

TECNOLÓGICO NACIONAL DE MÉXICO EN CELAYA



Diseño de filtros digitales FIR con aplicación en procesamiento de señales de audio

TESIS

QUE PARA OBTENER EL TÍTULO DE

LICENCIADO EN INGENIERÍA ELECTRÓNICA

PRESENTA

Francisco Navarro Padilla

ASESOR: JAVIER DÍAZ CARMONA

CELAYA, Gto.

2022

*Me gusta cuando mi Madre me hace reír, pero me gusta más cuando yo
la hago reír a ella. Para la mujer de mi vida, te amo mamá.*

*De la mano de mi Padre descubrí el mundo, conocí lo bueno y lo malo,
aprendí a vivir y a soñar. Dedicado a mi mejor maestro, Es un honor
llevar tu nombre.*

Agradecimientos

Javier Díaz Carmona. Doctor, sin su conocimiento, paciencia y forma de transmitir, este trabajo no habría sido posible. Gracias por su guía y orientación.

A todos mis profesores de la carrera que compartieron conmigo una pequeña parte de su sabiduría. Todos contribuyeron para lograr que hoy sea el profesionalista que soy.

Karina Padilla y Francisco Navarro. Ustedes se merecen mas que un simple agradecimiento en este papel, se merecen el cielo. Los amo padres.

Brayan Navarro. Hermano mío, muchas veces la necesidad de ser un buen ejemplo para ti fue lo que me impulso a ir mas lejos. Estoy orgulloso de ti.

Alan Cruz. Horas de aprendizaje, horas de diversión y una amistad que me lleva para toda la vida. Eres mi mano derecha siempre.

Índice general

Índice de figuras	VII
1. Marco de Referencia	1
1.1. Introducción	1
1.2. Justificación	2
1.3. Objetivo General	3
1.4. Objetivos Específicos	4
2. Marco Teórico	5
2.1. Definición de términos	5
2.1.1. Audio Digital	5
2.1.2. Transformada de Fourier	6
2.1.3. Transformada de Fourier de tiempo reducido	7
2.1.4. Filtros Digitales	8
2.1.5. Filtros FIR	9
2.1.6. Diseño de Filtros	10
2.1.7. DSPs	10
2.2. Software y hardware empleado	11
2.2.1. Matlab	11
2.2.2. Code Composer Studio	12
2.2.3. C5535 eZdsp	13
3. Diseño e Implementación de Filtros FIR	15
3.1. GUI	15
3.1.1. Archivo	16
3.1.2. Filtro	17

3.1.3. Gráficas	19
3.2. Implementación de Filtros en DSP	22
4. Resultados	25
4.1. GUI	25
4.2. DSP	30
4.2.1. Caso I, Pasa bajas	32
4.2.2. Caso II, Pasa altas	36
4.2.3. Caso III, Pasa banda	39
4.2.4. Caso IV, Rechazo banda	41
5. Conclusiones	45
A. Interfaz de usuario	49
A.1. GUILFIR	49
B. Programa en C	61
B.1. config.h	61
B.2. main.c	63
Bibliografía	65

Índice de figuras

2.1. Ejemplo de Espectrograma.	8
2.2. Estructura básica de los filtros FIR.	10
2.3. C5535 eZdsp.	14
3.1. Ventana principal de interfaz de usuario.	16
3.2. Panel Archivo.	16
3.3. Panel Filtro.	18
3.4. Ejemplo de archivo de exportación.	19
3.5. Panel Gráficas.	20
3.6. Ejemplo. Visualización de espectro completo.	21
3.7. Ejemplo. Visualización de Espectrograma	21
3.8. Función Shift.	23
3.9. Función Convolución.	23
4.1. Método: Parks-McLellan; Filtro: Pasa Bajas; Fc: $F_{pass} = 3500\text{Hz}$, $F_{stop} = 4500\text{Hz}$; Orden: 39; Vista: FFT	26
4.2. Método: Parks-McLellan; Filtro: Pasa Bajas; Fc: $F_{pass} = 3500\text{Hz}$, $F_{stop} = 4500\text{Hz}$; Orden: 39; Vista: STFT	26
4.3. Método: Aventanado; Filtro: Pasa Altas; Fc: 6000Hz; Orden: 16; Vista: FFT	27
4.4. Método: Aventanado; Filtro: Pasa Altas; Fc: 6000Hz; Orden: 16; Vista: STFT	27
4.5. Método: Parks-McLellan; Filtro: Pasa Banda; Fc: $F_{pass1} = 4000\text{Hz}$, $F_{stop1} = 5000\text{Hz}$, $F_{pass2} = 6000\text{Hz}$, $F_{stop2} = 7000\text{Hz}$; Orden: 39; Vista: FFT	28

4.6. Método: Parks-McLellan; Filtro: Pasa Banda; Fc: $F_{pass1} = 4000\text{Hz}$, $F_{stop1} = 5000\text{Hz}$, $F_{pass2} = 6000\text{Hz}$, $F_{stop2} = 7000\text{Hz}$; Orden: 39; Vista: STFT	28
4.7. Método: Aventanado; Filtro: Rechazo banda; Fc: $Fc1 = 4000\text{Hz}$, $Fc2 = 6000\text{Hz}$; Orden: 32; Vista: FFT	29
4.8. Método: Aventanado; Filtro: Rechazo banda; Fc: $Fc1 = 4000\text{Hz}$, $Fc2 = 6000\text{Hz}$; Orden: 32; Vista: STFT	29
4.9. Herramienta FFT en CCS	30
4.10. Configuración para la gráfica FFT en CCS	31
4.11. Prueba en DSP del filtro pasa bajas con un tono de 1kHz	33
4.12. Prueba en DSP del filtro pasa bajas con un tono de 6kHz	33
4.13. Prueba en DSP del filtro pasa bajas con una señal de audio, I	34
4.14. Prueba en DSP del filtro pasa bajas con una señal de audio, II	34
4.15. Prueba en DSP del filtro pasa bajas con una señal de audio, III	35
4.16. Prueba en DSP del filtro pasa bajas con una señal de audio, IV	35
4.17. Prueba en DSP del filtro pasa altas con un tono de 7kHz	36
4.18. Prueba en DSP del filtro pasa altas con un tono de 1kHz	36
4.19. Prueba en DSP del filtro pasa altas con una señal de audio, I	37
4.20. Prueba en DSP del filtro pasa altas con una señal de audio, II	37
4.21. Prueba en DSP del filtro pasa altas con una señal de audio, III	38
4.22. Prueba en DSP del filtro pasa altas con una señal de audio, IV	38
4.23. Prueba en DSP del filtro pasa banda con un tono de 5.5kHz	39
4.24. Prueba en DSP del filtro pasa banda con un tono de 2kHz	39
4.25. Prueba en DSP del filtro pasa banda con una señal de audio, I	40
4.26. Prueba en DSP del filtro pasa banda con una señal de audio, II	40
4.27. Prueba en DSP del filtro pasa banda con una señal de audio, III	40
4.28. Prueba en DSP del filtro pasa banda con una señal de audio, IV	41
4.29. Prueba en DSP del filtro rechaza banda con un tono de 10kHz	41
4.30. Prueba en DSP del filtro rechaza banda con un tono de 5kHz	42

ÍNDICE DE FIGURAS

IX

4.31. Prueba en DSP del filtro rechaza banda con una señal de audio, I	42
4.32. Prueba en DSP del filtro rechaza banda con una señal de audio, II	43
4.33. Prueba en DSP del filtro rechaza banda con una señal de audio, III	43
4.34. Prueba en DSP del filtro rechaza banda con una señal de audio, IV	43

Capítulo 1

Marco de Referencia

1.1. Introducción

El proyecto descrito en esta tesis está enfocado al diseño e implementación de filtros digitales FIR. Para la etapa de diseño se propone una interfaz gráfica de usuario (GUI) en MATLAB con las herramientas necesarias para obtener los coeficientes y la comprobar el funcionamiento del filtro. A diferencia de la herramienta Filter Designer, disponible en MATLAB, la GUI propuesta en este proyecto tiene funciones específicas para manipular las señales de audio. Entre las características principales de la interfaz se destaca la posibilidad de leer y grabar archivos de audio, aplicar el filtro directamente al conjunto de muestras cargadas, reproducir los audios filtrados y observar los espectrogramas de las señales de entrada y salida. La etapa de implementación se realizó mediante el entorno Code Composer Studio para la tarjeta de desarrollo eZdsp basada en el DSP C5535 de Texas Instruments. Para comprobar el funcionamiento se crean gráficas de magnitud FFT con la ayuda del depurador integrado en el propio entorno de desarrollo. Los periféricos integrados en la DSP son utilizados con el objetivo de aplicar en tiempo real los filtros previamente diseñados y probados en la GUI.

1.2. Justificación

Los filtros son ampliamente utilizados en la industria musical. No es un secreto que muchos de los instrumentos y herramientas utilizadas en los estudios de audio incorporan filtros que son indispensables para el tratamiento acústico de las grabaciones. La limpieza de ruido, la separación de instrumentos e incluso el análisis mismo de bandas espectrales específicas no sería posible sin los filtros.

Los ecualizadores son un gran ejemplo de la aplicación de los filtros digitales en la música. Un ecualizador es un dispositivo que modifica la amplitud de las señales de audio en función de su frecuencia, por lo que aumenta o atenúa la amplitud de las señales en determinadas frecuencias, logrando la modificación de la respuesta en frecuencia. En [1] se propone la implementación en hardware y software utilizando algoritmos de filtrado digital en una Tarjeta Demo Propeller de la compañía Parallax a través de un sistema basado en algoritmos de filtros digitales FIR. A pesar de que los ecualizadores son una herramienta propiamente utilizada en estudios musicales, el enfoque de este proyecto es el de aportar ayudas técnicas que faciliten la inclusión social de las personas con discapacidad.

En medicina los filtros de audio son utilizados para el diagnóstico de enfermedades de forma no invasiva. Por ejemplo, en 2016, el trabajo titulado Separación de fuentes sonoras emitidas por el corazón y pulmones aplicado al diagnóstico de enfermedades cardiovasculares y respiratorias, se explora la posibilidad de aumentar la exactitud de un buen diagnóstico por parte del especialista en medicina utilizando filtros FIR para distinguir los datos captados por el mismo receptor. [2]

En la Universidad Nacional de San Martín en Argentina se planteó el desarrollo de un sistema de audio que sea compatible con el Resonador Magnético (RM) de 3 Tesla del Centro Universitario de Imágenes Médicas (CEUNIM). A grandes rasgos, el objetivo del proyecto es diseñar un método para la captura de la voz de un paciente sometido a diagnóstico en el RM. Dado que los resonadores magnéticos generan un alto nivel de ruido acústico durante la adquisición de las imágenes, es necesario proveer de auriculares de gran aislación acústica al paciente para la escucha de los estímulos sonoros además de desarrollar un sistema para la captura de la señal de voz, para ello se debe realizar un adecuado procesamiento digital

con el fin de hacerla inteligible para el profesional que evaluará las imágenes. [3]

Entendiendo la importancia de los filtros de audio surge la siguiente pregunta ¿Cuáles son los mejores?. La respuesta corta es que depende de la aplicación aunque es cierto que el procesamiento de señales mediante el uso de filtros digitales ofrece una serie de ventajas con respecto al procesamiento en el dominio analógico. Entre los beneficios, se puede destacar su fácil ajuste a diferentes condiciones, esto implica que las características del filtro se pueden modificar completamente mediante el cambio de una variable en un programa. Este ajuste resulta más costoso en el dominio analógico, ya que implica un cambio en componentes hardware (en ocasiones topológico), sin mencionar el recurrente uso de componentes de precisión. Los filtros digitales, al ser un software, también son más estables a lo largo del tiempo y no sufren con las variaciones externas como la temperatura. De igual manera es mas sencilla la replicabilidad del filtro digital ya que el mismo programa puede ser copiado y ejecutado en otro hardware.

En particular el empleo de filtros FIR permite procesar señales sin distorsión en la fase, lo cual es muy adecuado para procesar señales de audio.

Existe otra ventaja de los filtros FIR con respecto a su contraparte IIR, la cuál es que presenta una pendiente mas pronunciada en la banda de transición. Esto permite obtener un mejor desempeño del filtro con un orden menor (o lo que es lo mismo, menos coeficientes) lo que implica menor procesamiento.

El presente proyecto presenta una herramienta propuesta para el diseño de filtros digitales FIR a través de una GUI, la cual fue desarrollada en Matlab. Además se describe la implementación de los filtros FIR en una tarjeta basada en un DSP. Con periféricos para el procesamiento de señales de audio.

1.3. Objetivo General

Diseñar en MATLAB e implementar físicamente filtros digitales FIR para el procesamiento de señales de audio en aplicaciones específicas.

1.4. Objetivos Específicos

- Investigar sobre el concepto de procesamiento digital en particular mediante filtros FIR.
- Elegir el tipo de procesamiento a realizar sobre las señales de audio.
- Diseñar una interfaz GUI en MATLAB para realizar el procesamiento de señales de audio.
- Realizar pruebas del sistema en MATLAB.
- Conocer las características de la tarjeta DSP a utilizar.
- Llevar a cabo la implementación de filtros digitales en un sistema basado en un DSP para realizar procesamientos seleccionados.
- Realizar pruebas experimentales.

Capítulo 2

Marco Teórico

2.1. Definición de términos

Con al finalidad de establecer las bases del desarrollo del presente trabajo, a continuación, se definen brevemente los términos más importantes.

2.1.1. Audio Digital

Hablar de audio digital, desde el punto de vista de la ingeniería, es hablar de muestreo y reconstrucción de señales. Una señal es la representación de una cantidad física y debido a que la mayoría de las señales son naturalmente continuas en magnitud y tiempo (incluyendo las señales de audio), para procesar las señales primero deben ser convertidas a un formato digital. A este proceso se le llama conversión Analógico-Digital (ADC). [4]

Una cantidad física, como la presión del sonido, se mide y se representa en su formato analógico, como una señal de voltaje. La señal de voltaje $v(t)$ puede tomar cualquier valor en cualquier momento. Para registrar con precisión segmentos de longitud finita de $v(t)$ con datos, se deben almacenar una cantidad infinita de datos para cada ínfimo valor de amplitud y tiempo. En los sistemas digitales no se cuenta con almacenamiento ilimitado y mucho menos se tiene un poder de procesamiento infinito que pueda trabajar con tal cantidad de datos.

La conversión Analógico-Digital, comúnmente conocida como digita-

lización, involucra los procesos de muestreo (digitalización de tiempo) y cuantificación (digitalización de amplitud)[5]. Estos procesos nos permiten establecer un rango de valores discretos en tiempo y amplitud con los que se puede trabajar digitalmente.

El muestreo consiste en tomar el valor de la señal que se quiere digitalizar cada determinado tiempo, a este tiempo se le conoce como periodo de muestreo (t_s). El inverso del periodo de muestreo es la frecuencia de muestreo (f_s) y es importante que se respete el teorema de Nyquist al momento de establecerlo. El teorema que demostró Nyquist en 1924 nos dice que basta con solo dos muestras por ciclo para que se pueda recuperar la señal original [6], en otras palabras, la frecuencia f_s debe ser al menos dos veces la señal máxima presente en la señal que se desea muestrear f_m , en la Ecuación 2.1 se representa el teorema de Nyquist.

$$f_s \geq 2f_m \quad (2.1)$$

La cuantificación nos permite establecer un conjunto discreto de valores que la señal puede tomar en cada instante de tiempo t_s . El conjunto de valores que se puede utilizar está dado por la resolución del ADC y la resolución que posea, si la resolución es de n bits, al tratarse de un sistema digital, los posibles valores son 2^n . Al reemplazar los valores continuos por los valores cuantificados introduce el llamado error de cuantificación o ruido de cuantificación, que tiende a disminuir cuanto mayor es la resolución del ADC empleado.[7]

Adicionalmente, se podría hablar de la codificación, que básicamente es un proceso extra que se realiza en los sistemas digitales para almacenar los datos muestreados. Para el desarrollo de este trabajo la codificación utilizada es la más básica, la codificación PCM (modulación por impulsos codificados, por sus siglas en inglés), que consiste en codificar los niveles en binario y la transmisión se realiza secuencialmente en un periodo de muestreo establecido.

2.1.2. Transformada de Fourier

Una de las herramientas mas importantes para desarrollar este trabajo es la transformada de Fourier. La Transformada de Fourier es una operación matemática utilizada para transformar una señal entre el dominio

del tiempo y el dominio de la frecuencia. Al aplicar la transformada de Fourier a una señal podemos observar características que vistas desde el dominio de tiempo son imperceptibles. Nos permite entender la composición de las señales desde un enfoque diferente; Por ejemplo, una señal de audio puede ser descompuesta en sus componentes espectrales individuales si le aplicamos la transformada de Fourier.

La Transformada de Fourier, como tal, es de naturaleza continua pero también existe una alternativa que puede ser aplicada en los sistemas no continuos llamada Transformada Discreta de Fourier o DFT.

La DFT de una señal discreta $f(n)$ de N muestras esta dada por la Ecuación 2.2. [8]

$$F(\omega) = \sum_{n=0}^{N-1} f(n)e^{-j\omega n} \quad (2.2)$$

Para fines del desarrollo del presente trabajo, la DFT nos servirá para conocer el espectro de frecuencias contenidas en una serie de muestras de audio. Mas adelante, en la sección de los filtro se entenderá la importancia de la Transformada de Fourier como método de comprobación y diseño para los filtros.

2.1.3. Transformada de Fourier de tiempo reducido

La Transformada de Fourier de Tiempo Reducido o STFT es un caso especial de la DFT que nos permite estudiar el espectro de frecuencia de secciones locales de un conjunto de muestras. La STFT se puede entender si suponemos que estamos escuchando una pieza musical en la que a lo largo del tiempo los instrumentos que suenan cambian. Por ejemplo, podemos escuchar trompetas al inicio de una canción, en otro instante se agregan percusiones y al final podemos escuchar solo una flauta. Al analizar el audio desde el punto de vista de la Transformada de Fourier, podríamos entender que la distribución de potencias del espectro de frecuencias es la misma en todo momento. La STFT nos permite un mejor entendimiento de como evoluciona el espectro de frecuencias a lo largo del tiempo ya que, básicamente, se dividen el total de muestras en conjuntos de muestras mas pequeños. [9]

Usualmente, la STFT se suele presentar en gráficas llamadas espectrogramas. El espectrograma es un tipo de gráfica tridimensional que se apoya del uso de colores para representar la cantidad de energía en una sección local. En la Figura 2.1 se muestra un ejemplo de espectrograma. En el eje vertical se tiene el rango de frecuencias y en el horizontal el tiempo; La barra de la derecha nos ayuda a entender la energía que hay de cada frecuencia en cada instante de tiempo, siendo azul la energía mas baja y amarillo la máxima energía.

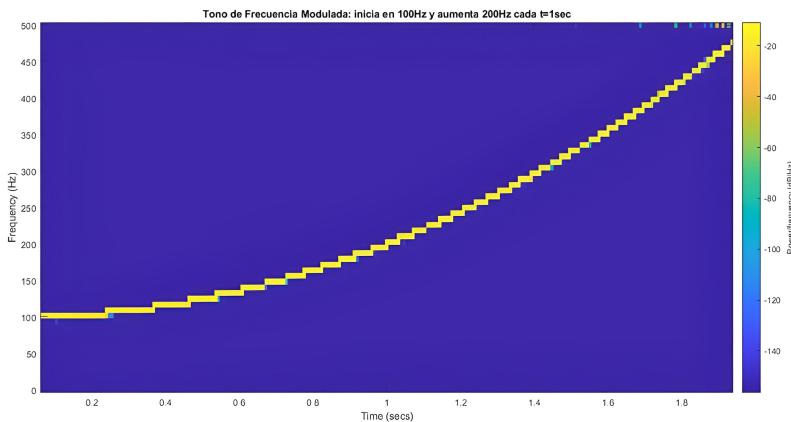


Figura 2.1: Ejemplo de Espectrograma.

2.1.4. Filtros Digitales

En su definición más general, un filtro puede definirse como cualquier procesamiento que altera las propiedades de una señal de sonido de una forma u otra. Un filtro digital es un proceso computacional o algoritmo mediante el cual una señal digital (una secuencia de muestras) se transforma en una segunda secuencia de muestras o una señal de salida digital.

Los filtros se utilizan ampliamente en todas las áreas del procesamiento de señales, forman la base del procesamiento de señales y se pueden aplicar a todo tipo de señales (sonido, imagen, vídeo, vibración, etc.). En el campo

de las señales de audio, definimos específicamente los filtros como objetos que alteran el contenido espectral o de frecuencia de una señal. De ahí su importancia fundamental en la música electroacústica. Los filtros se utilizan en realidad en una variedad de situaciones musicales, ya sea para la modificación radical de señales sintetizadas o para la localización espectral de grabaciones instrumentales. [10]

Los filtros digitales pueden clasificarse en 2 grandes clases, los filtros FIR (Finite Impulse Response) y los filtros IIR (Infinite Impulse Response). La diferencia entre estos tipos radica principalmente en la estructura de retroalimentación lo que genera salidas infinitamente largas al excitar el sistema con un pulso. La única forma de que una respuesta de impulso dure indefinidamente es si la salida del sistema se conecta de alguna manera al sistema mismo para generar una recursión, en cada nuevo momento la salida está determinada por la salida anterior. En pocas palabras la estructura de los filtros IIR está retroalimentada y la de los filtros FIR no.

2.1.5. Filtros FIR

Los filtros FIR (Finite Impulse Response) deben su nombre a que cuando la señal de entrada es un impulso, la salida tendrá un número finito de términos no nulos. En el caso de este tipo de filtros solo se toman en cuenta las entradas actuales y anteriores, nunca las salidas como en el caso de los IIR. Su expresión matemática en el dominio n es la que se muestra en la Ecuación 2.3, para la cual N es el orden del filtro y b_k son los coeficientes del filtro.

$$Y_n = \sum_{k=0}^{N-1} b_k X(n - k) \quad (2.3)$$

La estructura básica de los filtros FIR se muestra en la Figura 2.2

Una de las características que hacen que los filtros FIR sean útiles para las aplicaciones de audio y por la que fue elegido en este trabajo es que son de fase lineal. Esto significa que la forma de onda no se ve alterada al aplicar el filtro, característica muy útil para no evitar distorsionar las señales de audio.

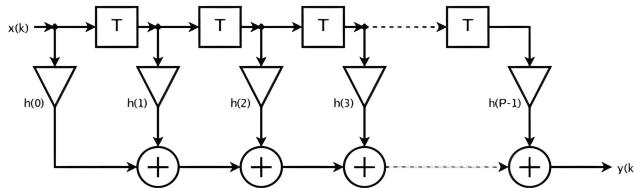


Figura 2.2: Estructura básica de los filtros FIR.

2.1.6. Diseño de Filtros

Para el diseño de filtros FIR existen básicamente 3 métodos principales:

1. Técnica de Ventanas o Aventanado.
2. Técnica de Muestreo en Frecuencia.
3. Técnica de diseño con Rizado Uniforme o Parks-McLellan.

Los métodos de diseño elegidos para ser aplicados en este trabajo fueron el método de Aventanado y el de Parks-McLellan. El método de Aventanado nos permite establecer el orden del filtro, ya que para algunas aplicaciones es importante tener control sobre este parámetro; por otro lado, el método de Parks-McLellan es capaz de generar un filtro con el cual la amplitud en la frecuencia de paso y rechazo se mantenga siempre en un rango de tolerancia arbitrario. Este par de métodos nos permiten abarcar muchos de los escenarios posibles que se llegan a presentar al momento de diseñar un filtro.

2.1.7. DSPs

Los sistemas DSP necesitan realizar operaciones aritméticas intensivas como la multiplicación y la suma. Estas tareas se pueden realizar en microprocesadores, microcontroladores, procesadores de señales digitales o circuitos integrados personalizados. Elegir el hardware correcto depende de la aplicación, el costo, o una combinación de ambos. Existen 5 clases de hardware que puede ser empleado como sistema DSP: [5]

1. Chips de propósito general (personalizados).
2. Matriz de compuertas programables (FPGA).
3. Microprocesadores o Microcontroladores de propósito general(μ P/ μ C).
4. Procesador digital de señales de propósito general.
5. Procesador digital de señales de propósito específico con aceleradores de hardware.

Un procesador digital de señales tiene los periféricos necesarios para muestrear señales analógicas, en ocasiones, los periféricos son especiales para áreas determinadas como el procesamiento de audio. Un DSP tiene la potencia, el hardware y las instrucciones requeridas para implementar y probar filtros digitales. En el desarrollo de este proyecto se decidió utilizar este tipo de hardware por los beneficios mencionados.

2.2. Software y hardware empleado

A continuación, se hace una revisión de las herramientas utilizadas para el diseño e implementación de los filtros del presente trabajo.

2.2.1. Matlab

MATLAB es la abreviatura de "MATriz LABoratory". Es un programa que realiza cálculos numéricos usando vectores y matrices, por lo que también puede manejar números escalares (reales y complejos), cadenas y otras estructuras de información más complejas. Matlab es un lenguaje de alto rendimiento para computación técnica que es tanto un entorno como un lenguaje de programación. Una de sus ventajas es que nos permite construir nuestras propias herramientas reutilizables. Podemos crear fácilmente nuestras propias funciones y programas especiales en código de Matlab, y podemos agruparlos en cajas de herramientas. Además contiene colecciones especializadas de archivos M para tratar clases específicas de problemas. Matlab, además de cálculo matricial y álgebra lineal, también

puede manejar polinomios, funciones, ecuaciones diferenciales ordinarias, gráficos, etc. [11]

La plataforma de MATLAB tiene funciones especializadas para el diseño de filtros. Cuenta con funciones que se pueden utilizar desde el workspace y también cuenta con una aplicación llamada filterDesigner. filterDesigner es una interfaz de usuario desde la que se pueden diseñar, analizar y modificar filtros; los filtros diseñados se pueden exportar tanto al workspace del propio Matlab como fuera de él en archivos de texto o de tipo header para otros lenguajes de programación, por ejemplo C.

MATLAB fue elegido como la herramienta de diseño principal para este proyecto. Se aprovechó la herramienta de diseño de filtros que tiene integrada el ambiente de MATLAB y también se hace uso de las prestaciones que tiene MATLAB para crear interfaces de usuario. En MATLAB se desarrolló una interfaz de usuario inspirada en su propia aplicación de diseño de filtros, la diferencia principal entre la herramienta integrada y la desarrollada en este trabajo se observa al momento de comprobar los filtros diseñados en piezas de audio.

2.2.2. Code Composer Studio

El software Code Composer Studio es un entorno de desarrollo integrado (IDE) compatible con la cartera de microcontroladores (MCU) y procesadores integrados de Texas Instruments. El software Code Composer Studio incluye un conjunto de herramientas para desarrollar y depurar aplicaciones integradas. Incluye un compilador optimizado de C/C++, un editor de código fuente, un entorno de creación de proyectos, un depurador, un analizador y muchas otras características. El IDE es intuitivo ya que proporciona una interfaz de usuario única que guía a través de cada paso del proceso de desarrollo de aplicaciones. El software Code Composer Studio combina los beneficios del marco del software Eclipse con las capacidades de depuración integradas avanzadas de TI para proporcionar a los desarrolladores un entorno de desarrollo atractivo y rico en funciones. [12]

Code Composer Studio nos vale como el ambiente de programación del microcontrolador elegido para implementar los filtros diseñados en MATLAB. Se aprovecha el depurador incluido en Code Composer Studio y la herramienta de análisis frecuencial incluido. Este entorno de programación nos

permite escribir el código y probarlo de tal forma que no se necesita mas equipo que el incluido en el software, por lo menos para lo que se pretende en este trabajo.

2.2.3. C5535 eZdsp

La C5535 eZdsp es una herramienta de pruebas para la TMS320C5535 Digital Signal Processor (DSP). Las características principales de la tarjeta utilizada en este proyecto son las siguientes: [13]

- TMS320C5535 DSP de Texas Instrument
- Texas Instruments TLV320AIC3204 Stereo Codec (entrada estéreo, salida estéreo)
- Ranura Micro SD
- Interfaz USB 2.0 - C5535
- 8 Mbytes Memoria SPI flash
- I2C OLED display
- 5 Leds
- 2 botones pulsadores
- Emulador embebido USB XDS100 JTAG
- Conector de Expansión
- Puntos de Medición de Alimentación
- Alimentación por Interfaz USB
- Compatible con Code Composer Studio v4

El códec estéreo integrado, la interfaz USB y el depurador embebido que integra la tarjeta la convierten en una excelente herramienta para la implementación y prueba de filtros digitales con enfoque al audio.

En la Figura 2.3 se muestra físicamente el frontal de la tarjeta. Se resalta el conector USB integrado y los conectores para entrada y salida de audio estéreo de 3.5mm ya que son los periféricos principales a los que accedemos en el desarrollo del proyecto.

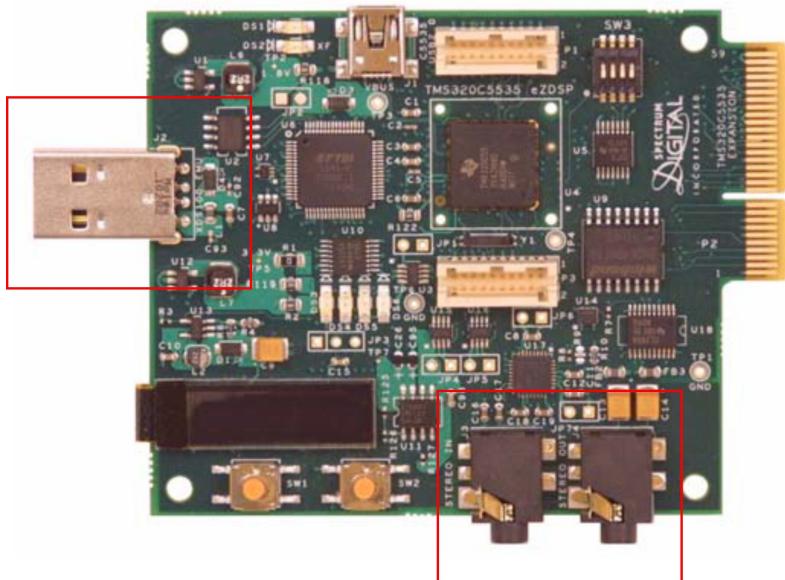


Figura 2.3: C5535 eZdsp.

Capítulo 3

Diseño e Implementación de Filtros FIR

3.1. GUI

La interfaz gráfica fue diseñada para tener todas las herramientas necesarias para el diseño y prueba de filtros FIR aplicados al audio. Se incluyeron dos maneras de leer archivos desde diferente fuente, dos métodos de diseño de filtros, una gráfica para la respuesta en frecuencia del filtro, una vista en tiempo del audio de entrada, una vista en tiempo del audio filtrado, una gráfica del espectro de frecuencia para el audio de entrada y una para el audio filtrado; tanto el audio de entrada como el audio de salida se pueden reproducir desde la propia interfaz; los coeficientes del filtro se pueden exportar en un archivo de cabecera para el lenguaje C.

La interfaz gráfica fue dividida en 3 secciones en las que se agrupan todas las opciones para generar, probar y exportar los coeficientes de los filtros. En la Figura 3.1 se muestra la ventana principal de la interfaz. El punto numero uno es el panel Archivo que contiene las funciones para cargar un archivo desde la computadora o grabar desde el micrófono; el punto numero dos es el panel que nos ayuda a diseñar el filtro y exportar sus coeficientes; El panel Gráficas, marcado con el número tres, presenta las vistas gráficas de los audios originales y filtrados, además contiene las funciones para reproducir cada uno de ellos.

16 CAPÍTULO 3. DISEÑO E IMPLEMENTACIÓN DE FILTROS FIR

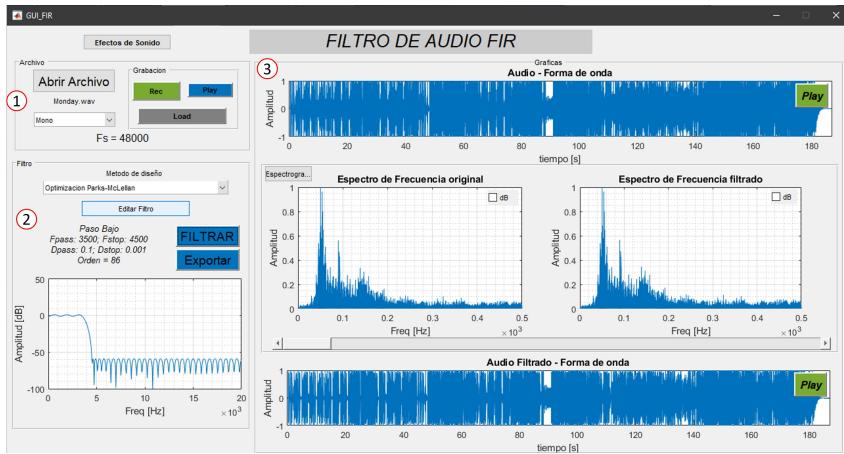


Figura 3.1: Ventana principal de interfaz de usuario.

3.1.1. Archivo

En esta sección se tienen las opciones que nos permiten leer o grabar un archivo de audio con el que trabajar. En la Figura 3.2 se observan los diferentes botones y menús del panel Archivo.



Figura 3.2: Panel Archivo.

La aplicación es capaz de navegar a través de los directorios del ordenador para leer un archivo de audio con extensión .wav. Para hacer uso de esta función presionamos en el botón Abrir, marcado con el numero uno en el panel Archivo. Se abrirá un nuevo menú para navegar y seleccionar el archivo que se importará. La GUI almacena las muestras contenidas en el

archivo y la frecuencia de muestreo se puede ver en la visualización número dos en el panel Archivo. Se añadió la opción para que el usuario pueda indicar el tipo de paneo que se realiza una vez leído el archivo. Las opciones se muestran al presionar en el menú desplegable con el numero tres del panel Archivo.

- Mono - Suma las muestras de ambos canales y las divide entre dos
- Mono L - Toma solo las muestras del canal izquierdo
- Mono R - Toma solo las muestras del canal derecho

En el panel Archivo se tiene un sub-panel que contiene los botones del cuatro al seis para grabar desde el microfono del ordenador, escuchar el audio grabado y cargar el audio en el programa, respectivamente.

3.1.2. Filtro

En este panel se agrupan las funciones para el diseño del filtro. En la Figura 3.3 se observa el panel con todos los recursos a los que se tiene acceso en él. La selección del método de diseño (incluyendo los parámetros del filtro), la gráfica de la respuesta en frecuencia del filtro, así como los botones para aplicar el filtro al audio cargado y para exportar el archivo que contiene los coeficientes del filtro, son funciones que se encuentran en el panel Filtro.

Para el diseño de los filtros en la GUI se programó un menú desplegable mediante el cuál se selecciona uno de los dos métodos de diseño de filtros FIR disponibles, Aventanado o Parks-McLellan. Cuando se elige el método, presionando en el menú señalado con el numero uno del panel Filtro, se debe llenar un formulario con los parámetros que dependen del método seleccionando.

Si se requieren cambiar los parámetros del filtro manteniendo el método de diseño se debe presionar el botón con el número dos del panel Filtro, esto redesplegará las ventanas correspondientes al formulario del método de diseño.



Figura 3.3: Panel Filtro.

Teniendo los parámetros del filtro se debería poder visualizar la respuesta en frecuencia del filtro diseñado en la gráfica numero tres del panel Filtro. Al mismo tiempo, los datos de diseño del filtro se estarán mostrando en el monitor con el numero cuatro del panel Filtro.

El botón número cinco, al ser presionado, aplica el filtro al audio original.

Para exportar los coeficientes del filtro se presiona el botón número seis del panel, posteriormente se abrirá una nueva ventana en la que se debe escribir el nombre y la dirección donde se guardará el documento creado. El documento consta de 4 partes: contiene la directiva para incluir la librería tmwtypes.h que define tipos de datos necesarios para la ejecución; incluye la definición de una constante int16, de nombre BL, igual al orden del filtro más uno; incluye un arreglo int16, llamado B, de tamaño igual al orden del filtro más uno que contiene los valores de los coeficientes del filtro; la ultima parte define un macro nombrado BUFFER_SIZE que será utilizado mas adelante por el DSP para declarar variables. En la Figura 3.4 se puede ver un ejemplo de un archivo generado desde la GUI para un filtro diseñado por el método de Aventanado, la ventana elegida es Barlett, el filtro es de orden 32, pasa bajas con Fc en 4000Hz.

```
#include "tmwtypes.h"
const int BL = 33;
const int16 T B[33] = {0, 47, 68, 62,
0, -162, -369, -553, -615, -457, 0, 782, 1846, 3079, 4300, 5330,
5963, 5330, 4308, 3079, 1846, 782,
0, -457, -615, -553, -369, -162, 0, 62, 68, 47, 0};
#define BUFFER_SIZE 33
|
```

Figura 3.4: Ejemplo de archivo de exportación.

3.1.3. Gráficas

Para poder corroborar el funcionamiento de los filtros diseñados se agregaron gráficas y funciones de reproducción de audio en este panel. En la Figura 3.5 se observa dicho panel y sus partes son detalladas a continuación.

20 CAPÍTULO 3. DISEÑO E IMPLEMENTACIÓN DE FILTROS FIR

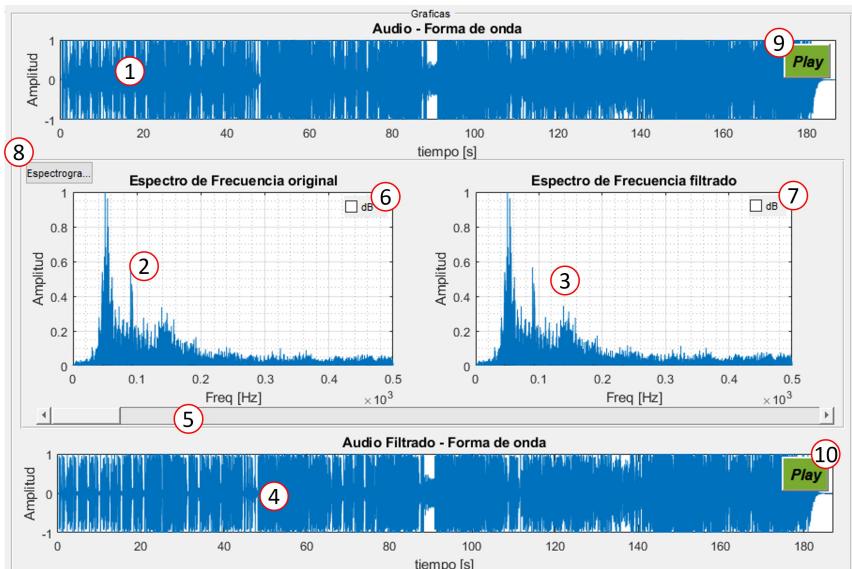


Figura 3.5: Panel Gráficas.

Cuando un nuevo archivo de audio es cargado en la aplicación a través del panel Archivo, la forma de onda se muestra en la gráfica número uno del panel Gráficas, al mismo tiempo se calcula la distribución de frecuencias de dicho audio desde 0 hasta $F_s/2$ y se muestra en la gráfica número dos.

La gráfica número tres muestra el espectro de frecuencia del audio al que se le aplica el filtro diseñado en el panel Filtro. En la gráfica cuatro se ve la forma de onda del audio filtrado.

Adicionalmente, se puede elegir visualizar la ganancia en unidades de decibeles(dB) simplemente marcando las casillas marcadas con los puntos seis y siete.

El punto cinco es una barra para desplazarnos a lo largo del espectro de frecuencias de las señales (las gráficas dos y tres), si la barra es desplazada completamente a la derecha, en las gráficas se puede ver el rango completo de frecuencias como se muestra en la Figura 3.6.

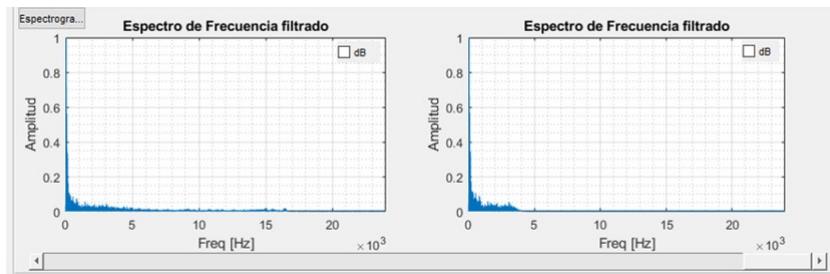


Figura 3.6: Ejemplo. Visualización de espectro completo.

Al presionar el botón con el número ocho, las gráficas dos y tres son remplazadas por el spectrograma del audio original y filtrado respectivamente, Figura 3.7. Se puede alternar entre estos dos modos de visualización presionando nuevamente sobre el botón.

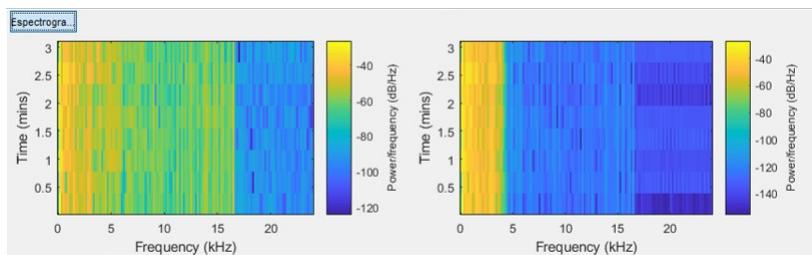


Figura 3.7: Ejemplo. Visualización de Espectrograma

Es posible reproducir los audios para comprobar auditivamente el resultado del filtro aplicado en las pistas. Al presionar sobre el botón nueve o diez estaremos escuchando el audio original y el audio filtrado, respectivamente. Presionando nuevamente sobre estos botones estaremos deteniendo la reproducción del audio.

3.2. Implementación de Filtros en DSP

Implementar un filtro FIR es sencillo si se conocen los principios con los que trabajan. A grandes rasgos lo que debemos hacer desde la tarjeta DSP que seleccionamos es encontrar la manera de programar la Ecuación 2.3 vista en la sección 2.1.5. A continuación se realiza una revisión del proceso que se lleva a cabo dentro de la DSP.

Lo primero que se hace en el programa es la configuración de los registros que hacen funcionar a la tarjeta. Para los primeros pasos se configura una frecuencia de muestreo de 48kHz. El codec de audio se configura utilizando funciones integradas en el entorno de desarrollo. [14]

Para importar el archivo que se describe al finalizar la Sección 3.1.2 basta con tener el archivo generado desde la GUI en la misma carpeta desde la que se ejecuta el programa para la DSP. Se agrega la linea `#include "coefs.h"` y tendremos disponible para usar en el programa la variable BL (tamaño del filtro) y B (coeficientes del filtro).

Una vez que se tiene la tarjeta configurada y el archivo de cabecera incluido en el programa, se realiza la lectura de los puertos de entrada de audio. Se utiliza la función `EZDSP5535_I2S_readLeft(lAddr)` y `EZDSP5535_I2S_readRight(rAddr)` para leer el canal izquierdo y el canal derecho del codec de audio. El argumento que necesita es una dirección en la que se almacenaran las muestras, este argumento se pasa mediante los apuntadores que en este caso son nombrados lAddr y rAddr.

De la Figura 2.2 se puede deducir que cada una de las muestras de salida es consecuencia de realizar una suma algebraica del producto entre las n muestras anteriores a la muestra actual y el correspondiente coeficiente (del filtro); por lo tanto, es necesario almacenar al menos n muestras de entrada, donde n es el orden del filtro. Se creó una función que desplaza los valores cada vez que se lee uno nuevo. En la Figura 3.8 se muestra la función shift, el argumento loop esta definido por el orden del filtro y `*x` es la dirección de inicio de la tabla en la cual se ordenan las muestras.

```

void shift (Int16 *x, int loop) {
    int i; /* Loop index */
    for(i=loop-1; i> 0; i--) {
        x[i] = x[i-1]; /* Shift old data x(n-i) */
    }
}

```

Figura 3.8: Función Shift.

Para aplicar el filtro se debe realizar una convolución entre los coeficientes del filtro y la tabla de muestras. Se creó una función para realizar el proceso. La función se llama fir_convolution y recibe como argumento la dirección inicial de la tabla donde se almacenan las muestras. En la Figura 3.9 se muestra como fue construida y también se aprecia como desde la misma función se lee el arreglo B, que contiene los coeficientes del filtro; del mismo modo, el tamaño del arreglo se lee desde la constante BL.

```

Int16 fir_convolution (Int16 *x) {
    Int32 yn = 0; /* Output of FIR filter */
    int i; /* Loop index */

    for(i=0; i<BL; i++) {
        yn += (Int32)B[i]*(Int32)x[i];/* Convolution of x(n) with h(n) */
    }
    yn = yn>> 15;
    return((Int16)yn);
}

```

Figura 3.9: Función Convolución.

La función fir_convolution devuelve la muestra que se escribirá en el canal de salida del codec correspondiente al canal que se muestreó. Para escribirlo se utiliza la función EZDSP5535_I2S_writeRight(lOut) y EZDSP5535_I2S_writeLeft(rOut) a las que se les da como argumento la salida respectiva a cada canal.

Capítulo 4

Resultados

4.1. GUI

Aunque el programa inicialmente fue diseñado para tener una frecuencia de muestreo de 48kHz, las pruebas nos hicieron considerar una reducción de este valor a poco menos de la mitad, los motivos son descritos en la sección dedicada a las pruebas con la DSP.

De la Figura 4.1 hasta la Figura 4.8 se muestran imágenes del diseño de 4 filtros hechos desde la interfaz GUI desarrollada en el presente documento: Se diseñó un filtro de cada tipo, pasa bajas, pasa altas, pasa banda y rechazo banda.

El filtro pasa bajas de las Figuras 4.1 y 4.2 fue diseñado por el método de Parks-McLellan, al igual que el filtro pasa banda de las figuras 4.5 y 4.6. El filtro pasa altas de las Figuras 4.3 y 4.4 junto con el filtro rechazo banda de las Figuras 4.7 y 4.8 fueron diseñados por el método de Aventanado.

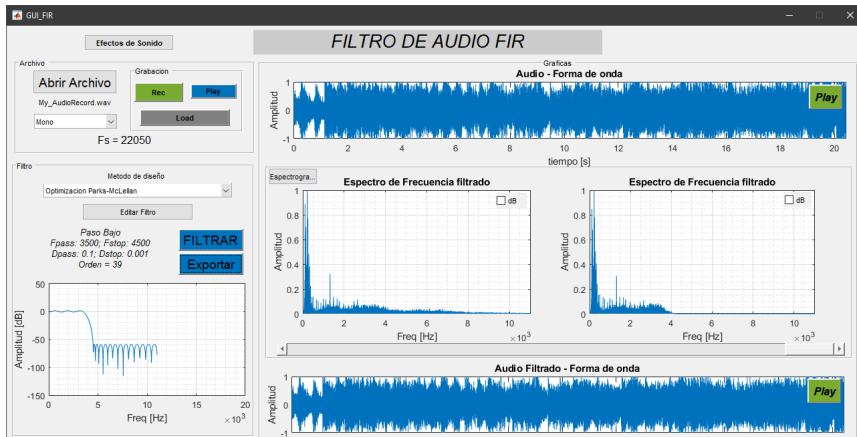


Figura 4.1: **Método:** Parks-McLellan; **Filtro:** Pasa Bajas; **Fc:** $F_{pass} = 3500\text{Hz}$, $F_{stop} = 4500\text{Hz}$; **Orden:** 39; **Vista:** FFT

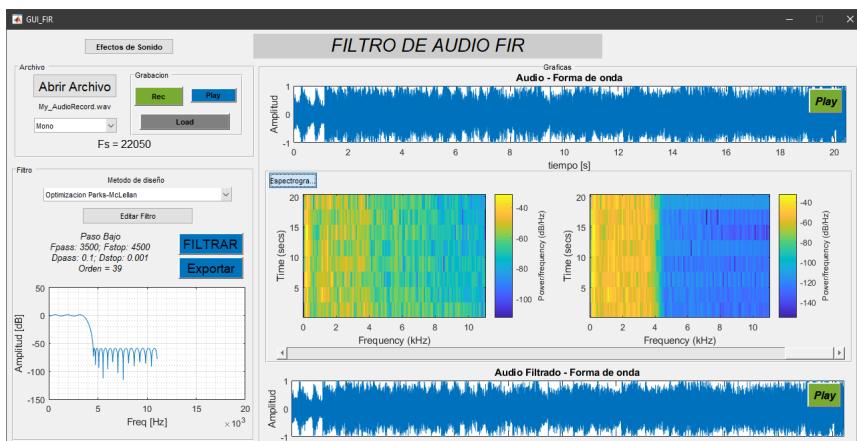


Figura 4.2: **Método:** Parks-McLellan; **Filtro:** Pasa Bajas; **Fc:** $F_{pass} = 3500\text{Hz}$, $F_{stop} = 4500\text{Hz}$; **Orden:** 39; **Vista:** STFT

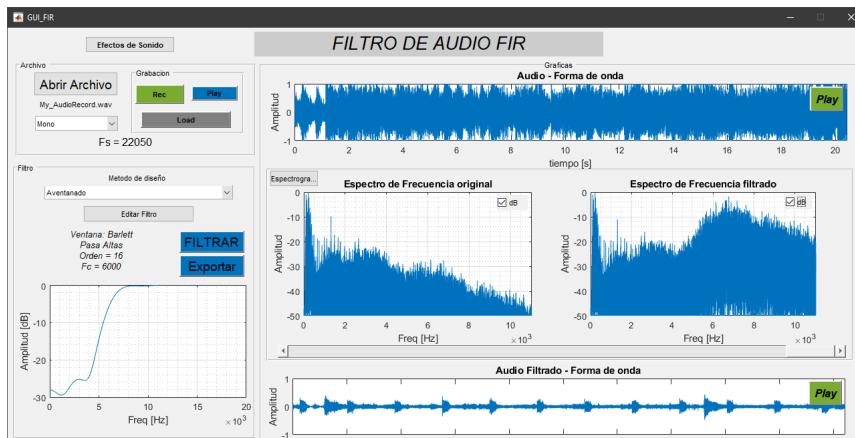


Figura 4.3: **Método:** Aventanado; **Filtro:** Pasa Altas; **Fc:** 6000Hz; **Orden:** 16; **Vista:** FFT

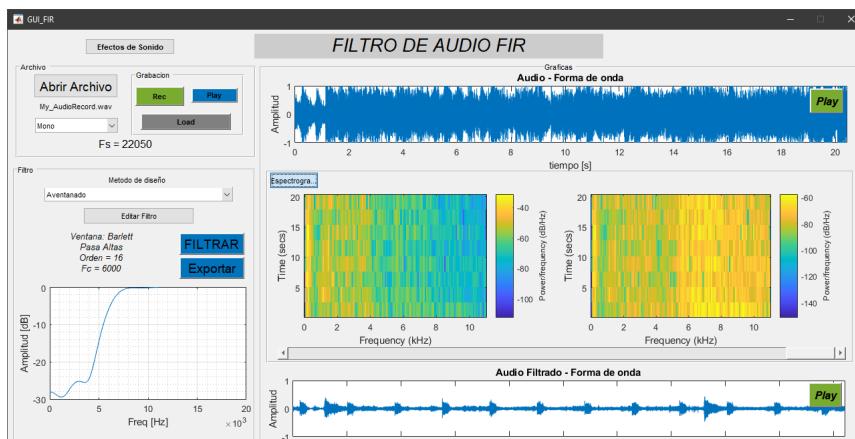


Figura 4.4: **Método:** Aventanado; **Filtro:** Pasa Altas; **Fc:** 6000Hz; **Orden:** 16; **Vista:** STFT

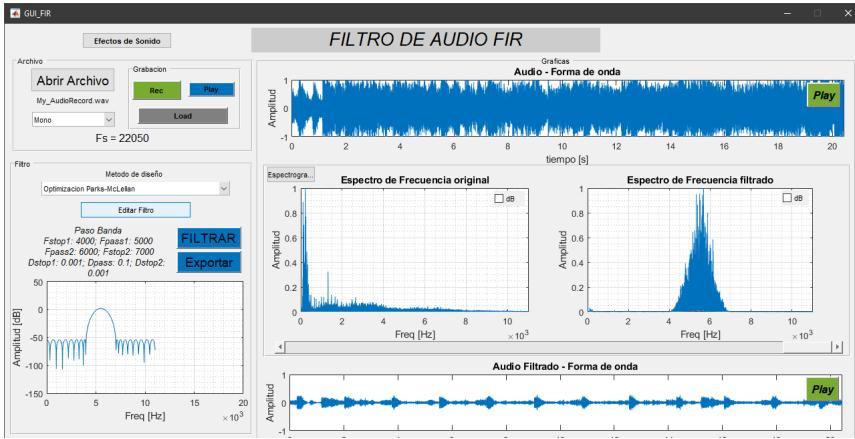


Figura 4.5: **Método:** Parks-McLellan; **Filtro:** Pasa Banda; **Fc:** $F_{pass1} = 4000\text{Hz}$, $F_{stop1} = 5000\text{Hz}$, $F_{pass2} = 6000\text{Hz}$, $F_{stop2} = 7000\text{Hz}$; **Orden:** 39; **Vista:** FFT

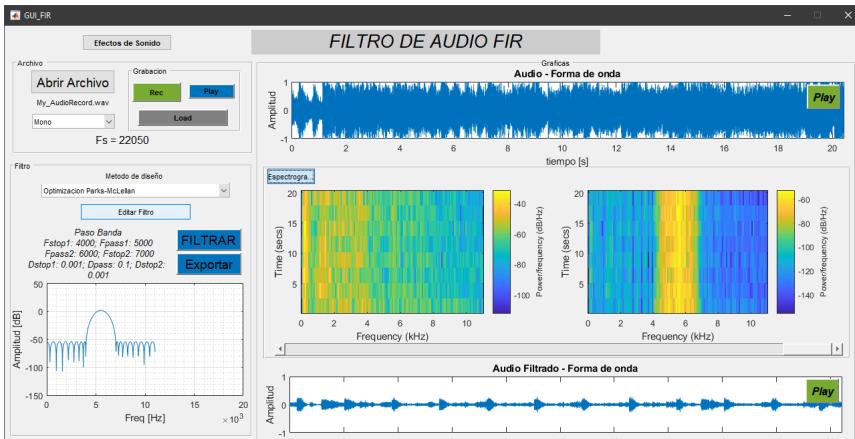


Figura 4.6: **Método:** Parks-McLellan; **Filtro:** Pasa Banda; **Fc:** $F_{pass1} = 4000\text{Hz}$, $F_{stop1} = 5000\text{Hz}$, $F_{pass2} = 6000\text{Hz}$, $F_{stop2} = 7000\text{Hz}$; **Orden:** 39; **Vista:** STFT

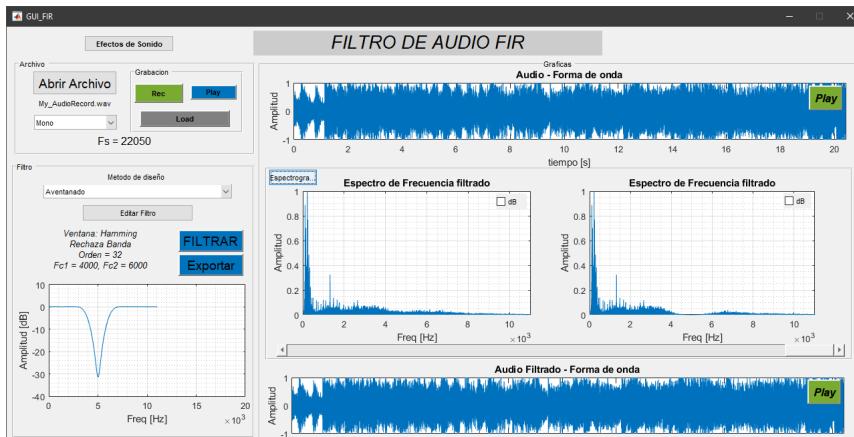


Figura 4.7: **Método:** Aventanado; **Filtro:** Rechazo banda; **Fc:** $F_{c1} = 4000\text{Hz}$, $F_{c2} = 6000\text{Hz}$; **Orden:** 32; **Vista:** FFT

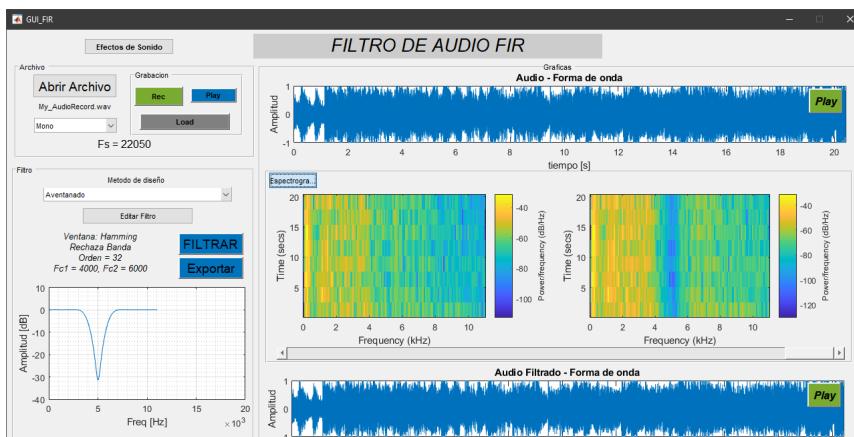


Figura 4.8: **Método:** Aventanado; **Filtro:** Rechazo banda; **Fc:** $F_{c1} = 4000\text{Hz}$, $F_{c2} = 6000\text{Hz}$; **Orden:** 32; **Vista:** STFT

4.2. DSP

Para comprobar el funcionamiento de los filtros en la DSP se utilizó el depurador incorporado en la versión 4 de Code Composer Studio y con ayuda de la herramienta para graficar la magnitud de la FFT que se muestra en la Figura 4.9 se pudo comprobar que cada uno de los filtros previamente diseñados cumplan con su propósito.

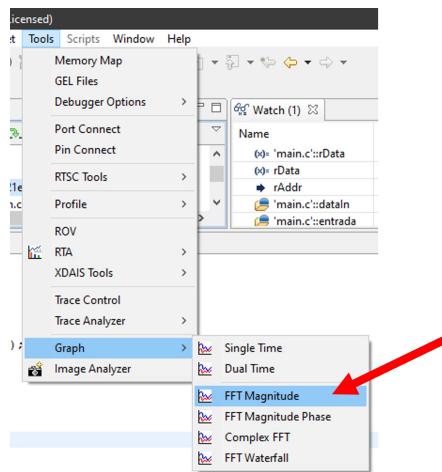


Figura 4.9: Herramienta FFT en CCS

Graficar con el depurador de CCS requiere agregar algunas líneas extras de código para generar una tabla en la memoria de la DSP. Esta tabla contiene los valores antes y después de aplicar el filtro. El almacenamiento de dichos datos provoca una alteración en el tiempo de ejecución del ciclo de adquisición de datos principal (la causa son los procesos de muestreo explicados en la sección 3.2); debido a esto, tenemos que considerar que la frecuencia de muestreo también está siendo modificada. El tamaño de la tabla se determinó tomando en cuenta la reducción de la frecuencia de muestreo que podría llegar a tener si la tabla de valores se vuelve muy grande.

Mediante pruebas se determinó que el tamaño óptimo para las tablas es de 50 muestras, lo que nos deja una frecuencia de muestreo de aproxi-

madamente 22kHz, suficiente para poder comprobar que los filtros están realizando su función. El búfer utilizado para almacenar estos valores es similar al que se utiliza para guardar las N muestras que son filtradas y la función que recoge las nuevas muestras en la tabla es la misma shift que se describe en la Sección 3.2 , pero el argumento loop, en este caso, es 50.

La configuración utilizada en la herramienta para graficar la FFT se muestra en la Figura 4.10. Es importante corroborar que la dirección de inicio coincida con la dirección de la tabla creada con los datos de interés.

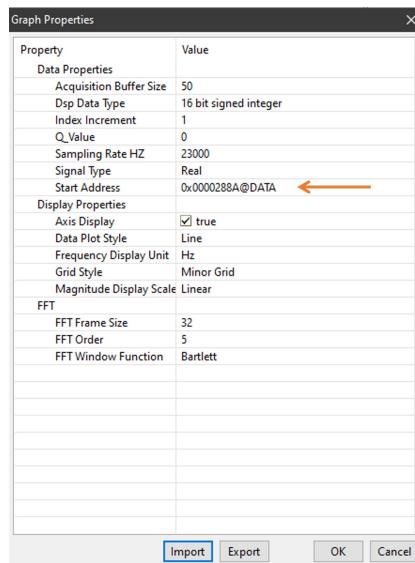


Figura 4.10: Configuración para la gráfica FFT en CCS

A continuación, se muestran las capturas mas representativas de la ejecución en la DSP de los filtros diseñados en la sección 4.1. De lado izquierdo siempre se muestra la señal de entrada y de lado derecho la señal de salida. Primero se realizó una prueba con un tono que se encuentra en la banda de paso del filtro, y después un tono que se encuentre en la banda de rechazo. Se espera que el tono mantenga su amplitud siempre que se encuentre en la banda de paso, en caso contrario, el tono se atenuara en función del filtro. Por ultimo se toman 4 capturas en diferente instante

mientras se lee desde la DSP una señal de audio de una canción, si los filtros están cumpliendo su función, los rangos del espectro de frecuencia comprendidos por la banda de rechazo se atenuaran y dicha alteración se visualizará al comparar ambas gráficas. Se repite el mismo proceso para cada uno de los filtros diseñados en la sección anterior.

4.2.1. Caso I, Pasa bajas

Para el primer ejemplo de implementación se tomo el filtro diseñado en la GUI de la figura 4.1, que se refiere a un filtro pasa bajas diseñado por el método de Parks-McLellan. La frecuencia de paso termina en 3500Hz y la frecuencia de corte empieza en 4500Hz. El orden es 39 y los 40 coeficientes obtenidos son los siguientes: -57, -293, -607, -802, -603, -21, 544, 567, -53, -751, -726, 202, 1193, 1039, -537, -2190, -1790, 1693, 6910, 10786, 10786, 6910, 1693, -1790, -2190, -537, 1039, 1193, 202, -726, -751, -53, 567, 544, -21, -603, -802, -607, -293, -57.

En la figura 4.11 se usa un tono de 1kHz que se encuentra en la banda de paso del filtro. Luego se toma un tono de 6kHz, que se muestra en la figura 4.12, este tono esta en la banda de rechazo.

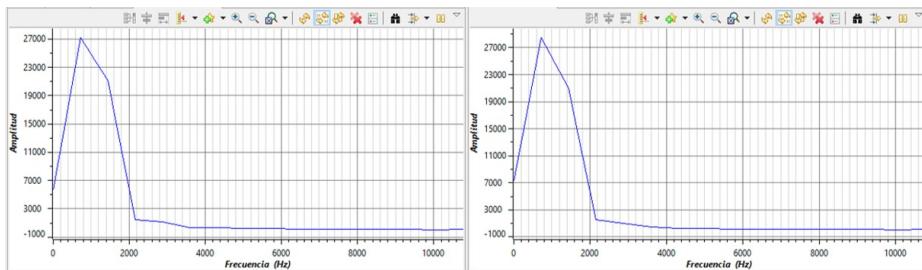


Figura 4.11: Prueba en DSP del filtro pasa bajas con un tono de 1kHz

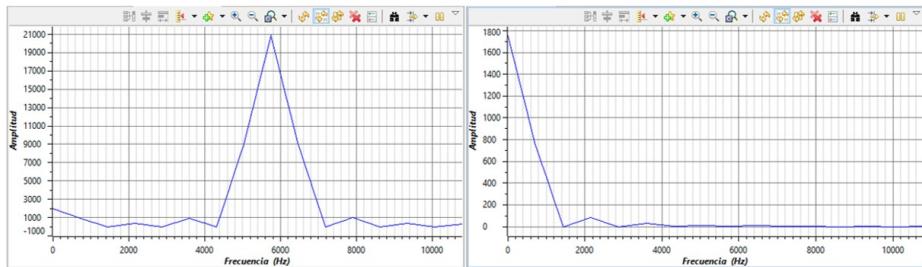


Figura 4.12: Prueba en DSP del filtro pasa bajas con un tono de 6kHz

De la Figura 4.13 a la Figura 4.16 se toman muestras mientras se reproduce una pieza musical.

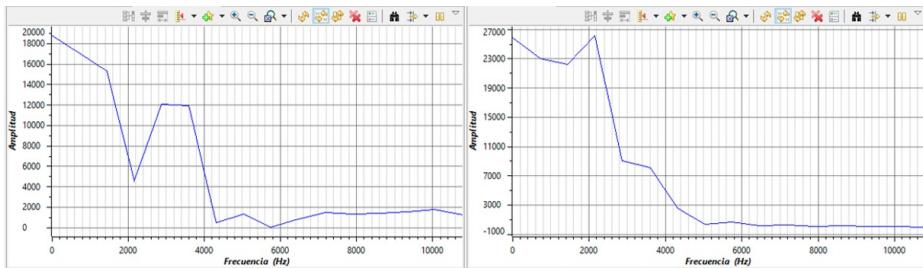


Figura 4.13: Prueba en DSP del filtro pasa bajas con una señal de audio, I

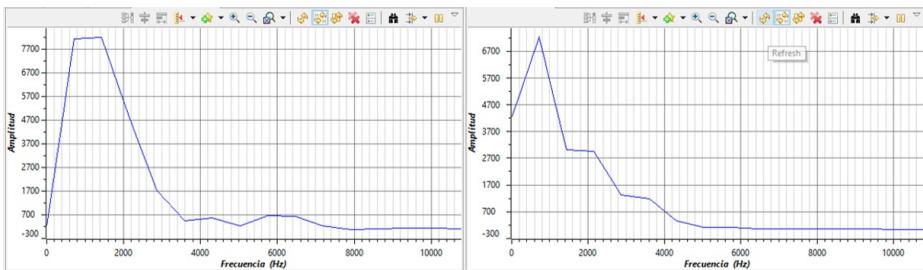


Figura 4.14: Prueba en DSP del filtro pasa bajas con una señal de audio, II

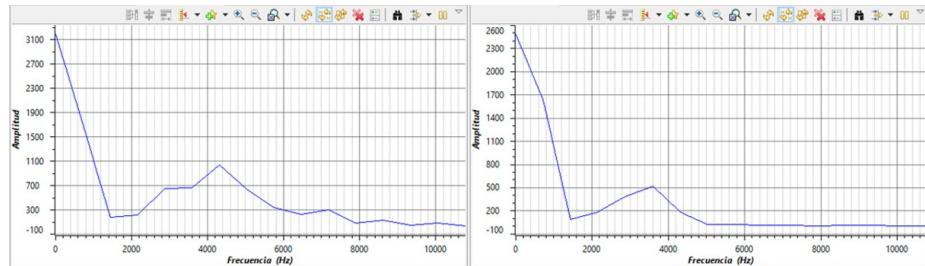


Figura 4.15: Prueba en DSP del filtro pasa bajas con una señal de audio, III

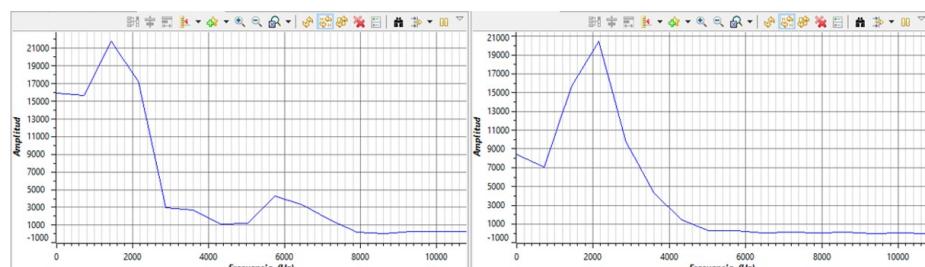


Figura 4.16: Prueba en DSP del filtro pasa bajas con una señal de audio, IV

4.2.2. Caso II, Pasa altas

En este caso, el filtro fue diseñado por el método de Aventanado. Es un filtro pasa altas con frecuencia de corte en 6000Hz. El orden es 16 y a continuación los 17 elementos que conforman los coeficientes: 0, 109, 335, -626, -717, 2071, 1118, -9419, 15563, -9419, 1118, 2071, -717, -626, 335, 109, 0. En la Figura 4.3 se puede ver la captura de la pantalla de diseño para este filtro.

Los tonos elegidos para probar el filtro son de 7kHz, que se encuentra en la banda de paso del filtro; y, 1kHz que no lo está. Las pruebas se pueden ver en la Figura 4.17 y Figura 4.18, respectivamente.

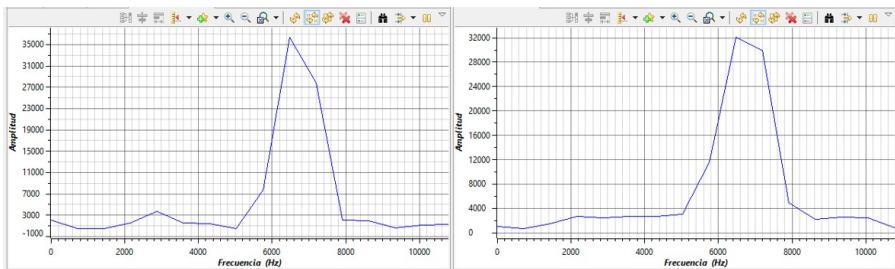


Figura 4.17: Prueba en DSP del filtro pasa altas con un tono de 7kHz

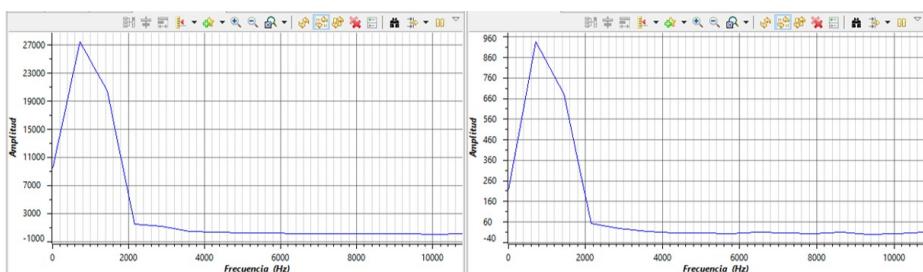


Figura 4.18: Prueba en DSP del filtro pasa altas con un tono de 1kHz

En las Figuras 4.19 ,4.20, 4.21 y 4.22
se tomaron capturas en diferente instante mientras el codec lee una
canción.

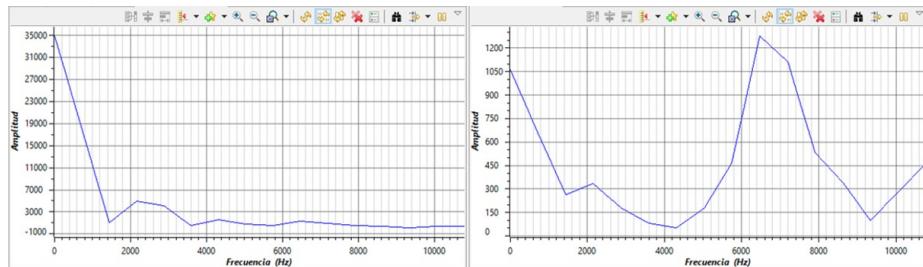


Figura 4.19: Prueba en DSP del filtro pasa altas con una señal de audio, I

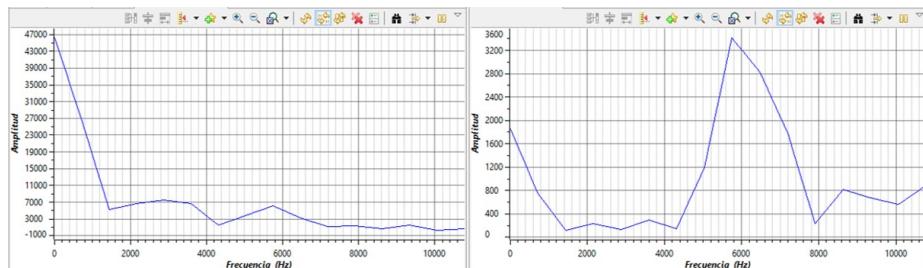


Figura 4.20: Prueba en DSP del filtro pasa altas con una señal de audio, II

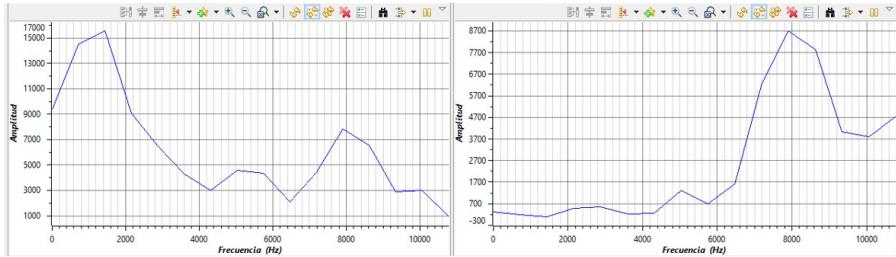


Figura 4.21: Prueba en DSP del filtro pasa altas con una señal de audio, III

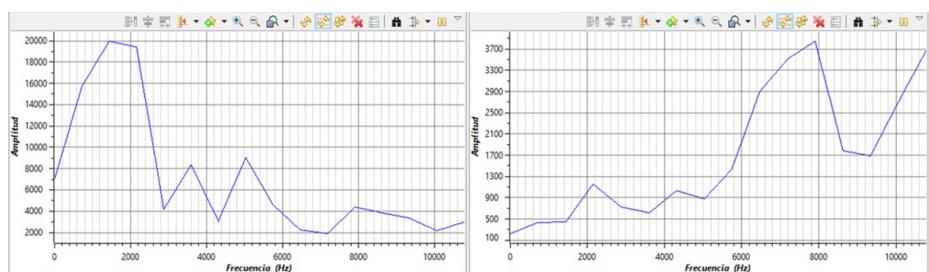


Figura 4.22: Prueba en DSP del filtro pasa altas con una señal de audio, IV

4.2.3. Caso III, Pasa banda

El filtro para este ejemplo es un pasa banda, mostrado en la Figura 4.5, con las siguientes frecuencias características: $F_{pass1} = 4000\text{Hz}$, $F_{stop1} = 5000\text{Hz}$, $F_{pass2} = 6000\text{Hz}$, $F_{stop2} = 7000\text{Hz}$. El resultado es un filtro de orden 39: -13, 16, -17, 79, 130, -243, -343, 552, 701, -1005, -1196, 1577, 1790, -2198, -2403, 2770, 2932, -3188, -3278, 3370, 3370, -3278, -3188, 2932, 2770, -2403, -2198, 1790, 1577, -1196, -1005, 701, 552, -343, -243, 130, 79, -17, 16, -13.

En la Figura 4.23 se utiliza un tono de 5.5kHz que se ubica justo en el centro de la banda de paso del filtro. En la Figura 4.24 se utiliza un tono de 2kHz para comprobar la banda de rechazo del pasa banda. **Nota:** No confundir el pico que se observa a la salida, es consecuencia del cambio de escala al momento de graficar.

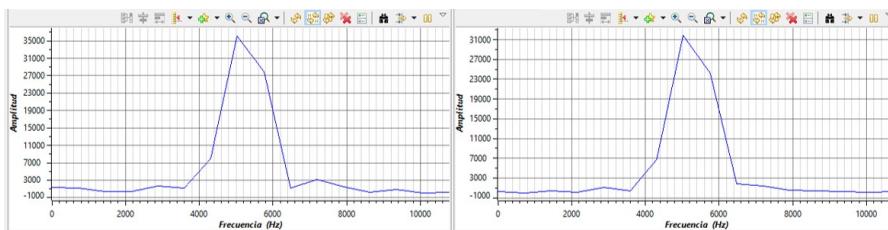


Figura 4.23: Prueba en DSP del filtro pasa banda con un tono de 5.5kHz

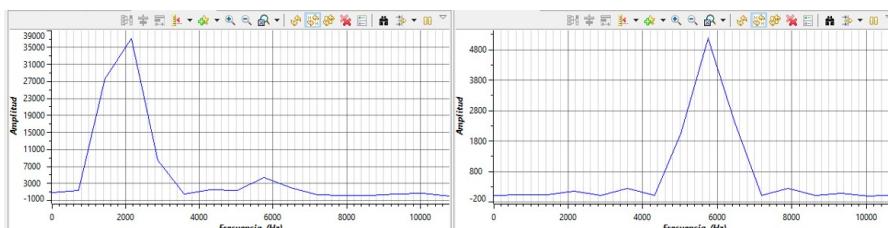


Figura 4.24: Prueba en DSP del filtro pasa banda con un tono de 2kHz

A partir de la Figura 4.25 hasta la Figura 4.28 se pueden observar 4 ejemplos del filtro aplicado mientras se muestrea una pieza musical.

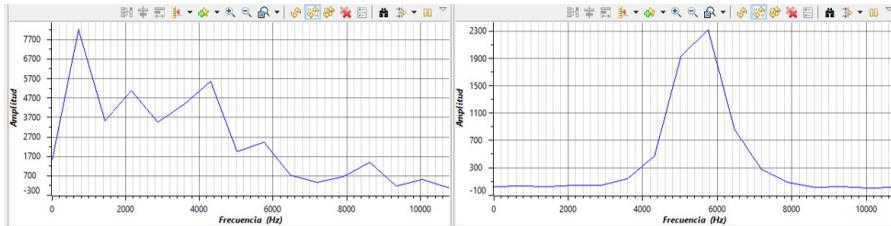


Figura 4.25: Prueba en DSP del filtro pasa banda con una señal de audio, I

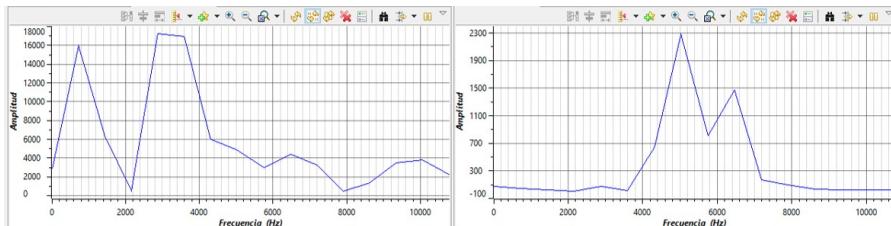


Figura 4.26: Prueba en DSP del filtro pasa banda con una señal de audio, II

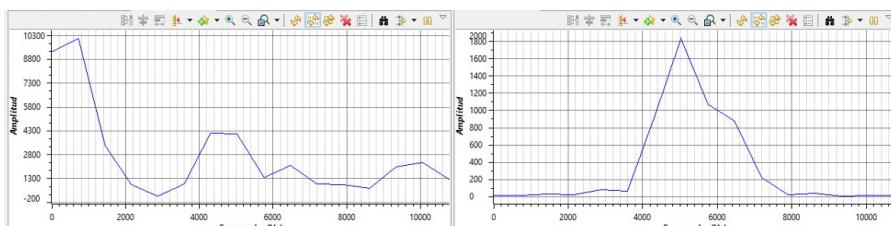


Figura 4.27: Prueba en DSP del filtro pasa banda con una señal de audio, III

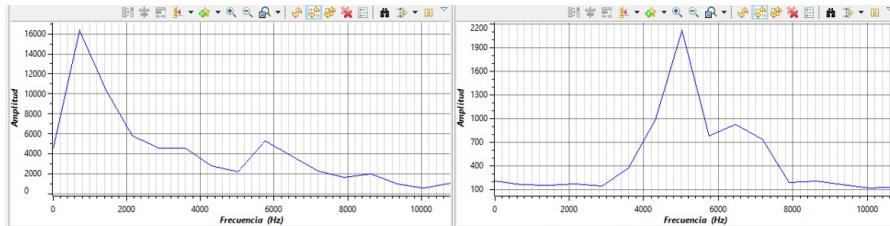


Figura 4.28: Prueba en DSP del filtro pasa banda con una señal de audio, IV

4.2.4. Caso IV, Rechazo banda

Para el último ejemplo se usa un filtro rechaza banda que tiene una frecuencia de corte entre 4000Hz y 6000Hz. El filtro fue diseñado por el método de Aventanado con un orden de 32. Los coeficientes calculados son los siguientes: -72, -91, 59, 128, -19, 4, 24, -553, -421, 1466, 1585, -2200, -3435, 2062, 5224, -850, 26943, -850, 5224, 2062, -3435, -2200, 1585, 1466, -421, -553, 24, 4, -19, 128, 59, -91, -72. En la Figura 4.7 se observa el filtro diseñado en la interfaz de usuario para este ejemplo.

La prueba de este filtro se realiza primero con un tono de 10kHz para comprobar la banda de paso, esto se observa en la Figura 4.29, se observa como la señal no sufre ningún cambio significativo. Para la banda de rechazo observamos en la Figura 4.30 el tono de 5kHz antes de ser atenuado y después de atenuarse por la acción del filtro rechaza banda.

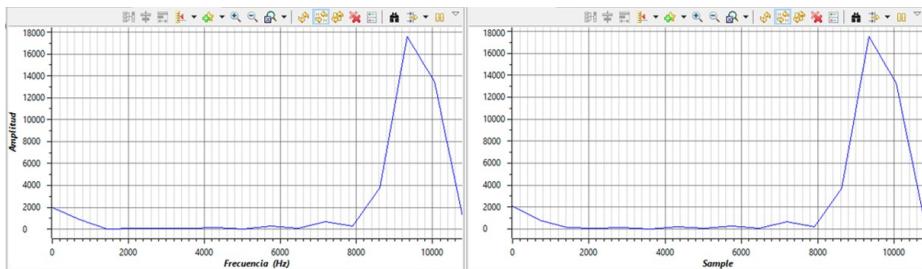


Figura 4.29: Prueba en DSP del filtro rechaza banda con un tono de 10kHz

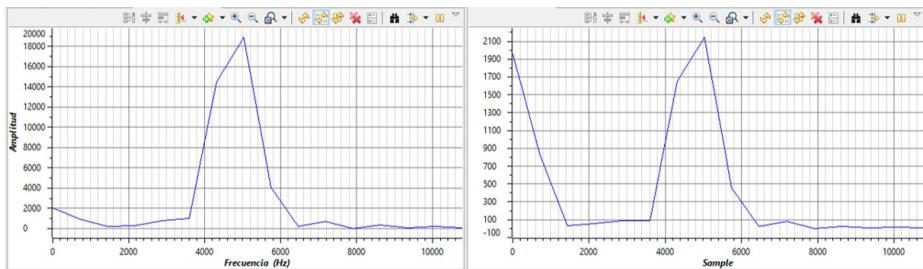


Figura 4.30: Prueba en DSP del filtro rechaza banda con un tono de 5kHz

La comprobación musical se puede observar a continuación en las figuras 4.31, 4.32, 4.33 y 4.34.

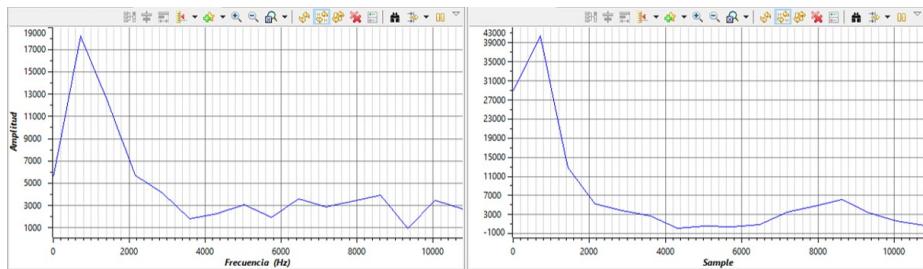


Figura 4.31: Prueba en DSP del filtro rechaza banda con una señal de audio, I

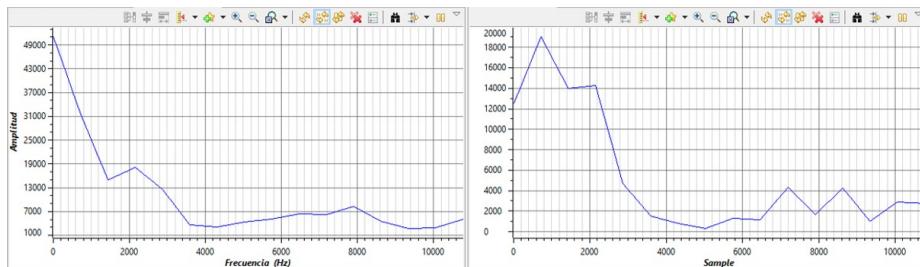


Figura 4.32: Prueba en DSP del filtro rechaza banda con una señal de audio, II

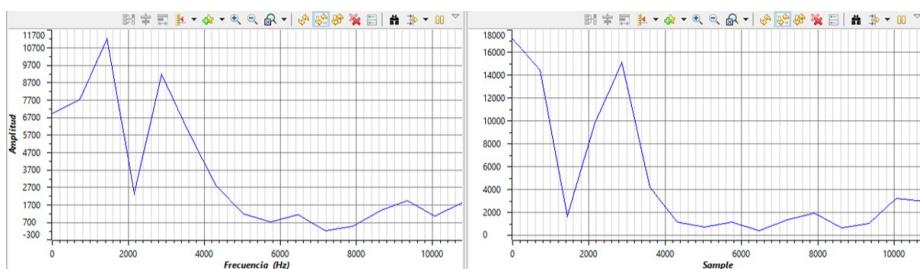


Figura 4.33: Prueba en DSP del filtro rechaza banda con una señal de audio, III

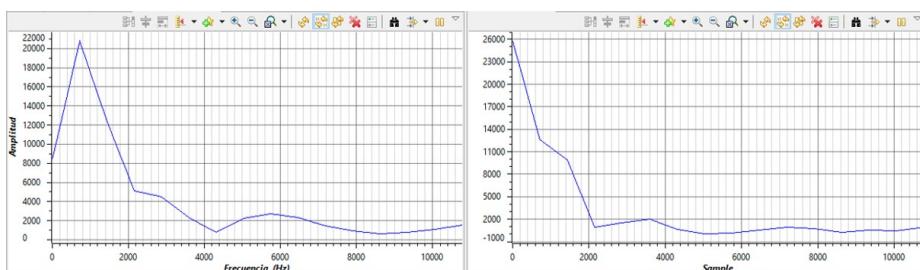


Figura 4.34: Prueba en DSP del filtro rechaza banda con una señal de audio, IV

Capítulo 5

Conclusiones

Los objetivos que se establecieron al inicio del desarrollo del proyecto fueron cubiertos al 100 %. El camino de aprendizaje recorrido desde el comienzo cuando se conocía bastante poco del tema hasta concluir el proyecto con la implementación de los filtros fue bastante satisfactorio.

En cuanto a la GUI el proceso fue muy cómodo, pues MATLAB es un ambiente bastante amigable y tiene su documentación completa, incluye ejemplos tanto desde la opción de ayuda dentro del programa como desde la pagina web oficial. Las herramientas están y solo debemos hacerlas encajar para construir herramientas mas grandes que nos faciliten el trabajo.

Desde el comienzo del proyecto se conocía la existencia de la aplicación incluida en MATLAB para el diseño de filtros, pero el proyecto requería ser mas ambiciosos. El objetivo se enfocaba en filtros aplicados al audio digital y para ese tipo de aplicaciones la aplicación de Filter Designer se queda algo corta, sobre todo, al tratar de filtrar un audio. Si se hubiera decidido trabajar únicamente con la aplicación que incluye MATLAB el proceso de diseño de los filtros seria mas complicado; primero se tendría que entrar a la herramienta, llenar el formulario par el filtro que queremos, luego debemos exportar el filtro al workspace e importar el audio con el que queremos probar el filtro, luego de aplicar el filtro a las muestras, comprobaríamos generando las gráficas a partir de los comandos que tenemos en Matlab, por ultimo, si quisieras reportar los resultados tenemos que darles formato a las gráficas. Por supuesto que el proceso descrito anteriormente se puede automatizar, pero, en general, valió la pena invertir tiempo en

el desarrollo de una nueva GUI especializada para los requerimientos del trabajo documentado.

Algo que hizo mas sencillo el desarrollo de la GUI fue el hecho de que no se trabajo con el muestreo en tiempo real, sino que partíamos de un archivo previamente muestreado. Esto afecto el proceso de diseño de filtros en cuanto a que el usuario pudiera definir la frecuencia de muestreo para el filtro. Es una limitante que el usuario deba tener un archivo muestreado a la frecuencia que necesita para poder generar un filtro en la GUI que funcione para ese caso.

En cuanto a la ejecución de los filtros en la tarjeta, se presentaron más problemas. Lo primero fue la poca compatibilidad entre versiones que tienen los productos de Texas Instruments y los archivos extras para poder depurar desde Code Composer Studio.

Al momento de la programación, y sobre todo para las pruebas, un claro problema en el tiempo de ejecución del microcontrolador nos hizo tomar ciertas consideraciones. Como se hablo en la Sección 4.2, Al momento de utilizar el depurador de Code Composer Studio era necesario modificar el ciclo principal de trabajo y eso provocaba que la frecuencia de muestreo que inicialmente fue establecida en 48kHz se viera reducida. Al final fue posible solventar este inconveniente pero se tuvo que sacrificar resolución en la gráfica de la FFT.

Este proyecto fue un gran reto, y se puede decir que a pesar de que se consiguieron los objetivos aun quedan aspectos por mejorar, a continuación se hacen algunas propuestas para un trabajo futuro.

- El primero punto que nos gustaría mejorar es la posibilidad de elegir la frecuencia de muestreo con la que se diseña el filtro en la GUI. Sería bastante beneficioso invertir tiempo en mejorar la GUI en este sentido.
- La frecuencia de muestreo para grabar voz desde la GUI es constante. Nos gustaría hacer que el usuario pudiera elegir arbitrariamente este valor, o por lo menos, elegir desde un lista de posibles valores.
- Los métodos de diseño con los que cuenta la GUI son suficientemente potentes para diseñar la gran mayoría de los filtros que se deseen, sin

embargo, se pueden agregar mas métodos de diseño y de esta manera abarcar mas escenarios.

- Aunque el programa funciona correctamente a 48 kHz omitiendo los procesos que conlleva graficar desde el Code Composer Studio, habría una importante mejora si se logra paralelizar el proceso de almacenamiento de muestras.
- Usar los periféricos incluidos en la tarjeta como los botones pulsadores la pantalla oled y los leds le agregarían dinamismo al proyecto.

Apéndice A

Interfaz de usuario

A.1. GUI_FIR

```
function varargout = GUI_FIR(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',
                    {'gui_Singleton'}, 'mfilename', ...
                    {'gui_Singleton'}, 'gui_Singleton', ...
                    'gui_OpeningFcn', @GUI_FIR_OpeningFcn, ...
                    'gui_OutputFcn', @GUI_FIR_OutputFcn, ...
                    'gui_LayoutFcn', [], ...
                    'gui_Callback', []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before GUI_FIR is made visible.
function GUI_FIR_OpeningFcn(hObject, eventdata, handles, varargin)

handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% --- Outputs from this function are returned to the command line.
function varargout = GUI_FIR_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;

%-----
%INICIO---INICIO---INICIO---INICIO---INICIO---INICIO---INICIO---
%-----

% --- Ejecutar al presionar el botón abrir archivo
function AbrirArchivo_Callback(hObject, eventdata, handles)
%buscando el archivo
global s;
global fs; global filename;
```

```

global v;
%condicion para rediseñar el filtro en funcion de la nueva frecuencia de
%uestreo
global ord_fc;
[filename, pathname] = uigetfile({'*.wav*','Audio (*.wav*)'},'Abre un audio');
[s fs] = audioread(filename); %archivo guardado en s, frecuencia de muestreo guardada en fs

if isempty(ord_fc) == false
    switch v
        case 1
            FIR_des(handles);
        case 2
            FIRPM_des(handles);
    end
end

set(handles.namemostrar, 'String', filename);
load_sound(handles);

% --- Ejecutar al seleccionar el tipo de paneo que se realiza al audio
function paneo_Callback(hObject, eventdata, handles)
global s; global fs;
global filename;

if isempty(s) == false
    [s fs] = audioread(filename);
    panear(handles);
end

% --- Ejecutar al presionar el boton de grabacion en el panel de archivo
function RecButton_Callback(hObject, eventdata, handles)
global s_rec;
global fs_rec;
fs_rec=44100;
r=audiorecorder(fs_rec, 16, 1);
record(r);
stop=1;
while stop == 1
    stop=get(hObject, 'Value');
    pause(0.0001)
end
pause(r);

s_rec=audiiodata(r,'double');

% --- Ejecutar al presionar el boton de play en el panel de archivo
function PlayButton_Callback(hObject, eventdata, handles)
global s_rec; global fs_rec;
sound(s_rec,fs_rec);

% --- Ejecutar al presionar el boton de load en el panel de archivo
function LoadButton_Callback(hObject, eventdata, handles)
global s_rec; global fs_rec;
global s; global fs;

s=s_rec;
fs=fs_rec;
load_sound(handles);

% --- Ejecutar al seleccionar el metodo de diseño
function metodo_Callback(hObject, eventdata, handles)
%borrando las graficas de filtrado
global v;
axes(handles.Frq_filtrado);
cla reset;
axes(handles.filtrado);
cla reset;

v = get(handles.metodo, 'Value');

```

```

switch v
    case 1
        FIR_formulario(handles);
    case 2
        FIRPM_formulario(handles);
end

% --- Ejecutar al pulsar el boton filtrar
function filtrar_Callback(hObject, eventdata, handles)
global filtro;
global s;global fs;
global y;

y = filter(filtro, 1, s);

axes(handles.Frq_filtrado);
dbaget(handles.DB_filtrado, 'Value');
graficar_frecuencia(y, fs, 'Freq [Hz]', 'Amplitud',...
    'Espectro de Frecuencia filtrado',db, handles);

duracion = length(s)/fs;
axes(handles.filtrado);
graficar_tiempo(y, duracion, "tiempo [s]", "Amplitud", 'Audio Filtrado - Forma de onda');

% --- Ejecutar al presionar el boton editar filtro
function filt_edit_Callback(hObject, eventdata, handles)
metodo_Callback(hObject, eventdata, handles)

% --- Slider de vista de graficas de Frecuencia
function slider2_Callback(hObject, eventdata, handles)
global fs;
valor = get(hObject, 'value');
limite = (valor * ((fs/2)-500))/10;
limite2 = limite+500;

if valor == 10
    limite = 0;
    limite2 = fs/2;
end

axes(handles.Frq_original);
xlim([limite limite2]);
axes(handles.Frq_filtrado);
xlim([limite limite2]);

% Botones de Reproduccion de pistas
function play_original_Callback(hObject, eventdata, handles)
global s;
global fs;
sound(s, fs)
stop=1;
while stop == 1
    stop=get(hObject, 'Value');
    pause(0.0001)
end
clear sound

function Play_filtrado_Callback(hObject, eventdata, handles)
global y;
global fs;
sound(y, fs)
stop=1;
while stop == 1
    stop=get(hObject, 'Value');
    pause(0.0001)
end
clear sound

% Botones de Diseño de las graficas de filtro

```

```

function dB_original_Callback(hObject, eventdata, handles)
global s;
global fs;
db=get(hObject, 'Value');
axes(handles.Frq_original);
graficar_frecuencia(s, fs, 'Freq [Hz]', 'Amplitud',...
    'Espectro de Frecuencia original',db, handles);

function dB_filtrado_Callback(hObject, eventdata, handles)
global y;
global fs;
db=get(hObject, 'Value');
axes(handles.Frq_filtrado);
graficar_frecuencia(y, fs, 'Freq [Hz]', 'Amplitud',...
    'Espectro de Frecuencia filtrado',db, handles);

% -----> Boton para mostrar el espectrograma en las graficas de los audios <-----
function espect_Callback(hObject, eventdata, handles)
global y;
global s; global fs;
global w1;

vget(hObject, 'Value');
if v == 1

fn2 = {'Barlett','Barthanwin','Blackman',...
    'Blackmanharris', 'Bohmanwin', 'Chebwin',...
    'Flattopwin', 'Gausswin', 'Hamming',...
    'Hann', 'Kaiser', 'Nuttallwin',...
    'Parzenwin', 'Rectwin', 'Taylorwin',...
    'Tukeywin', 'Triang'};

[w_string,tf] = listdlg('PromptString',{'Ventana',...
    '-----',''},...
    'SelectionMode','single','ListString',fn2);

orden=256;
switch w_string
case 'Barlett'
    w1 = window(@bartlett,orden);
case 'Barthanwin'
    w1 = window(@barthannwin,orden);
case 'Blackman'
    w1 = window(@blackman,orden);
case 'Blackmanharris'
    w1 = window(@blackmanharris,orden);
case 'Bohmanwin'
    w1 = window(@bohmanwin,orden);
case 'Chebwin'
    w1 = window(@chebwin,orden);
case 'Flattopwin'
    w1 = window(@flattopwin,orden);
case 'Gausswin'
    w1 = window(@gausswin,orden);
case 'Hamming'
    w1 = window(@hamming,orden);
case 'Hann'
    w1 = window(@hann,orden);
case 'Kaiser'
    w1 = window(@kaiser,orden);
case 'Nuttallwin'
    w1 = window(@nuttallwin,orden);
case 'Parzenwin'
    w1 = window(@parzenwin,orden);
case 'Rectwin'
    w1 = window(@rectwin,orden);
case 'Taylorwin'
    w1 = window(@taylorwin,orden);
case 'Tukeywin'
    w1 = window(@tukeywin,orden);
case 'Triang'

```

```

        w1 = window(@triang,orden);
    end

    set(handles.dB_original,'Visible','off')
    axes(handles.Frq_original);
    spectrogram(s, w1, 250, orden, fs)

    set(handles.dB_filtrado,'Visible','off')
    axes(handles.Frq_filtrado);
    spectrogram(y, w1, 250, orden, fs)
else
    set(handles.dB_original,'Visible','on')
    axes(handles.Frq_original);
    db=get(handles.dB_original, 'Value');
    graficar_frecuencia(s, fs, 'Freq [Hz]', 'Amplitud',...
        'Espectro de Frecuencia filtrado',db, handles);

    set(handles.dB_filtrado,'Visible','on')
    axes(handles.Frq_filtrado);
    db=get(handles.dB_filtrado, 'Value');
    graficar_frecuencia(y, fs, 'Freq [Hz]', 'Amplitud',...
        'Espectro de Frecuencia filtrado',db, handles);
end

% -----> Boton para crear un archivo C con los coeficientes del filtro <-----
function export_Callback(hObject, eventdata, handles)
export_coef();

% -----> Boton para entrar a la interfaz de efectos de sonido <-----
function efectos_Callback(hObject, eventdata, handles)
global s; global fs;
ruta = 'C:\Users\tauro\Desktop\RESIDENCIAS\Proyecto\temp\audio.wav';
audiowrite(ruta, s, fs);

Gui_Efects

% -----User Functions-----
%-----
%-----
```

function load_sound(handles)

global fs;

global s;

panear(handles);

s = normalizar(s);

%mostrando la frecuencia de muestreo

fs_string = int2str(fs);

fs_mostrar=strcat("Fs = ", fs_string);

set(handles.fsmostrar, 'String', fs_mostrar);

%borrando las graficas de filtrado

axes(handles.Frq_filtrado);

cla reset;

axes(handles.filtrado);

cla reset;

function FIR_des(handles) %Diseño de Filtro Aventanado

global fs; global filtro;

global ord_fc;global indx_type;global w;global w_string;

fc_string = ord_fc{2};

orden_string = ord_fc{1};

Fc = (2*str2num(fc_string))/fs;

orden = str2num(orden_string);

switch w_string

case 'Barlett'

w = window(@bartlett,orden + 1);

```

case 'Barthanwin'
    w = window(@barthanwin,orden + 1);
case 'Blackman'
    w = window(@blackman,orden + 1);
case 'Blackmanharris'
    w = window(@blackmanharris,orden + 1);
case 'Bohmanwin'
    w = window(@bohmanwin ,orden + 1);
case 'Chebwin'
    w = window(@chebwin,orden + 1);
case 'Flattopwin'
    w = window(@flattopwin,orden + 1);
case 'Gaußwin'
    w = window(@gausswin,orden + 1);
case 'Hamming'
    w = window(@hamming,orden + 1);
case 'Hann'
    w = window(@hann,orden + 1);
case 'Kaiser'
    w = window(@kaiser,orden + 1);
case 'Nuttallwin'
    w = window(@nuttallwin,orden + 1);
case 'Parzenwin'
    w = window(@parzenwin,orden + 1);
case 'Rectwin'
    w = window(@rectwin,orden + 1);
case 'Taylorwin'
    w = window(@taylorwin,orden + 1);
case 'Tukeywin'
    w = window(@tukeywin,orden + 1);
case 'Triang'
    w = window(@triang,orden + 1);
end

switch indx_type
    case 1
        filtro = fir1(orden, Fc , w);
        typefilt = 'Pasa Bajas';
        msg4 = strcat("Fc = ", fc_string);
    case 2
        filtro = fir1(orden,Fc, 'high', w);
        typefilt = 'Pasa Altas';
        msg4 = strcat("Fc = ", fc_string);
    case 3
        fc2_string = ord_fc{3};
        Fc2 = (2*str2double(fc2_string))/fs
        filtro = fir1(orden, [Fc Fc2], w);
        typefilt = 'Pasa Banda';
        msg4 = strcat("Fc1 = ", fc_string, " , Fc2 = ", fc2_string);
    case 4
        fc2_string = ord_fc{3};
        Fc2 = (2*str2double(fc2_string))/fs
        filtro = fir1(orden, [Fc Fc2], 'stop', w);
        typefilt = 'Rechaza Banda';
        msg4 = strcat("Fc1 = ", fc_string, " , Fc2 = ", fc2_string);
    end

%mostrando el filtro
msg1 = strcat("Ventana: ", w_string);
msg2 = typefilt;
msg3 = strcat("Orden = ", orden_string);

set(handles.filtromostrar, 'String', [msg1 msg2 msg3 msg4]);
graficar_filtro(handles);

function FIRPM_des(handles) %Diseño de Filtro Parks-McLellan
global filtro;
global n; global fo; global ao, global x;

```

```

filtro = firpm(n,fo,ao,x);
graficar_filtro(handles);

function graficar_filtro(handles)
global filtro; global fs;

%graficando el filtro
h=freqz(filtro);
w=linspace(0, fs/2, length(h));
axes(handles.Filtro);
plot(w, 20*log10(abs(h)))

grid on;
grid minor;
xlabel('Freq [Hz]');
ylabel('Amplitud [dB]');

ax = gca;
ax.XAxis.Exponent = 3;
xlim([0 20000]);

function [audio1] = panear(handles)
global s; global s1; global fs;
global paneo;

duracion = length(s)/fs;

paneo = get(handles.paneo, 'Value');
switch paneo
    case 1
        try
            audio1 = (s(:,1) + s(:,2))/2;
        catch
            audio1 = s;
        end
    case 2
        audio1 = s(:,1);
    case 3
        audio1 = s(:,2);
end

s = audio1;

axes(handles.original);
graficar_tiempo(audio1, duracion, "tiempo [s]", "Amplitud",...
'Audio - Forma de onda');
db = get(handles.dB_original, 'Value');
axes(handles.Frq_original);
graficar_frecuencia(s, fs, 'Freq [Hz]', 'Amplitud',...
'Espectro de Frecuencia original',db, handles);

function graficar_tiempo(onda, duracion, x_name, y_name, titulo)
%graficando la pista de audio
t=linspace(0,duracion, length(onda));
plot(t, onda)
title(titulo);
xlim([0 duracion]);
xlabel(x_name);
ylabel(y_name);
ylim([-1 1]);

function graficar_frecuencia(onda, frqs, x_name, y_name, titulo, dB, handles)
global fs;
espectro = fft(onda); %transformada de fourier de la onda a graficar
abs_esp = abs(espectro);
muestras = length(onda);
f=linspace(0,frqs/2,muestras/2);
mag_audio =abs_esp(1:muestras/2)/max(abs_esp);

```

```
%Magnitud en dB ó normalizada
if dB == true
    mag_audio = 20*log10(mag_audio);
end

plot(f, mag_audio)
title(titulo);
xlabel(x_name);
ylabel(y_name);

ylim([0 1]);
if dB == true
    ylim([-50 0]);
end

valor = get(handles.slider2, 'value');
limite = (valor * ((fs/2)-500))/10;
limite2 = limite+500;
if valor == 10
    limite = 0;
    limite2 = fs/2;
end
xlim([limite limite2]);
ax = gca;
ax.XAxis.Exponent = 3;
grid on; grid minor;

function [snd_norm] = normalizar(sonido)
maximo = max(abs(sonido));
snd_norm = sonido/maximo;

function FIR_formulario(handles)
%variables de diseño de filtro aventanado
global ord_fc;global indx_type;global w;global w_string;

% Tipo de Filtro
fn = {'Paso Bajo','Paso Alto','Paso Banda', 'Rechazo Banda'};
[indx_type,tf] = listdlg('PromptString',{'Tipo de Filtro',...
    '-----',''},'SelectionMode','single','ListString',fn);
% Orden y Fc
switch indx_type
    case 1
        dlgtitle = 'Paso Bajo';
        prompt = {'Orden del filtro','Frecuencia de corte[Hz']};
        dims = [1 35];
        definput = {'32', '4000'};
    case 2
        dlgtitle = 'Paso Alto';
        prompt = {'Orden del filtro','Frecuencia de corte[Hz']};
        dims = [1 35];
        definput = {'32', '4000'};
    case 3
        dlgtitle = 'Paso Banda';
        prompt = {'Orden del filtro','Frecuencia de corte inferior[Hz]',...
            'Frecuencia de corte superior[Hz']};
        dims = [1 40];
        definput = {'32', '4000', '12000'};
    case 4
        dlgtitle = 'Rechazo Banda';
        prompt = {'Orden del filtro','Frecuencia de corte inferior[Hz]',...
            'Frecuencia de corte superior[Hz']};
        dims = [1 40];
        definput = {'32', '4000', '12000'};
end
ord_fc = inputdlg(prompt,dlgtitle,dims,definput);

% Ventana
orden = str2num(ord_fc{1});
```

```

fn2 = {'Barlett', 'Barthanwin', 'Blackman',...
       'Blackmanharris', 'Bohmanwin', 'Chebwin',...
       'Flattopwin', 'Gausswin', 'Hamming',...
       'Hann', 'Kaiser', 'Nuttallwin',...
       'Parzenwin', 'Rectwin', 'Taylorwin',...
       'Tukeywin', 'Triang'};

[ indx_wind, tf ] = listdig('PromptString', {'Ventana del Filtro',...
       '-----', ''}, ...
       'SelectionMode', 'single', 'ListString', fn2);

w_string=fn2{indx_wind};

FIR_des(handles);

function FIRPM_formulario(handles)
global fs;
global n; global fo; global ao, global x;
% Tipo de Filtro
fn = {'Paso Bajo', 'Paso Alto', 'Paso Banda', 'Rechazo Banda'};
[ indx_type, tf ] = listdig('PromptString', {'Tipo de Filtro',...
       '-----', ''}, 'SelectionMode', 'single', 'ListString', fn);
% Orden y Fc
switch indx_type
    case 1
        dlgttitle = 'Paso Bajo';
        prompt = {'Frecuencia de paso [Hz]', 'Frecuencia de corte [Hz]'};;
        dims = [1 35];
        definput = {'3500', '4500'};

        prompt2 = {'Rizo de paso', 'Rizo de rechazo'};
        dims2 = [1 35];
        definput2 = {'0.1', '0.001'};

        ideal = [1 0];
    case 2
        dlgttitle = 'Paso Alto';
        prompt = {'Frecuencia de corte [Hz]', 'Frecuencia de paso [Hz]'};;
        dims = [1 35];
        definput = {'3500', '4500'};

        prompt2 = {'Rizo de rechazo', 'Rizo de paso'};
        dims2 = [1 35];
        definput2 = {'0.001', '0.1'};

        ideal = [0 1];
    case 3
        dlgttitle = 'Paso Banda';
        prompt = {'Frecuencia de corte 1 [Hz]', 'Frecuencia de paso 1 [Hz]',...
                  'Frecuencia de paso 2 [Hz]', 'Frecuencia de corte 2 [Hz]'};;
        dims = [1 40];
        definput = {'4000', '5000', '10000', '12000'};

        prompt2 = {'Rizo de rechazo 1', 'Rizo de paso', 'Rizo de rechazo 2'};
        dims2 = [1 35];
        definput2 = {'0.001', '0.1', '0.001'};

        ideal = [0 1 0];
    case 4
        dlgttitle = 'Rechazo Banda';
        prompt = {'Frecuencia de paso 1 [Hz]', 'Frecuencia de corte 1 [Hz]',...
                  'Frecuencia de corte 2 [Hz]', 'Frecuencia de paso 2 [Hz]'};;
        dims = [1 40];
        definput = {'4000', '5000', '10000', '12000'};

        prompt2 = {'Rizo de paso 1', 'Rizo de rechazo', 'Rizo de paso 2'};
        dims2 = [1 35];
        definput2 = {'0.1', '0.0001', '0.1'};

        ideal = [1 0 1];
end

```

```

frec_str = inputdlg(prompt,digtitle,dims,definput);
rizo_str = inputdlg(prompt2,digtitle,dims2,definput2);
frec = str2double(frec_str);
rizo = str2double(rizo_str);

[n,fo,ao,x] = firpmord(frec,ideal,rizo,fs,'cell');
digtitle2 = 'Orden';
prompt2 = {'Orden del filtro'};
dims2 = [1 35];
definput2 = {num2str(n)};
n_str = inputdlg(prompt2,digtitle2,dims2,definput2);
n = str2double(n_str);

FIRPM_des(handles);

%mostrando el filtro
msg1 = digtitle;
switch indx_type
    case 1
        msg2 = strcat("Fpass: ",frec_str{1}, "; Fstop: ", frec_str{2});
        msg3 = strcat("Dpass: ",rizo_str{1}, "; Dstop: ", rizo_str{2});
        msg4 = strcat("Orden = ", n_str);
        msg5 = '';
    case 2
        msg2 = strcat("Fstop: ",frec_str{1}, "; Fpass ", frec_str{2});
        msg3 = strcat("Dstop: ",rizo_str{1}, "; Dpass: ", rizo_str{2});
        msg4 = strcat("Orden = ", n_str);
        msg5 = '';
    case 3
        msg2 = strcat("Fstop1: ",frec_str{1}, "; Fpass1: ", frec_str{2});
        msg3 = strcat("Fpass2: ", frec_str{3}, "; Fstop2: ", frec_str{4});
        msg4 = strcat("Dstop1: ",rizo_str{1}, "; Dpass: ", rizo_str{2},...
                    "; Dstop2: ", rizo_str{3});
        msg5 = strcat("Orden = ", n_str);
    case 4
        msg2 = strcat("Fpass1: ",frec_str{1}, "; Fstop1: ", frec_str{2});
        msg3 = strcat("Fstop2: ", frec_str{3}, "; Fpass: ", frec_str{4});
        msg4 = strcat("Dpass1: ",rizo_str{1}, "; Dstop: ", rizo_str{2},...
                    "; Dpass2: ", rizo_str{3});
        msg5 = strcat("Orden = ", n_str);
    end
    set(handles.filtromostrar, 'String', [msg1 msg2 msg3 msg4 msg5]);
%Funcion para exportar los coeficientes del filtro en formato 16Bits_T
function export_coef(hObject, eventdata, handles)
global filtro;

[file,path] = uiputfile ('coef.h', 'Guardar filtro', 'C:\Users\tauro\Documents\workspace\Audio\' );
ruta = strcat(path, file)
archivo=fopen(ruta,'w');
f=int16(filtro*32767);      %Conversion de los coeficientes a 16Bits
l=length(filtro);

%Formato para que se pueda leer desde el CodeComposerStudio

fprintf(archivo, '#include "tmwtypes.h"\n');
fprintf(archivo, 'const int BL = %i;\n',1);
fprintf(archivo, 'const int16_T B[%i] = {\',1);

for c = 1:l
    if c == 1
        fprintf(archivo,'%i', f(c));
    else
        fprintf(archivo,'%i, ', f(c));
    end
end
fprintf(archivo, '};\n');
fprintf(archivo, '#define BUFFER_SIZE %i\n',l);

```

```
fclose(archivo);

%-----CreateFcn-----
function original_CreateFcn(hObject, eventdata, handles)

function Frq_original_CreateFcn(hObject, eventdata, handles)

function Frq_filtrado_CreateFcn(hObject, eventdata, handles)

function fsmostrar_CreateFcn(hObject, eventdata, handles)

function filtromostrar_CreateFcn(hObject, eventdata, handles)

function Filtro_CreateFcn(hObject, eventdata, handles)

function filtrar_CreateFcn(hObject, eventdata, handles)

function namemostrar_CreateFcn(hObject, eventdata, handles)

function paneo_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function metodo_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function slider2_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end
```


Apéndice B

Programa en C

B.1. config.h

```
Int16 AIC3204_rset( UInt16 regnum, UInt16 regval )
{
    UInt16 cmd[2];
    cmd[0] = regnum & 0x007F;           // 7-bit Device Register
    cmd[1] = regval;                  // 8-bit Register Data

    EZDSP5535_waitusec( 300 );

    return EZDSP5535_I2C_write( AIC3204_I2C_ADDR, cmd, 2 );
}

void init_aic3204 () {
/* Configure AIC3204 */
    AIC3204_rset( 0, 0x00 ); // Select page 0
    AIC3204_rset( 1, 0x01 ); // Reset codec
    EZDSP5535_waitusec(1000); // Wait 1ms after reset
    AIC3204_rset( 0, 0x01 ); // Select page 1
    AIC3204_rset( 1, 0x08 ); // Disable crude AVDD generation from DVDD
    AIC3204_rset( 2, 0x01 ); // Enable Analog Blocks, use LDO power
    AIC3204_rset( 123, 0x05 ); // Force reference to power up in 40ms
    EZDSP5535_waitusec(50000); // Wait at least 40ms
    AIC3204_rset( 0, 0x00 ); // Select page 0

/* PLL and Clocks config and Power Up */
    AIC3204_rset( 27, 0x0d ); // BCLK and WCLK are set as o/p; AIC3204(Master)
    AIC3204_rset( 28, 0x00 ); // Data offset = 0
    AIC3204_rset( 4, 0x03 ); // PLL setting: PLLCLK <- MCLK, CODEC_CLKIN <-PLL CLK
    AIC3204_rset( 6, 0x07 ); // PLL setting: J=7
    AIC3204_rset( 7, 0x06 ); // PLL setting: HI_BYT(E=D=1680)
    AIC3204_rset( 8, 0x90 ); // PLL setting: LO_BYT(E=D=1680)
    AIC3204_rset( 30, 0x88 ); // For 32 bit clocks per frame in Master mode ONLY
    // BCLK=DAC_CLK/N =(12288000/8) = 1.536MHz = 32*fs
    AIC3204_rset( 5, 0x91 ); // PLL setting: Power up PLL, P=1 and R=1
    EZDSP5535_waitusec(10000); // Wait for PLL to come up
    AIC3204_rset( 13, 0x00 ); // Hi_Byt(DOSR) for DOSR = 128 decimal or 0x0080 DAC oversampling
    AIC3204_rset( 14, 0x80 ); // Lo_Byt(DOSR) for DOSR = 128 decimal or 0x0080
    AIC3204_rset( 20, 0x80 ); // AOSR for AOSR = 128 decimal or 0x0080 for decimation filters 1 to 6
    AIC3204_rset( 11, 0x82 ); // Power up NDAC and set NDAC value to 2
    AIC3204_rset( 12, 0x87 ); // Power up MDAC and set MDAC value to 7
    AIC3204_rset( 18, 0x87 ); // Power up NADC and set NADC value to 7
    AIC3204_rset( 19, 0x82 ); // Power up MADC and set MADC value to 2
```

```

/* DAC ROUTING and Power Up */
AIC3204_rset( 0, 0x01 ); // Select page 1
AIC3204_rset( 12, 0x08 ); // LDAC AFIR routed to HPL
AIC3204_rset( 13, 0x08 ); // RDAC AFIR routed to HPR
AIC3204_rset( 0, 0x00 ); // Select page 0
AIC3204_rset( 64, 0x02 ); // Left vol=right vol
AIC3204_rset( 65, 0x00 ); // Left DAC gain to 0dB VOL; Right tracks Left
AIC3204_rset( 63, 0xd4 ); // Power up left,right data paths and set channel
AIC3204_rset( 0, 0x01 ); // Select page 1
AIC3204_rset( 16, 0x00 ); // Unmute HPL , 0dB gain
AIC3204_rset( 17, 0x00 ); // Unmute HPR , 0dB gain
AIC3204_rset( 9, 0x30 ); // Power up HPL,HPR
EZDSP5535_waitusec(100); // Wait

/* ADC ROUTING and Power Up */
AIC3204_rset( 0, 0x01 ); // Select page 1
AIC3204_rset( 52, 0x30 ); // STEREO 1 Jack
                           // IN2_L to LADC_P through 40 kohm
AIC3204_rset( 55, 0x30 ); // IN2_R to RADC_P through 40 kohmm
AIC3204_rset( 54, 0x03 ); // CM_1 (common mode) to LADC_M through 40 kohm
AIC3204_rset( 57, 0xc0 ); // CM_1 (common mode) to RADC_M through 40 kohm
AIC3204_rset( 59, 0x00 ); // MIC_PGA_L unmute
AIC3204_rset( 60, 0x00 ); // MIC_PGA_R unmute
AIC3204_rset( 0, 0x00 ); // Select page 0
AIC3204_rset( 81, 0xc0 ); // Powerup Left and Right ADC
AIC3204_rset( 82, 0x00 ); // Unmute Left and Right ADC
AIC3204_rset( 0, 0x00 ); // Select page 0
EZDSP5535_waitusec(100); // Wait

/* Initialize I2S */
EZDSP5535_I2S_init();

}

Int16 AIC3204_rget( Uint16 regnum, Uint16* regval )
{
    Int16 retcode = 0;
    Uint16 cmd[2];

    cmd[0] = regnum & 0x007F;           // 7-bit Device Register
    cmd[1] = 0;

    retcode |= EZDSP5535_I2C_write( AIC3204_I2C_ADDR, cmd, 1 );
    retcode |= EZDSP5535_I2C_read( AIC3204_I2C_ADDR, cmd, 1 );

    *regval = cmd[0];
    EZDSP5535_wait( 10 );
    return retcode;
}

Int16 aic3204_test( )
{
    /* Initialize I2C */
    EZDSP5535_I2C_init( );
    init_aic3204();
    return 0;
}

```

B.2. main.c

```
#define AIC3204_I2C_ADDR 0x18

#include "ezdsp5535.h"
#include "ezdsp5535_gpio.h"
#include "ezdsp5535_i2c.h"
#include "stdio.h"
#include "ezdsp5535_i2s.h"
#include "coefs.h"
#include <math.h>
#include <stdlib.h>
#include "config.h"

Int16 rData = 0;
Int16 lData = 0;
Int16 *lAddr;
Int16 *rAddr;
Int16 prueba = 10;
Int16 lOut;

struct stereo {
    Int16 L[BUFFER_SIZE];
    //Int16 R[BUFFER_SIZE];
};

struct stereo dataIn;

//DEBUG///////////
struct proof {
    Int16 muestra[50];
};

struct proof entrada;
struct proof salida;
////////////////////

Int16 fir_convolution (Int16 *x) {
    Int32 yn = 0; /* Output of FIR filter */
    int i; /* Loop index */

    for(i=0; i<BL; i++) {
        yn += (Int32)B[i]*(Int32)x[i];/* Convolution of x(n) with h(n) */
    }
    yn = yn>> 15;
    return((Int16)yn);
}

void shift (Int16 *x, int loop) {
    int i; /* Loop index */
    for(i=loop-1; i> 0; i--) {
        x[i] = x[i-1]; /* Shift old data x(n-i) */
    }
}

void filtro () {

    Int16 *signal_buffer=(Int16 *)calloc(BL, sizeof(Int16));
    while(1){

        EZDSP5535_I2S_readLeft(lAddr);
        //EZDSP5535_I2S_readRight(rAddr);

        dataIn.L[0]=*lAddr;
        lOut = fir_convolution(dataIn.L);
        shift(dataIn.L, BL);

        entrada.muestra[0]=lData;
    }
}
```

```
    shift(entrada.muestra, 50);

    salida.muestra[0]=l0ut;
    shift(salida.muestra, 50);

    EZDSP5535_I2S_writeRight(l0ut);
    EZDSP5535_I2S_writeLeft(lData);
    //EZDSP5535_I2S_writeLeft(*lData);

}

}

void main() {

    lAddr = &lData;
    rAddr = &rData;
    EZDSP5535_init( );
    aic3204_test( );

    filtro();
}
```

Bibliografía

- [1] Pablo Leonidas Cevallos Duque and Jorge Mauricio Quitama Haro. Diseño e implementación de un prototipo portátil de ecualizador de audio de bajo costo que funcione en tiempo real para compensar las deficiencias auditivas humanas. B.S. thesis, QUITO/EPN/2013, 2013.
- [2] Juan-L Albadalejo-Lijarcio. Separación de fuentes sonoras emitidas por el corazón y pulmones aplicado al diagnóstico de enfermedades cardiovasculares y respiratorias. 2016.
- [3] José Manuel Motta. Sistema de audio compatible con resonancia magnética. 2017.
- [4] Francis F Li and Trevor J Cox. *Digital Signal Processing in Audio and Acoustical Engineering*. CRC Press, 2019.
- [5] Sen Kuo, Bob Lee, and Wenshun Tian. *Real-Time Digital Signal Processing: Fundamentals, Implementations and Applications, 3rd Edition*. Wiley, 3rd ed. edition, 2013.
- [6] Marcelo Semeria. Los tres teoremas: Fourier-nyquist-shannon. Technical report, Serie Documentos de Trabajo, 2015.
- [7] José Pablo Alvarado Moya. Procesamiento digital de señales. *Instituto Tecnológico de Costa Rica*, 2011.
- [8] Jose Maria Giron-Sierra. *Digital signal processing with matlab examples, volume 1: signals and data, filtering, non-stationary signals, modulation*. Springer, 2016.

- [9] Leon Cohen. *Time-frequency analysis*, volume 778. Prentice hall, 1995.
- [10] Emilia Gómez Gutiérrez. Introducción al filtrado digital. *Catalunya: Escola Superior de Musica de Catalunya, Departamento de Sonología*, 2009.
- [11] MC Casado Fernández. Manual básico de matlab. *Edit. Complutense, Madrid*, 2009.
- [12] Code Composer Studio. Getting started guide. *Texas Instruments. – 2001*, 2006.
- [13] Spectrum Digital. Tms320c5535 ezdsp technical reference, 2011.
- [14] Rahulbehl. TMS320C5535 eZdsp USB Stick – Audio Playback, 05 2015.