UNIVERSIDAD DEL BÍO-BÍO
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

# Avance 1 Programación en c/c++:

# "Implementación de un Filtro FIR"

**Integrantes:**

**Maximiliano Cerda Cid**

**Mauricio Montaneares**

**Profesor:**

**Cristian Durán Faúndez**

**Fecha: 19/11/2021**

# Contenido

Implementación de un filtro FIR en lenguaje C ..... 3
Marco Teórico: ..... 3
Implementación del filtro ..... 5
Lectura de entrada ..... 5
IntToFloat ..... 5
firFloat ..... 6
floatToint: ..... 6
Escritura de salida: ..... 6
Algoritmo de computo Filtro FIR: ..... 6
Observación: ..... 6
Descripción en nivel de diagrama de flujo ..... 7
Aspectos importantes a señalar: ..... 7
Diagrama de flujo: ..... 7
Resultados del código: ..... 9
Bibliografía: ..... 10

## Implementación de un filtro FIR en lenguaje C

A la hora de realizar procesamiento de datos, los filtros son una de las primeras etapas a la hora transmitir y/o recibir información, actualmente la mayoría de las implementaciones de filtrado son implementadas en forma digital y por ende pueden ser escritas mediante algún lenguaje de programación que implementa algún algoritmo de filtrado, como en este caso específico, un filtro de respuesta a impulso finita o filtro FIR. Por otra parte, existen muchos **software** y lenguajes de **script** que pueden implementar y modelar de manera rápida y precisa un filtro de este tipo, sin embargo a la hora de realizar una implementación en tiempo real (como pudiese ser el procesado de voz en una conversación telefónica) lo que se requiere es una implementación eficiente a la hora de ejecutar el código, para dicho caso **C** es una opción viable dado que se caracteriza por ser un **lenguaje de "bajo nivel"** con respecto a otros lenguajes más populares como python, matlab, javascript. Además se pueden hacer optimizaciones interactuando con memorias y registros.

## Marco Teórico:

Un filtro de respuesta finita al impulso **FIR** se trata de un tipo de filtro **digital** cuya respuesta a un entrada tipo impulso tendrá como salida una serie de términos finitos no nulos.

La ecuación discreta que describe dicha definición entrada-salida se define de la siguiente forma:

$$y(n) = \sum_{i=0}^{M-1} b_i x(n-i) \qquad (1)$$

Donde $\boldsymbol{b_i}$ corresponde a los coeficientes obtenidos de la respuesta al impulso del filtro en cuestión **M** corresponde al lago del filtro u orden.

De la ecuación descrita en (1) se pueden inferir la siguiente información:

**1.-** para dicho algoritmo tendremos que implementar un total de **M** multiplicaciones correspondientes al largo dela ventana

**2.-** de dichas multiplicaciones se deberá hacer una suma acumulada del mismo largo de la ventana del filtro (básicamente es realizar una suma ponderada de las entradas).

**3.-** si tuviéramos una entrada de N muestras, el largo del vector requerido es de **N+M-1**

Para la implementación del filtro, existen variados tipos de ventanas y cada una de ellas dará distintos valores de coeficientes, al mismo tiempo distintas ventanas tendrán diferentes pros y contras. Para esta aplicación se utilizó la ventana de **hamming** (2), la ecuación que describe el cómputo de los coeficientes para la ventana con una secuencia en el dominio del tiempo de $\boldsymbol{0 \leq n \leq M-1}$ Es:

$$0.54 - 0.45\cos\frac{2\pi n}{M-1} \qquad (2)$$

Este tipo de ventana erradica El comportamiento oscilatorio en las proximidades del extremo de la banda del filtro se conoce como **fenómeno de Gibbs**.

Por otra parte podemos sintonizar el filtro usando la herramienta **matlab**, específicamente el **toolbox** de **filterdesigner**, con ello poder realizar la sintonización de manera sencilla y obtener los coeficientes del filtro.

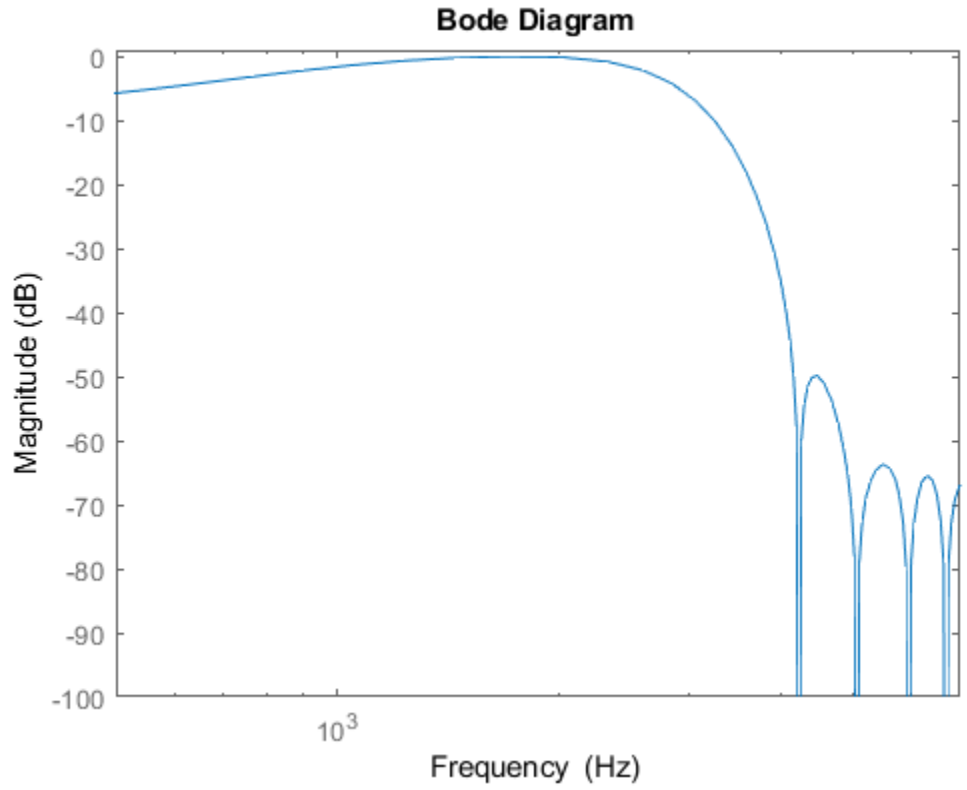


Figura 1 Respuesta en frecuencia del filtro FIR

Para este caso se utilizó un filtro de orden **N=63**, con frecuencia de corte de $\boldsymbol{f_{c1} = 500\ [Hz]}$ y $\boldsymbol{f_{c2} = 3000[Hz]}$ debido a que el foco de uso de este filtro es para un ejemplo de procesamiento de voz, para dicha aplicación esta banda de paso es más que suficiente.

## Implementación del filtro

En primera instancia, la implementación del filtro se puede describir usando un simple diagrama de bloques (Figura 2), en donde cada bloque implementa una función.

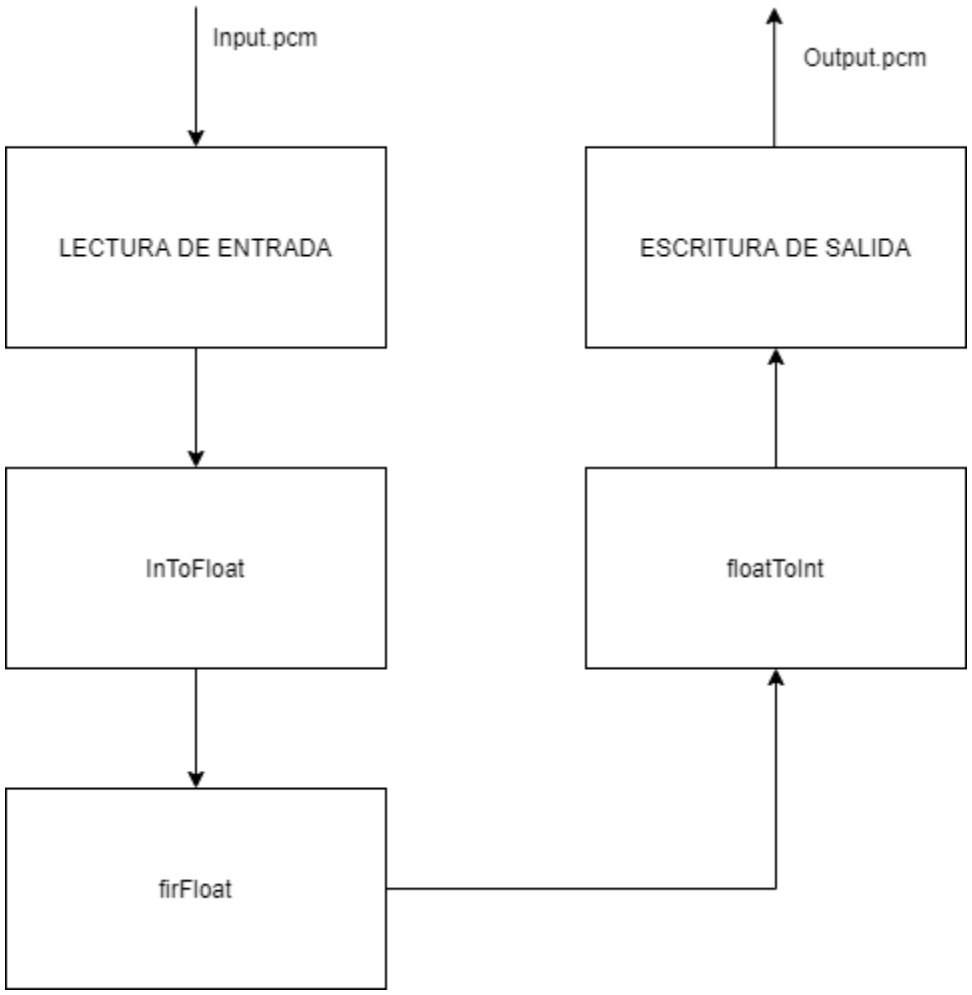


Figura 2 Estructura general del programa

**Lectura de entrada:** es simplemente una función para la lectura de archivos, en este caso se implementa con la función **fopen()** a este bloque, le asociaremos la función **fread()** aunque ambas no son parte de una misma función, cumplen la tarea definida por el bloque de lectura de salida. También cabe destacar que el formato de entrada es un archivo **.pcm** dicho archivo es un formato de modulación por código de pulso muy usado en los sistemas de sonido.

**IntToFloat:** toma los valores de entrada leídos del formato **.pcm** y realiza una conversión numérica a forma doble precisión **double.**

**firFloat:** es la función que realmente realiza la implementación del filtro, recibe todo los valores de entrada de manera directa, y dentro de esta función se realiza la multiplicación y posteriormente las suma acumulada y por consecuencia generando los retardos.

**floatToint:** una vez realizado el computo del filtro FIR, se toman los valores de formato doble precisión y se pasan a un formato de tipo entero **int16**, se debe destacar que dicha conversión es de formato binario de 16 bit con signo a formato entero.

**Escritura de salida:** como en el caso de lectura de entrada, implementaciones de la función **fopen()** y **fwrite()** constituyen al proceso de formato de salida del código, archivo de salida que será de formato .**pcm**

## Algoritmo de computo Filtro FIR:

Unas de las típicas arquitecturas que suele encontrase en los textos sobre procesamiento de señales es la que se muestra en la figura, en dicha arquitectura suele hacerse la multiplicación por cada coeficiente una vez la señal de entrada vaya pasando por cada uno de los retardos, hasta llegar al último coeficiente de la ventana.

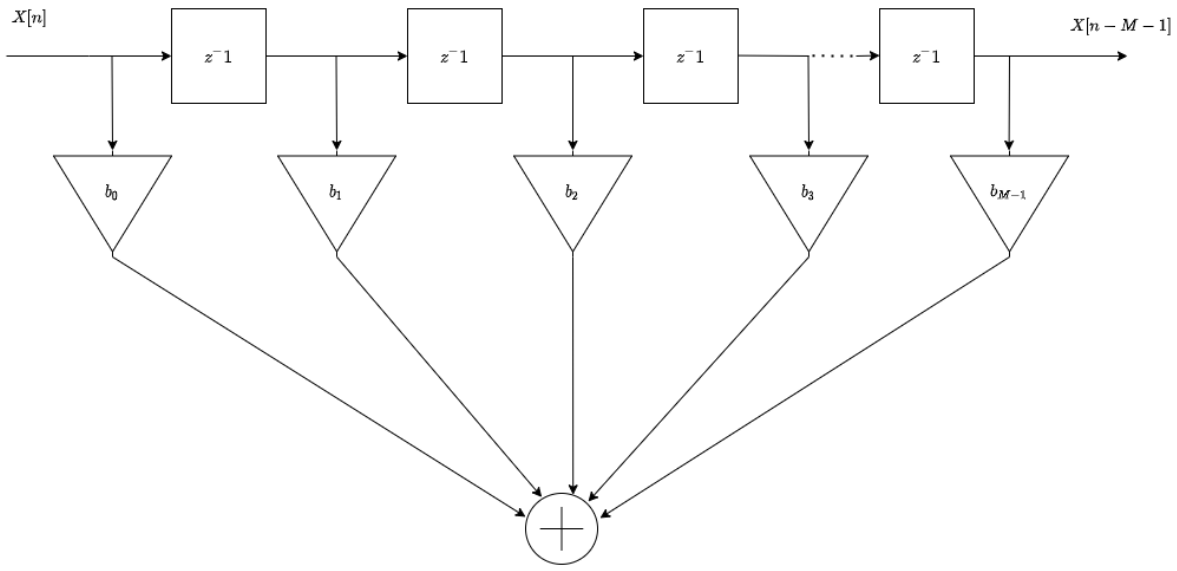


Figura 3 Implementación clásica de filtro tipo FIR

Debemos tener en cuenta que para cada salida habrá una cantidad **N** de multiplicación y acumulación que se suele abreviar como **MAC.** Se realizara un programa capaz de leer 80 muestras en un instante de tiempo, esto para simular la implementación de procesamiento en tiempo real, de esta manera, se realiza solo un llamado a la función que implementa el filtro y posteriormente a la salida se añaden los retardos, lo que hace mucho más eficiente si se llama 80 veces la función para cada muestra.

**Observación:** se elige 80 como el número de muestras procesadas por instante de tiempo debido a que suele ser la cantidad que se procesa en tiempo real para el procesamiento de voz, esto será comprobado con ejemplo posterior donde se utiliza el código del filtro.

## Descripción en nivel de diagrama de flujo

Se realiza una breve explicación en forma de diagrama de flujo de única función que realiza el cómputo de **MAC** dado que las funciones restantes solo realizan un trabajo de lectura y escritura.

**Aspectos importantes a señalar:**
**1.-Definicion de variables globales:**

**#define MAX_INPUT_LEN 80:** define el largo máximo que tendría la entrada del filtro, como dijimos previamente, será capaz de procesar 80 muestras en un instante de tiempo.

**#define MAX_FLT_LEN 63:** establece el largo máximo que tendrá la ventana del filtro, es decir el número de coeficientes de la respuesta impulso que obtuvimos previamente al sintonizar el filtro.

**#define BUFFER_LEN (MAX_FLT_LEN - 1 + MAX_INPUT_LEN):** en función de las dos constantes globales anteriores, se nombra una tercera constante global que decreta la longitud máxima que tendrá el **buffer** de entrada para mantener o almacenar toda las muestras de la entrada, cumple con la condición de vector requerido de N+M-1.

**2.-funciones de librerías string.h:**

| Nombre de la función | Descripción |
|---|---|
| **void *memset(s,c,n)** | **Coloca el carácter c en los primeros n caracteres de s, regresa s.** |
| **void *memcpy(s,ct,n)** | **Copia n caracteres de ct a s y regresa s.** |
| **void *memmove(s,ct,n)** | **Lo mismo que memcpy excepto que funciona aun si los bloques de memoria se traslapan** |

Figura 4 Tabla descriptiva de funciones de la librería string.h

**Diagrama de flujo:**
A continuación se presenta un diagrama de flujo con la secuencia de pasos que realiza la función **firFloat**, como muestra la figura, el primer paso de la función es asignar variables para generar la acumulación del producto como es **double acc**, la generación de dos apuntadores de doble precisión como los son **coeffp** e **inputtp** para los coeficientes y entrada, como también variables para los índices como **n** y **k**, que servirán para los ciclos iterativos.

Luego de generar dichas variables, el primero ciclo iterativo busca recorrer desde el final hasta el principio el arreglo **insamp** y realizar la multiplicación con **coeff**, todo esto usando los punteros. Saliendo de ese ciclo for se entra al siguiente ciclo que realiza el proceso de

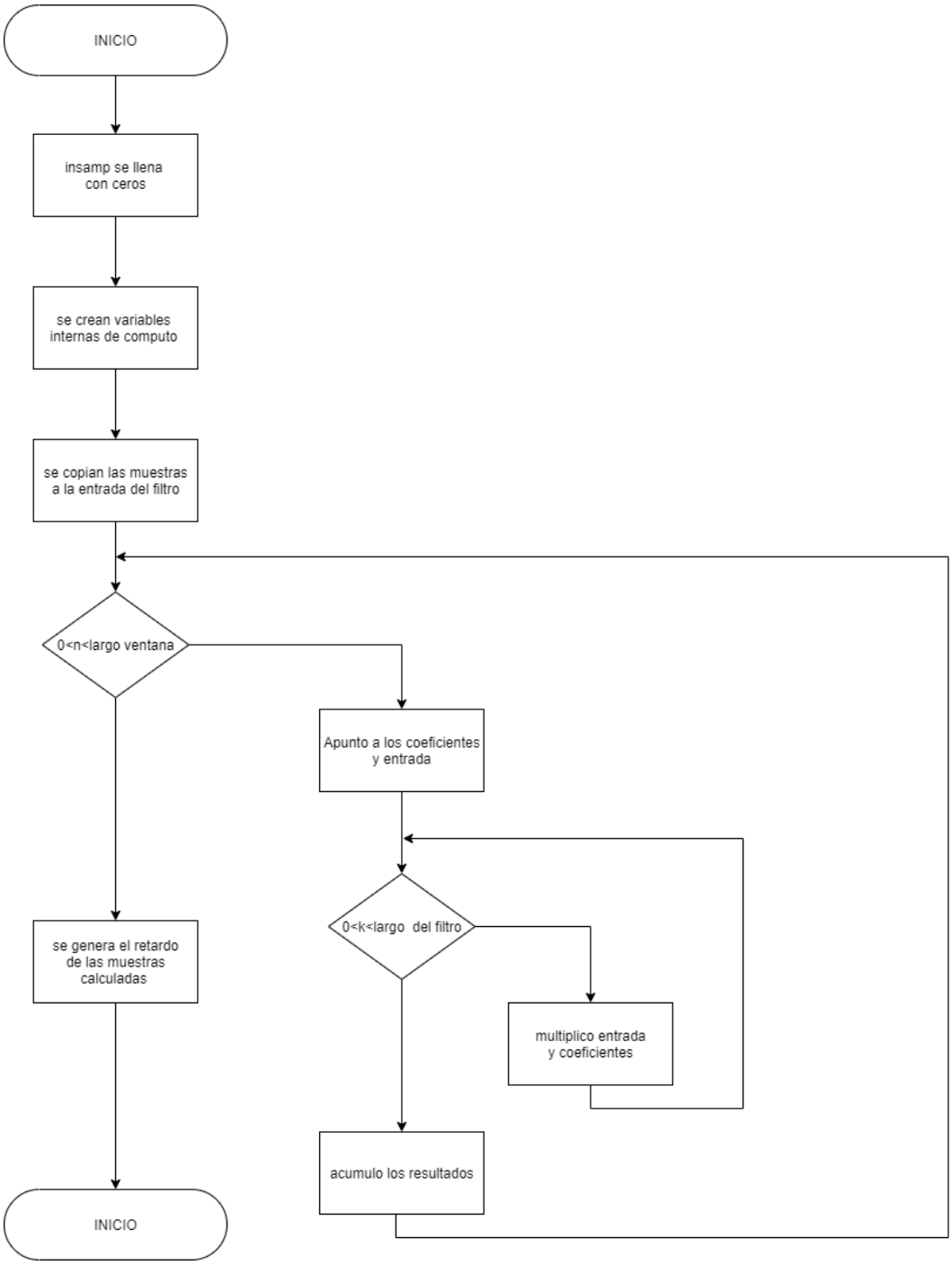


Figura 5 Diagrama de flujo del filtro FIR

acumulación para la muestra actual y finalmente saliendo de esa iteración se generan los respectivos retardos realizando un **shift** mediante la función **memmove.**

## Resultados del código:

Para comprobar el resultado real de código, además de tener una compilación exitosa usando **GNU Compiler Collection,** se generó un archivo de audio en formato **pcm** para la utilización del filtro, posteriormente se hizo una prueba de sonido del archivo de salida en formato **pcm** y el computo de **FFT** de los archivos de entrada y salida utilizando el software **Audacity** pudiendo comprobarse con la similitud de la respuesta en frecuencia de la entrada y la salida como muestran las figuras 6 y 7.

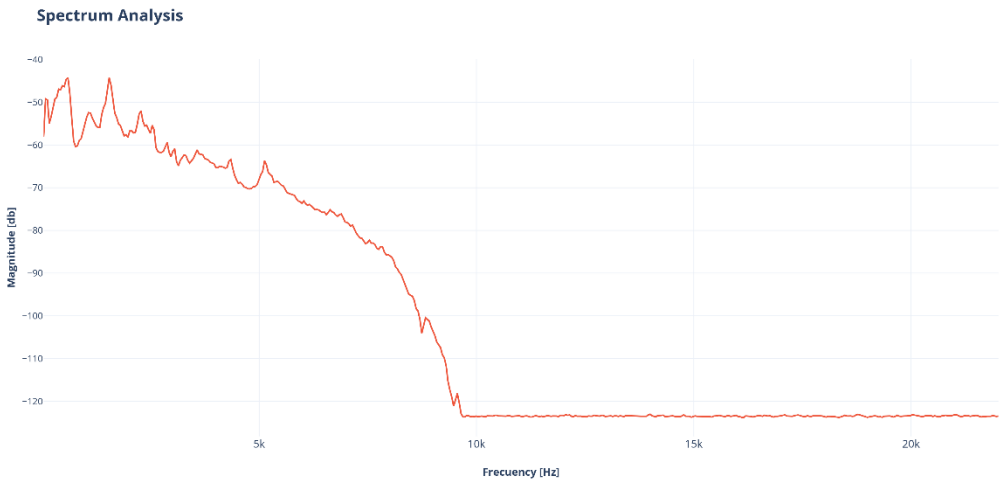


Figura 6 Espectro de la señal de entrada de audio

Una de las diferencias no solo comprobable en la audición, es la eliminación de ciertas frecuencias posterior a los **3 [Khz]** como se muestra en la figura esto es debido a los parámetros del filtro que fueron usados de manera intencional ya que el espectro de voz suele ir dentro de los **250** a **3000 [hz].**

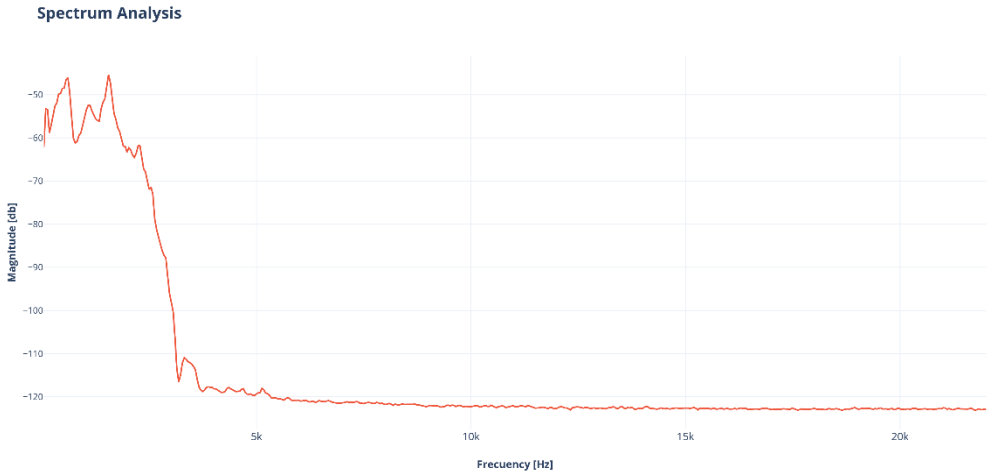


Figura 7 Espectro de la señal de salida de audio

Link de github: https://github.com/HALxmont/C-FIR-FILTER

**Bibliografía:**

**[1]–[3]** [1] J. G. Proakis and D. G. Manolakis, *Tratamiento digital de señales: Principios, algoritmos y aplicaciones.* 2007.

[2] Brian W.Kernighan and Dennis M.Ritchie:, "C Programming Language 2ed," *Prentice Hall*. 1988.

[3] "4c4a3c47af33d9f8ffb1a9bd00168ea5003253aa @ sestevenson.wordpress.com." [Online]. Available: https://sestevenson.wordpress.com/implementation-of-fir-filtering-in-c-part-1/.