabaja con una señal de audio se busca manipular la respuesta en frecuencia para poder reforzar o atenuar ciertos anchos de banda en el espectro. Esto es llevado a cabo por un ecualizador.

En concepto, se tienen una serie de filtros que abarcan todo el rango audible (aproximadamente entre 20Hz y 20KHz), el primer y último filtro ecualizan todas las frecuencias menores o mayores a su frecuencia de corte respectivamente mientras que el resto de los filtros se encargan de ecualizar un ancho de banda en torno a su frecuencia de corte. La cantidad de filtros y su respectivo ancho de banda es una decisión que generalmente depende del diseñador ya que presenta una solución de compromiso; mientras más filtros se tengan, más grados de libertad se tienen para ecualizar el espectro audible, pero esto implica un aumento en la complejidad y la operación del sistema.

El diseño más habitual es el de utilizar diez filtros totales cuyas frecuencias de corte resulten de multiplicar o dividir por dos la frecuencia de 1KHz. El arreglo de frecuencias de corte, entonces, es el siguiente: [31,25Hz 62,5Hz 125Hz 250Hz 500Hz 1KHz 2KHz 4KHz 8KHz 16KHz].

Como se mencionó previamente, el primer y último filtro tienen una respuesta tal que pueden reforzar o atenuar las frecuencias menores o mayores a su frecuencia de corte respectivamente. Sin embargo, como ocurre en todos los casos para los filtros de un ecualizador, el resto del espectro audible debe presentar una transferencia aproximadamente unitaria para no influir sobre los otros filtros. Esto se puede lograr con un tipo de filtro denominado “shelving filter”. Si bien el concepto de “shelving filter” está generalizado dentro de la bibliografía para cualquier orden, en este trabajo se tomó la decisión de utilizar el primer orden con el objetivo de reducir tanto la carga sobre el procesador como la complejidad del código.

Hay cuatro tipos de filtros shelving a implementar, para explicar esto, se puede recurrir a un ejemplo: la transferencia en variable continua de Laplace de un caso particular de filtro shelving de primer orden es la siguiente:

$$H(s) = \frac{s + V_0\omega_c}{s + \omega_c}\ , V_0 > 1 \qquad \text{Ec. 9.1.1}$$

Donde Vo es la ganancia en veces.

Si se realizan los límites de tendencia a frecuencia cero y a frecuencia infinita se puede tener una intuición sobre la forma del gráfico de respuesta en frecuencia:

$$\lim_{s\to 0} H(s) = \lim_{s\to 0} \frac{s + V_0\omega_c}{s + \omega_c} = V_0$$

$$\lim_{s\to \infty} H(s) = \lim_{s\to \infty} \frac{s + V_0\omega_c}{s + \omega_c} = 1$$

También se puede concluir por inspección que el diagrama de root locus tendrá un polo en $-\omega_c$ y un cero en $-V_0\omega_c$ , con lo cual, habrá aproximadamente una región de transición con una pendiente de 20dB por década entre estos dos.

Con toda esta información, se puede deducir que este filtro realiza un refuerzo sobre los graves con una ganancia de Vo. Un ejemplo se puede ver en la Figura 9.1.1.

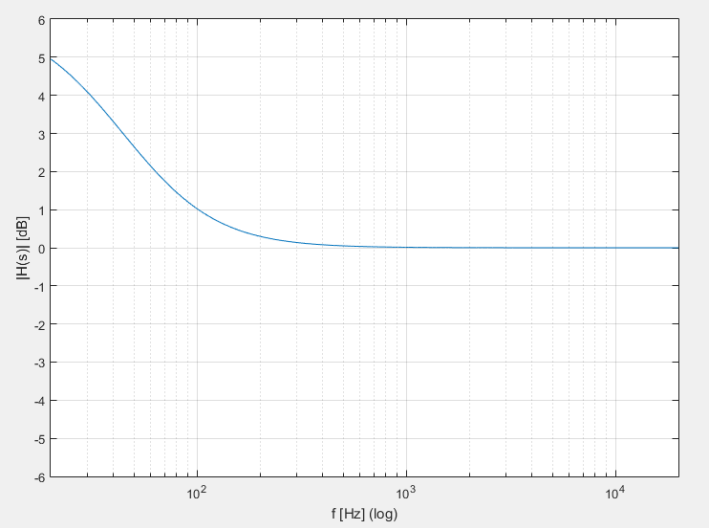


*Figura 9.1.1 – Filtro shelving de refuerzo de graves de la Ec. 9.1.1 con frecuencia de corte de 31,25 Hz y ganancia de 2 veces (6 dB).*

En este momento probablemente surge la duda sobre la necesidad de restringir Vo a valores mayores a uno (o de refuerzo) ya que valores menores que uno también respetarían las tendencias en los límites y el filtro funcionaría. El problema con esto es que lo que realmente se está buscando es tener la característica inversa en la atenuación (respecto al refuerzo) y usar un valor inverso en Vo no basta (ver Figura 9.1.2).

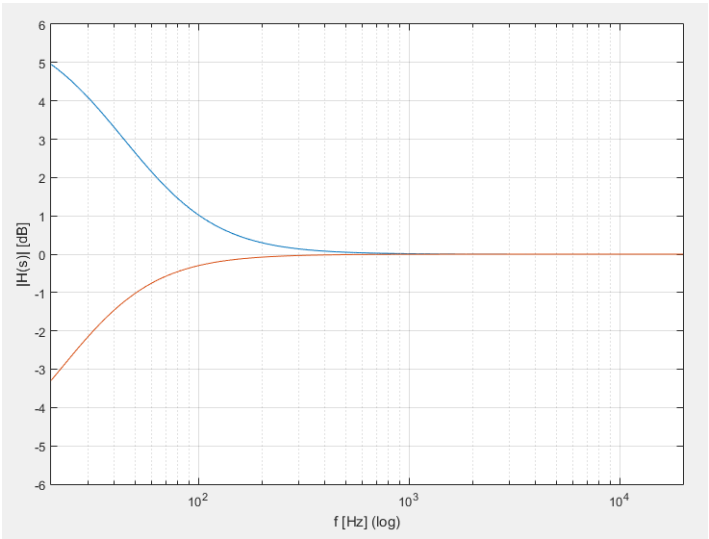


*Figura 9.1.2 - Comparación del filtro de la Figura 9.1.1 contra un filtro igual pero de ganancia inversa (-6 dB). Nótese la anti simetría de las curvas.*

Para lograr invertir esta característica, se deben cambiar de lugar los polos con los ceros. Por lo tanto, también debe modificar la función de transferencia:

$$H(s) = \frac{s + \omega_c}{s + \frac{\omega_c}{V_0}}, V_0 < 1 \qquad \text{Ec. 9.1.2}$$

Si ahora se compara una gráfica de esta función de transferencia de atenuación contra la función de transferencia de refuerzo para valores de Vo inversos, se puede ver la simetría de las características.

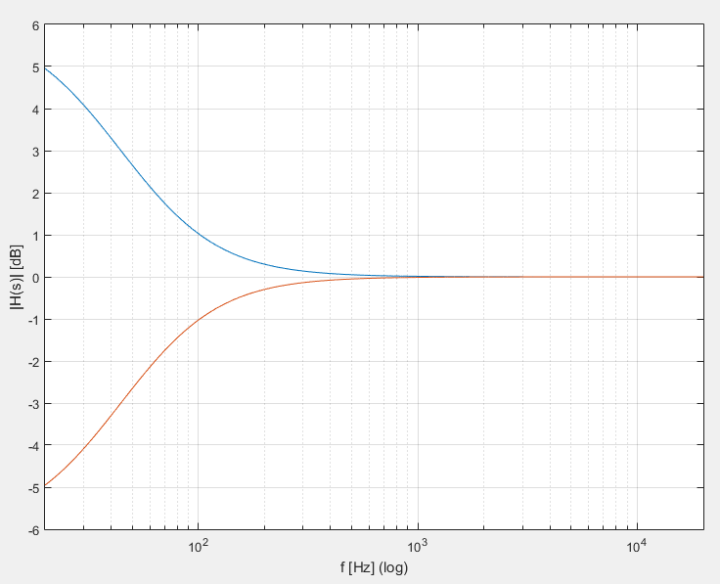


*Figura 9.1.3 - Comparación del filtro de la Figura 9.1.1 contra un filtro de la Ec. 9.1.2 con ganancia inversa (-6 dB). Nótese, en este caso, la simetría de las curvas.*

Este proceso se debe repetir para frecuencias altas, de esta manera, se obtienen cuatro filtros posibles en total. La Tabla 9.1.1 presenta un resumen de estas características.

| **Filtros Shelving de 1er orden** | **Refuerzo ($V_0 > 1$)** | **Atenuación ($V_0 < 1$)** |
|---|---|---|
| **Graves** | $H_{RG}(s) = \frac{s + V_0\omega_c}{s + \omega_c}$ | $H_{AG}(s) = \frac{s + \omega_c}{s + \frac{\omega_c}{V_0}}$ |
| **Agudos** | $H_{RA}(s) = \frac{V_0 s + \omega_c}{s + \omega_c}$ | $H_{AA}(s) = \frac{s + \omega_c}{\frac{s}{V_0} + \omega_c}$ |

*Tabla 9.1.1 – Funciones de transferencia para los filtros shelving de refuerzo y atenuación de agudos y graves.*

Una implementación de estos filtros de manera digital requiere obtener la ecuación de transferencia discreta correspondiente mediante la transformación bilineal. También sería deseable obtener un filtro sintonizable para poder realizar menos cuentas en el procesador al variar parámetros. Para realizar esto último, el camino más sencillo es el de relacionar estos filtros con los pasa todos sintonizables vistos anteriormente.

Los casos de refuerzo, tanto en graves como en agudos, son los más sencillos ya que los polos del filtro resultante son los mismos que el de un pasa todos de primer orden:

$$H_{RG}(s) = 1 + \left[\frac{V_0 - 1}{2}(1 - H_{AP}(s))\right]$$

$$H_{RA}(s) = 1 + \left[\frac{V_0 - 1}{2}(1 + H_{AP}(s))\right]$$

Para la atenuación, se debe tomar un filtro pasa todos modificado de tal manera de poder influir sobre los polos de la transferencia:

$$H_{AP(AG)}(s) = \frac{s - \frac{\omega_c}{V_0}}{s + \frac{\omega_c}{V_0}}$$

$$H_{AP(AA)}(s) = \frac{\frac{s}{V_0} - \omega_c}{\frac{s}{V_0} + \omega_c}$$

Si se transforman estos nuevos filtros a variable discreta, se obtiene un resultado muy similar al de un pasa todos normal. Solamente cambia el parámetro sintonizable c (ver Tabla 9.1.2).

| Parámetro c | Refuerzo | Atenuación |
|---|---|---|
| **Graves** | $c_{RG} = \frac{1 - \tan\left(\frac{\pi f_c}{f_s}\right)}{1 + \tan\left(\frac{\pi f_c}{f_s}\right)}$ | $c_{AG} = \frac{V_0 - \tan\left(\frac{\pi f_c}{f_s}\right)}{V_0 + \tan\left(\frac{\pi f_c}{f_s}\right)}$ |
| **Agudos** | $c_{RA} = \frac{1 - \tan\left(\frac{\pi f_c}{f_s}\right)}{1 + \tan\left(\frac{\pi f_c}{f_s}\right)}$ | $c_{AA} = \frac{1 - V_0 \tan\left(\frac{\pi f_c}{f_s}\right)}{1 + V_0 \tan\left(\frac{\pi f_c}{f_s}\right)}$ |

*Tabla 9.1.2 – Cálculo del parámetro c para los filtros shelving de refuerzo y atenuación de agudos y graves.*

Habiendo definido estas variaciones, se puede expresar las funciones de transferencia de los casos de atenuación de la siguiente manera:

$$H_{AG}(s) = 1 + \left[\frac{V_0 - 1}{2}\left(1 - H_{AP(AG)}(s)\right)\right]$$

$$H_{AA}(s) = 1 + \left[\frac{V_0 - 1}{2}\left(1 + H_{AP(AA)}(s)\right)\right]$$

| Filtros | Refuerzo | Atenuación |
|---|---|---|
| **Graves** | $H_{RG}(z) = 1 + \left[\frac{V_0 - 1}{2}(1 - H_{AP}(z, c_{RG}))\right]$ | $H_{AG}(z) = 1 + \left[\frac{V_0 - 1}{2}(1 - H_{AP}(z, c_{AG}))\right]$ |
| **Agudos** | $H_{RA}(z) = 1 + \left[\frac{V_0 - 1}{2}(1 + H_{AP}(z, c_{RA}))\right]$ | $H_{AA}(z) = 1 + \left[\frac{V_0 - 1}{2}(1 + H_{AP}(z, c_{AA}))\right]$ |

*Tabla 9.1.3 – Funciones de transferencia en variable z para los cuatro filtros de shelving.*

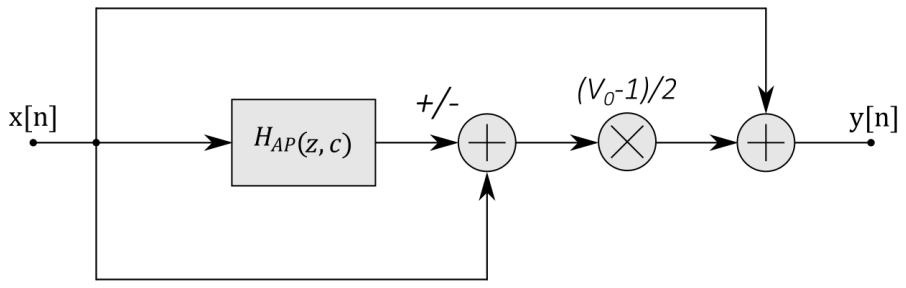


*Figura 9.1.4 – Diagrama en bloques de los filtros shelving.*

Una vez resueltos los filtros shelving, faltan definir los filtros restantes utilizados en frecuencias medias. Éstos, como se mencionó previamente, deben atenuar o reforzar un ancho de banda alrededor de su frecuencia central mientras que el resto de su respuesta en frecuencia debe ser aproximadamente unitaria. Este tipo de filtro se denominan “peak filter” o filtro pico. En concepto, no son muy distintos a un filtro de banda pasante ya que tampoco se pueden ser

implementados mediante una transferencia bilineal. Esto significa que se necesitan como mínimo dos polos para poder tener esta característica.

En la bibliografía, la transferencia de un peak filter se suele definir mediante la de un filtro pasa todos de segundo orden. Esto ayuda a simplificar el proceso de pasaje a variable discreta. También ocurre lo mismo que en el caso del filtro shelving ya que hay que diferenciar la transferencia implementada en el refuerzo contra la de la atenuación para poder mantener la simetría en la característica:

$$H_{RM}(z) = 1 + \left[\frac{V_0 - 1}{2}\left(1 - H_{AP2}(z, c_{RM})\right)\right], c_{RM} = \frac{1 - \tan\left(\frac{\pi f_c}{f_s}\right)}{1 + \tan\left(\frac{\pi f_c}{f_s}\right)}$$

$$H_{AM}(z) = 1 + \left[\frac{V_0 - 1}{2}\left(1 - H_{AP2}(z, c_{AM})\right)\right], c_{AM} = \frac{V_0 - \tan\left(\frac{\pi f_c}{f_s}\right)}{V_0 + \tan\left(\frac{\pi f_c}{f_s}\right)}$$

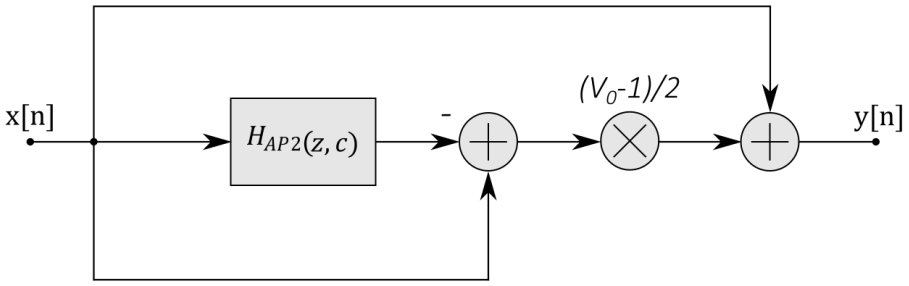


*Figura 9.1.5 – Diagrama en bloques de los peak filter.*

El parámetro del ancho de banda requiere una serie de consideraciones. Primero, aquí no tiene el mismo sentido tradicional que en un filtro pasa banda debido a que no se puede medir una caída de (por ejemplo) 3dB si el filtro amplifica menos de 3dB o, lo que es lo mismo, Vo < 2. Sin embargo, hay una relación entre el ancho de banda y la tasa de crecimiento o decrecimiento de la curva. El problema es entonces elegir el ancho de banda de cada uno de los ocho filtros de frecuencias medias de tal manera de cubrir de la manera más uniforme posible a todo el espectro de interés.

Todas las frecuencias de corte están separadas por un factor de dos, así que una buena idea es tratar de mantener el factor de calidad Q constante y multiplicar los anchos de banda por un factor de dos a medida que aumentan las frecuencias de corte. Esto significa que la única variable por determinar es el ancho de banda del primer filtro de frecuencias medias. Para poder diseñar este valor, se observó qué ocurría con el gráfico de respuesta en frecuencia del conjunto de los ocho filtros.

A medida que el ancho de banda inicial se aumentaba, la respuesta era más uniforme pero los filtros comenzaban a influir sobre sus vecinos cada vez más; si cada uno se calibraba para una ganancia de 6dB, la respuesta agregada se establecía a una ganancia mucho mayor (en la Figura 9.1.6 se puede ver que con un ancho de banda inicial de 50Hz y la respuesta llega hasta 12 dB):

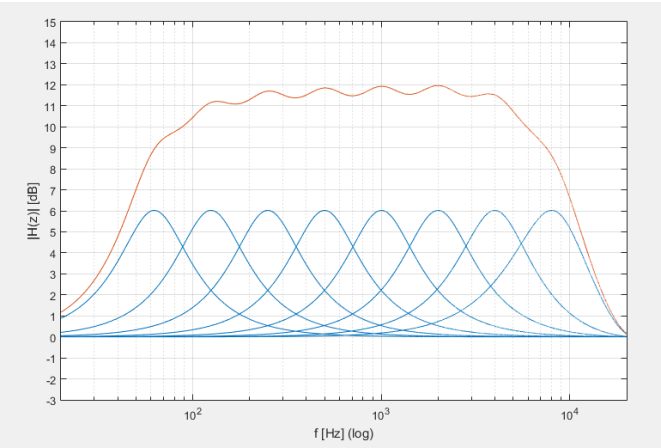


*Figura 9.1.6 – Ecualizador de diez bandas con filtros individuales calibrados a un ancho de banda inicial de 50 Hz y ganancia de 6 dB. La curva naranja representa la respuesta acumulada de todos los filtros mientras que las curvas azules representan a cada filtro empleado en el diseño de manera separada.*

Si el ancho de banda se disminuía lo suficiente, las ganancias se respetaban, pero no se llegaba a cubrir todo el ancho de banda de interés, solamente una porción pequeña alrededor de las frecuencias de corte (se puede apreciar en la Figura 9.1.7).

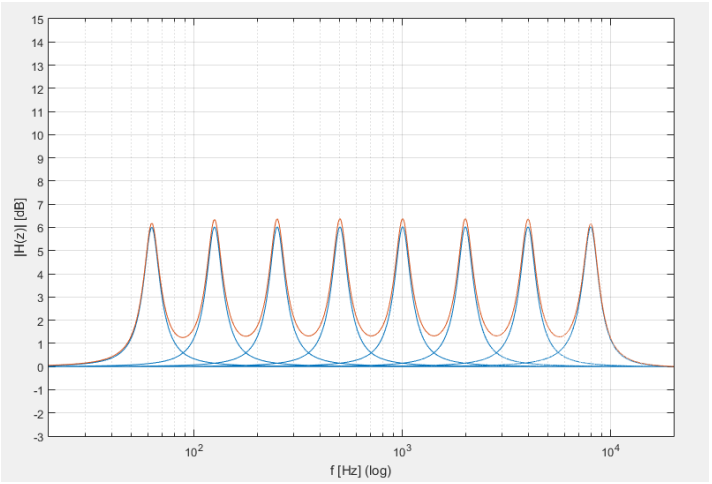


*Figura 9.1.7 - Ecualizador de diez bandas con filtros individuales calibrados a un ancho de banda inicial de 10 Hz y ganancia de 6 dB.*

Finalmente, se decidió que un ancho de banda inicial de alrededor 20Hz resuelve relativamente bien esta solución de compromiso. Si se deseasen solucionar los dos problemas juntos, se debería optar por filtros pico de mayor orden (lo cual aumentaría innecesariamente la complejidad de cómputo en este trabajo).

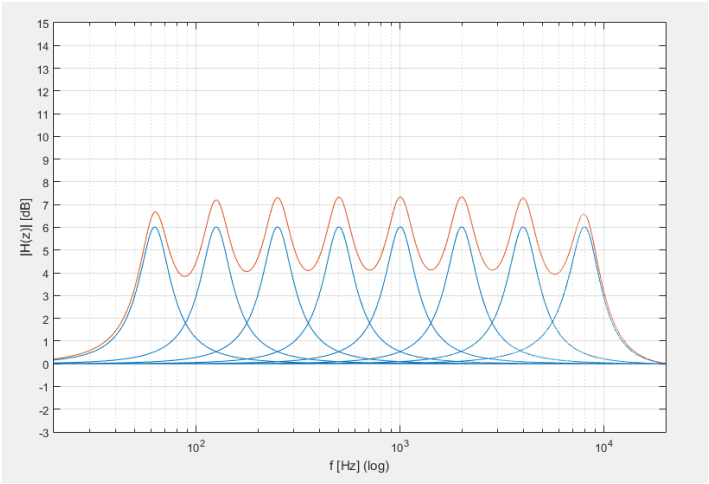


*Figura 9.1.8 - Ecualizador de diez bandas con filtros individuales calibrados a un ancho de banda inicial de 20 Hz y ganancia de 6 dB.*

Una vez diseñados los diez filtros, se puede ver cómo funciona el sistema completo en un solo gráfico (representado en la Figura 9.1.9).

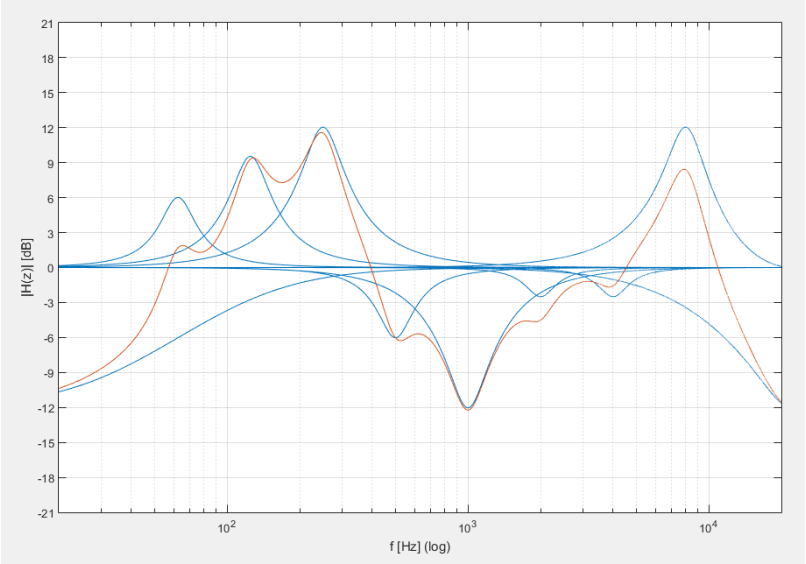


*Figura 9.1.9 - Ecualizador de diez bandas con filtros individuales calibrados a un ancho de banda inicial de 20 Hz y ganancia en veces según el arreglo Vo = [0.25 2 3 4 0.5 0.25 0.75 0.75 4 0.25].*

Como conclusión, se puede ver que los filtros de primer orden son lógicamente los que más influencia tienen sobre sus vecinos. Más allá de las limitaciones de usar ordenes bajos, el agregado de las respuestas suele estar dentro de un rango aceptable de decibeles respecto de las respuestas individuales en la mayoría de los casos.

## 9.2 - Implementación en Código: Ecualizador

Una parte importante del código se encuentra en la inicialización: se deben fijar las frecuencias de corte, anchos de banda, ganancias y parámetros para todos los filtros. Aquí se asume que se dispone de una variable global SR (sample rate) que puede proveer el valor de la frecuencia de muestreo para hacer las cuentas correspondientes.

Las funciones peak y shelve se escribieron como ecuaciones en diferencia a partir de las expresiones planteadas previamente. Éstas se invocan en orden de frecuencia de corte dentro de la rutina principal para llevar a cabo el efecto.

A continuación, un ejemplo del código:

```
//Ecualizador.c

#define N 10;

int salida = 0;
int i = 0;
float x0[N],x1[N],x2[N],y0[N],y1[N],y2[N];
float fc[N],fb[N],Vo[N],d[N],c[N];

void init()
{
    //frecuencias de corte
    fc[0] = 31.25;fc[1] = 62.5;fc[2] = 125;
    fc[3] = 250;fc[4] = 500;fc[5] = 1000;
    fc[6] = 2000;fc[7] = 4000;fc[8] = 8000;fc[9] = 16000;
    //anchos de banda
    for(i = 1; i < N-1; i++)
        fb[i] = 20 * pow(2, i-1);
    //ganancias
    Vo[0] = 0.25;Vo[1] = 2;Vo[2] = 3;
    Vo[3] = 4;Vo[4] = 0.5;Vo[5] = 0.25;
    Vo[6] = 0.75;Vo[7] = 0.75;Vo[8] = 4;Vo[9] = 0.25;
    //parametros low shelve
    if(Vo[0] >= 1)
        c[0] = (1-tan(3.1416*fc[0]/SR))/(1+tan(3.1416*fc[0]/SR));
    else
        c[0] = (Vo[0]-tan(3.1416*fc[0]/SR))/(Vo[0]+tan(3.1416*fc[0]/SR));
    //parametros peak
    for(i = 1; i < N-1; i++)
    {
        d[i] = -cos(2*3.1416*fc[i]/SR);
        if(Vo[i] >= 1)
            c[i] = (1-tan(3.1416*fb[i]/SR))/(1+tan(3.1416*fb[i]/SR));
        else
            c[i] = (Vo[i]-tan(3.1416*fb[i]/SR))/(Vo[i]+tan(3.1416*fb[i]/SR));
    }
    //parametros high shelve
    if(Vo[9] >= 1)
        c[9] = (1-tan(3.1416*fc[9]/SR))/(1+tan(3.1416*fc[9]/SR));
    else
        c[9] = (1-Vo[9]*tan(3.1416*fc[9]/SR))/(1+Vo[9]*tan(3.1416*fc[9]/SR));
}

int ecualizador(int entrada)
{
    salida = shelve(entrada,0);
    for(int i = 1; i < N - 1; i++)
        salida = peak(salida,i);
    salida = shelve(salida,9);
    return salida;
}
```

```
int peak(int in, int i)
{
    x2[i] = x1[i];
    x1[i] = x0[i];
    x0[i] = in;
    y2[i] = y1[i];
    y1[i] = y0[i];
    y0[i] = c[i] * x0[i] + d[i] * (1 + c[i]) * x1[i] + x2[i] - d[i] * (1 + c[i]) * y1[i] - c[i] * y2[i];
    in = 0.5 * (Vo[i] - 1) * (in - y0[i]) + in;
    return in;
}

int shelve(int in, int i)
{
    x1[i] = x0[i];
    x0[i] = in;
    y1[i] = y0[i];
    y0[i] = c[i] * x0[i] - x1[i] + c[i] * y1[i];
    if(i == 0)
        in = 0.5 * (Vo[i] - 1) * (in - y0[i]) + in;
    else
        in = 0.5 * (Vo[i] - 1) * (in + y0[i]) + in;
    return in;
}

```

## 9.3 - Auto-Wah

El Wah-Wah es uno de los efectos más famosos en el mundo de la música. Su nombre se basa en el peculiar sonido que produce.

Su construcción es bastante sencilla ya que solamente se requiere un filtro pasa banda cuya frecuencia de corte varíe de alguna forma con el tiempo. Si bien este parámetro es habitualmente controlado por el usuario mediante un pedal, en este caso, se optó por utilizar un LFO que maneje la variación automáticamente (ver anexo 16.1 - ). Debido a esto, un nombre más apropiado para esta implementación sería Auto-Wah.

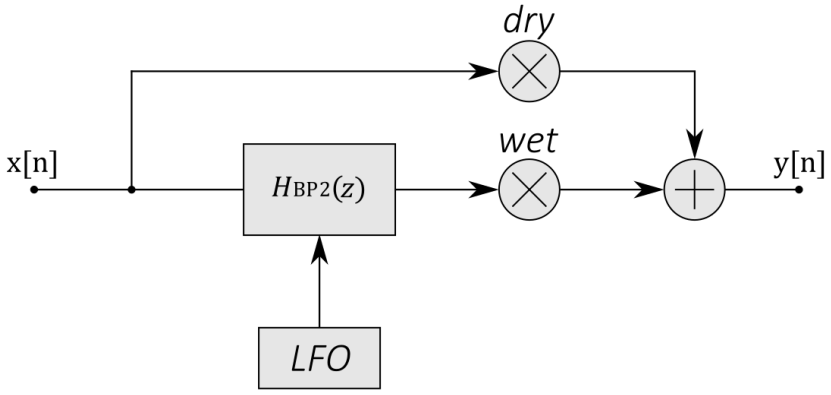


*Figura 9.3.1 – Diagrama en bloques para la implementación del Auto-Wah.*

La implementación del filtro pasa banda de segundo orden se puede hacer más eficiente mediante un filtro sintonizable como el de la Ec. 8.3.10.

Los controles "dry" y "wet" de la Figura 9.3.1 hacen referencia a la señal original o seca y a la señal alterada por el efecto o empapada. Son parámetros variables cuyo objetivo es mezclar las señales de tal manera que el efecto se note pero que, a la vez, no sea tan agresivo. Si bien esto se puede dejar a criterio del usuario, tampoco es raro encontrar diseños en los que el fabricante los deja en un valor fijo que él considera apropiado. Una tercera opción es la de reducir los grados de libertad de dos a uno mediante la relación:

$$dry = 1 - wet\,, 0 < dry < 1, 0 < wet < 1$$

Esto permite dar todas las proporciones posibles entre los parámetros mientras que se garantiza que se evita la saturación en el nodo de la suma.

## 9.4 - Implementación en Código: Auto-Wah

Para esta implementación se escribió la función BP2 que es llamada en cada ciclo para calcular de manera recursiva el filtro pasa banda sintonizable de segundo orden. Los parámetros que se pueden variar en este filtro son el ancho de banda y la frecuencia de corte (controlados por c y d respectivamente). Sin embargo, solamente es preciso actualizar la variable d en cada llamado ya que éste controla la frecuencia de corte entregada por el LFO. Por otro lado, la variable c solamente se debe cargar una vez. Para lograr esto, se define una función de inicialización que se invoca en el programa dentro de la secuencia de arranque (antes del ciclo de iteración).

En este ejemplo de código, también se optó por evitar la saturación a la salida mediante un único control de la mezcla denominado volumen:

```
1  //Auto-Wah.c
2
3  #define TRINAGULAR 0
4  #define SINUSOIDAL 1
5
6  int salida;
7  int x2,x1,x0,y2,y1,y0;
8  float fb = 100, volume = 0.5;
9  int modulacion = SINUSOIDAL;
10 float fcentral,c,d;
11
12 void init()
13 {
14   c = (1-tan(3.1416*fb/SR))/(1+tan(3.1416*fb/SR));
15   d = -cos(2*3.1416*fcentral/SR);
16 }
17
18 int autowah(int entrada)
19 {
20   fcentral = LFO(modulacion);
21   d = -cos(2*3.1416*fcentral/SR);
22   salida = BP2(entrada);
23   salida = volume * salida + (1-volume) * entrada;
24   return salida;
25 }
26
27 int BP2(int in)
28 {
29   x2 = x1;
30   x1 = x0;
31   x0 = in;
32   y2 = y1;
33   y1 = y0;
34   y0 = (1-c) * 0.5 * x0 - (1-c) * 0.5 * x2 - d * (1+c) * y1 - c * y2;
35   return y0;
36 }
```

## 9.5 - Phaser

Otro de los efectos que se pueden lograr mediante filtros variables en el tiempo es el Phaser. Su nombre se debe a que la respuesta en fase del sistema juega un papel importante dentro del procesamiento de su característico sonido. Para recrear este efecto en particular, se va a estudiar brevemente el funcionamiento de una contraparte analógica de este efecto muy popular en el mercado por el fabricante MXR: el pedal Phase 90.

Este diseño (ver Figura 9.5.1) consiste en tres etapas distintas. Primero se utiliza un buffer para adaptar impedancias. Luego hay una sección que emplea cuatro filtros pasa todos idénticos en serie al camino de la señal. Finalmente, la etapa de salidas adapta impedancias nuevamente y suma la salida de la primer y la segunda etapa. Lo más interesante ocurre en la etapa intermedia, donde estos filtros pasa todos de primer orden distorsionan la fase de la señal de entrada entre

0° y -180° cada uno. Luego de repetir cuatro veces este proceso, la fase puede encontrarse entre 0° y -720°. Las frecuencias para las cuales la respuesta de fase sea aproximadamente -180° o -540° van a ser atenuadas fuertemente al ser sumadas directamente con la señal original en la última etapa (dos señales idénticas sumadas en contrafase se cancelan).

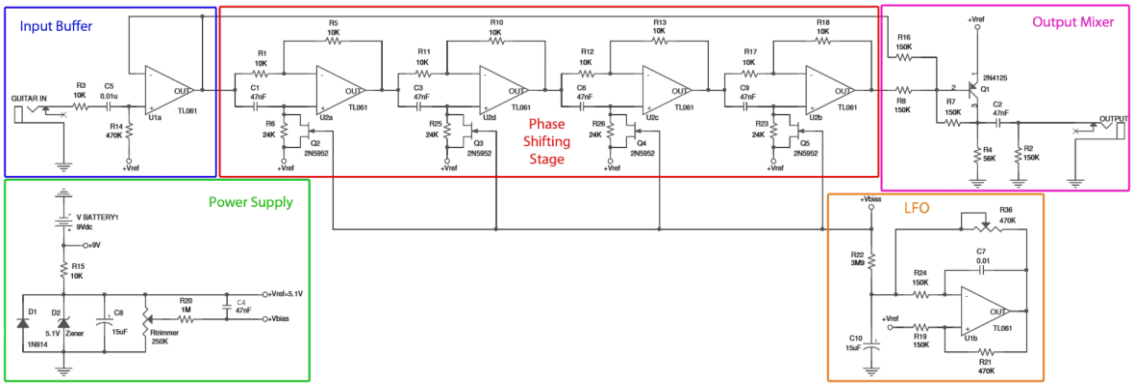


*Figura 9.5.1 – Circuito esquemático del pedal MXR phase 90.*

Esto significa que, la totalidad del filtro se comporta como dos filtros notch de segundo orden donde las dos frecuencias rechazadas son la que presentan desfasajes de -180° o -540° al pasar por la segunda etapa. Sin embargo, estas dos frecuencias no son fijas, sino que son controladas dinámicamente gracias al LFO analógico. Éste genera una tensión con forma aproximadamente triangular que excita las compuertas de cuatro transistores JFET. Los transistores se encuentran trabajando en zona lineal, lo que significa que son equivalentes a una resistencia entre drenaje y fuente cuyo valor en ohm es controlado por la tensión de compuerta. De esta manera, junto con la resistencia que tienen en paralelo, conforman un parámetro variable que hace dinámica la respuesta de estos filtros pasa todos.

En forma digital, se pueden implementar los filtros pasa todos sintonizables de primer orden según la Ec. 8.3.1. El resto se reduce a hacer las cuentas correspondientes entre la señal original y la señal con efecto. La Figura 9.5.2 presenta un diagrama en bloques de este sistema.

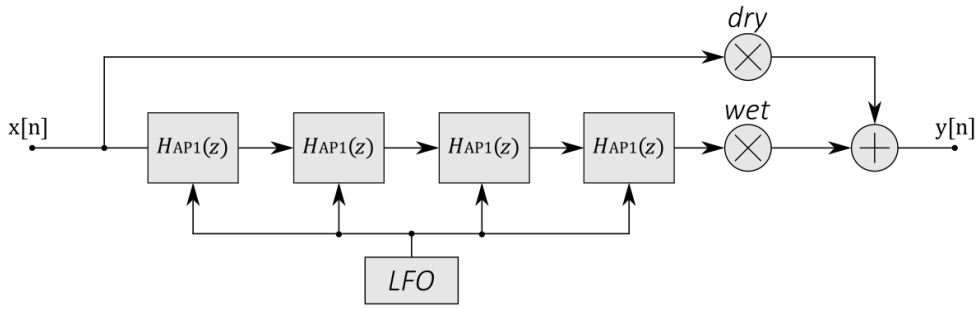


*Figura 9.5.2 – Diagrama en bloques para la implementación del Phaser.*

## 9.6 - Implementación en Código: Phaser

En este caso, la función AP1 utiliza arreglos para poder ser llamada con distintos índices. Esto permite utilizar la misma función para los cuatro filtros.

La variable volumen se volvió a utilizar para controlar la saturación en la salida. Cunado su valor se encuentra en 0.5 es cuando más notable son los notches ya que las amplitudes de la señal original y la señal invertida en fase son aproximadamente iguales.

Este es un ejemplo de código para un phaser:

```
//Phaser.c

#define TRINAGULAR 0
#define SINUSOIDAL 1

int salida;
int x2[4],x1[4],x0[4],y2[4],y1[4],y0[4];
float volume = 0.5;
int modulacion = SINUSOIDAL;
float fcentral,c;

int phaser(int entrada)
{
  fcentral = LFO(modulacion);
  c = (1-tan(3.1416*fcentral/SR))/(1+tan(3.1416*fcentral/SR));
  salida = AP1(entrada,0);
  salida = AP1(salida,1);
  salida = AP1(salida,2);
  salida = AP1(salida,3);
  salida = volume * salida + (1-volume) * entrada;
  return salida;
}

int AP1(int in, int i)
{
  x1[i] = x0[i];
  x0[i] = in;
  y1[i] = y0[i];
  y0[i] = c * x0[i] - x1[i] + c * y1[i];
  return y0[i];
}
```

# 10 - Efectos Basados en Delay

## 10.1 - Delay

La motivación detrás del "delay" (o demora en castellano) se puede encontrar en algunos fenómenos naturales. Por ejemplo: el rebote de una onda sonora sobre una superficie produce que el oyente escuche una copia de la señal original retrasada por un determinado tiempo. La ecuación que representa lo que está ocurriendo en el dominio del tiempo es la siguiente:

$$y(t) = x(t) + gx(t - \tau)\,, g > 0 \qquad \text{Ec. 10.1.1}$$

Donde τ es el retardo correspondiente y g representa la atenuación que sufre la onda al viajar por un camino más largo.

La interpretación de este fenómeno en el dominio de la frecuencia es muy interesante y está estudiada muy en profundidad en el campo de las telecomunicaciones. Cuando se producen multitrayectorias de una señal modulada, se obtienen una o más repeticiones no deseadas de la señal original que, según cuánto se hayan atrasado, pueden ocasionar interferencia constructiva o destructiva. Lo mismo ocurre al estudiar este sistema: si bien la señal demorada, no modifica la amplitud salvo por un factor de atenuación, sí modifica la fase. Suponiendo una entrada periódica; cuando ambas señales están en fase, se produce una interferencia constructiva (o una ganancia por el factor 1+g) y cuando ambas señales están en contrafase, se produce una interferencia destructiva (o una atenuación por el factor de 1-g). Para que ocurra lo primero, debe cumplirse que, teniendo un período T en la entrada:

$$\tau = kT, k \in \mathbb{Z}\ > 0 \qquad \text{Ec. 10.1.2}$$

Para que ocurra lo último, entonces:

$$\tau = \frac{(2k-1)}{2}T, k \in \mathbb{Z}\ > 0 \qquad \text{Ec. 10.1.3}$$

El comportamiento del sistema para todo τ se puede deducir pensando a las señales como vectores. La salida entonces sería una suma vectorial donde se supone que la señal original tiene fase nula en cierto instante respecto la fase que tendría la señal demorada.

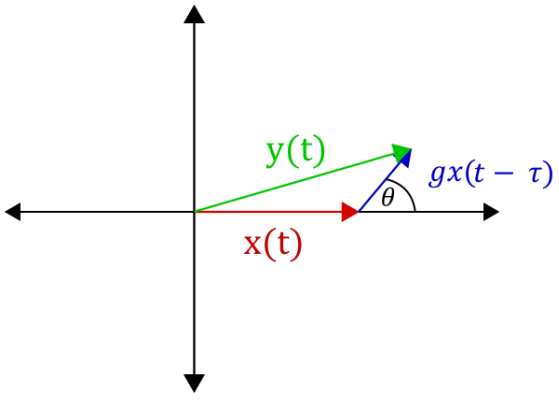


*Figura 10.1.1 – Representación de la Ec. 10.1.1 en forma vectorial con un desfasaje de ϑ grados y un factor de atenuación de g veces.*

En función de τ, se puede expresar la diferencia de fase θ como:

$$\theta = \frac{2\pi}{T} \cdot \tau = 2\pi f \tau$$

De esta manera, el módulo de la suma vectorial se puede establecer como:

$$|y(t)| = \sqrt{[1 + g \cdot \cos(2\pi f \tau)\,]^2 + [g \cdot \sin(2\pi f \tau)\,]^2} \qquad \text{Ec. 10.1.4}$$

En la Figura 10.1.2 se puede apreciar que la ganancia en veces permanece entre 1-g y 1+g. Sin embargo, los picos de refuerzo son mucho más suaves que los picos de atenuación. Esto permite entender la motivación detrás del nombre de este sistema, comúnmente llamado “comb filter” (o filtro peine en castellano). Específicamente, esta variedad se denomina FIR comb filter, ya que un impulso a la entrada genera una respuesta finita.

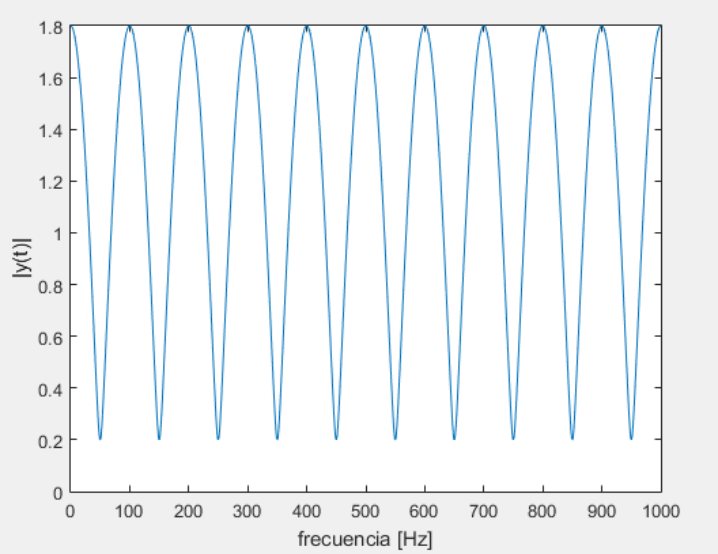


*Figura 10.1.2 – Gráfico de la Ec. 10.1.4 con τ = 0,01 s y g = 0,8.*

Si se desea llevar el sistema a un dominio discreto con una frecuencia de muestreo Sr, se obtienen estas expresiones:

$$y[n] = x[n] + gx[n - M]\,, M = \frac{\tau}{Sr} \qquad \text{Ec. 10.1.5}$$

$$H(z) = 1 + gz^{-M} \qquad \text{Ec. 10.1.6}$$

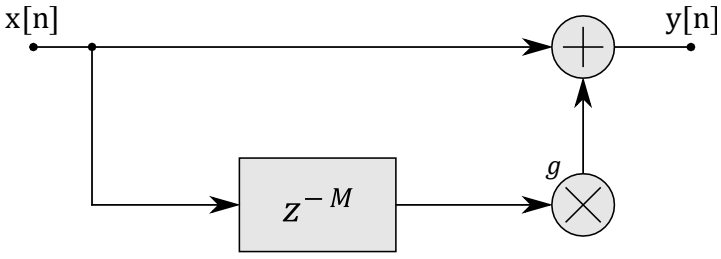


*Figura 10.1.3 – Diagrama en bloques para la implementación del Delay.*

## 10.2 - Echo

El "echo" (o eco en castellano) es una generalización del concepto de delay. Una sola repetición de la señal se considera como un echo puro, pero el concepto también contempla hasta la repetición indefinida de la señal. Del mismo modo que en el delay, hay una motivación presente en la naturaleza detrás de este efecto: al gritar en un ambiente abierto, es común escuchar varias repeticiones de la onda de sonido original.

Un modelo sencillo para representar este fenómeno es el siguiente:

$$y(t) = x(t) + gy(t-\tau)\,, 0 < g \leq 1 \qquad \text{Ec. 10.2.1}$$

$$y[n] = x[n] + gy[n-M]\,, M = \frac{\tau}{Sr} \qquad \text{Ec. 10.2.2}$$

$$H(z) = \frac{1}{1 - gz^{-M}} \qquad \text{Ec. 10.2.3}$$

Si se supone que se deja correr este filtro por un tiempo indeterminado, van a aparecer a la salida infinitas copias de la entrada original, que circulan indefinidamente por el bucle. Sin embargo, cada vez que completan una vuelta, éstas se atenúan en un factor de g y se retrasan τ segundos. En este caso, g no puede adoptar un valor mayor a la unidad por una condición de estabilidad.

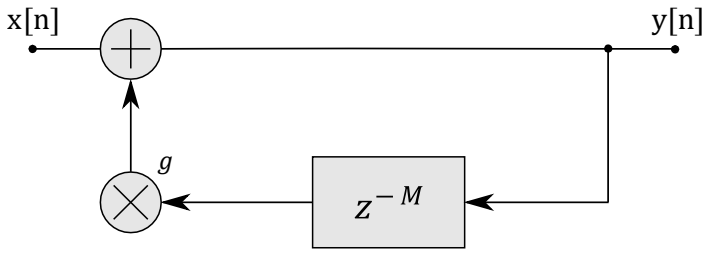


*Figura 10.2.1 – Diagrama en bloques para la implementación del Echo.*

Como comparación con el caso anterior, se puede averiguar qué pasa cuando τ es tal que todas las copias están en fase con la original o, mejor dicho, cuando cumple con la Ec. 10.1.2. Cuando se cumple esto, la amplitud a la salida debería ser la máxima posible. Esta expresión se puede evaluar fácilmente con la famosa identidad de una serie geométrica.

$$|y(t)| = \sum_{n=0}^{\infty} g^n = \frac{1}{1-g}$$

El otro caso extremo es cuando cada copia está en contrafase con la anterior. Esto se da cuando τ cumple la condición establecida por la Ec. 10.1.3 y se obtiene una serie geométrica alternada.

$$|y(t)| = \sum_{n=0}^{\infty} (-1)^n g^n = \sum_{n=0}^{\infty} (-g)^n = \frac{1}{1+g}$$

Si bien los cálculos se hacen más complicados, también es posible obtener el comportamiento del sistema para todo τ en base la representación de señales como vectores.

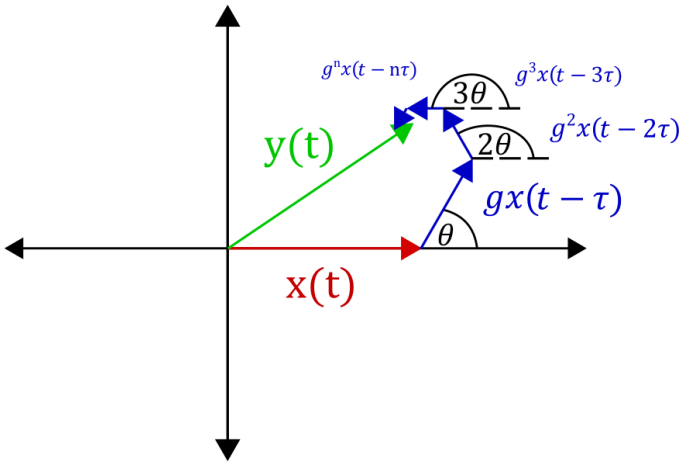


*Figura 10.2.2 - Representación de la Ec. 10.2.1 en forma vectorial con un desfasaje de ϑ grados y un factor de atenuación de g veces.*

Se puede deducir a partir de la Figura 10.2.2 que la suma vectorial corresponde al siguiente cálculo:

$$|y(t)| = \sqrt{\left[\sum_{n=0}^{\infty} g^n \cos(n\theta)\right]^2 + \left[\sum_{n=0}^{\infty} g^n \operatorname{sen}(n\theta)\right]^2}$$

Al resolver las series (ver anexo 16.2 - ), se obtiene la siguiente expresión:

$$|y(t)| = \frac{1}{\sqrt{[1 - g \cdot \cos(2\pi f\tau)]^2 + [g \cdot \operatorname{sen}(2\pi f\tau)]^2}} \qquad \text{Ec. 10.2.4}$$

Esta variante del filtro peine es denominada comúnmente IIR comb filter ya que un impulso en la entrada queda atrapado en el bucle y genera una salida infinita. Como era de esperar, la respuesta se encuentra acotada entre 1/(1+g) y 1/(1-g) pero, a comparación con el FIR comb filter, esta vez, los picos de atenuación con son mucho más suaves que los picos de refuerzo.

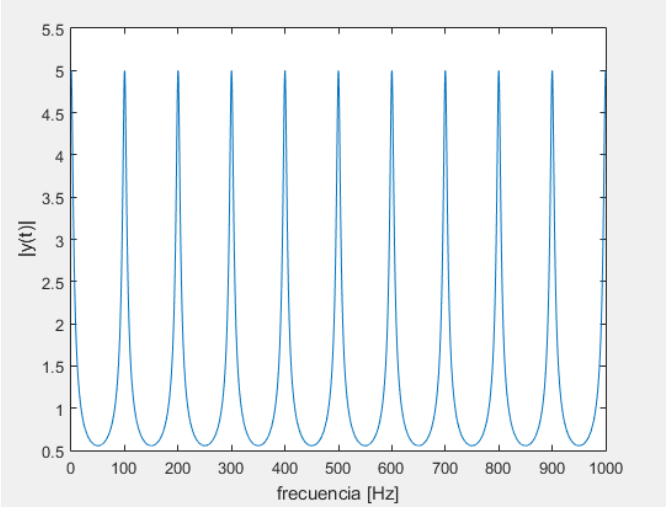


*Figura 10.2.3 - Gráfico de la Ec. 10.2.4 con τ = 0,01 s y g = 0,8.*

A fines prácticos, se puede variar el parámetro g para obtener la cantidad de repeticiones que se necesiten: si se precisan p repeticiones, hay que asegurar que $g^p$ sea un nivel de señal por debajo del umbral audible del oyente.

## 10.3 - Delay + Echo

Estos últimos dos filtros son los bloques básicos detrás de todos los efectos basados en delay. Generalmente, se busca utilizar ambas variedades en el mismo efecto, con lo cual, es de interés llegar a un sistema que los implemente juntos. Esta es la motivación detrás del "universal comb filter" (o filtro peine universal en castellano).

Lo más intuitivo sería aprovechar la linealidad de los sistemas para conectar ambos en serie en una configuración similar a la Figura 10.3.1.

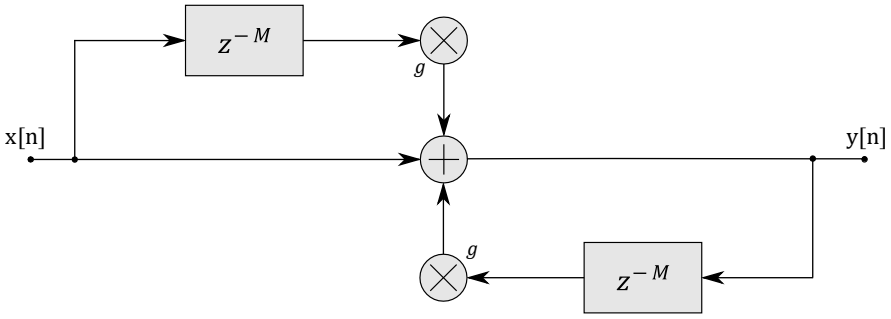


*Figura 10.3.1 - Conexión en serie de los diagramas en bloque del Delay y el Echo.*

La transferencia sería entonces:

$$H(z) = \frac{1 + gz^{-M}}{1 - gz^{-M}} \qquad \text{Ec. 10.3.1}$$

Sin embargo, hay una manera más eficiente de conseguir el mismo sistema ilustrada por la Figura 10.3.2:

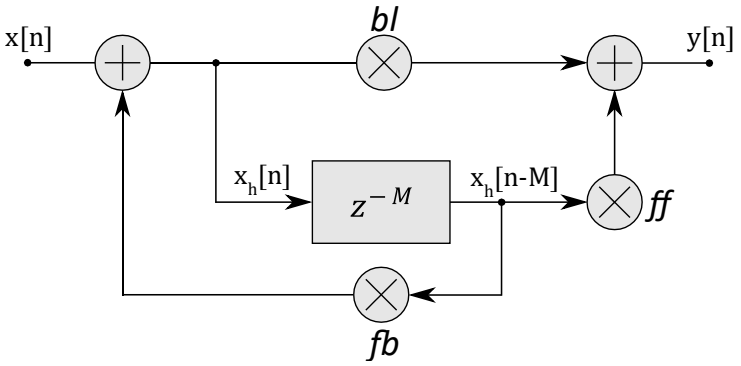


*Figura 10.3.2 – Diagrama en bloques para la implementación de Delay + Echo con un solo buffer de memoria.*

Donde los parámetros son ff (feed-forward o prealimentación), fb (feed-back o realimentación) y bl (blend o mezcla).

Para analizar su comportamiento, se pueden plantear las ecuaciones de los nodos y tratar de llegar a una expresión de su función de transferencia:

$$\begin{cases} y[n] = bl \cdot x_h[n] + ff \cdot x_h[n-M] \\ x_h[n] = x[n] + fb \cdot x_h[n-M] \end{cases} \Rightarrow \begin{cases} H(z) = bl \cdot \dfrac{X_h(z)}{X(z)} + ff \cdot z^{-M} \cdot \dfrac{X_h(z)}{X(z)} \\ \dfrac{X_h(z)}{X(z)} = 1 + fb \cdot z^{-M} \cdot \dfrac{X_h(z)}{X(z)} \end{cases}$$

Despejando y reemplazando en la primera ecuación se obtiene lo siguiente:

$$H(z) = \frac{bl + ff \cdot z^{-M}}{1 - fb \cdot z^{-M}}$$

Que se corresponde con la Ec. 10.3.1 salvo por un grado de libertad añadido por el parámetro bl. Observando la expresión, se puede concluir que ff controla al filtro FIR y el parámetro fb controla al filtro IIR.

Esta construcción del filtro es más eficiente que la primera en el sentido de que requiere la mitad memoria en su implementación (solamente usa un bloque de retardo en vez de dos) y, además, requiere levemente menos operaciones por muestra para la misma aplicación. La primera ventaja es fundamental ya que cuando se usa delay o echo, se busca una demora del orden del segundo y un buffer con un segundo de muestras debe ser en tamaño como la frecuencia de muestreo. Debido a esto, estos efectos tienen una carga importante de memoria en su aplicación.

Finalmente, el modelo utilizado en este trabajo para lograr el efecto combinado de delay/echo se muestra en la Figura 10.3.3:

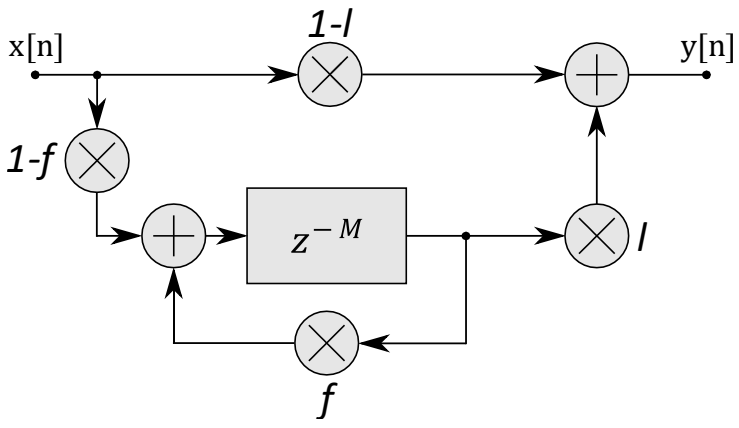


*Figura 10.3.3 – Diagrama en bloques para la implementación de Delay + Echo utilizado en el trabajo. Los parámetros cumplen con las condiciones 0 ≤ l ≤ 1, 0 ≤ f ≤ 1.*

Un planteo similar al del filtro anterior nos lleva a la expresión de la transferencia:

$$H(z) = \frac{(1-l) + [l(1-f) - f(1-l)] \cdot z^{-M}}{1 - f \cdot z^{-M}}$$

La motivación detrás del diseño de este filtro fue la de evitar un overflow en las operaciones de los nodos. Para cualquier valor del (level o nivel) y f (feed-back o realimentación), se obtienen proporciones correctas en cada suma para no excederse del rango dinámico del tipo de variable que corresponda en el programa.

## 10.4 - Implementación en Código: Delay + Echo

Uno de los mayores obstáculos a la hora de crear un algoritmo de un efecto basado en delay en tiempo real es el de lograr el bloque de retraso temporal de una manera eficiente.

Como ya se discutió, primero se necesita una gran cantidad de memoria RAM para poder tener en todo instante un arreglo de por lo menos M posiciones con instancias anteriores a la entrada del bloque (especialmente en el caso del efecto de delay/echo). Segundo, hay que hallar una buena manera de administrar esta entidad; se debe tener una lectura y escritura eficiente para lograr que funcione en tiempo real.

La manera más intuitiva sería intentar que la posición 0 del arreglo tenga siempre la posición más reciente de la entrada, con lo cual siempre se leería la posición M-1 para establecer la salida. Sin embargo, este método tiene un gran problema en la escritura ya que cada vez que entra un nuevo elemento, hay que trasladar todas las posiciones del arreglo a la siguiente para permitir la entrada de la nueva muestra en la posición 0 y el descarte de la muestra en la posición M. La cantidad de operaciones a realizar es una en la lectura, pero M en la escritura.

El procedimiento utilizado en este trabajo se basa en una escritura en modo de buffer circular; si la muestra anterior se guardó en la posición i-1 del buffer, la actual se guardará en la posición i y, cuando se llegue a la posición M-1, la muestra siguiente se guardará en la posición 0. El gran ahorro se produce gracias a que cada vez que se escribe una muestra nueva, se pisa el valor de la muestra a descartar, manteniendo todo actualizado con una sola operación. La cantidad de operaciones ahora es de una en la escritura y una en la lectura por cada ciclo.

Este método, sin embargo, presenta una complicación al tratar de obtener la muestra demorada. Si se desea un delay de k muestras, hay dos opciones de búsqueda en el arreglo ya que n-k puede encontrarse bajo el índice i-k o M+i-k dependiendo del caso. En varios procesadores se puede realizar esta comprobación por hardware mediante un mecanismo conocido como indexación modular. Este nombre alude al hecho de que hacer una operación de resto al dividir por M siempre va a dar el índice correcto para evaluar dentro del arreglo.

Dentro del procesador Cortex M7, sin embargo, no es posible declarar este tipo de estructuras y la comprobación de índices se debe realizar por software. La operación módulo no es conveniente ya que lleva a cabo una división. Para evitarla, se resetea el puntero a la posición i cada vez que desborda el arreglo y así solamente es preciso realizar una operación condicional.

| Índice | 0 | 1 | … | i-1 | i | i+1 | … | M-2 | M-1 |
|---|---|---|---|---|---|---|---|---|---|
| Muestra | n-i | n-(i-1) | | n-1 | n | n-(M-1) | | n-(i+2) | n-(i+1) |

*Figura 10.4.1 – Estructura del buffer circular utilizado para simular bloques de demora.*

Un simple código en lenguaje C para dar un ejemplo sería el siguiente:

```c
//Demora.c

//N > M
#define N 10
#define M 8

int salida;
int buffer[N] = {0};
int i = 0;

//demora de M muestras
int demora(int entrada)
{
  if(i == N)
    i = 0;
  buffer[i] = entrada;
  if(i >= M)
    salida = buffer[i-M];
  else
    salida = buffer[N+i-M];
  i++;
  return salida;
}
```

Una vez establecido el funcionamiento del bloque, el resto del código consiste simplemente en copiar las ecuaciones del sistema en bloques aprovechando esta estructura:

```c
//Delay.c

//N > M
#define N 10
#define M 8

int salida;
int buffer[N] = {0};
int i = 0;
float l = 0.5, f = 0.5;

//delay de M muestras
int delay(int entrada)
{
  if(i == N)
    i = 0;

  if(i >= M)
  {
    buffer[i] = (1-f) * entrada + f * buffer[i-M];
    salida = (1-l) * entrada + l * buffer[i-M];
  }
  else
  {
    buffer[i] = (1-f) * entrada + f * buffer[N+i-M];
    salida = (1-l) * entrada + l * buffer[N+i-M];
  }

  i++;
  return salida;
}
```

## 10.5 - Vibrato

El siguiente efecto basado en delay en orden de complejidad es el vibrato. Tiene similitudes con el efecto Doppler; cuando las ondas sonoras van acercándose y viajando cada vez por un camino más corto, se produce una contracción y el oído interpreta esto como un aumento en la frecuencia de este sonido y viceversa.

Se busca replicar este fenómeno a través de una modulación de la cantidad de muestras de delay. Gracias a esto, surge por primera vez la necesidad de introducir un LFO en los efectos basados en delay.

$$y(t) = x\big(t - \tau(t)\big) \qquad \text{Ec. 10.5.1}$$

$$y[n] = x[n - M(n)], M = \frac{\tau}{Sr} \qquad \text{Ec. 10.5.2}$$

$$H(z) = z^{-M(n)} \qquad \text{Ec. 10.5.3}$$

Algo importante para notar es que la muestra actual no se reproduce en lo absoluto, por lo tanto, está es una excepción en la cual no hay ningún filtro peine en juego. También, el valor medio del LFO debe ser tal que el retardo promedio entrada/salida no sea notable al oyente.

Una vez que alguien intenta implementar el vibrato sin más información que ésta generalmente se encuentra con un problema: se escucha un ruido constante indeseado a la salida introducido por el algoritmo. Esto se debe a que el LFO puede devolver valores continuos que luego se deben mapear a un valor discreto que es la cantidad de muestras de delay instantáneas. Por esto, es imperativo realizar algún tipo de interpolación sobre los valores del arreglo de posiciones previas. A través de estos mecanismos, se puede lograr una demora de, por ejemplo: 5,8 muestras.

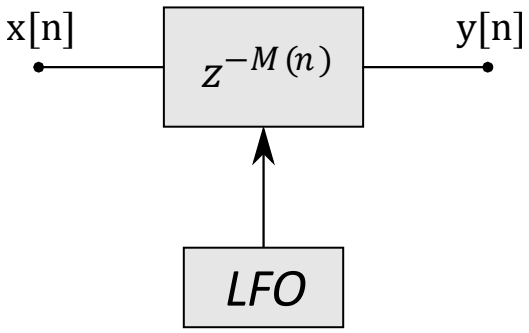


*Figura 10.5.1 – Diagrama en bloques para la implementación del Vibrato.*

Si bien hay interpolaciones más sofisticadas de mayor orden, en este trabajo se optó por utilizar una interpolación lineal, que es básicamente hacer un promedio ponderado de las dos muestras más cercanas al valor deseado. Un ejemplo sencillo sería: si se quiere un delay de 5,8 muestras, se calcula como:

$$\mathrm{x[n-5.8]} \approx 0.8 * \mathrm{x[n-6]} + 0.2 * \mathrm{x[n-5]}$$

Generalizando la expresión, se obtiene:

$$\mathrm{x[n-(M+frac)]} \approx \mathrm{frac} * \mathrm{x[n-(M+1)]} + (1-\mathrm{frac}) * \mathrm{x[n-M]}, 0 \leq \mathrm{frac} < 1$$

Se eligió esta interpolación básica ya que es la más rápida a nivel de complejidad de cálculo y produce resultados más que aceptables.

## 10.6 - Implementación en Código: Vibrato

A la hora de implementar el código del vibrato, se debe tener en cuenta que la interpolación produce una complejidad al aplicarse en el método de escritura y lectura del buffer de delay presentado anteriormente. Debido a que se precisan no una, sino dos posiciones previas del arreglo, ahora puede haber tres casos al leer. El primero es que ambas muestras se encuentren

en una parte previa del arreglo, con lo cual, habría que leer los índices i-M e i-(M+1). La segunda es que ambas se encuentren en una parte posterior, allí habría que buscar las posiciones N+i-M e N+i-(M+1). Finalmente, puede ocurrir el caso más raro que es cuando una muestra es la primera (o muestra 0) y la anterior es la muestra N-1.

Un código ejemplo para este efecto sería el siguiente:

```
1  //Vibrato.c
2
3  #define N 50
4
5  #define TRINAGULAR 0
6  #define SINUSOIDAL 1
7
8  int salida;
9  int buffer[N] = {0};
10 int M, i = 0;
11 int modulacion = TRIANGULAR;
12 float delay, frac;
13
14 int vibrato(int entrada)
15 {
16   //El LFO devuelve la cantidad de muestras de delay instantáneas
17   //hay que interpolar ya que puede ser un valor decimal
18   delay = LFO(modulacion);
19   //Se busca tener una expresión de la forma delay = M + frac
20   //para poder aplicar el algoritmo de interpolación lineal
21   M = floor(delay);
22   frac = delay - M;
23   if(i == N)
24     i = 0;
25   buffer[i] = entrada;
26   if(i >= M+1)
27     salida = (1-frac) * buffer[i-M] + frac * buffer[i-(M+1)];
28   else if(i < M)
29     salida = (1-frac) * buffer[N+i-M] + frac * buffer[N+i-(M+1)];
30   else
31     salida = (1-frac) * buffer[0] + frac * buffer[N-1];
32   i++;
33   return salida;
34 }
```

## 10.7 - Chorus

Como su nombre lo indica, el “chorus” (o coro, en castellano), trata de hacer que varios sonidos con ligeramente distintos tonos sean percibidos como uno solo, tal como ocurre en un coro o en un ensamble de instrumentos de cuerda. Una técnica sencilla para implementar este efecto en tiempo real es la de usar uno o más vibratos sumados a la señal original:

$$y(t) = x(t) + \sum_{i=1}^{N} x\left(t - \tau_i(t)\right) \qquad \text{Ec. 10.7.1}$$

$$y[n] = x[n] + \sum_{i=1}^{N} x[n - M_i(n)], M = \frac{\tau}{Sr} \qquad \text{Ec. 10.7.2}$$

$$H(z) = 1 + \sum_{i=1}^{N} z^{-M_i(n)} \qquad \text{Ec. 10.7.3}$$

Donde cada uno de los retardos con controlados por LFO.

Si se desea adaptar este sistema para evitar la posibilidad de saturación a la salida, se puede escalar la entrada por un factor menor o igual a 1/(N+1). Otra alternativa que brinda más posibilidades, pero también más tiempo de cómputo, es la de hacer ponderaciones en cada una de las sumas.

Para analizar lo que ocurre en el dominio de la frecuencia, se puede suponer primero que se utiliza un solo vibrato sumado a la entrada. En ese caso, se tendría un filtro peine FIR cuya demora es controlada por un LFO. Al variar el retardo, cambia la distancia entre los picos de refuerzo y los de atenuación, con lo cual, las frecuencias reforzadas y atenuadas son distintas en cada instante de tiempo. El hecho de paralelizar varios de estos filtros hace que el efecto sea mucho más apreciable.

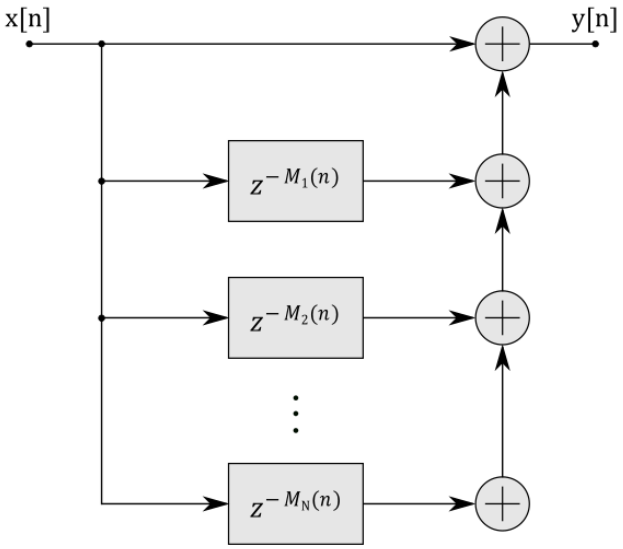


*Figura 10.7.1 – Diagrama en bloques para la implementación del Chours (no se muestran los LFO conectados a los retardos para simplificar el diagrama).*

## 10.8 - Implementación en Código: Chorus

Partiendo del código del vibrato, sólo son necesarias unas pocas modificaciones, para llegar a tener un chorus. Lo más importante es construir un ciclo de iteración para obtener tantos vibratos como haga falta y sumar la entrada a la salida.

En principio, haría falta llamar al LFO por cada uno de los vibratos que se deseen, sin embargo, se adoptó un criterio para llegar a utilizar solamente uno. El oscilador 1 genera una determinada forma de onda y se suma un valor de continua u “offset” distinto para generar cada uno de los otros N-1 osciladores.

Finalmente, se calcula experimentalmente un factor de escala para evitar que la salida no sature bajo condiciones normales de trabajo.

A continuación, un código a modo ejemplo con tres osciladores:

```
//Chorus.c

#define N 200

#define TRINAGULAR 0
#define SINUSOIDAL 1

int salida;
int buffer[N] = {0};
int vibrato;
int M, j, i = 0;
int modulacion = SINUSOIDAL;
int offset = 50;
float delay, frac, level = 0.5;

int chorus(int entrada)
{
  delay = LFO(modulacion);
  if(i == N)
    i = 0;
  buffer[i] = entrada;
  salida = entrada;
  for(j = 0 ; j < 3 ; j++)
  {
    M = floor(delay + j * offset);
    frac = delay + j * offset - M;
    if(i >= M+1)
      vibrato = (1-frac) * buffer[i-M] + frac * buffer[i-(M+1)];
    else if(i < M)
      vibrato = (1-frac) * buffer[N+i-M] + frac * buffer[N+i-(M+1)];
    else
      vibrato = (1-frac) * buffer[0] + frac * buffer[N-1];
    salida += level * vibrato;
  }
  salida = scale * salida;
  i++;
  return salida;
}
```

## 10.9 - Flanger

El flanger nació como una técnica de producción musical. En las épocas en las que se usaban grabadores magnéticos, el ingeniero usaría dos cintas para lograr el efecto. Al presionar su dedo suavemente sobre la cinta en una zona denominada “flange”, un grabador se retrasaría respecto del otro. Posteriormente, corregiría este retardo presionando sobre el otro grabador. La repetición de este proceso generaría una oscilación en el tiempo con un sonido que generalmente se describe como un “jet”.

De la misma manera en la que se utilizan “vibratos” en paralelo a la entrada para lograr el efecto de chorus, el flanger se puede lograr agregando filtros peine IIR controlados por un oscilador en paralelo a la entrada. En particular, aquí se utilizó una variante del filtro que introduce un retardo en la entrada.

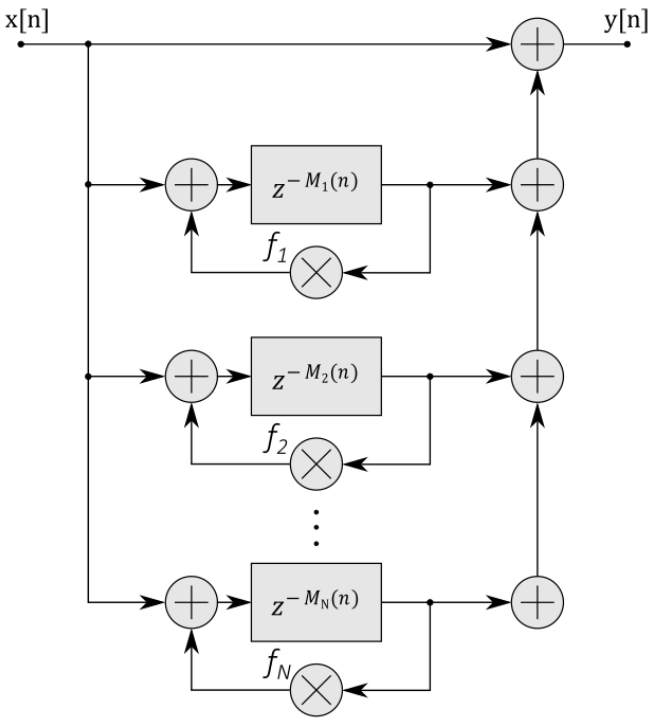


*Figura 10.9.1 – Diagrama en bloques para la implementación del Flanger.*

Cada uno de los filtros IIR presenta la siguiente función de transferencia:

$$H_i(z) = \frac{z^{-M_i(n)}}{1 + f_i \cdot z^{-M_i(n)}}$$

Por lo tanto, la transferencia de la totalidad del efecto en la manera en la que se implementó es la siguiente:

$$H(z) = 1 + \sum_{i=1}^{N} \frac{z^{-M_i(n)}}{1 + f_i \cdot z^{-M_i(n)}} \qquad \text{Ec. 10.9.1}$$

Donde, para que el efecto sea estable y distinguible:

$$-1 < f_i < 0, |f_i| \approx 1$$

Una diferencia fundamental que este efecto tiene con el chorus es que este último utiliza retardos del orden de las decenas de milisegundos, mientras que el retardo del flanger suele estar en el orden de los milisegundos. Otra característica muy buscada en un flanger es que el LFO llegue a tener un barrido que se aproxime lo más posible a la muestra actual en su valor de demora más pequeño.

## 10.10 - Implementación en Código: Flanger

Nuevamente, lo más sencillo es partir del código del chorus. Se debe crean tantos buffers como filtros peine IIR en paralelo se desee ya que cada uno tiene su rama de realimentación que depende solamente de sí mismo. Para lograr una muestra de demora al inyectar el feedback en la entrada de cada buffer, sencillamente se vuelve a utilizar las variables correspondientes antes de pisarlas ya que contienen la información de la última vez que se llamó a la función.

Se tomaron dos simplificaciones para poder optimizar levemente el cálculo. La primera fue la de evitar la saturación a la salida por medio de un factor de escala en serie al sistema calculado experimentalmente (al igual que el chorus). La segunda fue la de considerar que todos los valores de realimentación o "feedback" son los mismos de la siguiente manera:

$$f_1 = f_2 = \cdots = f_N = f$$

En el código ejemplo, se utilizan cinco filtros peine IIR en paralelo a la entrada:

```
//Flanger.c

#define N 80

#define TRINAGULAR 0
#define SINUSOIDAL 1

int salida;
int buffer[N][5] = {0};
int vibrato[5] = {0};
int M, j, i = 0;
int modulacion = SINUSOIDAL;
int offset = 10;
float delay, frac, feedback = -0.93;
float scale = 0.5, level = 0.5;

int flanger(int entrada)
{
  delay = LFO(modulacion);
  if(i == N)
    i = 0;
  salida = entrada;
  for(j = 0 ; j < 5 ; j++)
  {
    buffer[i][j] = entrada + feedback * vibrato[j];
    M = floor(delay + j * offset);
    frac = delay + j * offset - M;
    if(i >= M+1)
      vibrato[j] = (1-frac) * buffer[i-M][j] + frac * buffer[i-(M+1)][j];
    else if(i < M)
      vibrato[j] = (1-frac) * buffer[N+i-M][j] + frac * buffer[N+i-(M+1)][j];
    else
      vibrato[j] = (1-frac) * buffer[0][j] + frac * buffer[N-1][j];
    salida += level * vibrato[j];
  }
  salida = scale * salida;
  i++;
  return salida;
}
```

## 10.11 - Reverb

El efecto de "reverb" (o reverberación en castellano) consiste en la simulación de la acústica de un espacio concreto, como podría ser una iglesia o un salón de concierto. Es uno de los efectos más complejos a nivel de cómputo y a nivel teórico.

A diferencia de los efectos anteriores, la manera más sencilla e intuitiva de aplicarlo es mediante un filtro FIR. De esta manera, solamente hace falta medir la respuesta impulsiva en el espacio deseado y realizar una convolución con la entrada para obtener la salida. El problema entonces se reduce a encontrar esta respuesta impulsiva.

El procedimiento más conocido involucra utilizar un parlante y un micrófono en dos lugares a elección dentro de la habitación en cuestión. Se alcanzan mejores resultados cuando el parlante es excitado mediante una secuencia pseudo aleatoria de longitud máxima (también llamada M-sequence). La respuesta impulsiva se aproxima entonces como la transferencia medida

multiplicada por la autocorrelación de la señal pseudo aleatoria (que es conocida). Para evitar aliasing, el período de la excitación debe ser mayor que el tiempo de decaimiento de la reverberación. Finalmente, para obtener un resultado más representativo, se repite este proceso las veces que hagan falta y se hace un promedio de los resultados.

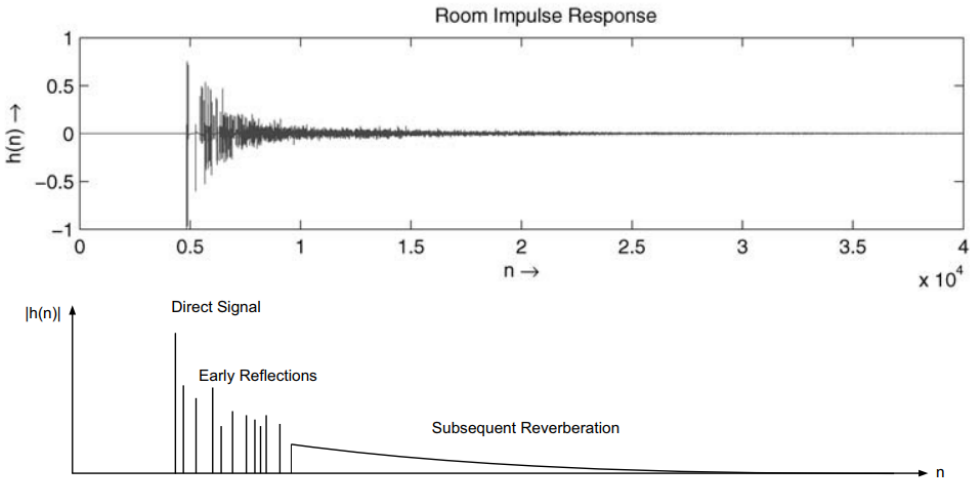


*Figura 10.11.1 – Ejemplo de una medición real de la respuesta impulsiva de una habitación (arriba) y la modelización teórica extrapolada de los resultados (abajo).*

Si se observan los resultados generados por estos experimentos, se pueden distinguir tres fases que siempre están presentes. Primero, se escucha la señal directa casi sin retardo, luego, aparecen las primeras reflexiones, las cuales son copias de la señal directa pero desfasadas y atenuadas por los rebotes. Finalmente, a medida que la densidad de rebotes aumenta, comienza la reverberación subsecuente, donde se puede apreciar una envolvente que decrece de manera exponencial. El tiempo de reverberación se define como el tiempo en que tarda la respuesta impulsiva en alcanzar los -60 dB.

Está técnica ha sido muy utilizada y ha ayudado mucho a la comprensión de este fenómeno. Sin embargo, también tiene sus complicaciones; si se desean cambiar los parámetros del efecto (como el tamaño de la habitación o la intensidad de los rebotes), se deben realizar mediciones y guardar una respuesta impulsiva en memoria para cada uno de los estados. Debido a estos y a otros motivos, fue de interés estudiar alguna manera de simular este fenómeno de manera paramétrica. Esto desembocó en uno de los modelos más populares para realizar el efecto: el reverberador de Schroeder.

Este modelo consiste en la acumulación en paralelo de filtros peine IIR seguida por la concatenación en serie de filtros pasa todo modificados para manejar demoras arbitrarias. Estos dos tipos de filtros se suelen preferir en parte por su cualidad de presentar envolventes decrecientes de manera exponencial a la salida. En particular, este trabajo aplica una variante especial desarrollada por Jezar at Dreampoint en su código de C++ de dominio público llamado “freeverb”. En total, se procesan 8 filtros peine IIR y 4 filtros pasa todo.

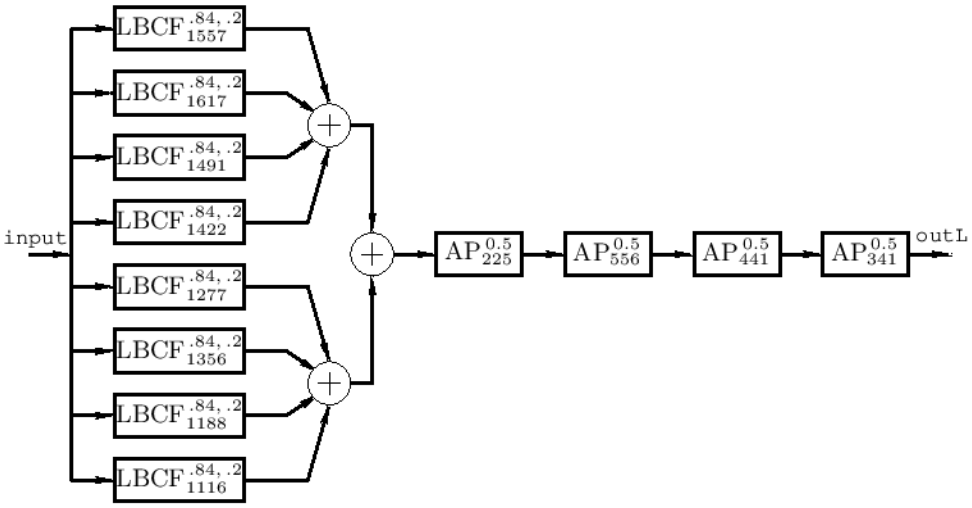


*Figura 10.11.2 – Diagrama en bloques del algoritmo Freeverb junto con los doce filtros individuales a utilizar y los valores por defecto recomendados para sus parámetros.*

El filtro peine IIR presenta algunas modificaciones comparado con el previamente expuesto. La línea de demora se mueve para retrasar también a la entrada y, más destacablemente, se utiliza un filtro pasa bajos de primer orden en el lazo de realimentación negativa. Esto último logra que el tiempo de reverberación sea dependiente de la frecuencia, generalmente se considera que esto es más representativo de lo que ocurre en un rebote real en términos de fidelidad sonora.

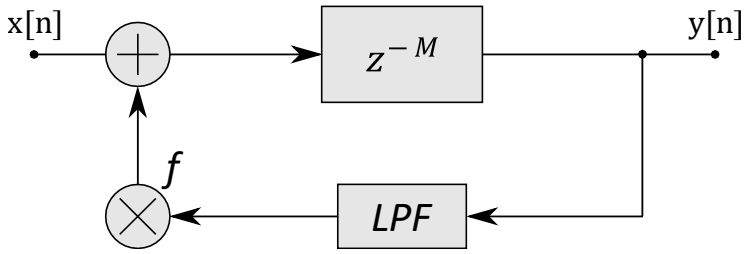


*Figura 10.11.3 – Diagrama en bloques del filtro LBCF utilizado en el algoritmo Freeverb.*

Partiendo de la Figura 10.11.3, la función de transferencia se puede definir como:

$$LBCF_M^{f,d} \triangleq \frac{z^{-M}}{1 - f\frac{1-d}{1-dz^{-1}}z^{-M}} \qquad \text{Ec. 10.11.1}$$

Donde M es la cantidad de muestras de retardo, f es la proporción de salida que se realimenta en la entrada y d es el parámetro que controla al filtro pasabajos. En este modelo, f ajusta el tiempo de reverberación, con lo cual, está relacionado con el tamaño de la habitación (llamado comúnmente room size). Por otro lado, d indica qué tanto se atenúan las altas frecuencias al producirse el rebote (llamado comúnmente damping).

Luego de acumular el resultado de 8 de estos filtros, se pasa a concatenar en serie los filtros pasa todo (ver Figura 10.11.4) que define el programa. Una sencilla comparación de la función de transferencia revela que este no se trata de un filtro pasa todo, sino que es una aproximación.

$$AP_M^g \triangleq \frac{-g + z^{-M}}{1 - gz^{-M}} \approx \frac{-1 + (g+1)z^{-M}}{1 - gz^{-M}} \qquad \text{Ec. 10.11.2}$$

Para poder ser un pasa todo, se debería cumplir que los numeradores sean proporcionales, entonces:

$$\begin{cases} -g = -k \\ 1 = k(g+1) \end{cases} \Longrightarrow g^2 + g - 1 = 0 \Longrightarrow g = \frac{\sqrt{5}-1}{2} \approx 0{,}618$$

Esto significa que g debe tener el mismo valor que el recíproco del número áureo, pero el valor por defecto utilizado en el código es de 0,5.

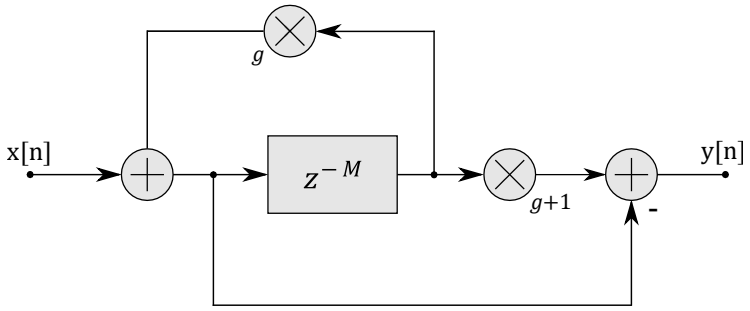


*Figura 10.11.4 - Diagrama en bloques del filtro AP utilizado en el algoritmo Freeverb.*

Finalmente, para completar el efecto, se debe sumar la entrada actual a la salida junto con el efecto procesado en las proporciones adecuadas.

## 10.12 - Implementación en Código: Reverb

El este código, cada uno de los doce filtros precisa tener un buffer definido para ejecutar el bloque de delay, debido a esto, el consumo de memoria es bastante alto. Además, procesar doce filtros por muestra consume muchos ciclos, lo que hace que este sea uno de los efectos que más carga le exige al DSP.

Para poder tener un código más sencillo de leer y de manipular, se decidió hacer una función especial para los dos casos de filtros. El ciclo principal llama a los doce filtros en orden, acumulando los filtros FBCF (multiplicados por un octavo para tratar de evitar saturación) y concatenando los filtros AP. Finalmente, hace una ponderación de la entrada junto con el cálculo obtenido para escribir la salida.

Este es un ejemplo del código utilizado:

```
//Reverb.c

int salida ;
int FBCF0_line[1617] = {0}, FBCF1_line[1617] = {0};
int FBCF2_line[1617] = {0}, FBCF3_line[1617] = {0};
int FBCF4_line[1617] = {0}, FBCF5_line[1617] = {0};
int FBCF6_line[1617] = {0}, FBCF7_line[1617] = {0};
float lpo[8] = {0};
int AP0_line[225] = {0}, AP1_line[556] = {0};
int AP2_line[441] = {0}, AP3_line[341] = {0};

float dry = 0.5, wet = 0.5, d = 0.2, f = 0.82, g = 0.5;
int FBCF_cont = 0, AP0_cont = 0, AP1_cont = 0, AP2_cont = 0, AP3_cont = 0;

int reverb (int entrada)
{
    if(FBCF_cont == 1617)
        FBCF_cont = 0;
    if(AP0_cont == 225)
        AP0_cont = 0;
    if(AP1_cont == 556)
        AP1_cont = 0;
    if(AP2_cont == 441)
        AP2_cont = 0;
    if(AP3_cont == 341)
        AP3_cont = 0;

    salida = 0.125 * FBCF_filter(entrada,FBCF0_line,1556,FBCF_cont,0);
    salida += 0.125 * FBCF_filter(entrada,FBCF1_line,1616,FBCF_cont,1);
    salida += 0.125 * FBCF_filter(entrada,FBCF2_line,1490,FBCF_cont,2);
    salida += 0.125 * FBCF_filter(entrada,FBCF3_line,1421,FBCF_cont,3);
    salida += 0.125 * FBCF_filter(entrada,FBCF4_line,1276,FBCF_cont,4);
    salida += 0.125 * FBCF_filter(entrada,FBCF5_line,1355,FBCF_cont,5);
    salida += 0.125 * FBCF_filter(entrada,FBCF6_line,1187,FBCF_cont,6);
    salida += 0.125 * FBCF_filter(entrada,FBCF7_line,1115,FBCF_cont,7);

    salida = AP_filter(salida,AP0_line,224,AP0_cont);
    salida = AP_filter(salida,AP1_line,555,AP1_cont);
    salida = AP_filter(salida,AP2_line,440,AP2_cont);
    salida = AP_filter(salida,AP3_line,340,AP3_cont);

    salida = dry * entrada + wet * salida;

    FBCF_cont++;
    AP0_cont++;
    AP1_cont++;
    AP2_cont++;
    AP3_cont++;

    return salida;
}

int FBCF_filter(int in, int[] line, int delay, int cont, int inst)
{
    line[cont] = in + f * lpo[inst];
    if(cont >= delay)
    {
        lpo[inst] = (1-d) * line[cont-delay] + d * lpo[inst];
        return line[cont-delay];
    }
    else
    {
        lpo[inst] = (1-d) * line[1617+cont-delay] + d * lpo[inst];
        return line[1617+cont-delay];
    }
}

int AP_filter(int in, int[] line, int delay, int cont)
{
    if(cont >= delay)
    {
        line[cont] = in + g * line[cont-delay];
        return (1+g) * line[cont-delay] - line[cont];
    }
    else
    {
        line[cont] = in + g * line[cont+1];
        return (1+g) * line[cont+1] - line[cont];
    }
}
```

## 10.13 - Octavador

El octavador, tal como su nombre lo dice, es el efecto que logra reproducir la entrada desplazada una cantidad entera de octavas en la salida. Es un caso específico del "pitch shifter" o desplazador de tono, el cual hace cambios arbitrarios en la frecuencia de la entrada. En este trabajo se consideró solamente el caso de lograr una octava más grave.

Existen diversas maneras de llevar a cabo esto. En el mundo analógico, lo más común es integrar el efecto con un procesamiento de distorsión no lineal. La señal de entrada se pasa con un comparador calibrado a su valor medio, esto genera una señal cuadrada del mismo período. Posteriormente, mediante flip-flops, se divide en dos la frecuencia y se suaviza ligeramente con un filtro pasabajos para evitar que se generen frecuencias no deseadas. Sin embargo, lo que se busca en este caso es tener una octava limpia sin ningún otro efecto así que este método no es satisfactorio.

Si no se tuviese la restricción de trabajar en tiempo real, se podría reproducir las muestras a la mitad de la frecuencia con la que se leen. Sin embargo, esto produce desbordamiento de la memoria y hace que la reproducción sea también más lenta en el dominio del tiempo. El método utilizado en este trabajo se inspira en este último y logra resolver estos dos problemas.

Primero se escribe la señal de entrada en un buffer de tamaño N, éste se lee con un puntero "A" que aumenta a la mitad de la frecuencia que el de escritura "W". Eventualmente la diferencia entre estos punteros llega a ser igual a M y el buffer desborda, en este entonces, el puntero "A" se iguala al puntero "W" y el ciclo se repite. Esto soluciona ambos problemas, ya que, debido a la actualización del puntero, se puede aplicar en tiempo real y la octava no se escucha distorsionada en el tiempo de una manera perceptible. Pero, por otro lado, aparecen otros dos problemas; se escucha un sonido de conmutación incómodo por cada vez que se actualiza el puntero y se introduce un retardo entrada salida de N/2 muestras.

La solución yace en introducir un nuevo puntero de lectura "B" que esté siempre desfasado N/2 muestras respecto de "A". La salida se calcula como una ponderación de la lectura en ambos punteros de tal manera que, en la transición, el volumen asignado al puntero sea nulo y el volumen sea máximo al encontrarse N/2 muestras desfasado respecto del puntero "W". Esta técnica es llamada "cross-fading" en la bibliografía, haciendo alusión a el hecho de que el volumen a la salida se mantiene contante a medida que se aumentan y disminuyen los volúmenes asignados a los punteros de lectura.

El mapeo de retardos a volúmenes es logrado en este trabajo mediante una sencilla interpolación lineal. Se supone que el arreglo tiene una cantidad N impar de posiciones para simplificar. Los valores de retardo 0 y N-1 deben tener volumen 0 y, en (N-1)/2, se debe llegar a tener volumen unitario buscando siempre una relación lineal, esto resulta en una función partida:

$$\begin{cases} vol = \dfrac{X}{\dfrac{N-1}{2}} \ , X \leq \dfrac{N-1}{2} \\ vol = \dfrac{(N-1)-X}{\dfrac{N-1}{2}} \ , X > \dfrac{N-1}{2} \end{cases} \qquad \text{Ec. 10.13.1}$$

Donde X es el retardo instantáneo.

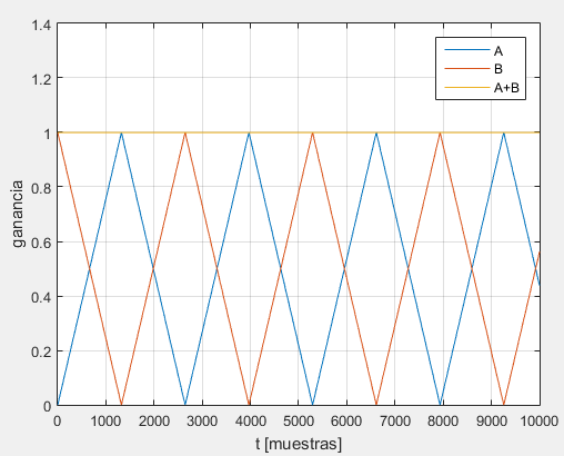


*Figura 10.13.1 – Gráfico ejemplo de la Ec. 10.13.1 aplicada a los punteros en contrafase A y B. Se asume un tamaño de buffer N de 2645 muestras.*

Si bien las transiciones no se escuchan, sigue siendo de importancia la cantidad de actualizaciones de puntero por segundo. Mientras más alta sea, peor será la calidad de la octava. Esto nos lleva a una solución de compromiso, ya que el tamaño de N debe ser lo más grande posible para mejorar la calidad del efecto, pero también debe ser lo más pequeño posible para lograr que el retardo de N/2 muestras no sea perceptible. Un valor comúnmente utilizado que logra buenos resultados es el de 30 ms de muestras.

Otra cosa que es de interés es escuchar la señal original junto con la octava más grave, para hacer esto, se considera el retardo de N/2 muestras y se lee el buffer siempre a esa distancia del puntero “W” para sumarlo al efecto.

Es importante destacar que está técnica de desplazamiento no funciona del todo bien con señales de voz ya que éstas contienen información de tono generado por las cuerdas vocales, pero también de formante que surge de la resonancia del tracto vocal. Esta última es distorsionada de tal manera que las palabras pierden su sonido natural. Para lidiar con esto, existen técnicas más sofisticadas como el filtrado mediante modelización física (physical modeling) o el solapamiento/adición sincrónica al tono (PSOLA). Sin embargo, no se aplicaron en este trabajo debido a que tienen una demanda muy alta de procesamiento, especialmente en tiempo real.

## 10.14 - Implementación en Código: Octavador

Primero se define un buffer de N posiciones (2645 en este caso). El índice de escritura W se incrementa todos los ciclos para ir llenando el buffer a medida que llegan nuevas entradas. Los índices de lectura A y B comienzan en 0 y N/2 respectivamente para poder estar siempre en contrafase. Estos últimos se incrementan a la mitad de frecuencia de la que se actualiza W, esto permite dos posibilidades. Se podría incrementar los contadores ciclo por medio por una unidad o se podría incrementarlos por 0.5 cada ciclo aplicando interpolación en el buffer. Últimamente, se optó por la primera opción ya que es más rápida y la diferencia de calidad de sonido no es notable.

```c
//Octavador.c

#define N 2645

int salida = 0;

int buffer[N] = {0};
float vol = 0;
int W = 0, A = 0, B = N/2;
int flag = 1;

int octavador(int entrada)
{
  //actualización de índices
    if(W >= N)
        W = 0;
    if(A >= N)
        A = A - N;
    if(B >= N)
        B = B - N;

    buffer[W] = entrada;

  //cálculo de volúmen relativo
    if(A <= (N-1)/2)
        vol = A/((N-1)/2.0);
    else
        vol = ((N-1)-A)/((N-1)/2.0);

  //cálculo de la salida
  if (W >= A)
      salida = vol * buffer[W-A];
  else
    salida = vol * buffer[N+W-A];
  if (W >= B)
    salida += (1-vol) * buffer[W-B];
  else
    salida += (1-vol) * buffer[N+W-B];
  if (W >= N/2)
      salida = salida + buffer[W-N/2];
  else
    salida = salida + buffer[W+N/2];

  //incremento de índices
    W++;
    flag = 1 - flag;
    if(flag == 1)
    {
        A = A + 1;
        B = B + 1;
    }
    return salida;
}
```

# 11 - Efectos Basados en Modulaciones

## 11.1 - Tremolo

El nombre de este efecto viene de la palabra temblor y está asociado a una variación periódica en la intensidad o volumen de la señal. Esto se lleva a cabo mediante una modulación de amplitud como se puede ver en la Figura 11.1.1.

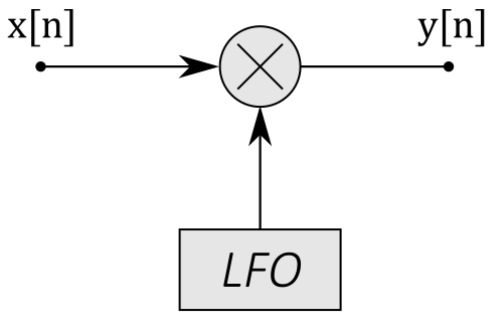


*Figura 11.1.1 – Diagrama en bloques para la implementación del Tremolo.*

El índice de modulación que multiplica a la señal es controlado por un LFO y su valor se debe ser mayor a cero para evitar distorsión por sobremodulación. También, el coeficiente debe ser menor a uno si se desea evitar la saturación a la salida.

## 11.2 - Implementación en Código: Tremolo

A nivel de implementación, el trémolo es uno de los efectos más sencillos. Solamente hay que asegurarse de configurar el LFO de tal manera que se respeten las reglas mencionadas previamente y multiplicar la entrada por el coeficiente resultante.

```
1  //Tremolo.c
2
3  #define TRINAGULAR 0
4  #define SINUSOIDAL 1
5
6  int salida;
7  int modulacion = SINUSOIDAL;
8  float coef;
9
10 int tremolo(int entrada)
11 {
12   coef = LFO(modulacion);
13   salida = coef * entrada;
14   return salida;
15 }
```

## 11.3 - Ringmod

El “ring modulator” (o modulador en anillo) es probablemente el efecto menos popular de todos los que se cubren en este trabajo. Es más común encontrarlo en música experimental o en géneros muy especiales.

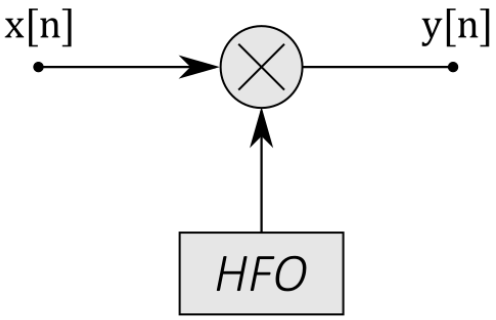


*Figura 11.3.1 – Diagrama en bloques para la implementación del Ringmod.*

Se trata de una multiplicación entre una señal periódica conocida y la señal de audio a transformar. Cuando la señal conocida es sinusoidal (como es el caso en este trabajo), el fenómeno que ocurre en el dominio de la frecuencia se denomina modulación por doble banda lateral. El término "anillo" solamente hace alusión a una configuración de diodos similar a un rectificador de onda completa utilizada en los circuitos de multiplicación analógica.

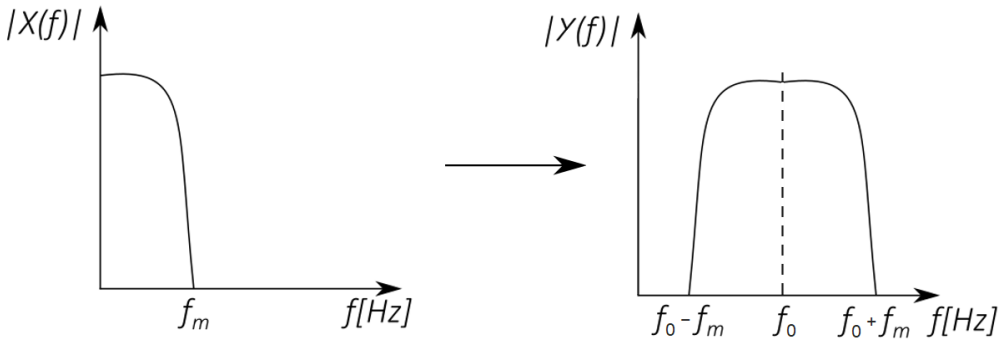


*Figura 11.3.2 – Comparación de espectro de frecuencias de una señal ejemplo a la entrada del Ringmod contra su correspondiente salida.*

Una diferencia interesante respecto a los anteriores efectos es que el oscilador local debe ser de una frecuencia suficientemente alta como para que la banda lateral inferior se encuentre dentro del espectro audible. Por esto, se lo puede denominar oscilador de alta frecuencia en este caso específico (HFO).

Esta doble banda lateral tiene implicaciones interesantes en una señal de audio; mientras más agudos son los sonidos de la banda lateral superior, más graves son los de la banda lateral inferior y viceversa. También existen técnicas que permiten separar las bandas laterales mediante la transformada de Hilbert para desarrollar un desplazador de tono (se suele aplicar como un filtro FIR en serie a la salida del modulador). Pero desafortunadamente presentan complicaciones de procesamiento para poder llevarse a cabo en tiempo real.

## 11.4 - Implementación en código: Ringmod

Si bien el oscilador debe ser de alta frecuencia, se puede reutilizar sin problemas la función LFO ya que la única diferencia yace en la configuración previa. Más allá de esto, el efecto es bastante similar al trémolo en la implementación, por lo cual, no hay muchas más complicaciones.

```
1  //Ringmod.c
2
3  #define SINUSOIDAL 1
4
5  int salida;
6  int modulacion = SINUSOIDAL;
7  float sine;
8
9  int ringmod(int entrada)
10 {
11   sine = LFO(modulacion);
12   salida = sine * entrada;
13   return salida;
14 }
```

# 12 - Efectos No Lineales

## 12.1 - Distorsión

En general, todos los efectos discutidos hasta ahora son lineales. Esto significa que la señal procesada tiene las mismas componentes de frecuencia que la señal original y sólo puede modificar la amplitud y la fase de dichas frecuencias. Por otro lado, los procesos no lineales generan frecuencias en la salida no presentes en la entrada de manera intencional o no intencional. Estas frecuencias pueden ser armónicas o inarmónicas según su relación a la frecuencia fundamental.

La distorsión es uno de los procesos no lineales más importantes en la historia contemporánea de la música. Las válvulas (y más tarde los diodos y transistores) se presentaban como los elementos básicos más importantes para presentar alinealidades en los circuitos analógicos. Los diseños de cada fabricante eran un gran secreto ya que estos procesos alineales se calibraban de manera manual escuchando la respuesta a la salida frente a cambios en el sistema. Esto lograba que cada marca y cada artista tuviese su sonido particular y lo use como un principal punto de promoción.

Cabe destacar esta oportunidad para hablar los tres términos más comunes que rondan el ámbito de los efectos no lineales: overdrive, fuzz y distorsión. Siempre se ha dado una especie de ambigüedad o intercambiabilidad entre estas palabras y lo cierto es que el significado depende de la referencia o de la fuente en cuestión. Sin embargo, para los propósitos de este trabajo, el fuzz logra un sonido radical, duro y con armónicos muy pronunciados, mientras que el overdrive se encuentra el en lado opuesto, otorgando una característica suave y cálida. Finalmente, la palabra distorsión se utilizará para describir a cualquier efecto dentro de ese espectro recientemente definido.

De todos los efectos comunes, la distorsión es el que más trabajo está teniendo para incorporarse al ámbito digital. El hecho de que los sistemas no lineales no se puedan caracterizar por una función de transferencia dificulta enormemente su reproducción mediante código. Esto ha significado que los resultados de las distorsiones digitales sean inferiores a aquellos logrados por sus contrapartes analógicas. Consecuentemente, si bien se han logrado mejoras impresionantes en el frente digital durante los últimos años, los aficionados musicales siempre mantuvieron una especie de nostalgia hacia los diseños analógicos en esta aplicación en particular.

Esta dificultad en la incorporación del ámbito digital ha llegado a tal punto en que las distorsiones digitales más reconocidas son aquellas que resuelven analíticamente las ecuaciones de modelos de los circuitos analógicos en cada ciclo de procesamiento mediante un método iterativo como Newton-Raphson. En otras palabras, ya no se trata de un deseo artístico de lograr la distorsión más increíble, sino de un deseo científico de simular lo analógico a tal punto en el cual no se pueda distinguir la diferencia entre ambos.

Una aplicación digital de una distorsión siempre debe tener en cuenta el aliasing. Se debe recordar que una señal muestreada dentro de un procesador siempre tiene componentes de frecuencia periódicas relacionadas a la frecuencia de muestreo. Debido a esto, introducir nuevas frecuencias en el proceso significa que éstas también van a aparecer sobre todas las copias del espectro, incluso en aquellas que se encuentran invertidas.

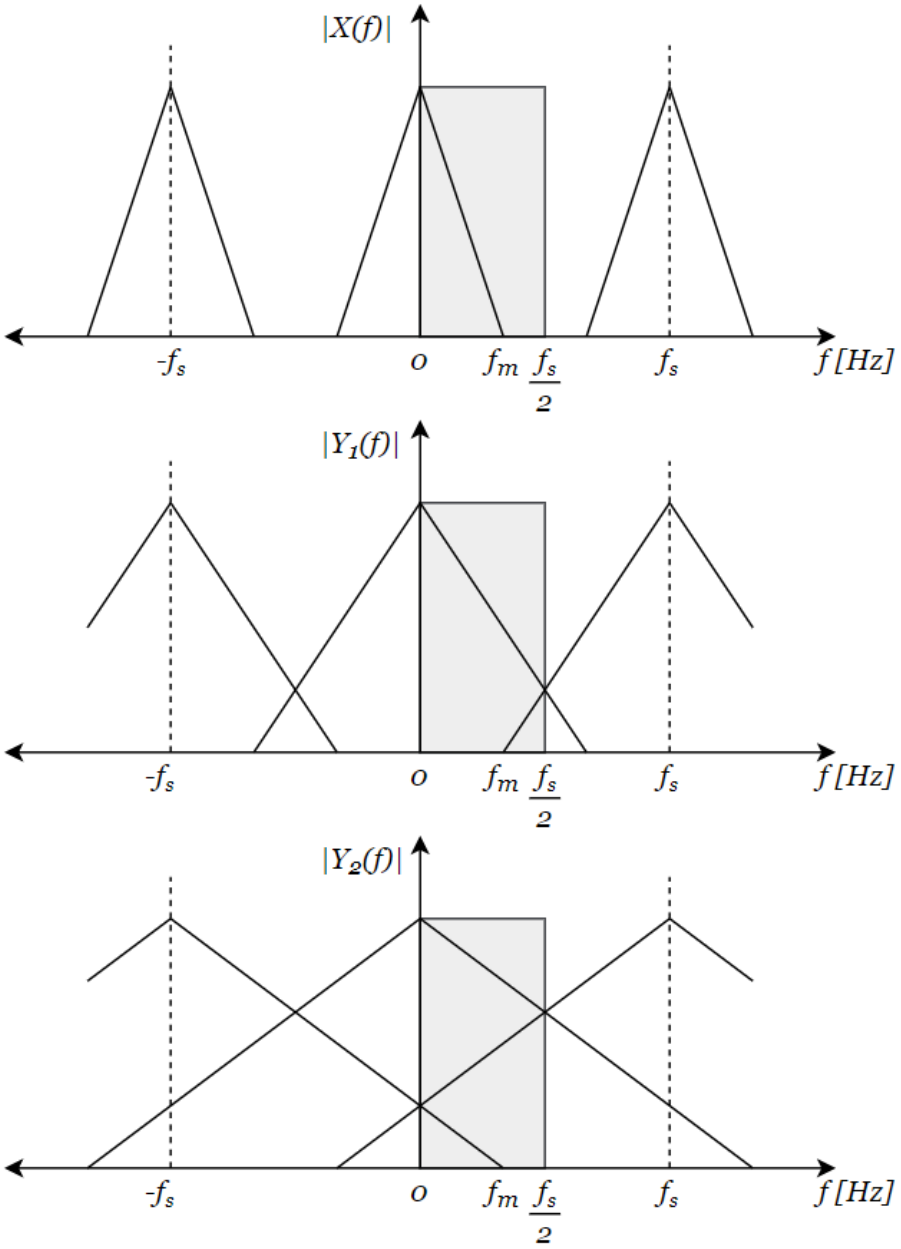


*Figura 12.1.1 – Comparación entre una entre entrada digital ejemplo y dos salidas correspondientes resultado de un proceso no lineal. La primera duplica el ancho de banda mientras que la segunda lo cuadriplica. Nótese el solapamiento de bandas por debajo de la frecuencia de Nyquist.*

Si bien hay casos extremos en los cuales se desea agregar aliasing a la señal para lograr distorsiones muy extremas, la gran mayoría de los diseños encuentran una solución para sacar o minimizar el efecto del aliasing. La primera de las soluciones comunes es el sobre-muestreo (ver Figura 12.1.2). Esta técnica realiza una interpolación de muestras antes de aplicar el proceso no lineal y se aplica un filtro pasabajos para atenuar las réplicas indeseadas. Luego de esto, se aplica la característica no lineal, se vuelve a filtrar y se termina con una decimación apropiada para restaurar la frecuencia de muestreo. La segunda solución consiste en aproximar la operación no lineal a un polinomio de Taylor con una cantidad finita de términos. Se puede demostrar que el término cuadrático del polinomio duplica el ancho de banda, el termino cúbico lo triplica y así sucesivamente. Ya que el aliasing comienza a partir de que el ancho de banda supera a la mitad de la frecuencia de muestreo, basta con aplicar un filtro pasa bajos con

frecuencia de corte distinta para cada término del polinomio para evitar el aliasing. Un diagrama en bloques ejemplo de esta estructura se puede ver en la Figura 12.1.3.

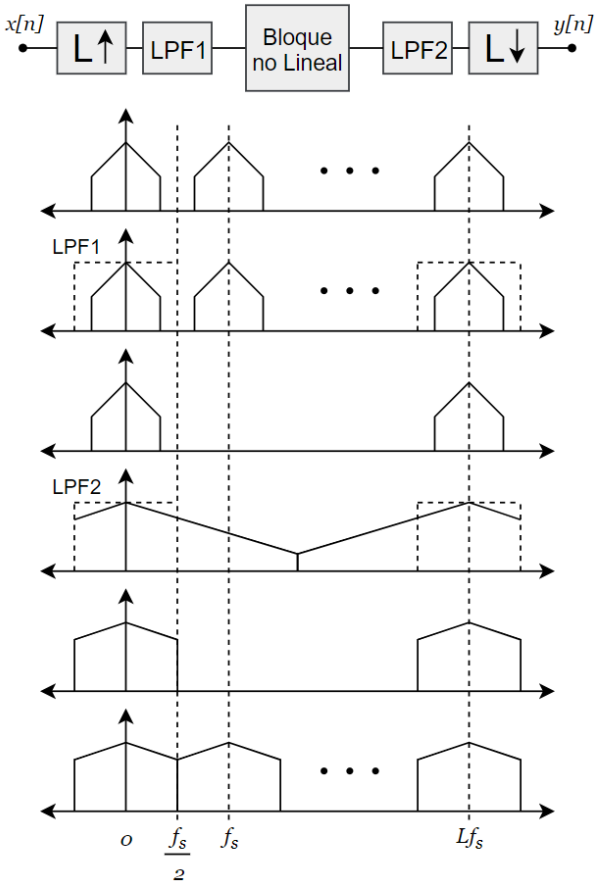


*Figura 12.1.2 – Protección al aliasing por sobre-muestreo.*

La realidad es que ambas opciones son extremadamente complejas para implementar en tiempo real, más aún sobre un sistema como el utilizado en este trabajo. Entonces, la solución de compromiso que se adoptó para enfrentar el aliasing es la de utilizar un filtro pasa bajos ajustado previamente y de manera empírica a la salida del sistema.

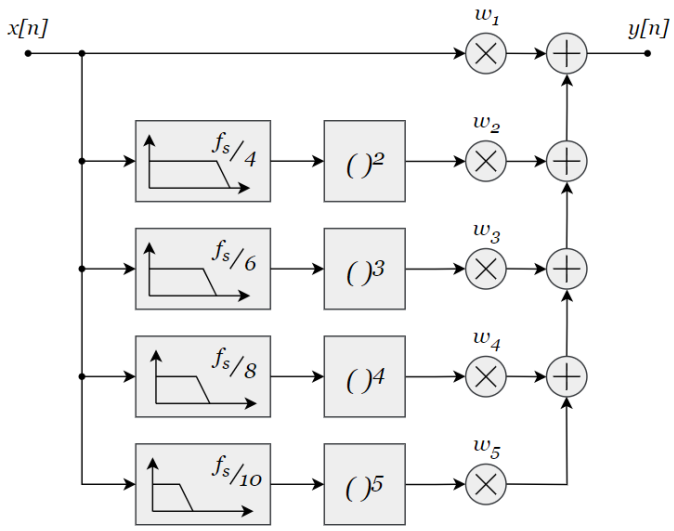


*Figura 12.1.3 – Protección al aliasing por separación de bandas.*

El proceso no lineal que se ha mencionado hasta ahora generalmente es resumido mediante un bloque denominado waveshaper (o conformador de onda) estático que se puede describir a través de una función no lineal que establece la relación entre los valores instantáneos de la señal a la entrada y salida del bloque. En este caso, la palabra estático hace referencia a que dicha función no presenta histéresis ni variación en el tiempo. También es importante destacar el hecho de que actúa sobre los valores instantáneos de la señal ya que existen procesadores no lineales que actúan sobre otros parámetros como es el caso de los compresores con la envolvente instantánea. Por último, otra distinción se suele realizar entre los waveshaper simétricos y los asimétricos. Para cumplir con la condición de simetría, la función en cuestión debe ser impar. En este trabajo solamente se tratará con waveshaper simétricos.

Un tipo de bloque básico que siempre vale la pena considerar son los hard-clipper; estos son aquellos que realizan los cambios más radicales a la señal buscando lograr un efecto más cercano al fuzz. Como primer ejemplo, existe el bloque comparador, el cual devuelve a la salida un valor si la entrada es positiva y devuelve el opuesto a dicho valor si la señal es negativa (suponiendo que el valor medio de la señal es cero). Retornar un valor fijo para distintos valores de señal es lo que en este ámbito se denomina clipping.

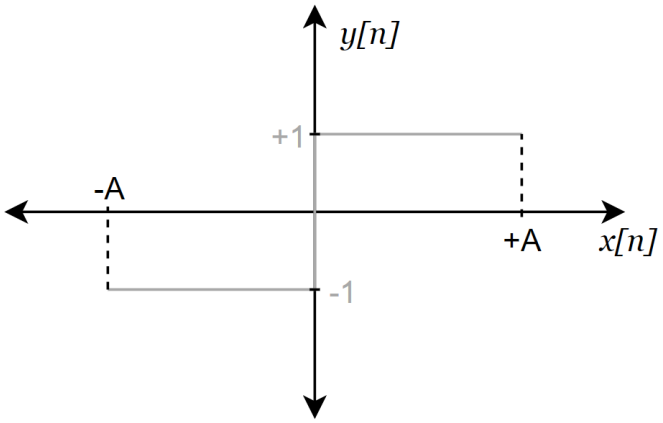


*Figura 12.1.4 – Ejemplo de waveshaper hard-clipper.*

Este sería el caso más extremo de fuzz y produce un sonido muy incómodo e indeseado en la mayoría de las situaciones. Un primer paso que se puede tomar para suavizar el resultado es el de salvar la discontinuidad de la función conectando los dos valores mediante una recta. La pendiente de esta recta es un parámetro importante por considerar ya que de ella depende el punto en el cual se hace la transición al clipping. Si, por algún motivo, la señal de entrada es lo suficientemente baja, nunca se superará el umbral de clipping y la salida no se encontrará distorsionada.

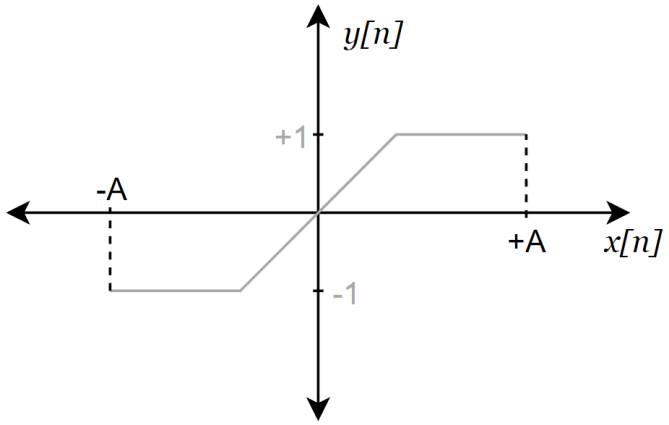


*Figura 12.1.5 – Ejemplo de waveshaper hard-clipper con una rampa para suavizar levemente la distorsión.*

A medida que se busca una aproximación a una distorsión más suave y cercana a un overdrive, más resulta de interés obtener una curva similar pero que cumpla con la condición de tener derivada primera en todos los puntos, luego derivada segunda y así sucesivamente. Así es como se llega a otra familia de waveshapers denominados soft-cippers.

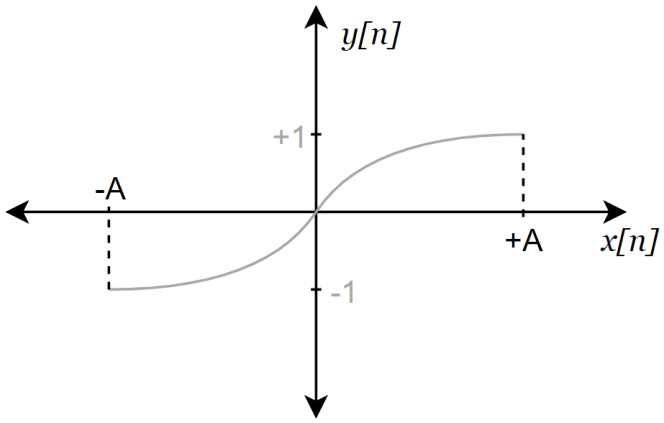


*Figura 12.1.6 – Ejemplo de waveshaper soft-clipper acotado.*

Para poder aprovechar de las ventajas de un soft-clipper al máximo, es muy conveniente utilizar un waveshaper que no se encuentre acotado. Esto significa que la función debe ser asintótica de tal manera de facilitar un dominio indeterminado hacia ambos lados. Una vez que se utiliza este tipo de bloque, se puede agregar un parámetro de ganancia antes del bloque que determinará en qué lugar del espectro de distorsiones se encontrará la salida. Cuando la ganancia tiende a infinito, el waveshaper tiene a ser un hard-clipper y cuando la ganancia es muy pequeña, se pasa a tener una salida casi no distorsionada.

Si bien los soft-clipper traen esta flexibilidad tan importante, tienen la gran desventaja de que introducen funciones matemáticas que presentan complicaciones de tiempo a la hora de realizar el procesamiento. Como consecuencia, durante mucho tiempo, el estado del arte de las distorsiones digitales se encontraba obsesionado con hallar la mejor función para lograr un soft-clipper (aquella que tenga la mejor relación entre calidad de distorsión y tiempo de procesamiento).

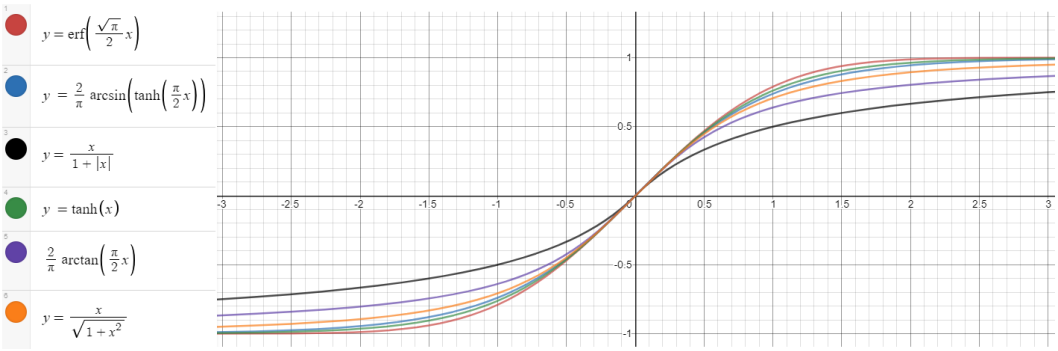


*Figura 12.1.7 – Funciones comunes para implementar waveshapers soft-clipping no acotados.*

Aplicar este bloque de distorsión a señales con alto contenido de graves suele producir sonidos no naturales característicos a las distorsiones digitales más básicas. Esto ha llevado a la utilización de filtros de deénfasis de graves antes del proceso no lineal. Las características de este filtro se suelen determinar de forma empírica. Si se desea volver a llenar el espectro de frecuencias atenuado, se debe aplicar un filtro de refuerzo de graves a la salida del waveshaper. Estos filtros de refuerzo y atenuación se pueden reutilizar del trabajo ya realizado en el ecualizador ya que se corresponden con los filtros shelving de graves (Ec. 9.1.1 y Ec. 9.1.2).

## 12.2 - Implementación en Código: Distorsión

El código utilizado para implementar la distorsión se desprende de manera sencilla a partir de lo discutido previamente. La señal amplificada mediante el parámetro de gain, se procesan mediante las funciones previamente definidas en el ecualizador para realizar el filtrado de deénfasis. Luego, este resultado se aplica a la entrada del waveshaper (que en este caso se trata de la función arco tangente) y se realiza, de manera optativa, el refuerzo de graves contrario a la atenuación previa. A la salida, también se puede agregar un filtro pasa bajos para tratar de combatir el aliasing. Finalmente, se combina la señal original con la señal distorsionada mediante un parámetro ajustable por el usuario.

Un tema que falta discutir es la amplitud de las muestras al ser procesadas por la distorsión. El waveshaper está calculado para recibir una señal que varía entre -gain y +gain y devolver una señal entre -1 y +1. Entonces, hace falta escalar la señal de una manera apropiada. Si la señal original está acotada entre -A y +A, se debe dividir por A a todas las muestras antes de aplicar la ganancia y se deben multiplicar por A todas las muestras luego de pasar por el waveshaper. También se puede agregar un control semi dinámico de volumen dividiendo la salida del waveshaper por un término dependiente de la ganancia (esto logra que, a altas ganancias, no se incremente el volumen de manera violenta).

Este es un ejemplo del código utilizado:

```
int salida = 0;
float aux=0;
float gain = 70.0, blend = 0.5;
int x1[2],x0[2], y1[2], y0[2];
float Vo[2] = {0.25 , 4}, fc[2] = {700 , 700}, c[2];

void init ()
{
    //parametros low shelve
    if(Vo[0] >= 1)
      c[0] = (1-tan(3.1416*fc[0]/SR))/(1+tan(3.1416*fc[0]/SR));
    else
      c[0] = (Vo[0]-tan(3.1416*fc[0]/SR))/(Vo[0]+tan(3.1416*fc[0]/SR));
}

int distorsion (int entrada)
{
    // escalar de -gain a +gain
    aux = (gain * aux)/8388607.0;
    // de-emphasis de graves
    aux = (float) shelve(entrada,0);
    // waveshaper
    aux = (2/3.1415)*atan(aux);
    // refuerzo de graves
    aux = (float) shelve(aux,1);
    // reescalado en funcion de la ganancia
    aux = aux * 8388607.0/ (gain/1.5);
    // blend con la entrada
    salida = blend * aux + (1 - blend) * entrada;
    return  salida;
}

int shelve(int in, int i)
{
    x1[i] = x0[i];
    x0[i] = in;
    y1[i] = y_0[i];
    y0[i] = c[i] * x0[i] - x1[i] + c[i] * y1[i];
    in = 0.5 * (Vo[i] - 1) * (in - y0[i]) + in;
    return in;
}
```

# 13 - Armado de Producto Final

## 13.1 - Integración entre la interfaz de usuario y los efectos

Como se mencionó en el capítulo 7, se diseñó una pantalla de interfaz gráfica para cada efecto implementado con su pedal correspondiente. A continuación, un resumen en orden de aparición de las 12 pantallas junto con los parámetros disponibles para que el usuario configure en cada una:

Delay + Echo


*Figura 13.1.1 – Pedal de Delay + Echo empleado en la interfaz gráfica.*

- El “level” indica el volumen de la totalidad del efecto, debido a esto, cuando se lleva el parámetro a cero, solo se debería escuchar la señal como si hubiese un bypass activado en el pedal. Si se lo lleva al máximo, se debería escuchar solamente el delay/echo sin la señal original.

- El “feedback” controla la cantidad de retroalimentación de la señal y, por ende, la cantidad de echo. El valor mínimo equivale a un delay puro sin repetición. Mientras más se aumenta este parámetro, más repeticiones se pueden escuchar de la señal original.

- El control “time” ajusta la cantidad de muestras de retardo. En otras palabras, ajusta el tiempo entre que se reproduce lo que toca el instrumento y las repeticiones.

Octavador


*Figura 13.1.2 - Pedal Octavador empleado en la interfaz gráfica.*

Este pedal no dispone de controles; una vez encendido, se duplica el sonido deseado en una octava más grave.

Chorus


*Figura 13.1.3 - Pedal de Chorus empleado en la interfaz gráfica.*

- El “rate” ajusta la frecuencia del LFO entre un valor mínimo y máximo predeterminados.

- La perilla de “depth” controla la amplitud del LFO entre un valor mínimo y máximo predeterminados.

Phaser


*Figura 13.1.4 - Pedal de Phaser empleado en la interfaz gráfica.*

- La perilla de “speed” ajusta la frecuencia del LFO entre un valor mínimo y máximo predeterminados.

Auto-Wah


*Figura 13.1.5 - Pedal de Auto-Wah empleado en la interfaz gráfica.*

- La perilla de “depth” controla la amplitud del LFO entre un valor mínimo y máximo predeterminados.

- El “rate” ajusta la frecuencia del LFO entre un valor mínimo y máximo predeterminados.

- El control “volume” indica cuánta señal limpia se mezcla con señal procesada en el efecto. El mínimo es exclusivamente señal limpia y el máximo es exclusivamente señal procesada.

- La última perilla permite personalizar la forma de onda del LFO. En orden, las posibilidades son onda cuadrada, onda triangular, rampa ascendente, rampa descendente, sinusoidal y, por

último, una onda triangular que cuya frecuencia se ajusta acorde a la amplitud recibida mediante el instrumento (ignora el valor de “rate”).

Reverb


*Figura 13.1.6 - Pedal de Reverb empleado en la interfaz gráfica.*

- El “decay” hace referencia hace referencia al tamaño de la habitación que se desea simular. Se puede ajustar entre un valor mínimo y máximo predeterminados.

- El control “mix” indica cuánta señal limpia se mezcla con señal procesada en el efecto. El mínimo es exclusivamente señal limpia y el máximo es exclusivamente señal procesada.

Distorsión


*Figura 13.1.7 - Pedal de Distorsión empleado en la interfaz gráfica.*

- El “level” indica cuánta señal limpia se mezcla con señal procesada en el efecto. El mínimo es exclusivamente señal limpia y el máximo es exclusivamente señal procesada.

- El “drive” es el responsable de controlar la ganancia que se le aplica a la señal antes de ser procesada por la distorsión. Se puede ajustar entre un valor mínimo y máximo predeterminados.

Flanger


*Figura 13.1.8 - Pedal de Flanger empleado en la interfaz gráfica.*

- El “manual” permite variar el valor de continua del LFO entre un valor mínimo y máximo predeterminados.

- La perilla de “width” controla la amplitud del LFO entre un valor mínimo y máximo predeterminados.

- La perilla de “speed” ajusta la frecuencia del LFO entre un valor mínimo y máximo predeterminados.

- El “regen” controla la cantidad de retroalimentación de la señal entre un valor mínimo y máximo predeterminados. Mientras más se aumente este parámetro, más notable será el efecto de flanger.

Tremolo


*Figura 13.1.9 - Pedal de Tremolo empleado en la interfaz gráfica.*

- La perilla de “depth” controla la amplitud del LFO entre un valor mínimo y máximo predeterminados.

- El “rate” ajusta la frecuencia del LFO entre un valor mínimo y máximo predeterminados.

- La perilla “wave” permite personalizar la forma de onda del LFO. En orden, las posibilidades son sinusoidal, onda cuadrada, onda triangular y rampa ascendente.

Vibrato


*Figura 13.1.10 - Pedal de Vibrato empleado en la interfaz gráfica.*

- El “rate” ajusta la frecuencia del LFO entre un valor mínimo y máximo predeterminados.

- La perilla de “depth” controla la amplitud del LFO entre un valor mínimo y máximo predeterminados.

- La perilla “mode” permite personalizar la forma de onda del LFO. En orden, las posibilidades son onda cuadrada, rampa ascendente, onda triangular y sinusoidal.

Ecualizador


*Figura 13.1.11 - Pedal Ecualizador empleado en la interfaz gráfica.*

El ecualizador cuenta con seis deslizadores que permiten controlar la ganancia de la señal alrededor de las frecuencias de 100 Hz, 200 Hz, 400 Hz, 800 Hz, 1600 Hz y 3200 Hz entre -12 dB y +12 dB.

Ringmod


*Figura 13.1.12 - Pedal de Ringmod empleado en la interfaz gráfica.*

- El control “frequency” ajusta la frecuencia modulante del HFO entre un valor mínimo y máximo predeterminados.

## 13.2 - Presentación física

El armado del dispositivo se diseñó con un caja plástica que se imprimió especialmente en 3D teniendo en cuenta las dimensiones para proveerlo de una presentación básica. Se tomó en consideración tanto la ubicación de los conectores de audio como la del display LCD para facilitarle al usuario el acceso a ellos. También se utilizaron ranuras para poder fijarlo mediante tornillos colocados sobre los separadores.

Además de lo descripto, se reemplazaron los conectores originales de la placa de 1/8 a conectores de 1/4, ya que estos últimos son más apropiados para la aplicación. El resultado final de estas modificaciones se muestra en las siguientes figuras:

*Figura 13.2.1 – Vista frontal superior del proyecto presentado.*

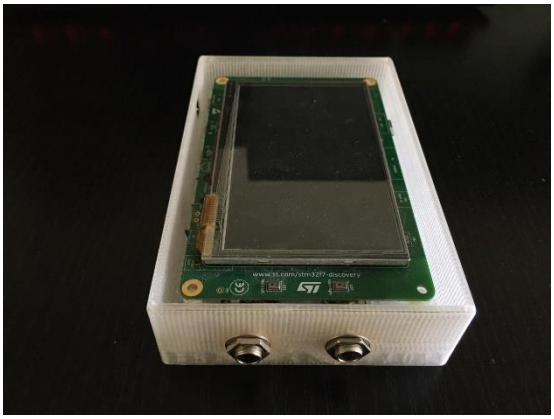

*Figura 13.2.2 – Vista lateral superior del proyecto presentado.*

# 14 - Conclusiones

## 14.1 - Producto Final y Especificaciones

Como resultado de este trabajo, se pudo desarrollar una interfaz de audio digital de una resolución de 24 bits a 48KHz (por encima del estándar de calidad de CD de 16 bits a 44,1KHz) implementada a través del kit STM32F746G Discovery. El códec WM8994 presente en la placa se utilizó como conversor analógico-digital y digital-analógico tomando su esquema de conversión delta-sigma como una contribución necesaria para poder llegar al nivel de calidad demandado por la aplicación. Mediante los periféricos SAI y DMA del procesador Cortex M7 integrado en la placa, se estableció una comunicación eficiente entre ambos chips capaz de liberar a la CPU de la mayor cantidad de carga posible mientras se encarga de realizar el DSP. Los extremos de la interfaz se hicieron accesibles mediante conectores jack 1/8 TRS, lo cual facilita la compatibilidad con implementos musicales y permite una salida estéreo.

Sobre esta base, se programaron desde cero un total de doce efectos de los más comunes en el mercado: ecualizador, auto-wah, phaser, delay+echo, vibrato, chorus, flanger, reverb, octavador, tremolo, ringmod y distorsión. Cada uno con una serie de parámetros configurables por el usuario con la opción de ajustarse de forma dinámica mientras el programa se encuentra en ejecución. Se pudo optimizar en gran parte la exigencia de estos efectos sobre el procesador implementando filtros IIR sintonizables en las instancias en las que correspondía.

La pantalla táctil TFT-LCD incluida en el kit se aprovechó para diseñar una interfaz gráfica intuitiva que facilita la interacción con el usuario. Los doce efectos se presentan en dos menúes con la opción de encendido y apagado desde allí mismo tocando los interruptores de las cajas pedaleras. Además, se incorporó la alternativa de acceder a cada uno de ellos individualmente para configurar los parámetros disponibles manipulando perillas de manera similar a lo que ocurre en un pedal convencional físico.

Se mantuvo un esquema de plataforma abierta a lo largo del proceso. Todo el hardware utilizado corresponde a una placa que se encuentra en el mercado al alcance de cualquier interesado. Lo mismo ocurre en el código; las librerías sobre las cuales se desarrolló el proyecto son libres de licencia y el código fuente se publicó en línea sobre la plataforma GitHub para permitir modificaciones y/o mejoras por parte de los usuarios.

## 14.2 - Comparación con Otros Productos del Mercado

En términos de costo, el mayor gasto consiste en conseguir el kit de desarrollo STM32F746G Discovery el cual se encuentra disponible en la página oficial de STM por US$ 55. También es necesario contar con una tarjeta micro SDHC de por lo menos 1 GB lo cual sumaría alrededor de US$ 5 para un total de US$ 60. Además de esto, hace falta acceso a un cable USB tipo A a USB mini-B, una computadora con Windows 7, 8, 10, Linux o MacOS y un lector de tarjetas SD para poder terminar de llevar a cabo el proyecto. Sin embargo, estos últimos elementos enumerados no se los toma como costos adicionales ya que se los considera de consumo generalizado.

Si bien ofrecer un producto competitivo, en cuanto al presupuesto, no es uno de los objetivos de este trabajo, resulta de interés realizar una breve comparación con las opciones similares actualmente presentes en el mercado. En este sentido, se evalúa la posición que ocuparía este desarrollo dentro del espectro comercial de pedaleras multi-efecto. Utilizando como referencia a los distribuidores de productos musicales más importantes en línea resulta claro que US$ 60

cae dentro del lado más económico por un buen margen. A partir de un valor aproximado de los US$ 100 es posible encontrar artículos más acabados a nivel empaquetado y funcionalidades. En el rango de los US$ 400 comienzan a aparecer opciones con interfaces gráficas que incorporan pantalla a color. Cabe reiterar que ningún de estos diseños comparten sus esquemáticos de hardware y mucho menos el código que utilizan ya que es todo propietario del fabricante.

En caso de querer evaluar la posibilidad de comercialización de este prototipo experimental con fines educativos, se debería certificar de acuerdo con las normativas internacionales tales como la IEC-61010, CE, entre otras.

## 14.3 - Proyección a Futuro

Además de ser un proyecto de plataforma abierta, lo cual propone una mentalidad de desarrollo continuo, esta pedalera tiene cabida para un sinfín de mejoras tanto por la parte del hardware como del software. A continuación, se exponen varios ejemplos, muchos de los cuales se pueden deducir de las características que suelen tener los productos similares.

La interacción con el usuario actualmente se encuentra limitada a la pantalla táctil que, si bien es bastante completa para ofrecer todas las opciones necesarias, a veces es preferible tener contrapartes mecánicas para ofrecer mayor robustez y practicidad. De esta manera, se podrían agregar perillas, interruptores de pie e incluso un pedal físico de expresión sincronizados con el programa.

Otro cambio que se puede hacer para mejorar a la comodidad de uso de la pedalera es el de incorporar alimentación recargable. Es posible utilizar baterías sin necesidad de hacer cambios en el proyecto si se desea ahorrar la fuente de alimentación y el cable correspondiente. Sin embargo, tener un sistema integrado de recarga sería la opción más atractiva ya que se elimina el costo de obtener baterías adicionales.

Respecto al empaquetado, es factible crear una caja más compleja que cubra toda la placa y permita acceso a los periféricos pertinentes. Para este diseño se pueden tomar algunas ideas como la adición de goma antideslizante en la base, un mecanismo para regular la inclinación de la pantalla, un compartimiento para almacenar baterías, etc.

Como proyecto de minimización de costos, un buen punto de comienzo sería probablemente un rediseño de la interfaz de audio. Actualmente corre sobre una placa de propósito general, la cual tiene una gran cantidad de funcionalidades desaprovechadas. Habría que tomar todos los componentes que se están usando, comprarlos por separado y soldarlos a una placa experimental. Esto permitiría el uso de productos sustitutos más económicos o incluso de integrados más potentes sin comprometer el costo original.

Si bien el código escrito es probablemente perfectible en términos de performance, constituye una base sólida para futuros desarrollos. Entre éstos, primero se debería considerar la viabilidad de reservar un amplio espacio de memoria para la grabación y reproducción de loops de audio que permitirían al usuario la construcción de sus propios acompañamientos. Otra alternativa destinada a completar las mencionadas opciones de acompañamiento sería la incorporación de la síntesis de sonidos. A través de estas prácticas, se agregarían variantes como la ejecución de patrones de batería predeterminados e incluso, en la misma pantalla LCD-TFT, un piano táctil.

Se abre a la vez la posibilidad a la implementación de modelización de amplificadores de manera digital para obtener un sonido más elaborado y más capacidades de personalización. También

se podría aprovechar la interfaz estéreo ya disponible para algoritmos de efectos en ambos canales tales como paneo y sonido 3D.

Finalmente, sobre esta pedalera multi-efecto, se podría estudiar la viabilidad de correr algoritmos de respuesta en frecuencia basados en la transformada rápida de Fourier. Entre éstos, un afinador de guitarra o bajo para indicar al usuario los ajustes necesarios en las clavijas para corregir la entonación de las cuerdas o un visualizador gráfico del espectro audible, por ejemplo, mediante un diagrama de barras.

## 14.4 - Reflexiones Finales

La motivación de llevar a cabo este proyecto surgió, en primera instancia, del interés particular de conectar los campos de la ingeniería electrónica con la música. Ambas áreas de especial afinidad para quienes encaramos este desarrollo.

En segunda instancia, el breve acercamiento a un prototipo construido durante la cursada de la asignatura de “Técnicas Digitales II” incentivó la decisión de encarar un nuevo camino con el objetivo de obtener un producto más acabado.

A partir de estas premisas, se inició una investigación de la bibliografía disponible y útil para solucionar los problemas planteados. En ese momento fue cuando se observó que la información sobre proyectos similares era escasa o nula. Esto llevó a la conclusión que se estaba tratando, en su gran mayoría, con desarrollos propietarios resguardados bajo licencia.

De ahí en más, gran parte del progreso realizado fue construido a base de múltiples ensayos, errores y correcciones personales. Si bien en ocasiones resultó un proceso tedioso, cada paso constituyó un valioso proceso de aprendizaje.

Finalmente, luego de terminar este proyecto, ponerlo a disposición de otros interesados representa, además de un orgullo personal, una base que ofrece cimientos para un amplio espectro de construcciones futuras.

# 15 - Bibliografía

## 15.1 - Libros


- Smith, Steven W. “The Scientist and Engineer's Guide to Digital Signal Processing”, 1997.
- Smith, J.O. “Physical Audio Signal Processing”, http://ccrma.stanford.edu/~jos/pasp/, libro en línea, 2010.
- Zölzer, Udo “DAFX: Digital Audio Effects”, 2002.
- Zölzer, Udo “Digital Audio Signal Processing”, 2da Edición, 2008.
- Dodge, Charles & Jerse, Thomas A. “Computer Music: Synthesis, Composition and Performance”, 1997.
- Ofrandis, S.J. “Introduction to Signal Processing”, 1996.


## 15.2 - Hojas de Datos


- STMicroelectronics, “STM32F75xxx and STM32F74xxx advanced Arm®-based 32-bit MCUs”, manual de referencia del MCU STM32F746G, 2018.
- STMicroelectronics, “Discovery kit for STM32F7 Series with STM32F746NG MCU”, manual de usuario del kit STM32F746G Discovery, 2017.
- Wolfson Microelectronics, “Multi-Channel Audio Hub CODEC for Smartphones”, hoja de datos del códec WM8994, 2012.


## 15.3 - Publicaciones


- Yeh, D.T. & Abel, J & Smith J.O, “Simulation of the Diode Limiter in Guitar Distortion Circuits by Numerical Solution of Ordinary Differential Equations”, 2007.
- Dattorro, J. “Effect Design”, 1997.
- Zölzer, Udo & Holters, Martin “Parametric Higher-Order Shleving Filters”, 2006.


## 15.4 - Artículos


- Bastien, Patrick (TC Helicon) “Pitch Shifting and Voice Transformation Techniques”.
- Park, Sangil (Motorola) “Principles of Sigma-Delta Modulation for Analog-toDigital Converters”.
- Kite, Thomas (Audio Precision) “Understanding PDM Digital Audio”, 2012.
- Yates, Randy “One-Bit Delta Sigma D/A Conversion”, 2004.
- Razavi, Behzad “The Delta-Sigma Modulator”, 2016.
- Embedded, “Accelerating complex audio DSP algorithms with audio-enhanced DMA”, 2006.
- Electrosmash, “MXR Phase 90 Analysis”.
- Davies, Paul, “The Secret History of Guitar Effects Pedals”, 2018.


## 15.5 - Clases


- Riskin, Eve & Ferré, Eddy & Lau, Wai Shan & Ngo, Bee, “Continuous Time Linear Systems”, 2014.
- Roberts, Stephen “Signal Processing & Filter Design”, 2003.


## 15.6 - Presentaciones


- Cohen, Ivan, “Fifty Shades of Distortion”, 2017.

# 16 - Anexo

## 16.1 - Oscilador de Baja Frecuencia

El “Low Frequency Oscillator” o LFO es una herramienta muy popular en la creación de efectos, originalmente se lograba con circuitos activos de realimentación positiva en los diseños analógicos. Básicamente se busca generar una forma de onda periódica específica para controlar algún parámetro del efecto dinámicamente conforme avanza dicha señal. El usuario tiene entonces opciones distintas a su disposición para controlar la amplitud, frecuencia y forma de onda del LFO. Si bien los nombres varían según el pedal, los más comunes para estos tres parámetros suelen ser “Depth”, “Rate” y “Sweep” respectivamente. El valor de continua u “Offset” suele ser fijo y establecido por el diseñador salvo algunas excepciones.

Las formas de onda que tienen saltos abruptos como la cuadrada o la rampa merecen un tratamiento especial ya que, si no se modifican, pueden producir conmutaciones que agregar contenido audible en el espectro de la señal. La manera más sencilla de lidiar con este problema es usar un filtro pasa bajos digital de primer orden en serie a la salida de estas formas de onda especiales.

A continuación, un código ejemplo de un LFO conformador de ondas cuadradas y triangulares:

```
//LFO.c

#define CUADRADA 0
#define TRIANGULAR 1

//parámetros de usuario
float rate = 3, depth = 500;
int modulacion = TRIANGULAR;

//parámetros de desarrollador
float fmedia = 800, lp = 0.99;

float faux, fcentral, finicial, ffinal;
float deltaf;
int periodo;
int cont = 0, flag = 0;

void inicilizacion()
{
  fcentral = fmedia;
    finicial = fmedia - depth;
    ffinal = fmedia + depth;
    periodo = (int) (SR/rate);
    deltaf = (ffinal - finicial)/periodo;  //m = Δy/Δx
}

float LFO(int modulacion)
{
  switch(modulacion)
  {
    case CUADRADA:
      if(cont == periodo/2)
      {
        if(faux == finicial)
          faux = ffinal;
        else
          faux = finicial;
        cont = 0;
      }
      fcentral = lp * fcentral + (1-lp) * faux;
      break;

    case TRIANGULAR:
      if(flag == 0)
        fcentral = fcentral + 2 * deltaf;
      if(flag == 1)
        fcentral = fcentral - 2 * deltaf;
      if(fcentral >= ffinal || fcentral <= finicial)
      {
        flag = 1 - flag;
        cont = 0;
      }
      break;
    cont++;
    return fcentral;
}
```

El parámetro “lp” fue elegido de manera experimental en este caso. Su valor debe ser lo suficientemente grande como para que no se escuchen ruidos de conmutación, pero suficientemente pequeño como para que no se distorsione la onda significativamente. La Figura 16.1.1 demuestra un caso en el cual se aplica un corrección excesiva.

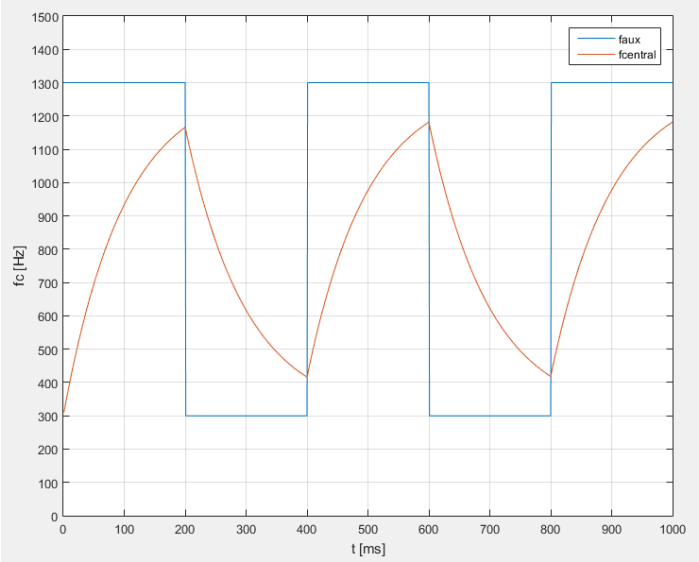


*Figura 16.1.1 – Comparación entre la salida cuadrada sin filtro y la salida cuadrada con filtro en el LFO. Los parámetros de ejemplo utilizados fueron: rate = 5 [Hz], depth = 500 [Hz], fmedia = 800 [Hz] y lp = 0.99.*

Cuando la corrección se realiza cuidadosamente, se debería obtener un resultado similar al de la Figura 16.1.2.

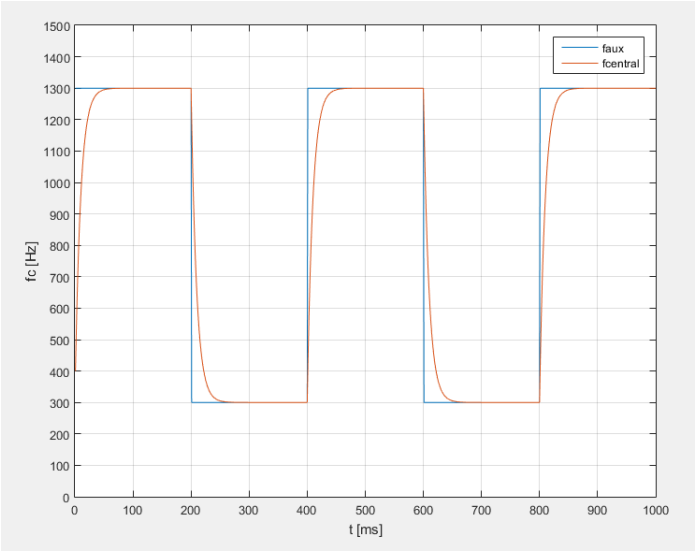


*Figura 16.1.2 - Comparación entre la salida cuadrada sin filtro y la salida cuadrada con filtro en el LFO. Los parámetros de ejemplo utilizados fueron: rate = 5 [Hz], depth = 500 [Hz], fmedia = 800 [Hz] y lp = 0.9.*

El código provisto se puede extender fácilmente para incluir otro tipo de ondas. En este trabajo en particular, se trabajó con las siguientes formas de onda: cuadrada, triangular, rampa ascendente, rampa descendente y sinusoidal. Tres de las cuales requieren el empleo de un filtro pasa bajos.

Por último, se le agregó en algunos casos al LFO la opción de actuar en función de la envolvente. En este modo, se ignora el parámetro de “rate” y se aplica un código que detecta el nivel de

señal a la entrada que, en base a eso, mediante alguna función ajusta dinámicamente el "rate". Se podría haber decidido usar una forma de onda sinusoidal ya que es la más común en términos prácticos. Sin embargo, ésta requiere significativamente más operaciones ya que se debe evaluar una función trigonométrica en lugar de calcular solamente sumas, restas y multiplicaciones. A esto se le sumaría la complejidad de calcular la envolvente y de ajustar las variables en cada muestra. Por esto, se decidió utilizar la segunda forma de onda más popular para este caso: la triangular.

## 16.2 - Demostración de la respuesta en frecuencia de un filtro peine IIR

Partiendo de la expresión planteada:

$$|y(t)| = \sqrt{\left[\sum_{n=0}^{\infty} g^n \cos(n\theta)\right]^2 + \left[\sum_{n=0}^{\infty} g^n \operatorname{sen}(n\theta)\right]^2}$$

Las sumatorias de la ecuación se pueden reducir a una serie geométrica expresando los términos de forma exponencial. Esto se facilita si se toman en cuenta las siguientes consideraciones:

$$g^n \cdot e^{in\theta} = g^n \cdot [\cos(n\theta) + i \cdot \operatorname{sen}(n\theta)]$$

$$Re\left(g^n \cdot e^{in\theta}\right) = g^n \cos(n\theta)$$

$$Im\left(g^n \cdot e^{in\theta}\right) = g^n \operatorname{sen}(n\theta)$$

$$g^n = e^{n \cdot \ln g}$$

De esta manera, se puede calcular la siguiente serie compleja de la cual se desprenden ambos resultados evaluando su parte real e imaginaria por separado. Primero, se resuelve aplicando la igualdad conocida de series geométricas:

$$\sum_{n=0}^{\infty} e^{n \cdot \ln g} \cdot e^{in\theta} = \sum_{n=0}^{\infty} e^{n \cdot (\ln g + i\theta)} = \sum_{n=0}^{\infty} \left[e^{(\ln g + i\theta)}\right]^n = \frac{1}{1 - e^{(\ln g + i\theta)}}$$

Luego se desarrolla la expresión resultante y se multiplica ambas partes de la fracción por el conjugado del denominador para poder separar más fácilmente la parte real y la parte imaginaria:

$$\frac{1}{1 - e^{(\ln g + i\theta)}} = \frac{1}{1 - g \cdot [\cos(\theta) + i \cdot \operatorname{sen}(\theta)]}$$

$$\frac{1}{1 - g \cdot [\cos(\theta) + i \cdot \operatorname{sen}(\theta)]} = \frac{1 - g \cdot \cos(\theta) + i \cdot g \cdot \operatorname{sen}(\theta)}{[1 - g \cdot \cos(\theta)]^2 + [g \cdot \operatorname{sen}(\theta)]^2}$$

Como se puede ver, los resultados de las series de las cuales se partió son los siguientes:

$$\sum_{n=0}^{\infty} g^n \cos(n\theta) = \frac{1 - g \cdot \cos(\theta)}{[1 - g \cdot \cos(\theta)]^2 + [g \cdot \operatorname{sen}(\theta)]^2}$$

$$\sum_{n=0}^{\infty} g^n \text{sen}\,(n\theta) = \frac{g \cdot \text{sen}\,(\theta)}{[1 - g \cdot \cos(\theta)]^2 + [g \cdot \text{sen}\,(\theta)]^2}$$

Reemplazando en el módulo de la salida y simplificando la expresión, se llega al resultado final:

$$|y(t)| = \sqrt{\left[\frac{1 - g \cdot \cos(\theta)}{[1 - g \cdot \cos(\theta)]^2 + [g \cdot \text{sen}\,(\theta)]^2}\right]^2 + \left[\frac{g \cdot \text{sen}\,(\theta)}{[1 - g \cdot \cos(\theta)]^2 + [g \cdot \text{sen}\,(\theta)]^2}\right]^2}$$

$$|y(t)| = \frac{1}{\sqrt{[1 - g \cdot \cos(2\pi f \tau)]^2 + [g \cdot \text{sen}\,(2\pi f \tau)]^2}}$$

## 16.3 - Código

https://github.com/codewarlocks/stm32f7-Pedalboard