

Facultad de Ingeniería - Universidad Nacional de Asunción

Departamento de Electrónica y Mecatrónica



Procesamiento Digital de Señales

Informe Final

Trabajo Práctico Final

' Efectos para una Señal de Audio '

Integrantes:

Camila Cáceres Kowalewski - 5.435.953

Esteban Orlando Ruíz Martínez - 5.265.557

Fecha de entrega: 05 - 12 - 2025

Profesores:

Dr. Ing. Enrique Vargas

Ing. Federico Moran

Índice general

1.	Introducción	2
2.	Descripción del problema	2
3.	Objetivos	3
3.1.	Objetivo general	3
3.2.	Objetivos específicos	3
4.	Alcance y Limitaciones	3
5.	Especificaciones del Proyecto	4
6.	Marco Teórico	6
6.1.	Señales de Audio y Digitalización	6
6.2.	Aliasing	6
6.3.	Filtros Anti-Aliasing y de Reconstrucción	7
6.4.	Filtros Digitales	7
6.5.	PSoC y arquitectura de procesamiento	9
6.6.	Procesamiento Basado en Retardos	10
6.7.	Modificación del Tono (<i>Pitch – Shifting</i>): Efecto Chipmunk	11
7.	Esquema General de Solución	12
7.1.	Cálculo de la Atenuación Mínima Requerida para el Filtro Anti-aliasing (A_{MIN})	12
7.2.	Cálculo de valores del filtro anti-aliasing	13
8.	Desarrollo de los efectos en MATLAB	18
8.1.	Efecto de Eco en MATLAB	18
8.2.	Efecto de Reverberación en MATLAB	21
8.3.	Efecto <i>Chipmunk</i> en MATLAB	24
9.	Implementación en el PSoC	26
9.1.	Top Design del PSoC	26
9.2.	Configuración del Convertidor Analógico–Digital (ADC)	27
9.3.	Configuración del Convertidor Digital–Analógico (DAC)	27
9.4.	Código Fuente del PSoC	28
10.	Conclusiones	30
11.	Apéndice	32

1. Introducción

El procesamiento digital de señales constituye una herramienta fundamental para la manipulación y mejora del audio en aplicaciones modernas como la música y el cine. A través de técnicas digitales es posible transformar una señal sonora en tiempo real, aplicando efectos que modifican características como la amplitud, el tiempo y la textura del sonido. Estos procesos permiten enriquecer la experiencia auditiva y comprender de manera práctica cómo una señal puede ser alterada mediante algoritmos y sistemas embebidos.

En este proyecto se implementaron diferentes efectos para una señal de audio —principalmente eco, reverberación y *chipmunk*— utilizando como plataforma de procesamiento un microcontrolador PSoC. Esta elección permitió desarrollar un sistema capaz de digitalizar la señal de entrada, aplicar los algoritmos correspondientes y reconstruir nuevamente una señal analógica lista para su reproducción. La estructura interna configurable del PSoC, junto con sus módulos ADC y DAC, proporciona un entorno adecuado para explorar conceptos fundamentales del procesamiento digital de señales, tales como retardos, filtrado digital y modulaciones.

Además de la programación de los efectos, se incorporaron filtros analógicos de anti-aliasing y reconstrucción con el fin de asegurar que el proceso de conversión entre los dominios analógico y digital mantenga la fidelidad del audio. Esta etapa es esencial para evitar distorsiones y garantizar que la señal procesada se mantenga dentro del rango audible sin artefactos no deseados.

2. Descripción del problema

En el procesamiento digital de audio, la generación de efectos como eco, reverberación y *chipmunk* requiere la manipulación precisa de la señal en el dominio del tiempo mediante técnicas de retardo, modulación (*pitch-shifting*) para el caso de *chipmunk*. Sin embargo, trasladar estos conceptos teóricos a una implementación real implica diversos desafíos.

En primer lugar, es necesario digitalizar la señal analógica sin perder información relevante, lo cual exige un adecuado acondicionamiento mediante filtros anti-aliasing y parámetros de muestreo apropiados. Una vez en el dominio digital, los algoritmos deben ser diseñados de modo que produzcan efectos perceptibles y estables, evitando saturación o pérdidas significativas de calidad.

Por otro lado, al trabajar con un microcontrolador como el PSoC, entran en juego limitaciones propias del hardware, tales como memoria disponible (lo cual afecta la capacidad de procesamiento y restringe el manejo de buffers para retardos) y la baja resolución de los convertidores que afecta la calidad del sonido. Estas limitaciones influyen directamente en la forma en que los efectos pueden implementarse y obligan a ajustar los modelos ideales utilizados en MATLAB a versiones optimizadas y realizables en un sistema embebido.

3. Objetivos

3.1. Objetivo general

Implementar un sistema completo de procesamiento digital de audio capaz de introducir una señal mediante un conector *jack*, aplicar efectos como eco, reverberación y *chipmunk* en tiempo real y escuchar mediante un parlante, utilizando un microcontrolador PSoC, garantizando una reproducción estable del audio.

3.2. Objetivos específicos

- Diseñar y programar los algoritmos necesarios para generar efectos de audio basados en retardos y modulaciones (*pitch-shifting*).
- Implementar los efectos en MATLAB para analizar su comportamiento ideal y validar los modelos teóricos.
- Configurar y utilizar los módulos ADC_DelSig y DV DAC del PSoC para digitalizar y reconstruir la señal de audio procesada.
- Implementar filtros analógicos de anti-aliasing y reconstrucción que aseguren una conversión adecuada entre los dominios analógico y digital.

4. Alcance y Limitaciones

Alcance

- Diseño e implementación de filtros analógicos de anti-aliasing y reconstrucción para garantizar una adecuada conversión entre los dominios analógico y digital.
- Digitalización de la señal de audio mediante el ADC Delta Sigma del PSoC y su posterior reconstrucción analógica mediante el DV DAC.
- Implementación de efectos de audio basados en técnicas de retardo y modulación, específicamente eco, reverberación y *chipmunk*.
- Simulación previa de los efectos en MATLAB para validar el comportamiento ideal de cada algoritmo antes de su implementación final.
- Integración de los algoritmos en el microcontrolador PSoC, permitiendo el procesamiento en tiempo real de la señal.
- Evaluación auditiva de la calidad de los efectos generados, verificando su funcionamiento estable y perceptible.

Limitaciones

- La cantidad de memoria disponible en el PSoC limita la longitud máxima de los buffers de retardo el tamaño de la reverberación.
- La diferencia de resolución entre el ADC Delta Sigma y el DV DAC hacen que se pierda cierta información en la señal procesada respecto a la de entrada.
- El procesamiento en tiempo real impone restricciones sobre la complejidad de los algoritmos como el aumento de velocidad del audio, evitando el uso de modelos muy pesados o con alto costo computacional.

5. Especificaciones del Proyecto

Las especificaciones del presente proyecto abarcan los parámetros técnicos, componentes y condiciones de operación necesarias para implementar un sistema de procesamiento digital de audio capaz de aplicar en tiempo real los efectos de eco, reverberación y *chipmunk*. Estas especificaciones describen el funcionamiento esperado del sistema, los requerimientos del hardware y las restricciones propias del entorno de desarrollo.

Especificaciones Generales

- El sistema debe adquirir una señal de audio analógica desde un conector *jack*, digitalizar mediante el ADC Delta Sigma del PSoC y procesarla en tiempo real.
- Los efectos implementados deben ser audibles, estables y perceptibles sin introducir distorsión excesiva.
- La señal procesada debe ser convertida nuevamente al dominio analógico mediante el DV DAC del PSoC para su reproducción.
- Se deben emplear filtros analógicos de anti-aliasing y reconstrucción con una frecuencia de corte cercana a 20 kHz.

Especificaciones del Hardware

- **Microcontrolador:** PSoC con módulos ADC Delta Sigma, DV DAC, Opamp y Hardware integrados.
 - **ADC Delta-Sigma:**
 - Resolución: 16 bits/muestra.
 - Frecuencia de muestreo: 48 kHz.
 - **DV DAC:**
 - Resolución: 12 bits/muestra.
 - Rango de voltaje de salida: 0 - 4.08 V.
 - **Hardware:**
 - Botón pulsador en pull-down y Diodo LED.
 - Glitch Filter como anti-rebote, configurado para tomar 3 muestras.
 - Clock para el funcionamiento del Glitch Filter.

- **Filtros analógicos externos:**

- **Tipo:** Activo pasa bajos Butterworth Sallen Key de cuarto y segundo orden.
- **Frecuencia de corte:** aproximadamente 20 kHz.
- **Ganancia:** del filtro anti-aliasing es aproximadamente $2.56[V/V]$ y del filtro de reconstrucción es $1.9[V/V]$.

Especificaciones de Funcionamiento

- La entrada analógica debe ser apta para ser conectada a un dispositivo emisor mediante un conector *jack*.
- El cambio entre efectos debe realizarse mediante un botón del PSoC.
- El procesamiento debe suceder sin interrupciones perceptibles para el usuario.
- La salida analógica debe ser apta para ser conectada a un amplificador o altavoz cualquiera mediante un conector *jack*.

6. Marco Teórico

El procesamiento digital de señales (DSP) permite manipular, analizar y transformar señales de audio mediante operaciones matemáticas realizadas en el dominio digital. A diferencia del procesamiento analógico, el DSP ofrece flexibilidad, precisión y la posibilidad de implementar algoritmos complejos capaces de modificar características fundamentales del sonido, tales como su amplitud, tiempo, tono y textura. En este proyecto se emplean técnicas de retardo, filtrado y modificación de tonos para generar los efectos de eco, reverberación y *chipmunk*.

6.1. Señales de Audio y Digitalización

Una señal de audio es una onda analógica continua en el tiempo. Para procesarla digitalmente, debe convertirse en una secuencia de muestras discretas mediante un convertidor analógico-digital (ADC). Este proceso requiere dos etapas fundamentales:

- **Muestreo:** toma valores de la señal en intervalos regulares definidos por la frecuencia de muestreo f_s . Para señales audibles, es común utilizar frecuencias de 44 a 48 kHz.
- **Cuantización:** asigna cada muestra a un nivel discreto representado por un número binario, cuya resolución es aconsejable que sea mayor a 12 bits para audio.

La digitalización debe realizarse respetando el teorema de Nyquist para evitar distorsiones por aliasing.

6.2. Aliasing

El aliasing es un fenómeno que aparece cuando una señal analógica contiene componentes de frecuencia superiores a la mitad de la frecuencia de muestreo y estas no son eliminadas antes de digitalizarla. Como resultado, dichas frecuencias “se pliegan” hacia el rango audible, generando componentes falsas que no estaban presentes en la señal original. Este efecto produce distorsiones significativas y altera la fidelidad del audio digitalizado.

La frecuencia que delimita esta condición es la *frecuencia de Nyquist*, definida como:

$$f_N = \frac{f_s}{2},$$

donde f_s es la frecuencia de muestreo. Para que una señal pueda ser representada correctamente en el dominio digital, todas sus componentes deben ser menores que f_N . Si esta condición no se cumple, las frecuencias superiores se interpretan erróneamente como tonos más bajos, lo que produce una representación incorrecta y una pérdida de información.

Una estrategia clásica para evitar el aliasing es el uso de un *filtro anti-aliasing*, encargado de atenuar las frecuencias superiores a f_N justo antes del proceso de digitalización.

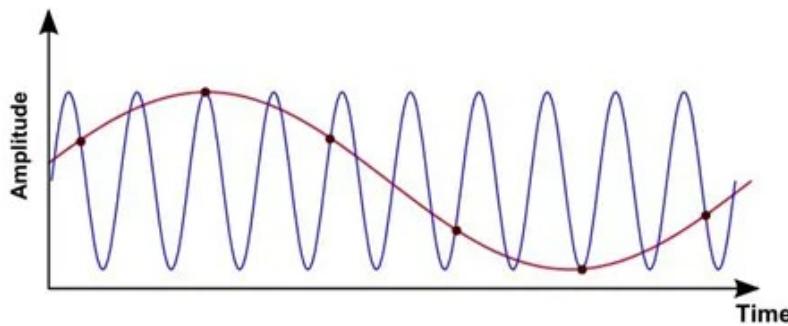


Figura 1: Ilustración del fenómeno de aliasing. **Fuente:** [1] Dynamic Signal Analysis Review - Data Physics.

6.3. Filtros Anti-Aliasing y de Reconstrucción

Antes de digitalizar la señal, es necesario eliminar las componentes de alta frecuencia superiores a $f_s/2$, mediante un *filtro anti-aliasing*. En este proyecto se implementa un filtro Butterworth de cuarto orden, obtenido mediante la conexión en cascada de dos etapas Sallen-Key de segundo orden. Esta configuración permite lograr una pendiente de atenuación más pronunciada y una mejor supresión de frecuencias no deseadas antes del ADC, reduciendo significativamente el riesgo de aliasing.

Por otro lado, al convertir nuevamente la señal digital a analógica mediante el DAC, aparecen réplicas de alta frecuencia que deben ser atenuadas mediante un *filtro de reconstrucción*. En este caso se utiliza un filtro Butterworth Sallen-Key de segundo orden, suficiente para suavizar la señal de salida y eliminar las componentes espectrales introducidas por el proceso de conversión digital-analógica.

6.4. Filtros Digitales

Filtros FIR

Un filtro FIR (Finite Impulse Response) es un tipo de filtro digital el cual cuenta con una respuesta al impulso finita. Este filtro tiene todos los polos en el origen, lo que lo vuelve un filtro estable y hace que sea de interés para aplicaciones en audio.

- **Filtros Comb FIR:**

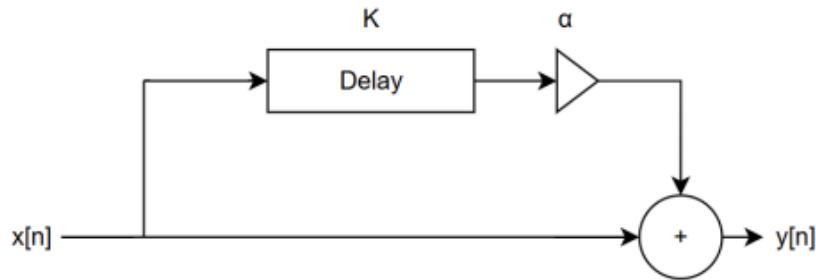


Figura 2: Diagrama de bloques del filtro Comb FIR. **Fuente:** [3] Comb filter feedforward.

Donde se puede sacar que:

$$y[n] = x[n] + \alpha \cdot x[n - K]$$

Su respuesta en frecuencia es:

$$H(e^{j\omega}) = 1 + \alpha \cdot e^{-j\omega K}$$

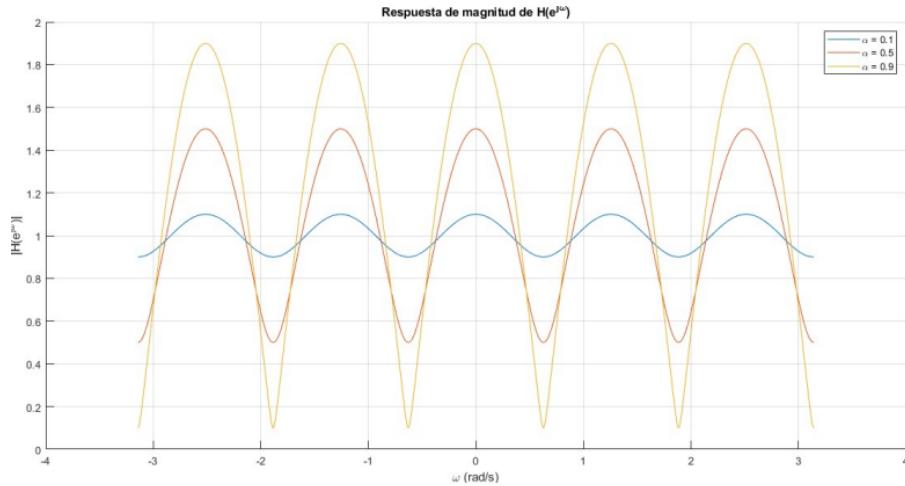


Figura 3: Magnitud de la respuesta en frecuencia del filtro FIR para varios valores de α . **Fuente:** Propia.

Filtros IIR

Un filtro IIR (Infinite Impulse Response) es un filtro digital el cual cuenta con una respuesta al impulso infinita. A diferencia de los filtros Fir, este filtro cuenta con polos y ceros que determinan su estabilidad.

- **Filtros Comb IIR:**

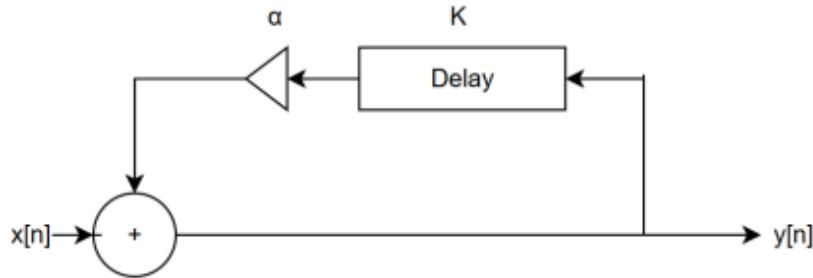


Figura 4: Diagrama de bloques del filtro Comb IIR. **Fuente:** [4] Comb filter feedback.

Donde se puede sacar que:

$$y[n] = x[n] + \alpha \cdot y[n - K]$$

Su respuesta en frecuencia es:

$$H(e^{j\omega}) = \frac{1}{1 - \alpha \cdot e^{-j\omega K}}$$

Para los valores de α menores a 1 el filtro es estable.

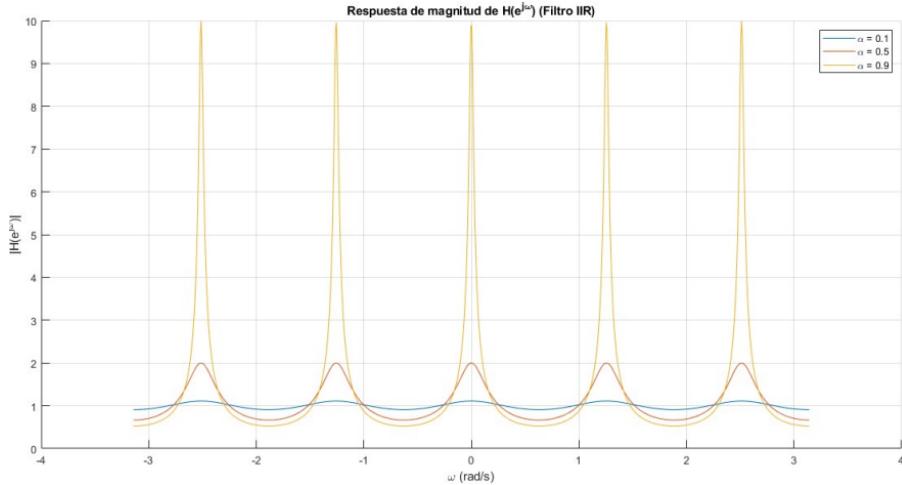


Figura 5: Magnitud de la respuesta en frecuencia del filtro IIR para varios valores de α . **Fuente:** Propia.

Se puede notar que cuando α se va acercando a 1, el filtro se vuelve cada vez más inestable.

6.5. PSoC y arquitectura de procesamiento

El PSoC (*Programmable System-on-Chip*) es una familia de microcontroladores programables que combina un núcleo de procesamiento con bloques analógicos y digitales reconfigurables. A diferencia de un microcontrolador convencional, el PSoC permite definir no solo el software, sino también gran parte del hardware interno, configurando amplificadores operacionales, convertidores A/D y D/A, filtros, módulos de comunicación y lógica digital según las necesidades del proyecto. Esta flexibilidad lo hace especialmente adecuado para aplicaciones de procesamiento de señales y sistemas embebidos donde se requiere integrar adquisición, acondicionamiento y procesamiento en un mismo dispositivo.

En este proyecto se emplea la familia PSoC 5LP, que incorpora un núcleo ARM Cortex-M3 junto con bloques dedicados a DSP (*Digital Signal Processing*), convertidores analógico-digital (ADC), digital-analógico (DAC) y amplificadores operacionales internos. Los módulos de procesamiento digital permiten implementar operaciones típicas de filtrado y manipulación de muestras, mientras que los bloques analógicos facilitan la conexión directa con señales de audio sin necesidad de una gran cantidad de circuitería externa.

Entre los componentes más relevantes para el sistema de efectos de audio se destacan:

- **ADC (Convertidor Analógico-Digital):** el PSoC 5LP dispone de un ADC de tipo *Delta-Sigma* (ADC_DelSig_1), con resoluciones de hasta 20 bits de los cuales se utilizarán 16 bits/muestra y frecuencias de muestreo adecuadas para audio (del orden de 40–48 kHz). Este módulo permite digitalizar la señal proveniente del filtro anti-aliasing, garantizando suficiente precisión para el procesamiento posterior.
- **DAC (VDAC):** el convertidor digital-analógico interno recibe los valores procesados en formato digital y los transforma en un voltaje continuo proporcional. En el proyecto se utiliza un VDAC de 8 bits, suficiente para reconstruir la señal de audio procesada y enviarla al filtro de reconstrucción y, finalmente, al amplificador y parlante.

- **Amplificadores operacionales internos:** el PSoC incluye opamps configurables que pueden trabajar como buffers, sumadores o etapas de adaptación de nivel. En la etapa de entrada se pueden emplear para llevar a cabo los filtros de anti-aliasing y reconstrucción, así como para montar la señal sobre un nivel de continua adecuado para el ADC y proteger el convertidor frente a variaciones de amplitud.
- **Bloques digitales programables y Hardware del PSoC:** permiten implementar *buffers* circulares para gestionar los retardos. También se utilizan el botón y el diodo LED de la placa PSoC.

El entorno de desarrollo PSoC Creator facilita el diseño tanto del hardware configurable como del firmware. A partir de un *Top Design* se interconectan los bloques analógicos (opamp, ADC, VDAC) y digitales (lógica de control, interrupciones) y luego se programa el microcontrolador en C para implementar los algoritmos de procesamiento. Esta integración permite definir un flujo completo: la señal analógica ingresa al opamp y al filtro anti-aliasing externo, es digitalizada por el ADC, procesada mediante los algoritmos de eco, reverberación y *chipmunk*, reconvertida a analógica por el DAC y finalmente filtrada y amplificada para su reproducción.

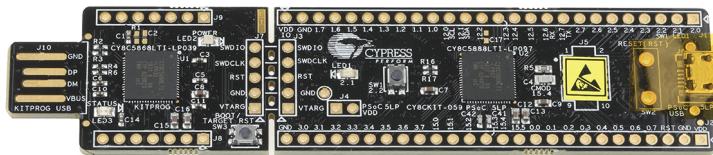


Figura 6: Psoc 5LP CY8C5888LTI-LP097. Fuente: [2] Infineon Technologies.

6.6. Procesamiento Basado en Retardos

Muchos efectos de audio se basan en almacenar muestras anteriores de la señal y recombinarlas con la señal actual. Este almacenamiento se realiza mediante un *buffer circular*, que permite retener una cantidad fija de muestras mientras se sobreescreiben de manera continua.

Eco

- **Eco Simple:**

Consiste en sumar a la señal original una copia retrasada en el tiempo atenuada (filtro *Comb FIR*):

$$y[n] = x[n] + \alpha x[n - D]$$

donde:

- D es el retardo entre muestras.
- α controla la intensidad del eco (normalmente menor a 1 para atenuarla).

- **Eco con Retroalimentación (Feedback)**

Este tipo de eco se utiliza para crear repeticiones que se desvanecen gradualmente, se trata de un filtro *Comb IIR* y sus fórmulas son:

$$y[n] = x[n] + G_{mix} \cdot d[n] \quad , \quad d[n] = x[n - L] + G_{fb} \cdot d[n - L]$$

donde:

- L es el retardo entre muestras.
- $d[n]$ es el *buffer* donde se guarda la señal retardada más reciente con una ganancia de 1 sumado los retardos anteriores atenuados.
- G_{mix} es la ganancia de mezcla del eco (normalmente menor a 1 para atenuarla).
- G_{fb} es la ganancia de retroalimentación (debe ser menor a 1 para que sea estable y para ir atenuando los retardos del feedback).

Nos quedamos con esta última opción debido a un análisis cualitativo entre ambos algoritmos (ya que en el mismo se puede apreciar un eco más natural) y que no es tan complejo computacionalmente.

Reverberación

La reverberación simula múltiples reflexiones sucesivas de un sonido dentro de un espacio cerrado. Puede modelarse combinando varios retardos con diferentes ganancias:

$$y[n] = x[n] + \sum_{k=1}^N \beta_k x[n - D_k]$$

Este algoritmo de Reverberación representa un conjunto de filtros *comb FIR*.

Esta es nuestra única opción para el proyecto, debido a otros algoritmos no mejoran tanto en calidad por la complejidad que se agrega.

6.7. Modificación del Tono (*Pitch – Shifting*): Efecto Chipmunk

El efecto *chipmunk* o *pitch – shifting* se obtiene elevando el tono de la voz sin variar la velocidad del audio. Una forma sencilla de lograrlo en un sistema embebido consiste en modificar el índice de lectura de la señal de tal forma que se lea más rápido que lo que se escribe, interpolando los valores en caso de obtener un índice de lectura decimal para suavizar la señal.

Como el índice de lectura avanza más rápido que el de escritura, para mantener la misma velocidad entonces cuando llega a un umbral la lectura realiza un salto atrás, lo suficiente para dar un espacio a la escritura pero también lo mínimo para que no sea perceptible al oído:

■ **Lectura acelerada con interpolación:**

$$y_{procesada}[n] = \text{INTERPOLAR}(x[n * k]) \rightarrow x[\text{floor}(n * k)] + x[\text{ceiling}(n * k)]$$

donde $k > 1$ provoca un aumento del tono (*pitch-up*).

■ **Solapamiento y Adición:**

$$\text{Si}(\text{distancia} < \text{UMBRAL}) \rightarrow n * k = n * k - L$$

donde L es un *gap* entre los índices de escritura y lectura (debe ser aproximadamente menor a 25[ms] para que no sea perceptible).

$$y[n] = (1 - G_{mix}) * x[n] + G_{mix} * y_{procesada}[n]$$

donde G_{mix} debe ser mayor a 0.8. Se le suma un pequeño porcentaje de la señal original para suavizar la salida en los saltos que realiza la señal procesada.

7. Esquema General de Solución

Para implementar un sistema de procesamiento digital de audio capaz de aplicar efectos en tiempo real, es necesario integrar adecuadamente las etapas analógicas y digitales que conforman el flujo completo de la señal. En este proyecto, el tratamiento del audio se divide en tres bloques principales: acondicionamiento analógico, procesamiento digital en el PSoC y reconstrucción analógica de la señal procesada.

7.1. Cálculo de la Atenuación Mínima Requerida para el Filtro Anti-aliasing (A_{MIN})

El diseño del filtro anti-aliasing se fundamenta en el principio de atenuar las componentes de frecuencia superiores a la de Nyquist por debajo del nivel de ruido de cuantificación del convertidor, haciéndolas indetectables para el ADC. Este nivel de ruido está directamente relacionado con la resolución del conversor.

Para el sistema ADC Delta-Sigma de 16 bits utilizado, con un rango de entrada diferencial de ± 1.024 V, se tienen los siguientes parámetros:

- **Resolución (N):** 16 bits
- **Rango de Escala Completa (V_{FSR}):** 2.048 V
- **Paso de Cuantificación (q)** El paso de cuantificación (q) es la mínima diferencia de voltaje que el ADC puede resolver:

$$q = \frac{V_{FSR}}{2^N} = \frac{2.048 \text{ V}}{2^{16}} = \frac{2.048 \text{ V}}{65,536} = \mathbf{31.25 \mu V} \quad (1)$$

- **Voltaje de Ruido de Cuantificación RMS (V_{n_RMS})** El ruido de cuantificación se modela como una distribución uniforme. Su valor eficaz (RMS) se toma como la referencia crítica para la atenuación requerida:

$$V_{n_RMS} = \frac{q}{\sqrt{12}} = \frac{31.25 \mu V}{\sqrt{12}} \approx \mathbf{9.02 \mu V} \quad (2)$$

- **Atenuación Mínima Requerida (A_{MIN})** La atenuación mínima requerida se calcula mediante la relación logarítmica entre el rango total del conversor (V_{FSR}) y el ruido de cuantificación RMS (V_{n_RMS}):

$$A_{MIN} = -20 \cdot \log_{10} \left(\frac{V_{n_RMS}}{V_{FSR}} \right) \quad (3)$$

Sustituyendo los valores calculados:

$$A_{MIN} = -20 \cdot \log_{10} \left(\frac{9.02 \times 10^{-6} \text{ V}}{2.048 \text{ V}} \right) = -20 \cdot \log_{10}(4.404 \times 10^{-6}) \approx \mathbf{-107.1 \text{ dB}} \quad (4)$$

7.2. Cálculo de valores del filtro anti-aliasing

Antes de ser digitalizada, la señal proveniente del jack de audio debe ser filtrada para evitar aliasing. En la sección anterior se realizó el cálculo de la atenuación mínima requerida en la $f_n = 24[kHz]$, el cual los cálculos para una frecuencia de corte de $20[kHz]$ es un filtro de orden **68**.

Para ello se optó con implementar un filtro Butterworth pasa-bajos de cuarto orden, construido mediante dos etapas Sallen-Key de segundo orden conectadas en cascada. Esta configuración proporciona una alta pendiente de atenuación por encima de la frecuencia de corte, garantizando que el ADC reciba únicamente el contenido spectral relevante del rango audible.

El diseño de cada etapa Sallen-Key requiere determinar los valores de las resistencias a partir de los capacitores seleccionados obteniendo factores de calidad $Q1 = 1.3$ y $Q2 = 0.54$ para Butterworth de 4to orden. En este caso se utilizaron capacitores comerciales de:

$$C_1 = 10 \text{ nF}, \quad C_2 = 10 \text{ nF}$$

Para un filtro Butterworth de segundo orden, los valores de las resistencias pueden obtenerse mediante la siguiente fórmula:

$$f_c = \frac{1}{2\pi RC_1}$$

donde f_c es la frecuencia de corte deseada y C_1 es el valor del condensador seleccionado. Para el cálculo de las resistencias, con un valor de $R = 780 \Omega$ (resistencias en serie de $680 \Omega + 100 \Omega$ y $C = 10, \text{nF}$, se obtiene la siguiente frecuencia de corte:

$$f_c = \frac{1}{2\pi(780)(10 \times 10^{-9})}$$

$$f_c \approx 20.4 \text{ kHz}$$

La ganancia y Factor de calidad de cada etapa, para $Rf1 = 1.5k\Omega$, $Rg1 = 10k\Omega$, $Rf2 = 27k\Omega$, $Rg2 = 22k\Omega$ es:

$$\begin{aligned} K1 &= 1 + \frac{Rf1}{Rg1} = 1.15, \quad Q1 = \frac{1}{3-K1} \approx 0.54 \\ K2 &= 1 + \frac{Rf2}{Rg2} = 2.23, \quad Q2 = \frac{1}{3-K2} \approx 1.29 \end{aligned}$$

La selección inicial de un filtro anti-aliasing de cuarto orden se basó en la disponibilidad de los amplificadores operacionales internos del PSoC. Sin embargo, la implementación con estos OpAmps internos no arrojó el rendimiento esperado en comparación con el obtenido mediante componentes discretos. Por lo tanto, se optó finalmente por utilizar el amplificador operacional externo LM324 para el diseño final del filtro.

A continuación, se presenta el circuito implementado en ORCAD, tomando los valores exactos obtenidos tras la medición de los capacitores utilizados en el diseño del filtro. Estos valores fueron empleados para simular el comportamiento del filtro pasa-bajos Butterworth de cuarto orden:

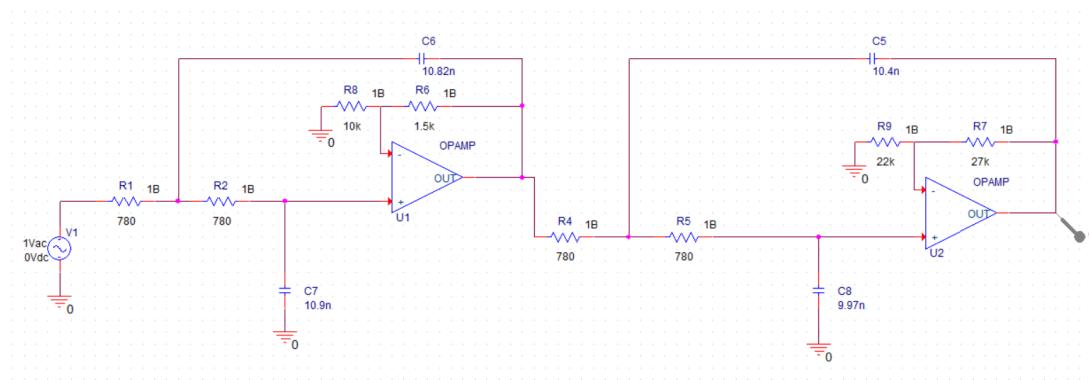


Figura 7: Filtro anti-aliasing Butterworth de cuarto orden implementado mediante dos etapas Sallen-Key.

Fuente: Propia

De la simulación del circuito anti-aliasing en ORCAD obtuvimos la siguiente respuesta en frecuencia:

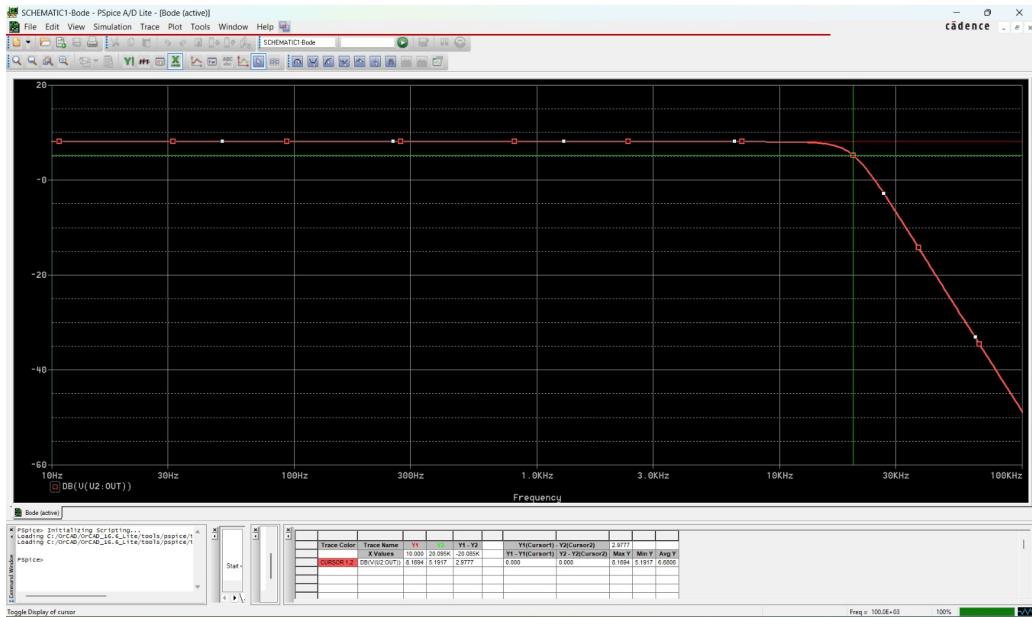


Figura 8: Respuesta en frecuencia del filtro anti-aliasing Butterworth de cuarto orden.**Fuente:** Propia

Pruebas del Filtro Anti-Aliasing

En la presente sección se describen las pruebas realizadas al filtro anti-aliasing, el cual fue implementado para evitar el fenómeno de aliasing en la digitalización de la señal de audio. Para ello, se utilizó una señal sinusoidal de entrada y se observó el comportamiento tanto de la entrada como de la salida del filtro en el osciloscopio. A continuación, se detallan los resultados obtenidos:

- La señal de entrada, representada por la onda amarilla, se aplicó al filtro anti-aliasing.
- La salida del filtro, mostrada en azul, mostró la atenuación de frecuencias no deseadas, especialmente las cercanas a la frecuencia de corte del filtro.
- Se pudo observar que la ganancia del sistema era notablemente constante en las frecuencias más bajas, mientras que las frecuencias cercanas a la frecuencia de corte comenzaban a atenuarse, evidenciando el comportamiento del filtro.

- Al introducir señales con frecuencias cercanas a la frecuencia de corte (aproximadamente 20 kHz), se evidenció una caída en la amplitud de la señal, lo cual es esperado debido a la naturaleza pasa-bajos del filtro.



Figura 9: Comparación a frecuencias bajas. **Fuente:** Propia

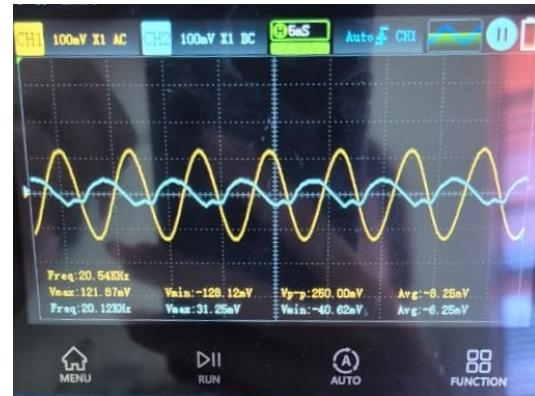


Figura 10: Comparación con frecuencia cerca-
na a la de corte. **Fuente:** Propia

Relación Señal-Ruido (SNR)

Antes del filtro anti-aliasing tenemos:

- V_{p_señal}:** 200 mV.
- V_{p_ruido}:** 1 mV (aproximación del promedio).
- SNR:**

$$SNR = 20 * \log_{10}(\frac{200}{1}) \approx 46[dB]$$

Luego del filtro-antialiasing con ganancia $K = 2.56$:

- V_{p_señal}:** 520 mV.
- V_{p_ruido}:** 2 mV (aproximación del promedio).
- SNR:**

$$SNR = 20 * \log_{10}(\frac{520}{2}) \approx 48.4[dB]$$

Luego del filtro se tiene mejor SNR, por lo que el filtro atenúa parte del ruido de entrada.



Figura 11: Medición en el osciloscopio de la señal de ruido luego del filtro anti-aliasing. **Fuente:** Propia

Procesamiento Digital en el PSoC

Una vez filtrada, la señal es digitalizada mediante el ADC Delta–Sigma del PSoC, configurado a 48 kHz. En el dominio digital se implementan los diferentes efectos utilizando buffers circulares y técnicas de manipulación temporal:

- **Eco:** retardo fijo mediante un buffer, con mezcla directa y realimentación.
- **Reverberación:** suma de múltiples retardos con distintas longitudes y ganancias, simulando reflexiones sucesivas.
- **Chipmunk:** aumento del tono mediante lectura acelerada del buffer circular.

El procesamiento ocurre dentro de la interrupción del ADC, asegurando operación estrictamente en tiempo real.

Rango Dinámico del Sistema (DR)

- **Rango Dinámico Teórico (DR_{Teórico}):** es cuando el único ruido es el de cuantización:

$$DR_{Teorico} = (6.02 * N + 1.76)dB$$

$$DR_{Teorico} = 6.02 * 16 + 1.76 = \mathbf{98.08 \text{ [dB]}}$$

- **Rango Dinámico (DR):** en el sistema tenemos un ruido externo el cual afecta el Rango Dinamico:

$$DR = 20 * \log_{10}\left(\frac{V_{RMS}}{V_{TRMS}}\right)$$

$$DR = 20 * \log_{10}\left(\frac{1.024[V]}{0.002[V]}\right) = \mathbf{54.18 \text{ [dB]}}$$

Reconstrucción y Salida Analógica

La señal procesada se envía al DVDAC del PSoC y posteriormente atraviesa un filtro de reconstrucción Butterworth Sallen-Key de segundo orden. Este filtro elimina las componentes de alta frecuencia generadas por la conversión digital-analógica, entregando una señal suave y continua apta para amplificación y reproducción en un parlante.

A diferencia del filtro anti-aliasing (cuya atenuación debe ser crítica debido a la estrecha banda de transición), el filtro de reconstrucción requiere un orden más bajo. Por lo tanto, un filtro de segundo orden es la elección más práctica y eficiente, ya que logra el alisado requerido con una baja complejidad de implementación, evitando los problemas de retardo de grupo y estabilidad que se asocian con filtros de orden superior.

El circuito utilizado se muestra a continuación:

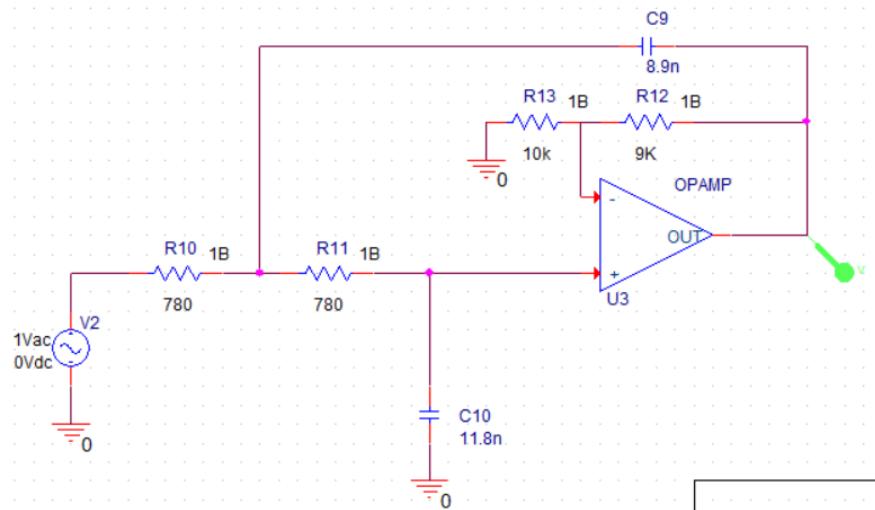


Figura 12: Filtro de reconstrucción Butterworth de segundo orden para suavizado de la salida del DAC. **Fuente:** Propia

Los valores principales son iguales al filtro anti-aliasing ($R = 780\Omega$ y $C = 10nF$) para obtener una frecuencia de corte de $20kHz$. La ganancia y factor de calidad de este filtro es:

$$K = 1 + \frac{R_f}{R_g} = 1.9, Q = \frac{1}{3-K} \approx 0.84$$

Resumen del Flujo de Señal

1. Entrada de audio desde jack.
2. Filtro anti-aliasing de cuarto orden.
3. Conversión analógico-digital mediante ADC Delta-Sigma.
4. Procesamiento del efecto seleccionado (eco, reverb o chipmunk).
5. Conversión digital-analógica mediante DAC.
6. Filtro de reconstrucción de segundo orden.
7. Salida hacia el parlante.

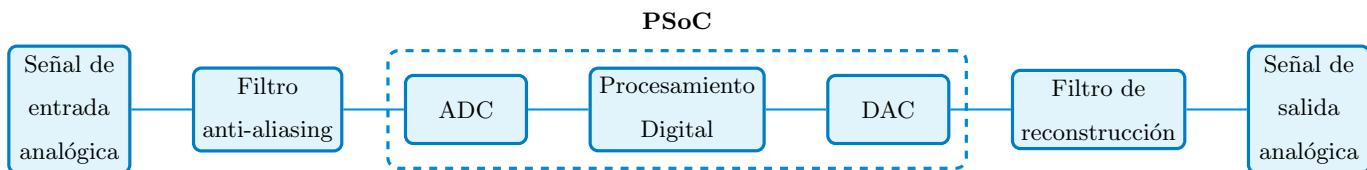


Figura 13: Diagrama general del flujo de procesamiento en el sistema de audio. **Fuente:** Propia

8. Desarrollo de los efectos en MATLAB

Con el fin de analizar el comportamiento de los efectos antes de su implementación en el PSoC, se realizaron pruebas en MATLAB utilizando archivos de audio reales.

8.1. Efecto de Eco en MATLAB

Para el efecto de eco se implementó un *buffer* circular con realimentación, emulando la misma estructura utilizada en el PSoC. El retardo se fija mediante un número de muestras determinado por el tiempo de retardo deseado y la frecuencia de muestreo, mientras que los parámetros de mezcla y *feedback* controlan la intensidad del efecto.

Listing 1: Implementación del efecto de eco en MATLAB.

```

clc;
clear;
close all;

% === Lectura de audio ===
filename = "C:\Users\kowac\Downloads\Adele_-_Some_Like_You_Original_(mp3.pm).mp3";
[x, fs] = audioread(filename);

% Si es estereo, usar solo un canal
if size(x,2) > 1
    x = x(:,1);
end

t = (0:length(x)-1)/fs;

% === Parametros del ECO (compatibles con el PSoC) ===
FS_HZ      = fs;
ECHO_DELAY_MS = 130;
ECHO_MIX    = 0.35;
ECHO_FEEDBACK = 0.3;
ECHO_MAX_SAMP = 4000;

desired = round(FS_HZ * (ECHO_DELAY_MS/1000));
echo_len = min(max(desired,1), ECHO_MAX_SAMP);

% Buffer circular
echo_buf = zeros(ECHO_MAX_SAMP,1);

```

```

echo_idx = 1;
y      = zeros(size(x));

for n = 1:length(x)
    x_n = x(n);

    % Muestra atrasada
    d = echo_buf(echo_idx);

    % d[n] = x[n] + ECHO_FEEDBACK * d[n-L]
    echo_buf(echo_idx) = x_n + ECHO_FEEDBACK*d;

    % Avance circular
    echo_idx = echo_idx + 1;
    if echo_idx > echo_len
        echo_idx = 1;
    end

    % y[n] = x[n] + ECHO_MIX * d[n]
    y(n) = x_n + ECHO_MIX*d;
end

% Normalizacion
maxval = max(abs(y));
if maxval > 1
    y = y/maxval;
end

% Graficos
figure;
plot(t, x, 'Color',[0.6 0 1], 'LineWidth',1.2); hold on; % lila
plot(t, y, 'Color',[0 0.6 1], 'LineWidth',1.2);
xlabel('Tiempo [s]');
ylabel('Amplitud');
legend('Original','Con Eco');
title('Efecto de Eco');

```

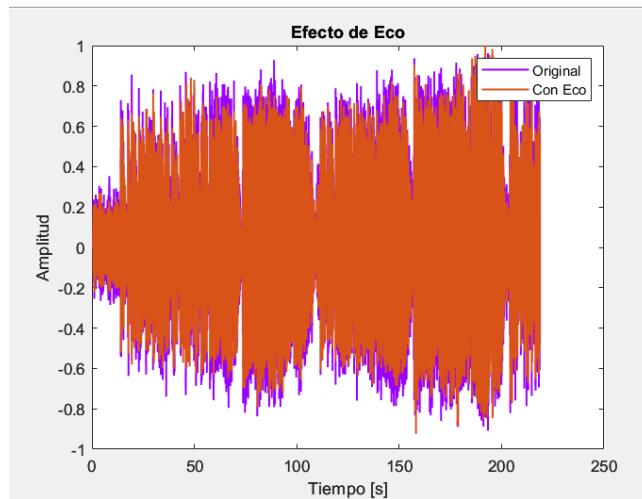


Figura 14: Efecto del Eco aplicado. **Fuente:** Propia

Efecto de Eco

El gráfico de la **Figura 14** muestra la comparación entre la señal **original** (en color lila) y la **señal con efecto de eco** (en color naranja). En el eje horizontal se representa el tiempo en segundos, mientras que en el eje vertical se encuentra la amplitud de la señal.

- **Señal Original (lila):** La señal original es una señal de audio sin ningún tipo de alteración, que mantiene su forma sinusoidal y su amplitud constante a lo largo del tiempo.
- **Señal con Eco (naranja):** La señal con el efecto de eco presenta la misma forma sinusoidal, pero con una modulación adicional en la amplitud debido al retardo introducido por el **eco**. En particular, se observa que la señal con eco tiene un **retardo** visible, lo que genera una copia atenuada de la señal original que se repite después de cierto tiempo, lo que es característico del **efecto de eco**.

El **efecto de eco** se implementa mediante un buffer circular, donde las muestras anteriores se almacenan y se retroalimentan con una ganancia (controlada por el parámetro `ECHO_FEEDBACK`) y una mezcla (`ECHO_MIX`) con la señal original. El resultado es la aparición de una **eco** que repite la señal a intervalos determinados por el parámetro de retardo `ECHO_DELAY_MS`.

Observaciones clave:

- Se puede notar que la **señal con eco** muestra una amplitud aumentada en algunas secciones, debido a la combinación de la señal original con su versión retardada.
- El **eco** puede verse claramente cuando la señal se repite a intervalos regulares, con un desfase temporal correspondiente al valor de retardo seleccionado (en este caso, 180 ms).
- La comparación entre la señal original y la señal con eco permite observar cómo se ha modificado la forma de onda debido al efecto aplicado.

La frecuencia de muestreo de $f_s = 48 \text{ kHz}$ garantiza que las muestras sean suficientemente densas para preservar la calidad del audio, sin introducir distorsiones no deseadas, mientras que el retardo de 180 ms genera un eco claramente perceptible.

8.2. Efecto de Reverberación en MATLAB

La reverberación se modeló como la suma de varios retardos con diferentes longitudes y ganancias, junto con una realimentación global. Esta estructura imita las múltiples reflexiones de un recinto cerrado y replica el algoritmo implementado en el PSoC mediante cuatro *buffers* con tamaños distintos.

Listing 2: Implementación del efecto de reverberación en MATLAB.

```

clc;
clear;
close all;

% === Lectura de audio ===
filename = "C:\Users\kowac\Downloads\Adele_-_Some_Like_You_Original_(mp3.pm).mp3";
[x, fs] = audioread(filename);

if size(x,2) > 1
    x = x(:,1);
end

t = (0:length(x)-1)/fs;

% === Parametros de la REVERB (como en el PSoC) ===
FS_HZ = fs;

RV_MAX1 = 1051;
RV_MAX2 = 1637;
RV_MAX3 = 2339;
RV_MAX4 = 3109;

RV_GAIN1 = 0.6;
RV_GAIN2 = 0.4;
RV_GAIN3 = 0.3;
RV_GAIN4 = 0.2;

RV_FEEDBACK = 0.35;
RV_MIX = 0.50;

% Buffers de retardo
rv1 = zeros(RV_MAX1,1);
rv2 = zeros(RV_MAX2,1);
rv3 = zeros(RV_MAX3,1);
rv4 = zeros(RV_MAX4,1);

i1 = 1; i2 = 1; i3 = 1; i4 = 1;

```

```

y = zeros(size(x));

for n = 1:length(x)
    x_n = x(n);

    % Muestras atrasadas
    d1 = rv1(i1);
    d2 = rv2(i2);
    d3 = rv3(i3);
    d4 = rv4(i4);

    % Suma ponderada de delays
    aux = RV_GAIN1*d1 + RV_GAIN2*d2 + RV_GAIN3*d3 + RV_GAIN4*d4;

    % feedback = x[n] + RV_FEEDBACK * aux
    feedback = x_n + RV_FEEDBACK*aux;

    % Guardar feedback en todos los buffers
    rv1(i1) = feedback;
    rv2(i2) = feedback;
    rv3(i3) = feedback;
    rv4(i4) = feedback;

    % Indices circulares
    i1 = i1 + 1; if i1 > RV_MAX1, i1 = 1; end
    i2 = i2 + 1; if i2 > RV_MAX2, i2 = 1; end
    i3 = i3 + 1; if i3 > RV_MAX3, i3 = 1; end
    i4 = i4 + 1; if i4 > RV_MAX4, i4 = 1; end

    % y[n] = x[n] + RV_MIX * aux
    y(n) = x_n + RV_MIX*aux;

end

% Normalizacion
maxval = max(abs(y));
if maxval > 1
    y = y/maxval;
end

% Graficos
figure;
plot(t, x, 'Color',[0.6 0 1], 'LineWidth',1.2); hold on; % lila
plot(t, y, 'LineWidth',1.2);
xlabel('Tiempo [s]');

```

```

ylabel('Amplitud');
legend('Original','Con Reverberacion');
title('Efecto de Reverberacion');

```

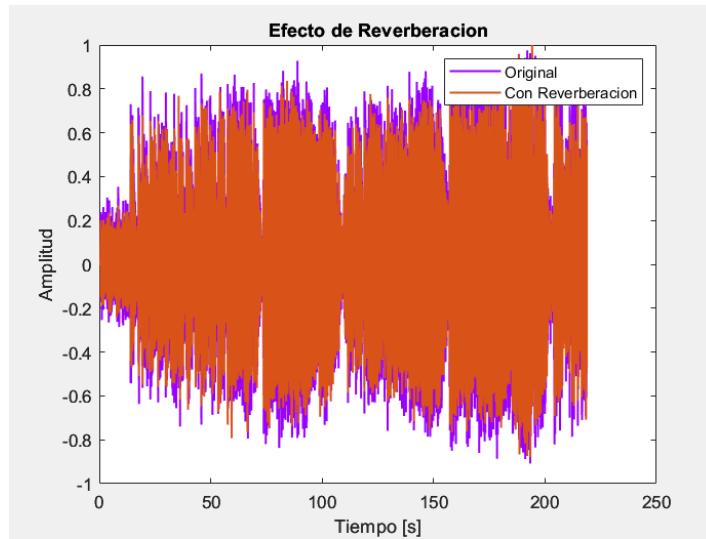


Figura 15: Efecto del Reverb aplicado. **Fuente:** Propia

Efecto de Reverberación

El gráfico de la **Figura 15** muestra la comparación entre la señal **original** (lila) y la señal con **reverberación** (naranja). En el eje horizontal está el tiempo y en el vertical, la amplitud de la señal.

- **Señal Original (lila):** La señal original es la señal de audio sin alteraciones.
- **Señal con Reverberación (naranja):** La señal con reverberación muestra la misma forma sinusoidal, pero con múltiples repeticiones atenuadas de la señal original, generando un efecto de **reverberación**. Este fenómeno es característico de la reverberación, donde las ondas reflejadas se suman a la señal original.

El **efecto de reverberación** se implementa utilizando varios buffers de retardo, donde las muestras anteriores se retroalimentan con diferentes ganancias. Los valores de ganancia para cada retardo se ajustan para crear un efecto más natural, con múltiples ecos a intervalos regulares.

Observaciones clave:

- La señal con reverberación muestra copias atenuadas de la señal original, lo que genera un efecto de reverberación en el tiempo.
- Las copias de la señal tienen una amplitud decreciente debido a la atenuación aplicada en cada retardo.
- El efecto de reverberación simula ambientes acústicos con múltiples reflejos, como en una habitación.

8.3. Efecto *Chipmunk* en MATLAB

El efecto *chipmunk* se obtiene elevando el tono de la señal. En el PSoC esto se logra leyendo el *buffer* de audio a una velocidad mayor mediante un índice fraccionario. En MATLAB se replicó este comportamiento realizando un remuestreo más rápido de la señal original y mezclándolo con una pequeña porción de la señal sin procesar.

Listing 3: Implementación del efecto chipmunk en MATLAB.

```

clc;
clear;
close all;
% === Lectura de audio ===
filename = "C:\Users\kowac\Downloads\Adele_-_Some_Like_You_Original_(mp3.pm).mp3";
[x, fs] = audioread(filename);

if size(x,2) > 1
    x = x(:,1);
end

t = (0:length(x)-1)/fs;
% === Parametros del CHIPMUNK ===
CHIP_RATIO = 1.3; % factor de pitch (igual que en el PSoC)
CHIP_MIX = 0.9; % mezcla procesada

N = length(x);
n_orig = 1:N;
n_new = 1:CHIP_RATIO:N; % lectura mas rapida
% Interpolacion lineal para obtener muestras en posiciones fraccionarias
x_int = interp1(n_orig, x, n_new, 'linear');

% Ajustar longitud para comparar
y_proc = x_int(:);
if length(y_proc) < N
    y_proc = [y_proc; zeros(N - length(y_proc),1)];
else
    y_proc = y_proc(1:N);
end
% Mezcla seca/humeda
y = (1-CHIP_MIX)*x + CHIP_MIX*y_proc;

% Normalizacion
maxval = max(abs(y));
if maxval > 1
    y = y/maxval;
end

```

```
% Graficos
figure;
plot(t, x, 'Color',[0.6 0 1], 'LineWidth',1.2); hold on;
plot(t, y,'LineWidth',1.2);
xlabel('Tiempo [s]');
ylabel('Amplitud');
legend('Original','Chipmunk');
title('Efecto Chipmunk');
```

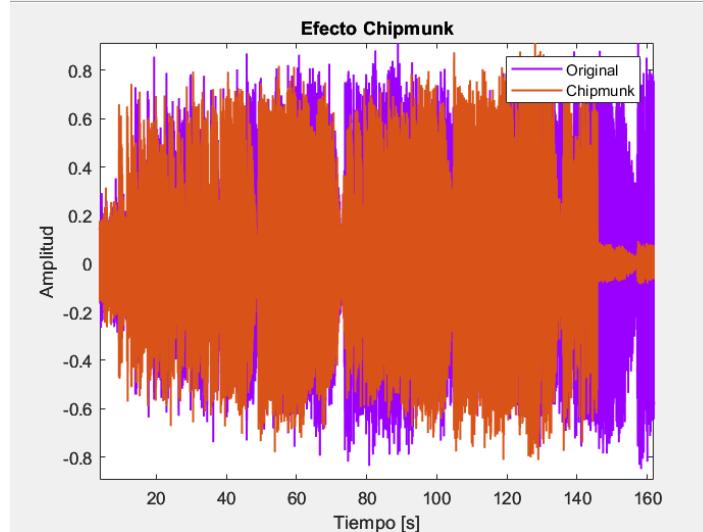


Figura 16: Efecto del Chipmunk aplicado. **Fuente:** Propia

Efecto Chipmunk

El gráfico de la **Figura 16** muestra la comparación entre la señal **original** (lila) y la señal con **efecto Chipmunk** (naranja). En el eje horizontal está el tiempo y en el vertical, la amplitud de la señal.

- **Señal Original (lila):** La señal original es la señal de audio sin alteraciones.
- **Señal con Chipmunk (naranja):** La señal con el efecto Chipmunk se acelera, aumentando su frecuencia y creando un tono más agudo. Esto se logra mediante un factor de pitch de 1.5, que acelera la señal y cambia su tono.

El **efecto Chipmunk** se implementa acelerando la señal mediante interpolación lineal, lo que ajusta la longitud de la señal y produce un sonido más rápido.

Observaciones clave:

- El **efecto Chipmunk** aumenta la frecuencia de la señal, haciéndola más aguda.
- La señal procesada mantiene la misma forma sinusoidal, pero con mayor frecuencia.
- La mezcla seca/húmeda se utiliza para ajustar la intensidad del efecto.

9. Implementación en el PSoC

La etapa digital del sistema fue implementada sobre un PSoC 5LP, aprovechando sus módulos analógicos configurables, el convertidor Delta-Sigma para la adquisición de la señal y el VDAC integrado para su reconstrucción. En esta sección se describe la configuración del hardware programable, el esquema general del *Top Design*, y los detalles más relevantes del firmware desarrollado en lenguaje C para aplicar los efectos en tiempo real.

9.1. Top Design del PSoC

En la Figura 17 se muestra el diagrama general utilizado en PSoC Creator. La estructura incluye:

- Un **ADC Delta-Sigma** configurado a 48 kHz y 16 bits para muestrear la señal filtrada.
- Un **DVDAC** de 12 bits de resolución para la generación de la señal procesada.
- Un **Opamp** interno configurado como buffer en la entrada.
- Un bloque **ISR** para gestionar la interrupción enviada por el ADC al finalizar cada conversión.
- Un **Glitch-Filter** conectado a un pulsador para el cambio de efecto.
- Un **LED** para indicar el estado del efecto seleccionado.

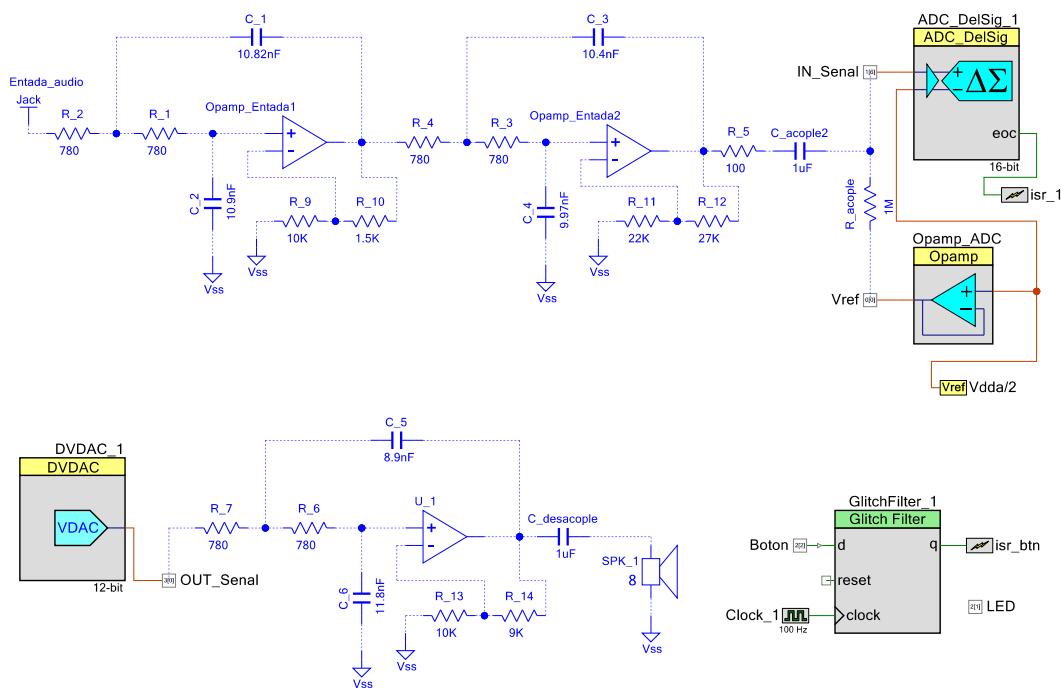


Figura 17: Top Design utilizado en PSoC Creator para la implementación de los efectos de audio. Fuente: Propia

9.2. Configuración del Convertidor Analógico–Digital (ADC)

El módulo ADC utilizado fue el *Delta-Sigma ADC* del PSoC. Los parámetros elegidos fueron los siguientes:

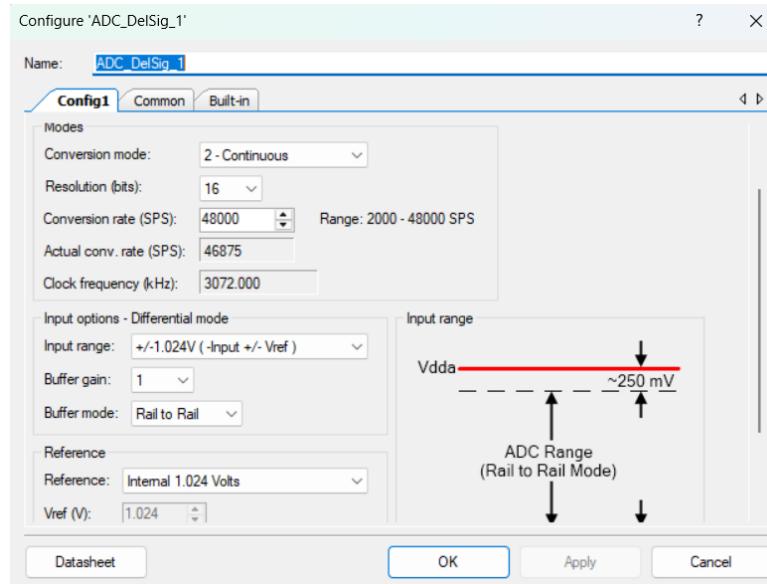


Figura 18: Configuración del Convertidor Analógico–Digital (ADC). **Fuente:** Propia

Esta configuración permite capturar de forma precisa señales de audio dentro del rango dinámico requerido y procesarlas en tiempo real. La interrupción generada por el ADC activa la rutina `Input_ISR` que transfiere la muestra a una variable global y habilita el procesamiento correspondiente.

9.3. Configuración del Convertidor Digital–Analógico (DAC)

Para la reconstrucción de la señal procesada se empleó el módulo *VDAC8*. Sus parámetros principales fueron:

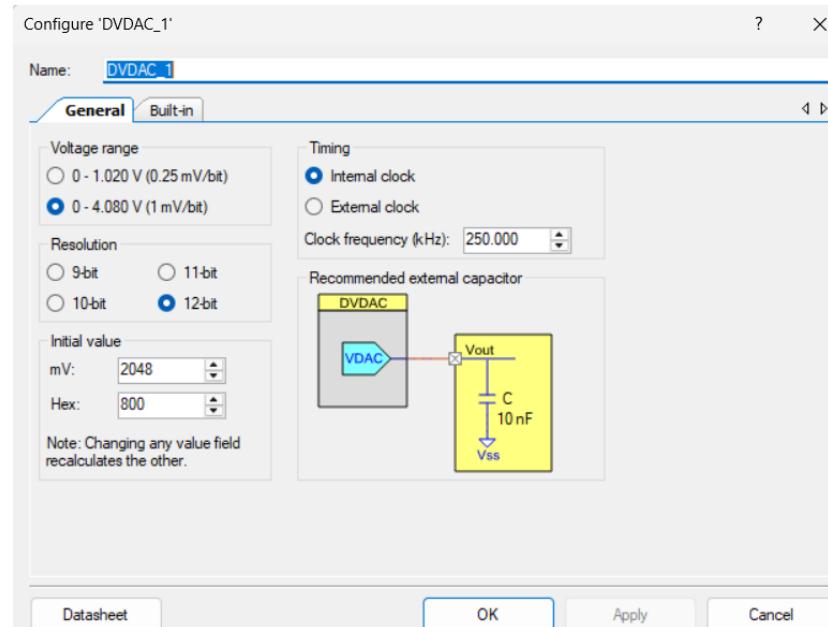


Figura 19: Configuración del Convertidor Digital–Analógico (DVDAC).**Fuente:** Propia

Antes de enviar cada muestra al DV DAC, el sistema realiza una verificación de saturación para asegurar que los valores estén dentro del rango permitido. La señal reconstruida se entrega al filtro analógico de segundo orden.

9.4. Código Fuente del PSoC

Debido a su extensión, el código completo se incluye en los Apéndices. En esta sección se resumen los módulos principales:

- **ISR Input_ISR:** copia la muestra del ADC y habilita el procesamiento.
- **ISR cambio:** ciclo entre los efectos NORMAL, ECHO, REVERB y CHIPMUNK.
- **Implementación de efectos:** cada uno basado en buffers circulares y mezclas controladas.
- **Salida analógica:** conversión a voltaje y protección contra saturación.

Tiempo de Resolución de los Efectos

El tiempo de resolución de los efectos aplicados a la señal, como el **eco**, la **reverberación** y el **efecto chipmunk**, depende de los parámetros específicos de cada efecto, tales como los **buffers de retardo**, la **frecuencia de muestreo**, y el número de **retardos** que cada efecto utiliza.

1. Tiempo de Resolución del Efecto Eco

El **tiempo de resolución** para el efecto de **eco** está determinado por el tamaño del **buffer de retardo** y la **frecuencia de muestreo**. En el código implementado, el tiempo de delay se estableció como una variable. Probamos variando esta variable entre valores estándar (de 100 ms a 180 ms) para determinar el óptimo. Concluimos que el mejor resultado audible se obtiene con un tiempo de **130 ms**.

2. Tiempo de Resolución del Efecto Reverberación

En el caso del efecto de **reverberación**, el tiempo de resolución depende de la cantidad de **retardos** y el tamaño de los **buffers** de la señal. En este proyecto, se emplean cuatro retardos con diferentes ganancias y tiempos de retardo. Los retardos y sus ganancias se definen de la siguiente forma:

$$\text{Tiempo de resolución de reverberación} = \frac{\text{RV_MAX1} + \text{RV_MAX2} + \text{RV_MAX3} + \text{RV_MAX4}}{f_s}$$

Sustituyendo los valores de los retardos $\text{RV_MAX1} = 1051$, $\text{RV_MAX2} = 1637$, $\text{RV_MAX3} = 2339$, $\text{RV_MAX4} = 3109$ y $f_s = 48\text{ kHz}$:

$$\text{Tiempo de resolución de reverberación} = \frac{1051 + 1637 + 2339 + 3109}{48000} \approx 0.22 \text{ segundos}$$

3. Tiempo de Resolución del Efecto Chipmunk

El efecto Chipmunk (cambio de tono agudo) se genera el código mediante un búfer circular con un retardo base de 21 ms, establecido inicialmente al posicionar el puntero de lectura a 1024 muestras (2048/2) detrás del puntero de escritura. Dado que la lectura avanza más rápido que la escritura, esta separación disminuye progresivamente hasta alcanzar un umbral crítico de 16 muestras (aproximadamente 0.3 ms). En ese punto, para evitar que la lectura alcance a la escritura y cause un glitch, el puntero de lectura realiza un salto controlado de ≈ 1024 muestras hacia atrás, restableciendo inmediatamente el retardo a 21 ms. Este ciclo continuo de acortamiento y restauración brusca del retardo es la técnica central que produce el efecto audible de tono elevado.

10. Conclusiones

El presente Trabajo Práctico Final cumplió satisfactoriamente con los objetivos propuestos, logrando la implementación de un sistema completo de procesamiento digital de audio en tiempo real sobre la plataforma PSoC. Se diseñaron y programaron con éxito los algoritmos basados en retardos y modulación necesarios para generar los efectos de eco, reverberación y chipmunk, validando previamente su comportamiento ideal mediante simulaciones en MATLAB. Además, se configuraron y utilizaron de manera efectiva los módulos ADC Delta Sigma y DVDAC del PSoC para la correcta digitalización y posterior reconstrucción de la señal de audio procesada , cumpliendo también con la implementación de filtros analógicos de anti-aliasing y reconstrucción para el acondicionamiento de la señal.

No obstante el éxito funcional, la fase de implementación enfrentó dificultades técnicas significativas. Una limitación clave fue la imposibilidad de utilizar los amplificadores operacionales internos del PSoC para las etapas de acondicionamiento de la señal, lo que obligó a depender exclusivamente de circuitería analógica externa. Esta restricción, sumada a las limitaciones prácticas, impidió la implementación de un filtro anti-aliasing del orden teóricamente suficiente (orden 68) necesario para cumplir con los requerimientos de la frecuencia de Nyquist.

En consecuencia, el filtro de cuarto orden finalmente implementado, si bien atenuó parte del ruido, no pudo proporcionar el flanco de corte ideal. Esto limitó la capacidad del sistema para atenuar eficazmente el ruido fuera de banda, impactando directamente en la Relación Señal/Ruido (SNR) y el rango dinámico del sistema. Sin embargo, a pesar de las limitaciones inherentes al hardware, como la memoria disponible que restringió la longitud de los buffers de retardo , y la disparidad de resolución entre el ADC de 16 bits y el DVDAC de 12 bits, el proyecto demostró ser completamente funcional. El sistema logró generar los tres efectos de audio requeridos, siendo perceptibles y estables en tiempo real, validando la comprensión y aplicación de los conceptos fundamentales del procesamiento digital de señales en un entorno embebido.

Mirando a futuro, el proyecto ofrece un margen considerable para mejoras. La optimización más crítica se centra en la etapa de filtrado, buscando alternativas para implementar un filtro anti-aliasing de mayor orden que permita una atenuación de ruido más efectiva, elevando consecuentemente la SNR y el rango dinámico del sistema. Adicionalmente, la migración a una plataforma PSoC con mayor memoria RAM y un DAC de mayor resolución (idealmente 16 bits) permitiría la implementación de buffers de retardo más largos para reverberación y eco más naturales, y reduciría la pérdida de información por cuantificación en la reconstrucción de la señal de audio.

Bibliografía

- [1] Análisis del efecto aliasing, <https://dataphysics.com/blog/dynamic-signal-analysis/dynamic-signal-analysis-review-part-3-aliasing/>
- [2] PSoC 5LP, <https://no.rs-online.com/web/p/microcontroller-development-tools/1244192>
- [3] https://commons.wikimedia.org/wiki/File:Comb_filter_feedforward.png
- [4] https://commons.wikimedia.org/wiki/File:Comb_filter_feedback.png
- [5] Autor(es). (Año). *Procesamiento Digital de Señales: Implementación de Efectos de Audio*. Universidad Nacional de Asunción. (Documento base utilizado como referencia para diseño y pruebas).
- [6] Autor(es). (Año). *Diseño e Implementación de Filtros Analógicos y ADC/DAC en PSoC*. Universidad Nacional de Asunción. (Material de referencia empleado para el diseño del filtro anti-aliasing y de reconstrucción).
- [7] Cypress Semiconductor. (2016). *PSoC 5LP Architecture Technical Reference Manual*. Recuperado de <https://www.infineon.com>
- [8] Cypress Semiconductor. (2015). *Delta-Sigma ADC Component Datasheet*. En: PSoC Creator Component Catalog.
- [9] Cypress Semiconductor. (2015). *VDAC8 Component Datasheet*. En: PSoC Creator Component Catalog.
- [10] Sedra, A., & Smith, K. (2016). *Microelectronic Circuits* (7ma ed.). Oxford University Press. (Capítulo de filtros activos y diseño Sallen-Key).
- [11] Royer, T. (2019). *Pitch-Shifting Algorithm Design and Applications in Music*. DiVA Portal. Recuperado de: <https://www.diva-portal.org/smash/get/diva2:1381398/FULLTEXT01.pdf>
- [12] Driedger, J. (2016). A Review of Time-Scale Modification of Music Signals. *Applied Sciences*, 6(2), 57. <https://doi.org/10.3390/app6020057>
- [13] Zölzer, U. (2011). *DAFX: Digital Audio Effects*. Wiley. (Referencia general de eco, reverberación y procesamiento en tiempo real).

11. Apéndice

Código en C donde se encuentran los efectos (eco, reverberación y *chipmunk*):

```

#include "project.h"
#include <stdbool.h>
#include <stdlib.h>
#include <cytypes.h> // Necesario para int16
#include <stdint.h>
#include <math.h>

////////// Variables del sistema
#define FS_HZ 48000.0f

// LED
#define LED_SLOW (FS_HZ / 4) // ~4 Hz
#define LED_FAST (FS_HZ / 6) // ~8 Hz
uint32_t led_counter = 0;

// Rango de Normalización y Escala para Conversión
#define MAX_VOLTAGE 4.08f // Máximo valor de voltaje
#define MAX_12_BIT 4095.0f // (2^12)-1 = 4095
// Muestra de salida
float sample_out;

////////// ESTADOS ///////////
#define NORMAL 0
#define ECHO 1
#define REVERB 2
#define CHIPMUNK 3

// Variable de Estados:
uint8 state = NORMAL; // Estado inicial
int8 open;           // Candado para el cambio de estados
int8 flag;

////////// EFECTOS - VARIABLES ///////////
// Efecto: Echo
#define ECHO_DELAY_MS 130      // Debe estar entre 100ms a 150ms
#define ECHO_MIX 0.35f
#define ECHO_FEEDBACK 0.3f
#define ECHO_MAX_SAMPLES 4000

```

```

static float echo_buf[ECHO_MAX_SAMPLES]; // Buffer para guardar los valores anteriores
static uint32_t echo_len = (uint32_t)(FS_HZ*(ECHO_DELAY_MS/1000.0f)); // Halla la cantidad de
muestras de retardo (L = 6240)
static uint32_t echo_index = 0;

// Efecto: Reverb
// Retardos maximos -> Salio bien con: 1000, 1600, 2400 y 3200 (usamos los primos cercanos)
#define RV_MAX1 1051      // Retardo maximo del DELAY 1
#define RV_MAX2 1637      // Retardo maximo del DELAY 2
#define RV_MAX3 2339      // Retardo maximo del DELAY 3
#define RV_MAX4 3109      // Retardo maximo del DELAY 4
// Ganancias
#define RV_GAIN1 0.6f     // Atenuacion del DELAY 1
#define RV_GAIN2 0.4f     // Atenuacion del DELAY 2
#define RV_GAIN3 0.3f     // Atenuacion del DELAY 3
#define RV_GAIN4 0.2f     // Atenuacion del DELAY 4

static float rv1[RV_MAX1], rv2[RV_MAX2], rv3[RV_MAX3], rv4[RV_MAX4]; // Buffers para guardar las
muestras anteriores
static uint32_t i1=0, i2=0, i3=0, i4=0;

#define RV_FEEDBACK 0.35f
#define RV_MIX 0.50f

// Efecto: Chipmunk
// Variables del efecto
#define CHIP_BUF_LEN 2048 // Longitud del buffer circular
#define PITCH_UP 1.35f // Aceleracion en la lectura (pitch-up)
#define CHIP_MIX 0.90f // Porcentaje de la señal con efecto que tendrá la salida

static float chip_buf[CHIP_BUF_LEN]; // Buffer circular

// Iniciamos los valores de escritura en 0 y lectura en 1024 (la mitad del buffer)
// Para el puntero de lectura se deja un gap con respecto al de escritura
uint32_t chip_wr = 0;           // Puntero de escritura
float chip_rd = (float)(CHIP_BUF_LEN/2); // Puntero de lectura

// Función para realizar la interpolación lineal
float chip_read_frac(void){
    // Normalizamos chip_rd para que se mantenga dentro del buffer circular
    while (chip_rd >= CHIP_BUF_LEN) chip_rd -= CHIP_BUF_LEN;
}

```

```

while (chip_rd < 0) chip_rd += CHIP_BUF_LEN;

int i0 = (int)chip_rd;      // En caso de que chip_rd = 10.7: i0 = 10 y i1 = 11
int i1 = i0+1;              // De esta forma hacemos la interpolacion entre chi_buf[i0] y
                           // chip_buf[i1]

if (i1>=CHIP_BUF_LEN) i1=0; // Limitamos i1 para que se mantenga en el buffer circular

float frac = chip_rd - (float)i0; // Porcentaje (para chip_rd = 10.7 -> frac = 0.7)
float v0 = chip_buf[i0];         // Valor del menor (10)
float v1 = chip_buf[i1];         // Valor del mayoy (11)

// Devolvemos la interpolacion lineal
return v0*(1.0f-frac) + v1*frac; // lectura = 0.3*v0 + 0.7*v1
}

//////////////////////////////////////////////////////////////// FUNCIONES ///////////////////////////////
// Inicializacion
void myStart(void);
// Interrupciones
CY_ISR_PROTO(Input_ISR);
CY_ISR_PROTO(cambio);
CY_ISR(Input_ISR);
CY_ISR(cambio);
// Efectos
void processAudio();
// LED
void led_update(void);

int main(void)
{
    CyGlobalIntEnable; /* Enable global interrupts. */
    // Inicializamos los componentes utilizados
    myStart();
    CyDelay(50);

    state = NORMAL; //estado inicial

    for(;;)
    {
        if(flag == 1){

```

```

        processAudio();
        flag = 0;
    }
    led_update();
}
}

// Inicializacion de componentes
void myStart(void){
    Opamp_ADC_Start();
    ADC_DelSig_1_Start();
    ADC_DelSig_1 IRQ_Start();
    ADC_DelSig_1_StartConvert();
    isr_1_StartEx(Input_ISR);
    isr_1_Enable();
    isr_btn_StartEx(cambio);
    isr_btn_Enable();
    DVDAC_1_Start();
    open = 0;
    flag = 0;
}

// Interrupcion de entrada
CY_ISR(Input_ISR){
    flag = 1;           // Cambiamos la bandera a 1
    isr_1_ClearPending(); // Limpiar interrupcion
}

// Interrupcion del boton, entra cuando se aprieta el boton
CY_ISR(cambio){
    isr_btn_ClearPending();           // Limpia la interrupcion
    if(open == 0){
        open = 1;
    } else{
        state = state+1;           // Luego de apretar el boton se cambia de estado
        if(state > CHIPMUNK) state = NORMAL; // Luego de CHORUS vuelve a NORMAL
    }
}

// Reiniciamos todos los valores: ...
// Echo
echo_index = 0;
for (uint32_t i = 0; i < ECHO_MAX_SAMPLES; i++) echo_buf[i] = 0.0f;
// Reverb
i1 = 0; i2 = 0; i3 = 0; i4 = 0;

```

```

for (uint32_t i = 0; i < RV_MAX1; i++) rv1[i] = 0.0f;
for (uint32_t i = 0; i < RV_MAX2; i++) rv2[i] = 0.0f;
for (uint32_t i = 0; i < RV_MAX3; i++) rv3[i] = 0.0f;
for (uint32_t i = 0; i < RV_MAX4; i++) rv4[i] = 0.0f;
// Chipmunk
chip_wr = 0;
chip_rd = (float)(CHIP_BUF_LEN/2);
for (uint32_t i = 0; i < CHIP_BUF_LEN; i++) chip_buf[i] = 0.0f;
// LED
led_counter = 0;
}

// Efectos
void processAudio(){
    // Conversion a float (0.0V a 4.08V)
    float sample_in = ADC_DelSig_1_CountsTo_Volts(ADC_DelSig_1_GetResult16());
    switch (state){
        // Efecto: NORMAL
        case NORMAL: {
            // y[n] = x[n]
            sample_out = sample_in;
        } break;

        // Efecto: ECHO
        case ECHO: {
            // d[n] = x[n-L] + ECHO_FEEDBACK*d[n-L]
            // d[n] son las muestras anteriores
            // Donde la ultima muestra es con ganancia 1
            // Y las anteriores estan atenuadas con ECHO_FEEDBACK
            float d = echo_buf[echo_index];
            echo_buf[echo_index] = sample_in + ECHO_FEEDBACK*d;

            // Limitamos hasta la maxima longitud de echo. Circular
            if(++echo_index >= echo_len) echo_index = 0;

            // y[n] = x[n] + ECHO_MIX*d[n]
            sample_out = sample_in + ECHO_MIX*d;
        } break;

        // Efecto: REVERB
        case REVERB: {
            // d1[n], d2[n], d3[n], d4[n] son las muestras anteriores con diferentes delays
            /* fb[n] = x[n] + RV_FEEDBACK*(RV_GAIN1*d1[n-RV_MAX1] + RV_GAIN2*d2[n-RV_MAX2]

```

```

        + RV_GAIN3*d3[n-RV_MAX3] + RV_GAIN4*d4[n-RV_MAX4]);
    */

    /* y[n] = x[n] + RV_MIX*(RV_GAIN1*d1[n-RV_MAX1] + RV_GAIN2*d2[n-RV_MAX2]
        + RV_GAIN3*d3[n-RV_MAX3] + RV_GAIN4*d4[n-RV_MAX4]);
    */

    // Leemos las muestras anteriores: rvk[n-Lk]
    float d1 = rv1[i1];
    float d2 = rv2[i2];
    float d3 = rv3[i3];
    float d4 = rv4[i4];

    // Creamos una variable auxiliar
    float aux = RV_GAIN1*d1 + RV_GAIN2*d2 + RV_GAIN3*d3 + RV_GAIN4*d4;

    // feedback[n] = x[n] + RV_FEEDBACK*aux
    float feedback = sample_in + RV_FEEDBACK*(aux);

    // Guardamos el nuevo feedback en el registro
    rv1[i1] = feedback;
    rv2[i2] = feedback;
    rv3[i3] = feedback;
    rv4[i4] = feedback;

    // Limitamos los indices a los valores maximos de retardo
    if (++i1>=RV_MAX1) i1=0;
    if (++i2>=RV_MAX2) i2=0;
    if (++i3>=RV_MAX3) i3=0;
    if (++i4>=RV_MAX4) i4=0;

    // Luego: y[n] = x[n] + RV_MIX*aux
    sample_out = sample_in + RV_MIX*aux;

} break;

// Efecto: CHIPMUNK
case CHIPMUNK: {
    //
    // Escribe muestra en buffer de a 1
    chip_buf[chip_wr] = sample_in;
    if(++chip_wr >= CHIP_BUF_LEN) chip_wr = 0;

    // Lee ms rpido x1.5 (pitch up)
    float chip = chip_read_frac(); // Hacemos la interpolacion lineal para chip_rd no
}

```

```

        entero

    chip_rd += PITCH_UP;

    // Evita que el puntero de lectura alcance al de escritura
    float dist = (float)chip_wr - chip_rd;
    while (dist < 0) dist += (float)CHIP_BUF_LEN;

    // Si hay una distacion de 16 muestras entre escritura y lectura
    if (dist < 16.0f) chip_rd -= (float)(CHIP_BUF_LEN/2); // Salta atrs L/2

    // y = 0.1*x + 0.9*chipmunk
    sample_out = (1.0f-CHIP_MIX)*sample_in + CHIP_MIX*chip; // Para suavizar los saltos

} break;
default:
break;
}

// Saturacion: rango (0V a 4.08V)
if (sample_out > MAX_VOLTAGE) sample_out = MAX_VOLTAGE;
else if (sample_out < 0.0f) sample_out = 0.0f;

// Escalar de 0.0V - 4.08V al rango del DAC (0 a +255) sumado 0.5f para el redondeo
uint16 output_val = (int16)(sample_out * ((float)MAX_12_BIT/MAX_VOLTAGE) + 0.5f);
// Asegurar que est dentro del rango [0, 255]
if (output_val > MAX_12_BIT) output_val = MAX_12_BIT;
else if (output_val < 0) output_val = 0;

DV_DAC_1_SetValue(output_val);

}

// LED
void led_update(void){
led_counter++;
switch(state){
case NORMAL:{                                // apagado
LED_Write(0);
} break;
case ECHO:{                                 // fijo
LED_Write(1);
} break;
case REVERB:{                               // parpadeo lento
}
}

```

```
if(led_counter > LED_SLOW){  
    LED_Write(!LED_Read());  
    led_counter = 0;  
}  
}  
} break;  
case CHIPMUNK:{ // parpadeo rapido  
    if(led_counter > LED_FAST){  
        LED_Write(!LED_Read());  
        led_counter = 0;  
    }  
} break;  
}  
}
```
