RAE

1. **TIPO DE DOCUMENTO:** Proyecto de Grado.
2. **TÍTULO:** Desarrollo de un plugin de Compresión de Audio usando el Banco de Filtros PQMF y la Transformada MDCT en el modelo MPEG-1 Layer II y Layer III: Una comparación de esquemas de compresión.
3. **AUTOR (ES):** Christian Rafael Mora Parga y Julián Esteban Nieto Díaz.
4. **LUGAR:** Bogotá D.C., Colombia.
5. **FECHA:**
6. **PALABRAS CLAVES:** MPEG-1, Transformaciones en frecuencia, Modelo Psicoacústico, Modelos Perceptuales, Compresión, DSP, Análisis Espectral, Síntesis Temporal, Señales y Sistemas.
7. **DESCRIPCIÓN DEL TRABAJO:** Se presenta una documentación y descripción de los algoritmos de compresión de audio MPEG-1 Layer II y III expuestos en la Normativa ISO/IEC-11172-3, adaptados para trabajar en *Tiempo Real* (TR) a través de un plugin. Fueron diseñados algunos algoritmos y readaptados otros de la Normativa para trabajar en TR. Se caracterizaron para su comparación ambos sistemas de compresión -Layer II (PQMF) y Layer III (PQMF-MDCT)- a través de parámetros objetivos como Funciones de Transferencias, THD, entre otras y se realizaron pruebas a un grupo de participantes según la recomendación ITU-BS-1116 y 1534 para la evaluación subjetiva del nivel de calidad de ambos esquemas de compresión.
8. **METODOLOGÍA:** Enfoque cuantitativo.
9. **CONCLUSIONES:** Las magnitudes de las funciones de transferencia de ambos sistemas Layer II y III, muestran variaciones de máximo $\sim \pm 6.2$ dBFS alrededor de los 0 dBFS indicando valores de cuantización fuera del umbral de distorsión por recorte (amplitudes que exceden el rango de -1 y 1). Los THDs resultan ser menores al 1% para todas las bandas inferiores a la 20 del PQMF, indicando que ambos sistemas son apropiados para la reconstrucción de señales. Ambos sistemas no alteran las características originales de distribución estadística de las señales procesadas, debido a sus procesos lineales, PQMF, MDCT y modelos de Cuantización, dando lugar a reconstrucciones fieles de las señales de audio. Las diferencias percibidas por los sujetos de prueba en los dos esquemas de compresión, durante la realización de pruebas subjetivas, es leve al observar que la apreciación global de estos dos Layers de acuerdo con sus promedios y distribución F es similar.

Desarrollo de un plugin de Compresión de Audio usando el Banco de Filtros PQMF y la Transformada MDCT en el modelo MPEG-1 Layer II y Layer III: Una comparación de esquemas de compresión

Christian Rafael Mora Parga, crmora@academia.usbbog.edu.co
Julián Esteban Nieto Díaz, jenieto@academia.usbbog.edu.co

Trabajo de grado presentado para optar al título de Ingeniero de Sonido

Asesor: Felipe Antonio Vallejo Monsalve, Ph.D.

Universidad de San Buenaventura
Facultad de Ingeniería
Ingeniería de Sonido
Bogotá D.C., Colombia
2020

Citar/How to cite	(Mora, & Nieto, 2020) ... (Mora et al., 2020)
Referencia/Reference Estilo/Style: APA 6th ed. (2010)	Mora, C.R. & Nieto, J. E. (2020). *Desarrollo de un plugin de compresión de Audio usando el Banco de Filtros PQMF y la Transformada MDCT en el modelo MPEG-1 Layer II y Layer III: Una comparación de esquemas de compresión.* (Trabajo de grado Ingeniería de Sonido). Universidad de San Buenaventura, Facultad de Ingeniería, Bogotá.

Línea de investigación en Diseño de Sistemas de Audio.

Bibliotecas Universidad de San Buenaventura

Biblioteca Digital (Repositorio)
http://bibliotecadigital.usb.edu.co

- Biblioteca Fray Alberto Montealegre OFM - Bogotá.
- Biblioteca Fray Arturo Calle Restrepo OFM - Medellín, Bello, Armenia, Ibagué.
- Departamento de Biblioteca - Cali.
- Biblioteca Central Fray Antonio de Marchena – Cartagena.

Universidad de San Buenaventura Colombia

Universidad de San Buenaventura Colombia - http://www.usb.edu.co/

Bogotá - http://www.usbbog.edu.co

Medellín - http://www.usbmed.edu.co

Cali - http://www.usbcali.edu.co

Cartagena - http://www.usbctg.edu.co

Editorial Bonaventuriana - http://www.editorialbonaventuriana.usb.edu.co/

Revistas - http://revistas.usb.edu.co/

Agradecimientos

Agradecemos a Dios y cada una de las personas que brindaron de su tiempo y esfuerzo para posibilitar el desarrollo de este proyecto; en especial, agradecemos a nuestro director, Felipe Antonio Vallejo Monsalve. De igual forma, a todos nuestros profesores quienes con su proceso de formación para con nosotros nos han permitido llegar a esta meta.

Gracias a los evaluadores del trabajo por las observaciones, sugerencias y preguntas hechas, las cuales ayudaron a mejorar este escrito.

Gracias a cada uno de nuestros familiares por el apoyo en nuestra formación profesional.

Contenido

Pág.

Lista de Anexos

Lista de figuras

Pág.

Lista de tablas

Pág.

RESUMEN

En el presente trabajo se presenta un estudio documental y comparativo de la implementación de los Esquemas de compresión del MPEG-1 Layer II y Layer III. Para ello se partió de la estructura documental estipulada en la normativa ISO-IEC 11172-3 (1993). Se hizo uso de algoritmos propios, diseñados para el funcionamiento de estos sistemas en formato de plugin en Tiempo Real. Uno de los principales resultados del trabajo es la afirmación de la posibilidad de la implementación de estos esquemas de compresión en Tiempo Real, y la confirmación de que los Esquemas de compresión Layer II y III diseñados para la normativa ISO-IEC 11172-3 (1993), son sistemas de procesamiento lineal cuya representación y reconstrucción de señales de audio es óptima y no presenta deficiencias algorítmicas. Por otro lado, el Esquema de compresión Layer III se presenta como una optimización algorítmica frente al Esquema Layer II, en términos de la síntesis debido al uso de la transformada MDCT.

Palabras clave: MPEG-1, Transformaciones en frecuencia, Modelo Psicoacústico, Modelos Perceptuales, Compresión, DSP, Análisis Espectral, Síntesis Temporal, Señales y Sistemas.

ABSTRACT

This paper presents a comparative and documentary study of the implementation of the Compression Schemes MPEG-1 Layer II and Layer III, based on the documentary structure stipulated on ISO-IEC 11172-3 (1993) normative. The use of propetary algorithms was made for the operation of these systems in Real-Time for the plugin format. One of the main results is the affirmation of the possibility of the implementation of these compression schemes in Real Time, as well, the confirmation that Layer II and III compression Schemes for ISO-IEC 11172-3 (1993), are linear processing systems whose representation and reconstruction of audio signals is optimal and does not present algorithmic deficiencies. On the other hand, the Layer III Compression Scheme is presented as an algorithmic optimization versus the Layer II Scheme, in terms of synthesis due to the use of the MDCT transformation.

Keywords: MPEG-1, Frequency transformations, Psychoacoustic Model, Perceptual Compression, DSP, Spectral Analysis, Temporal Synthesis, Signals and Systems.

INTRODUCCIÓN

En el presente trabajo de grado se destaca el proceso de implementación del Modelo de compresión MPEG-1, con la finalidad de desarrollar un plugin de compresión de audio en *tiempo real* por medio del framework de C++, destinado al DSP y programación de audio, JUCE. Este proyecto surge del planteamiento del grupo de Acústica Aplicada del Semillero de Compresión de Audio Digital, que entre los meses de febrero y noviembre del año 2017 inició la investigación que postulaba la necesidad de implementar *códecs* de audio en Tiempo Real. El trabajo actual fue retomado y reiniciado en el periodo académico 2018-2, por los autores de este escrito, estudiantes Christian R. Mora y Julián E. Nieto. Con este trabajo se pretendió documentar, de forma teórica y práctica (por medio de resultados empíricos de medición) los procesos que involucran el desarrollo de un sistema de compresión de audio cuya base algorítmica está presente en los actuales sistemas de compresión comerciales, y que han sido desarrollado por compañías como Sonnox y Izotope, destacando de estas su valor como herramienta en espacios destinados a la masterización y mezcla de temas musicales, podcast, etc.

Este trabajo ofrece no solamente a académicos, diferentes parámetros que se consideran pertinentes para la eficiencia de trabajo de un ingeniero de audio (como lo es el procesamiento en Tiempo Real), sino que también permite estimular la producción de conocimiento propositivo enfocado a la consolidación de productos de alto rendimiento en la industria musical. Por este motivo se pretende encaminar a los lectores del presente escrito, al análisis crítico de los primeros esquemas de compresión en aparecer en el mercado (codecs MPEG-1 Layer I al III), para su continua optimización y adaptación a las necesidades contemporáneas. Aquí presentamos una documentación teórica y practica de las distintas etapas del MPEG-1 Layer II y III que, al ser puestos a prueba, son comparados por medio de pruebas objetivas y subjetivas/perceptuales.

1. Capítulo I: Problema de Investigación

1.1 Planteamiento del Problema

En el desarrollo histórico de los últimos 60 años, la industria de telecomunicaciones ha trabajado en la optimización de los medios de transporte de información de audio a través de medios digitales para esto, la tecnología ha suscitado la preferencia en el crecimiento de la capacidad de transmisión informática en lugar del desarrollo de métodos que permitan la reducción de la tasa en bits de señales de audio (Bosi & Goldberg, 2004). En el contexto del audio, el *streaming* de este, a través de internet, de la mano con la diversidad de respuestas a la reducción de transmisión de datos, se ha hecho desde diferentes plataformas digitales que implementan en *diferido* diversos algoritmos de compresión de audio (por ejemplo, MP3, OGG, ACC). Esta diversificación de formatos de compresión ha resultado en diversas versiones de un mismo audio bajo el mismo nombre, lo que es crítico para los criterios de calidad originales de una producción de audio (Gans, 2015). En la actualidad, compañías como iZotope, Waves, FabFilter, entre otras, trabajan en el desarrollo de software de *tiempo real* con plugins para anticipar la escucha de un audio en un formato particular, para que así el ingeniero de audio trabaje de manera óptima de acuerdo con las necesidades del *streaming* digital (Herrera, 2016).

A partir de los años noventa, entidades han desarrollado estándares de compresión de audio como el MPEG-1 (Instituto de Fraunhofer), OggVorbis (Fundación-Xipg.Org) y WMA (Windows-Media-Audio), implementando algoritmos matemáticos de investigación fronteriza, para optimizar la calidad sonora en simultánea con la reducción de la tasa de datos. En el trabajo de Bosi & Goldberg (2004) se menciona que estas tecnologías se desarrollaron considerando modelos descriptivos/perceptuales de la escucha humana, los cuales permiten determinar qué estímulos acústicos son irrelevantes; siendo un criterio de procesamiento digital para efectuar dicha compresión. El primer estándar publicado por MPEG, el MPEG-1 (ISO/IEC 11172), desarrolla en sus dos primeras capas (Layer I y II), para un análisis y síntesis de señales de audio, la representación espectral de señales de audio con un banco de filtros convolutivos denominado por Bosi & Goldberg (2004) PQMF (Filtros Espejo de Pseudo-Cuadratura), los cuales permiten una representación del audio en 32 bandas igualmente espaciadas entre cada una de ellas. Gracias a la naturaleza estacionaria e impulsiva (cambios de amplitud repentinos en el tiempo) del audio, a la hora de hacer una reconstrucción temporal de este, con la representación espectral uniforme que permite los PQMF, el sistema se expone a omitir los cambios repentinos (también llamados “ataques”) que presentan las componentes impulsivas en el tiempo. Esto genera distorsiones en el audio de salida, a parte de la potencial aparición de *Aliasing* inherente a los PQMF (Bosi &

Goldberg, 2004, p. 273). Para solventar esta problemática el MPEG diseñó la capa Layer III, la cual utiliza un esquema de detección del comportamiento ondulatorio del audio, usando un esquema de filtros PQMF seguido de la MDCT (Transformada Modificada de Cosenos) con la que se pretende adaptar las ventanas de tiempo de trabajo de las señales para la representación de componentes estacionarias e impulsivas.

De acuerdo con lo descrito, el presente trabajo planteó como pregunta lo siguiente: ¿de qué manera se logró la optimización del esquema de compresión propuesto en la capa Layer II respecto a la capa Layer III? ¿Qué algoritmo se puede diseñar para implementar en *tiempo real* ambos esquemas de compresión del MPEG-1, Layer II y su optimización Layer III?

1.2 Objetivos

1.2.1 Objetivo General

Implementar y comparar la compresión de audio digital dada a partir del modelo MPEG-1 Layer II utilizando el banco de filtros PQMF vs. la implementación del MPEG-1 Layer III utilizando la transformada MDCT, esto a través del desarrollo de un plugin en tiempo real de compresión.

1.2.2 Objetivos Específicos

1. Implementar un plugin de compresión que permita la utilización del módulo de la transformada MDCT y Modelo Psicoacústico 2 en el esquema del MPEG-1 Layer III, excluyendo la codificación de datos.

2. Implementar un plugin de compresión que permita la utilización del módulo del banco de filtros PQMF y Modelo Psicoacústico 2 en el esquema del MPEG-1 Layer II, excluyendo la codificación de datos.

3. Caracterizar y comparar la compresión de la señal a partir de la utilización de la transformada MDCT y del banco de filtros PQMF, con parámetros de rendimiento objetivo: SNR, THD, espectro de potencia de la señal resultante, y función de transferencia de los plugins (uno con MDCT, otro con banco de filtros PQMF).

4. Evaluar con tests subjetivos, con base en la recomendación ITU-1116-1, con el tipo de prueba Double-Blind Triple Stimulus Test, el rendimiento del compresor del MPEG-1 Layer II y MPEG-1 Layer III.

1.3 Justificación

En el grupo de Acústica Aplicada del Semillero de Compresión de Audio Digital, bajo la guía y dirección del profesor Dr. Felipe Vallejo, entre los meses de Febrero y Noviembre del año 2017 existió el proyecto bajo el nombre de “Implementación de algoritmos de compresión de audio digital para el desarrollo de Plugins”. Este tenía como meta

implementar en t*iempo real* el esquema de compresión del MPEG-1 Layer III, en este se dieron esbozos para la interpretación y desarrollo de la algoritmia usada en el presente trabajo.

El hecho de que el tipo de señal de audio que escuchen millones de usuarios suscritos en diversas plataformas de *streaming* en internet -269.2 Millones, según Orús (2020), esté determinado por el tipo de formato digital de compresión, compromete en primera instancia, el trabajo de los ingenieros de audio que se encargan de mezclar y masterizar producciones musicales. De estos ingenieros, depende que el producto final del audio sea de alta fidelidad respecto al audio original (sin compresión en formato PCM). Además, que el último proceso de un ingeniero en producción de audio sea la masterización, permite que se trabaje masterizando según como se escuchará el audio en cierto formato de compresión. Compañías como Izotope y Sonnox, han dado un paso inicial al desarrollar un plugin que permite escuchar en *tiempo real* un audio que está siendo mezclado en un determinado DAW en formato MP3 y ACC, esto como se ha dicho, es útil para el productor musical, ya que le permite trabajar sobre un audio, contemplando la compresión generada por estos formatos en el momento de la mezcla, sin embargo, los algoritmos de estos sistemas ("*Ozone 7*" por Izotope y "*Fraunhofer Pro-Codec*" de Sonnox) están bajo confidencialidad empresarial. De acuerdo con esto, fue importante desarrollar el presente trabajo para diseñar un algoritmo que permitiese ver los efectos que tienen estos esquemas de compresión en la calidad del audio en *tiempo real*. Es importante notar que esto ahorra tiempo a la hora de generar un archivo de audio en cualquier formato de compresión, ya que no es necesario generar el archivo, labor que toma unos minutos para ver el efecto de la compresión sobre la mezcla. En vez de esto, el ingeniero puede escuchar el efecto de la compresión en tiempo real mientras hace la mezcla de audio.

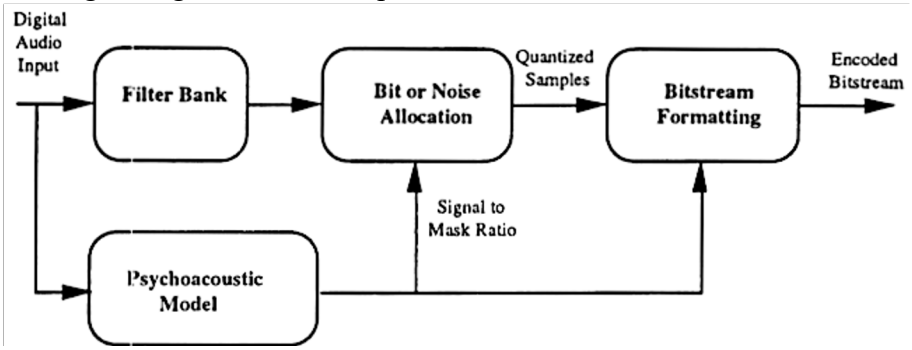
De otro lado, fue necesario comparar el par de esquemas de compresión propuestos por el grupo MPEG (Layer II y Layer III) para caracterizar las diferencias y las optimizaciones del desarrollo frente al Layer II para el diseño del Layer III, y de esta forma, también deducir elementos para la implementación en *tiempo real* de ambos sistemas en el presente trabajo.

2. Capítulo II: Marco Teórico

2.1 Introducción

Existe un compendio de técnicas utilizadas en cada una de las etapas de los *códecs* de audio MPEG, los cuales fundamentan su procesamiento en un análisis perceptual psicoacústico que permite una compresión con pérdidas. En el esquema del MPEG-1, (visualizado en la Figura 2–1) representa las técnicas presentes tales como: el análisis por medio de la representación espectral en el dominio de la frecuencia de las muestras temporales del audio (PQMF, STFT y MDCT), que es utilizado para realizar un proceso de evaluación perceptual llevado a cabo por un modelo psicoacústico (modelo matemático y algorítmico que simula el comportamiento de la escucha humana) y de acuerdo con los criterios objetivos resultantes de este modelo, se manipula el espectro asignando distintos niveles de cuantización para finalmente ser procesado por una etapa de síntesis, el cual lleva estas muestras espectrales, de nuevo al dominio del tiempo digital.

Figura 2–1: Diagrama genérico de bloques del códec de audio del MPEG-1.



Fuente: Imagen tomada de ISO/IEC. 11172-3 (Smith & Abel, 1993, p. 66).

En este capítulo, se describen algunos de los fundamentos básicos para tratar y representar señales en el dominio digital, así como las formas de procesamiento por bloques de muestras, haciendo énfasis en el uso del procesamiento en *tiempo real*. Se muestran las representaciones en tiempo-frecuencia como son los bancos de filtros FIR (QMF y PQMF), y las transformadas puramente en el dominio de la frecuencia DFT/FFT y MDCT, los fundamentos para comprender el modelo psicoacústico 2 utilizado en los Layer II y III, y seguido finalmente de los esquemas de cuantización usados para representar en "códigos" o estados, los valores que forman las señales de audio.

2.2 Técnicas de DSP en tiempo y frecuencia para el MPEG-1 Layer III

Se discuten varias de las técnicas para representar y procesar señales de audio en el dominio de la frecuencia, para así repartir el espectro en sub-bandas, con el propósito de reducir el tamaño de un audio sin perder calidad de escucha eliminando las redundancias perceptuales en señales de audio.

Aquí, la técnica general de procesamiento digital de señales consta en ingresar la señal de audio a través de un banco de filtros que reparten espectralmente la señal en una cantidad *k* de bandas en frecuencia. En la normativa ISO/IEC. 11172-3 (Smith & Abel, 1993, p. 96), y según Kuo, Lee, & Tian (2006, p. 308) y Bosi & Goldberg (2004) quienes toman esta idea para que cada una de estas bandas luego sea cuantizada con un número limitado de bits, que de acuerdo con el nivel de distorsión generada por la cuantización de las líneas de frecuencia y los umbrales de enmascaramiento calculados en el modelo psicoacústico, se ubica el mayor ruido de cuantización en las bandas de menor percepción de la escucha humana. Esta señal cuantizada es luego enviada a un decodificador donde la señal en cada banda es decuantizada y las bandas son combinadas para restaurar el espectro de la señal original.

Al hallar los estados de cuantización adecuados (proporcionales a la cantidad de bits necesarios para representar una muestra) para diversos componentes en frecuencia de una señal de audio, se consigue controlar el nivel de ruido de cuantización en cada uno de éstos, y así darles una mayor prelación a las componentes espectrales relevantes para la escucha humana. Por ejemplo, la cantidad de datos necesarios para representar señales en el tiempo de alta frecuencia es menor que la necesaria para representar señales de baja frecuencia, y así que estos datos representan menos eventos/participaciones, considerándose entonces menos relevantes y por lo tanto descartables.

2.2.1 Descripción de Sistemas Digitales y tipos de Convolución Discreta

Según Kuo et al. (2006) la manipulación de cualquier señal discreta y cuantizada, $x[n]$, ya sea por medio del escalamiento por algún factor constante b_l, por la suma sucesiva de versiones retrasadas en muestras de tiempo para $x[n]$, $\sum_{l=0}^{L-1} x[n-l]$), o por la combinación lineal de varias señales $x_l[n]$, $\sum_{l=0}^{L-1} x_l[n]$, son los tipos de operaciones generalizadas (combinables entre ellas) que describen cualquier tipo de sistema digital lineal invariante en el tiempo (LTI).

Así como en un sistema continuo tipo LTI (que suele ser el descriptor de un sistema físico analógico), se puede describir con una ecuación diferencial ordinaria, que es la superposición de la sucesión de derivadas n-ésimas de una señal de entrada $x(t)$, donde cada término está acompañado por coeficientes constantes α_n. Esta descripción resulta en la función continua $y(t)$ vista en la Ecuación (2.1).

$$y(t) = \sum_{n=0}^{N} \alpha_n \frac{d^n}{dt^n} x(t) = a_0 x(t) + a_1 \frac{dx(t)}{dt} + \alpha_2 \frac{d^2 x(t)}{dt^2} + \cdots + \alpha_N \frac{d^N x(t)}{dt^N} \quad (2.1)$$

Según Oppenheim & Schafer (1998) de forma análoga al dominio continuo, en el dominio discreto del tiempo digital, para la descripción matemática de un sistema LTI donde en vez de una combinación lineal de derivadas sucesivas, se computa la combinación lineal de retrasos de la señal de entrada $x[n]$, esto es visto en la Ecuación (2.2) (es decir que en para funciones continuas, al muestrear sus derivadas de orden l, estos se transforman en retrasos de enésimo orden: $\frac{d^l}{dt^l} x(t) \rightarrow x[n-l]$). Este tipo de superposición llamada *Ecuaciones en Diferencias*, describe a los sistemas "FIR", el cual sus siglas en ingles quieren decir sistema de Respuesta al Impulso Finita.

$$y[n] = \sum_{l=0}^{L-1} b_l x[n-l] = b_0 x[n] + b_1 x[n-1] + \cdots + b_{L-1} x[n-L+1] \quad (2.2)$$

Kuo et al. (2006) indican que al aplicar como señal de entrada, a un sistema discreto, un delta de Kronecker -Ecuación (2.3)-, análogo en dominio del tiempo a delta de Dirac, (el cual se describe como un pulso instantáneo en un momento determinado del tiempo), se llega a la Ecuación (2.4), donde $h[n]$ es la forma en que un sistema tal responde al impulso de entrada $\delta[n]$.

$$\delta[n] = \begin{cases} 1, n = 0 \\ 0, n \neq 0 \end{cases} \quad (2.3)$$

$$h[n] = \sum_{l=0}^{L-1} b_l \delta[n-l] = \{b_0, b_1, b_2, \dots, b_{L-1}\} \quad (2.4)$$

De forma que, al ver la Ecuación (2.4), un sistema FIR, está enteramente descrito por sus coeficientes b_l. Y la sucesión de estos coeficientes es lo que se conoce como respuesta al impulso o $h[n]$. Así que, como es mencionado por Oppenheim & Schafer (1998), cualquier sistema LTI puede realizarse como la convolución -Ecuación (2.2)- entre alguna señal de entrada $x[n]$ con la respuesta al impulso $h[n]$ propia del sistema al que ingresa $x[n]$.

Con una señal de entrada $x[n]$ de longitud finita U, junto a una respuesta al impulso $h[n]$ de longitud finita V, la convolución de estas dos señales resultan en la sucesión $y[n]$ el cual tendrá longitud de tamaño $U + V - 1$. El método de *Convolución Lineal* expresa la relación matemática legítima del proceso de someter a $x[n]$ al sistema descrito por $h[n]$.

$$n \in \mathbb{Z}^+$$
$$h[n] = b_n, \quad n = 0,1, \dots, V-1$$
$$x[n], \quad n = 0,1, \dots, U-1$$

y

$$M = U + V$$

$$y[n] = x[n] * h[n] = \sum_{l=0}^{M-1} h[l]x[n-l] = \sum_{l=0}^{M-1} x[l]h[n-l] \tag{2.5}$$

$$y[n], n = 0, \dots, M-2$$

Por causalidad $x[n < 0]$ y $h[n < 0]$ son iguales a cero, para tiempos anteriores al inicial.

Según Kuo et al. (2006) el proceso de cómputo de la definición directa dada por la Ecuación (2.5), es sumamente ineficiente como algoritmo, ya que no es simple de computar de forma directa haciendo uso de los índices negativos de cada sucesión de arreglos. Se da un ejemplo de la implementación directa de esto a continuación.

$$x[n] = \{7, 6\}, \qquad h[n] = \{5, 3\}$$
$$U = 2, \qquad V = 2, \qquad M = 4$$

$$x[0] = 7, \qquad h[0] = 5$$
$$x[1] = 6, \qquad h[1] = 3$$
$$x[2] = 0, \qquad h[2] = 0$$

$$n = 0, \qquad y[0] = h[0]x[0] + h[1]x[-1] + h[2]x[-2] = 5 \cdot 7 + 3 \cdot 0 + 0 \cdot 0 = 35$$
$$n = 1, \qquad y[1] = h[0]x[1] + h[1]x[0] + h[2]x[-1] = 5 \cdot 6 + 3 \cdot 7 + 0 \cdot 0 = 51$$
$$n = 2, \qquad y[2] = h[0]x[2] + h[1]x[1] + h[2]x[0] = 5 \cdot 0 + 3 \cdot 6 + 0 \cdot 7 = 18$$

$$y[n] = \{35, 51, 18\}$$

El siguiente algoritmo hecho con fines de programación, conocido como *Convolución Circular*, permite utilizar los índices de las sucesiones a convolucionar, lo cual resulta ser un método propicio para ser programado. En este método se introduce un concepto fundamental en el DSP, utilizado en varias ocasiones en el presente proyecto de grado, el cual es "*zero-padding*" o *relleno de ceros*, el cual consta de introducir ceros de forma explícita en ambas sucesiones (a diferencia de la *convolución lineal*). La técnica dicta que se deben introducir al menos tantos ceros en cada sucesión como la longitud propia de cada una, esto con el fin de que el procedimiento dé cómo resultado lo que se esperaría en una *convolución lineal* visto de forma gráfica en la Figura 2–2. En las siguientes secciones se verá el efecto de introducir ceros en un par de sucesiones, generando una interpolación en el dominio de la frecuencia.

El método de *convolución circular* aplicando el *zero-padding* con el par de sucesiones usadas en el ejemplo anterior se da a continuación.

$$x[n] = \{7, 6, 0, 0\}, \qquad h[n] = \{5, 3, 0, 0\}$$

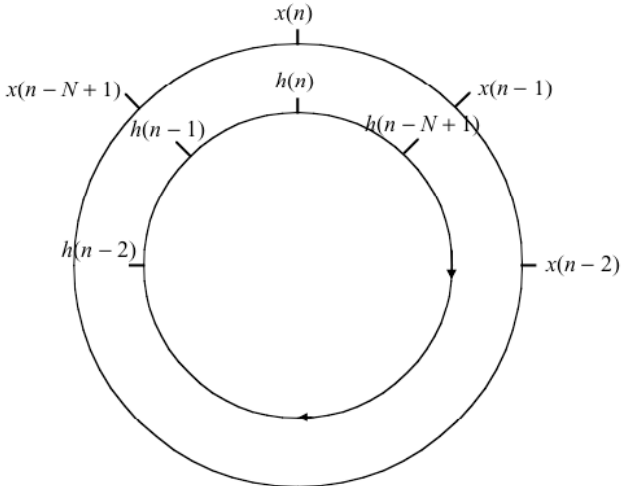
$$y[n] = x[n] \otimes h[n]$$

$$
\begin{aligned}
n = 0, \quad & y[0] = h[0]x[0] + h[3]x[1] + h[2]x[2] + h[1]x[3] \\
n = 1, \quad & y[1] = h[1]x[0] + h[0]x[1] + h[3]x[2] + h[2]x[3] \\
n = 2, \quad & y[2] = h[2]x[0] + h[1]x[1] + h[0]x[2] + h[3]x[3]
\end{aligned}
$$

$$
\begin{aligned}
n = 0, \quad & y[0] = 5 \times 7 + 0 \times 6 + 0 \times 0 + 3 \times 0 = 35 \\
n = 1, \quad & y[1] = 3 \times 7 + 5 \times 6 + 0 \times 9 + 0 \times 0 = 51 \\
n = 2, \quad & y[2] = 0 \times 0 + 3 \times 0 + 5 \times 0 + 0 \times 0 = 18
\end{aligned}
$$

$$y[n] = \{35, 51, 18\}$$

Figura 2–2: Ilustración del método de convolución circular.



Fuente: Imagen tomada de Kuo et al. (2006, p. 312).

Del método de *Convolución Circular*, cabe resaltar que es un proceso en el dominio del tiempo, y de este, nace un hecho importante tal como lo mencionan Kuo et al. (2006); el cual a diferencia del dominio continuo del tiempo, hace que la Transformada de Fourier de la convolución lineal $y(t) = x(t) * h(t)$, genera directamente el producto de las transformadas $X(j\Omega)$ y $H(j\Omega)$ de las dos señales $x(t)$ y $h(t)$ (lo que permite tener una descripción "completa" de $Y(j\Omega)$ en el dominio de la frecuencia, gracias a los infinitos puntos continuos de ésta). Mientras que para obtener dicha descripción "completa" para un par discreto de señales $x[n]$ y $h[n]$, se debe realizar la *Transformada Z* de manera teórica. Por lo que estos autores expresan, se computa la Transformada Discreta de Fourier (DFT) o su versión optimizada (FFT) de cada una de las señales individuales con *zero-padding*, y esta técnica llamada *Convolución Rápida* es el equivalente de la *Convolución Circular* en dominio del tiempo.

$$\text{FFT}\{y[n]\} = Y[k] = \text{FFT}\{x[n] \otimes h[n]\} = X[k] \cdot H[k] \tag{2.6}$$

Si se aplica la FFT, se conoce que esta trabaja con señales (sucesiones) con tamaños de los números que producen las potencias del 2, como se expresa en la Ecuación (2.7). Así que a veces se completan con ceros estas señales para poder aplicar la FFT.

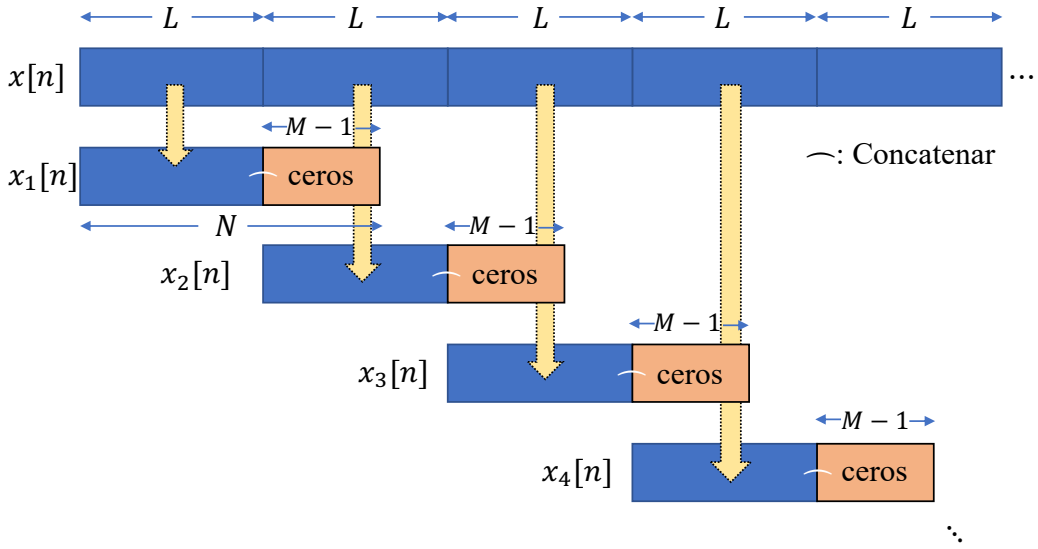
$$2^m = \{1,2,4,8,16,32,64,128,256,512,1024,\dots\}, \qquad m \in \mathbb{Z}^+ \tag{2.7}$$

En las siguientes secciones 2.2.2 y 2.2.3 se observa a través de la convolución, uno de los fenómenos más relevantes en estas transformaciones lineales, conocido como Aliasing Temporal, que permea la Transformada Modificada de Cosenos (MDCT) utilizada en el algoritmo del MPEG-1 Layer III.

2.2.2 Técnicas de convolución lineal en Tiempo Real, Overlap & Add

A la hora de hacer una transformación de un dominio a otro, para manipular una señal en dicho dominio y aprovechar las propiedades de dicho espacio funcional, es necesario eventualmente regresar al dominio original para obtener las señales finalmente procesadas. Lo anterior es el caso de la descomposición espectral que se hace a través del modelo de filtros híbridos PQMF-MDCT, donde primero se convoluciona 32 veces en paralelo un bloque de audio de entrada con los filtros PQMF, obteniendo la señal representada en 32 bandas igualmente espaciadas, y seguido de esto, cada una de estas 32 señales de contenido temporal, es transformada en 18 líneas reales de frecuencia con la transformada MDCT. Una vez logrado lo anterior, es posible volver al dominio del tiempo a partir de los mapeos inversos de cada uno de estos dos módulos, es decir, reconstruir la señal original implementando la transformada inversa IMDCT seguido de los filtros inversos PQMF. Esta sección trata de dos esquemas generales que ayudan a logran tales objetivos de reconstrucción temporal.

Figura 2–3: Discretización de la señal de entrada en bloques de tamaño L para el Overlap & Add.



Fuente: Imagen de elaboración propia.

Uno de los procedimientos para generar una transformación lineal en tiempo real o no, como bien puede ser una convolución rápida (por ejemplo, por medio de la Transformada de Fourier de Tiempo Corto) aplicada con alguna señal en forma de respuesta al impulso, es por medio del proceso descrito a continuación llamado Overlap & Add. Este proceso consta de discretizar una señal en trozos de tamaño L, concatenar ceros a éstos, y calcular

posteriormente la convolución (Rápida, Circular, Lineal o con algún otro algoritmo), para finalmente superponer estos fragmentos de arreglos. Kuo et al. (2006) mencionan de forma general algunos de los siguientes pasos:

1) Se computa la Transformación Espectral a la respuesta al impulso $h[n] \rightarrow H[k]$. Siendo el anterior un arreglo, puede almacenarse sin necesidad de volverse a calcular si solo se usan una vez en todo el procesamiento.

2) De la señal temporal $x[n]$, se generan secuencias finitas $x_\ell[n]$, de longitud $N = L + M - 1$, donde las primeras L muestras correspondes a L muestras actuales de $x[n]$, y se concatenan $M - 1$ ceros (*zero-padding*). Este proceso se visualiza en la Ecuación (2.8) y en la Figura 2–3. Se tienen L muestras del bloque ℓ y $M - 1$ ceros.

$$n = 0,1,\dots,N-1. \qquad N = L + M - 1$$

$$\begin{aligned} x_1[n] &= \{x[0], x[1], \dots, x[L-1], 0, \dots, 0\} \\ x_2[n] &= \{x[L], x[L+1], \dots, x[2L-1], 0, \dots, 0\} \\ &\vdots \\ x_\ell[n] &= \{x[L(\ell-1)], x[L(\ell-1)+1], \dots, x[\ell L-1], 0, \dots, 0\} \end{aligned} \tag{2.8}$$

3) A cada uno de los trozos $x_\ell[n]$, se computa la Transformación Espectral de tamaño N, hallando $X_\ell[k]$.

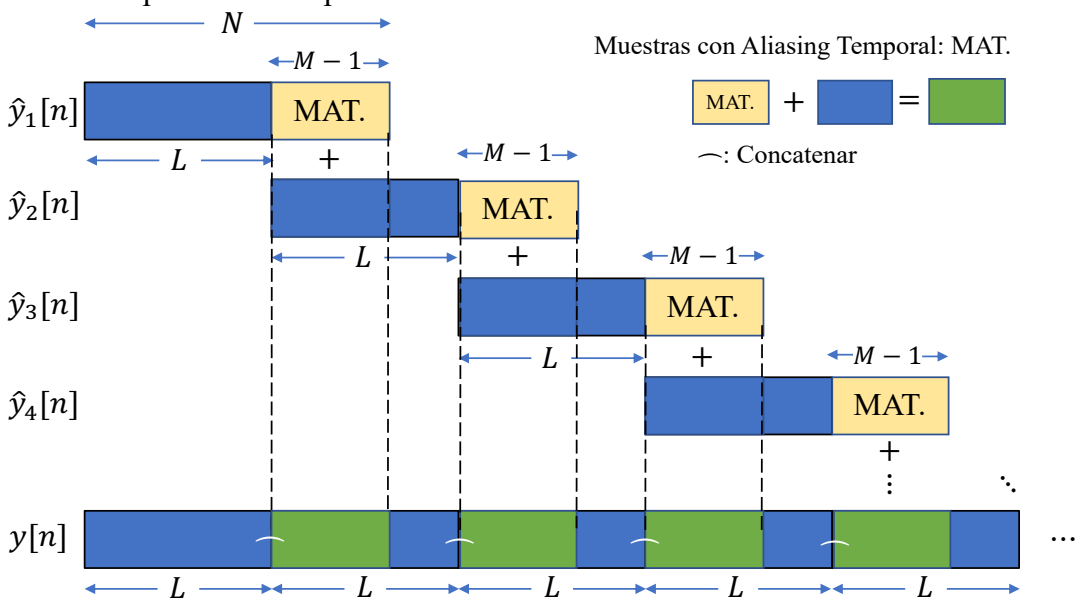
4) Se computa la multiplicación $Y_\ell[k] = X_\ell[k] \cdot H[k]$, para convolucionar el vector $X_\ell[k]$.

5) A cada uno de los trozos $Y_\ell[k]$, se computa la Transformación Inversa de tamaño N, hallándose $\hat{y}_\ell[n]$, los cuales corresponden a una versión con Aliasing Temporal respecto a $x_\ell[n]$.

6) De cada $\hat{y}_\ell[n]$ se deben sumar las últimas $M + 1$ muestras con las primeras $M + 1$ muestras de $\hat{y}_{\ell+1}[n]$, y se concatena el pedazo del bloque $\hat{y}_{\ell+1}[n]$ que no se sobrelape con $\hat{y}_\ell[n]$ y $\hat{y}_{\ell+2}[n]$ a lo que será $y[n]$ en una sucesión de este procedimiento. Este último proceso se visualiza en la Ecuación (2.9) y en la Figura 2–4.

$$y[n] = \left\{ \begin{array}{c} \hat{y}_1[0], \hat{y}_1[1], \dots, \hat{y}_1[L-1], \\ \hat{y}_1[L] + \hat{y}_2[0], \hat{y}_1[L+1] + \hat{y}_2[1], \dots, \hat{y}_1[N-1] + \hat{y}_2[M-1], \\ \hat{y}_2[M], \hat{y}_2[M+1], \dots, \hat{y}_2[L-1], \dots \\ \vdots \\ \hat{y}_\ell[L] + \hat{y}_{\ell+1}[0], \dots, \hat{y}_\ell[N-1] + \hat{y}_{\ell+1}[M-1], \hat{y}_{\ell+1}[M], \hat{y}_{\ell+1}[M+1], \dots \end{array} \right\} \tag{2.9}$$

Figura 2–4: Superposición de bloques de transformaciones inversas parciales de tamaño N para el Overlap & Add.



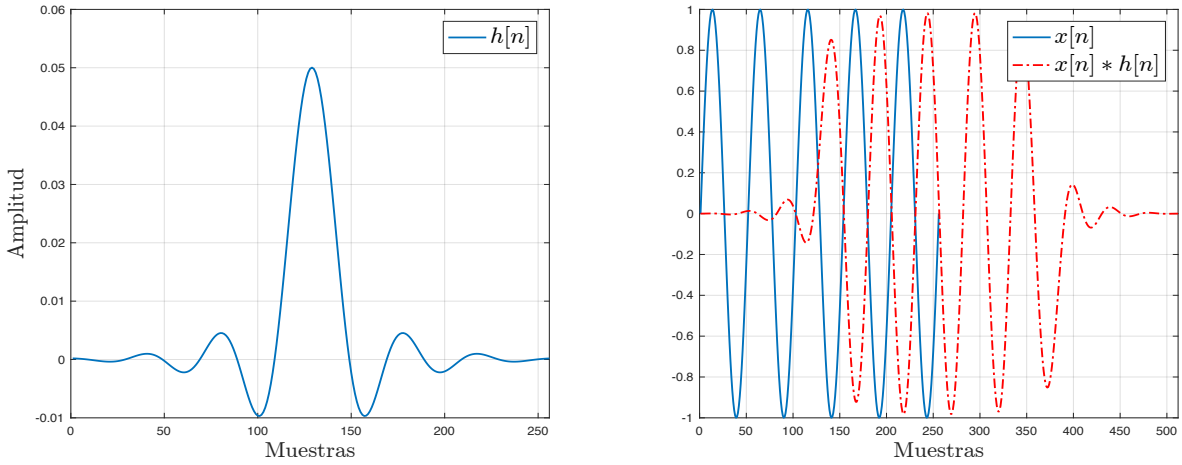
Fuente: Imagen de elaboración propia.

Según Kuo et al. (2006) al concatenar ceros a una señal temporal, y al calcular por ejemplo una DFT de la longitud de esa señal junto a los ceros, se genera una "interpolación" espectral, produciendo que la transformada tenga mayor cantidad de puntos para representar las componentes únicas de frecuencia de tal señal. Pero al hacer la transformación inversa para regresar al dominio del tiempo discreto, existe un exceso de muestras respecto a la señal original (gracias a la correspondiente cantidad de ceros añadidos), que, en caso de haber sido una transformación junto a un vector con valores de una respuesta al impulso, la transformada inversa corresponderá en tiempo a la señal original filtrada por tal vector, además de añadirse un desfase al ser una operación de filtrado.

En la Figura 2–5 se aprecia cómo la señal resultante de la convolución toma la forma de la señal en tiempo como una "combinación" con la forma de la señal $h[n]$, con el importante hecho que esta resultante tiene tantas muestras como la suma de las longitudes de las dos señales convolucionadas.

En la Figura 2–5 se hizo convolución a una señal periódica de 5 ciclos, la versión convolucionada con $h[n]$, muestra estos 5 ciclos en medio de los rizados temporales que pertenecen originalmente a $h[n]$ (que encierran en el centro al gran lóbulo con pico de amplitud 0.5), esto es un ejemplo de lo que Bosi & Goldberg (2004) denominan "Aliasing en el dominio del tiempo". La idea del Aliasing Temporal se retoma de nuevo en la Sección 2.5, cuando se explique las implicaciones matemáticas particulares del códec con la transformada MDCT.

Figura 2–5: Discretización de la señal de entrada en bloques de tamaño L para el Overlap & Add.



Fuente: Imagen de elaboración propia. Matlab_R2018a.

En el caso que el entorno del dispositivo A/D reciba bloques de datos de una longitud predeterminada, cada uno de estos bloques de datos puede ser la secuencia $x_\ell[n]$ (que, si se concatenaran, se generaría eventualmente a $x[n]$ de la Ecuación (2.8). Es conocido que en los entornos DAW y en entornos para programación de audio, que los bloques de audio que ingresan en tiempo real tienen usualmente tamaños correspondientes a las potencias del número 2.

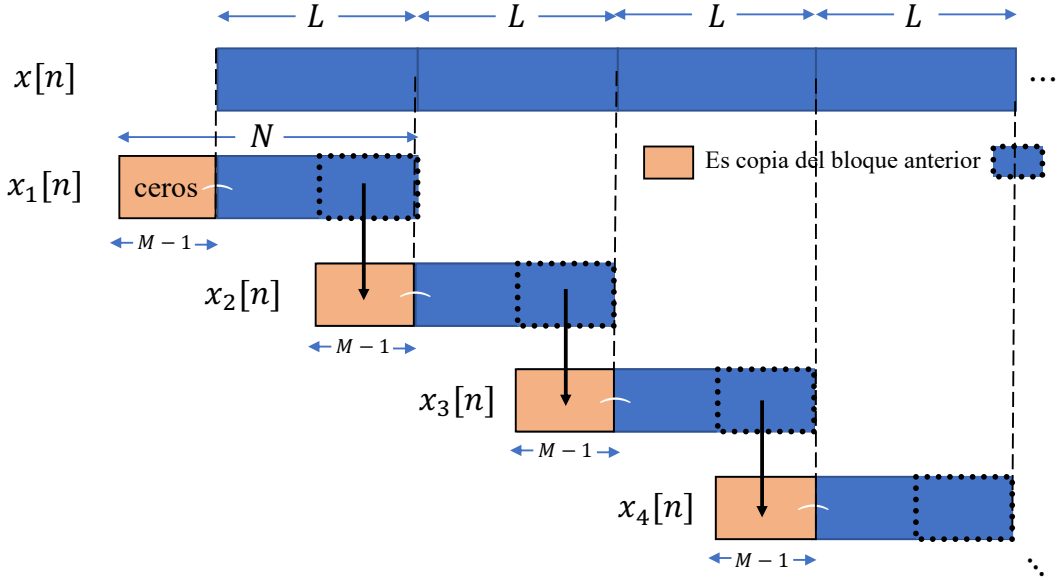
$$2^m = \{1,2,4,8,16,32,64,128,256,512,1024,\dots\}, \qquad m \in \mathbb{Z}^+ \tag{2.10}$$

Si se desea trabajar con bloques de tamaño distinto a las potencias del 2, los bloques entrantes de audio se pueden almacenar en arreglos auxiliares, para ser concatenados y fraccionados a gusto.

2.2.3 Técnicas de convolución lineal Tiempo Real, Overlap & Save

Otra de las técnicas más utilizadas en el DSP según (Oppenheim & Schafer, 1998) para realizar transformaciones lineales es la del Overlap & Save, la cual consta de nuevo en procesar señales finitas $x_\ell[n]$ de longitud N, que a diferencia del método de Overlap & Add, en vez de superponer los vectores, se almacenan trozos de los bloques anteriores, y una vez procesados los bloques completos, se descartan las muestras que suponen un aliasing temporal debido a que se procesaron bloques con secciones de datos iguales.

Figura 2–6: Realización de segmentos de bloques parciales de tamaño N para el Overlap & Save.



Fuente: Imagen de elaboración propia.

Algunos de los pasos mencionados por Kuo et al (2006) y Oppenheim & Schafer (1998) para realizar transformaciones lineales a señales reales a través del método de Overlap & Save son:

1) Al igual que en el procesamiento por Overlap & Add, se computa la Transformación Espectral a la respuesta al impulso $h[n] \rightarrow H[k]$, este puede almacenarse sin necesidad de volverse a calcular si solo se usan una vez en todo el procesamiento.

2) De la señal temporal $x[n]$, se generan secuencias finitas $x_\ell[n]$, de longitud $N = L + M - 1$, donde las primeras $M - 1$ muestras corresponden a $M - 1$ muestras del bloque $x_{\ell-1}[n]$, y las siguientes L muestras corresponden a los datos del arreglo $x[n]$. Este proceso se visualiza en la Ecuación (2.11) y en la Figura 2–6. Se tienen L muestras del bloque ℓ y $M - 1$ ceros. Para $x_1[n]$ se añaden $M - 1$ ceros y seguido, se añaden las primeras L muestras de $x[n]$.

$$n = 0,1,\dots,N-1. \qquad N = L + M - 1$$

$$\begin{gathered} x_1[n] = \{0,0,\dots,0,x[0],x[1],\dots,x[L-1]\} \\ x_2[n] = \{x[L-M+1],x[L-M+2],\dots,x[L-1],x[L],x[L+1],\dots,x[2L-1]\} \\ x_3[n] = \{x[2L-M+1],x[2L-M+2],\dots,x[2L-1],x[2L],x[2L+1],\dots,x[3L-1]\} \\ \vdots \\ x_\ell[n] = \{x[\ell L-M+1],\dots,x[\ell L-1],x[\ell L],x[\ell L+1],\dots,x[(\ell+1)L-1]\} \end{gathered} \tag{2.11}$$

3) A cada uno de los trozos $x_\ell[n]$, se computa la Transformación Espectral de tamaño N, hallando cada uno de los $X_\ell[k]$.

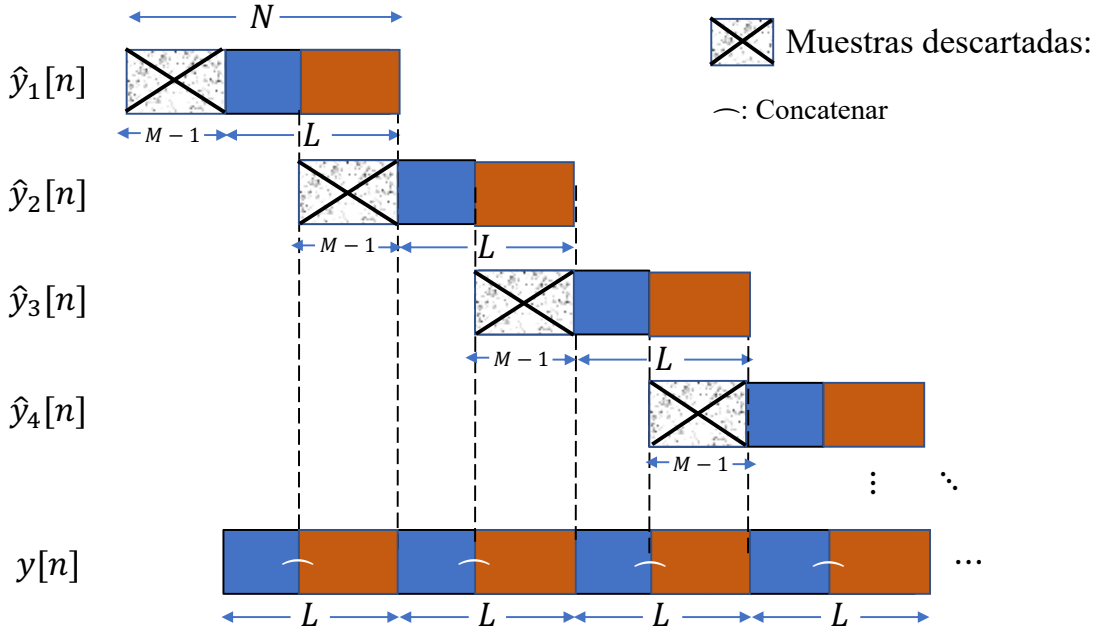
4) Se computa la multiplicación $\hat{Y}_\ell[k] = X_\ell[k] \cdot H[k]$.

5) A cada uno de los trozos $\hat{Y}_\ell[k]$, se computa la Transformación Inversa de tamaño N, hallando $\hat{y}_\ell[n]$, para las cuales las primeras $M-1$ muestras son información redundante respecto a $\hat{y}_{\ell-1}[n]$ (esta es otra forma de Aliasing temporal, donde se transformaron bloques que contenían la misma información).

6) De cada $\hat{y}_\ell[n]$ se deben descartar las primeras $M+1$ muestras ya que es información anteriormente procesada en las últimas $M-1$ muestras de $\hat{y}_{\ell-1}[n]$, y se concatenan las siguientes L muestras con el siguiente bloque al cual también le fueron descartadas las primeras $M-1$ muestras. Este último proceso se visualiza en la Ecuación (2.12) y en la Figura 2–7.

$$y[n] = \begin{Bmatrix} \hat{y}_1[M], \hat{y}_1[M+1], \dots, \hat{y}_1[N-1], \hat{y}_2[M], \hat{y}_2[M+1], \dots, \hat{y}_2[N-1], \dots \\ \dots, \hat{y}_\ell[M], \hat{y}_\ell[M+1], \dots, \hat{y}_\ell[N-1], \dots \end{Bmatrix} \quad (2.12)$$

Figura 2–7: Descarte y asignación de bloques de transformaciones inversas parciales de tamaño N para el Overlap & Save.



Fuente: Imagen de elaboración propia.

Como se explicará en la Sección 2.5, la transformada MDCT implica una combinación de las dos técnicas e ideas del Overlap-Add y Overlap-Save, para poder adquirir una reconstrucción óptima de señales.

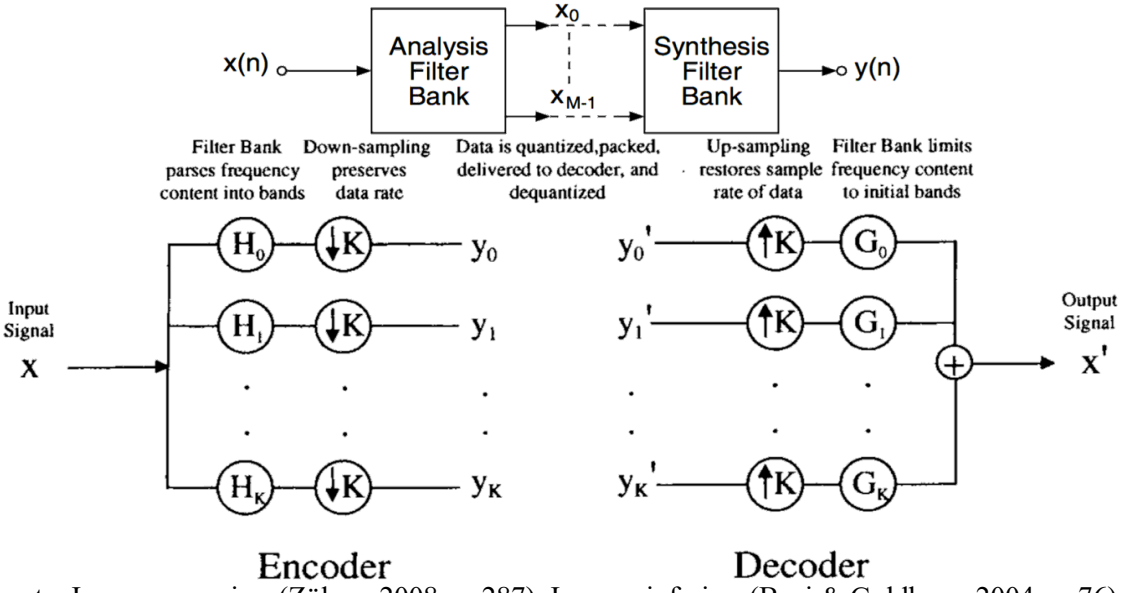
2.3 Filtros Espejo de Pseudo-Cuadratura (PQMF)

2.3.1 Decimación (Down Sampling)

Kuo et al. (2006) mencionan que la técnica con la cual la tasa de muestreo f_s aumenta dado un número entero *U* es llamada *interpolación*, mientras que la técnica que disminuye la tasa de muestreo por un factor entero *D* es llamada *decimación*, en la Figura 2–8 se ilustra la anterior idea al someter una señal de entrada a un banco de filtros.

La decimación de una señal con frecuencia de muestreo f_s' por un factor entero *D* resulta en una tasa de muestreo inferior $f_s'' = f_s'/D$. Esto anterior también puede ser realizado y comprendido al eliminar $D - 1$ muestras de la señal original que se encuentran en medio de las muestras de la señal decimada. El proceso de decimación genera un decrecimiento del ancho de banda de la señal original igualmente por un factor *D*. Además, si la señal original posee componentes en frecuencia por fuera del ancho de banda de la señal a ser decimada, ocurrirán problemas de *aliasing*.

Figura 2–8: Banco de filtros Pseudo-QMF. Muestra de la partición espectral de la señal de entrada $x[n]$, a través de $K + 1$ filtros de análisis $H_{K-1}(z)$ con decimación ($\downarrow$ K). Y reconstrucción temporal convolutiva de $y[n] = x'[n]$ con Overlap & Add dados filtros de síntesis $G_{K-1}(z)$ usando interpolación ($\uparrow$ K).



Fuente: Imagen superior: (Zölzer, 2008, p. 287). Imagen inferior: (Bosi & Goldberg, 2004, p. 76)

Al pasara una señal de audio por un banco de filtros de k bandas espectrales paralelas (tal como se ve en la Figura 2–8), se resulta con $K + 1$ señales del mismo tamaño que la señal original, note que esto ha multiplicado la tasa de muestreo de la señal de audio por el mismo factor K. Esto genera que se aumente la tasa de datos al procesar la señal por estos $K + 1$ filtros, y por esta razón se deben eliminar todas menos una de cada K muestras de la salida de cada filtro, para mantener constante la tasa de muestreo.

Los Filtros de Espejo de Cuadratura, permiten generar este par de estados de análisis y síntesis para generar una reconstrucción casi perfecta al ser usada en secuencia con la decimación. La representación de la Figura 2–8, se visualiza en la forma matricial que se genera por una sucesión de convoluciones en la Ecuación (2.28), donde cada fila representa la señal $x[n]$ en una banda k.

2.3.2 Filtros Espejo de Cuadratura (QMF)

Rothweiler (1983) junto a Bosi & Goldberg (2004) definen a esta clase de filtro $h_0[n]$ como aquel cuya magnitud es la imagen reflejada alrededor de $\pi/2$ (en el dominio de z) de la de otro filtro $h_1[n]$. Este par de filtros juntos, se conocen como Filtros Espejo de Cuadratura (QMF). El desarrollo teórico de esta sección se hará para tan solo dos filtros y sus inversos, en la siguiente sección se verá cómo se trabaja con 32 filtros.

Así, por motivos de síntesis, posteriores a una etapa de análisis por convolución, a partir de un par de filtros $h_0[n]$ y $h_1[n]$, se desea filtrar y reconstruir de manera perfecta la señal de entrada $x[n]$. Para esto se deben establecer cuáles son el par de filtros de síntesis $g_0[n]$ y $g_1[n]$ necesarios, los cuales "deconvolucionen" al par de filtros de análisis $h_0[n]$ y $h_1[n]$, y así poder obtener una "reconstrucción perfecta" de $x[n]$. La señal que se recupera a partir de la etapa de síntesis se denotará como $x'[n]$.

Las Transformada *Z* de los respectivos filtros de análisis y síntesis son:

$$TZ\{h_{(0,1)}[n]\} = H_{(1,2)}(z), \qquad TZ\{g_{(0,1)}[n]\} = G_{(1,2)}(z) \tag{2.13}$$

y de la señal de entrada $x[n]$ filtrada por ellos son:

$$\begin{aligned} TZ\{x[n] * h_0[n]\} &= Y_0(z), \qquad TZ\{x[n] * h_1[n]\} = Y_1(z) \\ Y_0(z) &= X(z) \cdot H_0(z), \qquad Y_1(z) = X(z) \cdot H_1(z) \end{aligned} \tag{2.14}$$

Como lo ilustra la Figura 2–8, la resultante del Overlap & Add en el dominio de z para la reconstrucción de $x[n]$ es $x'[n]$, en la que Bosi & Goldberg (2004) establecen la condición de "cuadratura" debido a la decimación, elevando a la potencia de dos el dominio de la expresión (2.14):

$$TZ\{x'[n]\} = X'(z) = Y_0(z^2)G_0(z) + Y_1(z^2)G_1(z) \tag{2.15}$$

El filtro $H_0(z)$ debe estar sometido a una restricción de simetría respecto a un segundo filtro $H_1(z)$, estando a un giro de π respecto a $H_0(z)$, lo cual es:

$$\pm H_0(\pm z) = \mp H_1(\mp z) \tag{2.16}$$

De la cual la respuesta al impulso de $H_0(z)$ es

$$h_0[n] = -(-1)^n h_1[n] \tag{2.17}$$

Si $h_1[n]$ define un filtro Pasa Bajos, luego $h_0[n]$ es un filtro Pasa Altos. Lo anterior se puede comprobar hallando la magnitud de la respuesta en frecuencia de $h_0[n]$, haciendo $z = e^{j\omega}$:

$$\left|H_0\left(e^{j\omega}\right)\right| = \left|H_1\left(e^{j(\omega-\pi)}\right)\right| \tag{2.18}$$

Lo cual en términos estrictamente digitales es:

$$|H_0(f)| = \left|H_1\left(f - \frac{f_s}{2}\right)\right| \tag{2.19}$$

Bosi & Goldberg (2004) muestran que con la forma de la Ecuación (2.14), y la representación de $Y_i(z)$ a partir de productos, y las condiciones de simetría de la Ecuación (2.16) que se puede representar la síntesis parcial $Y_i(z)$ aplicando la interpolación o *up-sampling* elevando a la raíz cuadrada el dominio de los filtros utilizados (para eliminar el efecto de la decimación de la potencia de dos), y así se pueda representar como la suma:

$$\begin{aligned} Y_i(z) &= \frac{1}{2}\left[H_i\left(z^{\frac{1}{2}}\right)X\left(z^{\frac{1}{2}}\right) + H_i\left(-z^{\frac{1}{2}}\right)X\left(-z^{\frac{1}{2}}\right)\right], \qquad i = 0,1 \\ Y_i(z^2) &= \frac{1}{2}[H_i(z)X(z) + H_i(-z)X(-z)] \end{aligned} \tag{2.20}$$

En la Ecuación (2.20) las partes representadas por $-z$ muestran los contenidos espectrales que se repiten en el espectro negativo del dominio de z, es decir el Aliasing. Reuniendo las Ecuaciones (2.15) y (2.20), $X'(z)$ resulta en:

$$X'(z) = \frac{1}{2}[H_0(z)X(z) + H_0(-z)X(-z)]G_0(z) + \frac{1}{2}[H_1(z)X(z) + H_1(-z)X(-z)]G_1(z)$$

Reorganizando los términos se obtiene:

$$\begin{aligned} X'(z) = \frac{1}{2}&[H_0(z)G_0(z) + H_1(z)G_1(z)]X(z) \\ &+ \frac{1}{2}[H_0(-z)G_0(z) + H_1(-z)G_1(z)]X(-z) \end{aligned} \tag{2.21}$$

Para crear una reconstrucción perfecta de la señal de entrada $x[n]$ a partir del procesamiento de este par de filtros espejo, se debe diseñar los filtros tal que:

$$G_1(z) = -H_0(z) \rightarrow g_1[n] = (-1)^n h_1[n] \tag{2.22}$$

$$G_0(z) = H_1(-z) \rightarrow g_0[n] = -(-1)^n h_1[n] \tag{2.23}$$

La Transformada *Z* de la señal de salida será:

$$X'(z) = \frac{1}{2}\left(H_0^2(z) - H_0^2(-z)\right)X(z) \tag{2.24}$$

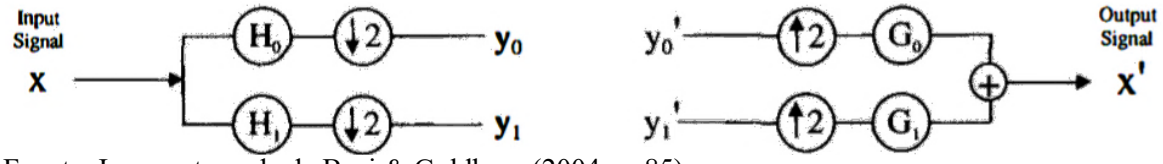
Luego para conseguir la reconstrucción perfecta $X'(z) = X(z)$ ilustrada en las Ecuaciones (2.25) y (2.26) retoman la idea de una formulación de un FIR con un único retraso de orden l. Lo cual implica una ecuación en diferencias de la forma $x'[n] = x[n-l]$, donde se indica que $x'[n]$ es una versión exacta de $x[n]$ solo que retrasada l muestras. Por lo que, con la condición impuesta por la Ecuación (2.25), se adquiere una reconstrucción perfecta -Ecuación (2.26)-. En la Figura 2–9 se muestra que se debe construir el filtro $h_0[n]$ tal que:

$$H_0^2(z) - H_0^2(-z) = 2z^{-l} \tag{2.25}$$

$$X'(z) = X(z)z^{-l} \tag{2.26}$$

$$x'[n] = x[n-l] \tag{2.27}$$

Figura 2–9: Reconstrucción perfecta de banco de filtros de dos canales. Decimando e interpolando por un factor de 2.



Fuente: Imagen tomada de Bosi & Goldberg (2004, p. 85).

2.3.3 Banco de filtros PQMF

El oído humano se divide naturalmente entre 20 y 30 bandas críticas de frecuencia, por lo que se ha de implementar un banco de filtros con un número cercano de bandas a las bandas críticas del oído. Se ha podido desarrollar una generalización aproximada multibanda de la técnica del par de filtros QMF en paralelo a partir de los trabajos de Nussbaumer, 1981) y Rothweiler (1983).

La idea que se plantea al implementar el banco de filtros PQMF (también conocidos como *filtros polifásicos*) es tomar un filtro Pasa Bajo base, cuya banda de paso sea lo suficientemente estrecha y modular copias de este dentro de todo el espectro audible. Este filtro al ser modulado debe proveer en su banda de transición un decaimiento lo suficientemente rápido para que haya un traslapo despreciable con los filtros aledaños y así sea pueda cancelar los efectos de *aliasing*, así como se revisó en la sección anterior.

El banco de filtros PQMF consiste en k canales o sub-bandas, de las cuales cada canal es un filtro Pasa Bajo $h[n]$ modulado por un coseno. El esquema de codificación dado en el MP3 consiste en repartir la señal de audio $x(t)$ en 32 bandas de frecuencia igualmente espaciadas, usando el filtro base $h[n]$ modulado $h_k[n]$ a 32 bandas, como es visto en la Ecuación (2.28). En la matriz vista en esta ecuación todavía no se aplica decimación para la señal de entrada x, si no que se generan representaciones filtradas completas de x, en cada sub-banda k del tamaño original de x. A este fenómeno se le conoce como banco de filtro con *sobre-muestreo*.

$$s_k[n] = h_k[n] * x[n]$$

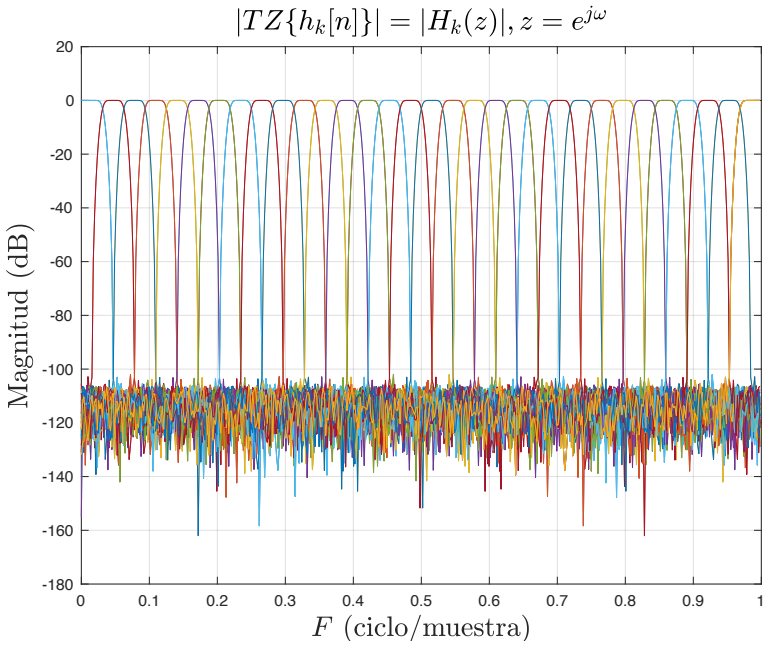
$$k = 0, 1, \dots, K-1, \qquad n = 0,1 \dots, N-1, \qquad K = 32, \qquad N = 512$$

$$s[k,n] = \begin{bmatrix} s_0[0] & s_0[1] & \cdots & s_0[n] & \cdots & s_0[N-1] \\ s_1[0] & s_1[1] & \cdots & s_1[n] & \cdots & s_1[N-1] \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ s_k[0] & s_k[1] & \cdots & s_k[n] & \cdots & s_k[N-1] \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ s_{K-1}[0] & s_{K-1}[1] & \cdots & s_{K-1}[n] & \cdots & s_{K-1}[N-1] \end{bmatrix} \tag{2.28}$$

$$h_k[n] = h_0[n] \cos\left[(2k+1)(n-16)\frac{\pi}{64}\right], \qquad k = 0,1, \dots, 31, \qquad n = 0,1, \dots, 511. \tag{2.29}$$

$h_0[n]$ al ser modulado por $\cos\left[(2k+1)(n-16)\frac{\pi}{64}\right]$ traslada el filtro base pasa-bajos a una banda de frecuencia alrededor de la frecuencia $(2k+1)\pi/64$, permitiendo que $h_k[n]$ se convierta en una sucesión de filtros pasa banda alrededor de cada frecuencia $(2k+1)\pi/64$, con un ancho de banda igual a $\pi/32$ (Figura 2–10). Estos 32 canales modulados, representan una aproximación de la cantidad total de bandas críticas del oído humano, para lo cual cada ancho de banda es aproximadamente de $20\text{k Hz}/32 = 625$ Hz (o en frecuencia digital normalizada $\pi/32$, $f_s/64$, o igualmente $f_N/32 = 689$ Hz).

Figura 2–10: Respuesta en frecuencia de los filtros k-ésimos ($k = 0, 1, \dots, 31$) de análisis (prototipo)

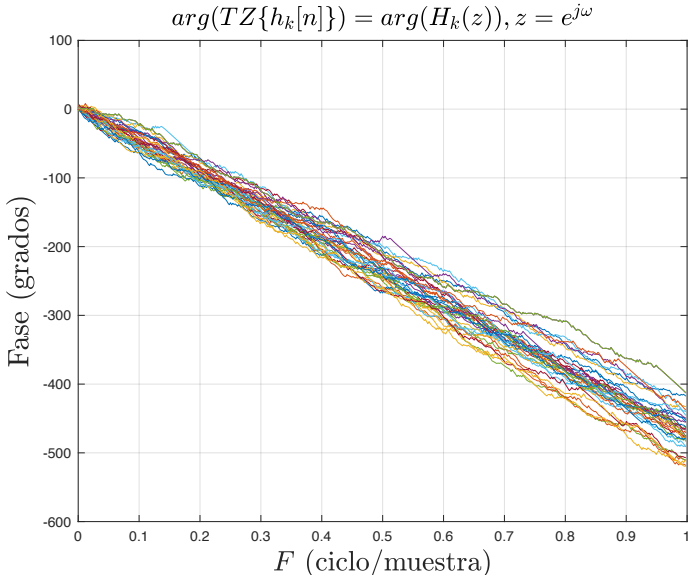


Fuente: Imagen de elaboración propia usando Matlab_R2018a.

Se elige trabajar con el filtro base p_0 con ventana de longitud de 512 muestras, con el fin de generar un traslapo entre bandas de 512/32=16 muestras, lo cual genera un desplazamiento en el espectro digital (hop-size) de 32 muestras. Se pretende mantener la linealidad en la

fase de cada uno de estos filtros, dado al comportamiento psicoacústico frente a la conducta funcional de las fases.

Figura 2–11: Fase (desenvuelta) de los filtros k-ésimos ($k = 0, 1, \dots, 31$) de análisis (prototipo).



Fuente: Imagen de elaboración propia usando Matlab_R2018a.

Cada una de las convoluciones $s_k[n]$, de la señal $x[n]$, con cada uno de los k filtros $h_k[n]$, va a generar una versión (temporal) de $x[n]$ filtrada en la sub-banda k, de esto se observa un incremento en la tasa de muestreo respecto a $s_k[n]$, por una cantidad $k = 32$, al procesar k veces $x[n]$. A causa de este incremento de la tasa de muestreo, se debe decimar, y para este caso, la implementación de la decimación (*down sampling*) se halla implícito dentro de la formulación matemática de la convolución (precisamente en el argumento de x, en la forma "$32 \times n - m$", para ir "saltando" de a treinta y dos muestras por cada uno de los dieciséis n, con esto se genera un arreglo de 32×16 muestras en $s_k[n]$ por cada una de la n sumatorias.):

$$s[k,n] = h[k,n] * x[n]$$
$$s_k[n] = \sum_{m=0}^{511} h_k[m]x[32n - m] \qquad (2.30)$$
$$k, n, m \in \mathbb{Z}, \quad k = 0, 1, \dots, 32, \quad n = 0, 1, \dots, 15, \quad \text{y} \quad m = 0, 1, \dots, 511$$

k: índice de frecuencia digital.
n: índice explícito de decimación para el tiempo digital de la señal de entrada x.
m: índice de tiempo propio del banco de filtros h_k.

Es importante recordar que la definición de la Ecuación (2.30) se implementa como se vio en la Sección 2.2.1 para una convolución de tipo lineal.

Si $x[n]$ es la señal de audio de entrada con un tamaño de 512 muestras, dentro de $s_k[n]$, cada uno de los valores de $x[n]$ se recorre como es visto en la Tabla 2-1.

Variando tanto a n como a m:

Tabla 2-1: Visualización de la implementación directa de la Ecuación (2.30).

$\boldsymbol{n}$	$\boldsymbol{m=0}$	$\boldsymbol{m=1}$	$\boldsymbol{m=2}$	$\boldsymbol{m=511}$
0	$\boldsymbol{h_k[0]x[0]}$	$\boldsymbol{h_k[1]x[-1]}$	$\boldsymbol{h_k[2]x[-2]}$	$\boldsymbol{h_k[511]x[-511]}$
1	$h_k[0]x[32]$	$h_k[1]x[31]$	$h_k[2]x[30]$	$h_k[511]x[-479]$
2	$h_k[0]x[64]$	$h_k[1]x[62]$	$h_k[2]x[61]$	$h_k[511]x[-447]$
3	$h_k[0]x[96]$	$h_k[1]x[95]$	$h_k[2]x[94]$	$h_k[511]x[-415]$
15	$\boldsymbol{h_k[0]x[480]}$	$\boldsymbol{h_k[1]x[479]}$	$\boldsymbol{h_k[2]x[478]}$	$\boldsymbol{h_k[511]x[-31]}$

De la implementación directa de esta convolución, se muestran $512 \times 32 = 16384$ multiplicaciones y sumas, tan solo para entregar 32 salidas de $s_k[n]$ para cada bloque de 32 muestras de la señal de entrada $x[n]$.

Una implementación optimizada, descrita por Bosi y Goldberg (2003) y establecida en la normativa ISO/IEC. 11172-3 (Smith & Abel, 1993), cambian $s_k[n]$ de la Ecuación (2.30) por la Ecuación (2.31).

$$s_k[n] = \sum_{r=0}^{63} \sum_{q=0}^{7} h_k[64q + r] \cdot x[32n - 64q - r] \tag{2.31}$$

Definiendo $m = 64q + r$ y sustituyendo en (2.30) se obtiene (2.31), con esto, se logra utilizar la propiedad par del coseno en la Ecuación (2.32) para iterar sobre valores negativos y positivos:

$$\cos(\pi l) = (-1)^l, \qquad l \in \mathbb{Z}^+ \tag{2.32}$$

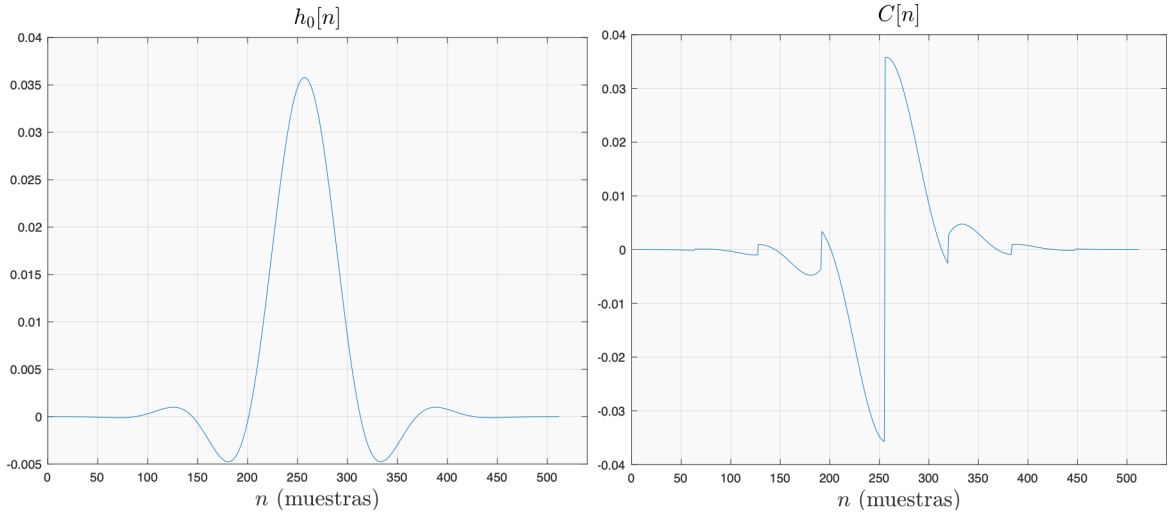
$$\cos\left[(2k+1)(64q + r - 16)\frac{\pi}{64}\right] = \cos\left[(2k+1)(r-16)\frac{\pi}{64} + q\pi\right] = \\ = \begin{cases} \cos\left[(2k+1)(r-16)\frac{\pi}{64}\right] \text{ si } \lfloor n/64 \rfloor \text{ es par} \\ -\cos\left[(2k+1)(r-16)\frac{\pi}{64}\right] \text{ si } \lfloor n/64 \rfloor \text{ es impar} \end{cases} \tag{2.33}$$

$$C[n] = (-1)^{\text{int}\lfloor n/64 \rfloor} h_0[n] \rightarrow C[n] = \begin{cases} -h_0[n] \text{ si } \lfloor n/64 \rfloor \text{ es par} \\ h_0[n] \text{ si } \lfloor n/64 \rfloor \text{ es impar} \end{cases} \tag{2.34}$$

Luego, es posible ver la reducción que permiten las Ecuaciones (2.33) y (2.34) en la Ecuación (2.31) para obtener la Ecuación (2.35).

$$h_k[64q + r] = C[64q + r] \cdot \cos\left[(2k+1)(r-16)\frac{\pi}{64}\right] \tag{2.35}$$

Figura 2–12: Filtro base de análisis $h_0[n]$ y transformación $C[n]$.



Fuente: Imagen de elaboración propia usando Matlab_R2018a.

Obsérvese la distinción entre $C[n]$ y el filtro base $h_0[n]$ al implementar la formulación de la Ecuación (2.34). La definición de la Ecuación (2.35) permite nombrar a $h_k[n]$ como la respuesta al impulso "prototipo", y al filtro $C[n]$ como la versión manipulada de $h_k[n]$ en la Figura 2–12. Usando la notación del estándar para obtener:

$$M[k,r] = \cos\left[(2k+1)(r-16)\frac{\pi}{64}\right], \qquad k = 0,1,\dots,31, \qquad r = 0,1,\dots,64. \tag{2.36}$$

La Ecuación (2.35) se sintetiza en:

$$h_k[64q+r] = C[64q+r] \cdot M[k,r] \tag{2.37}$$

y la Ecuación (2.31) para $s_k[n]$:

$$s_k[n] = \sum_{r=0}^{63}\sum_{q=0}^{7} C[64q+r] \cdot M[k,r] \cdot x[32n-64q-r] \tag{2.38}$$

Reorganizando la Ecuación (2.38):

$$s_k[n] = \sum_{r=0}^{63} M[k,r] \sum_{q=0}^{7} C[64q+r] \cdot x[32n-64q-r] \tag{2.39}$$

La Ecuación (2.39) resulta en una formulación que contiene 80 multiplicaciones y adiciones por muestra.

Finalmente es posible organizar a $s_k[n]$ como un arreglo o una matriz de tamaño $k \times n$, donde cada fila k es una de las 32 sub-bandas, y cada columna n es una de las 16 muestras que componen a la señal $x[n]$ filtrada y decimada en la sub-banda k.

$$s_k[n] = \{s_0[0] \dots s_0[15], s_1[0] \dots s_1[15], \dots, s_{31}[0] \dots s_{31}[15]\}$$

$$s_k[n] = \begin{bmatrix} s_0[0] & s_0[1] & \cdots & s_0[n] & \cdots & s_0[15] \\ s_1[0] & s_1[1] & \cdots & s_1[n] & \cdots & s_1[15] \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ s_k[0] & s_k[1] & \cdots & s_k[n] & \cdots & s_k[15] \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ s_{31}[0] & s_{31}[1] & \cdots & s_{31}[n] & \cdots & s_{31}[15] \end{bmatrix}_{32\times16} = s[k,n] \tag{2.40}$$

Nótese que si la decimación no se hubiese planteado tanto en la Ecuación (2.30) como en la Ecuación (2.40), se hubiese obtenido un sobre muestreo para $s_k[n]$, de 32×512 muestras por cada *buffer* de entrada de 512 muestras, viéndose también como una matriz de 32×512, de la forma vista en la Ecuación (2.41). Donde cada s_k de 512 muestras sería una versión "completa" de $x[n]$ filtrada en la sub-banda k.

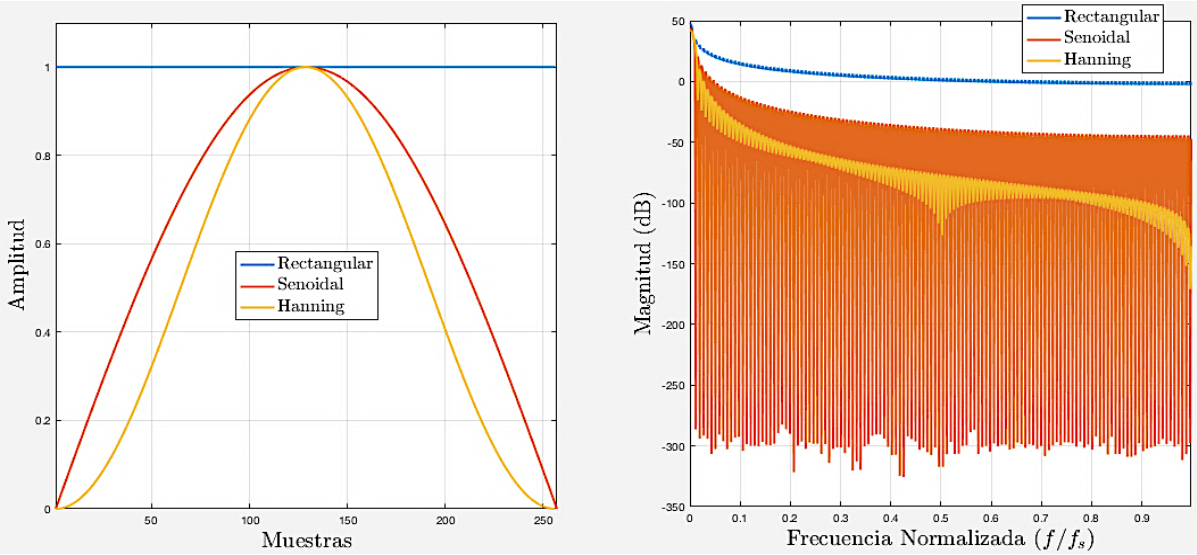
$$s_k[n] = \{s_0[0] \dots s_0[511], s_1[0] \dots s_1[511], \dots, s_{31}[0] \dots s_{31}[511]\}$$

$$s_k[n] = \begin{bmatrix} s_0[0] & s_0[1] & \cdots & s_0[n] & \cdots & s_0[511] \\ s_1[0] & s_1[1] & \cdots & s_1[n] & \cdots & s_1[511] \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ s_k[0] & s_k[1] & \cdots & s_k[n] & \cdots & s_k[511] \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ s_{31}[0] & s_{31}[1] & \cdots & s_{31}[n] & \cdots & s_{31}[511] \end{bmatrix}_{32\times512} = s[k,n] \tag{2.41}$$

2.3.4 Revisión de la ventana senoidal

Parte de la exigencia de procesar señales en tiempo real, es trabajar con bloques que información finita, para la cual, ésta debe tener un ancho de banda limitado (como al discretizarse/digitalizarse una señal análoga). Si se tiene como entrada, bloques concatenados de información, y se desea separarlos en bloques de igual longitud para ser procesadas, la primera idea es utilizar una ventana rectangular $w_R[n]$ limitada en su dominio desde la muestra $n = 0$ hasta la $n = N$. Siendo una señal de extensión indeterminada $x[n]$, al ser multiplicada por una señal ventana finita $w_R[n]$ ℓ cantidad de veces de forma sucesiva, se obtendrán $x_\ell[n]$ parciales con los cuales se pretende trabajar individualmente. Bosi & Goldberg (2004) plantean la cuestión si al limitar temporalmente una señal también ésta está limitada espectralmente de forma adecuada para trabajar.

Figura 2–13: Gráfica izquierda, Comparación en el dominio del tiempo digital, Gráfica derecha, comparación en el dominio del de la frecuencia, entre ventanas Rectangular, Senoidal y Hanning. Calculado a partir de 248 muestras.



Fuente: Imagen de elaboración propia usando Matlab_R2018a.

$$w_R[n] = \begin{cases} 1, 0 \leq n \leq N-1 \\ 0, \quad \text{otros} \end{cases} \tag{2.42}$$

$$w_S[n] = \sin\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right)\right] \tag{2.43}$$

$$w_H[n] = \frac{1}{2}\left(1 - \cos\left[\frac{2\pi}{N}\left(n + \frac{1}{2}\right)\right]\right) \tag{2.44}$$

La STFT (Transformada de Fourier de Tiempo Corto) de la señal limitada en tiempo $x_\ell[n]$ es una convolución lineal en el dominio del tiempo discreto con la señal ventana $w_x[n]$.

La DFT mostrada en la Figura 2–13 revela decaimiento lento de la pendiente en el dominio de la frecuencia para $w_R[n]$ (Ecuación (2.42)), decae a razón de $1/|f|$. Con un decaimiento tan lento como el de esta ventana rectangular, no se ve una limitación tipo filtro, para el ancho de banda para $x_\ell[n]$, no siendo una buena opción para evitar efectos de aliasing, además, las discontinuidades que hay en los extremos de la ventana rectangular, suponen la aparición de frecuencias inexistentes según el fenómeno de Gibbs, esto anterior posibilita distorsiones armónicas críticas.

Mejores elecciones de ventanas con las cuales se evitan discontinuidades en sus bordes son por ejemplo la ventana Senoidal (Ecuación (2.43)) o la ventana Hanning (Ecuación (2.44)). Este par de funciones en el dominio de la frecuencia decaen mucho más rápido que la ventana Rectangular, sirviendo como filtros que limitan el ancho de banda de cada sucesión $x_\ell[n]$, es decir que permiten una mejor resolución espectral y posibles efectos de aliasing. La Figura 2–13, revela nuevamente que los lóbulos en el dominio de la frecuencia de la

ventana Hanning decaen mucho más rápido que los de la ventana Senoidal, proveyendo una mejor resolución espectral, pero la causa de esto es una peor definición en el dominio del tiempo, en términos de la amplitud que se le asigna a cada término espectral. A lo anterior se le conoce como compromiso tiempo-frecuencia. Se verá más adelante (Sección 2.4.1) que en la normativa ISO-IEC 11172-3, se trabaja con una ventana senoidal en el análisis y síntesis espectral usando la Transformada Modificada de Cosenos muy similar a la descrita en la Ecuación (2.43), para el modelo psicoacústico, se trabaja con una ventana tipo Hanning.

2.4 Transformada Modificada de Cosenos (MDCT)

2.4.1 Generalidades para la implementación en el MPEG1-Layer-3

Thiagarajan & Spanias (2011) plantean que, al localizar espectralmente los umbrales de enmascaramiento, se requiere que las señales con contenido armónico estable y bien definido, durante el momento del análisis, se observe una alta resolución en frecuencia (mayor a 32 sub-bandas discretas).

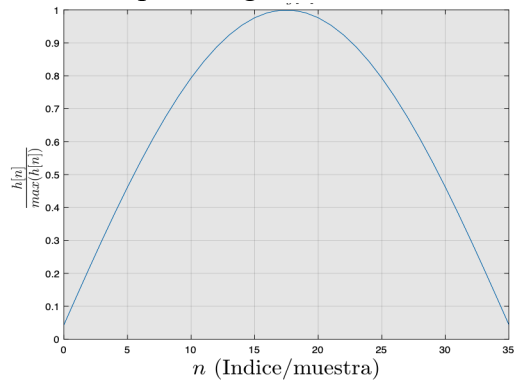
El estudio analítico del efecto del ventaneo sobre señales, indica que los cambios repentinos en el dominio del tiempo (señales transitorias rápidas) requieren una resolución en tiempo adecuada que pueda acotar y contener esa información (una ventana "corta"); información que permita estimar con precisión los umbrales de enmascaramiento temporal en cada acontecimiento temporal.

La problemática con las señales de audio es que poseen una naturaleza que no suele hallarse la mayor cantidad de tiempo en estados estacionarios de oscilación, más bien, está compuesto por intervalos breves de tiempo de estados estacionarios y transitorios. Por lo tanto, un codificador de audio apropiado debe ser adaptativo a estos cambios de estado, y debe ejecutar un algoritmo que reconozca éstos, respecto a una óptima descomposición tiempo-frecuencia. Lo anterior indica que el banco de filtros que prosigue debe tener resoluciones variantes en los dominios de tiempo y de frecuencia. Se pretende un modelo híbrido de filtros, que cambie de acuerdo con las características alternantes de las señales.

El modelo "híbrido" de filtros se concilia al procesar la salida de 36 muestras temporales para cada una de las 32 sub-bandas espectrales del PQMF usando la Transformada Modificada de Cosenos, con lo cual, al subdividir (transformar) cada una de las 36 muestras por pada sub-banda con un bloque de MDCT de 18 frecuencias, aumenta la resolución espectral en $32 \times 18 = 576$ frecuencias, donde el audio decimado en cada una de las 32 sub-bandas es transformada en 18 frecuencias en el dominio real. Al producto de 36 muestras del gránulo ℓ, de la ventana $w_\ell[n]$ (usada tanto en el análisis como en la síntesis) con la señal decimada en la sub-banda k, $s_\ell[k,n]$, la denominamos $\psi_\ell[k,n]$ en la Ecuación (2.45).

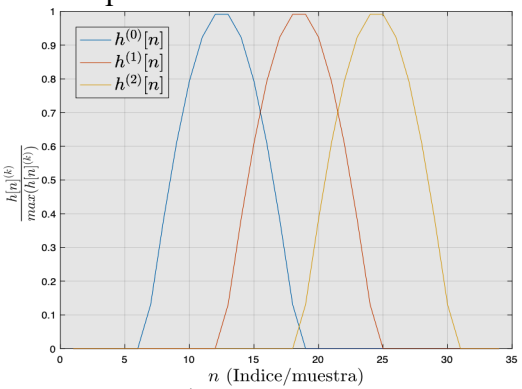
$$\psi_\ell[k,n] = \psi_{k,\ell}[n] = s_\ell[k,n] \cdot w_\ell[n], \quad k = 0,1,\dots,31, \quad n = 0,1,\dots,35, \quad \ell = 0 \dots \infty \tag{2.45}$$

Figura 2–14: Ventana tipo “Long” o “Normal” usada en la MDCT.



$$w_{long}[n] = \sin\left[\frac{\pi}{36}\left(n+\frac{1}{2}\right)\right], n = 0, 1, \dots, 35 \qquad (2.46)$$

Figura 2–15: Ventana tipo “Short” usada en la MDCT.

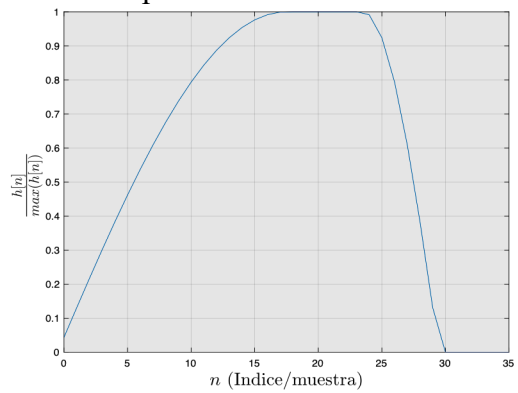


$$w^{(k)}[i] = s^{(k)}[i]\sin\left[\frac{\pi}{12}\left(i+\frac{1}{2}\right)\right], i = 0, 1, \dots, 11. \quad k = 0,1,2$$

$$s^{(k)}[i] = \begin{cases} s[i+6], & k = 0.\ i = 0, \dots, 11 \\ s[i+12], & k = 1.\ i = 0, \dots, 11 \\ s[i+18], & k = 2.\ i = 0, \dots, 11 \end{cases} \qquad (2.47)$$

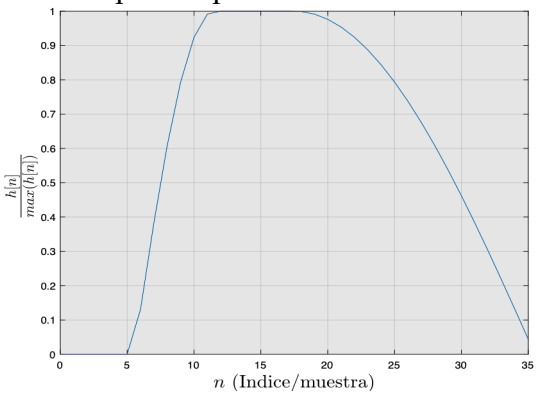
$$w_{short}[n] = \{w^{(0)}[i], w^{(1)}[i], w^{(2)}[i]\}, \quad n = 0, 1, \dots, 35, \quad i = 0, 1, \dots, 11$$

Figura 2–16: Ventana tipo “Start” usada en la MDCT.



$$w_{start}[n] = \begin{cases} \sin\left[\frac{\pi}{36}\left(n+\frac{1}{2}\right)\right], & n = 0, 1, \dots, 17 \\ 1, & n = 18, \dots, 23 \\ \sin\left[\frac{\pi}{12}\left(n-18+\frac{1}{2}\right)\right], & n = 24, \dots, 29 \\ 0, & n = 30, \dots, 35 \end{cases} \qquad (2.48)$$

Figura 2–17: Ventana tipo “Stop” usada en la MDCT.



$$w_{stop}[n] = \begin{cases} 0, & n = 0, 1, \dots, 5 \\ \sin\left[\frac{\pi}{12}\left(n-6+\frac{1}{2}\right)\right], & n = 1, \dots, 11 \\ 1, & n = 12, \dots, 17 \\ \sin\left[\frac{\pi}{36}\left(n+\frac{1}{2}\right)\right] & n = 18, \dots, 35 \end{cases} \qquad (2.49)$$

$$N = 36, \qquad n = 0, \dots, N-1., \qquad \xi = 0, \dots, \frac{N}{2} - 1.$$
$$n = 0,1, \dots ,35, \qquad \xi = 0,1, \dots ,17.$$
$$\Psi_{k,\ell}[\xi] = \sum_{n=0}^{N-1} \psi_{k,\ell}[n] \cos\left[\frac{2\pi}{N}\left(n + \frac{N}{4} + \frac{1}{2}\right)\left(\xi + \frac{1}{2}\right)\right] \tag{2.50}$$

$$\bar{\bar{\gamma}}_{k,\ell}[n] = \frac{4}{N} \sum_{\xi=0}^{\frac{N}{2}-1} \Psi_{k,\ell}[\xi] \cos\left[\frac{2\pi}{N}\left(n + \frac{N}{4} + \frac{1}{2}\right)\left(\xi + \frac{1}{2}\right)\right], \tag{2.51}$$
$$\psi_{k,\ell}[n] \neq \bar{\bar{\gamma}}_{k,\ell}[n]$$

Así como es descrito en la Ecuación (2.45) para hacer uso adecuado de la MDCT se debe ventanear la entrada $s_{k,\ell}[n]$ con $w_\ell[n]$, igualmente, la salida de la IMDCT $\bar{\bar{\gamma}}_{k,\ell}[n]$ debe ser ventaneada por la misma ventana $w_\ell[n]$ utilizada para $s_{k,\ell}[n]$ en el bloque ℓ.

$$\gamma_{k,\ell}[n] = \bar{\bar{\gamma}}_{k,\ell}[n] \cdot w_\ell[n], \qquad \psi_{k,\ell}[n] \neq \gamma_{k,\ell}[n] \tag{2.52}$$

En la normativa ISO-IEC 11172-3 (pg. 37) se menciona “la primera mitad de un bloque de 36 muestras se traslapa con la segunda mitad del bloque previo. La segunda mitad del bloque actual es almacenado para ser usando en el siguiente y repetir el sobrelapamiento”. Esta descripción es vital para el procesamiento de transformaciones llevadas a cabo por la MDCT. Se transforman (tanto en la MDCT como en la IMDCT) sucesiones bloques de muestras que contienen trozos de bloques anteriores.

La descripción detallada de la *reconstrucción perfecta* dada con $\gamma_{k,\ell}[n]$ a partir del número de muestras de entrada para los PQMF se da en el capítulo del Desarrollo Ingenieril.

2.4.2 Condiciones de las ventanas para la reconstrucción perfecta con la MDCT

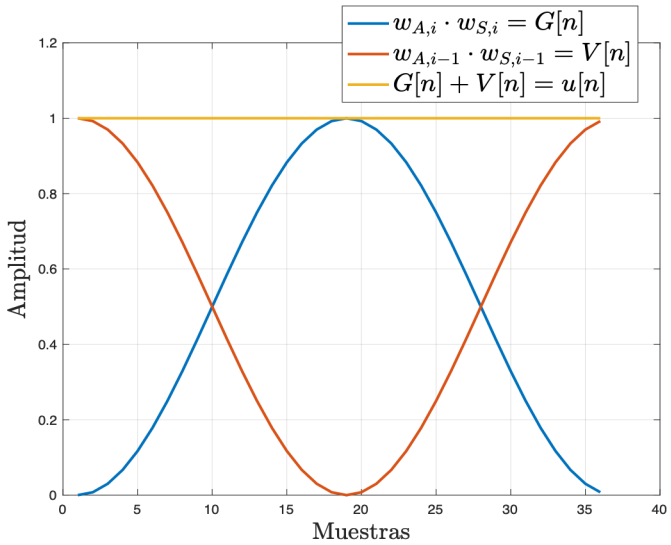
Como fue explicado en secciones anteriores, en la etapa previa al análisis (previo a realizar la transformada directa) se ventanea el bloque (gránulo) i-ésimo entrante al sistema, la ventana utilizada aquí es la ventana de análisis $w_{a,i}[n]$. Posterior a la etapa de la transformación inversa, se ventanea en el mismo bloque i-ésimo con la ventana de síntesis $w_{s,i}[n]$. Como indican Bosi & Goldberg (2004) la operación intuitiva e implícitamente existente en la codificación entre estas dos ventanas, es su producto, que en caso de ser ambas la misma ventana, se llega simplemente a $w_i^2[n]$. A la hora de hacer el Overlap & Add para la síntesis después de la aplicación de la transformada inversa, las ventanas de cada banda del bloque previo $w_{A,S,i-1}$ se suman con las de actual $w_{A,S,i}$, como es visto en la Ecuación (2.53). Las ventanas de cada gránulo se diferencian una respecto a la otra por un corrimiento en el dominio del tiempo de la media de su longitud $N/2$.

$$w_{A,i}[n] \cdot w_{S,i}[n] + w_{A,i-1}[n + N/2] \cdot w_{S,i-1}[n + N/2] = u[n] = \mathbf{1} \tag{2.53}$$

Donde $u[n]$ es el escalón unitario. $u[n] = \begin{cases} 1, n \geq 0 \\ 0, n < 0 \end{cases}$, $n = 0,1, \dots, N-1$.

La expresión de la Ecuación (2.53), muestra que el diseño de estas ventanas de análisis y síntesis deben generar valores unitarios al superponerse en el momento de la reconstrucción, esto es con el propósito de que las señales que acompañan a las ventanas no sufran deformaciones al sobrelaparse. Existirían distorsiones en la reconstrucción si la Ecuación (2.53) resultase una función distinta a la del escalón unitario. Los 4 tipos de ventana mostrados en la sección 2.4.1 están diseñadas para que en sus regiones de cruce se llegue al escalón unitario. Un ejemplo de esto se da con la ventana tipo "Long", de la Ecuación (2.46) en la Figura 2–18.

Figura 2–18: Relación general de diseño de las ventanas de análisis y síntesis para el TDAC.



Fuente: Imagen de elaboración propia usando Matlab_R2018a.

2.4.3 Elección de la MDCT sobre la transformada STFT.

Como expresan Kuo et al. (2006) de la implementación de una STFT es posible lograr una buena representación en el dominio de la frecuencia para señales de audio, al tomar bloques de longitud finita de tiempo y transformarlas. Estas muestras espectrales pueden ser cuantizadas usando criterios psicoacústicos, y después de este procedimiento, decuantizar estas muestras en frecuencia usando la transformada inversa, además de aplicar Overlap & Add entre bloques procesados para eliminar los efectos de las ventanas usadas en la etapa de la transformada directa. Esto se debe a que se generan "artefactos" o distorsiones generadas por superponer bloques en el dominio del tiempo, ya que como se vio en la sección 2.2.2, se pueden superponer señales que individualmente ya contienen toda la información espectral originalmente dada, pero que al superponerlas, se generan componentes nuevas (aliasing).

Como explican Bosi & Goldberg (2004, p. 124), "el problema principal con una implementación de este esquema de codificación usando la STFT, es que el procedimiento

de Overlap & Add aumenta la tasa de datos en el dominio de la frecuencia sin antes aportar alguna ventaja en la codificación". Siendo lo anterior una certeza con esta transformada tradicional, lo deseable sería que, a la entrada y salida de las etapas de los sistemas de análisis y síntesis, la tasa de datos general se mantuviese constante, es decir que sin tener en cuenta la cuantización por algún modelo psicoacústico, la cantidad de información espectral en el momento de la entrada de la etapa de análisis debería ser la misma que en la etapa de salida de síntesis.

La MDCT fue diseñada pensando en mantener constante la tasa de datos espectral al hacer síntesis con su transformada inversa. Estos efectos se reducen sustancialmente ventaneando previa y posteriormente de la transformada directa e inversa, como se mencionó en los párrafos anteriores, estos efectos indeseados son inevitables usando la STFT, pero las ventanas para la MDCT están diseñadas para un sobrelapamiento del 50%, es decir, el *hop-size* es de $M = N/2$, donde N es la longitud de cada bloque procesado. Lo cual significa que se debe procesar y almacenar N muestras de datos transformados para cada M nuevas muestras que entran al sistema. Esto anterior indica que previo a cualquier ganancia de codificación por eliminación de redundancia e irrelevancia de datos, se está aumentando la tasa de datos por un factor de N/M, que, en este caso, es el doble.

Una de las ganancias de utilizar un 50% de sobrelapamiento con la MDCT es que las zonas sobrelapadas de los bloques (es decir, no hay zonas donde no exista superposición) no sufren un incremento en la tasa de datos, como sí sucede al implementar la STFT. Y es en esta superposición en la que Princen, Johnson & Bradley (1987) definen a la MDCT como una transformada de Cancelación de Aliasing en el Dominio Temporal (TDAC). Como fue mencionado en la sección anterior, la MDCT no es invertible directamente como la DFT para recuperar una señal en el dominio del tiempo. Dado que este procesamiento con la MDCT se da a sucesiones de bloques donde cada uno contiene el 50% de los datos del bloque anterior, esta mezcla de redundancia es inicialmente lo que se conoce como *Aliasing Temporal* (semejante a lo visto en la sección 2.2.2 y 2.2.3), donde se explicó por separado que se superponían aritméticamente bloques y se guardaban los datos de bloques anteriores. El códec del MPEG-1 Layer III muestra en su algoritmia que el en Overlap & Add para los bloques transformados con IMDCT se da la cancelación de este artefacto de aliasing temporal.

En palabras de Bosi & Goldberg (2004) "Para señales reales, solo $N/2$ de las muestras espectrales para una transformación de N muestras con un TDAC son independientes, lo cual indica que para señales de audio, solo se requieren $N/2$ muestras de frecuencia para cada bloque para su reconstrucción total".

2.4.4 Comparación entre las Transformadas MDCT y la DFT

En esta subsección como en la sección siguiente (Derivación de las condiciones del TDAC para la Reconstrucción Perfecta con la Transformada MDCT) se pretende aportar con ciertos desarrollos matemáticos y verificaciones, las proposiciones sin aparente explicación y demostración halladas en los textos citados a lo largo de este trabajo de grado.

La DFT se expresa como en la Ecuación (2.54) y la expresión básica según Kuo et al. (2006) de la MDCT en (2.58, donde $\phi[k,n]$ es cualquier función que sirve como argumento de la *regla de transformación* (Revisar Anexo 1).

$$\begin{aligned} \tilde{X}[k] &= \sum_{n=0}^{N-1} x[n] e^{-j\phi[k,n]} = \text{DFT}\{x[n]\}, \qquad \tilde{X}[k] \in \mathbb{C} \\ \phi[k,n] &= \frac{2\pi}{N} kn, \qquad k = 0,1,\dots,N-1, \qquad e^{-j\phi[k,n]} = W_N^{\phi[k,n]} \end{aligned} \tag{2.54}$$

Donde su transformada inversa IDFT es:

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} \tilde{X}[k] \boldsymbol{W}_N^{-\phi[k,n]} = \text{IDFT}\{\tilde{X}[k]\} \tag{2.55}$$

La idea es demostrar que la parte real de la DFT, $\text{Re}\{\tilde{X}[k]\}$ es la MDCT:

$$\text{MDCT}\{x[n]\} = \text{Re}\big\{\text{DFT}\{x[n]\}\big\} \tag{2.56}$$

y a su vez, que las transformadas inversas no son iguales $\text{IMDCT}\{X[k]\} \neq \text{IDFT}\{\tilde{X}[k]\}$, y por lo tanto, la representación directa de la transformada inversa IMDCT debe someterse a un procesamiento posterior, para obtener como resultante una señal muy próxima a la señal original:

$$\text{IMDCT}\{X[k]\} = x'[n] \neq x[n] = \text{IDFT}\{\tilde{X}[k]\} \tag{2.57}$$

La transformada MDCT es:

$$\begin{aligned} X[k] &= \sum_{n=0}^{N-1} x[n] \cos(\phi[k,n]) = \text{MDCT}\{x[n]\}, \qquad X[k] \in \mathbb{R} \\ \phi[k,n] &= \frac{2\pi}{N}\left(n + \frac{N}{4} + \frac{1}{2}\right)\left(k + \frac{1}{2}\right), \qquad k = 0,1,\dots,\frac{N}{2} - 1 \end{aligned} \tag{2.58}$$

Y su transformada inversa IMDCT:

$$x'[n] = \frac{4}{N} \sum_{k=0}^{N/2-1} X[k] \cos(\phi[k,n]) = \text{IMDCT}\{X[k]\} \tag{2.59}$$

Para la demostración, se propone una señal periódica discreta $x[n]$ descrita como un tono puro de la forma $A_0 \cos\left(\frac{2\pi}{N} k_0 n\right)$ con frecuencia discreta k_0. La señal de entrada puede expresarse en términos de un par de exponenciales complejas como:

$$x[n] = \cos\left(\frac{2\pi}{N}k_0 n\right) = \frac{1}{2}\left(e^{-j\frac{2\pi}{N}k_0 n} + e^{j\frac{2\pi}{N}k_0 n}\right)$$

Al ingresar $x[n]$ en la expresión (2.54) de la DFT:

$$\tilde{X}[k] = A_0 \sum_n \cos\left(\frac{2\pi}{N}k_0 n\right) e^{-j\phi[k,n]} = \frac{A_0}{2}\sum_n \left(e^{-j\frac{2\pi}{N}k_0 n} + e^{j\frac{2\pi}{N}k_0 n}\right) e^{-j\phi[k,n]}$$
$$\tilde{X}[k] = \frac{A_0}{2}\sum_n \left[e^{-j\left(\frac{2\pi}{N}k_0 n + \phi[k,n]\right)} + e^{j\left(\frac{2\pi}{N}k_0 n - \phi[k,n]\right)}\right] \quad (2.60)$$

La parte real de la DFT -Ecuación (2.60)- resulta en:

$$X[k] = \mathrm{Re}\{\mathrm{DFT}\{x[n]\}\}$$
$$= \frac{A_0}{2}\sum_n \left[\cos\left(\frac{2\pi}{N}k_0 n + \phi[k,n]\right) + \cos\left(\frac{2\pi}{N}k_0 n - \phi[k,n]\right)\right] \quad (2.61)$$

Usando la misma señal de entrada $x[n]$ en la definición (2.58), de la MDCT:

$$A_0 \sum_n \cos\left(\frac{2\pi}{N}k_0 n\right)\cos(\phi[k,n]) = \frac{A_0}{4}\sum_n \left(e^{-j\frac{2\pi}{N}k_0 n} + e^{j\frac{2\pi}{N}k_0 n}\right)\left(e^{-j\phi[k,n]} + e^{j\phi[k,n]}\right)$$

$$= \frac{A_0}{4}\sum_n \left(e^{-j\left(\frac{2\pi}{N}k_0 n + \phi[k,n]\right)} + e^{j\left(\frac{2\pi}{N}k_0 n + \phi[k,n]\right)} + e^{-j\left(\frac{2\pi}{N}k_0 n - \phi[k,n]\right)} + e^{j\left(\frac{2\pi}{N}k_0 n - \phi[k,n]\right)}\right)$$

$$= \frac{A_0}{4}\sum_n \left[2\cos\left(\frac{2\pi}{N}k_0 n + \phi[k,n]\right) + 2\cos\left(\frac{2\pi}{N}k_0 n - \phi[k,n]\right)\right]$$

$$X[k] = \mathrm{MDCT}\{x[n]\} = \frac{A_0}{2}\sum_n \left[\cos\left(\frac{2\pi}{N}k_0 n + \phi[k,n]\right) + \cos\left(\frac{2\pi}{N}k_0 n - \phi[k,n]\right)\right] \quad (2.62)$$

Con las ecuaciones (2.61) y (2.62) se demuestra que $\mathrm{MDCT}\{x[n]\} = \mathrm{Re}\{\mathrm{DFT}\{x[n]\}\}$ para señales periodicas. Y, por tanto, para un análisis espectral el uso de la MDCT es enteramente válida. Así:

$$\mathrm{MDCT}\{x[n]\} \equiv \mathrm{Re}\{\mathrm{DFT}\{x[n]\}\} = \mathrm{Re}\left\{\sum_n x[n] W_N^{\phi[k,n]}\right\}$$
$$= X[k] = \sum_n x[n]\mathrm{Re}\left\{W_N^{\phi[k,n]}\right\} = \sum_n x[n]\cos(\phi[k,n]) \quad (2.63)$$

En el Anexo 2, se realiza el desarrollo de la Ecuación (2.61) de la DFT junto a su inversa para el argumento $\phi[k,n] = \frac{2\pi}{N}kn$. Aquí se desarrollará el de la MDCT con el mismo argumento con el cual aparecerán características que muestran la diferencia entre estas transformadas y parte del por qué es un algoritmo particularmente adecuado para un códec de audio.

De nuevo con $x[n] = \cos\left(\frac{2\pi}{N}k_0 n\right)$, y $\phi[k,n] = \frac{2\pi}{N}kn$, en la Ecuación (2.58)

$$\begin{aligned}\text{MDCT}\{x[n]\} &= \frac{A_0}{4}\sum_{n=0}^{N-1}\left(e^{-j\left(\frac{2\pi}{N}k_0 n+\phi[k,n]\right)} + e^{j\left(\frac{2\pi}{N}k_0 n+\phi[k,n]\right)} + e^{-j\left(\frac{2\pi}{N}k_0 n-\phi[k,n]\right)} + e^{j\left(\frac{2\pi}{N}k_0 n-\phi[k,n]\right)}\right)\\ &= \frac{A_0}{4}\sum_{n=0}^{N-1}\left(e^{-j\frac{2\pi}{N}(k+k_0)n} + e^{j\frac{2\pi}{N}(k+k_0)n} + e^{j\frac{2\pi}{N}(k-k_0)n} + e^{-j\frac{2\pi}{N}(k-k_0)n}\right)\end{aligned}$$

La anterior expresión gracias a la ausencia de la parte imaginaria de la exponencial compleja, contiene dos sumatorias más a comparación de las resultantes en la DFT, las cuales se pueden abreviar en dos grupos:

$$s_{n1,\pm} = \sum_{n=0}^{N-1} e^{\pm j\frac{2\pi}{N}(k-k_0)n} = \sum_{n=0}^{N-1} 1 = N \;\text{ si }\; k = k_0$$

$$s_{n2,\pm} = \sum_{n=0}^{N-1} e^{\pm j\frac{2\pi}{N}(k+k_0)n} = \sum_{n=0}^{N-1} 1 = N \;\text{ si }\; k = -k_0$$

$$\text{MDCT}\{x[n]\} = X[k] = \frac{A_0}{4}\left(s_{n1,+} + s_{n1,-} + s_{n2,+} + s_{n2,-}\right)$$

Haciendo el desarrollo de las series de potencia:

$$s_{n(1,2),\pm} = \sum_{n=0}^{N-1} e^{\pm j\frac{2\pi}{N}(k\pm k_0)n} = 1 + e^{\pm j\frac{2\pi}{N}(k\pm k_0)} + \cdots + 1 + e^{\pm j\frac{2\pi}{N}(k\pm k_0)(N-1)}$$

Se multiplica a $s_{n(1,2),\pm}$ por $e^{\pm j\frac{2\pi}{N}(k\pm k_0)}$ para iniciar la serie en $n = 1$ y acabarla en $n = N$:

$$s_{n(1,2),\pm}\cdot e^{\pm j\frac{2\pi}{N}(k\pm k_0)} = e^{\pm j\frac{2\pi}{N}(k\pm k_0)} + e^{\pm j\frac{2\pi}{N}(k\pm k_0)(2)} + \cdots + e^{\pm j\frac{2\pi}{N}(k\pm k_0)(N)}$$

Se restan las dos anteriores series y resulta la forma cerrada de $s_{n(1,2),\pm}$:

$$s_{n(1,2),\pm}\left(1 - e^{\pm j\frac{2\pi}{N}(k\pm k_0)}\right) = 1 + e^{\pm j\frac{2\pi}{N}(k\pm k_0)(N)} \;\therefore$$

$$s_{n1,\pm} = \frac{1 - e^{\pm j2\pi(k-k_0)}}{1 - e^{\pm j\frac{2\pi}{N}(k-k_0)}} = 0, \qquad k = k_0 \; \forall \; (k - k_0) \in \mathbb{Z}$$

$$s_{n2,\pm} = \frac{1 - e^{\pm j2\pi(k+k_0)}}{1 - e^{\pm j\frac{2\pi}{N}(k+k_0)}} = 0, \qquad k = -k_0 \; \forall \; (k + k_0) \in \mathbb{Z}$$

Por lo tanto, de forma similar a la DFT:

$$s_{n1,\pm} = \begin{cases} N, k = k_0 \\ 0, k \neq k_0 \end{cases} = N\delta[k - k_0], \qquad s_{n2,\pm} = \begin{cases} N, k = -k_0 \\ 0, k \neq -k_0 \end{cases} = N\delta[k + k_0]$$

De los cuatro sumandos: $2\delta[k - k_0] + 2\delta[k + k_0]$

$$X[k] = N\frac{A_0}{2}(\delta[k - k_0] + \delta[k + k_0]), \qquad k = -\frac{N}{2}, \dots, 0, 1, \dots, \frac{N}{2} - 1 \tag{2.64}$$

Pero como en la MDCT las frecuencias van hasta la mitad de la sucesión se tiene:

$$X[k] = N\frac{A_0}{2}\delta[k - k_0], \qquad k = 0, 1, \dots, \frac{N}{2} - 1 \tag{2.65}$$

O

$$X[k] = N\frac{A_0}{2}\delta[k + k_0], \qquad k = -\frac{N}{2}, \dots, -1, 0 \tag{2.66}$$

Mientras que la DFT en la Ecuación (7.7) es el doble de la expresión (2.64):

$$\text{DFT}\left\{\cos\left(\frac{2\pi}{N}k_0 n\right)\right\} = \tilde{X}[k] = N\frac{A_0}{2}(\delta[k - k_0] + \delta[k + k_0])$$

Es decir que, para $\phi[k, n] = \frac{2\pi}{N}kn$ en las reglas de transformación y transformando tonos puros, la DFT y la MDCT son iguales, lo cual sugiere un ahorro computacional en la programación al solo computar la mitad de las frecuencias y evitar el uso de números complejos.

Aplicando la IMDCT a la expresión (2.64) y con el desarrollo de la serie hasta $N/2$:

$$x'[n] = \frac{4}{N}\sum_{k=0}^{N/2-1} N\frac{A_0}{4}\delta[k-k_0]\cos\left(\frac{2\pi}{N}kn\right)$$

$$= \frac{A_0}{2}\left(\sum_{k=0}^{\frac{N}{2}-1}\delta[k-k_0]e^{j\frac{2\pi}{N}k_0n} + \sum_{k=0}^{\frac{N}{2}-1}\delta[k-k_0]e^{-j\frac{2\pi}{N}k_0n}\right)$$

$$= \frac{A_0}{2}\left(0+\cdots+e^{-j\frac{2\pi}{N}k_0n}+0+\cdots+e^{j\frac{2\pi}{N}k_0n}+\cdots+0\right) = \sum_{k=-N/2+1}^{0} A_0\delta[k+k_0]\cos\left(\frac{2\pi}{N}kn\right) =$$

$$x'[n] = \frac{A_0}{2}\left(e^{-j\frac{2\pi}{N}k_0n} + e^{j\frac{2\pi}{N}k_0n}\right) = A_0\cos\left(\frac{2\pi}{N}k_0n\right)$$

$$x'[n] = \text{IMDCT}\{X[k]\} = x[n] = A_0\cos\left(\frac{2\pi}{N}k_0n\right) \tag{2.67}$$

Claramente la IMDCT genera una reconstrucción perfecta de la onda original con el argumento de Fourier $\phi[k,n] = \frac{2\pi}{N}kn$.

Al aplicar el argumento de la MDCT $\phi[k,n] = \frac{2\pi}{N}\left(n+\frac{N}{4}+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)$, a las expresiones de la DFT y MDCT, $\tilde{X}[k]$ y $X[k]$ resultan más intrincadas:

$$\text{DFT}\{x[n]\} = \tilde{X}[k] = A_0\sum_{n=0}^{N-1}\cos\left(\frac{2\pi}{N}k_0n\right)e^{-j\frac{2\pi}{N}\left(n+\frac{N}{4}+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)} =$$

$$\tilde{X}[k] = \frac{A_0}{2}\left(\sum_{n=0}^{N-1} e^{-j\frac{2\pi}{N}\left[k_0n+\left(n+\frac{N}{4}+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right]} + \sum_{n=0}^{N-1} e^{-j\frac{2\pi}{N}\left[-k_0n+\left(n+\frac{N}{4}+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right]}\right)$$

Ambas series de potencias pueden escribirse como:

$$s_{n\pm} = \sum_{n=0}^{N-1} e^{-j\frac{2\pi}{N}\left[\pm k_0n+\left(n+\frac{N}{4}+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right]} = e^{-j\frac{2\pi}{N}\left[\left(\frac{N}{4}+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right]} + \cdots + e^{-j\frac{2\pi}{N}\left[\pm k_0(N-1)+\left(N-1+\frac{N}{4}+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right]}$$

Se multiplica a s_n por $e^{-j\frac{2\pi}{N}\left(\pm k_0+k+\frac{1}{2}\right)}$, así:

$$s_{n\pm}\cdot e^{-j\frac{2\pi}{N}\left(\pm k_0+k+\frac{1}{2}\right)} = e^{-j\frac{2\pi}{N}\left[\pm k_0n+\left(1+\frac{N}{4}+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right]} + \cdots + e^{-j\frac{2\pi}{N}\left[\pm k_0(N)+\left(N+\frac{N}{4}+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right]}$$

Se restan estas dos series y resulta la forma cerrada de $s_{n\pm}$.

$$s_{n\pm} = \frac{e^{-j\frac{2\pi}{N}\left(\frac{N}{4}+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)} - e^{-j\frac{2\pi}{N}\left(\pm k_0+\left(\frac{5N}{4}+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right)}}{1 - e^{-j\frac{2\pi}{N}\left(k\pm k_0+\frac{1}{2}\right)}} \tag{2.68}$$

$$\tilde{X}[k] = \frac{A_0}{2}\left(\frac{e^{-j\frac{2\pi}{N}\left(\frac{N}{4}+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)} - e^{-j\frac{2\pi}{N}\left(k_0+\left(\frac{5N}{4}+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right)}}{1 - e^{-j\frac{2\pi}{N}\left(k+k_0+\frac{1}{2}\right)}} + \frac{e^{-j\frac{2\pi}{N}\left(\frac{N}{4}+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)} - e^{-j\frac{2\pi}{N}\left(-k_0+\left(\frac{5N}{4}+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right)}}{1 - e^{-j\frac{2\pi}{N}\left(k-k_0+\frac{1}{2}\right)}}\right) \tag{2.69}$$

Y en la MDCT:

$$X[k] = A_0 \sum_{n=0}^{N-1} \cos\left(\frac{2\pi}{N} k_0 n\right) \cos\left(\frac{2\pi}{N}\left(n+\frac{N}{4}+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right)$$

Se generan de nuevo cuatro series:

$$X[k] = \frac{A_0}{4}\left(\sum_{n=0}^{N-1} e^{-j\frac{2\pi}{N}\left[k_0 n+\left(n+\frac{N}{4}+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right]} + \sum_{n=0}^{N-1} e^{-j\frac{2\pi}{N}\left[-k_0 n+\left(n+\frac{N}{4}+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right]} + \sum_{n=0}^{N-1} e^{j\frac{2\pi}{N}\left[k_0 n+\left(n+\frac{N}{4}+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right]} + \sum_{n=0}^{N-1} e^{-j\frac{2\pi}{N}\left[k_0 n+\left(n+\frac{N}{4}+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right]}\right)$$

Las cuales cumplen con el mismo desarrollo de la Ecuación (2.68). La representación cerrada para las dos primeras series es:

$$s_{n\pm} = \sum_{n=0}^{N-1} e^{-j\frac{2\pi}{N}\left[\pm k_0 n+\left(n+\frac{N}{4}+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right]} = \frac{e^{-j\frac{2\pi}{N}\left(\frac{N}{4}+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)} - e^{-j\frac{2\pi}{N}\left(\pm k_0+\left(\frac{5N}{4}+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right)}}{1 - e^{-j\frac{2\pi}{N}\left(k\pm k_0+\frac{1}{2}\right)}} \tag{2.70}$$

Y las otras dos son las conjugadas de $s_{n\pm}$:

$$s^*_{n\pm} = \sum_{n=0}^{N-1} e^{j\frac{2\pi}{N}\left[\pm k_0 n+\left(n+\frac{N}{4}+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right]} = \frac{e^{j\frac{2\pi}{N}\left(\frac{N}{4}+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)} - e^{j\frac{2\pi}{N}\left(\pm k_0+\left(\frac{5N}{4}+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right)}}{1 - e^{j\frac{2\pi}{N}\left(k\pm k_0+\frac{1}{2}\right)}} \tag{2.71}$$

Así:

$$X[k] = \frac{A_0}{4}(s_{n+} + s_{n-} + s_{n+}^* + s_{n-}^*) \tag{2.72}$$

Y estableciendo:

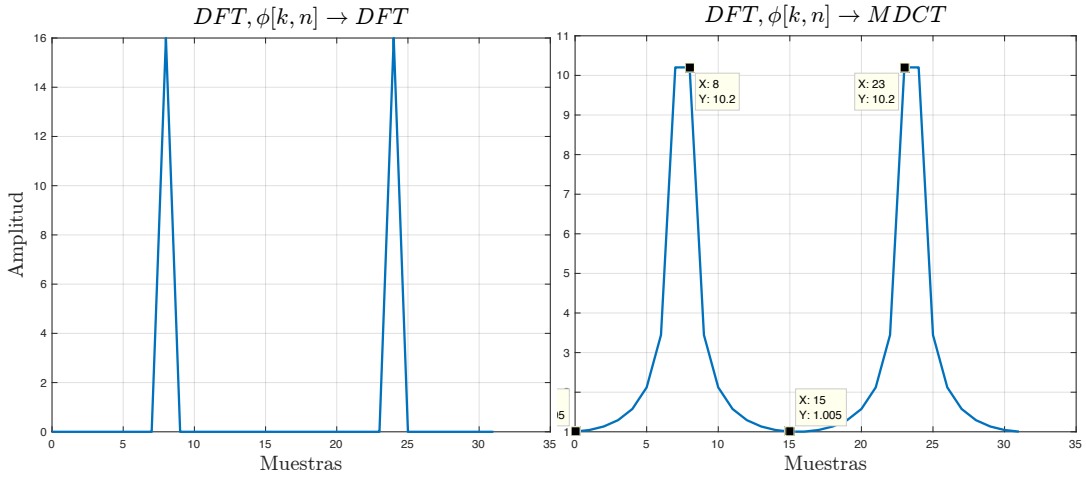
$$x = \frac{2\pi}{N}\left(\frac{N}{4} + \frac{1}{2}\right)\left(k + \frac{1}{2}\right), \qquad y_{\pm} = \frac{2\pi}{N}\left(\pm k_0 + \left(\frac{5N}{4} + \frac{1}{2}\right)\left(k + \frac{1}{2}\right)\right), \qquad z_{\pm} = \frac{2\pi}{N}\left(k \pm k_0 + \frac{1}{2}\right)$$

$X[k]$ resulta en una función puramente real como se demostró en la expresión (2.63:

$$\begin{aligned} s_{n+} + s_{n+}^* &= \frac{1}{1 - \cos(z_+)}[\cos(x) - \cos(y_+) - \cos(z_+ - x) + \cos(z_+ - y_+)] \\ s_{n-} + s_{n-}^* &= \frac{1}{1 - \cos(z_-)}[\cos(x) - \cos(y_-) - \cos(z_- - x) + \cos(z_- - y_-)] \end{aligned} \tag{2.73}$$

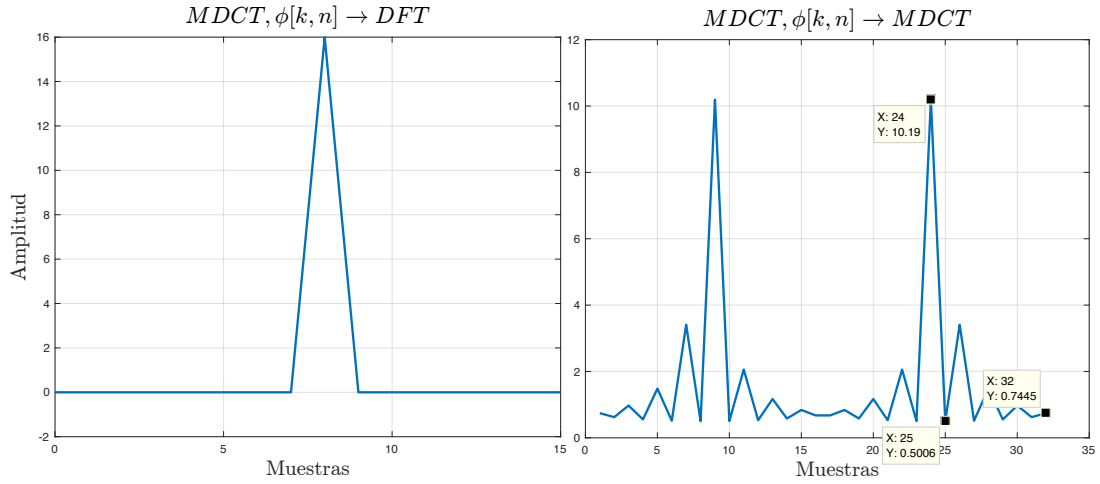
Realizando un ejemplo de la DFT y la MDCT para $N = 32, k_0 = 8$ con ambos argumentos $\phi[k,n]$ para las reglas de transformación, es claro que en la Figura 2–19 izquierda es la misma que la Figura 2–20 izquierda, aunque solo se hayan computado la mitad de las frecuencias para la MDCT. La expresión (2.69), para la Figura 2–19 derecha que usa el $\phi[k,n]$ de la MDCT claramente no genera dos impulsos, sino una función que converge progresivamente en pulsos centrados en la frecuencia propia de la señal k_0, esto es una distribución espectral alrededor de la componente espectral del tono puro k_0, lo cual genera que los picos (máximos) tengan un valor inferior a $A_0 \times N/2$. A demás como es visible en la gráfica, la magnitud no se cruza por el eje de las abscisas, por lo que no tiene soluciones puramente reales. Un comportamiento semejante ocurre con la MDCT utilizando su argumento $\phi[k,n]$. Al no tener parte imaginaria, su magnitud revela unos valores mínimos oscilantes sobre la unidad, por lo cual sus raíces en este ejemplo también son complejas. La parte real de la DFT es la misma función y genera la misma gráfica.

Figura 2–19: Magnitud de la DFT de $N = 32$, de un tono puro, $k_0 = 8$ ciclos/N, $\phi[k,n]$ de la DFT (Gráfica izquierda) y MDCT (Gráfica derecha).



Fuente: Imagen de elaboración propia usando Matlab_R2018a.

Figura 2–20: Magnitud de la MDCT de $N = 32$, de un tono puro, $k_0 = 8$ ciclos/N, con $\phi[k,n]$ de la DFT (Gráfica izquierda) y MDCT (Gráfica derecha).



Fuente: Imagen de elaboración propia usando Matlab_R2018a.

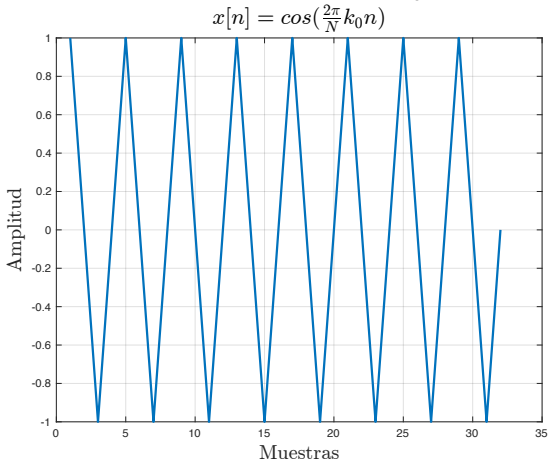
En la Figura 2–20, se muestra las reconstrucciones de la señal original (Figura 2–21) a partir de las transformadas inversas IDFT e IMDCT para un tono puro de 8 ciclos/Periodo. Estas transformaciones fueron computadas usando las expresiones en series de las Ecuaciones (2.54), (2.55), (2.58), y (2.59).

En la Figura 2–22 izquierda, la IDFT muestra una aparente simetría en la señal respecto a la mitad de su extensión en la cual se da cierta distorsión. La IMDCT (Figura 2–22 derecha) muestra una forma de onda igualmente simétrica respecto a su mitad, y una forma de onda distinta a la original, pero más estable respecto su forma/periodicidad.

Es claro que el hecho de no computar la mitad de las frecuencias para el cálculo de la MDCT y sumarlas en su inversa a comparación de la DFT donde se usan tantas frecuencias como la extensión de los arreglos temporales, establece una automatización, así como evitar el uso de números complejos (los cuales suelen doblar el almacenamiento por cada número real a la hora de ser programados).

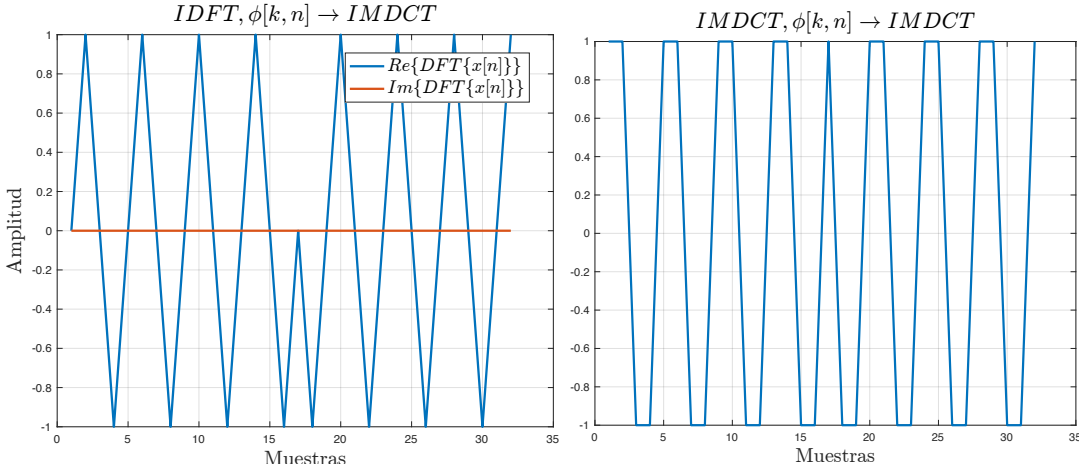
El precio de usar la MDCT es el uso del argumento $\phi[k,n] = \frac{2\pi}{N}\left(n + \frac{N}{4} + \frac{1}{2}\right)\left(k + \frac{1}{2}\right)$, el cual genera el fenómeno de simetría con distorsión que se vio anteriormente, para esto Princen et al. (1987) diseñan la MDCT con este $\phi[k,n]$ para trabajar transformando bloques que se sobrelapan a la mitad, ventaneándolos para que al computar su transformada inversa deban sobrelaparse de nuevo a la mitad y superponerse (sumarse), para obtener la reconstrucción perfecta de la señal original, sin aumentar la tasa de información espectral original dada la superposición de componentes. La justificación matemática de esto anterior se revela y desarrolla en la Sección 2.5.

Figura 2–21: Tono puro cosenoidal con frecuencia de $k_0 = 8$ ciclos/N.



Fuente: Imagen de elaboración propia. Matlab_R2018a.

Figura 2–22: Transformadas inversas IDFT (gráfica izquierda) e IMDCT (gráfica derecha) del tono puro de frecuencia $k_0 = 8$ ciclos/N.



Fuente: Imagen de elaboración propia usando Matlab_R2018a.

2.5 Derivación de las condiciones del TDAC para la Reconstrucción Perfecta con la Transformada MDCT

En esta sección se hará provecho de algunas herramientas del álgebra lineal, para desarrollar aquí la síntesis de ideas fundamentales tras el diseño y la implementación de la Transformada Modificada de Cosenos.

Así como las ventanas deben permitir la reconstrucción perfecta a través de la relación de sus estados distintos en las etapas de análisis y síntesis (Sección 2.4.2), el argumento (*regla de transformación,* -ver el Anexo 1-) en forma de coseno, que llamaremos $\boldsymbol{C}_N = \cos\left[\frac{2\pi}{N}\left(n+\frac{N}{4}+\frac{1}{2}\right)\left(\xi+\frac{1}{2}\right)\right]$ que está en la transformada MDCT y en su inversa -Ecuaciones (2.50) y (2.51)-, debe cumplir con una condición de relación unitaria, así como las ventanas.
Para la MDCT y su inversa, existe una *regla de transformación*, esta es una estructura matricial que permite la transformación de N muestras temporales de entrada en $N/2$ muestras espectrales, las cuales son transformadas de nuevo en N muestras temporales de salida. Esta es $\boldsymbol{C}_N$.

Dentro de los criterios para la reconstrucción perfecta, la *regla de transformación* $\boldsymbol{C}_N$, debe estar sometida a un criterio muy similar al de generar una sucesión unitaria ($u[n]$), como fue mencionado para las ventanas de análisis y síntesis. Respecto a estas dos etapas, se busca que las operaciones conjuntas de $\boldsymbol{C}_N$ en la transformada directa y la transformada inversa, permitan la menor variabilidad (distorsión/deformación) para las señales a transformar.

De las Ecuaciones (2.50) y (2.51), en los parámetros $\boldsymbol{C}_N$ respectivos, es visible la diferencia del término $4/N$ que se halla en la Ecuación (2.51) respecto a la Ecuación (2.50). Bosi & Goldberg (2004) aprovechan este término y las variables en las cuales se recorren ambas series de cada transformada para nombrar dos reglas de transformaciones distintas a partir $\boldsymbol{C}_N$, una en la etapa de análisis: ($\boldsymbol{C}_A$) y otra en la etapa de síntesis: ($\boldsymbol{C}_S$). La regla $\boldsymbol{C}_S$ contiene al término $4/N$ como factor y la serie que lo acompaña es evaluada en la sub-banda ξ en vez de n como es el caso de $\boldsymbol{C}_A$, por lo que se interpreta como la matriz transpuesta de $\boldsymbol{C}_A$ (se retiró el subíndice N para abreviar la notación).

Dado que se piensa en desplazamientos del 50% entre bloques sucesivos, este criterio es implícito pero existente dentro de la matriz $\boldsymbol{C}_N$, para lo cual Bosi & Goldberg (2004) introducen en el factor temporal dentro del coseno en un sumando de $N/2$, los $\boldsymbol{C}_N$ que contiene este desfase de $N/2$ llevando el subíndice "1": $\boldsymbol{C}_{N,1}$. Así resultan cuatro matrices, expresadas analíticamente en la Ecuación (2.74).

$$\begin{aligned}\boldsymbol{C}_{A,1}[\xi,n] &= \cos\left[\frac{2\pi}{N}\left(n+\frac{3N}{4}+\frac{1}{2}\right)\left(\xi+\frac{1}{2}\right)\right]\\ \boldsymbol{C}_{A,2}[\xi,n] &= \cos\left[\frac{2\pi}{N}\left(n+\frac{N}{4}+\frac{1}{2}\right)\left(\xi+\frac{1}{2}\right)\right]\end{aligned} \tag{2.74}$$

$$\boldsymbol{C}_{S,1}[n,\xi] = \frac{4}{N}\boldsymbol{C}_{A,1}^{T}[\xi,n] = \frac{4}{N}\boldsymbol{C}_{A,1}[n,\xi]$$
$$\boldsymbol{C}_{S,2}[n,\xi] = \frac{4}{N}\boldsymbol{C}_{A,2}^{T}[\xi,n] = \frac{4}{N}\boldsymbol{C}_{A,2}[n,\xi]$$

Y de forma matricial las expresiones (2.75) y (2.76):

$$\boldsymbol{C}_{A,(1,2)}[\xi,n] = \begin{matrix} n=0, & n, & n=N-1 \\ \begin{bmatrix} C_N[0,0] & \dots & C_N[N-1,0] \\ \vdots & \ddots & \vdots \\ C_N\left[0,\frac{N}{2}-1\right] & \dots & C_N\left[N-1,\frac{N}{2}-1\right] \end{bmatrix}_{N/2\times N} & \begin{matrix} \xi=0, \\ \xi, \\ \xi=N/2-1 \end{matrix} \end{matrix} \tag{2.75}$$

$$\boldsymbol{C}_{S,(1,2)}[n,\xi] = \frac{4}{N}\begin{matrix} \xi=0, & \xi, & \xi=N/2-1 \\ \begin{bmatrix} C_N[0,0] & \dots & C_N\left[0,\frac{N}{2}-1\right] \\ \vdots & \ddots & \vdots \\ C_N[N-1,0] & \dots & C_N\left[\frac{N}{2}-1,N-1\right] \end{bmatrix}_{N\times N/2} & \begin{matrix} n=0, \\ n, \\ n=N-1 \end{matrix} \end{matrix} \tag{2.76}$$

Entonces la transformada directa MDCT se puede escribir como:

$$\Psi_{k,\ell}[\xi] = \sum_{n=0}^{N-1} \psi_{k,\ell}[n]\boldsymbol{C}_{A,2}[\xi,n] \tag{2.77}$$

Y la transformada inversa IMDCT:

$$\bar{\bar{\gamma}}_{k,\ell}[n] = \sum_{\xi=0}^{\frac{N}{2}-1} \Psi_{k,\ell}[\xi]\boldsymbol{C}_{S,2}[n,\xi] = \frac{4}{N}\sum_{\xi=0}^{\frac{N}{2}-1} \Psi_{k,\ell}[\xi]\boldsymbol{C}_{A,2}^{T}[\xi,n] \tag{2.78}$$

Como es visible en la Figura 2–18, la superposición de cada mitad (lados izquierdo y derecho) de las señales de análisis y síntesis de por sí generan un vector unitario, lo cual ayuda a entender las Ecuaciones (2.79) y (2.80) dadas por Bosi & Goldberg (2004, p. 126), donde se dividen las ventanas de análisis y síntesis en dos señales aparte, izquierda (subíndice L) $w_L[n]$ y derecha (subíndice R) $w_R[n]$.

$$W_{S,i,R}[n]\cdot\boldsymbol{C}_{S,1}\cdot\boldsymbol{C}_{A,2}\cdot W_{A,i,L}[n] = W_{S,i,L}[n]\cdot\boldsymbol{C}_{S,2}\cdot\boldsymbol{C}_{A,1}\cdot W_{A,i,R}[n] = \boldsymbol{0} \tag{2.79}$$

$$W_{S,i,L}[n]\cdot\boldsymbol{C}_{S,2}\cdot\boldsymbol{C}_{A,2}\cdot W_{A,i,L}[n] + W_{S,i-1,R}[n]\cdot\boldsymbol{C}_{S,1}\cdot\boldsymbol{C}_{A,1}\cdot W_{A,i-1,R}[n] = \boldsymbol{1} = \boldsymbol{I} \tag{2.80}$$

Las formulaciones de las Ecuaciones (2.79) y (2.80), pretenden servir como los criterios de diseño completos que reúnen las funciones ventana, las reglas de transformación, los desplazamientos del 50% y el sobrelapamiento entre bloque consecutivos para conseguir

una reconstrucción perfecta utilizando las etapas de análisis y síntesis para la Transformada Modificada de Cosenos.

Reuniendo las Ecuaciones (2.77) y (2.78) para revisar las relaciones entre ventanas y *reglas de transformación* del granulo ℓ-ésimo para la señal temporal $s_\ell[k,n]$ de la banda k se tiene la siguiente expresión:

$$\bar{\bar{\gamma}}_{k,\ell}[n] = \sum_{\xi=0}^{\frac{N}{2}-1}\left[\sum_{n=0}^{N-1}\psi_{k,\ell}[n]\boldsymbol{C}_{A,2}[\xi,n]\right]\boldsymbol{C}_{S,2}[n,\xi]$$

Y multiplicando a ambos lados por la ventana de síntesis $w_{S,\ell}[n]$:

$$w_{S,\ell}[n]\bar{\bar{\gamma}}_{k,\ell}[n] = w_{S,\ell}[n]\sum_{\xi=0}^{\frac{N}{2}-1}\left[\sum_{n=0}^{N-1}w_{A,\ell}[n]s_\ell[k,n]\boldsymbol{C}_{A,2}[\xi,n]\right]\boldsymbol{C}_{S,2}[n,\xi] \tag{2.81}$$

Usando la misma relación, pero para el granulo anterior $\ell-1$:

$$\bar{\bar{\gamma}}_{k,\ell-1}[n] = \sum_{\xi=0}^{\frac{N}{2}-1}\left[\sum_{n=0}^{N-1}\psi_{k,\ell-1}[n]\boldsymbol{C}_{A,1}[\xi,n]\right]\boldsymbol{C}_{S,1}[n,\xi] =$$

$$w_{S,\ell-1}\left[n+\frac{N}{2}\right]\bar{\bar{\gamma}}_{k,\ell}[n] = w_{S,\ell-1}\left[n+\frac{N}{2}\right]\sum_{\xi=0}^{\frac{N}{2}-1}\left[\sum_{n=0}^{N-1}w_{A,\ell-1}\left[n+\frac{N}{2}\right]s_{\ell-1}[k,n]\boldsymbol{C}_{A,1}[\xi,n]\right]\boldsymbol{C}_{S,1}[n,\xi] \tag{2.82}$$

Sumando las Ecuaciones (2.81) y (2.82) y separando las ventanas en lados derecho e izquierdo, se adquiere la relación de Bosi & Goldberg (2004) de la Ecuación (2.80).

Una vez demostrado de forma independiente la relación unitaria entre ventanas de análisis y síntesis, lo mismo haremos para las reglas de transformación $\boldsymbol{C}_N$. Para revisar la relación entre las reglas de transformación $\boldsymbol{C}_N$, dado que son matrices y no arreglos unidimensionales, según las Ecuaciones (2.79) y (2.80), el orden de los productos genera resultados distintos, por lo que la secuencia de los procesos de transformación indica tal orden y por tanto, el desarrollo de las series que define las variables independientes de las funciones.

Recapitulando estas ideas, y observando que $\boldsymbol{C}_{S,(1,2)}$ actúa posteriormente sobre los efectos de $\boldsymbol{C}_{A,(1,2)}$, se observa que cada uno de los vectores sobre la transformada MDCT de la banda k-ésima ($\Psi_{k,\ell}[\xi]$) -cada una de las 32 filas de la Ecuación (2.83)- al aplicarle la regla de transformación $\boldsymbol{C}_{S,(1,2)}$ va a permitir que cada $\bar{\bar{\gamma}}_{k,\ell}[n]$ sea la imagen (ver Anexo 1) de cada $\Psi_{k,\ell}[\xi]$ sobre la acción de $\boldsymbol{C}_{S,(1,2)}$.

$$\Psi_{k,\ell}[\xi] = \begin{matrix} \xi = 0, & \xi, & \xi = N/2-1 \\ \begin{bmatrix} \Psi_{1,\ell}[0] & \dots & \Psi_{1,\ell}[N/2-1] \\ \vdots & \ddots & \vdots \\ \Psi_{32,\ell}[0] & \dots & \Psi_{32,\ell}[N/2-1] \end{bmatrix} & \begin{matrix} k=1, \\ k, \\ k=32 \end{matrix} \end{matrix} \tag{2.83}$$

Esto anterior quiere decir que $\boldsymbol{C}_{S,(1,2)}\left(\Psi_{k,\ell}[\xi]\right) = \bar{\bar{\gamma}}_{k,\ell}[n]$. Y por tanto se espera el producto matricial para un mismo bloque de la forma:

$$\begin{matrix} \boldsymbol{C}_{S,1} \cdot \boldsymbol{C}_{A,1} \\ \boldsymbol{C}_{S,2} \cdot \boldsymbol{C}_{A,2} \end{matrix} \tag{2.84}$$

Y entre bloques consecutivos:

$$\begin{matrix} \boldsymbol{C}_{S,1} \cdot \boldsymbol{C}_{A,2} \\ \boldsymbol{C}_{S,2} \cdot \boldsymbol{C}_{A,1} \end{matrix} \tag{2.85}$$

Recordemos los tamaños de los $\boldsymbol{C}_N$: $\boldsymbol{C}_{A,1}, \boldsymbol{C}_{A,2} \in \mathcal{M}_{N/2\times N}$ y $\boldsymbol{C}_{S,1}, \boldsymbol{C}_{S,2} \in \mathcal{M}_{N\times N/2}$. Donde $N = 32$ (número de bandas) en la implementación. Bosi & Goldberg (2004) sugieren que "solo $N/2$ muestras del dominio de la frecuencia son independientes de una TDAC de N muestras, implicando que solamente se requieren $N/2$ muestras de frecuencia de cada bloque transformado para una reconstrucción completa"; esto significa que el producto matricial entre los $\boldsymbol{C}_{S,(1,2)}$ con los $\boldsymbol{C}_{A,(1,2)}$ debe realizarse con todas las matrices reducidas a un tamaño $N/2 \times N/2$. Como se verá mas adelante, si este producto se realiza con los tamaños completos (resultando en matrices de tamaño $N \times N$), las matrices cuadradas diagonales internas resultantes de tamaño $N/2 \times N/2$ son copias anti-simetricas entre ellas.

El producto matricial C de un par de matrices A y B es visualizable como el producto interno iterativo de las columnas de B sobre las filas de A, de la forma: $C_{ij} = \langle A_{ik}, B_{kj} \rangle = \sum_{k=1}^{N} A_{ik} \cdot B_{kj}$. De esta forma, es más sencillo interpretar cada fila o columna de $\boldsymbol{C}_{S,(1,2)}$ y de $\boldsymbol{C}_{A,(1,2)}$ como vectores independientes que operan entre ellos -Ej.: Ecuación (2.86)-. Sabiendo que se genera el producto interno de las columnas de $\boldsymbol{C}_{S,(1,2)}$ (vectores en función de n) con las filas de $\boldsymbol{C}_{A,(1,2)}$(vectores en función de n), las series se recorren sobre la variable de frecuencia ξ.

$$\boldsymbol{C}_{S,(1,2)}[n] = \frac{4}{N} \begin{matrix} n=0 & n=1 & n=N-1 \\ \left\{ \begin{bmatrix} C_N[0,0] \\ \vdots \\ C_N\left[0, \frac{N}{2}-1\right] \end{bmatrix}, \begin{bmatrix} C_N[1,0] \\ \vdots \\ C_N\left[1, \frac{N}{2}-1\right] \end{bmatrix}, \dots, \begin{bmatrix} C_N[N-1,0] \\ \vdots \\ C_N\left[N-1, \frac{N}{2}-1\right] \end{bmatrix} \right\} & \begin{matrix} \xi = 0, \\ \vdots \\ \frac{N}{2}-1 \end{matrix} \end{matrix} \tag{2.86}$$

Hemos de revisar la ortogonalidad entre las reglas de transformación vistas en la Ecuación (2.74), para hallar los espacios nulos (el Kernel de $\boldsymbol{C}_N$, -ver Anexo 1-) entre los espacios vectoriales de las relaciones (2.84) y (2.85), y hallar tal condición para la cual el producto

interno de estas Ecuaciones resulte en la matriz identidad $\boldsymbol{I}$, dada por la Ecuación (2.80). Así, proponemos hallar las condiciones de ortogonalidad (nulidad) y de unicidad de $\boldsymbol{C}_N$ -operaciones que se dan conjuntamente en las Ecuaciones-relaciones (2.84) y (2.85)-, a partir de demostrar las Ecuaciones (2.87) y (2.88).

Unicidad: Al tener índices de tiempo iguales $\boldsymbol{C}_S$ y $\boldsymbol{C}_A$, son un par igual de vectores, para los cuales su producto y proyección (ver Anexo 1) debe ser unitaria.

$$\langle \boldsymbol{C}_{S,(1,2)}[n], \boldsymbol{C}_{A,(1,2)}[n] \rangle = \frac{4}{N} \sum_{\xi=0}^{N/2-1} \cos^2 \left[\frac{2\pi}{N} \left(n + \frac{N}{4} + \frac{1}{2} \right) \left(\xi + \frac{1}{2} \right) \right] = 1 \qquad (2.87)$$

Ortogonalidad: Al tener índices de tiempo distintos n y m para $\boldsymbol{C}_S$ y $\boldsymbol{C}_A$ respectivamente, se generan cosenos distintos, para los cuales su producto interno y por tanto proyección, ha de ser nula.

$$\langle \boldsymbol{C}_{S,(1,2)}[n], \boldsymbol{C}_{A,(2,1)}[m] \rangle =$$
$$= \frac{4}{N} \sum_{\xi=0}^{N/2-1} \cos \left[\frac{2\pi}{N} \left(n + \frac{N}{4} + \frac{1}{2} \right) \left(\xi + \frac{1}{2} \right) \right] \cos \left[\frac{2\pi}{N} \left(m + \frac{N}{4} + \frac{1}{2} \right) \left(\xi + \frac{1}{2} \right) \right] = 0 \qquad (2.88)$$

Demostración de Unicidad: Con esta operación se demuestra que aparecen vectores unitarios en la Ecuación (2.84), dado al producto interno de los vectores al ser iguales sobre n y ξ para los $\boldsymbol{C}_N$ de síntesis y análisis.

$$\langle \boldsymbol{C}_{S,(1,2)}[n], \boldsymbol{C}_{A,(1,2)}[n] \rangle = \frac{4}{N} \sum_{\xi=0}^{N/2-1} \cos^2 \left[\frac{2\pi}{N} \left(n + \frac{N}{4} + \frac{1}{2} \right) \left(\xi + \frac{1}{2} \right) \right] = 1$$

$$\cos^2(x) = \frac{1}{2}[1 + \cos(2x)] = \frac{1}{2} + \frac{1}{4}\left(e^{-j2x} + e^{j2x}\right), \qquad f(n) = n + \frac{N}{4} + \frac{1}{2}$$

$$\langle \boldsymbol{C}_{S,(1,2)}[n], \boldsymbol{C}_{A,(2,1)}[n] \rangle = \frac{4}{N} \sum_{\xi=0}^{N/2-1} \frac{1}{2} + \frac{4}{N} \sum_{\xi=0}^{N/2-1} \frac{1}{4} \left(e^{-j\frac{4\pi}{N} f(n)\left(\xi+\frac{1}{2}\right)} + e^{j\frac{4\pi}{N} f(n)\left(\xi+\frac{1}{2}\right)} \right)$$

El desarrollo de la serie de potencias geométrica hasta $N/2$ de ambas exponenciales complejas es:

$$s_\xi = \sum_{\xi=0}^{N/2-1} e^{\pm j\frac{4\pi}{N} f(n)\left(\xi+\frac{1}{2}\right)} = e^{\pm j\frac{4\pi}{N} f(n)\left(\frac{1}{2}\right)} + e^{\pm j\frac{4\pi}{N} f(n)\left(\frac{3}{2}\right)} + \cdots + e^{\pm j\frac{4\pi}{N} f(n)\left(\frac{N}{2}-1+\frac{1}{2}\right)}$$

Se multiplica la serie s_ξ por $e^{\pm j\frac{4\pi}{N} f(n)}$:

$$s_\xi \cdot e^{\pm j\frac{4\pi}{N}f(n)} = e^{\pm j\frac{4\pi}{N}f(n)\left(\frac{3}{2}\right)} + e^{\pm j\frac{4\pi}{N}f(n)\left(\frac{5}{2}\right)} + \cdots + e^{\pm j\frac{4\pi}{N}f(n)\left(\frac{N}{2}+\frac{1}{2}\right)}$$

Se restan las dos series anteriores para cancelar los términos iguales y así hallar una expresión exacta de s_ξ:

$$s_\xi - s_\xi e^{\pm j\frac{4\pi}{N}f(n)} = e^{\pm j\frac{2\pi}{N}f(n)} - e^{\pm j\frac{4\pi}{N}f(n)\left(\frac{N}{2}+\frac{1}{2}\right)}$$

$$s_\xi = \frac{e^{\pm j2\pi f(n)\left(\frac{1}{N}\right)} - e^{\pm j2\pi f(n)\left(1+\frac{1}{N}\right)}}{1 - e^{\pm j4\pi f(n)\left(\frac{1}{N}\right)}}$$

Revisando la aparente nulidad del numerador de s_ξ, dados los pares enteros de las potencias de la exponencial compleja que acompañan al factor (2π):

$$e^{\pm j2\pi f(n)\left(\frac{1}{N}\right)} - e^{\pm j2\pi f(n)\left(1+\frac{1}{N}\right)} = 0, \qquad e^{\pm j2\pi f(n)\left(\frac{1}{N}\right)} = e^{\pm j2\pi f(n)} e^{\pm j2\pi f(n)\left(\frac{1}{N}\right)}$$

$$1 = e^{\pm j2\pi f(n)} \therefore e^{\pm j2\pi f(n)\left(\frac{1}{N}\right)} - e^{\pm j2\pi f(n)\left(\frac{1}{N}\right)} = 0$$

$$\therefore 1 = e^{\pm j2\pi\left(n+\frac{N}{4}+\frac{1}{2}\right)} \ \forall \ 2\left(n+\frac{N}{4}+\frac{1}{2}\right) \in \mathbb{Z}^+\text{pares}$$

Al ver la condición anterior, se demuestra que el numerador de s_ξ es cero, y así se realiza solamente la serie de la constante $1/2$, $N/2$ veces:

$$\langle \boldsymbol{C}_{S,(1,2)}[n], \boldsymbol{C}_{A,(2,1)}[n] \rangle = \frac{4}{N}\sum_{\xi=0}^{N/2-1}\frac{1}{2} = \frac{4}{N}\left(\frac{N}{4}\right) = 1$$

Demostración de ortogonalidad: Con esta operación se demuestra que aparecen vectores nulos en la Ecuación (2.84), dado al producto interno de los vectores al ser diferentes sobre n y ξ para los $\boldsymbol{C}_N$ de síntesis y análisis. Por otra parte, esta demostración también satisface la Ecuación (2.85) donde se observa el producto interno de cosenos desfasados al 50% en el tiempo, es decir al sumar $N/2$ en el factor temporal de $\boldsymbol{C}_N$ (siendo así que $\boldsymbol{C}_{A,2}$ y $\boldsymbol{C}_{S,2}$ son los Kernel respectivos de $\boldsymbol{C}_{A,1}$ y $\boldsymbol{C}_{S,1}$).

$$\langle \boldsymbol{C}_{S,(1,2)}[n], \boldsymbol{C}_{A,(1,2)}[m] \rangle = \frac{4}{N}\sum_{\xi=0}^{N/2-1}\cos\left[\frac{2\pi}{N}\left(n+\frac{N}{4}+\frac{1}{2}\right)\left(\xi+\frac{1}{2}\right)\right]\cos\left[\frac{2\pi}{N}\left(m+\frac{N}{4}+\frac{1}{2}\right)\left(\xi+\frac{1}{2}\right)\right] = 0$$

$$a = \frac{2\pi}{N}\left(n+\frac{N}{4}+\frac{1}{2}\right)\left(\xi+\frac{1}{2}\right), \qquad b = \frac{2\pi}{N}\left(m+\frac{N}{4}+\frac{1}{2}\right)\left(\xi+\frac{1}{2}\right)$$

$$\cos(a)\cos(b) = \frac{1}{4}\left(e^{-j(a+b)} + e^{-j(a-b)} + e^{j(a-b)} + e^{j(a+b)}\right)$$

$$a+b = \frac{2\pi}{N}\left(\frac{N}{2} + 1 + n + m\right)\left(\xi + \frac{1}{2}\right), \qquad a-b = \frac{2\pi}{N}(n-m)\left(\xi + \frac{1}{2}\right)$$

$$\left\langle \boldsymbol{C}_{S,(1,2)}[n], \boldsymbol{C}_{A,(1,2)}[m] \right\rangle = \frac{1}{4}\sum_{\xi=0}^{N/2-1}\left[e^{-j(a+b)} + e^{-j(a-b)} + e^{j(a-b)} + e^{j(a+b)}\right]$$

De forma similar a la demostración anterior, se usa el desarrollo de la serie de potencias geométrica hasta $N/2$ para obtener expresiones exactas de los cuatro sumandos anteriores, $A_{\pm}$ y $B_{\pm}$:

$$A_{\pm} = \sum_{\xi=0}^{N/2-1} e^{\pm j(a+b)} = \sum_{\xi=0}^{N/2-1} e^{\pm j\frac{2\pi}{N}\left(\frac{N}{2}+1+n+m\right)\left(\xi+\frac{1}{2}\right)} = \frac{e^{\pm j\pi\left(\frac{N}{2}+1+n+m\right)\left(\frac{1}{N}\right)} - e^{\pm j\pi\left(\frac{N}{2}+1+n+m\right)\left(1+\frac{1}{N}\right)}}{1 - e^{\pm j\frac{2\pi}{N}\left(\frac{N}{2}+1+n+m\right)}}$$

$$B_{\pm} = \sum_{\xi=0}^{N/2-1} e^{\pm j(a-b)} = \sum_{\xi=0}^{N/2-1} e^{\pm j\frac{2\pi}{N}(n-m)\left(\xi+\frac{1}{2}\right)} = \frac{e^{\pm j\pi(n-m)\left(\frac{1}{N}\right)} - e^{\pm j\pi(n-m)\left(1+\frac{1}{N}\right)}}{1 - e^{\pm j\frac{2\pi}{N}(n-m)}}$$

Revisando los exponentes de $A_{\pm}$ y $B_{\pm}$, es evidente que solo se distinguen en los términos $a+b$ y $a-b$, así se pueden reemplazar estos términos por una variable auxiliar $x \in \mathbb{Z}^+$ y se escribe, $C_{\pm}$ como la generalización de $A_{\pm}$ y $B_{\pm}$:

$$C_{\pm} = \frac{e^{\pm j\pi x\left(\frac{1}{N}\right)} - e^{\pm j\pi x\left(1+\frac{1}{N}\right)}}{1 - e^{\pm j\frac{2\pi}{N}x}} = \frac{e^{\pm j\pi x\left(\frac{1}{N}\right)} - e^{\pm j\pi x\left(\frac{1}{N}\right)}e^{\pm j\pi x}}{1 - e^{\pm j\frac{2\pi}{N}x}}$$

$$e^{\pm j\pi x} = (-1)^x \;\; \forall \;\; x \in \mathbb{Z}^+$$

$$\therefore C_{\pm} = \frac{e^{\pm j\pi x\left(\frac{1}{N}\right)} - (-1)^x e^{\pm j\pi x\left(\frac{1}{N}\right)}}{1 - e^{\pm j\frac{2\pi}{N}x}} = \frac{e^{\pm j\pi x\left(\frac{1}{N}\right)}[(-1)^x - 1]}{e^{\pm j\frac{2\pi}{N}x} - 1}$$

Finalmente, al sumar los pares de series $A_+ + A_- = 0 = B_+ + B_-$, se adquiere el valor esperado de cero que comprueba la ortogonalidad entre cosenos de diferentes índices de tiempo:

$$C_+ + C_- = \frac{e^{-j\pi x\left(\frac{1}{N}\right)}[(-1)^x - 1]}{e^{-j\frac{2\pi}{N}x} - 1} + \frac{e^{j\pi x\left(\frac{1}{N}\right)}[(-1)^x - 1]}{e^{j\frac{2\pi}{N}x} - 1} =$$

$$= \frac{\left(e^{-j\frac{\pi}{N}x} - e^{j\frac{\pi}{N}x}\right)[(-1)^x - 1] + \left(e^{j\frac{\pi}{N}x} - e^{-j\frac{\pi}{N}x}\right)[(-1)^x - 1]}{2 - e^{-j\frac{2\pi}{N}x} - e^{j\frac{2\pi}{N}x}} =$$

$$= \frac{[(-1)^x - 1]\left(e^{j\frac{\pi}{N}x} - e^{-j\frac{\pi}{N}x} + e^{-j\frac{\pi}{N}x} - e^{j\frac{\pi}{N}x}\right)}{2 - e^{-j\frac{2\pi}{N}x} - e^{j\frac{2\pi}{N}x}} = 0, \qquad \forall\, x : (n-m), \left(\frac{N}{2} + 1 + n + m\right) \in \mathbb{Z}^+$$

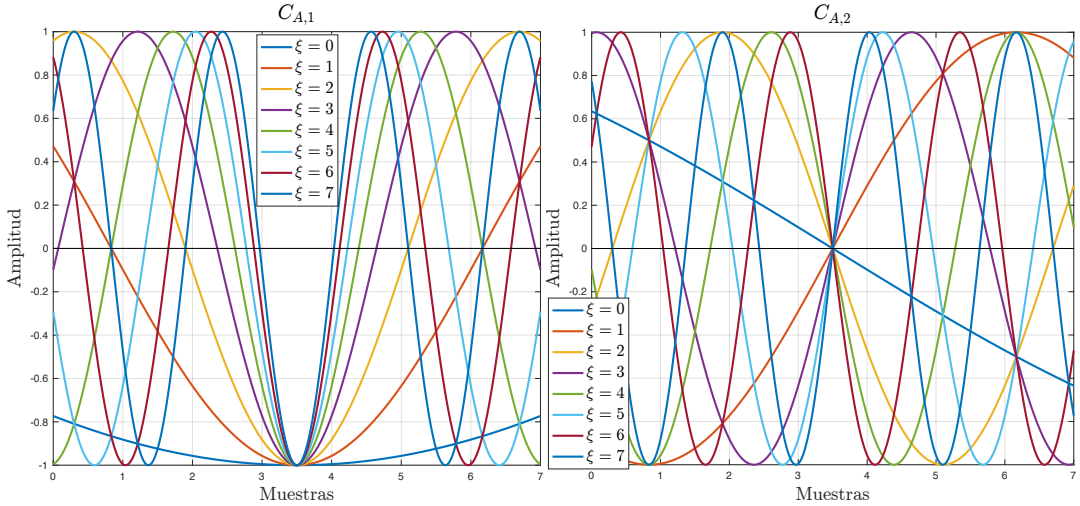
$$\therefore \left\langle \boldsymbol{C}_{S,(1,2)}[n], \boldsymbol{C}_{A,(1,2)}[m] \right\rangle = 0$$

Este resultado anterior aplica para las cuatro series A_+, A_-, B_+ y B_-, reemplazando x por $(n-m)$ en la suma de $A_+ + A_-$ y reemplazando x por $\left(\frac{N}{2} + 1 + n + m\right)$ en la suma $B_+ + B_-$.

Esta ortogonalidad confirma que las reglas de transformación de síntesis $\boldsymbol{C}_{S,(2,1)}$ respecto a sus transpuestas de análisis $\boldsymbol{C}_{A,(1,2)}$ son sus respectivas Kernel. Es decir, a través de $\boldsymbol{C}_{S,(2,1)}$, $\boldsymbol{C}_{A,(1,2)}$ adquiere valores nulos que permiten proyecciones a cero sobre distintos índices de frecuencia (sub-bandas ξ) -como ocurre en la Ecuación (2.79)-.

$$\boldsymbol{C}_{S,(2,1)} = \ker\left(\boldsymbol{C}_{A,(1,2)}\right) \therefore \boldsymbol{C}_{S,(2,1)} \cdot \ker\left(\boldsymbol{C}_{A,(1,2)}\right) = \left\langle \boldsymbol{C}_{S,(2,1)}[n], \boldsymbol{C}_{A,(1,2)}[n] \right\rangle = 0 \qquad (2.89)$$

Figura 2–23: Filas (sub-bandas ξ) de las Reglas de Transformación (Matrices) de la MDCT, $\boldsymbol{C}_{A,1}$ (izquierda) y $\boldsymbol{C}_{A,2}$ (derecha) para $N = 16$.



Fuente: Imagen de elaboración propia usando Matlab_R2018a.

La expresión matricial de la Ecuación (2.91) revela los valores numéricos de la Ecuación (2.84), y además la forma en que las Reglas de Transformación se proyectan unas sobre las otras. Es decir, retomando lo aprendido de las demostraciones de unicidad y ortogonalidad, del producto matricial $\boldsymbol{C}_{S,1} \cdot \boldsymbol{C}_{A,1}$ y $\boldsymbol{C}_{S,2} \cdot \boldsymbol{C}_{A,2}$ al proyectarse un par de cosenos iguales, dados sus índices de tiempo, se genera la diagonal unitaria $\boldsymbol{I}$, pero cuando estos cosenos son

diferentes al tener índices de tiempo distintos, estos son ortogonales y por tanto sus proyecciones son nulas.

En las proyecciones realizadas entre cosenos, la anti-diagonal positiva $\boldsymbol{J}$ en la Ecuación (2.91) se genera debido a la propiedad simétrica-par vista en la Figura 2–23 (izquierda), y la anti-diagonal negativa $-\boldsymbol{J}$ en la Ecuación (2.92) se genera debido a la propiedad anti-simétrica impar vista en la Figura 2–23 (derecha). Las matrices que relacionan las operaciones pertinentes entre cosenos se dan a continuación:

$$\boldsymbol{J} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ \vdots & \vdots & \vdots & \iddots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 1 & 0 & 0 & \cdots & 0 \end{bmatrix}_{\frac{N}{2}\times\frac{N}{2}} \tag{2.90}$$

$$\boldsymbol{C}_{S,1} \cdot \boldsymbol{C}_{A,1} = \boldsymbol{I} + \boldsymbol{J} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 1 & 0 & \cdots & \iddots & 0 \\ 0 & 0 & 1 & 1 & \cdots & 0 \\ \vdots & \vdots & 1 & 1 & \vdots & \vdots \\ 0 & 1 & \vdots & \vdots & \ddots & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}_{\frac{N}{2}\times\frac{N}{2}} \tag{2.91}$$

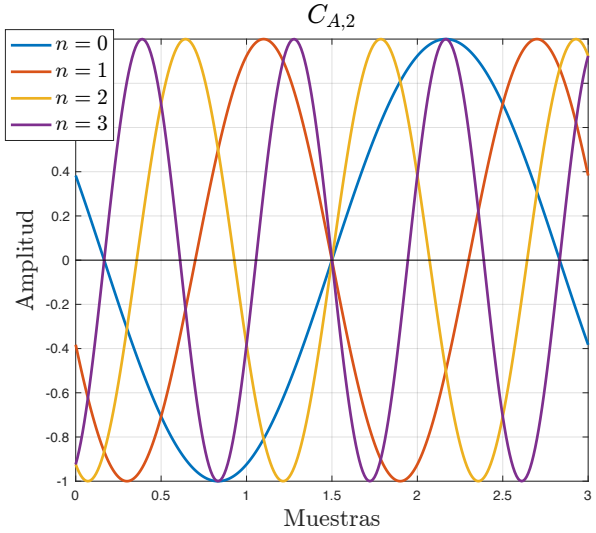
$$\boldsymbol{C}_{S,2} \cdot \boldsymbol{C}_{A,2} = \boldsymbol{I} - \boldsymbol{J} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & -1 \\ 0 & 1 & 0 & \cdots & \iddots & 0 \\ 0 & 0 & 1 & -1 & \cdots & 0 \\ \vdots & \vdots & -1 & 1 & \vdots & \vdots \\ 0 & -1 & \vdots & \vdots & \ddots & 0 \\ -1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}_{\frac{N}{2}\times\frac{N}{2}} \tag{2.92}$$

$$\boldsymbol{I} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}_{\frac{N}{2}\times\frac{N}{2}} \tag{2.93}$$

Si se hace por ejemplo $\boldsymbol{C}_{S,1} \cdot \boldsymbol{C}_{A,1}$ o $\boldsymbol{C}_{S,2} \cdot \boldsymbol{C}_{A,2}$, para $N = 8$, basta con variar los índices de tiempo hasta $N/2$ para obtener los resultados de las Ecuaciones anteriores. Extender el cálculo recorriendo los índices de tiempo hasta N, generará la misma matriz identidad $\boldsymbol{I}$ sumada con la matriz anti-diagonal $\boldsymbol{J}$ del signo contrario respecto a la matriz $\boldsymbol{J}$ del cálculo con las primeros $N/2$ índices de tiempo. Lo anterior se muestra en la Ecuación (2.94), y ocurre gracias a que n después de $N/2$ toman valores anti-simétricos respecto a su primera mitad como es visible en la Figura 2–24.

$$\boldsymbol{C}_{S,(1,2)} \cdot \boldsymbol{C}_{A,(1,2)} = \boldsymbol{I} \pm \boldsymbol{J} = \begin{bmatrix} 1 & 0 & 0 & \pm 1 \\ 0 & 1 & \pm 1 & 0 \\ 0 & \pm 1 & 1 & 0 \\ \pm 1 & 0 & 0 & 1 \end{bmatrix} \quad \boldsymbol{0}_{4\times4} \quad \begin{bmatrix} 1 & 0 & 0 & \mp 1 \\ 0 & 1 & \mp 1 & 0 \\ 0 & \mp 1 & 1 & 0 \\ \mp 1 & 0 & 0 & 1 \end{bmatrix}_{8\times8} \tag{2.94}$$

Figura 2–24: Columnas (tiempos n) de la Regla de Transformación (Matriz) de la MDCT, $\boldsymbol{C}_{A,2}$ para $N = 4$. Fuente: Imagen de elaboración propia. Matlab_R2018a.



La Ecuación (2.80) que establece la condición para la reconstrucción perfecta es la relación matemática de las ventanas y las reglas de transformación en las etapas de análisis y síntesis en dos bloques consecutivos de datos, esta es igual a la matriz identidad de la Ecuación (2.93). Es notable que en este cálculo se cancelan las matrices anti-diagonales $\boldsymbol{J}$, y la diagonal unitaria principal $\boldsymbol{I}$ se mantiene unitaria (y no igual a dos) gracias a los efectos de las ventanas en cada bloque de datos procesado. Este proceso es calculado variando los índices de tiempo hasta N, por lo que el resultado será la matriz unitaria de la Ecuación (2.95).

$$W_{S,i,L}[n] \cdot \boldsymbol{C}_{S,2} \cdot \boldsymbol{C}_{A,2} \cdot W_{A,i,L}[n] + W_{S,i-1,R}[n] \cdot \boldsymbol{C}_{S,1} \cdot \boldsymbol{C}_{A,1} \cdot W_{A,i-1,R}[n] =$$

$$\boldsymbol{I} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & & & & & \\ 0 & 1 & 0 & \cdots & 0 & & & & & \\ 0 & 0 & 1 & \cdots & 0 & & \mathbf{0}_{\frac{N}{2}\times\frac{N}{2}} & & & \\ \vdots & \vdots & \vdots & \ddots & 0 & & & & & \\ 0 & 0 & 0 & 0 & 1 & & & & & \\ & & & & & 1 & 0 & 0 & \cdots & 0 \\ & & \mathbf{0}_{\frac{N}{2}\times\frac{N}{2}} & & & 0 & 1 & 0 & \cdots & 0 \\ & & & & & 0 & 0 & 1 & \cdots & 0 \\ & & & & & \vdots & \vdots & \vdots & \ddots & 0 \\ & & & & & 0 & 0 & 0 & 0 & 1 \end{bmatrix}_{N\times N} \tag{2.95}$$

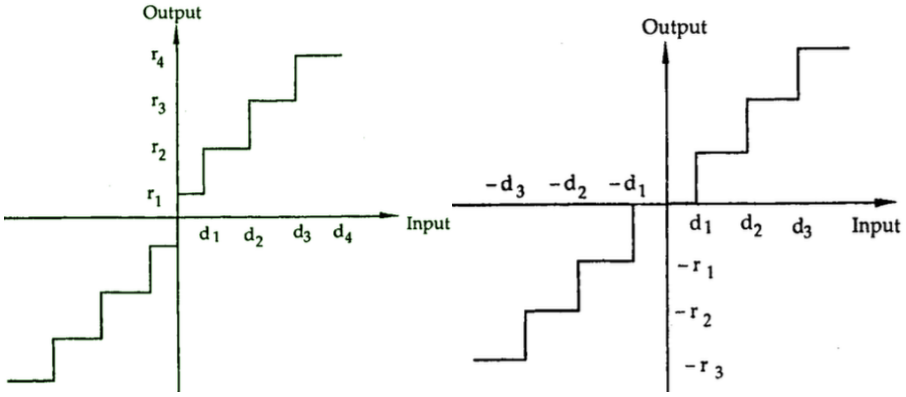
2.6 Modelos de Cuantización

Rao & Yip (1990) definen la cuantización como el mapeo de amplitudes continuas en "estados" o niveles, los cuales son representados con una cantidad finita de bits. Así, el proceso inverso, la de-cuantización es el mapeo de *R* niveles de bits en amplitudes continuas. La forma en que estos dos procesos se realizan, para realizar distintas representaciones funcionales de las señales de entrada y salida depende del modelo de cuantización aplicado.

2.6.1 Cuantizadores Uniformes

Las amplitudes de una señal pueden ser tanto positivas como negativas, por ello se suelen utilizar cuantizadores que representen de forma simétrica todo el rango de amplitudes, logrando tener una referencia del valor mínimo como eje de simetría y repartir en 2 cantidades iguales de nivel, las amplitudes positivas y negativas. Los cuantizadores "Midrise" no poseen una representación del nivel 0, mientras que los "Midtread" sí, como se representa en la **Figura 2–25**.

Figura 2–25: Cuantizadores uniformes simétricos. Tipo Midrise (imagen izquierda), tipo Midtread (imagen derecha).

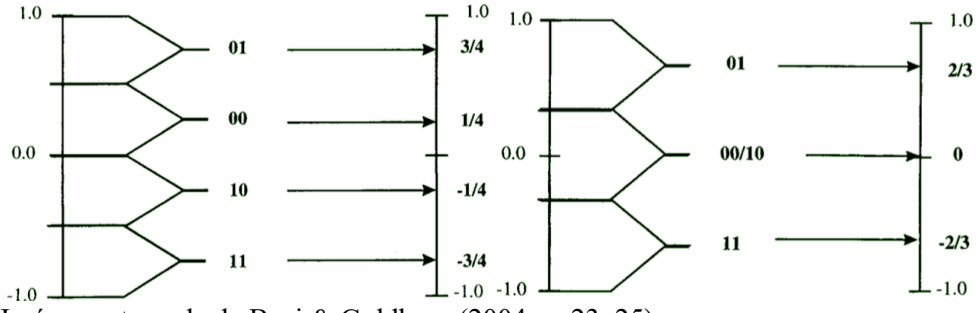


Fuente: Imágenes tomada de Rao & Yip (1990).

Los cuantizadores uniformes Midtread al ser capaces de tener un nivel de salida igual a 0, poseen un número impar de $(2^R - 1)$ estados de salida, en cambio los Midrise poseen una cantidad par de (2^R) estados de salida, al no tener el nivel 0. Bosi & Goldberg (2004) sugieren que a pesar del menor número de niveles que genera el cuantizador Midtread, gracias a la naturaleza de su distribución, este tipo de cuantizador entrega mejores resultados para señales de audio.

Como ejemplo, en la **Figura 2–26**, se ilustran los estados posibles para representar señales de audio entre los valores -1 y 1 con un par de codificadores uniformes Midrise y Midtread de 2 bits.

Figura 2–26: Cuantizadores uniformes simétricos de 2 bits Tipo Midrise (imagen izquierda), tipo Midtread (imagen derecha).



Fuente: Imágenes tomada de Bosi & Goldberg (2004, p. 23, 25).

Como regla, existe siempre un valor de entrada máximo antes de distorsión $|x_{max}|$y la distancia uniforme Δ, a la cual va a estar separada cada amplitud de entrada $x(t)$. El tamaño Δ es inversamente proporcional al número de rangos de entrada N una vez se haya seleccionado x_{max}.

$$N = 2 \times \frac{|x_{max}|}{\Delta} \rightarrow \Delta_{\text{midrise}} = 2 \times \frac{|x_{max}|}{2^R}, \qquad \Delta_{\text{midtread}} = 2 \times \frac{|x_{max}|}{2^R - 1} \tag{2.96}$$

Rao & Yip (1990) mencionan que cualquier cuantizador uniforme tiene como error de redondeo (q) máximo, la mitad del ancho de la distancia Δ ($q = \pm \Delta/2$) a cualquier nivel de entrada (entre $|x_{max}|$). Lo crítico es que este error puede ser considerable grande al representar amplitudes muy bajas. Así, Bosi & Goldberg (2004) concluyen que gracias a la percepción respecto a la escucha relativa del error en amplitud variable es mayor que el tamaño del error absoluto, los cuantizadores uniformes tienen un bajo desempeño perceptual, por lo cual nace la necesidad de hallar cuantizadores no-uniformes los cuales hacen proporcional el error de redondeo a la magnitud de la amplitud de entrada.

Figura 2–27: Algoritmo de cuantización y decuantización uniforme tipo Midtread para *R* bits.

Quantize:

$$\text{code}(\text{number}; R) = [s][|\text{code}|]$$

where

$$s = \begin{cases} 0 & \text{number} \geq 0 \\ 1 & \text{number} < 0 \end{cases}$$

$$|\text{code}| = \begin{cases} 2^{R-1} - 1 & \text{when } |\text{number}| \geq 1 \\ \text{INT}(((2^R - 1)|\text{number}| + 1)/2) & \text{elsewhere} \end{cases}$$

Dequantize:

$$\text{number}(\text{code}; R) = \text{sign} * |\text{number}|$$

where

$$\text{sign} = \begin{cases} 1 & \text{if } s = 0 \\ -1 & \text{if } s = 1 \end{cases}$$

$$|\text{number}| = 2.|\text{code}| / (2^R - 1)$$

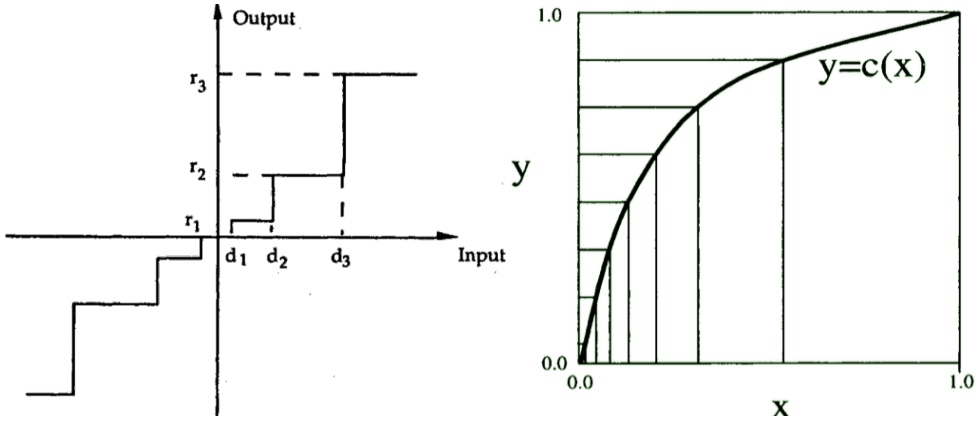
Fuente: Imágenes tomada de Bosi & Goldberg (2004, p. 26).

2.6.2 Cuantizadores no Uniformes

Aquellos cuantizadores cuyo paso de cuantización Δ sea variable respecto a los valores de amplitud entrantes, se denomina Cuantizador no Uniforme (Figura 2–28). Para esto se utiliza el método de "companding", el cual comprime una señal de entrada y luego la expande. Bosi & Goldberg (2004) lo describen al lograr pasar la señal de entrada a través de una función monotónica ascendente $y[n] = f(x[n])$, para luego ser cuantizada de forma uniforme generando a $y_{uq}[n]$. La decuantización es llevada a cabo primero de-cuantizando los valores uniformes $y_{uq}[n]$ a valores $y'[n]$ y luego pasándolos por la función inversa $x'[n] = f^{-1}(y'[n])$. La función $f(x)$ suele ser antisimétrica alrededor de 0, produciendo que las

amplitudes negativas se mapeen a valores positivos, lo cual permite definir completamente a $f(x)$ más bien como $f(|x|)$. Siendo así, el rango de valores de entrada entre -1.0 y 1.0 se mapean entre 0.0 y 1.0, supeditando a $f(x)$ como una función monotónica ascendente para que esta pueda ser fácilmente invertible.

Figura 2–28: Cuantización no uniforme (imagen izquierda). Efecto de la companding en las distancias Δ para un cuantizador Midtread de 4 bits Tipo Midrise (imagen derecha).



Fuente: Imagen derecha tomada de Bosi & Goldber (2004, p. 28). Imagen izquierda tomada de Rao & Yip (1990).

Este modelo general no-uniforme resulta muy útil ya que dependiendo de la forma funcional de $f(x)$, el ruido generado por el error de cuantización puede ser trasladado de las pequeñas amplitudes de entrada a las amplitudes de entrada mayores.

2.6.3 Companding de ley de potencias

Haciendo que la función monotónica $f(x)$ procese con una potencia constante p su argumento de entrada $x[n]$, se puede lograr una distribución de error de cuantización baja en amplitudes cercanas a 0 y mayores a medida que $|x[n]|$ se acerca a +1.0.

$$f_{pow}(y'[n]) = |x[n]|^p \tag{2.97}$$

En la implementación del esquema de compresión del MPEG 1-Layer III mostrada en la Sección 4.4.3, la potencia p es igual a $3/4 = 0.75$ en donde se utiliza un cuantizador midtread no uniforme que cuantiza las líneas de frecuencia de la MDCT.

2.7 Marco de Normativo

En la sección 2.7.1 se presentan algunas de las estipulaciones necesarias para implementar los sistemas Layer II y III, que se establecen en la normativa ISO/IEC. 11172-3 (Smith & Abel, 1993).

En la sección 2.7.2 Se presenta un resumen de las descripciones del método para la evaluación subjetiva del "nivel de calidad intermedia de los sistemas de codificación de audio", dado en la Recomendación ITU-R BS-1534, (ITU-R BS.1534-1, 2015, p. 1), que se

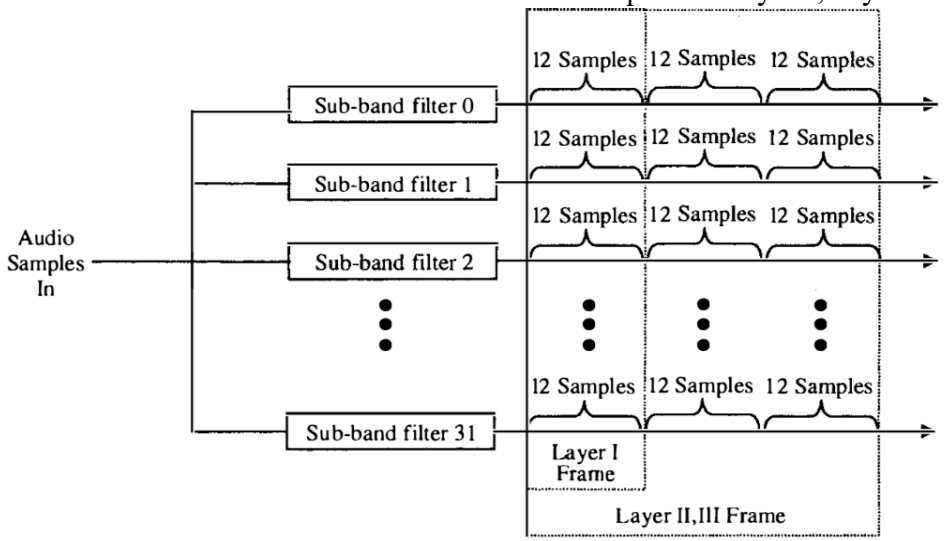
presenta como una extensión particular para sistemas de compresión audio de la Recomendación ITU-R-BS-1116-1, mencionado en (ITU-R BS.1534-1, 2015, p. 1).

2.7.1 Revisión de la Normativa ISO/IEC 11172-3 (1993)

A lo largo de todo el presente documento, se realizan las referencias pertinentes de la normativa ISO/IEC. 11172-3 (Smith & Abel, 1993) que hacen alusión a las condiciones y algoritmos para implementar los esquemas de compresión Layer II y Layer III. No obstante, aquí se mencionan algunos de los puntos fundamentales más importantes que estipula dicha normativa.

En los Alcances de ISO/IEC. 11172-3 (Smith & Abel, 1993, p. 1) menciona que todos los modelos algorítmicos han sido diseñados para las frecuencias de muestreo de 32k, 44.1k y 48k Hz. En la sección 2.4.2 de ISO/IEC. 11172-3 (Smith & Abel, 1993, p. 20) "Semantics for the audio bitstream syntax" define el "frame" para los Layer I, II y III. Un frame en el Layer I contiene 384 muestras de información de audio PMC, mientras que los Layer II y III contienen 1152 muestras de información.

Figura 2–29: Estructura de los marcos de datos de los esquemas Layer I, II y III.



Fuente: Imagen tomada de Bosi & Goldberg (2004, p. 297).

En la Sección 2 ISO/IEC. 11172-3 (Smith & Abel, 1993, p. 2) "Technical Elements", se define la mayor parte de la terminología usada a través de la normativa, en particular, retomamos aquí las definiciones más relevantes a lo largo de todo el documento: la definición de "gránulo", la cual para el Layer II es: Un set de 3 secciones consecutivas de 12 muestras de cada una de las 32 subbandas de las muestras filtradas por los PQMF, y en el Layer III, un gránulo son 576 muestras de las líneas de frecuencia de la MDCT (estas definiciones anteriores se visualizan en la Figura 2–29); la definición de "Bark" es una unidad que representa a una banda crítica, teniendo en cuenta que la "escala Bark" es un mapeo no lineal de la escala de frecuencia del rango de audio correspondiente a la selectividad espectral de la escucha humana; la definición de "scalefactor" es un factor por el cual un set de valores numéricos son escalados antes de la cuantización.

En la seción C.1 de ISO/IEC. 11172-3 (Smith & Abel, 1993, p. 66) "Encoder", describe de forma general cada una de las etapas en cada Layer del MPEG-1. De forma resumida, en la sección C.1.1.1.1 menciona que existen 2 tipos de bancos de filtros: PQMF (banco de filtros polifásico) utilizado en los Layer I y II, y el híbrido, PQMF-MDCT para el Layer III. En la sección C.1.1.1.2 se menciona los 2 Modelos Psicoacústicos disponibles y sus usos, de los cuales se menciona que tanto el Layer I y II aceptan el Modelo Psicoacústico 1 y 2 (el 1 siendo presentado en el Anexo D.1 p. 109 de la normativa y el 2 en el Anexo D.2 p. 128) y el Layer III usa exclusivamente el Mod. Psicoacústico 2 con sus respectivas variaciones descritas en la sección C.1.5.3 p. 80. Todos los modelos psicoacústicos arrojan un arreglo que representa la Relación Señal a Enmascarador (Ecuación (4.30) de la Sección 0 del presente trabajo), y en particular el Mod. Psicoacústico 2, determina el tipo de bloque de procesamiento actual ("Long", "Start", "Short" o "Stop"). Los requisitos procedimentales del Modelo Psicoacústico 2 utilizado para los Layer II y III en el presente trabajo se describen en la Sección 0. p. 82.

El modelo híbrido PQMF-MDCT se describe en la sección C.1.5.3.3 p. 95 de la normativa, junto a los tipos de ventanas mostrados en la Figura 2–14 a la Figura 2–17. El modelo híbrido de transformación inversa IMDCT-IPQMF con las condiciones de Overlap & Add/Save se describe en la sección 2.4.3.4.10 p.36 de la normativa. En la sección C.1.5.4 p. 98 de la normativa se habla modelo de cuantización y de compresión perceptual e informática planteada para las líneas de frecuencia de la MDCT, esto se explica en la sección 4.4 para el Layer III y se muestra la adaptación propuesta para el presente trabajo de este esquema de cuantización para el Layer II en la sección 4.5.

2.7.2 Revisión de la ITU-R BS. 1534-1. "Método para la evaluación subjetiva del nivel de calidad intermedia de los sistemas de codificación"

La recomendación ITU-R BS.1534 hace uso del Método MUSHRA (*Multiple Stimuli with Hidden Reference and Anchor*), el cual está pensado para realizar test a los audios procesados por codecs de audio a través de una evaluación subjetiva, permitiendo cuantificar la calidad percibida de señales que han sufrido un proceso de compresión con pérdidas. Además, es una opción frente al MOS. (*Mean Opinon Score*) y es de preferencia, ya que requiere de un menor número de participantes que permiten lograr una prueba satisfactoria con resultados estadísticos concluyentes. En el método MUSHRA se presenta al oyente un audio de referencia, y un cierto número de audios señuelo, en el que se incluye una versión oculta de la referencia (esto no lo sabe el sujeto de prueba) junto con otros audios bien sean de mayor o menor calidad (estos con el fin de ajustar los valores del oyente a una escala absoluta en el que las puntuaciones bajas representan el audio de baja calidad y viceversa).

Requisitos: La clasificación de prueba consta de dos puntos importantes; uno con base en la capacidad del sujeto para realizar graduaciones repetidas consistentes, el otro con base en inconsistencias de una calificación individual en comparación con el resultado promedio de todas las materias para un ítem dado. Si hay un sujeto que representa una desviación en la

recolección de datos, al no realizar una mala calificación en los audios (de señuelo), este sujeto se debe eliminar de la prueba.

Cantidad de sujetos: Cuando las condiciones de una prueba de audición están estrictamente controladas tanto en el aspecto técnico como en el conductual, la experiencia ha demostrado que los datos de no más de 20 sujetos a menudo son suficientes para sacar conclusiones apropiadas de la prueba.

Prueba y señales de uso: El método de prueba MUSHRA utiliza el material audio original sin procesar con ancho de banda completo como señal de referencia (que también se utiliza como referencia oculta), así como al menos una señal señuelo. En (ITU-R BS.1534-1, 2015, p. 5) recomienda usar una longitud que no exceda los 20s para evitar fatigar a los oyentes y reducir la duración total de la prueba. La selección de las piezas de audio, consta de elegir pistas de audio grabadas en alta calidad sin compresión alguna – para esto se utilizan pistas de canciones almacenadas en calidad de grabación de CD o PCM (WAV o AIFF)-, que han pasado por procesos de captura, grabación y masterización a nivel profesional.

En (ITU-R BS.1534-1, 2015, p. 6) sección 5.1 se describe el conjunto de señales a usar, este consta de todas las señales bajo prueba y al menos una señal adicional (señuelo) que es una versión filtrada con un filtro paso bajo de la señal original. El ancho de banda de esta señal adicional debe ser de 3.5 kHz. Se pueden usar otros tipos de señuelos que muestran tipos similares de impedimentos como los sistemas bajo prueba. Estos tipos de impedimentos pueden incluir:

- Limitación de ancho de banda de 7 kHz o 10 kHz.
- Imagen estéreo reducida.
- Ruido Blanco adicional.
- Pérdidas de paquetes de información.

La característica del filtro de paso bajo con frecuencia de corte en 3.5 kHz debe ser la siguiente:

- Frecuencia de corte igual a 3.5k Hz
- Rizado de banda de paso máximo = ± 0.1 dB
- Atenuación mínima a 4k Hz = 25 dB
- Atenuación mínima a 4.5k Hz = 50 dB.

Los señuelos adicionales están destinados a proporcionar una indicación de cómo los sistemas bajo prueba se comparan con niveles de calidad de audio bien conocidos y no deben usarse para rescatar los resultados entre diferentes pruebas.

Fase de entrenamiento/familiarización: Para obtener resultados confiables, es obligatorio capacitar a los sujetos en sesiones especiales de capacitación antes de la prueba. En (ITU-R BS.1534-1, 2015, p. 6) sección 5.2, se menciona que esta capacitación es importante para obtener resultados confiables. El entrenamiento debe al menos exponer al sujeto a la gama completa y la naturaleza de las deficiencias de los audios y todas las señales de prueba que se experimentarán durante la prueba. Esto se puede lograr utilizando varios métodos: un

sistema de reproducción de cinta simple o un sistema interactivo controlado por computadora. Es en este proceso donde el sujeto interactúa con el entorno de prueba y los dispositivos utilizados, así mismo se adecua al método de evaluación. Durante la fase de apreciación, se hace una explicación oral de las instrucciones de la prueba, teniendo en cuenta que la memoria auditiva a largo y mediano plazo no es fiable, para este motivo se usa un método de computación de resultados instantáneos.

Presentación de estímulos: MUSHRA es un método de prueba de estímulo múltiple "double blind" con referencia oculta y señuelo(s) oculto(s), mientras que la Recomendación UIT-R BS.1116 utiliza un método de prueba de "Triple estímulo ciego con referencia oculta", es en este punto donde se confrontan ambas recomendaciones, usando las valoraciones de la Tabla 2-2.

Tabla 2-2: Tabla de calificaciones para la escala de degradación de cinco notas.

Degradación	Nota
Imperceptible	5
Perceptible, pero no molesta	4
Ligeramente molesta	3
Molesta	2
Muy molesta	1

Fuente: Tabla tomada de BS.1116-1 (1997, p. 4)

2.7.3 Revisión ITU-R BS.1116-1, "Métodos para la evaluación subjetiva de pequeñas deficiencias en sistemas de audio"

El alcance de esta recomendación parte para sistemas de audio de alta calidad con "pequeñas deficiencias. Aceptable en sistemas de audio de calidad baja como sistemas de distribución de internet o *broadcast*, donde hay tasas de transmisión de datos limitadas. Sistemas de calidad media, que incluyen aplicaciones digitales relacionadas a modulación AM, *broadcast* satelital y sistemas de comunicación por línea telefónica.

Diseño experimental: En la evaluación subjetiva de estas pequeñas deficiencias es necesario un método científico formal, caracterizando las pruebas subjetivas a partir de sus condiciones experimentales de control y manipulación, para permitir la cuantificación de datos suministrados por la observación humana. Se hace necesario un diseño experimental cuidadoso, para evitar una contaminación por parte de las variables no controlables y que por esto no se de paso a ambigüedades. En situaciones en las que las deficiencias pueden presentar una distribución de forma homogénea en las pruebas, es necesario generar alteraciones en el proceso de prueba. Una no homogeneidad en la prueba puede representar en factores como la dificultad de esta, por esto se sugiere una distribución no homogénea en la presentación de estímulos. Las pruebas auditivas no deben representar sobrecarga en el sujeto de prueba, ya que da paso a la falta de precisión en la observación de este.

Selección de sujetos: La adquisición de datos debe ser realizada con sujetos de prueba que sean capaces de identificar las deficiencias del sistema de evaluación. Cuanto mayor sea la

calidad alcanzada por los sistemas a probar, más importante es poseer sujetos experimentados.

Criterios en la selección de sujetos: los oyentes deben tener experiencia escuchando el sonido de manera crítica. Tales oyentes darán un resultado más confiable más rápidamente que los oyentes sin experiencia. También es importante tener en cuenta que la mayoría de los oyentes no experimentados tienden a ser más sensibles a los diversos tipos de artefactos después de una exposición frecuente.

Método de prueba: En la evaluación subjetiva de un sistema de reproducción de audio que considera pequeñas degradaciones. Es utilizado un método "sensible" y "estable" que permite identificar dichas degradaciones, este es conocido como "método de triple estimulo doble ciego con referencia oculta". Este consta de la participación de un solo sujeto a la vez, este debe seleccionar entre tres (3) diferentes estímulos ("*a*", "*b*", "*c*"). La referencia conocida como estímulo *a* (el original sin comprimir), la referencia oculta *b* (idéntica a la referencia) y el objeto *c* (como versión procesada) –estas se asignan de forma aleatoria en cada prueba-. El participante debe comparar las degradaciones de *a* respecto a *b* y las de *c* respecto a *a* (teniendo en cuenta la escala de degradación).

3. Capítulo III: Diseño Metodológico

3.1 Tipo y enfoque de la investigación

El enfoque de investigación es de tipo Cuantitativo. El sistema implementado en este trabajo de grado es un procesador digital de señales de audio, encargado de realizar una compresión perceptual, por medio de varios algoritmos que realizan manipulaciones matemáticas a las señales de audio. Se realizaron cálculos/mediciones de diferentes parámetros cuantitativos de la salida de las señales de este procesador de audio, para evaluar el rendimiento de este y lograr comparar los esquemas MPEG-1 Layer II y Layer III. Los parámetros cuantitativos son: Función de Transferencia, Distorsión Armónica Total (THD), Relación Señal a Ruido (SNR) y Densidad de Potencia Espectral (PSD) -Parámetros del Objetivo 3-. De igual forma se aplicó, un enfoque subjetivo/perceptual para realizar el análisis de los datos que arroja el "Double-Blind Test", de "Ensayo multi-estímulo con referencia y patrón oculto" de la recomendación (ITU-R BS.1534-1, 2015) tomado como base de la recomendación (BS.1116-1, 1997) (Objetivo 4) el cual consiste en pruebas subjetivas de escucha crítica. De estas pruebas subjetivas se obtienen resultados estadísticos (enfoque cuantitativo), para determinar la degradación del audio al ser comprimido por ambos métodos de compresión (utilizando Layer II y Layer III).

3.1.1 Recopilación de los datos cuantitativos

Para las mediciones de los parámetros cuantitativos, se optó por utilizar como medio de diseño de señales de entrada y cálculo/procesamiento de los datos de salida del sistema Layer II y III, al software Matlab. Los datos de salida del par de sistemas (Layer II y III) se han tomado directamente del *promt* del entorno de programación Xcode, utilizando la biblioteca estándar de C++ "std" para imprimir en consola los números que representan las muestras de salida procesadas.

3.2 Variables de estudio

Las variables de estudio dependientes, independientes y foráneas a las que el presente trabajo está sujeto se presentan en la Tabla 3-1.

Tabla 3-1: Tabla Variables dependientes, independientes y foráneas.

Variables	
Dependientes	• Pruebas Objetivas: -Relación Señal a Ruido (SNR). -Espectro de Potencia resultante. -Distorsión Armónica Total (THD). -Función de Transferencia de los Esquemas Layer II y III. • Pruebas Subjetivas con base en la Recomendación ITU-1116-1 utilizando el "Doble-Blind Test" de la Recomendación UIT-R BS 1534: - Calificación de cualidad del audio de los sujetos de prueba. - "Experiencia" de los Oyentes.
Independientes	-Transformadas matemáticas: (PQMF, STFT, MDCT) -Flujo de procesamiento (estructura del algoritmo). -Señales de prueba. -Entorno de programación. -Procesador (Computadora y software de ejecución). -Latencia. -Audífonos para las pruebas subjetivas.
Foráneas	-Percepción subjetiva del oyente. -Resultados del semillero de Investigación de Compresión de Audio Digital del año 2017-2.

3.3 Instrumentos

Dado que en ambos esquemas MPEG-1 Layer II y III, la primera etapa de análisis se lleva a cabo a través de los PQMF que se entienden como base para el análisis espectral, y la transformada MDCT que se presenta como un añadido en el esquema Layer III respecto al Layer II. En este sentido, se compararon ambos esquemas de compresión (Layer II y III) al discriminar los efectos introducidos gracias a la MDCT en los resultados del Layer III frente a los resultados netos del Layer II.

Audios o señales de prueba: Utilizando Tonos puros, Barridos sinusoidales (*chirp*) y Ruido blanco se obtienen respectivamente los parámetros de THD, Función de Transferencia y Espectro en Potencia. Todas las señales usadas no deben presentar ningún tipo de compresión (por ejemplo en PCM: WAV o AIFF). Dado que en ISO/IEC. 11172-3 (Smith & Abel, 1993) los sistemas de compresión descritos son parametrizados a las frecuencias de muestreo de 32k, 44.1k y 48k Hz, se decidió medir ambos sistemas (Layer II y III) a dos de estas frecuencias (44.1k y 48k Hz).

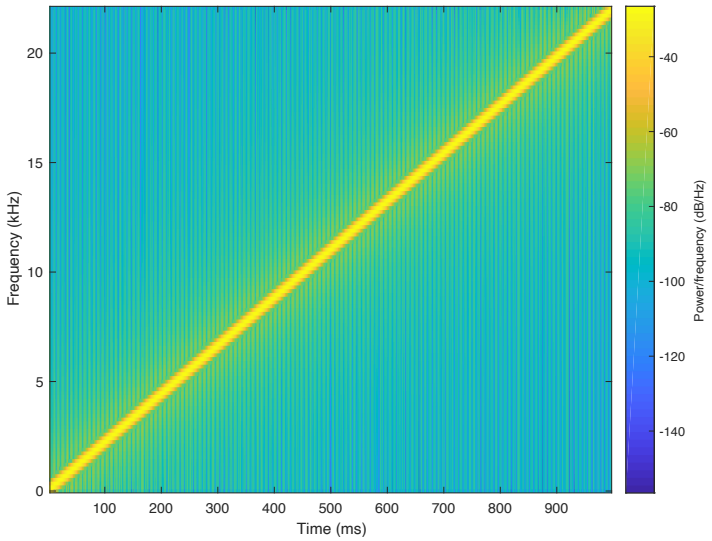
En el software de computo Matlab, se diseñaron los mismos tipos de señales de entrada para las pruebas anteriormente mencionadas a las frecuencias 44.1k y 48k Hz. En el Anexo 7, se presentan los códigos para el diseño de cada una de las señales anteriormente mencionadas, además de los códigos de cómputo para la obtención de los resultados presentados en el Capítulo 5.

THD: Como se define en la normativa 519-2014-IEEE, el THD es la medida de la "Distorsión Armónica Total" presente en una señal al ser excitada por un tono puro, el cual se da como la razón entre la suma de las potencias de los armónicos presentes en los respectivos componentes espectrales de la señal y la potencia de la frecuencia fundamental. Ya que los filtros PQMF son utilizados para el análisis espectral en los esquemas MPEG-1 Layer II y III, se optó por diseñar 32 tonos puros para cada una de las frecuencias correspondientes de los centros de los anchos de banda de cada una de las 32 bandas del PQMF a 44.1k y 48k Hz, es decir, en total 128 tonos puros particulares, de los cuales se halla a cada uno, un resultado de THD. En Matlab se ha usado la función *thd(pxx,f,'psd','aliased')*, donde se especifica la entrada como la estimación de densidad espectral de potencia (PSD) unilateral donde *pxx* es un arreglo de frecuencias correspondiente a las estimaciones del PSD. Y la opción *'aliased'* reporta los armónicos de la fundamental que son producto del aliasing dentro del rango de Nyquist. Se ha seleccionado trabajar con este parámetro dado que Matlab recomienda esta opción cuando la señal de entrada ha sido submuestreada (es decir decimada), tal como sucede en el procedimiento de convolución con los PQMF.

SNR: La Relación Señal a Ruido se define como el rango o proporción en dB que existe entre la potencia de una señal y la potencia del ruido contenido en esta señal. De igual forma que con la THD, se procedió por medir el SNR en cada una de las 32 bandas del PQMF en ambos Layer a las frecuencias de muestreo de 44.1k y 48k Hz. Matlab halla este parámetro al simplemente ingresar una señal en el dominio del tiempo y su frecuencia de muestreo, de la forma *snr(x,fs)*, este encuentra el ruido inherente a la señal de entrada y entrega los resultados en dBFS.

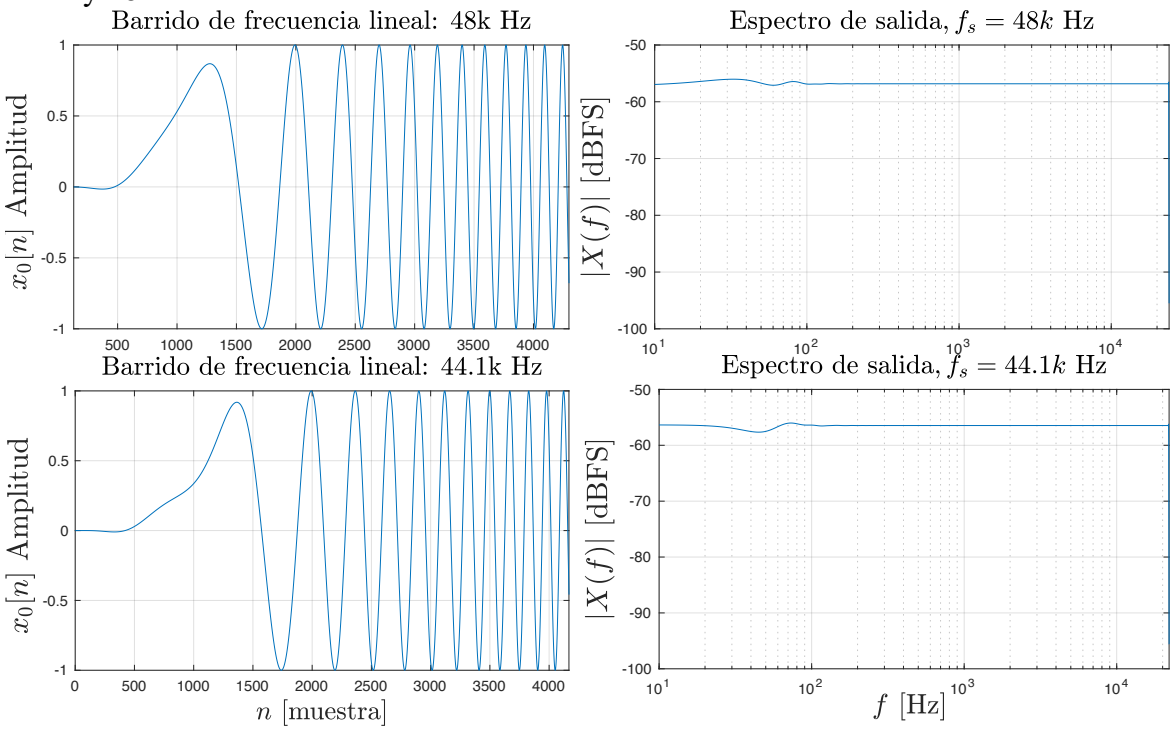
Función de Transferencia $H\left(e^{j\omega}\right)$: Kuo et al. (2006) lo definen como la razón entre el par de Transformadas Z de la señal de salida $y[n] \rightarrow Y(z)$ y entrada $x[n] \rightarrow X(z)$ de un sistema, para el cual es posible usar alguna de las representaciones parciales de la Transformada Z, como lo es la Transformada de Fourier Discreta haciendo $z = e^{j\omega}$. Para hallar $H\left(e^{j\omega}\right)$ en el sistema (Layer II y III), se diseñó todo el rango de frecuencias $e^{j\omega}$ en Matlab en forma de una función *chirp* o barrido de frecuencias (ventaneando los extremos para que el espectro sea plano), donde cada frecuencia adyacente en relación a su posición temporal, está separada de forma lineal, como se muestra en la Figura 3–1. El rango de los barridos de frecuencia usados para las frecuencias de muestreo de 44.1k y 48k Hz va hasta la frecuencia de Nyquist de cada una, y se ingresan al sistema tantas muestras como sean necesarias para que se recorran todas las frecuencias presentes (es decir 44100 y 48000 muestras). El espectro de esta señal y su versión en el dominio del tiempo se ve en la **Figura 3–2**.

Figura 3–1: Espectrograma de función *chirp* lineal de 1 segundo a 44100 Hz.



Fuente: Imagen de elaboración propia usando Matlab_R2017a.

Figura 3–2: Gráficas temporales y espectrales de las funciones chirp usadas en el sistema 44.1k y 48k Hz.



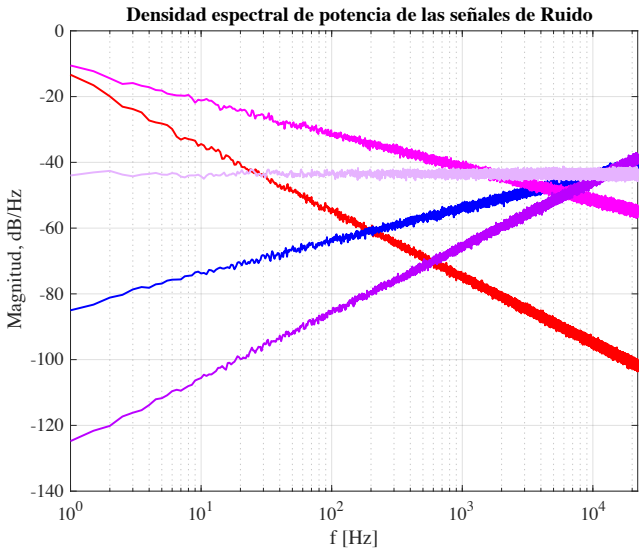
Fuente: Imagen de elaboración propia, usando Matlab_R2017a.

Espectro de potencia (Densidad de potencia espectral, PSD) y Función de Autocorrelación: La función PSD muestra la cantidad de potencia contenida en cada componente espectral de alguna señal, como es descrito por Brown & Hwang (1997), este se usa para caracterizar estadísticamente un sistema.

Para poner a prueba los efectos del sistema sobre la distribución de los valores de una señal aleatoria y ver si genera efectos no lineales sobre esta, o altera la autocorrelación de la señal de entrada, como recomienda Brown & Hwan, (1997) se utilizó ruido blanco gaussiano (WGN), el cual comprende una distribución Gaussiana de los valores de amplitud tomados en cada muestra con su propiedad intrínseca de no poseer correlación estadística. Lo anterior indica que el WGN presenta una PSD discreta plana (para señales continuas la PSD del Ruido Blanco es constante), o sin "coloración" a diferencia del ruido rosa o azul como se ilustra en la Figura 3–3.

La correlación según Brown & Hwang (1997), es el grado en el que un par de variables aleatorias se relacionan de forma lineal, y la autocorrelación o correlación cruzada, no es más que la correlación de una señal con una versión desfasada en el tiempo de sí misma, y permite ver que los valores de correlación cruzada son cero (ya que las muestras no tiene relación estadística una respecto a la otra) a excepción cuando la señal temporal se halla en el tiempo de desfase, resultando en su varianza en ese instante.

Figura 3–3: Densidad espectral de potencia de distintos tipos de señales de ruido de 10 segundos, cada color corresponde al tipo de señal (Ruidos: Rosa -Flicker-, Rojo-Browiano, Azul, Morado y Blanco).



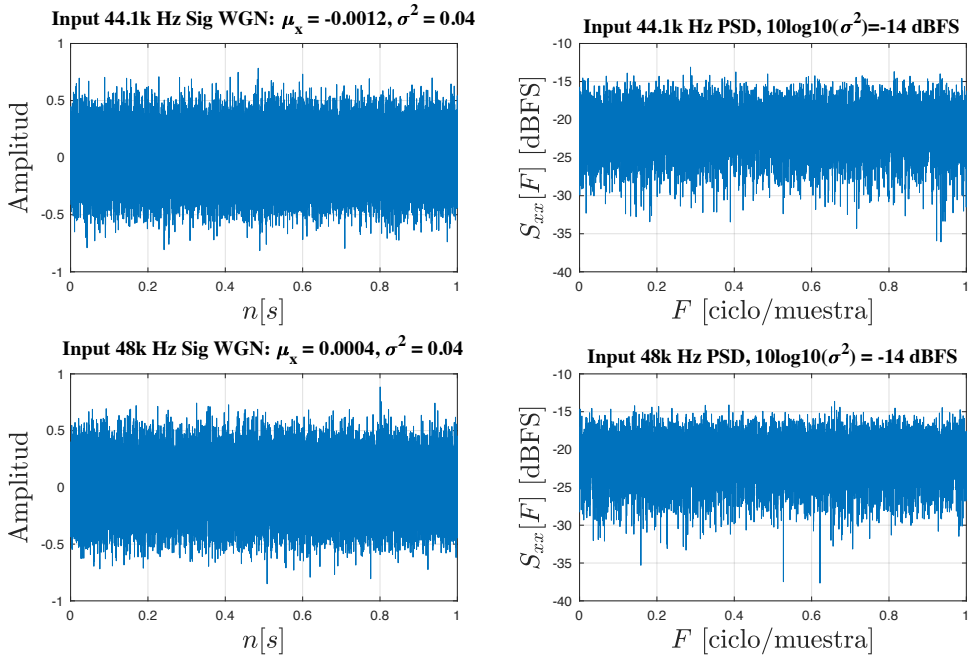
Fuente: Imágenes de elaboración propia, usando Matlab_R2017a.

El diseño del WGN para el presente trabajo se halla en el Anexo 7, teniendo en cuenta que para sistemas discretos, las amplitudes están acotadas entre 1 y -1, se generó una media aritmética $\mu_x = 0$, y una desviación estándar $\sigma = 0.2$, la cual genera que la distribución de amplitudes en la Curva de Normalidad (Campana Gaussiana) sea entorno a 0 y los puntos de inflexión varíen aproximadamente máximo hasta $\pm 5\sigma = \pm 1$. La autocorrelación R_{xx} del WGN es igual a cero sobre todo el dominio del tiempo a excepción del tiempo o retraso τ que es igual a la varianza (σ^2), y su PSD S_{xx} debería ser constante o plana alrededor de la varianza sobre todo el espectro de la señal. La variación de estos parámetros será tomada

como criterio para observar el grado de distorsión que generan ambos sistemas (Layer II y III).

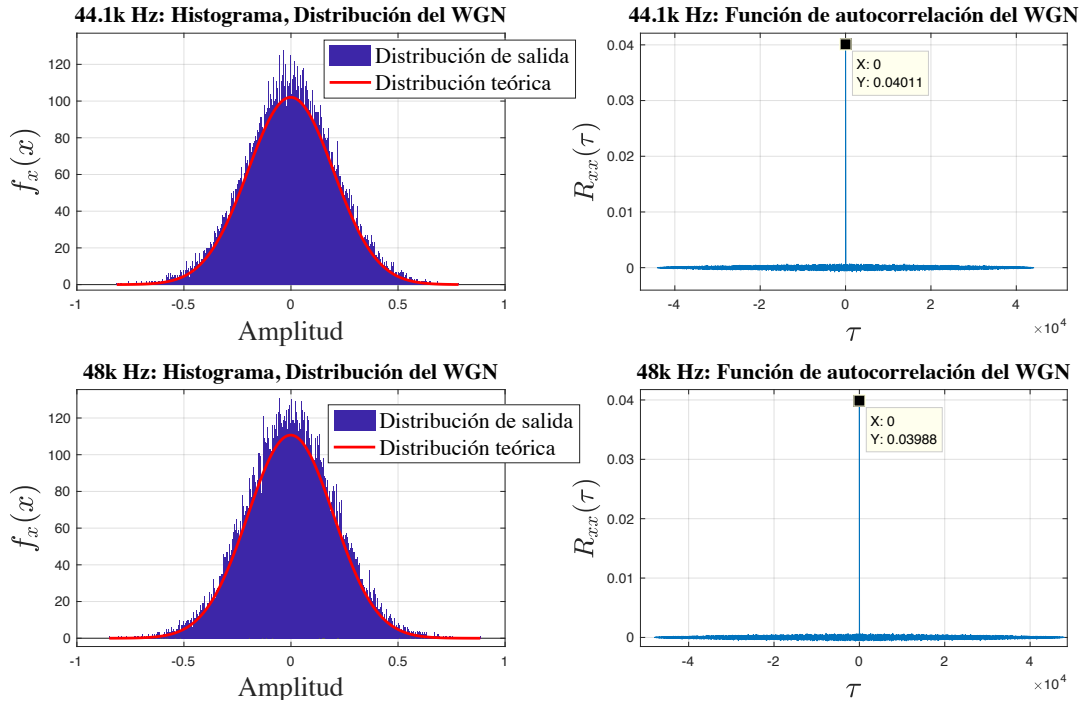
Las características estadísticas anteriormente mencionadas, se muestran en la Figura 3–4 para las señales WGN a 44.1k y 48k Hz. Teniendo como parámetro de entrada las señales en el tiempo WGN se implementaron las funciones de Matlab *mean()* para hallar la media aritmética μ_x y la función *std()* para hallar la desviación estándar σ, ambas mostradas en los títulos de cada gráfico. La autocorrelación R_{xx}, tal como explican Brown & Hwang (1997), tiene varios métodos de cálculo, uno de ellos siendo la convolución de la señal de prueba, la versión reversada y conjugada de sí misma (si la señal es real, claramente no se conjuga como ocurre en este caso. Para llevar a cabo lo anterior se utilizó en Matlab la función *conv(flipud(noise),noise)*. Como se visualiza en la **Figura 3–5**, la autocorrelación $R_{xx}(\tau)$ es igual a σ^2 solo cuando $\tau = 0$, es decir, $R_{xx}(\tau) = \sigma^2\delta(\tau)$.

Figura 3–4: Visualización de las señales WGN usadas en los sistemas Layer II y III para 44.1k y 49k Hz. WGN en el tiempo (gráficas izquierdas) y en la frecuencia (gráficas derechas).



Fuente: Imágenes de elaboración propia, usando Matlab_R2017a.

Figura 3–5: Visualización de las características estadísticas de las señales WGN usadas en los sistemas Layer II y III para 44.1k y 49k Hz. Curvas de normalidad a fs/32 bins (gráficas izquierdas) y Función de autocorrelación (gráficas derechas).



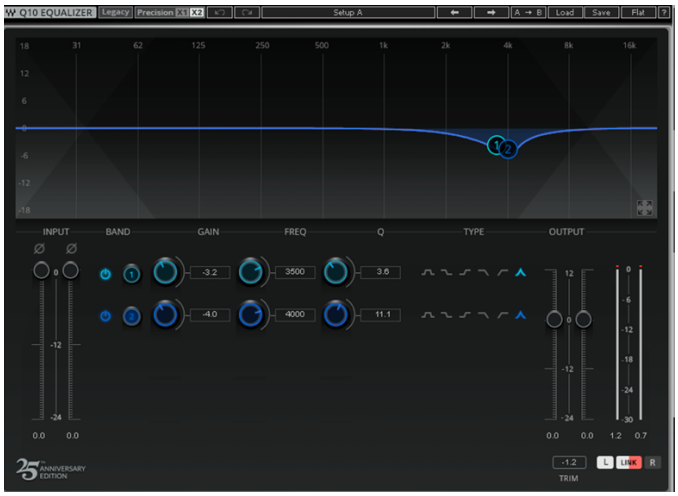
Fuente: Imágenes de elaboración propia, usando Matlab_R2017a.

Test subjetivo, participantes, encuestas: Se hizo uso de la Recomendación ITU-R BS.1534 (Método para la evaluación subjetiva del nivel de calidad intermedia de los sistemas de codificación) como extensión de la recomendación ITU-R BS. 1116-1 para sistemas de codificación de audio (esto se estipula en las consideraciones en (ITU-R BS.1534-1, 2015, p. 1)), con una muestra de mínimo 20 sujetos según la ITU-R BS.1534 (sección 4.2 Tamaño del grupo de participantes y sección 3.3 de (BS.1116-1, 1997, p. 3)). Según lo anterior, se realizaron las pruebas tipo "Double-Blind Test", con una muestra de 25 participantes, que cuentan con un entrenamiento auditivo (musical, técnico o empírico), para el "Ensayo multi-estímulo con referencia y patrón oculto". Esta prueba al arrojar datos de forma estadística se toma como un instrumento cuantitativo, que indica bajo cinco notas calificativas, cinco niveles subjetivos de degradación. Se utilizaron los mismos audios de prueba, designados para comparar el funcionamiento del compresor con ambos esquemas Layer II y Layer III.

En el Anexo 10, se encuentra la plantilla de la encuesta entregada a los participantes de prueba y en el Anexo 11 se encuentran los consentimientos aprobados por cada uno de ellos (25 consentimientos).

En la Sección 2.7.2 del Marco Normativo se explica en detalle el método de prueba a los participantes y las condiciones de diseño de filtros aplicados a las canciones en forma de señuelo para la prueba. A continuación, se muestra el diseño hecho en el DAW Reaper, con el plugin "Q10" de la compañía Waves en la Figura 3–6.

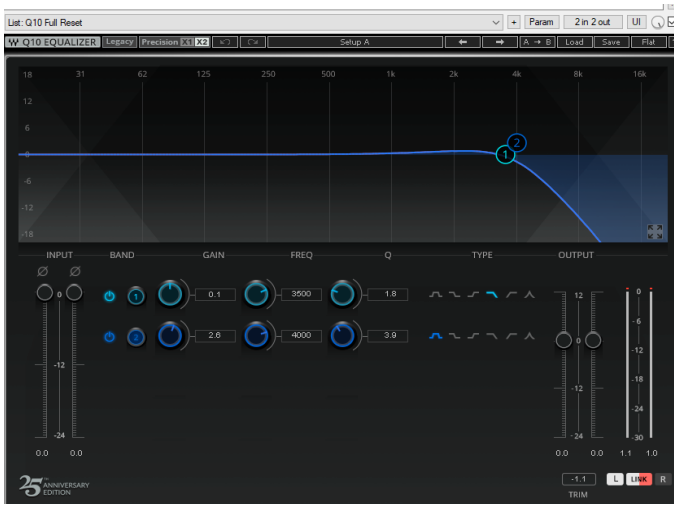
Figura 3–6: Primer filtro tipo campana para los señuelos, hecho con el Ecualizador "Q10" de Waves.



Fuente: Imágenes de elaboración propia (captura de pantalla).

En la Figura 3–6, se muestra el diseño de un filtro tipo campana, con la intención de asemejar su respuesta en frecuencia a la procesada por un filtro pasa bajos, salvo que sus recortes en frecuencia son menos pronunciados utilizando la banda de 3.5k Hz (-3.2 dBFS) y de 4k Hz (-4 dBFS) con factores de calidad/selectividad Q correspondientes a 3.6 y 11.1. Este es utilizado como señuelo al comparar un codec donde la degradación de audio por compresión es menor, es decir para el esquema de compresión Layer III.

Figura 3–7: Segundo filtro tipo pasa bajos para los señuelos, hecho con el Ecualizador "Q10" de Waves.



Fuente: Imágenes de elaboración propia (captura de pantalla).

El segundo filtro señuelo, en la **Figura 3–7**, se asemeja más a la recomendación dada en (BS.1116-1, 1997), para ser utilizado en las muestras de audio en la cuales la sesión utiliza un codec donde la degradación de audio es mas perceptible, es decir para el esquema Layer II.

Para la selección de los temas musicales, para el proceso de evaluación en las pruebas subjetivas, se buscó comparar el desempeño de ambos Layer II y III con señales de características variadas. Teniendo en cuenta que el plugin desarrollado es implementado en procesos de post-producción, se seleccionaron canciones con características específicas en loudness, peak level, estilo musical, interpretación, instrumentación, género.

Caracterizando el primer audio del experimento, se escogió el tema “As I Am” de Dream Theater, en las secciones en las cuales la voz y la guitarra se hacen presentes en mayor manera. Se destaca la presencia de una guitarra distorsionada “crunch”, con fuerte presencia en el rango de 3k a 6kHz visto en la Figura 3–8 utilizando el plugin Voxengo SPAN, esto permitió focalizar la atención en el desempeño del plugin de compresión en este rango de frecuencia.

Figura 3–8: Respuesta en frecuencia de los 20 segundos de la canción “As I Am”.

Fuente: Imágenes de elaboración propia (captura de pantalla).

La segunda canción elegida fue “Solitary Shell” de Dream Theater, con presencia de guitarra acústica y sintetizadores, además de poseer una mezcla bastante balanceada (esto permite hacer una referencia en las adiciones de distorsión armónica del algoritmo de MPEG-1 debido al contenido en alta frecuencia de los armónicos de la guitarra acústica y sintetizador). En la Figura 3–9 de la izquierda se puede apreciar las pérdidas superiores a 16k Hz respecto a la imagen derecha, así como la adición de distorsión representada en clips digitales.

Figura 3–9: Respuesta en frecuencia de los 20 segundos de la canción “Solitary Shell”.

Fuente: Imágenes de elaboración propia (captura de pantalla).

En el tercer, cuarto y quinto de estos temas, “Seven Seas” de Avishai Cohen, “Saeta” de Miles Davis y “Almoraima” de Paco de Lucía, se seleccionaron debido a su carácter percusivo con consonantes de tipo oclusivo, que representan variaciones instantáneas en tiempo representando oportunidades para la detección de ataques en el modelo Psicoacústico 2 del Layer III. Destacando particularmente del quinto tema la presencia de palmas y guitarra percutida, en el que la transición entre estas se presenta repentinamente a lo largo de la selección de audio establecida.

4. Capítulo IV. Desarrollo Ingenieril

En la Sección 4.1, se inicia explicando la estructura general de la programación hecha en el entorno de programación JUCE, nombrando y describiendo los propósitos generales de cada una de las clases implementadas en C++ con sus respectivas instancias, seguido de una descripción detallada del diseño conceptual de la algoritmia del Esquema de compresión Layer III para ser implementando en Tiempo Real en la Sección 4.2, 0 y 4.4, y para el Layer II en la Sección 4.5. En el Anexo 6, se encuentran todos los códigos de los archivos utilizados para el sistema completo.

4.1 Estructura de la Programación en el marco de Aplicación JUCE

El sistema implementado en el marco de aplicación JUCE es escrito e interpretado en el lenguaje de programación C++, este consta de 15 archivos con extensión ".cpp" (el cual contiene las definiciones de los métodos de las clases) y ".h" (que contiene las declaraciones de las clases y sus métodos), aquí se encuentran los frameworks base de procesamiento de audio (entrada, procesamiento y salida de audio) e imagen (Interfaz gráfica de usuario) además de las clases que permiten los "procesamientos internos": PQMF, MDCT, etc.

El framework de procesamiento de audio lleva el nombre de *PluginProcessor*, donde el método *processBlock* contiene al algoritmo de procesamiento que se muestra en la Figura 7–9 del Anexo 4 (en la Figura 7–10 se aplican las clases descritas a continuación), aquí se manipula la entrada de muestras de audio, la forma en que estas muestras entran y sales de los procesamientos internos y el ingreso de estas muestras al buffer de salida. El archivo que contiene las clases y métodos de procesamiento de imagen es llamado *PluginEditor*. En el archivo *PluginProcessor.h* se encuentran las instancias/objetos de cada una de las clases de los procesamientos internos:

```
//Instancias/objetos de las clases externas:
CircularPQMF_Buffer x;
PQMF_Analysis PQMF_processor; float * output_analisis;
IPQMF_Synthesis IPQMF_processor; float * output_sintesis;
s_ProcessBandCoef_MAT s; //para Matriz de 32X36 que alberga las muestras filtradas por PQMF
MDCT_processing MDCT_processor;
//En esta implementación, se aplica el modelo psicoacústico 2 del Layer 3 en el Layer 2:
Psychoacoustic_Mod2_Layer3 Psyco_Layer2; //Psychoacoustic_Mod2_Layer2 Psyco_Layer2;
Psychoacoustic_Mod2_Layer3 Psyco_Layer3;
Quantization_Mod_Layer2 Quant_Layer2;
Quantization_Mod_Layer3 Quant_Layer3;
```

Estas instancias hacen alusión a los siguientes archivos con sus respectivas clases descritas en la **Tabla 4-1** para cada uno de sus respectivos archivos.

Tabla 4-1: Nombre, y descripción de uso de los archivos y clases del sistema implementado.

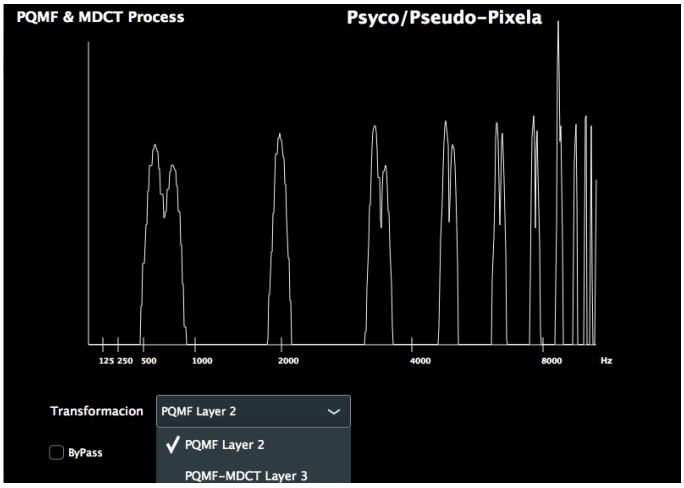
Nombre archivo de declaraciones (".h")	Nombre de las Clases presentes en el archivo	Descripción de uso.	Nombre de la Instancia /Objeto
PQMF_Analysis.h	1) CircularPQMF_Buffer 2) PQMF_Analysis 3) s_ProcessBandCoef_MAT	1) Genera un arreglo dinámico de tamaño N (512), con método de ingreso de muestras en forma circular. Utilizado para albergar las muestras de audio entrantes a los PQMF. Instancia "x". 2) Genera el filtrado PQMF por convoluciones de las muestras de audio albergadas en el arreglo dinámico definido en la clase CircularPQMF_Buffer, para la instancia "x". 3) Genera una matriz dinámica de dimensión 32×36, que organiza las muestras filtradas por banda *k* y tiempo *n.*	1) x 2) PQMF_processor 3) s
MDCT_Processing.h	1) MDCT_Matrix 2) MDCT_Processing	1) Genera una matriz dinámica de dimensión $N \times M$, utilizada internamente en la clase MDCT_Processing para generar matrices de tamaño variable referentes a las matrices explicadas en la sección 4.2, visualizadas también en la Figura 4–4 y Figura 4–5. 2) Genera las transformaciones al dominio de la frecuencia de los datos de la matriz dinámica que se encuentra en el objeto "s", estos procesos se dan también de acuerdo con el tipo de bloque (Long, Short, etc) que se indica en el modelo Psicoacústico de la clase "Psychoacoustic_Mod". También se encuentran aquí los procesos de transformación inversa IMDCT.	1) Ψ, γ_Aliased, γ_Al_second_Half, s_recons 2)MDCT_Processor
Psychoacoustic_Mod.h	1) Psychoacoustic_Mod2_Layer3	1) Realiza el análisis espectral y perceptual del Modelo Psicoacústico 2 de acuerdo a la Sección 0 y la	1) Psyco_Layer2, Psyco_Layer3

	2) Psychoacoustic_Mod2_Layer 2	**Figura 4–6** a la **Figura 4–10** de las muestras de entrada de audio que también recibe la instancia "x". Aquí se define el tipo de bloque, y los umbrales de enmascaramiento mínimos permisibles *xmin* a través del parámetro *SMR* (Relación Señal a Enmascarador). Para la implementación actual, este se utiliza para el Layer 3 y 2 (adaptado para utilizar solamente los SMR). 2) Realiza el análisis espectral y perceptual del Modelo Psicoacústico 2 exclusivo para el Layer 2, esta clase funciona de acuerdo con lo explicado en la normativa ISO/IEC. 11172-3 (Smith & Abel, 1993, p. 128), totalmente funcional para la implementación presente, pero no utilizado en la implementación actual.	
Quantization_Mod.h	1) Quantization_Mod_Layer3 2) Quantization_Mod_Layer2	1) Utilizado en el Layer 3, este realiza el proceso de cuantización y amplificación/distorsión de las muestras espectrales de la MDCT contenidas en las matrices de la instancia "Ψ" a partir de la distorsión permisible *xmin* resultante del modelo Psicoacústico 2. 2) Utilizado en el Layer 2, este realiza el proceso de cuantización y amplificación/distorsión de las muestras temporales de cada banda k de la matriz contenida en la instancia "s" a partir de la distorsión permisible *xmin* resultante del modelo Psicoacústico 2 adaptado para el Layer 2.	1) Quant_Layer3 2) Quant_Layer2
IPQMF_Synthesis.h	1) IPQMF_Synthesis	1) Utilizado en los Layer 2 y 3, realiza los procesos de de-convolución o filtrado inverso de la matriz "s_recons" para el Layer 3, y de la matriz distorsionada "s" para el Layer 2, para la reconstrucción comprimida perceptualmente en el dominio del tiempo como se ilustra en la **Figura 4–3** derecha, de acuerdo con el algoritmo de la **Figura 4–5**, **Tabla 4-2** y **Tabla 4-4**.	1) IPQMF_processor

TablasDatos.cpp		En este archivo están contenidos todos los arreglos y matrices de: -Los coeficientes del filtro base $c_0[n]$ (PQMF) y $d_0[n]$ (IPQMF) -Todos las matrices y arreglos pertinentes halladas en el documento (ISO/IEC. 11172-3, Smith & Abel, 1993) para los modelos psicoacústicos y de cuantización.	

En la Figura 4–1 se presenta la interfaz gráfica de usuario resultante del plugin y aplicación, en esta se encuentra un visualizador de espectro simple en escala lineal de frecuencia, un ComboBox que permite seleccionar entre los esquemas de compresión Layer II y III y una opción de "ByPass" que permite escuchar el audio sin procesamiento.

Figura 4–1: Interfaz gráfica de usuario del plugin y programa en funcionamiento.



Fuente: Imágenes de elaboración propia.

JUCE permite compilar el plugin en distintos formatos (VST, AU, AAX, etc) así como en formato "standalone" el cual es una versión independiente de cualquier DAW, que funciona como un ejecutable.

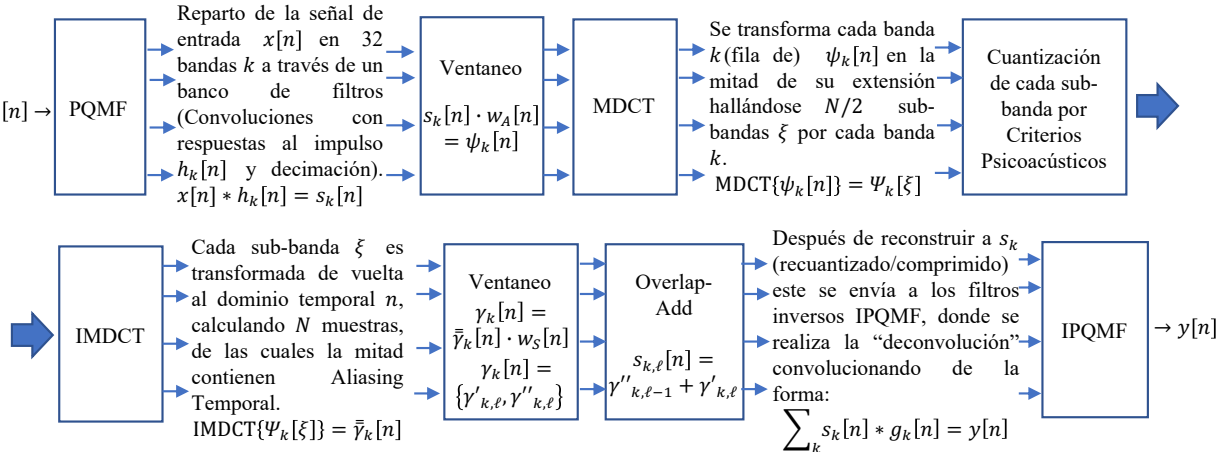
Un método de vital importancia que contiene JUCE contenido en la clase principal en el archivo PluginProcessor, denominado prepareToPlay(), este es llamado cada vez que el número de muestras de entrada al sistema cambia o la frecuencia de muestreo cambia, resultando útil para destruir los arreglos dependientes del número de muestras anterior al actual y poder generar nuevos arreglos adecuados a las dimensiones necesarias. Por ejemplo, para Modelo Psicoacústico 2, las dimensiones de los arreglos cambian dependiendo de la frecuencia de muestreo. En el archivo Psychoacoustic_Mod, en el método fs_select() se envían las ordenes para construir las matrices y arreglos utilizados durante todo el algoritmo del modelo (dependiendo si el sistema está a 32k, 44.1k, o 48k Hz). Es importante recalcar que la implementación actual toma las muestras únicamente del canal número 0 de entrada (canal de entrada activo en un DAW). Actualmente las muestras de este canal 0, procesadas por el sistema, se envían a los n canales de salida estipulados en cualquier DAW. Procesar

varios canales de entrada y enviarlos a distintas salidas requeriría para ser un protocolo óptimo, procesos en paralelo, usando *threads* en C++ para cada bloque ℓ del canal n, o procesar en serie los bloques de entrada de n canales, lo cual genera retrasos en tiempo de $(muestras\ por\ bloque)/f_s[\text{segundos}] + \text{tiempo de otros procesos}$, entre cada bloque, lo cual no es óptimo.

4.2 Esquema algorítmico de Entrada y Salida de datos en Tiempo Real para los filtros PQMF y la Transformada MDCT

Los DAW suelen trabajar a 2^M $(M \in \mathbb{N})$ muestras de entrada, y en la normativa ISO/IEC. 11172-3 (Smith & Abel, 1993) dispuesta para procesadores que no trabajan en tiempo real, se hace la descripción del códec con $2^9 = 512$ muestras de entrada, aparte de una generalización de trabajo con datos a factores de 32ℓ muestras (donde ℓ es el índice del gránulo actual). En la **Figura 4–2** se ilustra el esquema de procesamiento del Layer III.

Figura 4–2: Esquema general del modelo híbrido PQMF-MDCT de compresión y sus inversas.



Fuente: Imagen de elaboración propia.

La primera etapa de transformación/análisis de todo el sistema es a través del banco de filtros PQMF, el cual subdivide N muestras temporales de entrada en $N/32$ muestras distribuidas en 32 bandas (k) de frecuencia (esto a través de decimación para evitar un incremento de 32 veces la tasa de datos). En el esquema izquierdo de la Figura 4–3 (PQMF) se ilustra cómo se toman de a 32 muestras de audio entrantes a un arreglo circular de tamaño de 512, y por cada 32 muestras de audio se entregan 32 datos filtrados (cada uno en una de las 32 bandas), estos son correspondientes a una muestra temporal de cada una de las 32 bandas. Si el proceso anterior se realiza 16 veces, se tendrán 512 datos, donde en cada banda (k) habrán 16 muestra de tiempo. El banco de filtros híbrido compuesto por PQMF-MDCT-IMDCT e IPQMF, indica que las $N/32$ muestras (n) de cada banda (k) que salen del PQMF deben ingresar a la etapa de transformación por MDCT. En la normativa ISO-IEC-11172-3 (1993) se indica que la etapa del MDCT trabaja transformando 36 muestras temporales de la banda k-ésima en 18 muestras espectrales reales (sub-bandas ξ), y por tanto del PQMF se deben

entregar al menos 36 muestras por cada una de las 32 bandas, es decir, un total 1152 (2*576) muestras filtradas y repartidas en 32 bandas.

Un detalle esencial de trabajar con la MDCT es el de reducir a la mitad el número de muestras con las que se trabaja, y esta mitad de nuevo transformarlas al número original de muestras, esto, como se describió en el marco teórico, genera aliasing temporal (secciones de bloques de señales en el tiempo que nunca existieron, pero que al superponerse se reconstruyen en la señal original).

Este proceso de reconstruir N muestras a partir de $N/2$ muestras se explicó en las secciones 2.3.4 y 2.5, y se notificó que para esto se trabaja con un sobrelapamiento del 50% entre bloques sucesivos de datos. Esta idea se sintetiza al revisar de nuevo los conceptos de: 1) Overlap & Add (2.2.2) y 2) Overlap & Save (2.2.3).

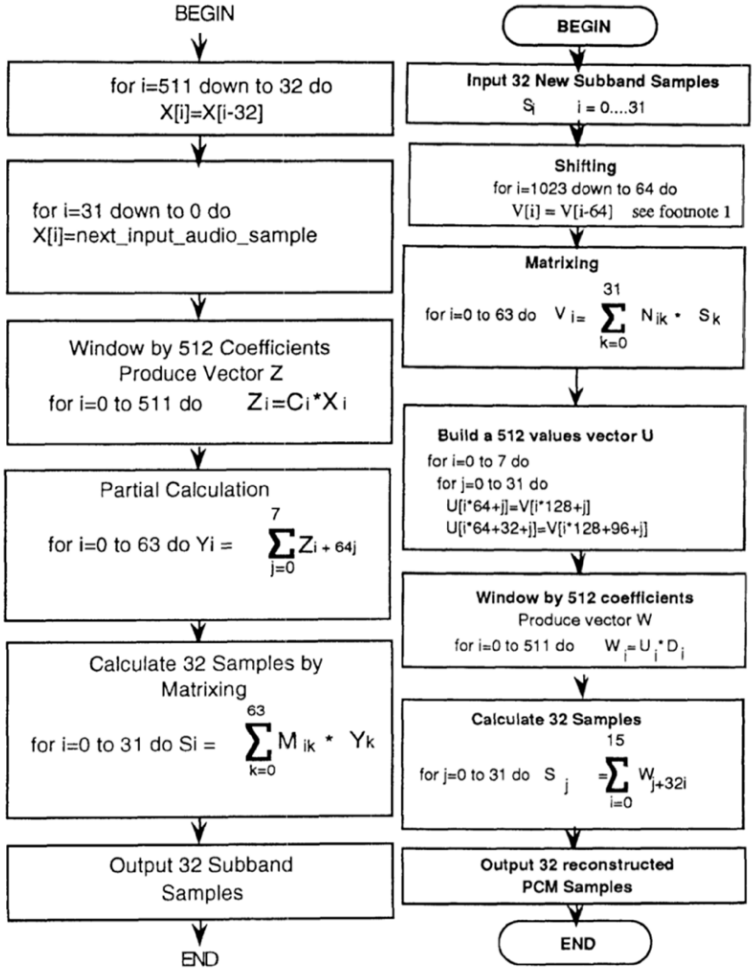
Para la MDCT se aplican estas técnicas de la siguiente forma: Overlap & Save: de bloques consecutivos de N muestras, se toma la mitad final del bloque $\ell - 1$ y la primera mitad del bloque siguiente ℓ, estos se concatenan y se transforman con MDCT, y se realiza este mismo procedimiento indefinidamente: esto quiere decir que se está incrementando la tasa de datos al doble, pero este efecto se elimina en el siguiente paso. Del Overlap & Add: Los bloques ya transformados con MDCT deben superponerse al 50% así como al inicio y sumarse con este sobrelapamiento para eliminar los efectos del aliasing temporal.

A continuación, se hará una descripción de la Figura 4–**4** y la Figura 4–5, las cuales ilustran de forma general, los esquemas de análisis y síntesis con el modelo híbrido (PQMF-MDCT) de transformación para la implementación y diseño del códec en Tiempo Real. Al final de la Sección 2.3 que habla de los filtros PQMF, se muestra que al hacer la convolución $x[n] * h_k[n]$ con una respuesta al impulso $h_k[n]$, donde cada k (de 0 a 31) representa la banda y modulación perteneciente al filtro base $h_0[n]$ (estos filtros en la Figura 2–10 se representan como pasa-bandas de ancho $f_N/32$ [muestras/segundo]), se generan 32 versiones filtradas de $x[n]$ en cada una de estas secciones del espectro.

Al tener 512 muestras de audio de entrada, y al decimar en cada banda, hay 16 muestras de tiempo por banda (fila) como lo enseña la Ecuación (2.40). Suponiendo que los bloques de entrada son a tamaños N, se pueden esperar tantos bloques como sea necesario para trabajar con $32n$ muestras, esto es aprovechable para la exigencia de la MDCT de transformar bloques completos de 1152 muestras de a secciones de 36 muestras, es decir que, si por bloque entran 32 muestras, se esperan 36 bloques.

En general, según la exigencia de la MDCT y la normativa ISO-IEC-11172-3 (1993) (Smith & Abel, 1993, p. 96) siempre cada banda k resultante de los PQMF debe tener 36 muestras de tiempo, así en 32 bandas se tendrán 1152 muestras, que se organizarán en una matriz de 32×36.

Figura 4–3: Algoritmos propuestos en la ISO-IEC-11172-3 (1993) para implementar los bancos de filtros PQMF (esquema izquierdo) e IPQMF (esquema derecho).



Fuente: Imágenes tomadas de la normativa ISO/IEC. 11172-3 (Smith & Abel, 1993, p. 78 y 39).

La Ecuación (4.1) muestra la matriz que debe entrar a la etapa de ventaneo seguido de la transformada MDCT (Figura 4–4 y Figura 4–5). El algoritmo por diseñarse debe siempre esperar tantos bloques para obtener 1152 muestras, o en el caso opuesto, tomar al menos 1152 muestras de dicho bloque.

$$x[n] * h_k[n] = s_k[n] = \begin{bmatrix} s_0[0] & s_0[1] & \cdots & s_0[n] & \cdots & s_0[35] \\ s_1[0] & s_1[1] & \cdots & s_1[n] & \cdots & s_1[35] \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ s_k[0] & s_k[1] & \cdots & s_k[n] & \cdots & s_k[35] \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ s_{31}[0] & s_{31}[1] & \cdots & s_{31}[n] & \cdots & s_{31}[35] \end{bmatrix}_{32\times 36} \tag{4.1}$$

En la Figura 4–**4** se ilustra el diagrama de trabajo del modelo de filtros híbrido, denotando que un gránulo o bloque $\ell - 1$ tiene 576 muestras de entrada en $x_{\ell-1}[n]$, y el bloque siguiente ℓ en $x_\ell[n]$ contiene de igual forma 576 muestras, por lo que para cada gránulo al pasar por los PQMF resulta la convolución: $s_{k,\ell-1}[n] = x_{\ell-1} * h_k$ y $s_{k,\ell}[n] = x_\ell * h_k$, así

se puede concatenar $s_{\ell-1}[n]$ seguido de $s_{\ell}[n]$ ($s_{k,\ell-1}[n] \frown s_{k,\ell}[n]$) y reescribir de forma más fiel a la implementación la Ecuación (4.1) como:

$$s_{k,\ell-1}[n] \frown s_{k,\ell}[n] = \{s_{k,\ell-1}[n], s_{k,\ell}[n]\}$$
$$= \begin{bmatrix} s_{0,\ell-1}[0] & s_{0,\ell-1}[1] & \cdots & s_{0,\ell-1}[17] & s_{0,\ell}[0] & s_{0,\ell}[1] & \cdots & s_{0,\ell}[17] \\ s_{1,\ell-1}[0] & s_{1,\ell-1}[1] & \cdots & s_{1,\ell-1}[17] & s_{1,\ell}[0] & s_{1,\ell}[1] & \cdots & s_{1,\ell}[17] \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ s_{k,\ell-1}[0] & s_{k,\ell-1}[1] & \cdots & s_{k,\ell-1}[17] & s_{k,\ell}[0] & s_{k,\ell}[1] & \cdots & s_{k,\ell}[17] \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ s_{31,\ell-1}[0] & s_{31,\ell-1}[1] & \cdots & s_{31,\ell-1}[17] & s_{31,\ell}[0] & s_{31,\ell}[1] & \cdots & s_{31,\ell}[17] \end{bmatrix}_{32\times 36} \quad (4.2)$$

En la Sección 2.4.1 se mencionó que existen 4 tipos de ventana (iguales para las etapas de análisis y síntesis) compuestas por 36 muestras, además que el producto de cada una de las filas de la Ecuación (4.3) con dichas ventanas de análisis $w_A[n]$ generan la matriz de entrada a la etapa de la MDCT Ecuación (4.3).

$$\psi_{k,\ell}[n] = \left(s_{k,\ell-1}[m] \frown s_{k,\ell}[m]\right) \cdot w_A[n]$$
$$n = 0, \dots, 35, \qquad m = 0, \dots, 17, \qquad k = 0, \dots, 31 \quad (4.3)$$

Se coloca el subíndice $\ell \pm c$ en la matriz/arreglo del gránulo (medio frame) del cual este se genera, por ejemplo, $\gamma_{k,\ell}[n]$ (IMDCT ventaneado con aliasing temporal) fue originado en el gránulo ℓ y está constituido por $s_{k,\ell-1}[n]$ y $s_{k,\ell}[n]$ (donde claramente $s_{k,\ell-1}[n]$ fue obtenido en el gránulo $\ell - 1$ y $s_{k,\ell}[n]$ que fue generado en el gránulo ℓ).

La MDCT transformará las 36 muestras en tiempo de cada banda k, en 18 muestras de frecuencia real, obteniendo en total 576 sub-bandas (a partir de las 1152 muestras de tiempo, mapeadas a 18 líneas de frecuencia por cada una de las 32 bandas k). Usando la expresión analítica (4.4) se esperan 2,304 operaciones (una serie de 36 sumas de 36 productos por cada 32 bandas). Si el modelo psicoacústico determina que se deben usar ventanas tipo short, se toman 3 trozos de cada fila k de $s_{k,\ell}[n]$, para lo cual se ventanean de a 12 muestras y estos se transforman con la MDCT a 12 muestras y se concatenan los resultados de forma sucesiva.

$$\Psi_{k,\ell}[\xi] = \sum_{n=0}^{N-1} \psi_{k,\ell}[n] \cdot \boldsymbol{C}_{A,2} = \begin{bmatrix} \Psi_{0,\ell}[0] & \Psi_{0,\ell}[1] & \cdots & \Psi_{0,\ell}[n] & \cdots & \Psi_{0}[17] \\ \Psi_{1,\ell}[0] & \Psi_{1,\ell}[1] & \cdots & \Psi_{1,\ell}[n] & \cdots & \Psi_{1,\ell}[17] \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \Psi_{k,\ell}[0] & \Psi_{k,\ell}[1] & \cdots & \Psi_{k,\ell}[n] & \cdots & \Psi_{k,\ell}[17] \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \Psi_{31,\ell}[0] & \Psi_{31,\ell}[1] & \cdots & \Psi_{31,\ell}[n] & \cdots & \Psi_{31,\ell}[17] \end{bmatrix}_{32\times 18} \quad (4.4)$$

En la Figura 4–5 se muestra que una vez se tenga la matriz espectral $\Psi_{k,\ell}[\xi]$ (transformada MDCT), se procede a recuantizar cada fila de ésta según el modelo psicoacústico, así se asignan cantidades específicas de bits a cada valor de $\Psi_{k,\ell}[\xi]$ en cada índice de frecuencia ξ para cada fila de la banda k. Y una vez hecha esta compresión perceptual se envía este nuevo $\Psi_{k,\ell}[\xi]$ recuantizado a la transformada inversa IMDCT. La IMDCT computa a $\bar{\bar{\gamma}}_{k,\ell}[n]$

con n al doble de la extensión de ξ (generándose Aliasing Temporal), y este al ventanearse justo con la ventana respectiva de síntesis $w_S[n]$ (igual a la de análisis $w_A[n]$) se produce $\gamma_{k,\ell}[n]$, la cual corresponde a la matriz que por sub-banda k contiene 36 muestras temporales.

$$\gamma_{k,\ell}[n] = w_S[n]\,\bar{\bar{\gamma}}_{k,\ell}[n] = \sum_{\xi=0}^{\frac{N}{2}-1} w_S[n]\,\Psi_{k,\ell}[\xi]\,\boldsymbol{C}_{S,2}$$

$$= \begin{bmatrix} \gamma_{0,\ell}[0] & \gamma_{0,\ell}[1] & \cdots & \gamma_{0,\ell}[n] & \cdots & \gamma_{0,\ell}[35] \\ \gamma_{1,\ell}[0] & \gamma_{1,\ell}[1] & \cdots & \gamma_{1,\ell}[n] & \cdots & \gamma_{1,\ell}[35] \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \gamma_{k,\ell}[0] & \gamma_{k,\ell}[1] & \cdots & \gamma_{k,\ell}[n] & \cdots & \gamma_{k,\ell}[35] \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \gamma_{31,\ell}[0] & \gamma_{31,\ell}[1] & \cdots & \gamma_{31,\ell}[n] & \cdots & \gamma_{31,\ell}[35] \end{bmatrix}_{32\times 36} \tag{4.5}$$

$\gamma_{k,\ell}[n]$, se puede denotar como sus dos mitades concatenadas, siendo $\gamma'_{k,\ell}$ la primera mitad y $\gamma''_{k,\ell}$ la segunda, así $\gamma_{k,\ell}[n] = \{\gamma'_{k,\ell}, \gamma''_{k,\ell}\}$. Para obtener la reconstrucción perfecta de $s_{k,\ell-1}[n]$ a partir de $\gamma_{k,\ell}[n]$, se hace Overlap & Add de la forma:

$$s_{k,\ell-1}[n] = \gamma''_{k,\ell-1} + \gamma'_{k,\ell} \tag{4.6}$$

O

$$s_{k,\ell-1}[n] = \gamma_{k,\ell-1}[m] + \gamma_{k,\ell}[m'], \qquad m = 18,\ldots,35, \qquad n, m' = 0,\ldots,17 \tag{4.7}$$

Donde $\gamma''_{k,\ell-1}$ es la segunda mitad de $\gamma_{k,\ell-1}[n]$ del granulo $\ell-1$, y $\gamma'_{k,\ell}$ es la primera mitad de $\gamma_{k,\ell}[n]$ del gránulo siguiente ℓ. Una vez obtenido $s_{k,\ell-1}[n]$, estos se envían a los filtros inversos IPQMF (para realizar una "de-convolución"), donde es aplicable el algoritmo de la Figura 4–3 (Derecha), para así finalmente tener disponibles 576 muestras comprimidas de audio para la salida $y_\ell[n]$. El mismo proceso se realiza de igual forma para los bloques ℓ y el consecutivo $\ell+1$, tal como se ilustra en la Figura **4–4** y Figura **4–5**.

En la Figura 4–4 y Figura 4–5 se visualiza el procedimiento del modelo híbrido de filtros del sistema para tan solo 2 gránulos. En la Tabla **4-2** y en la Tabla **4-3** se plantea cómo se trabaja con los datos procesados por los filtros PQMF para un caso de 7 gránulos, donde en la **Tabla 4-2** se presenta cómo funciona el sistema de forma general para todas las muestras de 32 k bandas, mientras que en la Tabla 4-3 se presenta cómo funciona el sistema en particular para una sola de las 32 k bandas, en este caso se toma como ejemplo la banda $k = 0$, y de la forma en que se presenta para este ejemplo se debe realizar para todas las demás 31 bandas.

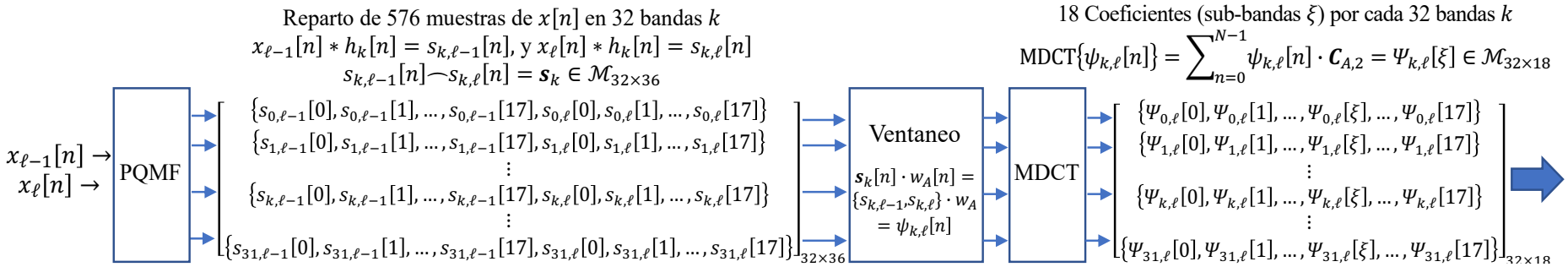
En la Tabla 4-3 para el primer gránulo $\ell = 1$, al no existir muestras de entrada anteriores a éste, a cada fila k de $s_{k,\ell}[n]$ se deben incluir $N/2$ ceros al principio (*zero-padding*) (que se concatenan) para poder ser enviados a la MDCT. Para los gránulos $\ell = 2$ en adelante se toman las 576 muestras $s_{k,\ell-1}[n]$ del gránulo anterior $\ell-1$, esto de por sí (como se mencionó en la sección 2.4.3 y al inicio de esta sección), dobla por bloque/gránulo la tasa de datos, ya que en el gránulo ℓ ingresan 576 muestras y se toman además las 576 muestras

del bloque anterior $\ell - 1$, este efecto se deshace cuando se realiza el Overlap & Add al 50% de la forma ${\gamma''}_{k,\ell-1} + {\gamma'}_{k,\ell}$, de lo cual resulta un arreglo tan solo de 576 muestras nuevamente. Como se hace explícito en las Ecuaciones (4.6) y (4.7) y en la última columna de la Tabla 4-3 ("$s[0, n, \ell][18]$"), eventualmente se consigue la reconstrucción de cualquier $s_{k,\ell}[n]$ a partir de los datos procesados del gránulo siguiente $\ell + 1$, lo cual indica que hay un retraso directo de 1152 muestras (24 ms a 48k muestras/s) en el sistema o de 36 muestras por cada una de las 32 bandas.

Ya se estableció con anterioridad que el tamaño estándar de un bloque o gránulo de audio a procesar es de 576 muestras y un "frame" está constituido por dos de estos gránulos, es decir, cualquier $x_\ell[n]$ tiene un tamaño de 576, el cual va ser filtrado por 32 Respuestas al Impulso $h_k[n]$ generándose el respectivo $s_{k,\ell}[n]$, para 18 líneas de tiempo discreto para cada banda k, dándose la matriz de 576 muestras. En la Tabla **4-4** se ilustran los números exactos de muestras que se toman y procesan suponiendo que el buffer de audio de entrada es de $2^9 = 512$ muestras. Esta diagramación es realizable para cualquier tamaño $c = 2^M$ dado que cualquiera de estos c se puede descomponer en factores de 32 muestras para todo entero $M \geq 5$ y así poder completar cualquier arreglo $x_\ell[n]$ de 576 muestras.

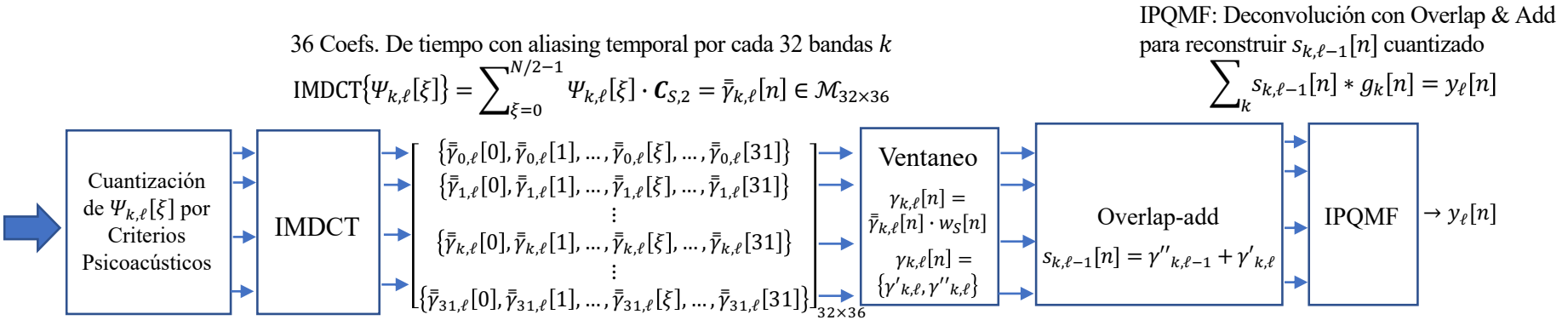
Así en el ejemplo de la Tabla **4-4,** en el gránulo $\ell = 0$, se tienen 512 muestras de entrada, en el siguiente gránulo $\ell = 1$, entran otras 512 muestras, de las cuales solo se requieren las primeras 64 para completar 576 con el buffer anterior, generando a $x_1[n]$ y a $s_{k,1}[n]$ (junto con 576 ceros concatenados en la primera mitad de la matriz), las 448 muestras que sobran servirán junto a las 128 del bloque siguiente $\ell = 2$ para constituir las 576 muestras de $x_2[n]$ y de $s_{k,2}[n]$. Este proceso se hace un número indefinido de veces como se ilustra en la Tabla **4-4**, haciendo de este un proceso iterativo sobre el número de muestras que entran, sobran y faltan para completar a cualquier arreglo buffer de entrada $x_\ell[n]$ y banco de filtros $s_{k,\ell}[n]$ (o igualmente $y_\ell[n]$). En este ejemplo, de un buffer de tamaño de 512 muestras, se requieren 9 bloques de procesamiento para agotar las muestras sobrantes (fila de color **azul**, gránulo $\ell = 8$) y así en el bloque décimo ($\ell = 9$) se reinicia el patrón del número de muestras que entran, sobran y faltan. Este patrón es el mismo para las muestras que salen al ser reconstruidas (parte final izquierda de la tabla vista en $\ell = 2$ y la fila de color **marrón** $\ell = 10$), con un retraso de 1152 muestras.

Figura 4–4: Esquema de análisis genérico del modelo híbrido PQMF-MDCT, para dos gránulos (1 frame) cada uno de tamaño de 576 muestras.



Fuente: Imagen de elaboración propia.

Figura 4–5: Esquema de síntesis genérico del modelo híbrido inverso IMDCT-IPQMF.



Fuente: Imagen de elaboración propia.

Tabla 4-2: Procedimiento post PQMF generalizado de análisis y síntesis del modelo híbrido para 32 bandas k, para 7 bloques de procesamiento.

Granulo ℓ	$s[k,n,\ell]$ 576 muestras (temporales)									Ventaneo de trozos concatenados		$\psi[k,n,\ell]$ 1152 muestras (temporales)	
1	$s[k,n,1]$	ceros(576)⁀	$s[k,n,1]$							$\{\mathbf{0}(576), s[k,n,1]\}\cdot w$	=	$\psi[k,n,1]$	→
2	$s[k,n,2]$		$s[k,n,1]$⁀	$s[k,n,2]$						$\{s[k,n,1], s[k,n,2]\}\cdot w$	=	$\psi[k,n,2]$	→
3	$s[k,n,3]$			$s[k,n,2]$⁀	$s[k,n,3]$					$\{s[k,n,2], s[k,n,3]\}\cdot w$	=	$\psi[k,n,3]$	→
4	$s[k,n,4]$				$s[k,n,3]$⁀	$s[k,n,4]$				$\{s[k,n,3], s[k,n,4]\}\cdot w$	=	$\psi[k,n,4]$	→
5	$s[k,n,5]$					$s[k,n,4]$⁀	$s[k,n,5]$			$\{s[k,n,4], s[k,n,5]\}\cdot w$	=	$\psi[k,n,5]$	→
6	$s[k,n,6]$						$s[k,n,5]$⁀	$s[k,n,6]$		$\{s[k,n,5], s[k,n,6]\}\cdot w$	=	$\psi[k,n,6]$	→
7	$s[k,n,7]$							$s[k,n,6]$⁀	$s[k,n,7]$	$\{s[k,n,6], s[k,n,7]\}\cdot w$	=	$\psi[k,n,7]$	→

Continuación:

			$\Psi[k,n,\ell]$ 576 muestras (espectrales)				$\gamma[k,n,\ell]$ 1152 muestras (temporales)	Overlap & Add									$s[k,n,\ell]$ 576 muestras (temporales)
→	MDCT$\{\psi[k,n,1]\}$	=	$\Psi[k,n,1]$	→	IMDCT$\{\Psi[k,n,1]\}\cdot w$	=	$\gamma[k,n,1]$	$\gamma'[k,n,1]$								=	ceros(576)
→	MDCT$\{\psi[k,n,2]\}$	=	$\Psi[k,n,2]$	→	IMDCT$\{\Psi[k,n,2]\}\cdot w$	=	$\gamma[k,n,2]$	$\gamma''[k,n,1]+$	$\gamma'[k,n,2]$							=	$s[k,n,1]$
→	MDCT$\{\psi[k,n,3]\}$	=	$\Psi[k,n,3]$	→	IMDCT$\{\Psi[k,n,3]\}\cdot w$	=	$\gamma[k,n,3]$		$\gamma''[k,n,2]+$	$\gamma'[k,n,3]$						=	$s[k,n,2]$
→	MDCT$\{\psi[k,n,4]\}$	=	$\Psi[k,n,4]$	→	IMDCT$\{\Psi[k,n,4]\}\cdot w$	=	$\gamma[k,n,4]$			$\gamma''[k,n,3]+$	$\gamma'[k,n,4]$					=	$s[k,n,3]$
→	MDCT$\{\psi[k,n,5]\}$	=	$\Psi[k,n,5]$	→	IMDCT$\{\Psi[k,n,5]\}\cdot w$	=	$\gamma[k,n,5]$				$\gamma''[k,n,4]+$	$\gamma'[k,n,5]$				=	$s[k,n,4]$
→	MDCT$\{\psi[k,n,6]\}$	=	$\Psi[k,n,6]$	→	IMDCT$\{\Psi[k,n,6]\}\cdot w$	=	$\gamma[k,n,6]$					$\gamma''[k,n,5]+$	$\gamma'[k,n,6]$			=	$s[k,n,5]$
→	MDCT$\{\psi[k,n,7]\}$	=	$\Psi[k,n,7]$	→	IMDCT$\{\Psi[k,n,7]\}\cdot w$	=	$\gamma[k,n,7]$						$\gamma''[k,n,6]+$	$\gamma'[k,n,7]$		=	$s[k,n,6]$

Tabla 4-3: Caso del procedimiento post PQMF de análisis y síntesis del modelo híbrido para la banda $k = 0$, para 7 bloques de procesamiento.

Granulo ℓ	$s[0,n,\ell]$ [18]									Ventaneo de trozos concatenados		$\psi[0,n,\ell][36]$	
1	$s[0,n,1]$	ceros(18)⁀	$s[0,n,1]$							$\{\mathbf{0}(18), s[0,n,1]\}\cdot w$	=	$\psi[0,n,1]$	→
2	$s[0,n,2]$		$s[0,n,1]$⁀	$s[0,n,2]$						$\{s[0,n,1], s[0,n,2]\}\cdot w$	=	$\psi[0,n,2]$	→
3	$s[0,n,3]$			$s[0,n,2]$⁀	$s[0,n,3]$					$\{s[0,n,2], s[0,n,3]\}\cdot w$	=	$\psi[0,n,3]$	→
4	$s[0,n,4]$				$s[0,n,3]$⁀	$s[0,n,4]$				$\{s[0,n,3], s[0,n,4]\}\cdot w$	=	$\psi[0,n,4]$	→
5	$s[0,n,5]$					$s[k,n,4]$⁀	$s[0,n,5]$			$\{s[0,n,4], s[0,n,5]\}\cdot w$	=	$\psi[0,n,5]$	→
6	$s[0,n,6]$						$s[k,n,5]$⁀	$s[k,n,6]$		$\{s[0,n,5], s[0,n,6]\}\cdot w$	=	$\psi[0,n,6]$	→
7	$s[0,n,7]$							$s[k,n,6]$⁀	$s[k,n,7]$	$\{s[0,n,6], s[0,n,7]\}\cdot w$	=	$\psi[0,n,7]$	→

Continuación:

			$\Psi[0,n,\ell]$ [18]				$\gamma[k,n,\ell][36]$	Overlap & Add								$s[0,n,\ell][18]$
→	MDCT$\{\psi[0,n,1]\}$	=	$\Psi[0,n,1]$	→	IMDCT$\{\Psi[0,n,1]\}\cdot w$	=	$\gamma[0,n,1]$	$\gamma'[0,n,1]$							=	ceros(18)
→	MDCT$\{\psi[0,n,2]\}$	=	$\Psi[0,n,2]$	→	IMDCT$\{\Psi[0,n,2]\}\cdot w$	=	$\gamma[0,n,2]$	$\gamma''[0,n,1]+$	$\gamma'[0,n,2]$						=	$s[0,n,1]$
→	MDCT$\{\psi[0,n,3]\}$	=	$\Psi[0,n,3]$	→	IMDCT$\{\Psi[0,n,3]\}\cdot w$	=	$\gamma[0,n,3]$		$\gamma''[0,n,2]+$	$\gamma'[0,n,3]$					=	$s[0,n,2]$
→	MDCT$\{\psi[0,n,4]\}$	=	$\Psi[0,n,4]$	→	IMDCT$\{\Psi[0,n,4]\}\cdot w$	=	$\gamma[0,n,4]$			$\gamma''[0,n,3]+$	$\gamma'[0,n,4]$				=	$s[0,n,3]$
→	MDCT$\{\psi[0,n,5]\}$	=	$\Psi[0,n,5]$	→	IMDCT$\{\Psi[0,n,5]\}\cdot w$	=	$\gamma[0,n,5]$				$\gamma''[0,n,4]+$	$\gamma'[0,n,5]$			=	$s[0,n,4]$
→	MDCT$\{\psi[0,n,6]\}$	=	$\Psi[0,n,6]$	→	IMDCT$\{\Psi[0,n,6]\}\cdot w$	=	$\gamma[0,n,6]$					$\gamma''[0,n,5]+$	$\gamma'[0,n,6]$		=	$s[0,n,5]$
→	MDCT$\{\psi[0,n,7]\}$	=	$\Psi[0,n,7]$	→	IMDCT$\{\Psi[0,n,7]\}\cdot w$	=	$\gamma[0,n,7]$						$\gamma''[0,n,6]+$	$\gamma'[0,n,7]$	=	$s[0,n,6]$

Tabla 4-4: Visualización del número de muestras de entrada y salida para el procesamiento pre, durante y post-MDCT para gránulos de 512 muestras. (Se lee de derecha a izquierda la consecución de procesamientos -leer la dirección de las flechas-)

←	Entradas para MDCT		1152 Muestras	$s[k,\ell][576]$	**s[k,1]**	**s[k,2]**	**s[k,3]**	**s[k,4]**	**s[k,5]**	**s[k,6]**	**s[k,7]**	**s[k,8]**	**s[k,9]**	**s[k,10]**	**s[k,11]**	**s[k,12]**	
←	$\psi[k,\ell] = \{s[k,\ell-1], s[k,\ell]\} \cdot w$		Concatenación de bloques sucesivos $s[k,\ell-1] \frown s[k,\ell]$	Bloque/ Granulo $\boldsymbol{\ell}$	576	576	576	576	576	576	576	576	576	576	576	576	Entrada de Muestras Audio [512]
	-		-	**0**	512												512
←	ψ[k,1]	=	ceros(576)⁀s[k,1]	**1**	+64	448											512
←	ψ[k,2]	=	s[k,1]⁀s[k,2]	**2**		+128	384										512
←	ψ[k,3]	=	s[k,2]⁀s[k,3]	**3**			+192	320									512
←	ψ[k,4]	=	s[k,3]⁀s[k,4]	**4**				+256	256								512
←	ψ[k,5]	=	s[k,4]⁀s[k,5]	**5**					+320	192							512
←	ψ[k,6]	=	s[k,5]⁀s[k,6]	**6**						+384	128						512
←	ψ[k,7]	=	s[k,6]⁀[k,7]	**7**							+448	64					512
←	ψ[k,8]	=	s[k,7]⁀s[k,8]	**8**								+512	0				512
←	ψ[k,9]	=	s[k,8]⁀s[k,9]	**9**									+512				512
←	ψ[k,10]	=	s[k,9]⁀s[k,10]	**10**									+64	448			512
←	ψ[k,11]	=	s[k,10]⁀s[k,11]	**11**										+128	+384		512
←	ψ[k,12]	=	s[k,11]⁀s[k,12]	**12**											+192	320	512
←	ψ[k,13]	=	s[k,12]⁀s[k,13]	**13**												+256	256

Continuación:

	Número muestras de salida hacia IPQMF, e igualmente post-PQMF/pre-MDCT											Overlap-add, 576 muestras reconstruidas listas para IPQMF			Salidas de la IMDCT, 1152 muestras ventaneadas con Aliasing Temporal	←	salidas de la MDCT [1152]	←
$\boldsymbol{\ell}$	$\boldsymbol{s}_{k,10}$	$\boldsymbol{s}_{k,9}$	$\boldsymbol{s}_{k,8}$	$\boldsymbol{s}_{k,7}$	$\boldsymbol{s}_{k,6}$	$\boldsymbol{s}_{k,5}$	$\boldsymbol{s}_{k,4}$	$\boldsymbol{s}_{k,3}$	$\boldsymbol{s}_{k,2}$	$\boldsymbol{s}_{k,1}$	576 ceros	$s[k,\ell] = \gamma''[k,\ell-1] + \gamma'[k,\ell]$			$\gamma[k,\ell] = \bar{\gamma}[k,\ell]w = \text{IMDCT}\{\Psi[k,\ell]\}w = \{\gamma'[k,\ell], \gamma''[k,\ell]\} \cdot w$	←	$\Psi[k,\ell] = \text{MDCT}\{\psi[k,\ell]\} = \{\Psi'[k,\ell], \Psi''[k,\ell]\}$	←
0												-			-		-	
1											512	ceros(576)	=	γ'[k,1]	γ[k,1]	←	Ψ[k,1]	←
2										448	64	**s[k,1]**	=	γ"[k,1]+γ'[k,2]	γ[k,2]	←	Ψ[k,2]	←
3									384	128		**s[k,2]**	=	γ"[k,2]+γ'[k,3]	γ[k,3]	←	Ψ[k,3]	←
4								320	192			**s[k,3]**	=	γ"[k,3]+γ'[k,4]	γ[k,4]	←	Ψ[k,4]	←
5							256	256				**s[k,4]**	=	γ"[k,4]+γ'[k,5]	γ[k,5]	←	Ψ[k,5]	←
6						192	320					**s[k,5]**	=	γ"[k,5]+γ'[k,6]	γ[k,6]	←	Ψ[k,6]	←
7					128	384						**s[k,6]**	=	γ"[k,6]+γ'[k,7]	γ[k,7]	←	Ψ[k,7]	←
8				64	448							**s[k,7]**	=	γ"[k,7]+γ'[k,8]	γ[k,8]	←	Ψ[k,8]	←
9			0	512								**s[k,8]**	=	γ"[k,8]+γ'[k,9]	γ[k,9]	←	Ψ[k,9]	←
10			512									**s[k,9]**	=	γ"[k,9]+γ'[k,10]	γ[k,10]	←	Ψ[k,10]	←
11		448	64									**s[k,10]**	=	γ"[k,10]+γ'[k,11]	γ[k,11]	←	Ψ[k,11]	←
12	384	128										**s[k,11]**	=	γ"[k,11]+γ'[k,12]	γ[k,12]	←	Ψ[k,12]	←
13	192											**s[k,12]**	=	γ"[k,12]+γ'[k,13]	γ[k,13]	←	Ψ[k,13]	←

Fuente. Elaboración propia.

4.2.1 Código principal para el procesamiento en TR. del Layer III.

```
if (bool_preBlock) { #3 if
    for (int i = 0; i < i_end; i++) (*)
        auxArray[i] = recons_signal[i + ips];
    ips += i_end;
    if (ips == 576) #4 if
       ips = 0;
}
for (int blockIndex = 0; blockIndex < buffer.getNumSamples()/32; blockIndex++){ (**)
//Indexación de 32 muestras en el buffer circular. Ingresan 32 muestras por cada iteración de blockIndex.
    for (int i = 0; i < 32; i++){
        x.addSample(*buffer.getReadPointer(0,i + blockIndex*32));
        Psyco_Layer3.ingresoMuestraFifo((*buffer.getReadPointer(0,i + blockIndex*32));
    }
//Procesamiento del buffer circular con el PQMF e indexación de datos en la matriz de 32X36.
    output_analisis = PQMF_processor.PQMF_Filtering(x.buffer);
    s.add_Data(output_analisis);
    //apenas se llene la matriz s de 576 muestras, 32datos*18veces:
    if (globalBlockIndex == 17) { #5 if
        Psyco_Layer3.run_Model();
        MDCT_processor.set_blockType(Psyco_Layer3.block_type,Psyco_Layer3.past_block_type);
        MDCT_processor.ventaneo_preMDCT(s.bufferMat); MDCT_processor.direct_MDCT();
        Quant_Layer3.set_blockType(Psyco_Layer3.block_type, Psyco_Layer3.past_block_type);
        Quant_Layer3.inputMats(MDCT_processor.Ψ.matriz,
        MDCT_processor.Ψ_short,Psyco_Layer3.xmin_L,Psyco_Layer3.xmin_S);
        Quant_Layer3.resetVariables(); Quant_Layer3.iterationLoop();
        MDCT_processor.inverse_MDCT(); MDCT_processor.overlap_Add();
        float s_k[32];
        for (int n = 0; n < 18; n++) {
        //s_k se llena de 32 subbandas por cada índice de tiempo n
            for (int k = 0; k < 32; k++)
                s_k[k] = MDCT_processor.s_recons.matriz[k][n];
        //32 subbandas se de-convolucionan
            output_sintesis = IPQMF_processor.IPQMF_Filtering(s_k);
            for (int k = 0; k < 32; k++) //paso de matriz a arreglo unidimensional
                recons_signal[k + n*32] = *(output_sintesis + k);
         }
         interpol_recons(recons_signal); //corrección de muestras por promediación
         MDCT_processor.cop_half_Alias_Recons();
         s.circularData();
         Psyco_Layer3.copyOutputs();
         Psyco_Layer3.past_block_type = Psyco_Layer3.block_type;
    }
    globalBlockIndex++; globalBlockIndex %= 18;
}
if (bool_postBlock) { #1 if
    for (int j = 0; j < j_end; j++) (***)
        auxArray[j + i_end] = recons_signal[j];
    //corrección por promediación. cada 576 muestras
    auxArray[i_end-1] = 0.5*(auxArray[i_end - 2] + recons_signal[0]);
    ips = j_end;
}
//se llena el buffer de salida
for (int channel = 0; totalNumInputChannels < totalNumOutputChannels; ++channel){
    auto* channelData = buffer.getWritePointer(channel);
    for(int n = 0; n < buffer.getNumSamples(); n++) (****)
        channelData[n] = auxArray[n];
}
if ((buffer.getNumSamples() + ips) < 576) { #2 if
    i_end = buffer.getNumSamples();
    bool_preBlock = true; bool_postBlock = false;
}else{
    bool_preBlock = true; bool_postBlock = true;
    i_end = 576 - ips;
    j_end = buffer.getNumSamples() - i_end;
}
```

El código anterior, hallado en el archivo PluginProcessor.cpp del plugin en el método processBlock(), es la base fundamental de todo el procesamiento de datos de entrada y salida del sistema. En los condicionales *if* con los numerales en rojo (#1, 2, 3 y 4) se encuentra la algoritmia que permite asignar al buffer de salida por secciones (si es necesario), partes del arreglo recons_signal[] que contiene 576 muestras reconstruidas en tiempo comprimidas

perceptualmente. Es decir, la estructura de salida de datos (para cualquier buffer de entrada de tamaño 2^N muestras -solo los valores $N = 4,5,6,7,8,9$-) ejemplificada para un buffer de tamaño 512 en la Tabla 4-4, se logra a través del contenido de estos condicionales *if*.

El arreglo recons_signal[] (tamaño de 576) contiene a la señal de salida totalmente procesada $y_\ell[n]$ (tamaño de 576) del gránulo ℓ-ésimo (es decir, recons_signal[] $= y_\ell[n]$). Para la siguiente tabla (Tabla **4-5** y Tablas del Anexo 5) se aprovecha la notación del gránulo ℓ-ésimo, para mostrar de qué gránulo ℓ de procesamiento se obtienen los datos de salida en el arreglo auxArray[] que es el encargado de enviar las muestras de $y_\ell[n]$ al buffer de salida en los respectivos bucles *for* marcados con **(*)** y **(***)**. Es importante mencionar que el bucle *for* **(***)** se utiliza exclusivamente cuando todas las muestras de $y_{\ell-1}$ se han agotado en el *for* **(*)** y aún hace falta llenar el buffer de salida, por lo cual se hace uso de y_ℓ, es decir, el bucle *for* **(***)** se utiliza para terminar de llenar el buffer de salida.

Las variables de control denominadas i_end, ips y j_end, se encargan de definir los límites de los bucles *for* **(*)** y **(***)** para asignar un número específico de muestras de iteración. La variable booleana bool_preBlock hace referencia al uso del *for* **(*)** en el **#1 if** y la otra variable booleana bool_postBlock al *for* **(***)** en el **#3 if.** Ya que es imposible asignar muestras de salida antes de que existan (asignación en primera instancia de muestras en el *for* **(*)**), las variables bool_preBlock y bool_postBlock se inicializan respectivamente en $false$ y $true$, la variable j_end a el tamaño del buffer de salida, j_end e ips a 0, de forma que el primer *for* en utilizarse sea **(***)**. Dado que recons_signal[] contiene 576 muestras reconstruidas, siempre se cumple que para buffers de entrada y salida de tamaños enteros menores a $2^9 = 512$, en el *for* **(***)** en su primer uso, no se agotan las muestras de recons_signal[] en la asignación de salida, y siempre entrando este **#1 if** se hace igual a la variable ips al valor máximo al cual iteró el *for* **(***)**, es decir $ips = j\_end$. El **#2 if** se encarga de revisar que no se hallan agotado las 576 muestras de recons_signal[] en su asignación con el buffer de salida, revisando la desigualdad ((tamañoDelBuffer_I/O + ips) < 576, asignando los valores lógicos de bool_preBlock y bool_postBlock para los respectivos *for* **(*)** y **(***)**. En la Figura 7–9 del Anexo 4, se ilustra la lógica del código principal de arriba.

El bucle *for* **(**)** es el encargado de repartir en factores de 32 las muestras de entrada. Las cuales hacen parte de la propuesta algorítmica introducida por la normativa en el banco de filtros PQMF. Se introducen de a 32 muestras al buffer circular usado para los PQMF y el Modelo Psicoacústico 2. El **#5 if** es vital en el flujo del procesamiento ya que permite las transformaciones en frecuencia de la MDCT, IMDCT y IPQMF, a demás del uso del modelo psicoacústico y la compresión perceptual, esto solo ocurre cuando el bucle *for* **(**)** haya iterado 18 veces, es decir, se hallan obtenido 576 muestras de los filtros PQMF. Finalmente, las muestras reconstruidas de auxArray[] se envían al buffer de salida en el *for* **(****)** para n canales de salida definidos en la variable totalNumOutputChannels.

En el Anexo 4, se encuentra el diagrama de flujo del Código Principal para el Layer III mostrado al inicio de esta sección, estos diagramas de flujo para el Layer II son semejantes a los utilizados, omitiendo el uso de las instancias de la MDCT, y el respectivo modelo

psicoacústico y de cuantización. La versión exacta para el Layer II se encuentra en el Anexo 6 en el archivo pluginProcessor.cpp.

Tabla 4-5: Flujo de la lógica de asignación de muestras de salida, para un buffer de salida de 256 muestras.

		(∗), #3 if bool_preBlock		(∗∗∗), #1 if bool_postBlock
i_end, ips, j_end	$y_\ell \rightarrow$	for (i = 0; i < i_end; i++) auxArray[i]=recons_signal[i+ips]	$y_\ell \rightarrow$	for (j = 0; j < j_end; j++) auxArray[j+i_end]=recons_signal[j]
0,0,256			$y_0 \rightarrow$	for (i = 0; i < 256) auxArray[i+0]=recons_signal[i]
256,256,null	$y_0 \rightarrow$	for (i = 0; i < 256) auxArray[i]=recons_signal[i+256]		
64,512,192	$y_0 \rightarrow$	for (i = 0; i < 64) auxArray[i]=recons_signal[i+512]	$y_1 \rightarrow$	for (j = 0; j < 192; j++) auxArray[j+64]=recons_signal[j]
256,192,null	$y_1 \rightarrow$	for (i = 0; i < 256) auxArray[i]=recons_signal[i+192]		
128,448,128	$y_1 \rightarrow$	for (i = 0; i < 128) auxArray[i]=recons_signal[i+448]	$y_2 \rightarrow$	for (j = 0; j < 128; j++) auxArray[j+128]=recons_signal[j]
256,128,null	$y_2 \rightarrow$	for (i = 0; i < 256) auxArray[i]=recons_signal[i+128]		
192,384,64	$y_2 \rightarrow$	for (i = 0; i < 192) auxArray[i]=recons_signal[i+384]	$y_3 \rightarrow$	for (j = 0; j < 64; j++) auxArray[j+192]=recons_signal[j]
256,64,null	$y_3 \rightarrow$	for (i = 0; i < 256) auxArray[i]=recons_signal[i+64]		
256,320,null	$y_3 \rightarrow$	for (i = 0; i < 256) auxArray[i]=recons_signal[i+320]		
0,0,256		Se repite el mismo patrón de arriba…	$y_4 \rightarrow$	for (i = 0; i < 256) auxArray[i+0]=recons_signal[i]

Fuente. Elaboración propia.

4.3 Marco de análisis espectral provisto por el Modelo Psicoacústico 2 para el Layer II y III

En paralelo a la implementación del banco de filtros híbrido PQMF-MDCT, se realiza un análisis psicoacústico descrito en las secciones D.2 “Psychoacoustic Model 2” cuyas variaciones para el Layer III se explican en la sección C.1.5.3 “Layer III Encoding” de la normativa ISO/IEC. 11172 (Smith & Abel, 1993, p. 80 y 128), esto con el fin de moldear y recuantizar las líneas espectrales de la MDCT y determinar el tipo de bloque de presente (“Long”, “Short”, etc.) teniendo en cuenta el modelado de enmascaramiento frecuencial que plantea el Modelo Psicoacústico 2. Este modelo en el Layer III tiene como propósito en particular, discriminar el tipo de señal en los gránulos actuales a través de un parámetro cuantitativo llamado “Entropía Perceptual” (PE) para así evitar el artefacto de compresión “Pre-Echo” para lo cual, según este parámetro PE, se cambia la longitud y tipo de ventana a usarse para la MDCT directa e inversa. El modelo psicoacústico con el propósito de re-escalar (moldear) las líneas de la MDCT, provee para cada *factor de escala* un coeficiente de relación señal a enmascarador ($SMR[sb]$ siendo este arreglo, la salida final de cómputo del Modelo Psicoacústico 2).

En el esquema del Layer III, se trabaja con un buffer de entrada $x_{\ell,l}[n]$ de tamaño 576 (el cual contiene las mismas muestras que entran al banco de filtros híbrido), donde ℓ es el índice del gránulo general, y l $(0 \leq l \leq 1)$ es el gránulo particular en el modelo psicoacústico, del cual solo considera el gránulo actual ℓ y el pasado $\ell - 1$. Una vez el modelo sea capaz de calcular la Entropía Perceptual, el algoritmo discrimina si el bloque entrante $x_{\ell,l}[n]$ posee cambios repentinos en el dominio del tiempo, y si es así, este se reparte en 3 bloques tipo "short" -cada uno de tamaño de 192 muestras- (de lo contrario este se trata como un solo bloque tipo "long" del mismo tamaño de 576 muestras), si esto resulta así, el algoritmo vuelve a recalcular los umbrales de enmascaramiento para cada trozo "short", y halla calcula 3 arreglos $SMR[sb]$, para cada trozo de 192 muestras. Esta idea anterior más todo el esquema general del Modelo Psicoacústico 2 se reúne en la Figura 4–6 a la Figura 4–**9**. Para las frecuencias de muestreo 48k, 44.1k y 32k Hz se entregan por la normativa en las tablas C.7 y C.8. valores distintos predispuestos para todos los cálculos de Modelo Psicoacústico.

Para $x_{\ell,l}[n]$ se realizan 2 cómputos del Modelo Psicoacústico 2, uno para la primera mitad del buffer de entrada $x_{\ell-1,0}[n]$ y otro para la segunda mitad $x_{\ell,1}[n]$, y los $SMR[sb]$ para $x_{\ell-1,0}[n]$ y $x_{\ell,1}[n]$ se comparan en el esquema de cuantización (Sección 4.4). Los cómputos generados aquí son para calcular un *índice de tonalidad* $tbb[\mathrm{z}]$, el cual expresa y discrimina qué componentes espectrales son más próximos a ser "tonales" o "ruido". Esto anterior se estima como la función de la *imprevisibilidad* ($cw[k]$) de los componentes frecuenciales de los datos temporales.

$$cw[k] = \frac{\sqrt{\left[R_\ell[k]\cos(\phi_\ell[k]) - \hat{R}_\ell[k]\cos(\hat{\phi}_\ell[k])\right]^2 + \left[R_\ell[k]\sin(\phi_\ell[k]) - \hat{R}_\ell[k]\sin(\hat{\phi}_\ell[k])\right]^2}}{R_\ell[k] + \left|\hat{R}_\ell[k]\right|}$$

$$w_{Hann}[n] = \frac{1}{2}\left[1 - \cos\left(\frac{2\pi}{N}\left(n - \frac{1}{2}\right)\right)\right], n = 0, \ldots, 255 \vee 1023 \tag{4.8}$$

Donde $R_\ell[k]$ y $\phi_\ell[k]$ corresponden a la magnitud y fase respectivamente de la FFT de $x_{\ell,l}[n]$ y $\hat{R}_\ell[k]$ y $\hat{\phi}_\ell[k]$ se entienden como la magnitud y fase predichas con extrapolación lineal según ISO/IEC. 11172-3 (Smith & Abel, 1993, p. 96):

$$\hat{R}_\ell[k] = 2R_{\ell-1}[k] - R_{\ell-2}[k], \qquad \hat{\phi}_\ell[k] = 2\phi_{\ell-1}[k] - \phi_{\ell-2}[k] \tag{4.9}$$

En la implementación para hallar *cw*[k], sus primeros 6 índices se calculan con una FFT de 1024 puntos de las 576 muestras de $x_{\ell,l}[n]$ (para esta porción, *cw*[k] se denomina $cw_{long}[k]$), para los siguientes 200 índices se utilizan 3 FFTs de 256 puntos de las mismas 576 muestras de $x_{\ell,l}[n]$, solo que repartidas consecutivamente en trozos de 192 muestras.

Se deben aplicar ventanas tipo Hanning a los respectivos trozos de 192 y 576 muestras. El rango de $cw[k]$ está dado por α, el cual es la suma de los números de la columna "FFT-

Lines" de las tablas C.7 de la normativa ISO-IEC-11172-3 (1993), en este rango los valores 206 en adelante son una constante de 0.4.

$$cw[k] = \begin{cases} cw_{long}[k], & 0 \leq k < 6 \\ cw_{short}[k], & 6 \leq k < 206 \\ 0.4, & 206 \leq k < \alpha \end{cases} \tag{4.10}$$

Una vez calculado $cw[k]$ de la Ecuación (4.10), se prosigue a reunir las 1024 líneas de energía espectral de la DFT en las "particiones" del dominio de la escala Bark (distancias $\mathrm{z}(f)$ basilares de las bandas críticas de escucha). Para esto se suman las energías en las lineas de frecuencias que corresponden a cada partición de la escala Bark. Así la energía en cada partición de Bark es $eb[\mathrm{z}]$, y la energía ponderada en cada partición es $cb[\mathrm{z}]$:

$$eb[b] = \sum_{k=\omega_l[b]}^{\omega_h[b]} R^2[k], \qquad cb[b] = \sum_{k=\omega_l[b]}^{\omega_h[b]} R^2[k] \cdot cw[k] \tag{4.11}$$

Los intervalos de las particiones de la escala Bark $\omega_l[b]$ y $\omega_h[b]$ para el Layer II son explícitos en las tablas D.3 de la normativa ISO-IEC-11172-3 (1993), mientras que para el Layer III el uso de $\omega_l[b]$ y $\omega_h[b]$ se abrevia con el número de líneas de frecuencia a sumar "FFT-Lines" de las tablas C.7.

Antes de modelar los patrones perceptuales en forma de umbrales de excitación de la membrana basilar a partir de las energías $eb[b]$ y $cb[b]$ aplicando la transformación/convolución con la función de Esparcimiento $F[dz] \rightarrow sprdngf[i,j]$ (derivada de la *función de Esparcimiento* de Schroeder explicada en (Bosi & Goldberg, 2004, p. 186)) se debe calcular dicha función de esparcimiento a partir de la expresión (4.12):

$$\begin{gathered} 10\mathrm{Log}_{10}(F[dz])[\mathrm{dB}] = \\ 15.811 + 7.5(a \cdot dz + 0.474) - 17.5\sqrt{1.0 + (a \cdot dz + 0.474)^2} + 8 \\ \cdot \min\left(0, (a \cdot dz + 0.5)^2 - 2(a \cdot dz + 0.5)\right) \end{gathered} \tag{4.12}$$

Donde dz es la diferencia en la escala de Bark entre las posiciones o particiones basilares en las frecuencias discretas de los tonos enmascaradores y enmascarados, y a es un coeficiente dependiente de la distancia dz:

$$dz = \mathrm{z}[k_{enmascarado}] - \mathrm{z}[k_{enmascarador}], \qquad a = \begin{cases} 3.0, & dz \geq 0 \ \vee j \geq i \\ 1.5, & otros \end{cases} \tag{4.13}$$

Figura 4–6: Esquema General del Modelo Psicoacústico 2 para el Layer III.

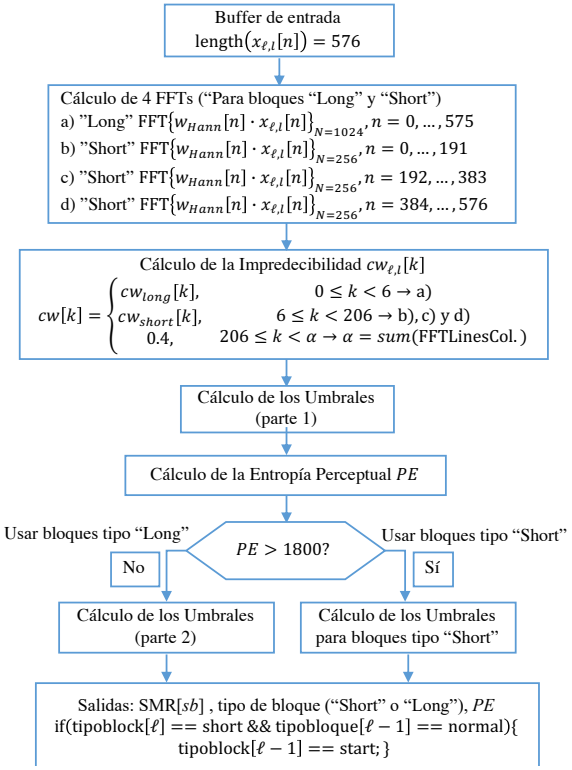


Figura 4–7: Esquema del Cálculo de los Umbrales de Enmascaramiento parte 1.

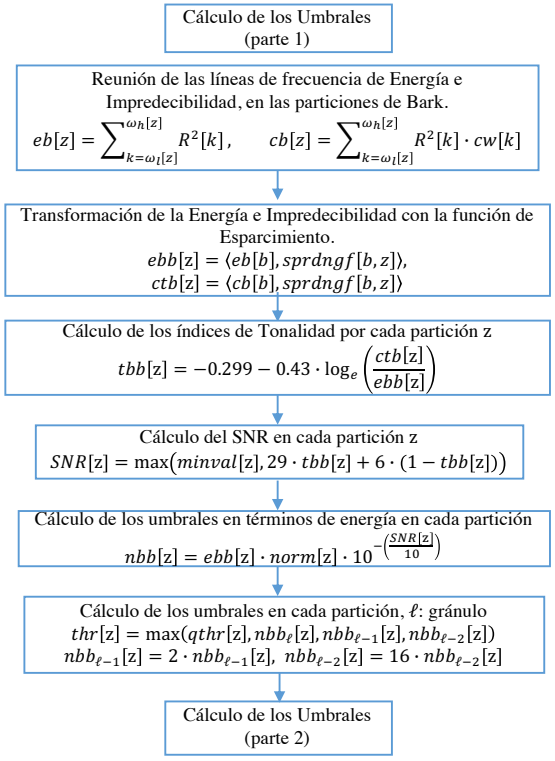


Figura 4–8: Esquema del Cálculo de los Umbrales para bloques “Short”.

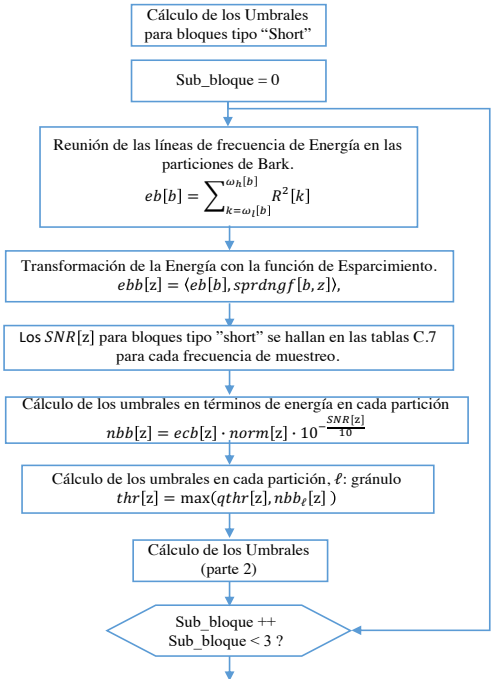
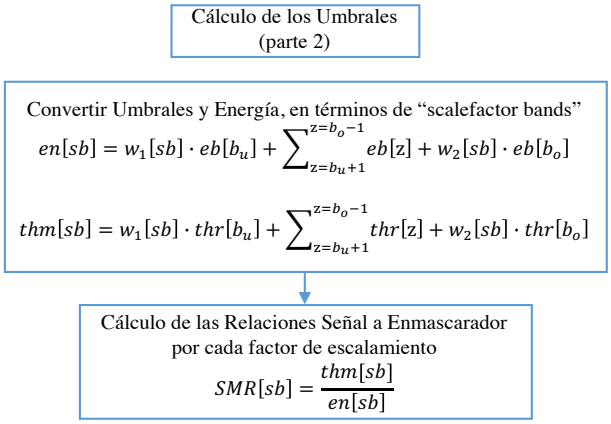


Figura 4–9: Esquema del Cálculo de los Umbrales de Enmascaramiento parte 2.



Fuente: Figuras hechas a partir de ISO/IEC. 11172-3 (Smith & Abel, 1993, p. 95).

En la implementación según la normativa (Bosi & Goldberg, 2004, p. 291), esta función de Esparcimiento -Ecuación (4.12)- se reparte en varias parte y se halla de la siguiente forma, usando los siguiente cálculos y variables:

$$tmpx[i,j] = 1.05(bval[i] - bval[j]) = a \cdot dz \tag{4.14}$$

Donde $bval[\mathrm{z}]$ son la mediana de los valores de la escala Bark en la partición z. Estos son dados en las Tablas C.7 de la normativa ISO-IEC-11172-3 (1993). Y la diferencia $bval[i] - bval[j]$ representa a dz.

$$x[i,j] = 8 \cdot \min\big(0, (tmpx[i,j] + 0.5)^2 - 2(tmpx[i,j] + 0.5)\big) \tag{4.15}$$

Donde min(n,m) es una función que entrega el valor mínimo entre los dos argumentos de entrada.

$$\begin{aligned} tmpy[i,j] = {} & 15.811389 + 7.5(tmpx[i,j] + 0.474) \\ & - 17.5\sqrt{1.0 + (tmpx[i,j] + 0.474)^2} \end{aligned} \tag{4.16}$$

Finalmente:

$$sprdngf[i,j] = \begin{cases} 0, & tmpy[i,j] \leq 100 \\ 10^{\frac{x[i,j]+tmpy[i,j]}{10}}, & tmpy[i,j] > 100 \end{cases} \tag{4.17}$$

La función de Esparcimiento, $sprdngf[i,j]$ está conformada por una matriz cuadrada simétrica (su transpuesta es igual a sí misma, así como ocurre en los factores de giro de la DFT, $\boldsymbol{W}_N^{nk}$) gracias a la diferencia dz recopilada en la matriz de la Ecuación (4.14), y dada la forma de producto interno en las Ecuaciones (4.18), éstas se interpretan más bien como mapeos o transformaciones al dominio de z Bark, más no como una convolución estricta como sugiere la normativa en (ISO/IEC. 11172-3, Smith & Abel, 1993, p. 130).

$$ebb[\mathrm{z}] = \sum_{b=1}^{b_{máx}} eb[b] \cdot sprdngf[b,z], \qquad ctb[\mathrm{z}] = \sum_{b=1}^{b_{máx}} cb[b] \cdot sprdngf[b,z] \tag{4.18}$$
$$\mathrm{z}, b = 1,2,\dots,b_{máx}$$

Las dimensiones de b acogen en (4.11) la extensión ($b_{máx}$) del número de particiones z. La expresión de $ebb[\mathrm{z}]$ muestra la energía particionada $eb[b]$ en el dominio perceptual z de la escala Bark, así mismo la energía ponderada de imprevisibilidad $cb[b]$ respecto a $ctb[\mathrm{z}]$.

Se prosigue a calcular los *índices de tonalidad* $tbb[\mathrm{z}]$ en cada partición z, lo cual según Thiagarajan & Spanias (2011, p. 35), discrimina entre las componentes tonales y no tonales de la señal de entrada $x_{\ell,l}[n]$, a partir de qué tan "tonal" es la naturaleza espectral de los componentes vistos por medio de la expresión normalizada $cbb[\mathrm{z}]$ de la función

imprevisibilidad en el dominio de Bark $ctb[\mathrm{z}]$. En la normativa se impone la condición $0 < tbb[\mathrm{z}] < 1$.

$$cbb[\mathrm{z}] = \log_e\left(\frac{ctb[\mathrm{z}]}{ebb[\mathrm{z}]}\right), \qquad tbb[\mathrm{z}] = -0.299 - 0.43 \cdot cbb[z] \tag{4.19}$$

Thiagarajan & Spanias (2011, p. 36) sintetizan conceptualmente hasta este punto del modelo que, dado que las medidas de imprevisibilidad son transformadas al dominio de Bark, a través de la función de esparcimiento que determina la energía de los enmascaradores en cualquier frecuencia, la resultante función de *índice de tonalidad* muestra la naturaleza tonal de los componentes enmascaradores dominantes en cualquier frecuencia discreta.

El paso siguiente es hallar los umbrales de enmascaramientos, a partir de una función "offset" respecto al patrón de excitación. Este "offset" es hallado evaluando los *índices de tonalidad* en cada partición de la escala Bark, determinando así el nivel de enmascaramiento global. La función offset viene dada en la expresión (4.20), donde la relación "Tono Enmascarando Ruido" (TMN) es igual a 29 dB, y la relación "Ruido Enmascarando Tono" (NMT) es igual a 6 dB.

$$O[\mathrm{z}][\mathrm{dB}] = 29 \cdot tbb[\mathrm{z}] + 6 \cdot (1 - tbb[\mathrm{z}]) \tag{4.20}$$

Los valores de $O[\mathrm{z}]$ son interpolados con base a los índices de tonalidad de un enmascarador de ruido a un valor dependiente de la frecuencia definido en la norma ISO/IEC. 11172-3 (Smith & Abel, 1993, p. 129) para un enmascarador tonal llamado $minval[\mathrm{z}]$, dado en las tablas C.7, esto con el fin de hallar el mayor valor partición por partición, y este ser usado con la relación Señal a Ruido $SNR[\mathrm{z}]$. En caso de tratarse de bloques tipo "short", el arreglo $SNR[\mathrm{z}]$ se halla en las tablas C.7.

$$SNR[\mathrm{z}] = \max(minval[\mathrm{z}], O[\mathrm{z}]) \tag{4.21}$$

La sucesión $minval[\mathrm{z}]$ se comporta como los valores límite inferiores para la relación señal a ruido, en toda la escala Bark. $SNR[\mathrm{z}]$ en términos de potencia está dado por la siguiente expresión.

$$bc[\mathrm{z}] = 10^{-\left(\frac{SNR[\mathrm{z}]}{10}\right)} \tag{4.22}$$

Según Thiagarajan & Spanias (2011, p. 35) es necesario aplicar una re-normalización a la energía en el dominio de Bark $ebb[\mathrm{z}]$, multiplicando punto a punto $ebb[\mathrm{z}]$ con el inverso de la ganancia de la energía de la función de Esparcimiento $norm[\mathrm{z}]$ (la cual se da en las tablas C.7), dado que la misma función de Esparcimiento a causa de su forma incrementa la energía estimada en cada banda crítica. Así, la energía que describe el umbral de enmascaramiento en cada partición es dada por la Ecuación (4.23)

$$nbb[\mathrm{z}] = ebb[\mathrm{z}] \cdot norm[\mathrm{z}] \cdot bc[\mathrm{z}] \tag{4.23}$$

Y los umbrales actuales de enmascaramiento $thr[\mathrm{z}]$ en el frame son derivados de la comparación con el *umbral en silencio (umbral absoluto de escucha)* analíticamente dado por la Ecuación (4.24), pero que se implementa tomando los valores de las tablas C.7.

$$qthr[k][\mathrm{dB_{SPL}}] = 3.64\left(\frac{k}{1000}\right)^{-0.8} - 6.5e^{-0.6\left(\frac{k}{1000}-3.3\right)^2} + 0.001\left(\frac{k}{1000}\right)^4 \qquad (4.24)$$

$$thr[\mathrm{z}] = \max(qthr[\mathrm{z}], nbb_{\ell}[\mathrm{z}], nbb_{\ell-1}[\mathrm{z}], nbb_{\ell-2}[\mathrm{z}]) \qquad (4.25)$$

Donde $nbb_{\ell-1}[\mathrm{z}]$ y $nbb_{\ell-2}[\mathrm{z}]$ son los umbrales energéticos de los gránulos pasados respecto al actual , $nbb_{\ell}[\mathrm{z}]$. Estos son redefinidos por medio del siguiente escalamiento como indica la normativa ISO/IEC. 11172-3 de Smith & Abel (1993, p. 92) permitiendo seguir haciendo efectivo el modelo predictivo de extrapolación de la función de imprevisibilidad $cw[k]$, generando énfasis (mayores amplitudes en los términos de bloques anteriores):

$$nbb_{\ell-1}[\mathrm{z}] = 2 \cdot nbb_{\ell-1}[\mathrm{z}], \qquad nbb_{\ell-2}[\mathrm{z}] = 16 \cdot nbb_{\ell-2}[\mathrm{z}] \qquad (4.26)$$

En caso de tratarse de bloques tipo “short”, $thr[\mathrm{z}]$ se calcula como:

$$thr[\mathrm{z}] = \max(qthr[\mathrm{z}], nbb_{\ell}[\mathrm{z}]) \qquad (4.27)$$

A diferencia del Modelo Psicoacústico 1, los umbrales de enmascaramiento no son repartidos sobre las líneas de frecuencia de la FFT, si no que estos umbrales por cada partición z, son convertidos directamente en bandas de factores de escala (*scalefactor bands*). Estos factores de escala sirven para determinar los tamaños de paso de cuantización para una banda en particular.

Con lo anterior se asegura que el ruido por cuantización esté por debajo del umbral de enmascaramiento computado. Los factores de escala por banda son grupos de líneas de frecuencias los cuales son escalados por un solo factor de escala y de acuerdo con las líneas que recojan, estas se aproximan a las bandas críticas de escucha.

Así, los valores del arreglo de la Relación Señal a Enmascarador ($SMR[\mathrm{z}]$) evalúa los factores de escala por cada banda, a través del umbral de enmascaramiento $thr[\mathrm{z}]$. Esto supone un mapeo de los umbrales $thr[\mathrm{z}]$ en cada partición z, a las bandas (sb) de escalamiento propuestas en la normativa. En general, para cada frecuencia de muestreo, hay 21 bandas de escalamiento para bloques tipo *long*, y 12 para bloques tipo *short*.

Tabla 4-6: Ejemplo de tabla C.7 de parámetros de cálculo para los Umbrales de Enmascaramiento en las particiones de la escala Bark.

Table C.7.d -- Sampling_frequency = 48 kHz short blocks

no.	FFT-lines	qthr	norm	SNR (db)	bval
0	1	4,532	0,970	-8,240	0,000
1	1	0,904	0,755	-8,240	1,875
2	1	0,029	0,738	-8,240	3,750
3	1	0,009	0,730	-8,240	5,437
4	1	0,009	0,724	-8,240	6,857
5	1	0,009	0,723	-8,240	8,109
6	1	0,009	0,723	-8,240	9,237
7	1	0,009	0,723	-8,240	10,202
8	1	0,009	0,718	-8,240	11,083
9	1	0,009	0,690	-8,240	11,864
10	1	0,009	0,660	-7,447	12,553
11	1	0,009	0,641	-7,447	13,195
12	1	0,009	0,600	-7,447	13,781
13	1	0,009	0,584	-7,447	14,309
14	1	0,009	0,532	-7,447	14,803
15	1	0,009	0,537	-7,447	15,250
16	1	0,009	0,857	-7,447	15,667
17	1	0,009	0,858	-7,447	16,068
18	1	0,009	0,853	-7,447	16,409
19	2	0,018	0,824	-7,447	17,044
20	2	0,018	0,778	-6,990	17,607
21	2	0,018	0,740	-6,990	18,097
22	2	0,018	0,709	-6,990	18,528
23	2	0,018	0,676	-6,990	18,930
24	2	0,018	0,632	-6,990	19,295
25	2	0,018	0,592	-6,990	19,636
26	3	0,054	0,553	-6,990	20,038
27	3	0,054	0,510	-6,990	20,486
28	3	0,054	0,513	-6,990	20,900
29	4	0,114	0,608	-6,990	21,305
30	4	0,114	0,673	-6,020	21,722
31	5	0,452	0,637	-6,020	22,128
32	5	0,452	0,586	-6,020	22,512
33	5	0,452	0,571	-6,020	22,877
34	7	6,330	0,616	-5,229	23,241
35	7	6,330	0,640	-5,229	23,616
36	11	9,947	0,597	-5,229	23,974
37	17	153,727	0,538	-5,229	24,312

Fuente: Imagen tomadas de la normativa ISO-IEC-11172-3 (1993) (Thiagarajan & Spanias, 2011, p. 35).

La energía $en[sb]$ y umbral $thm[sb]$ de enmascaramiento en cada banda de escalamiento sb están dadas por:

$$en[sb] = w_1[sb] \cdot eb[b_u] + \sum_{\mathrm{z}=b_u+1}^{\mathrm{z}=b_o-1} eb[\mathrm{z}] + w_2[sb] \cdot eb[b_o] \tag{4.28}$$

$$thm[sb] = w_1[sb] \cdot thr[b_u] + \sum_{\mathrm{z}=b_u+1}^{\mathrm{z}=b_o-1} thr[\mathrm{z}] + w_2[sb] \cdot thr[b_o] \tag{4.29}$$

Los parámetros b_u y b_o representan los intervalos de las particiones z, en los cuales se encuentran cada unas de las bandas de escalamiento sb (siendo b_u el límite inferior y b_o el superior). Así el primer ($eb[b_u]$ y $thr[b_u]$) y último valor ($eb[b_o]$ y $thr[b_o]$) entre los intervalos de la serie, son escalados respectivamente con $w_1[sb]$ para el límite inferior y $w_2[sb]$ para el superior.

En la sumatoria se superponen los valores de $en[\mathrm{z}]$ y $thr[\mathrm{z}]$ cuyas particiones z se hallan entre los valores escalados de $en[\mathrm{z}]$ y $thr[\mathrm{z}]$ en los límites b_u y b_o de las particiones.

Tabla 4-7: Ejemplo de tabla C.8 de parámetros para convertir los Umbrales de Enmascaramiento en factores de escala.

Table C.8.b -- Sampling_frequency = 44,1 kHz long blocks

no. sb	cbw	bu	bo	w1	w2
0	3	0	4	1,000	0,056
1	3	4	7	0,944	0,611
2	4	7	11	0,389	0,167
3	3	11	14	0,833	0,722
4	3	14	17	0,278	0,139
5	1	17	18	0,861	0,917
6	3	18	21	0,083	0,583
7	3	21	24	0,417	0,250
8	3	24	27	0,750	0,805
9	3	27	30	0,194	0,574
10	3	30	33	0,426	0,537
11	3	33	36	0,463	0,819
12	4	36	40	0,180	0,100
13	3	40	43	0,900	0,468
14	3	43	46	0,532	0,623
15	3	46	49	0,376	0,450
16	3	49	52	0,550	0,552
17	3	52	55	0,448	0,403
18	2	55	57	0,597	0,643
19	2	57	59	0,357	0,722
20	2	59	61	0,278	0,960

Fuente: Imagen tomadas de la normativa ISO-IEC-11172-3 (1993) (Smith & Abel, 1993, p. 88).

La salida final del Modelo Psicoacústico 2 está dada por la relación Señal a Enmascarador en cada banda de escalamiento:

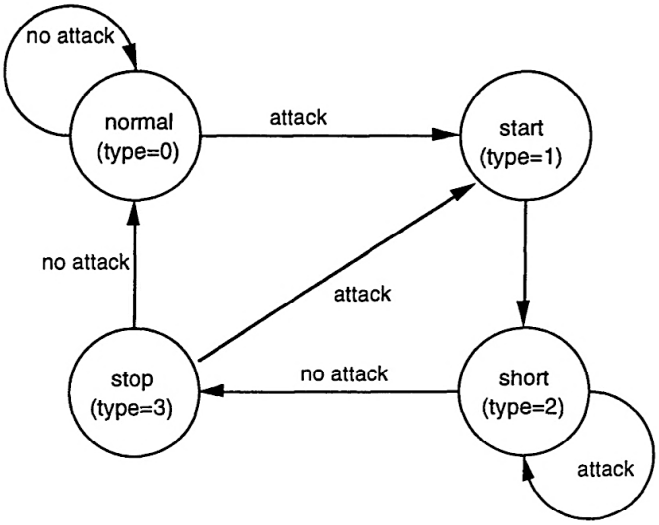
$$SMR[sb] = \frac{thm[sb]}{en[sb]} \tag{4.30}$$

El Modelo Psicoacústico 2 en el Layer III, calcula la Entropía Perceptual, para discriminar la naturaleza del tipo de señal que entra, con el fin de evitar el artefacto de compresión conocido como "pre-echo", cambiando el tipo de ventana que se usa en la etapa MDCT-IMDCT. Gracias al modelo de extrapolación de $cw[k]$ la Entropía Perceptual (PE) al detectar o no un ataque en el bloque ℓ, cambia el tipo de ventaneo del bloque $\ell + 1$. La introducción de la idea de la Entropía Perceptual resulta útil en un códec de audio al ver la definición que sugiere Bosi & Goldberg (2004); la Entropía Perceptual (PE) indica el número promedio mínimo de bits por muestra de frecuencia, necesitados para codificar una señal sin introducir una diferencia perceptual respecto a la señal original. Dada la intensidad propia de una señal y las intensidades relativas de los umbrales de enmascaramiento generados por dicha señal, a cada banda de frecuencia en un intervalo de tiempo la Entropía Perceptual se halla. $cbwidth[\mathrm{z}]$ es en las tablas C.8 "cbw".

$$PE = -\sum_{z=0}^{b_{máx}-1} cbwidth[\mathrm{z}] \cdot \log_e\left(\frac{thr[\mathrm{z}]}{eb[\mathrm{z}]+1}\right) \tag{4.31}$$

Del diagrama de cambio de estado del tipo de ventana (Figura 4–10), (Bosi & Goldberg, 2004) la Entropía Perceptual toma un valor mayor que 1800 al identificar señales con cambios rápidos en el tiempo; el algoritmo, indica un cambio del tipo de ventana actual, a una ventana tipo "Short". Siempre se usan un par consecutivo de ventanas, "Start" y "Stop", para implementarse en medio del uso de las ventanas "Long" y "Short".

Figura 4–10: Diagrama algorítmico de cambio de estado para el tipo de ventana.

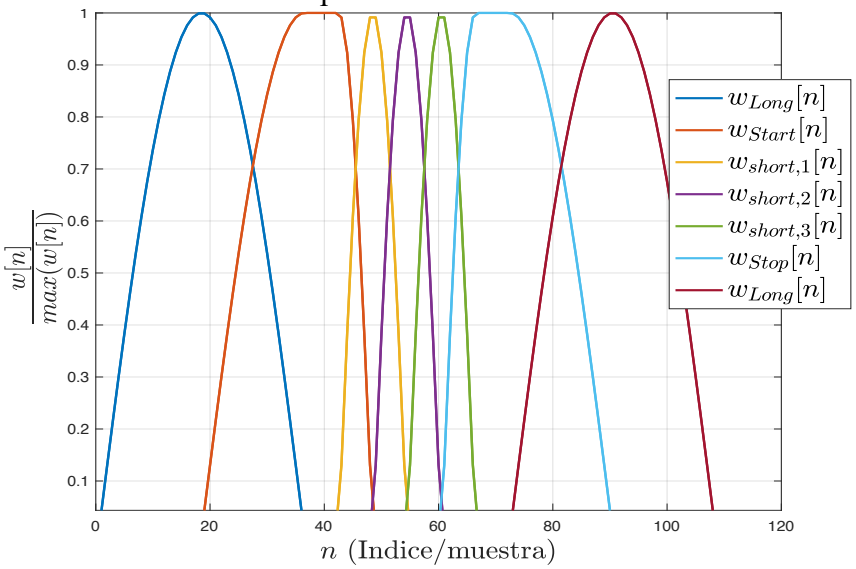


Fuente: Tomado de ISO/IEC. 11172-3 (Smith & Abel, 1993, p. 95).

Las ventanas tipo “Long”, “Start” y “Stop” están superpuestas por 18 muestras, al igual que las ventanas “Start” y “Stop” respecto a las “Short”. Las ventanas tipo “Short” están superpuestas a 6 muestras. Estas distancias de superposición son las implementadas, según el criterio del 50% o $\frac{N}{2}$ de traslapo entre bloques de información para la MDCT directa e inversa. El siguiente código muestra la lógica para definir el tipo de bloque de acuerdo con la detección o ausencia de un ataque, según el valor de la entropía perceptual PE. Es muy importante considerar el tipo de bloque actual y el pasado (gránulos ℓ y $\ell - 1$), ya que la detección de un ataque precedido de un bloque diferente del de tipo Short, hace que se ignore la naturaleza del bloque post-acontecido del bloque donde se halló el ataque, como se ilustra en la **Figura 4–12**. La lógica de la Figura 4–10, supedita al sistema a que siempre que se halle un ataque, se implemente una ventana tipo Start en el gránulo $\ell - 1$ seguido de una Short en el gránulo ℓ, ignorando lo que ocurra y se detecte en el bloque ℓ, ya que la entropía perceptual actúa indicando el cambio de tipo de ventana sobre los bloques posteriores al actual, si este modelo detecta un ataque en el gránulo ℓ, le indica al sistema que ventanee con una ventana Short al bloque $\ell + 1$.

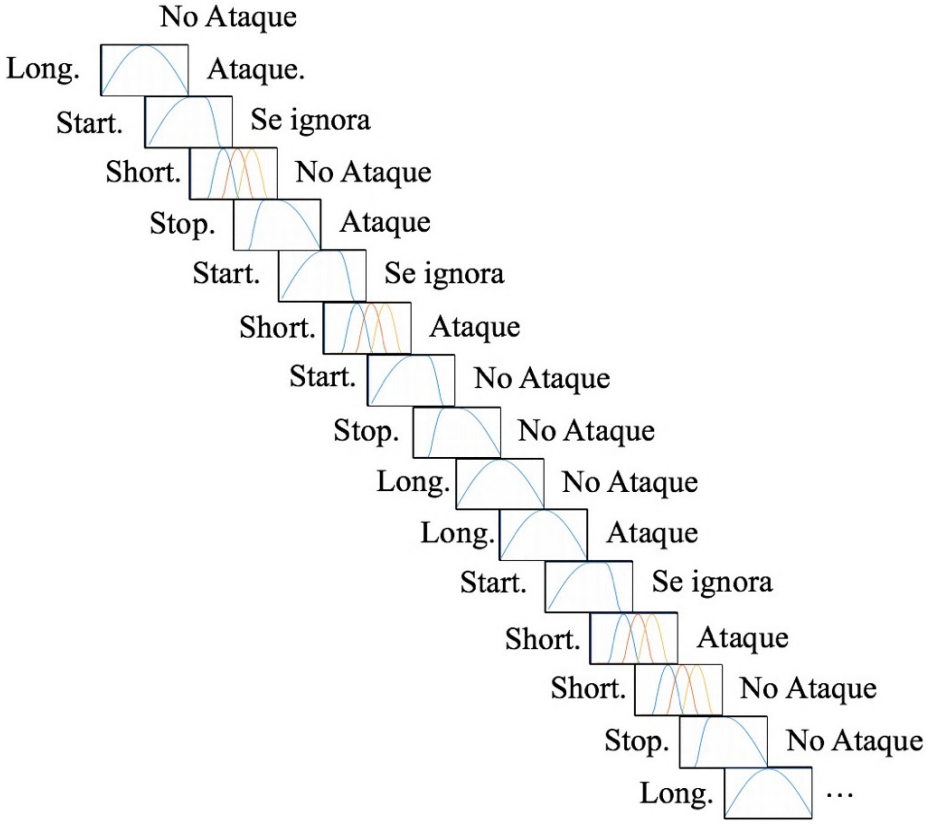
```
//Cálculo de la Entropía Perceptual PE
PE = 0.0f;
for (int b = 0; b < b_L; b++) //no se calcula para bloques Short
    PE += C_7_Long[1][b]*logf(thr_L[b]/(eb_L[b] + 1.0f));
PE *= -1;

//if (PE > 1800) //ataque detectado: short block

if (((PE > 1800) && (p_block_type == 0)) || ((PE > 1800) && (p_block_type == 10))){ //Start
    block_type = 1;
}
else if ((p_block_type == 1) || ((PE > 1800) && (p_block_type == 2)) || ((PE > 1800) && (p_block_type
== 3))){ //short
    block_type = 2;
}
else if ((PE <= 1800) && (p_block_type == 2)){ //Stop
    block_type = 3;
}
else if ((PE <= 1800) || ((PE <= 1800) && (p_block_type == 3))){//Long
    block_type = 0;
}
```

Figura 4–11: Ilustración del cambio típico de ventanas.



Fuente: Imagen de elaboración propia a partir de Matlab_R2018a.

Figura 4–12: Ejemplo de cambio de bloque, con asignación de tipo de bloque.



Fuente: Imagen de elaboración propia.

4.4 Control y esquema de compresión por Cuantización Espectral para el Layer III.

El modelo planteado para Layer III que asigna distintos estados de cuantización (análogos y directamente proporcionales a la cantidad de bits) a cada sub-banda de la MDCT, está supeditada a la información obtenida de los umbrales de enmascaramiento del modelo psicoacústico. En la sección C.1.5.4 de la normativa ISO/IEC. 11172-3 (Smith & Abel, 1993, p. 98), se describe el modelo de cuantización y ubicación en bits a partir de unos tres niveles de procesamiento, dos de estos siendo un par de bucles anidados, las cuales se encargan de la cuantización de las líneas de frecuencia y el control de la distorsión generada por la cuantización.

4.4.1 Primer Nivel: Cálculo de información de selección de los factores de escala (scfsi).

Los scfsi contienen la información de los factores de escala (agrupados en ciertos intervalos dados por la variable scfsi_bands) del gránulo actual ℓ y se deben usar para el pasado $\ell - 1$. Esta información es útil ya que, en el esquema general los bits que no se usen para transmitir los factores de escala, son usados para la codificación entrópica por Huffman. Si scfsi es igual a 0, se transmiten los factores de escala respectivos de cada gránulo, si es igual a 1, los factores de escala del primer gránulo se utilizan para el segundo también. Para determinar el uso de scfsi, se debe almacenar la siguiente información:

a) El tipo de bloque. (Si es tipo Short o Long (o Start, Stop)).

b) La energía total del gránulo actual, sumando todas las líneas de frecuencia de la MDCT.

$$en_{tot,\ell} = \text{int}\left(\log_2\left(\sum_{k=0}^{31}\sum_{\xi=0}^{17}\left|\Psi_{k,\ell}[\xi]\right|^2\right)\right) \tag{4.32}$$

c) La energía en cada banda de los factores de escala, reuniendo las 576 muestras de $\Psi_{k,\ell}[\xi]$, en un arreglo unidimensional $xr_\ell[\xi]$, donde $lbl[sb]$ es cada uno de los índices iniciales de que las líneas de frecuencia que reúne cada banda de factor de escala sb, y $bw[sb]$ el número de líneas de frecuencia dentro de cada banda sb:

$$en_\ell[sb] = \text{int}\left(\log_2\left(\sum_{\xi=lbl[sb]}^{lbl[sb]+bw[sb]-1}|xr_\ell[\xi]|^2\right)\right) \tag{4.33}$$

$lbl[sb]$ y $bw[sb]$, están dadas para cada frecuencia de muestreo en las tablas B.8 de la norma ISO-IEC 11172-3.

d) La distorsión permitida en cada banda de los factores de escala a partir de los $SMR_\ell[sb]$ que generó el modelo psicoacústico:

$$xmin_\ell[sb] = \frac{SMR_\ell[sb]}{bw[sb]} \times \sum_{\xi=lbl[sb]}^{lbl[sb]+bw[sb]-1} |xr_\ell[\xi]|^2 \tag{4.34}$$

La cual no es más que el umbral de enmascaramiento en cada banda sb repartido sobre cada uno de los anchos de estas bandas:

$$xmin_\ell[sb] = \frac{thm_\ell[sb]}{en_\ell[sb]} \cdot \frac{1}{bw[sb]} \sum_{\xi=lbl[sb]}^{lbl[sb]+bw[sb]-1} |xr_\ell[\xi]|^2$$

En la ecuación anterior, la serie que halla las energías en cada banda sb a partir de $|xr_\ell[\xi]|^2$, sirve para normalizar o cancelar los valores del cociente $en_\ell[sb]$ definido en la Ecuación (4.30).

Finalmente $xmin_\ell[sb]$ se pasa a términos de N bits tal como en los pasos b) y c):

$$xm_\ell[sb] = \text{int}(\log_2(xmin_\ell[sb])) \tag{4.35}$$

A partir de estos cálculos, hay cuatro criterios que entrega la normativa para determinar si scfsi se usará en general. Si alguno de estos cuatro criterios no se cumple, el scfsi se desactiva (esto quiere decir que scfsi es igual a 0 para todas las scfsi_bands):

1. Los valores de $\Psi_{k,\ell}[\xi]$ no son todos iguales a 0.
2. Ninguno de los 2 gránulos es tipo Short.
3. $\left|en_{tot,\ell-1} - en_{tot,\ell}\right| < 10$
4. $\sum_{sb}|en_{\ell-1}[sb] - en_\ell[sb]| < 100$, haciendo la comparación para cada sb.

Los valores a los cuales se comparan los argumentos de la izquierda son entregados como recomendación en la normativa, pensando en que scfsi solo activará en caso de una energía o distorsión de magnitud similar. En caso de que scfsi no sea desactivado, después de pasar por las cuatro condiciones de arriba, hay dos criterios más para cada banda scfsi_band, en donde ambas condiciones se deben cumplir para activar a scfsi en dicha scfsi_band:

1. $\sum_{\text{scfsi\_band[sb]}}|en_{\ell-1}[sb] - en_\ell[sb]| < 10$, haciendo la suma sobre los intervalos de cada scfsi_band
2. $\sum_{\text{scfsi\_band[sb]}}|xm_{\ell-1}[sb] - xm_\ell[sb]| < 10$, haciendo la suma sobre los intervalos de cada scfsi_band

4.4.2 Segundo Nivel: Control de distorsión (Bucle externo)

El bucle externo controla el ruido que produce la cuantización de las líneas de frecuencia de la MDCT, que se realiza en el Tercer Nivel (Bucle interno). Los factores de escala son usados para darle forma al ruido de cuantización, de tal forma que este quede enmascarado. Como se verá, las líneas de frecuencia son deformadas (escaladas) en primera instancia después de realizar la cuantización, y dependiendo del ruido generado por este procedimiento, se escala de nuevo cada valor de $xr_\ell[\xi]$, para luego volver a ser cuantizado y así (proceso iterativo) hasta que el ruido de cuantización sea admisible gracias a su enmascaramiento, produciendo la compresión perceptual del espectro.

El ruido generado por la cuantización en cada una de las bandas sb de los factores de escala se calcula con la siguiente función:

$$xfsf[sb] = \frac{1}{bw[sb]} \cdot \sum_{\xi=lbl[sb]}^{lbl[sb]+bw[sb]-1} \left(|xr_\ell[\xi]| - (ix_\ell[\xi])^{\frac{4}{3}} \cdot 2^{\frac{1}{4}(qquant+quantanf)} \right)^2 \qquad (4.36)$$

Donde $ix_\ell[\xi]$ son las líneas de frecuencia cuantizadas. Y se hace la diferencia entre las líneas de frecuencia actuales $xr_\ell[\xi]$ con los valores $(ix_\ell[\xi])^{\frac{4}{3}} \cdot 2^{\frac{1}{4}(qquant+quantanf)}$ que representan la decuantización parcial de los valores cuantizados $ix_\ell[\xi]$. Con esta diferencia se halla la distorsión introducida por $ix_\ell[\xi]$. Es necesario decuantizar a $ix_\ell[\xi]$, para poder hacer la correcta operación de la resta sobre el intervalo de valores de $xr_\ell[\xi]$.

El siguiente código (hallado en el método amp_scalefactor_bands()) dentro del método outerloop() para bloques tipo Long, muestra cómo el ruido de cuantización es amplificado a través de la multiplicación de las líneas de frecuencia con los respectivos factores de escala en cada banda.

```
if (scalefac_scale[granulo] == 0)
    ifqstep = sqrtf(2.0);
else
    ifqstep = powf(2.0, 0.5*(1.0 + scalefac_scale[granulo]));

for (sb = 0; sb < 21; sb++)
    counter ++;
    if (xfsf_L[sb] > xmin_L[granulo][sb]) {
        xmin_L[granulo][sb] *= powf(ifqstep, 2.0);
        for (int ξ = sb_beg[sb]; ξ < (sb_end[sb]+1); ξ++)
            xr_array[ξ] *= ifqstep;
    }
if (counter == 0)
    endLoop = true;
```

La compresión perceptual se muestra en el código de arriba, donde existe la condición de realizar el escalamiento de $xr_\ell[\xi]$ solo si los valores de distorsión $xfsf[sb]$ generadas sobre cada ancho de banda sb por la cuantización $ix_\ell[\xi]$ son mayores que el umbral de enmascaramiento $thm_\ell[sb]$ implícito en los valores de $xmin_\ell[sb]$. Así, eventualmente se esperaría que en todas las bandas sb, el ruido $xfsf[sb]$ no superase al umbral mínimo de enmascaramiento que expresa $xmin_\ell[sb]$, esto se controla con la variable entera "counter" y la booleana "endloop", los cuales actúan en el método iterationloop() en el código de abajo:

```
do{
    outerLoop();
    iteracion++;
}
while (endLoop == false);
```

La normativa también muestra la posibilidad de amplificar las bandas superiores del espectro denotadas por el arreglo pretab[], esto se hace una función llama preemphasis(). Esta opción solo se activa si en todas las cuatro bandas superiores de los factores de escala la distorsión actual supera el umbral después de la primera llamada del método innerLoop()

```
for (int scfsi_band = 0; scfsi_band < 4; scfsi_band++)
    if (scfsi[scfsi_band])
        for (int sb = begIndex[scfsi_band]; sb < endIndex[scfsi_band]; sb++)
            for (int ξ = sb_beg[sb]; ξ < (sb_end[sb]+1); ξ++)
                xr_array[ξ] *= powf(ifqstep, float(pretab[sb]));
```

La definición del código del método outerloop() es el siguiente:

```
void Quantization_Mod_Layer3::outerLoop(){
    innerLoop();
    calc_dist();
    preemphasis();
    amp_scalefactor_bands();
}
```

4.4.3 Tercer Nivel: Cuantización (Bucle interno)

Dado que para este proyecto se acordó no implementar la codificación de datos, que abarca manipular los bits *per se* y su ubicación para generar los *headers* del códec del MPEG-1 en las etapas de codificación y decodificación. Por lo tanto, no se implementa en el presente, la compresión por pérdidas de la tasa de datos por bloque llevada a cabo en su mayoría por el algoritmo de codificación entrópica de Huffman.

En la normativa ISO-IEC 11172-3, los procedimientos de más bajo nivel como los acabados de mencionar, son llevados acabo en este tercer nivel. De acuerdo con lo dicho, se decidió usar la cuantización $ix[\xi]$ únicamente como datos del control de distorsión para llevar a cabo la compresión perceptual.

$$ix[\xi] = \text{nint}\left(\left(\frac{|xr_\ell[\xi]|}{2^{\frac{1}{4}(qquant+quantanf)}}\right)^{\frac{3}{4}} - 0.0946\right) \tag{4.37}$$

La Ecuación (4.37) de cuantización es de tipo no uniforme de compansión de potencias Midtread, donde el cociente $2^{\frac{1}{4}(qquant+quantanf)}$ muestra cierta semejanza a la ecuación de cuantización uniforme Midtread del método de la Figura 2–27. La suma de las variables $qquant + quantanf$, se entiende como los "pasos de cuantización", que dependen del estado máximo (o cantidad de bits máxima) del rango de codificación de las tablas de Huffman, este se encuentra en la página 61 de la normativa, según la variable linbits = 13, lo cual representa al estado $2^{13} = 8192$. Según esta idea, los valores de $xr_\ell[\xi]$ deben ser cuantizados en un bucle hasta que su máximo valor sea menor que el estado 8192, y por

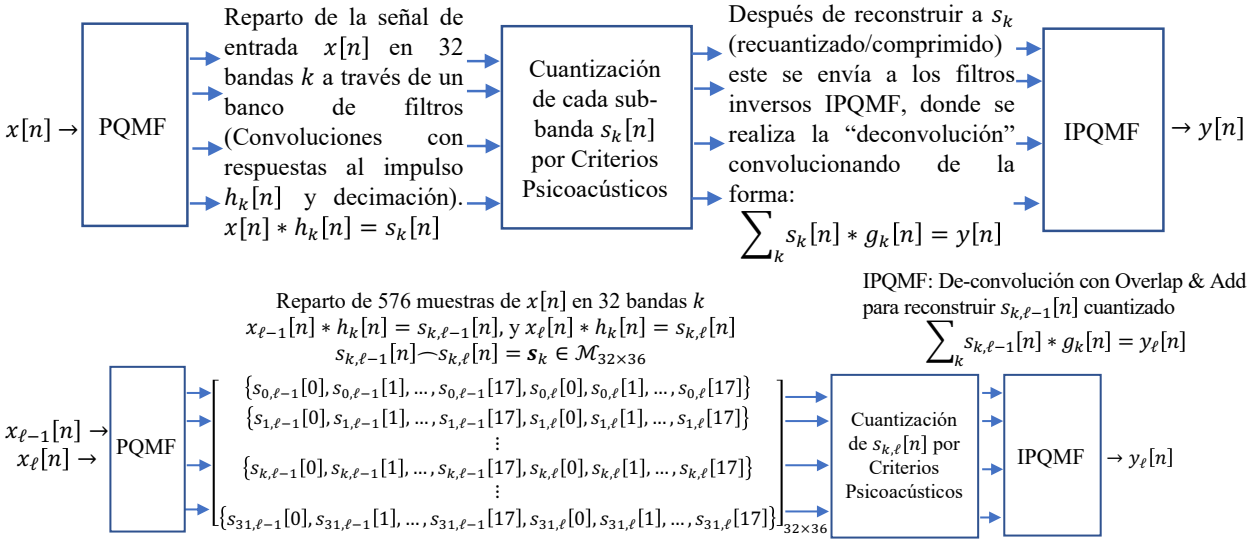
cada iteración la variable $qquant$ incrementa su valor en bits por 1, haciendo que la cuantización sea mayor y se reduzcan los valores de $xr_\ell[\xi]$. El algoritmo para bloques tipo Long que describe lo anterior es el siguiente:

```
do{
    qquant[0] ++;
    quantize(0);
    ix_max(0);
}
while (max_ix[0] > 8206);

void Quantization_Mod_Layer3::quantize(int bl){
    float deno;

    if (block_type == 2){ //bloque tipo short
        deno = powf(2.0,(qquant[bl + 1] + quantanf)*0.25);
        for (int ξ = 0; ξ < 192; ξ++){
            ix_S[bl][ξ] = nint(powf(fabsf(xr_S_array[bl][ξ])/deno,0.75) - 0.0946);
        }
    }else{ //cualquier otro tipo de bloque distinto al short
        deno = powf(2.0,(qquant[0] + quantanf)*0.25);
        for (int ξ = 0; ξ < 576; ξ++){
            ix_L[ξ] = nint(powf(fabsf(xr_array[ξ])/deno,0.75) - 0.0946);
        }
    }
}
```

4.5 Variaciones para la implementación del Esquema Layer II

Se presenta aquí una descripción de los parámetros necesarios para implementar el esquema base del Layer II según (ISO/IEC. 11172-3, Smith & Abel, 1993, p. 128), el diagrama de procesamiento se visualiza en la **Figura 4–13**.

Figura 4–13: Esquema general del esquema de compresión MPEG-1 Layer II.



Fuente: Imagen de elaboración propia.

El esquema de compresión Layer II se diferencia del Layer III por la ausencia de aplicación de la transformada MDCT y el tipo de modelo de cuantización sugerido en la normativa

ISO/IEC. 11172-3 (Smith & Abel, 1993, p. 31). Ambos Layer utilizan el Modelo Psicoacústico 2 explicado en la sección 0, en el cual la sección D.2 de la normativa ISO/IEC. 11172-3 (Smith & Abel, 1993, p. 128) explica el modelo Psicoacústico 2 base que se aplica al Layer II. Debido a que en el presente proyecto se omitieron los procesos de codificación de datos (creación de ficheros y asignación de bits para estos, compresión por pérdidas debido a la codificación entrópica de Huffman en el Layer III y "Alocación" o ubicación de bits para el Layer II), al considerar que a los niveles de cuantización respectivos se les reparte una cantidad de bits específicos dependiendo del modelo Psicoacústico en el Layer II, por lo que fue necesario diseñar un esquema de compresión que permitiese emular las pérdidas perceptuales (pero no informáticas o de cantidad de información en bits), así como se logró para el Layer III, en el procedimiento de la sección anterior (4.4). Es importante mencionar que la estructura de entrada y salida de muestras de audio es el mismo que se utiliza para el Layer III, visualizado en la Figura 4–3 y Figura 4–4 y la Tabla 4-4, exceptuando el uso de los métodos de la MDCT.

El diseño realizado para un modelo de cuantización (Modelo base de compresión perceptual), adaptado al Layer II para omitir la codificación de datos y lograr una compresión perceptual manipulando el espectro, se hizo con base al modelo de cuantización ya utilizado en el Layer III, en donde en vez de manipular escalando las líneas de frecuencia $xr_\ell[\xi]$ de la MDCT, se alteran las muestras temporales/espectrales salientes de los filtros PQMF $s_k[n]$ escalandolas, como se explicará a continuación. Este esquema de compresión también posee tres niveles que operan de forma idéntica a los utilizados para el Layer III, en los que dos de ellos son iterativos, y se presentan como métodos de control de la distorsión agregada a las muestras de $s_k[n]$ de acuerdo con los umbrales de enmascaramiento calculados en el Modelo Psicoacústico 2.

4.5.1 Descripción de los 3 niveles de procesamiento para la compresión perceptual en el Layer II

Primer nivel: Como se describió en la sección anterior, todos los cálculos preliminares que permiten seleccionar los factores de escala scfsi, se hacen a partir de cálculos energéticos con las líneas de frecuencia $xr_\ell[\xi]$ de la MDCT, aquí, se realiza una transformada real tipo STFT de las muestras de tiempo $s_\ell[n,k]$ filtradas por los PQMF en las bandas k, ventaneada por una función $w_{Hann}[n]$ tipo Hanning de 36 muestras como se expresa en la Ecuación (4.38). Es importante mencionar que únicamente se realiza esta transformación para el análisis (y no síntesis) durante todo el modelo explicado aquí.

$$sr_\ell[\xi,k] = \sum_{n=0}^{35} w_{Hann}[n]\cdot s_\ell[n,k]\cdot \cos\left(\frac{2\pi}{N}\xi n\right) \qquad (4.38)$$
$$\xi = 0,1,\dots,17, \qquad k = 0,1,\dots,31, \qquad sr_\ell[\xi,k] \in \mathbb{R}$$

n son los 36 índices de tiempo discreto de las 18 líneas ξ de frecuencia de la transformada y donde k es el índice de las 32 bandas del PQMF. Aquí se obtiene una matriz $sr_\ell[\xi,k]$ de dimensiones 18×32 (iguales a las dimensiones de la transformada MDCT $\Psi_{k,\ell}[\xi]$), donde el término $\cos\left(\frac{2\pi}{N}\xi n\right)$ se comporta como las bases reales obtenidas de los Factores Giro

complejos de $\boldsymbol{W}_N^{nk}$ de la DFT usual. Cabe aclarar que $sr_\ell[\xi,k]$, es básicamente una transformada MDCT pero con el argumento de Fourier $\phi[\xi,n]=\frac{2\pi}{N}\xi n$ en sus bases, tal como se demostró en la sección 2.4.4 en la Ecuación (2.63). Es necesario recordar que es suficiente con calcular la mitad (18) de los índices de frecuencia en lugar de la extensión completa de los índices temporales (36), ya que la segunda mitad corresponde a las frecuencias negativas.

Calculando $sr_\ell[\xi,k]$ como se expresa en la Ecuación (4.38), se logra un análisis semejante al del Layer III para el presente modelo de compresión perceptual. En el primer nivel, al igual que en el descrito para el Layer III, se reinician todas las variables de control para el frame actual. Siempre se hayan filtrado por PQMF 576 muestras temporales, se realiza el cálculo de la Ecuación (4.38) en el código siguiente:

```
//Primer cálculo de la STFT real:
for (int k = 0; k < 32; k++)
    for (int ξ = 0; ξ < 18; ξ++){
        part_sum = 0;
        for (int n = 0; n < 36; n++)
            part_sum += w_hann[n] * s[k][n] * re_W[ξ][n];
        sr_array[k*18 + ξ] = part_sum;
        sr[k][ξ] = part_sum;
    }
```

La matriz $sr_\ell[\xi,k]$ se debe reorganizar de tal forma que resulte en un arreglo unidimensional $sr_\ell[\xi]$ como se expresa en el código anterior con sr_array[k*18 + ξ] = part_sum;. La distorsión permitida $xmin_\ell[sb]$ en cada banda de los factores de escala a partir de los $SMR_\ell[sb]$ que generó el modelo psicoacústico 2 se realiza con la Ecuación (4.39):

$$xmin_\ell[sb]=\frac{SMR_\ell[sb]}{bw[sb]}\times\sum_{\xi=lbl[sb]}^{lbl[sb]+bw[sb]-1}|sr_\ell[\xi]|^2 \qquad (4.39)$$

Los Parámetros que permiten controlar qué fragmentos del espectro se deben manipular por medio de los indicadores scfsi, tal como se explicó en la sección 4.4.1: $en_{tot,\ell}, en_\ell[sb]$ y $xm_\ell[sb]$, se calculan de igual forma, a excepción del uso de las muestras de $sr_\ell[\xi]$. En el siguiente código se muestra cómo el primer nivel contiene al segundo y tercero, y como estos están anidados en un bucle condicional do-while, que se rompe apenas se indique que no hay bandas faltantes por amplificar, esto se da por medio de la variable booleana end_Loop.

```
do{
    outerLoop();
    iteracion++;
}
while (endLoop == false);
gr++; gr %= 2;
```

Segundo Nivel: Tal como pretende el segundo nivel de control en el esquema del Layer III (Sección 4.4.2), aquí se controla la cantidad de ruido introducido por la cuantización $is_\ell[\xi]$ de las muestras convolucionadas que contiene $s_\ell[n,k]$ que se realiza en el nivel 3. El ruido

$sfsf[sb]$ que se introduce por cuantización es el calculado en la Ecuación (4.40), obedeciendo la misma lógica explicada en la Sección 4.4.2 al calcular los $xfsf[sb]$.

$$sfsf[sb] = \frac{1}{bw[sb]} \cdot \sum_{\xi=lbl[sb]}^{lbl[sb]+bw[sb]-1} \left(|sr_\ell[\xi]| - (is_\ell[\xi])^{\frac{4}{3}} \cdot 2^{\frac{1}{4}(qquant+quantanf)} \right)^2 \qquad (4.40)$$

La distorsión $sfsf[sb]$ debe ser hallada (en el método amp_scalefactor_bands()) una vez se haya calculado de nuevo $sr_\ell[\xi, k]$, lo cual significa que por cada iteración de cuantización (en el método innerLoop()), se calcula $sfsf[sb]$ habiendo recalculado antes la STFT $sr_\ell[\xi, k]$ como se muestra en el siguiente código:

```
void Quantization_Mod_Layer2::outerLoop(){
      //segundo y enésimo cálculo de la STFT real:
      float part_sum;
      for (int k = 0; k < 32; k++)
          for (int ξ = 0; ξ < 18; ξ++){
              part_sum = 0;
              for (int n = 0; n < 36; n++)
                  part_sum += s[k][n] * w_hann[n] * re_W[ξ][n];
              sr_array[k*18 + ξ] = part_sum; sr[k][ξ] = part_sum;
          }

      innerLoop(); //siempre se cuantiza al menos una vez.
      calc_dist();
      preemphasis();
      amp_scalefactor_bands();
}
```

De nuevo, la distorsión introducida por la cuantización y la amplificación de las bandas de frecuencia está supeditada a los umbrales de enmascaramiento que indica el parámetro $xmin_\ell[sb]$. La amplificación de las bandas y el control de acceso a este proceso está dada en el siguiente código:

```
int counter = 0;
endLoop = true; //si no entró a algún scfsi[scfsi_band], se sale del do-while
for (int scfsi_band = 0; scfsi_band < 4; scfsi_band++){
    if (scfsi[scfsi_band])
        for (int sb = begIndex[scfsi_band]; sb < endIndex[scfsi_band]; sb++){
            endLoop = false;
            if (xfsf[sb] > xmin[gr][sb]) {
                counter ++;
                xmin[gr][sb] *= powf(ifqstep, 2.0);
                for (int k = B_8_II[0][scfsi_band]; k < B_8_II[1][scfsi_band]; k++)
                    for (int n = 0; n < 18; n++)
                        s[k][n] *= ifqstep/36.0f;
            }
            if ((counter == 0) || (sb == endIndex[3] - 1))
                endLoop = true;//todas las bandas están por debajo del límite de thm[sb]
        }
}
```

De este código es importante resaltar que el factor de escalamiento para los $s_\ell[n, k]$, se reduce por un factor de 36 (de la forma ifqstep/36), dado que en el Layer III, cada factor de escalamiento fue diseñado en ISO/IEC. 11172-3 (Smith & Abel, 1993, p. 103) para estar acompañada por un factor 1, recordando que al manipular la transformación en frecuencia de la MDCT le lleva a tener un rango entre -18 y 18 en plano real. Con esta técnica se reduce la distorsión introducida a los $s_\ell[n, k]$.

Tercer nivel: Aquí tal como se presenta en el código a continuación, se realiza la operación de cuantización de las muestras de $s_\ell[n,k]$ filtradas por los PQMF, que sirven como control para observar el límite de ruido introducido por cuantización en $sfsf[sb]$. Este proceso se realiza con las variables y estructura algorítmica de forma muy similar a como se explicó en la Sección 4.4.3.

```
void Quantization_Mod_Layer2::innerLoop(){
    do{
        qquant[0] ++;
        quantize(0);
        ix_max(0);
    }
    while (max_ix[0] > 8206); //nivel de cuantización sugerido en la norma 2^(13)+14

}

void Quantization_Mod_Layer2::quantize(int bl){
    float deno = powf(2.0,(qquant[0] + quantanf)*0.25);
    for (int ξ = 0; ξ < 576; ξ++)
        ix[ξ] = nint(powf(fabsf(sr_array[ξ])/deno, 0.75) - 0.0946);
}
```

4.6 Metodología para la obtención de los parámetros objetivos

En la sección 3.1.1 se hizo una breve descripción de la metodología para obtener los datos de salida de ambos sistemas de compresión (Layer II y III) y se mencionaron las condiciones para el diseño de cada una de las señales necesarias para obtener cada parámetro estipulado en el Objetivo Específico 3 del presente trabajo. Los datos numéricos adquiridos se obtuvieron con el uso del sistema (Layer II y III) en formato ejecutable (En JUCE "Audio Application") el cual consta de los mismos archivos y algoritmia utilizados para el plugin, pero este siendo un ejecutable (ej. extensión .app o .exe) de donde se hace provecho del *promt* de Xcode para imprimir las muestras de salida y/o cualquier otro parámetro deseado.

En la Tabla 4-8, se muestran los arreglos y matrices que albergan las señales de prueba para cada parámetro del Objetivo Específico número 3. Los arreglos de cada fila de la Tabla 4-8 fueron diseñados según los códigos en Matlab en el Anexo 7, siguiendo la regla general de que su longitud debe ser un factor entero de 32, para ser procesado como se muestra en la Tabla 4-9 y como se explicó en la sección 4.2.1.

Tabla 4-8: Arreglos de entrada para las señales de prueba del Objetivo Específico 3.

	Parámetro	Arreglo que alberga la señal de prueba de entrada.

1	Función de Transferencia.	`extern float au441k[220640]; //Señal Chirp para fs = 44.1 k Hz`
		`extern float au48k[240000]; //Señal Chirp para fs = 48k Hz`

2	THD y SNR ambos en las 32 bandas del PQMF.	`extern float armons_44[32][1152];//Señal: tonos puros para fs = 44.1 k Hz`
		`extern float armons_48[32][1152];//Señal: tonos puros para fs = 48 k Hz`

3	Densidad del espectro de potencia (PSD).	`extern float white_noise44k[44128];//Señal WGN para fs = 44.1 k Hz`
		`extern float white_noise48k[48000];//Señal WGN para fs = 48 k Hz`

Fuente. Elaboración propia.

Las señales de la **Tabla 4-8**, como se muestra en la columna 3 de la **Tabla 4-9**, ingresan al sistema para ser procesadas a través del método "x.addSample()" del PQMF y el método "Psyco_Layer2.ingresoMuestraFifo()" del modelo psicoacústico. Cada arreglo de la **Tabla 4-8** se recorre usando la variable entera `au_index`, que se inicia en 0 en el método "prepareToPlay()".

Tabla 4-9: Uso de los Arreglos (ej. para f_s = 48k Hz) de entrada de la Tabla 4-8.

	Parámetro	Código en el *for* (**) del código de la sección 4.2.1

1	Función de Transferencia.	`for(int blockIndex = 0; blockIndex < bufferToFill.numSamples/32; blockIndex++){`
		`  for(int i = 0; i < 32; i++){`
		`  //MEDICIÓN con chirp:`
		`  x.addSample(au48k[au_index]);`
		`  Psyco_Layer2.ingresoMuestraFifo(au48k[au_index]);`
		`  au_index++;`
		`  if (au_index == 240000)`
		`  std::cout << "END!" << std::endl;`
		`  au_index %= 240000;`
		`  ...`

2	THD y SNR ambos en las 32 bandas del PQMF.	`for(int blockIndex = 0; blockIndex < bufferToFill.numSamples/32; blockIndex++){`
		`  for(int i = 0; i < 32; i++){`
		`//Tonos puros para MEDICIÓN de THD y SNR (tonos puros en las 32 bandas del PQMF):`
		`  int ind = 31; //ind es el número del armónico en la banda k deseada`
		`  x.addSample(armons_48[ind][au_index]);`
		`  Psyco_Layer2.ingresoMuestraFifo(armons_48[ind][au_index]);`
		`  au_index++;`
		`  if (au_index == 1152)`
		`  std::cout << "END!"<< std::endl;`
		`  au_index %= 1152;`
		`  ...`

3	Densidad del espectro de potencia (PSD).	`for(int blockIndex = 0; blockIndex < bufferToFill.numSamples/32; blockIndex++){`
		`  for(int i = 0; i < 32; i++){`
		`  //MEDICIÓN con ruido blanco gaussiano para la PSD:`
		`  x.addSample(white_noise48k[au_index]);`
		`  Psyco_Layer2.ingresoMuestraFifo(white_noise48k[au_index]);`
		`  au_index++;`
		`  if (au_index == 48000)`
		`  std::cout << "END!" << std::endl;`
		`  au_index %= 48000;`
		`  ...`

Fuente. Elaboración propia.

Para imprimir las muestras del audio de salida en el *promt* de Xcode en este programa tipo "Audio Application" de JUCE, se utilizó la función estándar de C++ *cout* de *std* como se muestra a continuación en el *for* (****) del código de la sección 4.2.1.

```
for (int n = 0; n < bufferToFill.numSamples; n++){
    rightbuffer[n] = auxArray[n];
    leftbuffer[n] = auxArray[n];
    std::cout << auxArray[n] << std::endl;
}
```

En cada fila de la **Tabla 4-9** existe un condicional *if* que revisa que se haya recorrido todo el arreglo respectivo, y una vez cumplida esta condición se imprime "END!" para indicar la última muestra de salida procesada en el código de arriba. Todas las muestras que se imprimen en el *promt* de Xcode se copian y pegan en un archivo ".m" de Matlab para ser procesados luego como arreglos de entrada como se muestra en los códigos del Anexo 8.

En la **Tabla 4-8**, fila 1, las señales de prueba *chirp* en el arreglo `au_k[muestra]` están diseñadas para 5 ciclos de toda su extensión en frecuencia, para así asegurar que se recorran todas las frecuencias pertenecientes a dichas señales. Es decir, estas señales tardan 5 segundos y recorren 5 veces todo su espectro (en 1 segundo contienen todas sus componentes de frecuencia). Como se mencionó en la sección 3.3, se optó por medir THD y SNR en cada una de las 32 bandas del PQMF, para esto en la fila 2 de la Tabla 4-8, se muestra la matriz `armons_[armónico][muestra]` que alberga en cada una de sus 32 filas, 1152 muestras de cada tono puro cuya frecuencia fundamental corresponde al centro del ancho de banda de cada filtro del PQMF (en la Tabla 4-9 fila 2, se muestra que con la variable entera "ind" se elige manualmente el armónico deseado para procesar). De forma similar al proceso llevado a cabo para ingresar las muestras *chirp* al sistema, se hizo para ingresar la señal de Ruido Blanco Gaussiano WGN a 44.1k y 48k Hz con el arreglo `white_noise_k[]`.

4.7 Metodología y condiciones para la realización de las pruebas subjetivas

En la ejecución de los test subjetivos y retomando las caracterizaciones del test subjetivo discutido en la sección 3.3, se plantearon los experimentos por medio de dos sesiones de evaluación de los correspondientes Layers II y III, en el que se planifican los resultados de la siguiente manera (donde "5" es la mayor valoración según la Tabla 2-2: que se asigna para los audios originales sin ninguna clase de procesamiento, y donde "x" es cualquier valoración de 1 a 5 para el resto de audios procesados: "Alterado" y "Comprimido".

En el entrenamiento 1 y 2, la disposición de la sesión es la siguiente:
Frec muestreo 44.1k Hz, 24 bits, nombre archivo "Test1_asIAm"

Disposición Prueba	Original	Original Alterado	Comprimido	Original	Original	Original Alterado	Comprimido
Valor	5	x	x	5	5	x	x

En el entrenamiento 1 y 2, la disposición de la sesión 2 es la siguiente:
Frec muestreo 44.1k Hz, mono, 24 bits, nombre archivo "Test2_solitaryShell"

Disposición Prueba	Original Alterado	Original	Comprimido	Original	Original Alterado	Original	Comprimido
Valor	x	5	x	5	x	5	x

En el entrenamiento 1 y 2, la disposición de la sesión 3 es la siguiente:
Frec muestreo 48k Hz, mono, 24 bits, nombre archivo "Test3_avishaiCohen"

Disposición sesión	Comprimido	Original	Original	Original Alterado	Comprimido	Original Alterado	Original
Valor	x	5	5	x	x	5	x

En el entrenamiento 1 y 2, la disposición de la sesión 4 es la siguiente:
Frec muestreo 48k Hz , mono, 24 bits, nombre archivo "Test4_milesDavis"

Disposición sesión	Comprimido	Original	Original	Original Alterado	Original Alterado	Original	Comprimido
Valor	x	5	5	x	x	5	x

En el entrenamiento 1 y 2, la disposición de la sesión 5 es la siguiente:

Frec muestreo 44.1k Hz, mono, 24 bits, nombre archivo “Test5_pacoDeLucia”

Disposición sesión	Comprimido	Original	Original	Original Alterado	Comprimido	Original Alterado	Original
Valor	x	5	5	x	x	5	x

Fuente. Elaboración propia.

Haciendo mención, en que un entrenamiento consta de 5 sesiones, en cada una de estas sesiones se hacen 3 comparaciones subjetivas comprimido vs Original, original vs señuelo y las tres señales. De las cuales se obtiene por sesión 3 valoraciones subjetivas del test original, 2 valoraciones subjetivas para el señuelo y el audio comprimido. A partir de estos resultados es que se realiza la prueba de hipótesis y posterior análisis de datos.

Una vez planificada y delimitada la recolección de encuestas, se realizó este proceso de recopilación de datos en dos ocasiones diferentes para una muestra total de 25 sujetos (como se estipula en ITU-R BS.1534-1 (2015, p. 5) al realizar la prueba bajo condiciones técnicas adecuadas con sujetos con experiencia de escucha crítica, se requieren al menos 20 sujetos para poder extraer conclusiones fiables). La primera prueba subjetiva comprendida durante los días 7 de diciembre de 2019 al 10 de febrero de 2020, del cual se recolectaron 19 pruebas subjetivas, con una muestra de evaluadores en el que se destacan músicos, compañeros de ingeniería de sonido (población de personas con entrenamiento auditivo para la escucha crítica) que aportase a las pruebas al tener resultados significativos en la evaluación edl plugin. Se realizó una segunda recolección de datos entre el 10 de febrero y el 27 de febrero de 2020 con el propósito de completar 25 encuestas para tener mayor veracidad en el proceso de análisis subjetivo.

En relación con el marco normativo descrito en la sección 2.7 y 3.3, las condiciones de la medición, se llevaron acabo con unos auriculares profesionales Beyerdynamic DT 770 PRO (250 OMH) con estructura cerrada que presentan aislamiento fuerte para aplicaciones de control y monitoreo, para permitir desvincular variables relacionadas a la acústica propia del recinto donde se hizo la prueba; el recinto fue el salón de “Sonido en Vivo” en el segundo piso del edificio Guillermo De Ockham; se utilizó un computador Macbook Pro 2012 (Ram 16GB, procesador 2.5GHz Intel Core i5 sistema operativo OS X El Capitan) con el DAW Reaper y una interfaz de audio profesional de dos canales Focusrite Scarlett 2i4.

El proceso de evaluación por sujeto se realiza de la siguiente manera:

Actividad	Tiempo Realización
Bienvenida, contextualización y evaluación de prueba	3 minutos
Fase de evaluación Layer III	
Realización del primer entrenamiento	Determinado por el usuario, ya que este suministra la valoración después de haber comparado de acuerdo con el criterio propio
Descanso	2 minutos
Fase de evaluación Layer II	

Realización del segundo entrenamiento	Determinado por el usuario, ya que este suministra la valoración después de haber comparado de acuerdo con el criterio propio
Finalización Test Subjetivo	

Distribución demográfica de los encuestados:

Género	Ocupación		
	Ingeniero(a)	Músico(a)/productor(a)	Total
Masculino	9	15	24
Femenino	1	0	1

Los datos recolectados fueron tabulados en Excel, como se visualiza en la **Figura 4–14** estableciendo el formato de respuesta de la sesión, el proceso de análisis a la prueba de hipótesis es el test ANOVA cuyos resultados serán discutidos posteriormente en el siguiente capítulo.

Figura 4–14: Ejemplo 1 del uso de herramienta de análisis de Excel.

RESUMEN

Grupos	*Cuenta*	*Suma*	*Promedio*	*Varianza*
Columna 1	24	107	4,458333333	0,25906
Columna 2	24	104	4,333333333	0,23188
Columna 3	24	107	4,458333333	0,25906

ANÁLISIS DE VARIANZA

Origen de las variaciones	*Suma de cuadrados*	*Grados de libertad*	*Promedio de los cuadrados*	*F*	*Probabilidad*	*Valor crítico para F*
Entre grupos	0,25	2	0,125	0,5	0,60871114	3,129643983
Dentro de los grupos	17,25	69	0,25			
Total	17,5	71				

Luego se hizo uso de la herramienta de análisis estadístico de Excel, en el cual se agruparon los datos por factores determinados por la característica de estudio, audio original, audio señuelo y audio comprimido como es visible en la **Figura 4–15**. Todos los resultados de las pruebas subjetivas con sus respectivas hojas de cálculo son anexados al presente documento.

Figura 4–15: Ejemplo 2 del uso de herramienta de análisis de Excel.

Disposición Prueba	comprimido	original	original	original altera	original altera	original	comprimido
Valor	x	5	5	x	x	5	x
Media aritmética	3,68	4,4	4,6	3,8	3,6	4,52	3,8
Desviación Estándar	0,69041051	0,57735027	0,5	0,70710678	0,70710678	0,58594653	0,7132825
Varianza	0,47666667	0,33333333	0,25	0,5	0,5	0,34333333	0,50877193
Moda	3	4	5	4	3	5	4

5. Capítulo V. Presentación de Resultados y Análisis

5.1 Presentación y análisis de resultados objetivos

Como se mencionó en la subsección 3.3, ambos sistemas de compresión MPEG-Layer II y III, se pusieron a prueba en dos frecuencias de muestreo, 44.1k y 48k Hz, de las cuales en cada parámetro de medición (Función de Transferencia, Densidad de Potencia, Distribución estadística y THD) se hizo una distinción a partir de diferencias numéricas entre los esquemas para adquirir una comparación de sus desempeños. Según lo explicado en las secciones 4.2 y 4.5 el modelo MPEG-Layer II es la base algorítmica que se complementa y extiende usando la transformada MDCT en el modelo MPEG-Layer III.

5.1.1 Espectros de señales de salida y Funciones de Transferencia

Como es visible de la Figura 3–2 de la sección 3.3, la magnitud del espectro $X(f)$ original de la señal *chirp* $x_0[n]$ de entrada es enteramente plano/constante alrededor de 56.5dBFS desde su origen hasta su frecuencia de Nyquist de diseño. Esta señal de entrada frente a los resultados de medición en la Figura 5–1 y Figura 5–2 muestran una deformación de la respuesta en frecuencia original en las Funciones de Transferencia $H(f)$ y Espectros de Salida $Y(f)$ para ambos sistemas (Layer II y III) a ambas frecuencias de muestreo (44.k y 48k Hz).

En general, en los cuatro resultados, se presenta un primer incremento progresivo (lóbulo) desde los 20Hz (30Hz para 44.1k Hz) hasta los 70 Hz, que muestra un máximo en 45Hz (en promedio) de +4dBFS para $|Y(f)|$ y +6dBFS $|H(f)|$. También, recordando la forma funcional constante que se presenta en la Figura 3–2 de la magnitud del espectro de $X(f)$ en escala lineal de las abscisas, se nota un comportamiento similar "plano" en las Figura 7–11 y Figura 7–12 (Anexo 9), que muestra en escala lineal el eje de frecuencias. Aunque este comportamiento aparentemente plano se de en el espectro de todos los resultados, aplicando un histograma para cada resultado de $H(f)$ en dBFS, los resultados varían aproximadamente -7 y $+5.4$ dBFS alrededor de los 0 dBFS, y para $|Y(f)|$ en dBFS, los resultados varían de igual forma de forma estimada -7 y $+5.4$ dBFS alrededor de -56 dBFS (lo anterior se visualiza en la Figura 5–4).

$$|\Delta\text{mag}\{H(f)\}| = \left| |H_{\text{Layer 2}}(f)| - |H_{\text{Layer 3}}(f)| \right| \tag{5.1}$$

Para observar la distinción que hay del Layer II respecto al III (dada la introducción de la MDCT en el modelo base del PQMF), se calculó la diferencia de magnitudes en escala lineal de $|H(f)|$ a ambas frecuencias de muestreo -Ecuación (5.1)-, la visualización de esta diferencia está en la Figura 5–3, aquí se muestra que a 44.1k Hz la mayor diferencia existente es en baja frecuencia (desde los 10Hz hasta los 70Hz, en la Figura 7–14 del Anexo 9; esta diferencia revela un aporte de -17dBFS de forma plana por parte del Layer III), para 48k Hz se da un comportamiento similar en baja frecuencia en magnitud pero funcionalmente no plano si no mas bien ascendente desde los -20dBFS. A partir de los 100Hz ascendiendo hasta los 350 Hz las diferencias son irregulares y significativas en magnitud, llegando hasta los 0 dBFS, el Layer II presenta una mayor distorsión armónica en este rango, mientras el Layer III presenta menos distorsión armónica hasta los 400 Hz (esto también es notorio en la Figura 5–2). Este fenómeno no ocurre entre Layers a 44.1k Hz. Finalmente, a $f_s =$ 44.1k Hz desde los 500 Hz y los 1k Hz a $f_s =$ 48k Hz la diferencia entre Layers empieza a ser cada vez menor hasta llegar a ser despreciable (niveles más bajos de cuantización) como revelan todas figuras de resultados de Función de Transferencia.

Al ser estos resultados magnitudes, implica que el rango de valores lineales están delimitados entre 0 y 2 (en vez de -1 y 1), y en escala logarítmica entre "-inf" y 6.02 dBFS, esto es importante mencionarlo ya que en la Figura 5–3 y Figura 7–13 del Anexo 9, los resultados de ambos Layer II y III a 48.k Hz en baja frecuencia revelan una serie de muestras que exceden la amplitud de 2, la banda de frecuencia que corresponde a estas amplitudes está entre los 10-70 Hz como es visible en la Figura 7–14, lo cual implica distorsiones por recorte, igualmente ocurre en las amplitudes de $\left|H_{\text{Layer 2}}(f)\right|$ entre los 200 Hz y 300 Hz a diferencia de $\left|H_{\text{Layer 3}}(f)\right|$, cuyas amplitudes entre los 100 y 1k Hz están por debajo de 2.

Figura 5–1: Función de Transferencia y Espectro de Salida de ambos Layer a 44.1 kHz.

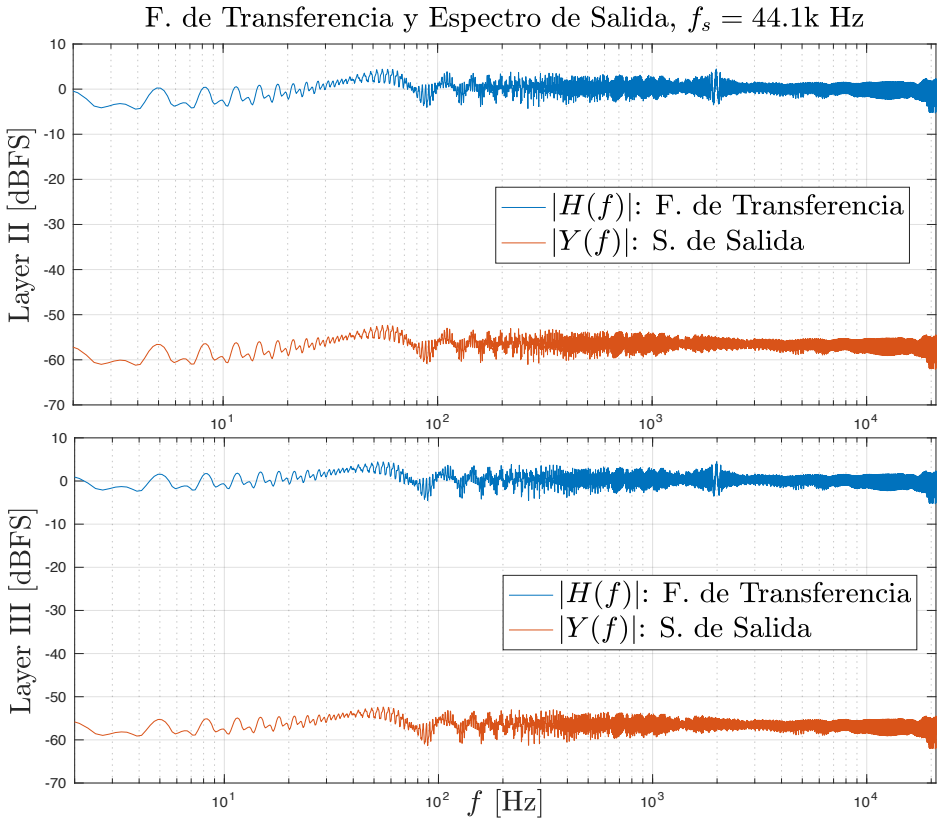


Figura 5–2: Función de Transferencia y Espectro de Salida de ambos Layer a 48 kHz.

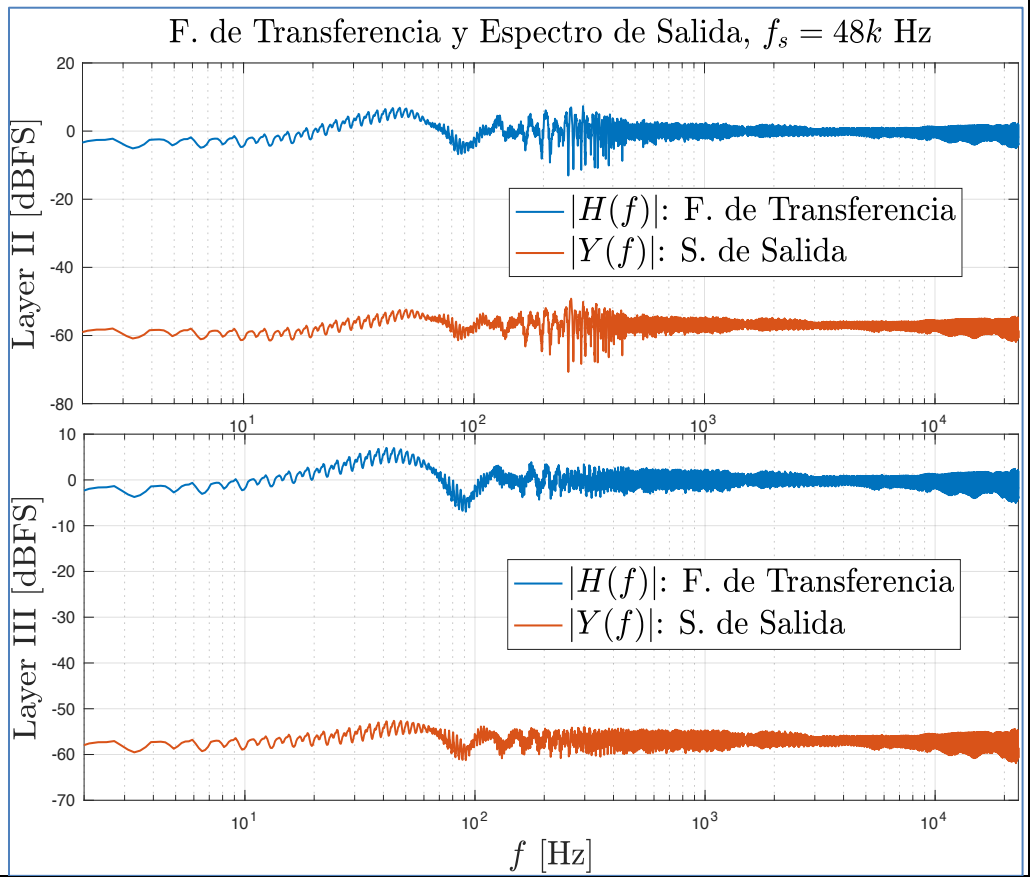


Figura 5–3: Diferencia de Magnitudes de la Funciones de Transferencia de ambos Layer a 44.1 k y 48k Hz.

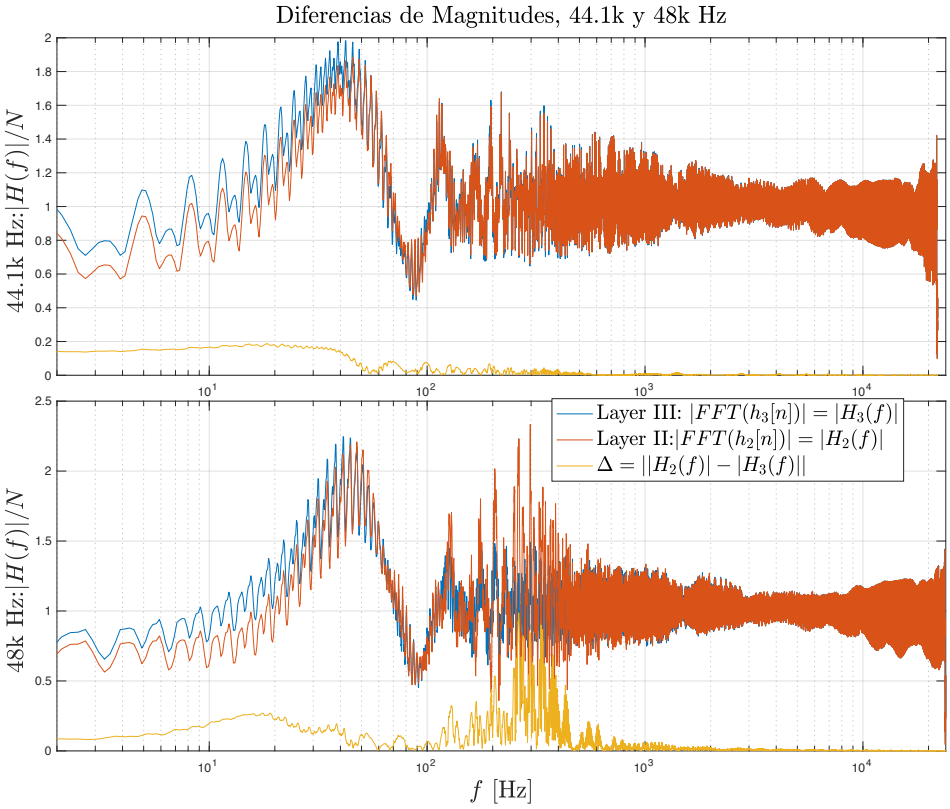
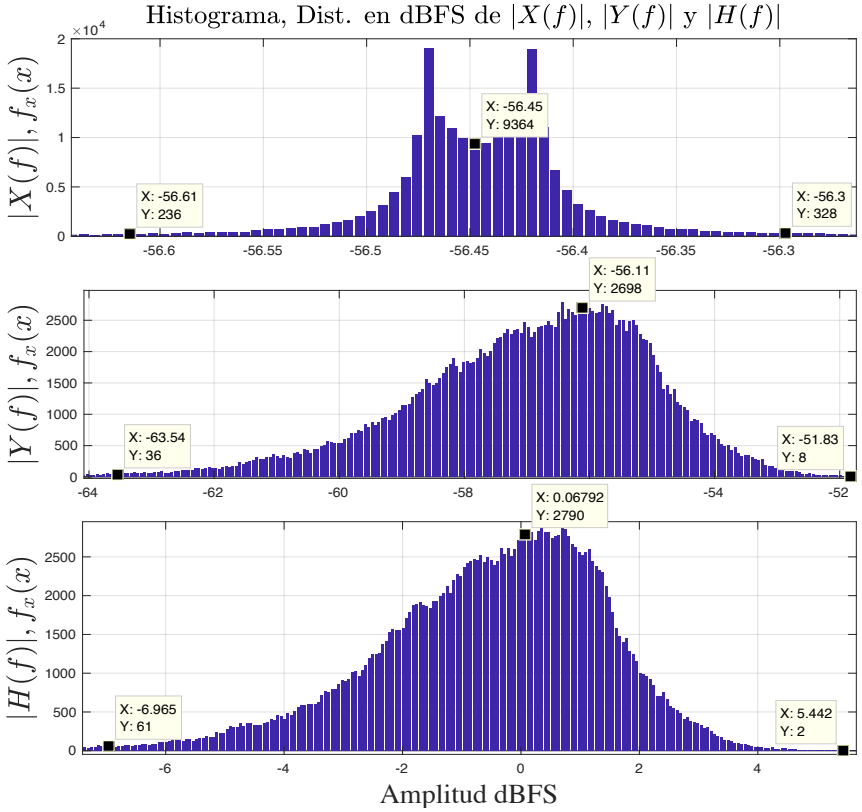


Figura 5–4: Distribución genérica promedio de los resultados de $H(f)$ y $Y(f)$ en dBFS.



5.1.2 THD y SNR de las 32 bandas del PQMF

En la Figura 5–5 y Figura 5–6 se visualizan los porcentajes de Distorsión Armónica Total para cada una de las 32 bandas del PQMF de los Layer II y III (los resultados numéricos se encuentran en la Tabla 7-5 y Tabla 7-6 del Anexo 9).

Hay un comportamiento general para ambos Layer en ambas frecuencias de muestreo, el cual es el aumento casi lineal de THD por banda, es decir en alta frecuencia hay mayores porcentajes de THD respecto a la frecuencia media, y en la frecuencia media hay mayores porcentajes de THD respecto a la baja frecuencia. A 48k y 44.1k Hz los porcentajes de THD alcanzan un factor máximo de 10^{-3} hasta la 5ta banda (3750 y 3445 Hz) y un factor máximo de 10^{-1} hasta la banda 19 (14,250 y 13,092 Hz), las bandas consecutivas a la 19, muestran valores mayores al 0% de THD, hallando valores considerables del 12% y 15% en las bandas 26 (19,500 a $f_s = 48$k Hz y 17,916 Hz a $f_s = 44.1$k Hz) y 32 (24,000 Hz a $f_s = 48$k Hz). Un resultado notorio es la nulidad de THD en las bandas 8, 16 y 24 siendo 8 y 16 factores enteros del número de bandas del PQMF (32), pero para la cual la relación entre estas 3 bandas se da más bien como factores enteros de la banda 8. Estos resultados se dan para los Layer II y III a ambas frecuencias de muestreo.

Al hallar el aporte de THD de la MDCT en el Layer III respecto al Layer II, para 44.1k Hz la mayor diferencia de porcentajes se dio en la banda 6 (4500 Hz), el resto de las diferencias de porcentajes de THD para las demás bandas se encuentran por debajo de magnitudes de 10^{-5}. A 48k Hz se encuentran mayores diferencias entre los Layer II y III (fenómeno que igualmente se dio al comparar las Funciones de Transferencia en la sección anterior). Las mayores diferencias están en las bandas 3, 5, 11, 16, y 32.

Los resultados de Relación Señal a Ruido (SNR) para cada una de las 32 bandas del PQMF se dan en la Figura 5–7 y Figura 5–8, numéricamente se encuentran en la Tabla 7-7 y Tabla 7-8 del Anexo 9) (se prefirió dejar toda la extensión numérica de cada resultado de SNR para visualizar las diferencias entre ellas). En las figuras de estos resultados se ve una forma funcional bien definida de los SNR en todas las bandas para ambos Layer a ambas frecuencias de muestreo, estas son un par de curvas cóncavas alrededor de un máximo de 97dBFS en la banda central (16) en la cual los mínimos son en las bandas 11, y 25. En general los 4 resultados, se da el comportamiento que para todas las primeras bandas hasta la número 16, existe menor presencia de niveles de ruido. En las bandas donde se presenta menor SNR, a 48k Hz para la banda 11: Layer 2: 13 dBFS y Layer 3: 19.2 dBFS, banda 25: Layer 2: y Layer 3: 6.5 dBFS. A 44.1k Hz para la banda 11: Layer 2: y Layer 3: 19.2 dBFS, banda 25: Layer 2: y Layer 3: 6.5 dBFS.

La diferencia de SNR entre el Layer II y III fue hallada con la Ecuación (5.2), en las figuras anteriormente mencionadas, los valores semejantes de SNR de ambos Layer llevan a $\Delta_{\mathrm{SNR}}(f)$ a valores negativos (aquellas diferencias cuyo valor son mayores a 0 y menores 1). De nuevo a $f_s = 48$k Hz en más bandas respecto a los resultados a 44.1k Hz se encuentran mayores diferencias.

$$\Delta_{\mathrm{SNR}}(f) = 10\log_{10}\left(\left|10^{-\left(\frac{SNR_{\mathrm{L.2}}(f)}{10}\right)} - 10^{-\left(\frac{SNR_{\mathrm{L.3}}(f)}{10}\right)}\right|\right)\ [\mathrm{dBFS}] \qquad (5.2)$$

Figura 5–5: Porcentaje de THD en las 32 bandas del PQMF en 44.1k Hz.

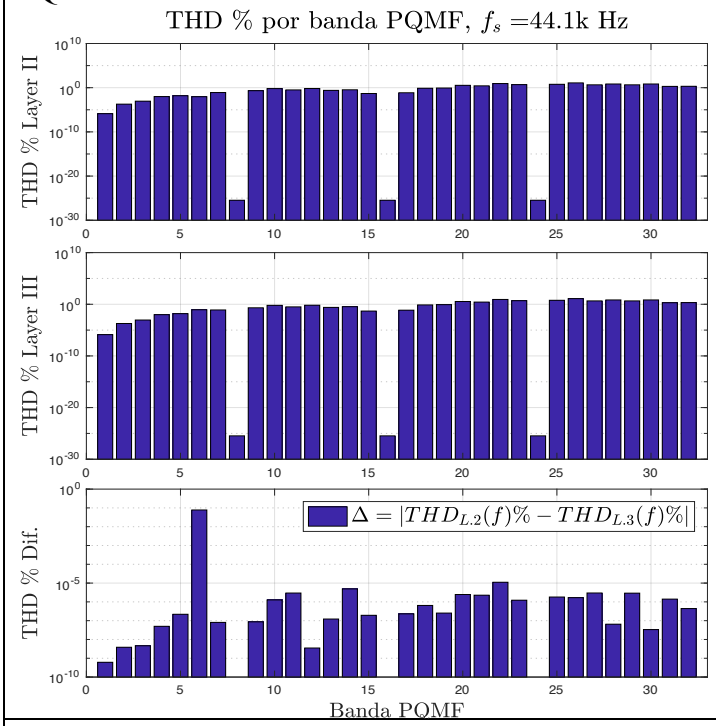


Figura 5–6: Porcentaje de THD en las 32 bandas del PQMF en 48k Hz.

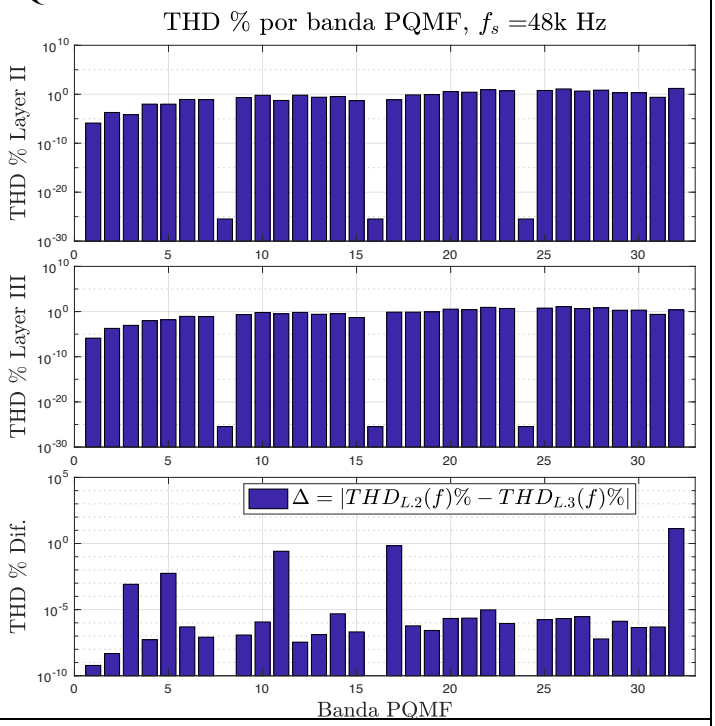


Figura 5–7: SNR en dBFS en las 32 bandas del PQMF a 44.1k Hz.

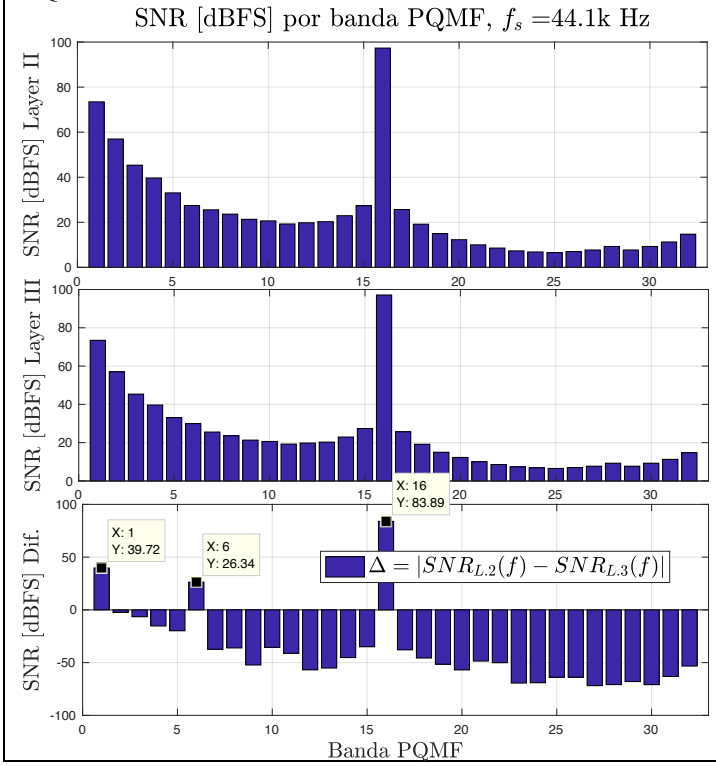
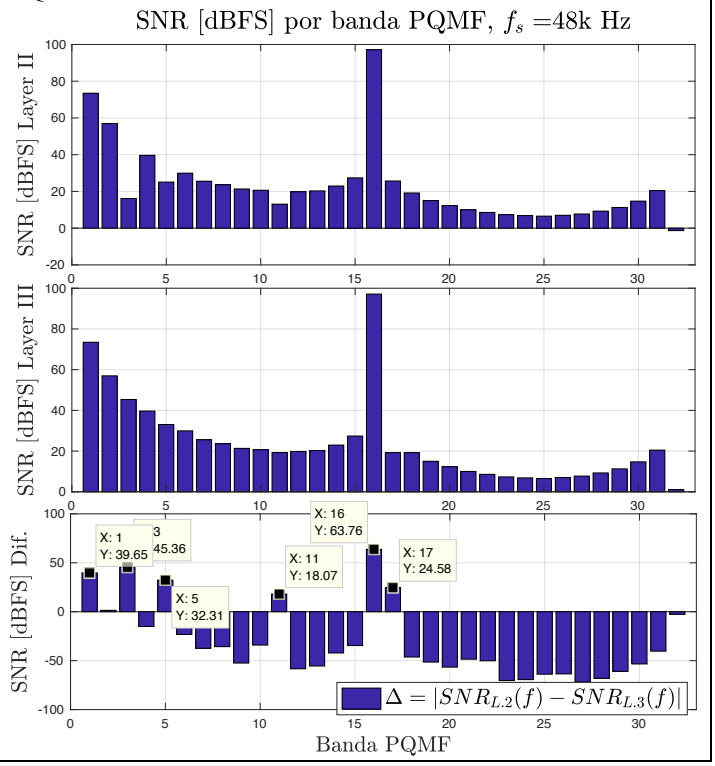


Figura 5–8: SNR en dBFS en las 32 bandas del PQMF a 48k Hz.



5.1.3 Parámetros estadísticos: Densidad de Potencia Espectral (PSD) y Autocorrelación

Al introducir en los sistemas (Layer II y III) una señal aleatoria, caracterizada como el resultado de un proceso estocástico, se pretendió observar si su función de distribución y su correlación inicial seguían presentando sus propiedades una vez dicha señal saliera procesada por ambos sistemas. Se ha visto con anterioridad que el esquema de compresión Layer II se presenta como una base algorítmica respecto al esquema Layer III, que, para la síntesis a parte de los modelos de cuantización, el Layer III añade frente al Layer II, la transformada MDCT. Es importante resaltar lo anterior para esta sección de análisis, ya que una parte de los motivos de la utilización de la MDCT, fue generar un análisis y **síntesis** de las líneas de tiempo de los PQMF, es decir, de estos resultados se puede evaluar qué tan fiel es la reconstrucción de las muestras de los filtros PQMF a través de la MDCT observando si varía la correlación estadística con el uso exclusivo de los PQMF en el Layer II frente al uso extra de la MDCT en el Layer III.

Los resultados de la Figura 5–9 y Figura 5–10 muestran que para ambos Layer (II y III) en cualquiera de las dos frecuencias de muestreo (44.1k y 48k Hz), sus resultados son idénticos al procesar la señal WGN, no obstante, al comparar y comprobar su diferencia (confirmando que la misma señal se procesó a través de sistemas distintos) se hizo la resta de los valores del WGN procesado por el Layer II respecto al Layer III, (es decir: $\mathrm{WGN}_{\mathrm{L.2}}[n] - \mathrm{WGN}_{\mathrm{L.3}}[n]$, así como la prueba booleana que indica "1" si un arreglo es igual a otro o "0" si son distintos en Matlab: *isequal*($\mathrm{WGN}_{\mathrm{L.2}}[n], \mathrm{WGN}_{\mathrm{L.3}}[n]$) el cual resultó en "0") observando que los efectos introducidos en la dispersión de los datos y la correlación estadística gracias a la MDCT existen, graficando esta diferencia en la Figura 5–11. En la Figura 5–11, es claro que las señales de salida de ambos Layer no son réplicas literales una de la otra, pero sí son idénticas, y por tanto, su diferencia para todo tiempo n, no es cero, si no mas bien valores muy cercanos entre ellos y además próximos a cero, causando que su varianza y desviación estándar sean cero. Así, las características de distribución de la PSD no varían respecto al WGN original, ya que aquí se muestra que el aporte de distribución estadística de la MDCT en todo el espectro ronda sobre magnitudes de potencias muy bajas, llegando a los límites más bajos de cuantización posible que permite la profundidad/resolución en bits del sistema computacional, como se muestra en la Figura 5–11 derecha "-inf dBFS" o mirando los valores de amplitud en dBFS, los cuales rondan alrededor de los -145 dBFS.

Recordando las características iniciales de la señal de Ruido Blanco Gaussiano (WGN) vistas en la Figura 3–3, Figura 3–4 y Figura 3–5, se afirmó que su PSD es plana en el espectro (tiende a ser constante), su varianza σ^2 de diseño es igual a 0.04, su media aritmética μ_x es igual a -0.0012 a $f_s = 44.1\text{k}$ Hz y 0.0004 a $f_s = 48\text{k}$ Hz, su curva de normalidad se presenta como una aproximación de la curva Gaussiana y su Autocorrelación $R_{xx}(\tau)$ cumple con ser igual a $\sigma^2 \cdot \delta(\tau)$ para todo desfase temporal τ. Es visible en los resultados aquí expuestos (Figura 5–9 y Figura 5–10) que la media aritmética μ_x estimada no varió significativamente respecto al valor original de diseño 0, varió en un error absoluto de 7.8×10^{-3}, para el muestreo a 44.1k Hz y 0 para el muestreo en 48k Hz. La varianza

estimada σ^2 varió 1×10^{-3} para 44.1k Hz y 0.9×10^{-3} para a $f_s = 48$k Hz. Los resultados anteriores permiten a ambos sistemas generar señales cuyo comportamiento no varíe en su función de Normalidad Gaussiana y su Autocorrelación no varíe de $\sigma^2\delta(\tau)$ como se aprecia en la Figura 5–10.

Figura 5–9: Visualización de las señales de salida WGN en los sistemas Layer II y III para 44.1k y 48k Hz. WGN en el tiempo (gráficas izquierdas) y en la frecuencia (gráficas derechas).

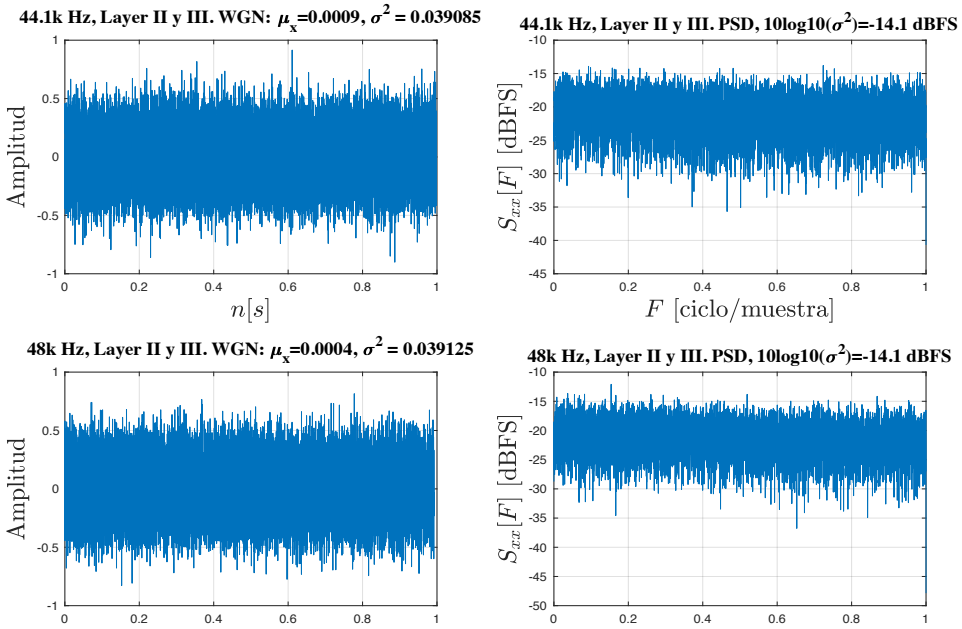


Figura 5–10: Visualización características estadísticas de las señales de salida WGN de los sistemas Layer II y III para 44.1k y 48k Hz. Curvas de normalidad a f_s/32 bins (gráficas izquierdas) y Función de autocorrelación (gráficas derechas).

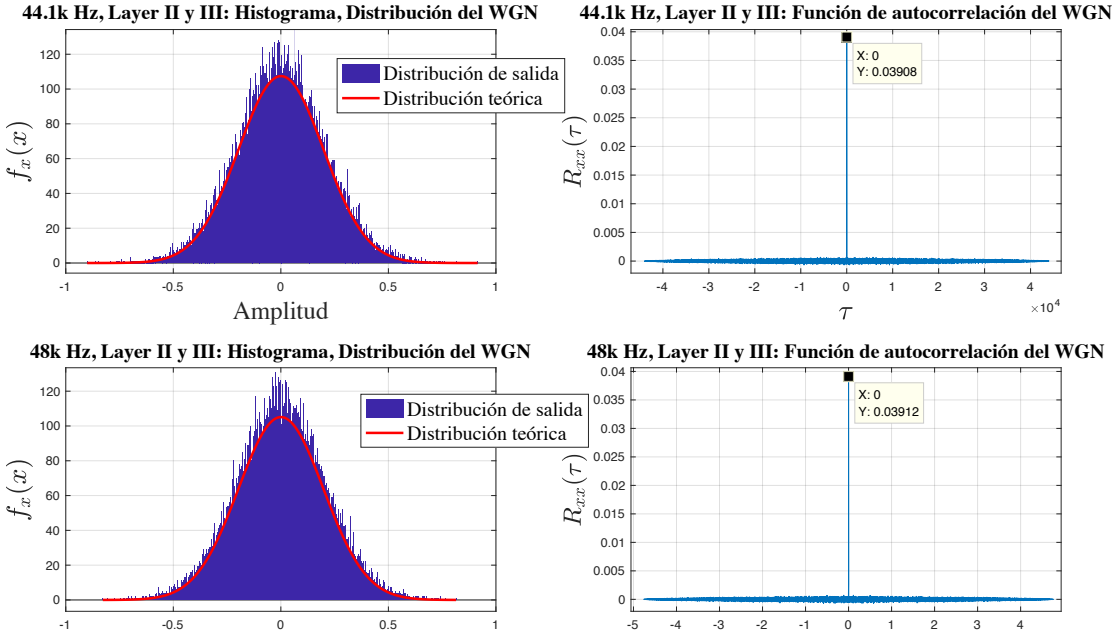
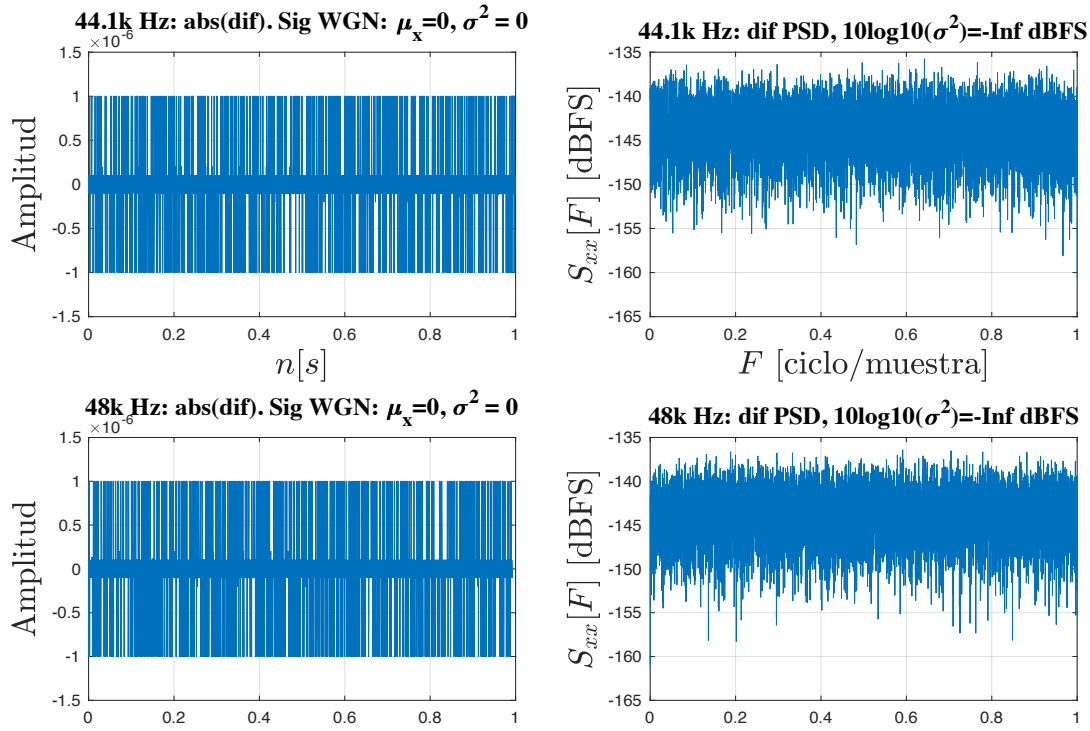


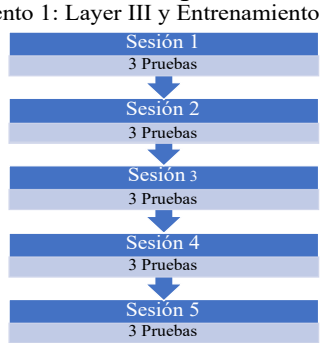
Figura 5–11: Visualización de la diferencia de las señales de salida WGN en los sistemas Layer II y III para 44.1k y 48k Hz. WGN en el tiempo (gráficas izquierdas) y en la frecuencia (gráficas derechas).



5.2 Resultados y análisis de las pruebas perceptuales MUSHRA (ITU. 1534)

En esta sección se discute, evalúa y analizan los resultados obtenidos de las pruebas subjetivas, una vez concluido el proceso de evaluación experimental del método de evaluación de codecs MUSHRA (ITU-R BS.1534-1, 2015) como extensión de (BS.1116-1, 1997), como se ilustra en la Figura 5–12, disponiendo de cinco sesiones dentro de las cuales hay 3 pruebas o preguntas de evaluación subjetiva, estas cinco pruebas se elaboraran en 2 laboratorios/situaciones de prueba, en el primero se realizó la evaluación del plugin comprimiendo con el esquema Layer III y en el segundo experimento se desarrolló el test con el Layer II. Las disposiciones de los 5 entrenamientos se dan al final de esta sección.

Figura 5–12: Esquema de implementación del proceso de las pruebas subjetivas.



De acuerdo con el propósito mismo del método de evaluación, en el que se presenta un audio original, uno modificado y uno comprimido en orden establecido con el diseño de la prueba, se busca analizar la comparación subjetiva y de percepción de elementos como distorsión, componentes en frecuencia y demás degradaciones posibles, que se puedan dar debido al proceso de codificación. Esta equiparación subjetiva brindada por el usuario permite confrontar el funcionamiento del plugin en referencia a señales dispuestas de forma controlada, para así establecer un grado calificativo subjetivo neto que puede ser cotejado con el funcionamiento del plugin en sus diferentes Layers. Debido a que se está evaluando una característica similar en diferentes pruebas (la percepción subjetiva del audio original, comprimido y señuelo), se somete a una hipótesis (ya que la recolección de estos datos fue impuesto a diferentes experiencias que buscan ser contrastadas, utilizando una muestra de 25 sujetos y resultados) para determinar la significancia del valor obtenido durante las encuestas. Determinando una hipótesis nula en el caso en el que no hay diferencia o efecto entre los datos, se determina un nivel de confianza $\alpha = 0.05$, para determinar si la hipótesis es acertada o rechazada.

Las hipótesis para probar constan de: Una hipótesis nula en la que la media de los originales es mayor e igual a 4; si se rechaza esta hipótesis se pasa a hipótesis alternativa correspondiente a la media de calificación donde el valor inferior va a rechazar dicha hipótesis. Con el motivo de análisis se determina un parámetro estadístico de prueba de distribución F ilustrada en la Figura 5–13 propia de la prueba de hipótesis ANOVA ("Analysis of Variance"). Teniendo entre diferentes posibilidades, determinadas como sugerencia la normatividad ITU-1534 en la siguiente tabla:

Prueba de hipótesis	Estadístico de prueba
Prueba Z	Estadístico Z
Pruebas t	Estadístico t
ANOVA	Estadístico F
Pruebas de chi-cuadrada	Estadístico de chi-cuadrada

Figura 5–13: Gráfica de distribución tipo F sub α. Para el análisis ANOVA.

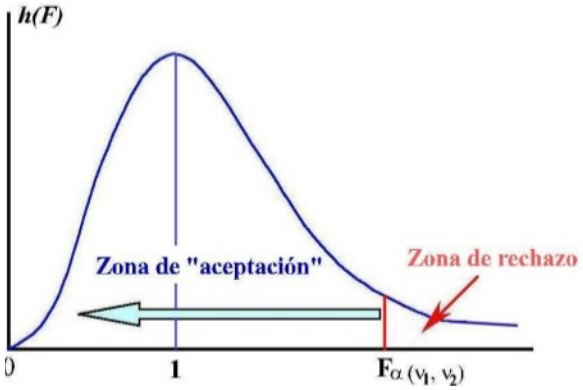


Imagen tomada de Spiegel & Schiller (2007).

El estadístico de prueba F correspondiente está representada en la distribución Fischer, útil debido a sus grados de libertad permitidos y aplicación a muestras reducidas; además sutilizado en dos poblaciones diferentes (las poblaciones se asumen diferentes debido que fueron comparadas respecto a un audio diferente). Los resultados de las siguientes tablas para los Audios Originales, Comprimidos y Originales Alterados, de los resultados de valor F y valor crítico vs sus curvas de distribución, se pueden visualizar en la **Figura 5–14**.

Distribución audios Originales –Layer III			Distribución audios Originales – Layer II	
Experimento	Valor F calculado	Valor crítico (tabla)	Valor F calculado	Valor crítico (tabla)
1	0,55714286	3,12390745	2,9194313	3,12390745
2	0,44709898	3,12390745	3,7761194	3,12390745
3	1,11013216	3,12390745	0,82014388	3,12390745
4	0,96771621	3,12390745	0,82014388	3,12390745
5	0,67826087	3,12390745	2,0458716	3,12390745

En la evaluación de los audios originales en la sesión de entrenamiento donde se utiliza el plugin Layer III, ninguna hipótesis fue rechazada. De las cuales destacamos el experimento 4 en el cual se aproximó a la distribución 1. En la evaluación de los audios originales en la sesión de entrenamiento donde se utiliza el plugin Layer II, ninguna hipótesis fue rechazada. De las cuales destacamos el experimento 5 en el cual se aproximó a la distribución 1.

Teniendo en cuenta los audios comprimidos:

Distribución audios Comprimido–Layer III			Distribución audios Comprimido– Layer II	
Experimento	Valor F calculado	Valor crítico (tabla)	Valor F calculado	Valor crítico (tabla)
1	2,36363636	4,04265213	1,64102564	4,04265213
2	2,41509434	4,04265213	1	4,04265213
3	0,7804878	4,04265213	0,33962264	4,04265213
4	0,7804878	4,04265213	0,405845	4,04265213
5	0,30508475	4,04265213	0,305275	4,04265213

Al comparar la evaluación subjetiva de los audios comprimidos, se encontró que las hipótesis son aprobadas dando un mayor grado de validez en los experimentos realizados en el layer III, con valores cercanos mayor distribución, sin embargo, teniendo una tendencia de F hacia 0.3 en el layer II. Respecto al audio utilizado como señuelo:

Distribución audios Originales Alterados– Layer III	Distribución audios Originales Alterados– Layer II

Experimento	Valor F calculado	Valor crítico (tabla)	Valor F calculado	Valor crítico (tabla)
1	0,38567475	4,04265213	0,28345	4,04265213
2	0,28508487	4,04265213	0,44281	4,04265213
3	0,30128495	4,04265213	0,14458	4,04265213
4	0,29007473	4,04265213	0,12345	4,04265213
5	0,30508475	4,04265213	0,22486	4,04265213

Las hipótesis son aprobadas, sin embargo, las distribuciones en comparación con las originales son lejanas a 1 y son equiparables entre los Layer III con una tendencia a 0.3 y alejándose de esta tendencia en las evaluaciones realizadas en Layer II, destacando una más perceptible degradación del audio. Una vez aprobadas las hipótesis, se categorizan los promedios de las sesiones destacando de estas el análisis del audio comprimido puesto que es el tema de análisis principal.

Figura 5–14: Gráfica de distribución tipo F, con valores críticos, Para el análisis ANOVA.

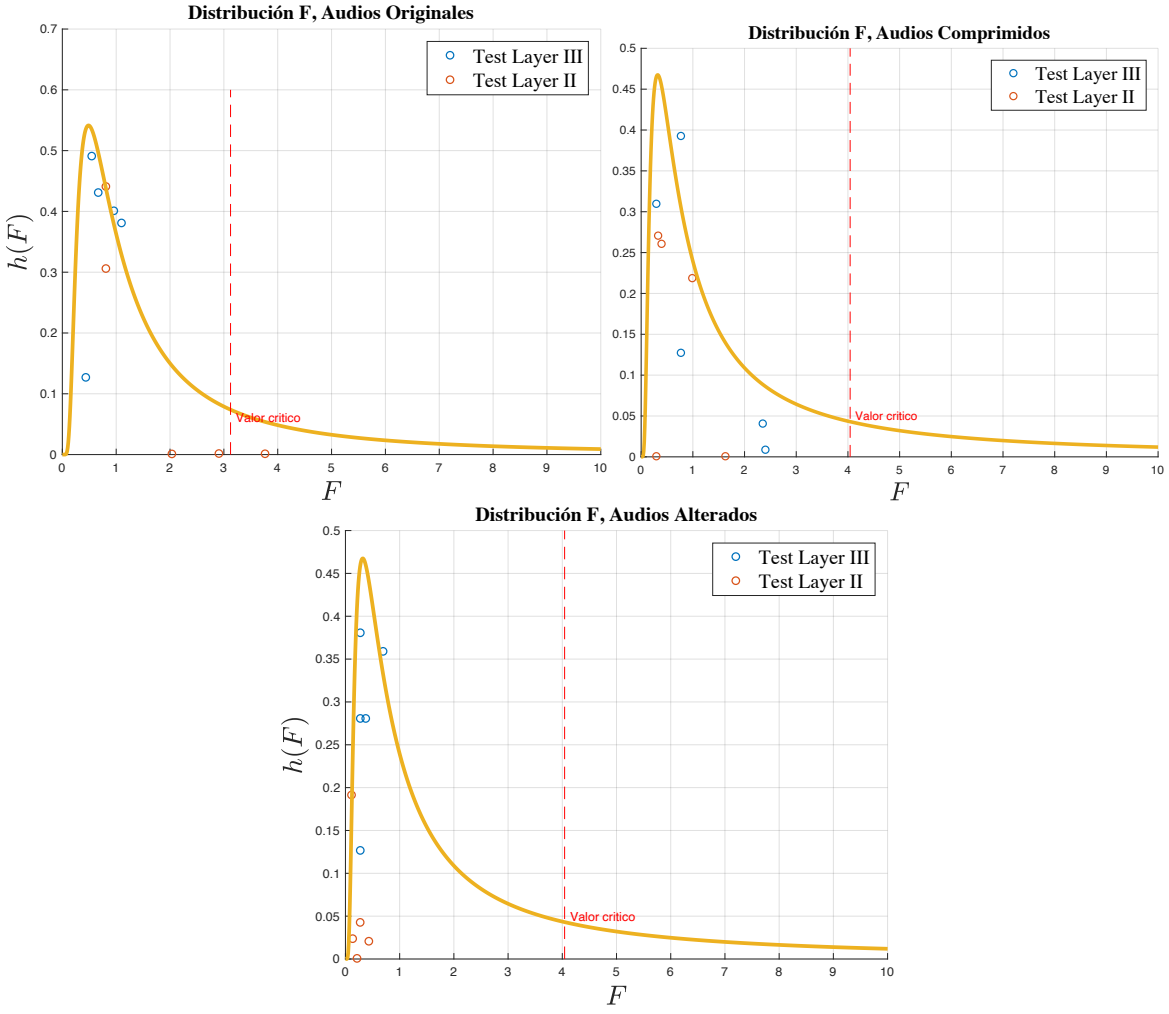
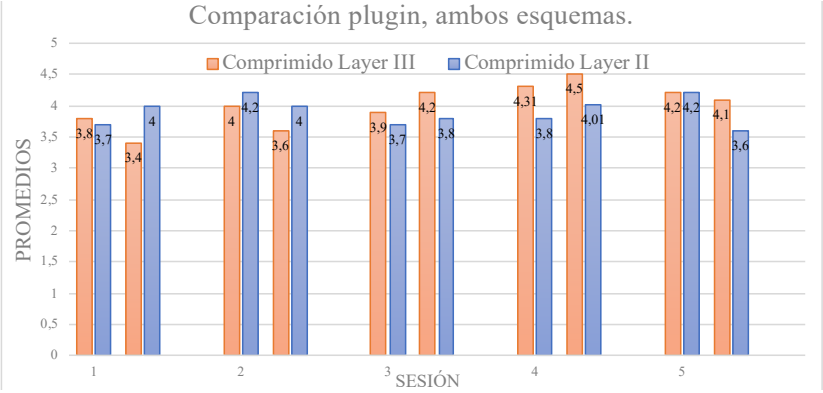
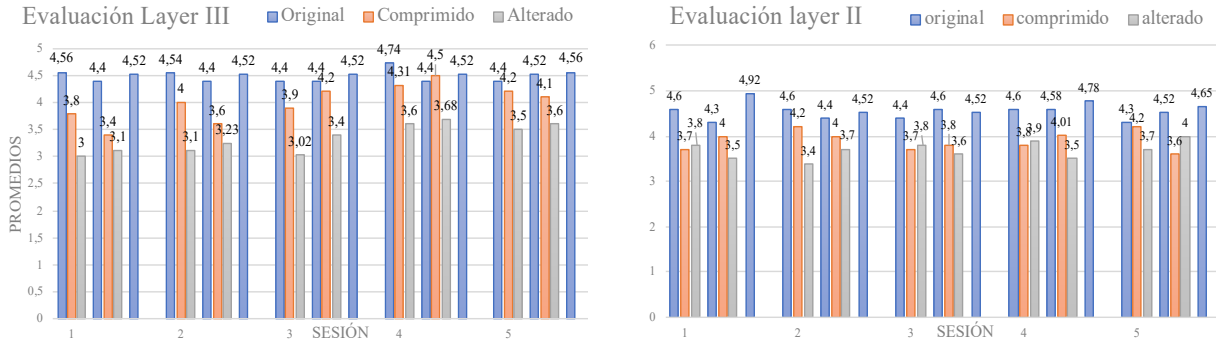


Figura 5–15: Comparación de promedios de las valoraciones de ambos Layer II y III.



Como se visualiza en la **Figura 5–15**, se encontró de esta comparación de audios comprimidos entre Layer diferencias menores a 0.7 en la percepción, sin embargo, en sesiones como la 4 y 5, donde hay por lo menos cinco valores decimales de diferencia en audios en los cuales se utilizó una frecuencia de muestreo de 48kHz y de 44.1k Hz, destacando la mayor percepción de diferencia en la sesión de 48k Hz, favoreciendo la valoración del funcionamiento del Layer III. Canciones caracterizadas por el cambio de secciones representadas en el cambio de ventanas de los diferentes layers, caso particular en el que se destaca la posible apreciación de los evaluadores.

Figura 5–16: Comparación de la compresión con Layer III – Señuelo vs Layer II- señuelo



Al recopilar la información de los valores promedio en las pruebas y las sesiones, encontramos que hay una mayor diferencia de percepción de evaluación subjetiva en el Layer III respecto al audio señuelo, denotando un funcionamiento de mejor apreciación en calidad sonora para el oyente; comparado con la comparación realizada entre el Layer II y el audio señuelo, en el que hay casos precisos donde el audio señuelo llega a ser percibido con mayor agrado (sesiones 3 y 4 de la **Figura 5–16** derecha) que el audio comprimido. Respecto a las evaluaciones propias durante sesiones se toman los experimentos mejor validados por la prueba de hipótesis, para el posterior análisis. Al comparar el funcionamiento de los Layers respecto al original en la Figura 5–18 se hallaron mejores valoraciones, para el Layer III siendo más cercanas a la percepción original del audio.

Figura 5–17: Gráficas de valoraciones de Layers vs audios originales.

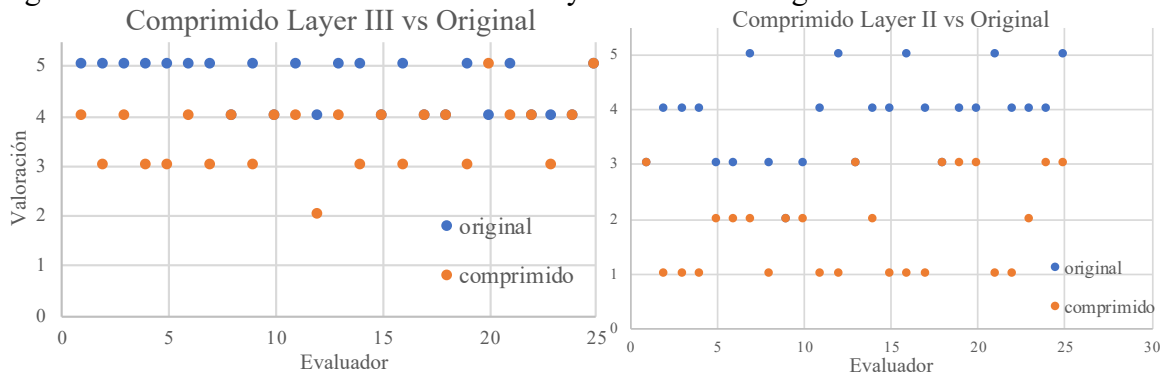
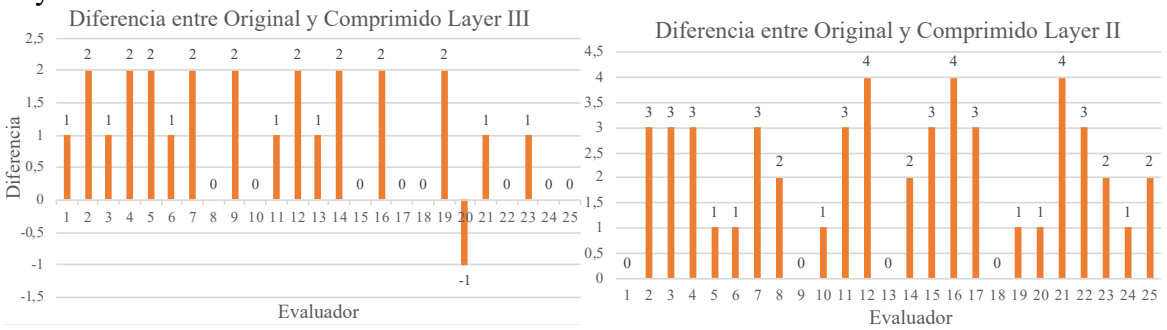


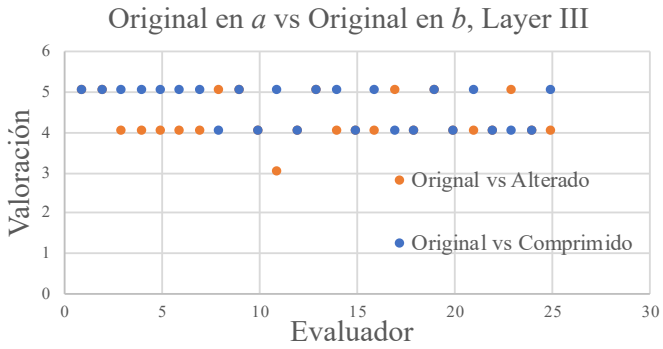
Figura 5–18: Gráficas de diferencias de *a* y *b* para audios originales y comprimidos por los Layer.



Encontrando menor diferencia (**Figura 5–18**) entre ambas selecciones para las pruebas correspondientes al Layer III. Otra manera de comparar la precepción del audio alterado respecto al comprimido es tener en cuenta la influencia de la percepción respecto al audio original. Es decir, al tomar de la prueba Original vs alterado y tomar el valor calificado como original (cercano a 5), y al realizar la selección del valor calificado como original de la prueba original vs comprimido.

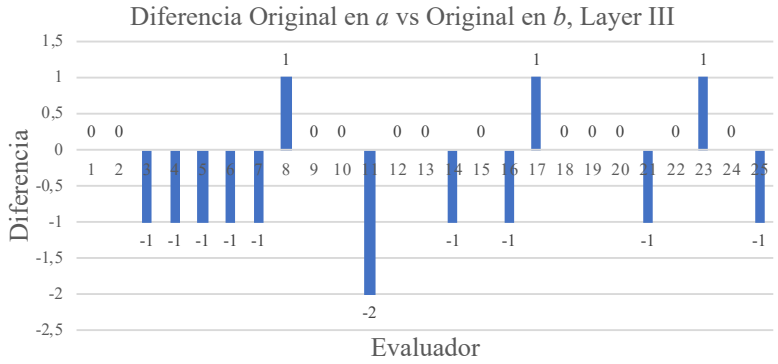
Realizando la siguiente operación (original vs alterado) − (original vs comprimido), se espera un comportamiento en el que el valor para (original vs comprimido) sea menor y el valor de (original vs alterado) sea mayor, es decir que, al aplicar esta operación de resta, si se obtiene un resultado negativo el comportamiento del audio señuelo fue mejor al del audio comprimido teniendo en cuenta la influencia del audio original. Cabe resaltar que la diferencia entre entrenamientos radica en el uso del Layer III y Layer II (en ese orden determinado).

Figura 5–19: Valoración de los resultados originales en *a* vs los originales en *b* por para el Layer III.



Entre más bajo sea la valoración del original vs el comprimido visto en la **Figura 5–19**, implica que el evaluador dio un puntaje de evaluación menor al correspondiente al audio original y de alta calidad; significando esto que la percepción de este audio fue degradada al compararse por la presencia del audio comprimido.

Figura 5–20: Gráficas de diferencias para audios originales en *a* y en *b* y comprimidos por el Layer III.



Esta mejor percepción del audio codec es representada en la diferencia de percepción de audio original vista en la Figura 5–20, donde una diferencia nula significa/representa una evaluación objetiva y correcta de selección del audio original, una diferencia negativa resuelve una percepción de mayor calidad del señuelo. Se observa en la Figura 2–20 que la degradación que aporta el Layer II al audio original es menor e, es decir, el original sigue sonando en mayor calidad respecto al codec (la codificación no es comparable en nivel de percepción sonora agradable, al punto de hacerlo comparable con el formato sin compresión), se encuentran mayor degradación nula (diferencia igual a cero) y una mayor percepción agradable del señuelo, representado en mayores diferencias negativas vistas en la Figura 5–22.

Figura 5–21: Valoración resultados originales en *a* vs originales en *b* por para el Layer II.

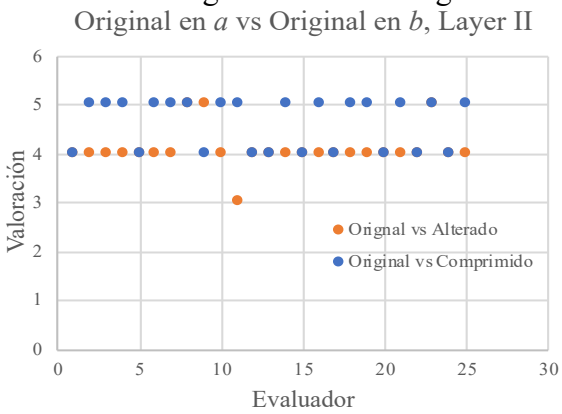
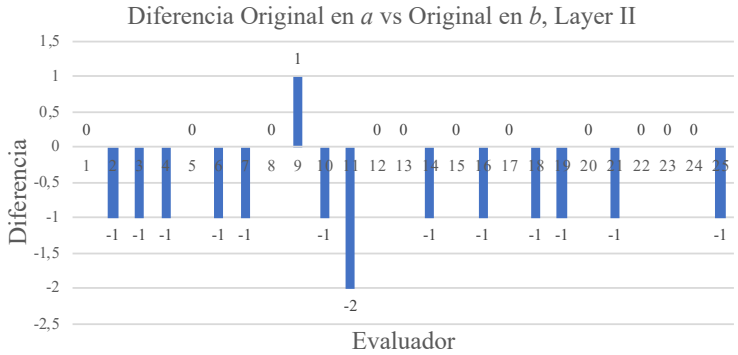


Figura 5–22: Valoración de los resultados originales en *a* vs los originales en *b* por para el Layer II.



6. Capítulo VI. Conclusiones y recomendaciones

6.1 Conclusiones de los Parámetros Objetivos

6.1.1 Conclusiones acerca de los Espectros de Salida y Funciones de Transferencia de ambos sistemas

Los resultados y análisis en la sección 5.1.1 muestran que los espectros de salida $Y(f)$ y funciones de transferencia $H(f)$ de ambos sistemas Layer II y III, a las frecuencias de muestreo de 44.1k y 48k Hz, han sido distorsionados esencialmente por los filtros PQMF, dado que el proceso de análisis MDCT-IMDCT en el Layer III ha generado algunas leves variaciones en la forma funcional de $\left|Y_{\text{Layer 2}}(f)\right|$ y $\left|H_{\text{Layer 2}}(f)\right|$ comparado con $\left|Y_{\text{Layer 3}}(f)\right|$ y $\left|H_{\text{Layer 3}}(f)\right|$. Lo anterior también demuestra que, en la práctica el método de síntesis a través de Overlap & Add/Save con la IMDCT de los coeficientes de espectrales del PQMF permiten una reconstrucción fiable de señales. No obstante, a la frecuencia de muestreo de 48k Hz se vio una menor distorsión armónica en el rango de frecuencias de 100 Hz a 1k Hz en $|Y(f)|$ y $|H(f)|$ del Layer III en comparación con estos mismos parámetros del Layer II, esto debido a los cambios de ventana a tipo Start-Short en los tiempos respectivos para las frecuencias mencionadas, se permitió una mejor reconstrucción en estas secciones del espectro en el Layer III, esto es igualmente visualizable en 44.1k Hz pero dándose menores magnitudes de distorsión armónica en el espectro en ambos Layer.

En las figuras 7.1 y 7.2 del Anexo 9, la magnitud del espectro se ve plano en todos los resultados, con variaciones promedio de ± 6.2 dBFS (Figura 5-4) alrededor de 0 dBFS, estos valores están dentro del límite de distorsión por recorte (amplitudes en el dominio del tiempo que exceden el rango de -1 y 1), lo cual permite a ambos sistemas generar representaciones aceptables de las señales procesadas.

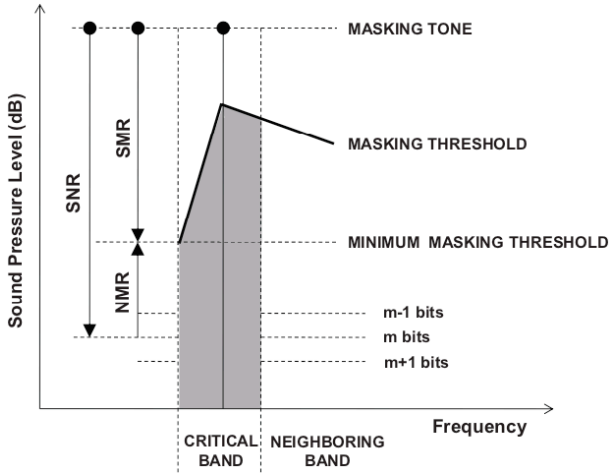
Otro hecho relevante es la baja distorsión y poca diferencia espectral entre Layers que se presenta en alta frecuencia (desde 1000 Hz hasta la frecuencia de Nyquist), esto sucede porque al aumentar la frecuencia de las señales entrantes, la representación de las componentes senoidales son menos precisas, es decir, el grado de fidelidad de la representación de las señales depende de la frecuencia de muestreo, y las señales cuyas componentes están próximas a la frecuencia de Nyquist, que poseen una representación vaga frente a su forma análoga y continua en el dominio del tiempo.

6.1.2 Conclusiones acerca de los THD y SNR de las bandas del PQMF

Para todos los resultados en general de THD, los porcentajes de distorsión armónica total resultan ser menores al 1% para todas las bandas inferiores a la 20 (15k Hz a $f_s = 48$k Hz y 13.7kHz a $f_s = 44.1$k Hz) lo cual indica de nuevo que el sistema es apropiado para la reconstrucción de señales, los máximos porcentajes menores al 16% se presentan en la banda 26 (19.5k Hz a $f_s = 48$k Hz y 17.7k Hz a $f_s = 44.1$k Hz), lo que contempla un rango de frecuencias al que el ser humano es poco sensible.

Tomando de nuevo la idea del último párrafo de conclusiones en la sección 6.1.1, el sistema tiende a generar mayores porcentajes de THD a medida que la frecuencia aumenta, esto se da por la representación de los tonos puros que se ingresaron a ambos sistemas como también a las representaciones adquiridas de las reconstrucciones por el modelo hibrido IPQMF-IMDCT en el Layer III e IPQMF en el Layer II, es decir, la representación senoidal completa dependiente del número de muestras por unidad de tiempo de las señales a alta frecuencias, produce formas difusas de las señales pretendidas, muchas veces tendiendo a discontinuidades en sus formas de onda, y generando armónicos extra en el dominio de la frecuencia, tal como lo postula el fenómeno de Gibbs como se explica en (Kuo et al., 2006, p. 198).

Figura 6–1: Relación Ruido a Enmascarador del Mod. Psicoacústico 2.



Fuente: Imagen tomada de Thiagarajan & Spanias (2011, p. 58).

Los datos obtenidos y analizados de SNR apoyan de nuevo la idea de mayores cantidades de distorsión armónica (o de presencia de ruido en este caso) en los rangos espectrales de alta frecuencia, en particular después de la banda 16 del PQMF o de $f_s/4 = f_N/2$. El Modelo Psicoacústico 2 se halla involucrado aquí gracias a que el indicador SMR (Relación Señal a Enmascarador) en la expresión de $xmin$ (Ecuación (4.34): distorsión permitida debajo del umbral de enmascaramiento) junto al indicador $xfsf$ (Ecuación (4.36): ruido espectral inherente a la señal debido a la cuantización) verifican a través de su comparación que el ruido generado al cuantizar coeficientes procesados de la MDCT y de los del PQMF en el caso del Layer II, estén por debajo de los umbrales de enmascaramiento. La Figura 6–1, permite visualizar cómo el ruido inherente (conteniendo también al ruido $xfsf$) a las señales, está supeditado a estar debajo de los respectivos umbrales de enmascaramiento que

indica el modelo Psicoacústico, así, la Relación Ruido a Enmascarador NMR está en términos de la diferencia: NMR = SNR − SMR, donde SNR = $xfsf + R$, y donde R es el ruido propio de la señal de audio. Ahora bien, debido a que los resultados medidos de SNR son los resultantes de la salida de ambos Layer II y III, estos SNR son los permitidos por el modelo psicoacústico 2.

6.1.3 Conclusiones acerca de los Parámetros Estadísticos.

Gracias a los resultados de los parámetros estadísticos, con los que se pretendió observar si al ingresar una señal aleatoria estocástica (WGN Gaussiano) a ambos sistemas (Layer II y III), variaban las propiedades de la función de Distribución Normal $f_x(x)$ (parámetro del cual depende la Densidad de Potencia Espectral PSD y función de Autocorrelación $R_{xx}(\tau)$) de las señales WGN originales, permitiendo ver si ambos sistemas no se comportaron de tal forma que la distribución normal fuese alterada (quizá de forma no lineal).

Al observar que la función de distribución normal de ambos Layer, a las frecuencias de muestreo de 44.1k y 48k Hz, no varió en esencia, manteniendo de forma semejante las características originales de Normalidad Gaussiana de la WGN inicial, a través de una varianza de salida σ^2 en ambos casos muy próximas a la original de 0.04, y una media aritmética μ_x muy cercana a cero, indica que la PSD de salida de ambos sistemas no varió en su forma funcional y que esta es plana alrededor de sus respectivas varianzas de salida σ^2 (como se observó en la Figura 5–9), dando lugar a que la Autocorrelación mostrase la no-relación estadística entre las muestras de la WGN procesada (La correlación entre los datos de WGN es 0 a excepción de la condición temporal mencionada en la sección 5.1.3 que resulta en $\sigma^2\delta(\tau)$).

Gracias a lo anterior, se concluye que ambos esquemas de compresión MPEG-1 Layer II y III no alteran las características de distribución estadística de las señales procesadas, debido a sus procesos lineales, PQMF, MDCT y modelos de Cuantización, dando lugar a reconstrucciones fieles de las señales de audio.

6.2 Conclusiones de las pruebas perceptuales

Considerando que el análisis hecho a partir de los resultados de las pruebas subjetivas, fue contemplando una muestra poblacional, haciendo uso de una hipótesis de prueba, se buscó que los resultados dados por los evaluadores fuesen significativos, en este sentido, se destaca una alta fiabilidad en la mayoría de los datos obtenidos al decir que los evaluadores seleccionados distinguieron de manera acertada los audios en los que se presentaba un audio procesado por el sistema de compresión implementado.

Al comparar el sistema implementado del esquema Layer II contra el Layer III, se encontró que a partir de las encuestas realizadas, los procesos de compresión de audio pueden incurrir en percepciones de degradación de calidad respecto a un audio original sin pérdidas, como fue el caso de la comparación de la capa Layer II, que a su vez (de forma contraria) como lo ocurrido en la evaluación del esquema Layer III, el audio comprimido puede percibirse de

forma agradable al comparase con el audio original. Las diferencias percibidas por los sujetos de prueba en los dos esquemas de compresión, durante la realización de pruebas subjetivas, es leve al observar que la apreciación global de estos dos Layers de acuerdo con sus promedios y distribución F es similar.

6.3 Conclusiones de la Implementación programada

Respecto a la programación e implementación de código en el marco de aplicación JUCE, a la hora de tratar con espacios de memoria variable (matrices de *n* dimensiones de cualquier tipo de datos) para alterar sus dimensiones o acceder a su información y no generar errores por intento duplicación de espacios de información, es sumamente recomendable trabajar con las herramientas que el entorno mismo marco de aplicación concede (en este caso JUCE). Para esto fue necesario utilizar objetos de la clase *AudioBuffer<type>* tipo flotante para albergar como arreglos auxiliares las muestras de audio salientes, los cuales fueron "recons_signal" y "auxArray" vistos al inicio de la sección 4.2.1, siendo "recons_signal" de longitud contante 576 y "auxArray" siendo un arreglo unidimensional de longitud variable, que toma un tamaño igual al número de muestras de entrada por bloque cada vez que el método prepareToPlay() es llamado. En general, es importante que el algoritmo sea capaz de destruir (si es necesario) e inicializar de forma adecuada los espacios en memoria de tamaños variables, para esto se recomienda usar arreglos dinámicos de la forma: *type *array = new <type> [size]*, o utilizar las distintas clases suministradas por las librerías estándar de C++ para generar arreglos como lo es *std::array* o las clases que suministra JUCE.

El funcionamiento pretendido en Tiempo Real para los algoritmos de procesamiento de análisis y síntesis con los bancos de filtros PQMF-IPQMF (Layer II) y el modelo Híbrido PQMF-MDCT-IMDCT-IPQMF (Layer III) fue logrado a través del diseño e implementación de la estructura lógica y aritmética explicada en la sección 4.2 (particularmente en 4.2.1), la cual obedece al flujo aritmético que se muestra en las tablas del Anexo 5 para los siguientes tamaños del buffer entrada de datos $2^N = \{32,64, \dots ,512\}$, $N = 5,6 \dots ,9$.

6.4 Recomendaciones

Ya que en el presente trabajo se excluyó la codificación de datos, omitiendo la manipulación de bits y, por tanto, ignorando el cambio de las tasas de pérdida de información por unidad de tiempo, se recomienda implementar la codificación completa de los esquemas de compresión Layer II y III descritos en las secciones C.1.5.2.7 ("Bit Allocation") para el Layer II y C.1.5.4.4.3 a C.1.5.4.4.8 para el Layer III (que hace uso de la codificación entrópica de Huffman), de la normativa ISO-IEC 11172-3. Por otro lado, también se recomienda extender la posibilidad del funcionamiento del sistema para bloques de tamaños mayores a 512 muestras, lo cual se piensa implicaría repetir n veces el procesamiento visto en la Tabla 4-5 para 512 muestras. También se recomienda optimizar la programación para que el sistema procese en paralelo por medio de *threads* ("hilos") u otros métodos de programación, n canales de entrada (cuestión mencionada al final de la sección 4.1) y no solo 1 canal.

Referencias

Bosi, M., & Goldberg, R. (2004). Introduction to Digital Audio Coding and Standards. *Journal of Electronic Imaging*. https://doi.org/10.1117/1.1695413

Brown, R. G., & Hwang, P. Y. C. (1997). Introduction to Random Signals and Applied Kalman Filtering. En *International Journal of Group Psychotherapy*. https://doi.org/10.1521/ijgp.2010.60.4.455

BS.1116-1, I.-R. (1997). 1116-1: Methods for the subjective assessment of small impairments in audio systems including multichannel sound systems. *International Telecommunication Union, Geneva*.

Gans, D. (2015). Digital vs Analog Audio: An Overview. Recuperado de Digital vs Analog Audio: An Overview website: https://www.klipsch.com/blog/digital-vs-analog-audio

Herrera, M. (2016). *Digital Radio, Part 1: Perceptual Audio Compression*. Bogotá: Editorial Bonaventuriana.

ISO/IEC. 11172-3, Smith, J. O., & Abel, J. S. (1993). ISO\IEC 11172-3: Information Technology - Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5 Mbit/s - Part 3: Audio. *ISE\IEC JTC 1\SC 29\ WG 11*.

ITU-R BS.1534-1. (2015). Method for the subjective assessment of intermediate quality level of audio systems. *International Telecommunication Union*.

Kuo, S. M., Lee, B. H., & Tian, W. (2006). Real-Time Digital Signal Processing: Implementations and Applications: Second Edition. En *Real-Time Digital Signal Processing: Implementations and Applications: Second Edition*. https://doi.org/10.1002/0470035528

L. Grossman, S. (2008). *Álgebra Lineal* (6a ed.; Pablo E. Roig Vázquez & Carlos Zúñiga Gutiérrez, Eds.). México, D.F.: McGRAW-HILL.

M.R. Spiegel, J. Schiller, R. A. S. (2007). *Probabilidad y Estadística, Schaum's Outline of Theory and Problems of Probability and Statistics* (2da Edició). México, D.F.

Nussbaumer, H. J. (1981). Pseudo-QMF Filter Bank. *IBM Tech.*, *24*, 3081–3087.

Oppenheim, A. V, & Schafer, R. W. (1998). Discrete Time Signal Processing 2nd Edition. En *Book*.

Orús, A. (2020). Número de suscriptores de servicios de música en streaming a nivel mundial durante el primer semestre de 2019, por plataforma. Recuperado de statista.com website: https://es.statista.com/estadisticas/942349/principales-plataformas-de-musica-en-streaming-del-mundo-segun-suscriptores/

Princen, J. P., Johnson, A. W., & Bradley, A. B. (1987). SUBBAND/TRANSFORM CODING USING FILTER BANK DESIGNS BASED ON TIME DOMAIN ALIASING CANCELLATION. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*.

Rao, K. R., & Yip, P. (1990). Discrete Cosine Transform. Algorithms, Advantages,

Applications. En *Discrete Cosine Transform. Algorithms, Advantages, Applications*. https://doi.org/10.1016/b978-0-08-092534-9.50007-2

Rothweiler, J. H. (1983). POLYPHASE QUADRATURE FILTERS - A NEW SUBBAND CODING TECHNIQUE. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*.

Thiagarajan, J. J., & Spanias, A. (2011). Analysis of the MPEG-1 Layer III (MP3) Algorithm Using MATLAB. *Synthesis Lectures on Algorithms and Software in Engineering*. https://doi.org/10.2200/s00382ed1v01y201110ase009

Zölzer, U. (2008). Digital Audio Signal Processing. En *Digital Audio Signal Processing*. https://doi.org/10.1002/9780470680018

Zölzer, U., & Smith III, J. O. (2003). DAFX—Digital Audio Effects. *The Journal of the Acoustical Society of America*. https://doi.org/10.1121/1.1616923

7. Anexos

Anexo 1: Transformaciones Proyecciones, Kernel y Reglas de Transformación

En algebra lineal, una *transformación* o *mapeo*, ocurre entre espacios vectoriales. La Transformación Lineal de un espacio vectorial V a un espacio vectorial U es un mapeo $T: V \to U$, para todo vector $\boldsymbol{u}$ y $\boldsymbol{v}$ en V y cualquier constante c. Un ejemplo simple de espacios vectoriales V y U son $\mathbb{R}^2$ y $\mathbb{R}^3$ (donde $\mathbb{R}^2$ es el conjunto de todos los vectores existentes compuestos de números reales con solo 2 componentes). Así T es una regla que asigna a cada vector $\boldsymbol{u}$ en $\mathbb{R}^n$ un único vector $T(\boldsymbol{u})$ en $\mathbb{R}^m$.

Por ejemplo, las propiedades vistas en cualquier transformada de Fourier se dan y se pueden escribir con esta notación de transformaciones vectoriales: $T(c\boldsymbol{u} + c\boldsymbol{v}) = c\big(T(\boldsymbol{u}) + T(\boldsymbol{v})\big)$.

En términos de la DFT, la *regla de transformación* T se entiende como el Factor de Giro, la matriz $\boldsymbol{W}_N^{nk} = e^{-j\frac{2\pi}{N}nk}$, y la señal/sucesión $x[n]$ que será *transformada* o *mapeada* en el dominio de $\boldsymbol{W}_N^{nk}$ es el vector $\boldsymbol{u}$. Finalmente, la sucesión resultante $X[k]$ se entiende como la *imagen* de $x[n]$ bajo la acción de T o de $\boldsymbol{W}_N^{nk}$.

La DFT se puede escribir como:

$$X[k] = T(x[n]) = \boldsymbol{W}_N^{nk}(x[n]) = \langle x[n], \boldsymbol{W}_N^{nk} \rangle = \sum_{n=0}^{N-1} x[n] \cdot \boldsymbol{W}_N^{nk} \tag{7.1}$$

Donde $\langle x[n], \boldsymbol{W}_N^{nk} \rangle$ es el producto interno o producto punto entre $x[n]$ y los vectores fila de $\boldsymbol{W}_N^{nk}$. Igualmente como indican ISO/IEC. 11172- (Smith & Abel, 1993) se puede escribir como el producto matricial entre un vector $\boldsymbol{x} = x[n]$ y $\boldsymbol{W}_N^{nk}$.

$$X[k] = \boldsymbol{X} = \boldsymbol{W}_N^{nk}\boldsymbol{x} \tag{7.2}$$

Kernel: El Kernel de T, escrito como $\ker(T)$, es el conjunto de todos los vectores existentes en V que se mapean mediante T a $\mathbf{0}$ (vector de ceros) en otro espacio vectorial U. Por ejemplo, un vector $\boldsymbol{v}$ contenido en el espacio V al aplicarse una transformación dada por T tal que se genere un vector $\mathbf{0}$. Es decir: $\ker(T) = \{\boldsymbol{v} \text{ en } V : T(\boldsymbol{v}) = \mathbf{0}\}$. Lo cual, si $T = \boldsymbol{W}_N^{nk}$ entonces el $\ker(T) = \ker(\boldsymbol{W}_N^{nk})$ es el espacio nulo de la matriz $\boldsymbol{W}_N^{nk}$: $\ker(\boldsymbol{W}_N^{nk}) = \text{nulo}(\boldsymbol{W}_N^{nk})$.

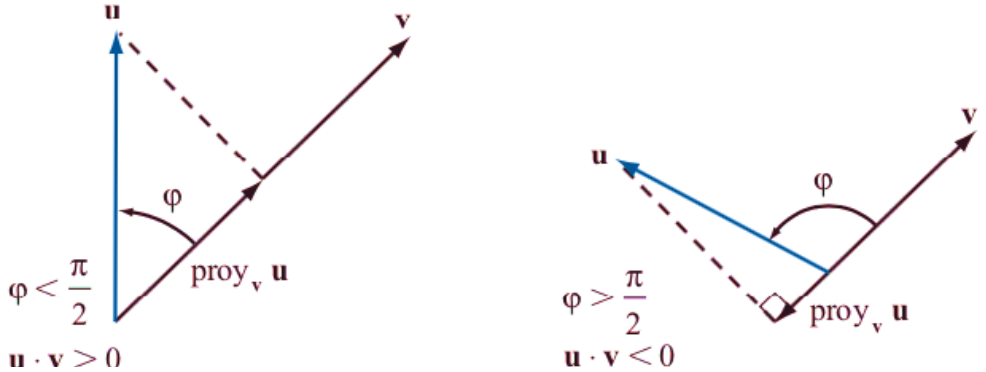
En palabras, el Kernel de una transformación matricial es el espacio nulo de dicha Matriz ($\boldsymbol{W}_N^{nk}$ o $\boldsymbol{C}_N$), y esta es una matriz que permite lo anterior.

Por ejemplo, si una matriz $\boldsymbol{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 1 \end{bmatrix}$, el Kernel de $\boldsymbol{A}$ es la matriz columna $\ker(\boldsymbol{A}) = \begin{bmatrix} 0 \\ -a \\ 2a \end{bmatrix}$ donde a es cualquier escalar. Al hacer el producto matricial entre estas dos:

$$\boldsymbol{A} \cdot \ker(\boldsymbol{A}) = \begin{bmatrix} 1 \cdot 0 - 0 \cdot a + 0 \cdot 2a \\ 0 \cdot 0 - 2 \cdot a + 2 \cdot 2a \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \mathbf{0}$$

Se genera un vector de ceros, el cual representa un espacio nulo de $\boldsymbol{A}$ a partir de su Kernel.

Figura 7–1: Representación gráfica en $\mathbb{R}^2$de proyección entre dos vectores.



Fuente: Imagen tomada de Grossman (2008).

Proyecciones: "Sean $\boldsymbol{u}$ y $\boldsymbol{v}$ dos vectores diferentes de cero, la proyección de $\boldsymbol{u}$ sobre $\boldsymbol{v}$ es un vector denotado por $proy_{\boldsymbol{v}}(\boldsymbol{u})$, que se define por la Ecuación (7.3", es la definición matemática dada por Grossman (2008, p. 238).

$$proy_{\boldsymbol{v}}(\boldsymbol{u}) = \frac{\langle \boldsymbol{u}, \boldsymbol{v} \rangle}{\langle \boldsymbol{v}, \boldsymbol{v} \rangle} \cdot \boldsymbol{v} \qquad (7.3)$$

El producto interno $\langle \boldsymbol{u}, \boldsymbol{v} \rangle$, es la operación que ocurre en las transformadas DFT y MDCT, que es mejor escribir con $\boldsymbol{u} = x[n]$ y $\boldsymbol{v}_k = \boldsymbol{W}_N^{nk}$, donde $\boldsymbol{v}_k$ es una matriz y el producto interno $\langle \boldsymbol{u}, \boldsymbol{v_k} \rangle$, ocurre entre la señal $\boldsymbol{u} = x[n]$ para cada una de las filas de $\boldsymbol{v}_k = \boldsymbol{W}_N^{kn}$ (las sucesiones sobre n de $e^{-j\frac{2\pi}{N}kn}$ para cada fila k).

Es útil entender la proyección ya que ayuda a conceptualizar mejor a las transformaciones ortogonales (DFT, MDCT, etc) que son productos internos observando la diferencia entre la proyección $proy_{\boldsymbol{W}_N^{nk}}(x[n])$ y el producto interno $\langle x[n], \boldsymbol{W}_N^{nk} \rangle$.

Se mostrará en la proyección $proy_{\boldsymbol{W}_N^{nk}}(x[n])$ en qué valores o componentes de tiempo n hay un valor o valores de $x[n]$ que existan en $\boldsymbol{W}_N^{nk}$, y esto se hace por cada vector (fila) $\boldsymbol{v}_k$ que se genera al variar la frecuencia k.

Otra interpretación de la proyección es la comparación de los componentes espectrales de una señal $x[n]$ sobre los componentes espectrales de un set finito de sinusoides $\boldsymbol{W}_N^{nk}$, así se busca y se mide "qué tanto de las componentes de $x[n]$ hay en todo un espectro $\boldsymbol{W}_N^{nk}$".

Un ejemplo de lo anterior, para $N = 8$:

La señal de entrada o vector: $x[n] = \{0, -1, 0, 1, 0, -1, 0, 1\}$ el cual será proyectado sobre los vectores fila de $\boldsymbol{W}_N^{nk}$:

$$\boldsymbol{W}_N^{nk} = e^{-j\frac{2\pi}{N}kn}$$

$$= \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \frac{1}{\sqrt{2}}(1-j) & -j & \frac{-1}{\sqrt{2}}(1+j) & -1 & \frac{-1}{\sqrt{2}}(1-j) & j & \frac{1}{\sqrt{2}}(1+j) \\ 1 & -j & -1 & j & 1 & -j & -1 & j \\ 1 & \frac{-1}{\sqrt{2}}(1+j) & j & \frac{1}{\sqrt{2}}(1-j) & -1 & \frac{1}{\sqrt{2}}(1+j) & -j & \frac{-1}{\sqrt{2}}(1-j) \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & \frac{-1}{\sqrt{2}}(1-j) & -j & \frac{1}{\sqrt{2}}(1+j) & -1 & \frac{1}{\sqrt{2}}(1-j) & j & \frac{-1}{\sqrt{2}}(1+j) \\ 1 & j & -1 & -j & 1 & j & -1 & -j \\ 1 & \frac{1}{\sqrt{2}}(1+j) & j & \frac{-1}{\sqrt{2}}(1-j) & -1 & \frac{-1}{\sqrt{2}}(1+j) & -j & \frac{1}{\sqrt{2}}(1-j) \end{bmatrix}$$

La transformación $X[k] = \langle x[n], \boldsymbol{W}_N^{nk} \rangle$ genera seis espacios nulos y dos componentes de frecuencia $k = 2$ y $k = 6$.

$$X[k] = \langle x[n], \boldsymbol{W}_N^{nk} \rangle = \begin{bmatrix} 0 \\ 0 \\ -4j \\ 0 \\ 0 \\ 0 \\ 4j \\ 0 \end{bmatrix}, \langle \boldsymbol{W}_N^{nk}, \boldsymbol{W}_N^{nk} \rangle^2 = \begin{bmatrix} 8 \\ 8 \\ 8 \\ 8 \\ 8 \\ 8 \\ 8 \\ 8 \end{bmatrix}$$

Finalmente, la proyección del vector $x[n]$ sobre cada uno de las sinusoides $\boldsymbol{W}_N^{nk}$, genera una extensión sobre la cantidad de componentes en tiempo n (filas), de los resultados del

producto interno $\langle x[n], \boldsymbol{W}_N^{nk} \rangle$ para cada frecuencia k, lo cual indica la existencia o la no existencia de las componentes periódicas de $x[n]$ sobre $(N-1)$, k frecuencias de $\boldsymbol{W}_N^{nk}$ para toda la duración de $x[n]$ (la cual debe ser la misma en tiempo -columnas- de $\boldsymbol{W}_N^{nk}$).

$$proy_{\boldsymbol{W}_N^{nk}}(x[n]) = \frac{\langle x[n], \boldsymbol{W}_N^{nk} \rangle}{\langle \boldsymbol{W}_N^{nk}, \boldsymbol{W}_N^{nk} \rangle^2} \cdot \boldsymbol{W}_N^{nk}$$

$$= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{1}{2}j & -\frac{1}{2} & \frac{1}{2}j & \frac{1}{2} & -\frac{1}{2}j & -\frac{1}{2} & \frac{1}{2}j & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2}j & -\frac{1}{2} & -\frac{1}{2}j & \frac{1}{2} & \frac{1}{2}j & -\frac{1}{2} & -\frac{1}{2}j & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Por lo anterior, hablar de una transformación de un dominio temporal a uno espectral usando cálculos como lo de la DFT, es hablar también de *proyecciones* entre vectores.

Anexo 2: Algunas propiedades y demostraciones de la DFT

La función "Delta de Kronecker" descrita por (L. Grossman, 2008) se da como en la Ecuación (7.4), pero también es describible a través de la Ecuación (7.5).

$$\delta[k] = \begin{cases} 1, k = 0 \\ 0, k \neq 0 \end{cases} \therefore \delta[k - k_0] = \begin{cases} 1, k = k_0 \\ 0, k \neq k_0 \end{cases} \tag{7.4}$$

$$\delta[k - k_0] = \frac{1}{N}\sum_{n=0}^{N-1} e^{\pm j\frac{2\pi}{N}(k-k_0)n} = \frac{1 - e^{\pm j2\pi(k-k_0)}}{1 - e^{\pm j\frac{2\pi}{N}(k-k_0)}} : 0 \leq k_0 \leq N - 1 \tag{7.5}$$

La Ecuación (7.5) en forma de serie geométrica de potencias es posible dadas las condiciones de la Ecuación (7.6):

$$\delta[k - k_0] = \begin{cases} \dfrac{1}{N}\displaystyle\sum_{n=0}^{N-1} 1 = \frac{N}{N} = 1 & k = k_0 \\ \dfrac{1 - e^{\pm j2\pi(k-k_0)}}{1 - e^{\pm j\frac{2\pi}{N}(k-k_0)}} = 0 & k \neq k_0 \; \forall \; (k - k_0) \in \mathbb{Z} \end{cases} \tag{7.6}$$

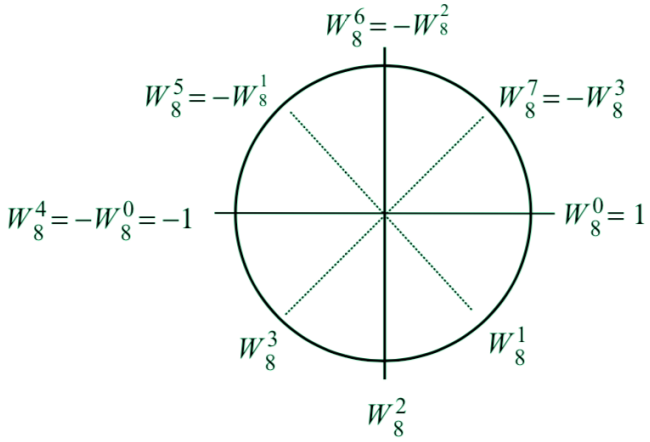
Estas definiciones resultan muy útiles a la hora de observar el comportamiento de la DFT y su transformada inversa.

Si $x[n] = A_0 \cos\left(\frac{2\pi}{N} k_0 n\right), n = 0,1, \dots, N - 1$. su DFT es:

$$\tilde{X}[k] = A_0 \sum_{n=0}^{N-1} \cos\left(\frac{2\pi}{N} k_0 n\right) e^{-j\frac{2\pi}{N}kn} = \frac{A_0}{2}\left[\sum_{n=0}^{N-1} e^{-j\frac{2\pi}{N}(k-k_0)n} + \sum_{n=0}^{N-1} e^{-j\frac{2\pi}{N}(k+k_0)n}\right]$$

Y teniendo en cuenta que en el plano complejo $W_N^k = e^{-j\frac{2\pi}{N}k}$ (su rango) es simétrico, como en el ejemplo de la Figura 7–2 para $N = 8$. Con este círculo unitario (Kuo et al., 2006) sugieren que el dominio k de W_N^k y de $\tilde{X}[k]$ pase a tener una parte negativa, la cual resguarda la misma información que la otra positiva. Así todas las componentes de $\tilde{X}[k]$ son simétricas respecto a la mitad del arreglo, y de forma conceptual se puede reorganizar k como $-N/2 \leq k \leq N/2 - 1$.

Figura 7–2: Visualización del dominio simétrico de los Factores de Giro de la DFT con $N = 8$.



Fuente: Imagen tomada de Zölzer & Smith III (2003)

Y el par de series de $\tilde{X}[k]$ contienen cuatro condiciones, donde aparece el delta de Kronecker $\delta[k \pm k_0]$:

$$\sum_{n=0}^{N-1} e^{-j\frac{2\pi}{N}(k-k_0)n} = \frac{1 - e^{-j2\pi(k-k_0)}}{1 - e^{-j\frac{2\pi}{N}(k-k_0)}} = \begin{cases} N, k = k_0 \\ 0, k \neq k_0 \end{cases} = N\delta[k - k_0]$$

$$\sum_{n=0}^{N-1} e^{-j\frac{2\pi}{N}(k+k_0)n} = \frac{1 - e^{-j2\pi(k+k_0)}}{1 - e^{-j\frac{2\pi}{N}(k+k_0)}} = \begin{cases} N, k = -k_0 \\ 0, k \neq -k_0 \end{cases} = N\delta[k + k_0]$$

Y la solución de esta DFT resulta en:

$$\tilde{X}[k] = N\frac{A_0}{2}(\delta[k - k_0] + \delta[k + k_0]), \qquad k = -\frac{N}{2}, \dots, -1,0,1, \dots, \frac{N}{2} - 1 \qquad (7.7)$$

Estas dos componentes $\frac{A_0}{2}\delta[k \pm k_0]$ muestran la simetría mencionada anteriormente y sugieren la observación del Teorema de Parseval, la cual según Kuo et al. (2006 p. 308) establece que la distribución energética de una señal debe mantenerse igual en el dominio del tiempo y la frecuencia de la forma:

$$E = \sum_{n=0}^{N-1} |x[n]|^2 = \frac{1}{N}\sum_{k=0}^{N-1} |X[k]|^2 \qquad (7.8)$$

El resultado de la Ecuación (7.7, muestra que la energía de una señal se divide de forma igual sobre la parte negativa y positiva de su espectro. En este caso, la mitad de la energía se encuentra en un impulso en la parte negativa de $\tilde{X}[k] \rightarrow \tilde{X}[-k_0]$, y la otra mitad en la posición simétrica del lado positivo de $\tilde{X}[k] \rightarrow \tilde{X}[k_0]$.

La IDFT de la Ecuación (7.7) es:

$$x[n] = \frac{1}{N}\sum_{k=-N/2}^{\frac{N}{2}-1} \tilde{X}[k]\cdot e^{j\frac{2\pi}{N}kn}$$

$$= \frac{A_0}{2}\sum_{k=-N/2}^{\frac{N}{2}-1} \delta[k-k_0]e^{j\frac{2\pi}{N}kn} + \frac{A_0}{2}\sum_{k=-N/2}^{\frac{N}{2}-1} \delta[k+k_0]e^{j\frac{2\pi}{N}kn}$$

$$\frac{A_0}{2}\sum_{k=-N/2}^{\frac{N}{2}-1} \delta[k-k_0]e^{j\frac{2\pi}{N}kn} = \frac{A_0}{2}\left(0+\cdots+\delta[k_0-k_0]e^{j\frac{2\pi}{N}k_0n}+\cdots+0\right) = \frac{A_0}{2}e^{j\frac{2\pi}{N}k_0n}$$

$$\frac{A_0}{2}\sum_{k=-N/2}^{\frac{N}{2}-1} \delta[k+k_0]e^{j\frac{2\pi}{N}kn} = \frac{A_0}{2}\left(0+\cdots+\delta[-k_0+k_0]e^{-j\frac{2\pi}{N}k_0n}+\cdots+0\right)$$

$$= \frac{A_0}{2}e^{-j\frac{2\pi}{N}k_0n}$$

Al sumar las dos series, se obtiene la reconstrucción exacta de $x[n]$:

$$x[n] = \frac{A_0}{2}\left(e^{j\frac{2\pi}{N}k_0n} + e^{-j\frac{2\pi}{N}k_0n}\right) = A_0\cos\left(\frac{2\pi}{N}k_0n\right)$$

Anexo 3: Notas y técnicas de programación aplicadas (PQMF-IPQMF).

En informática se le conoce al Buffer Circular como una estructura de datos, clasificada en el modelo First In First Out –FIFO-. De manera conceptual una estructura FIFO es un método y elemento de evaluación de listas o *stack* (también es adaptable al uso de punteros y asignación dinámica de memoria) en el que el primer elemento en entrar es el primero en salir, es decir, el primer elemento apunta al frente de la lista de todos los elementos y el segundo elemento apunta al último elemento de la lista, este modelo se diferencia de las estructuras Last In First Out (LIFO).

La utilidad de estos modelos radica en el uso de tipos de datos abstractos –aquellos que poseen operaciones específicas de acuerdo con una serie de datos presentes en el modelo que se va a implementar-.

La abstracción de datos es un concepto que proviene de la programación orientada a objetos en el cual la caracterización de un determinado objeto engloba sus principales características y funciones (atributos y métodos), esto permite utilizar la abstracción del objeto en situaciones determinadas, permitiendo la optimización de procesos. Así mismo, los tipos de dato abstracto engloban métodos específicos dirigidos a determinados tipos de dato, siendo la del circular Buffer (Circular Queue, Ring Buffer) un claro ejemplo de los que es un tipo de dato abstracto.
Datos:

- Head: Índice que apunta al primer elemento con valor del Buffer.
- Tail: Índice que apunta al último elemento con valor del Buffer.
- Buffer: Arreglo o lista (asignación dinámica), el cual contiene los datos en Estructura tipo Circular Buffer.

Posición i	0	1	2	3	4	5	6
Dato							

Imagen. Representación de Buffer Circular a través de arreglo lineal.

Proceso de indexación de un buffer circular. [3]

Estado inicial de la lista

	Head: apunta al primer elemento de la lista
	Tail: apunta a último elemento del arreglo

Posición i	0	1	2	3	4	5	6
Dato	1	2	3	4	5		

Añadir elemento utilizando el método addSample(n), insertando el elemento n.
Estado inicial de la lista

Posición i	0	1	2	3	4	5	6
Dato	1	2	3	4	5	6 = n	

Una vez insertado un elemento se debe eliminar, reemplazar o sacar el dato que apunta a la primera posición.

Posición i	0	1	2	3	4	5	6
Dato		2	3	4	5	6 = n	

El proceso de add(n) y delete se repite, en este caso el arreglo lineal se encuentra lleno, sin embargo, es una implementación en anillo, es decir los índices i = 0, 1 son utilizados

Posición i	0	1	2	3	4	5	6
Dato			3	4	5	6	7 = n

Posición i	0	1	2	3	4	5	6
Dato	1	2	3	4	5	6	7

Análisis de los métodos e implementación.

Operación Modulo.

- addSample(n)

En primer lugar, es evaluado el estado del Buffer – si este se encuentra lleno o vacío -, al estar lleno no se podrán añadir mas elementos, de lo contrario se aumenta el índice de la ultima posición (Tail) en una unidad, para así añadir el elemento n en el nuevo índice Tail de el Buffer.[2]

Operación Modulo.
El modulo es una operación utilizada para calcular el resto de la operación de división.
Sea tamaño = 6
Tail = 3; (3 + 1) % 6 -> 4 % 6 = 4
Tail = 4; (4 + 1) % 6 -> 5 % 6 = 5
Tail = 5; (5 + 1) % 6 -> 6 % 6 = 0
Tail = 6; (6 + 1) % 6 -> 7 % 6 = 1

Una optimización realizada en el Buffer circular, teniendo en cuenta que está enfocada la implementación del Buffer al procesamiento convolutivo de audio, implica una indexación de datos de forma contraria.

Proceso de Indexación de la optimización- aplicación en audio, con propósitos para desarrollo de PQMF.

En un primer momento se debe determinar el tamaño del buffer que va a ser utilizado – en este caso un tamaño de 512 muestras-, se determina una variable – entera y privada- llamada bufferSize, la cual va a representar el tamaño del arreglo, a su vez se crea un índice que va a recorrer de derecha a izquierda el arreglo – reversar el arreglo-. El arreglo se representa en la variable buffer – representado por medio de un operador de indirección *, para acceder a los valores de dicho arreglo-, a su vez se hace una asignación dinámica de memoria en el arreglo con la palabra reservada NEW.

Índice = i	8	7	6	5	4	3	2	1	0
Dato									

Proceso del método addSample(), en la versión optimizada.
En las siguientes dos tablas se representa el proceso de añadir muestras, utilizando la posición del índice, en este caso puntual la optimización cambia al insertar los valores en reversa, es decir el primer valor del arreglo de entrada se asigna a el ultimo valor del buffer, indicado en la posición del índice 0

Índice = i	8	7	6	5	4	3	2	1	0
Dato									1

Índice = i	8	7	6	5	4	3	2	1	0
Dato								2	1

Para la implementación del filtro de análisis PQMF, en vez de ingresar bloques de a 32 muestras, de la primera a la última, y así sucesivamente hasta completar 16 iteraciones como lo indica la normativa (ISO_IEC_11172-3_1993) y como lo muestra la Figura 7–3; se optó por ingresar las muestras de la última a la primera desde el último hasta el primer espacio de bloques de a 32 muestras como lo indica la Figura 7–4, es decir, en forma de un buffer circular inverso.

Figura 7–3: Muestras de ingreso según la recomendación de la Normativa ISO-IEC-11172-3, ingreso de la primera a la última desde el primer hasta el último espacio en el arreglo de a bloques de 32 muestras.

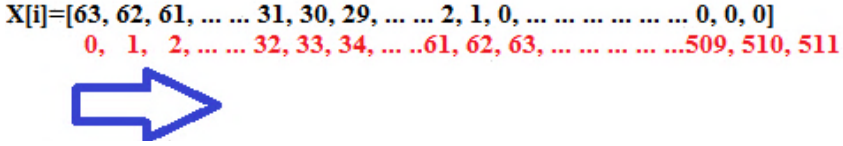
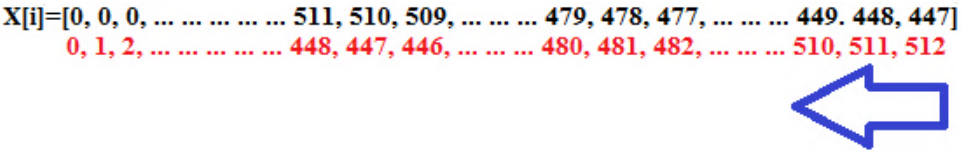
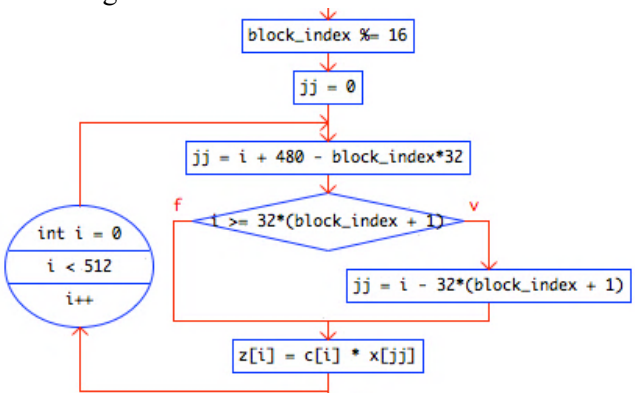


Figura 7–4: Forma de ingreso implementado de las muestras de entrada al buffer circular, estas ingresan directamente de la última muestra al último hasta el primer espacio.



Esto mediante el siguiente algoritmo de la Figura 7–5:

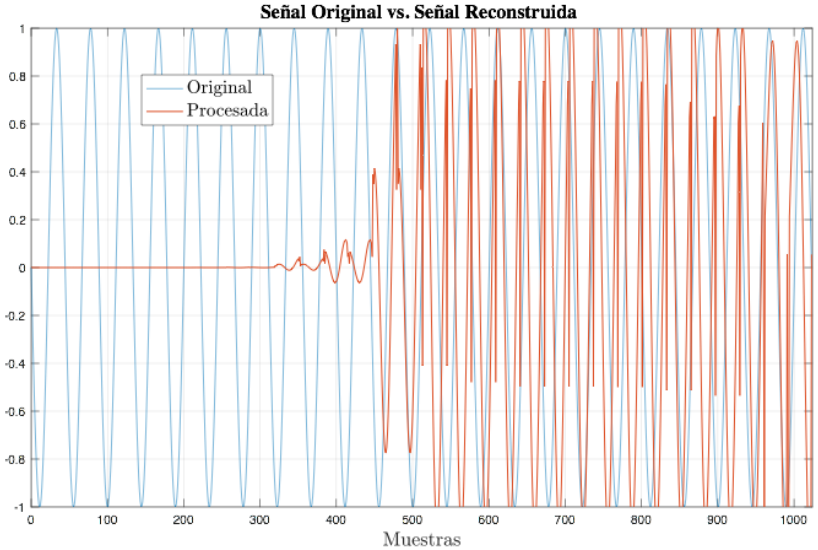
Figura 7–5: Algoritmo de ingreso de muestras al buffer circular hacia el PQMF.




```
float *PQMF_Analysis::PQMF_Filtering(float x[]){

    block_index %= 16; //siempre x[] debe tener un tamaño de 512
    jj = 0;
    for(int i = 0; i < 512; i++) {
        jj = i + 480 - block_index*32;
        if (i >= 32*(block_index + 1)){
            jj = i - 32*(block_index + 1);
        }
        z[i] = c[i] * x[jj];
    }

    block_index++;
...
```

La razón de esta implementación es debido a que la implementación del filtro PQMF en tiempo real, requiere optimizar procesos para evitar latencia y distorsión en la señal como se evidencia en la Figura 7–6. Este algoritmo permite simplificar dos bloques iterativos de 512 muestras en uno solo, como se aprecia en las dos primeras partes en la Figura 4–3, del diagrama del PQMF.

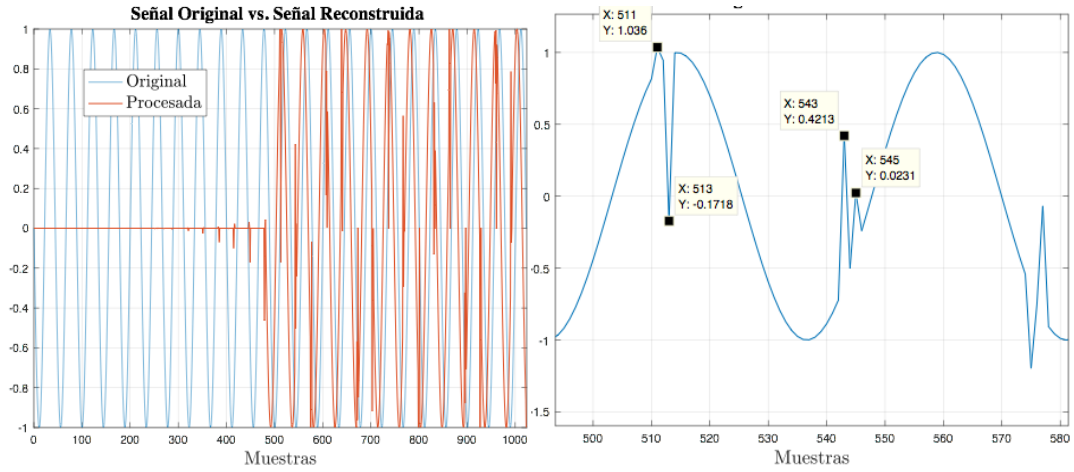
Figura 7–6: Primera reconstrucción de la señal pasando por PQMF.



Al implementar el algoritmo de la Figura 7–5, se genera una mejora en la señal reconstruida en cuanto a la frecuencia como puede verse en la Figura 7–7 izquierda. Sin embargo, aún se

presentan fuertes distorsiones en la amplitud de la señal reconstruida. Al realizar una ampliación, el cual se presenta en la Figura 7–7 derecha, es notable que dichas distorsiones se generan en la muestra 32 y 2da; en posiciones dentro del arreglo corresponden a la 31 y 1 respectivamente.

Figura 7–7: Señal reconstruida al implementar el algoritmo de indexación (Zoom Fig. derecha).



Una solución simple a este problema es realizar un promedio con el dato anterior y siguiente de cada uno de los puntos que presentan distorsiones. Esta solución es bastante óptima para cada 2da muestra dentro del arreglo, ya que se conoce el dato anterior y el siguiente para hacer el promediado, esto se realiza en 3 fragmentos de código distintos:

En el método IPQMF_Filtering():

```
float *IPQMF_Synthesis::IPQMF_Filtering(float y[]){
    ...
//Corrección por promediación de la muestra 1
    x[1] = 0.5*(x[0] + x[2]);

    return x;
}
```

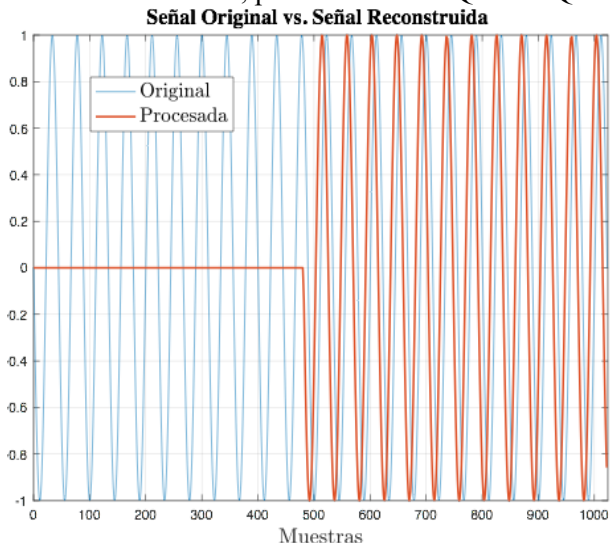
En el método processBlock() **#5 if**, sección 4.2.1:

```
void interpol_recons(float *arr){
    for (int i = 31; i < 544; i+=32)
        arr[i] = 0.5*(arr[i-1] + arr[i+1]);
}
```

En el método processBlock() **#1 if**, sección 4.2.1:

```
if (bool_postBlock) {
    for (int j = 0; j < j_end; j++)
        auxArray[j + i_end] = recons_signal[j]; //corrección por prom. cada 576 muestras
    auxArray[i_end-1] = 0.5*(auxArray[i_end-2] + recons_signal[0]);
    ips = j_end;
}
```

Figura 7–8: Señal reconstruida totalmente, para el modelo PQMF-IPQMF. Salida del Layer II.



Anexo 4: Diagramas de Bloque del Sistema

Figura 7–9: Bloque Externo del algoritmo de procesamiento del Esquema Layer III:

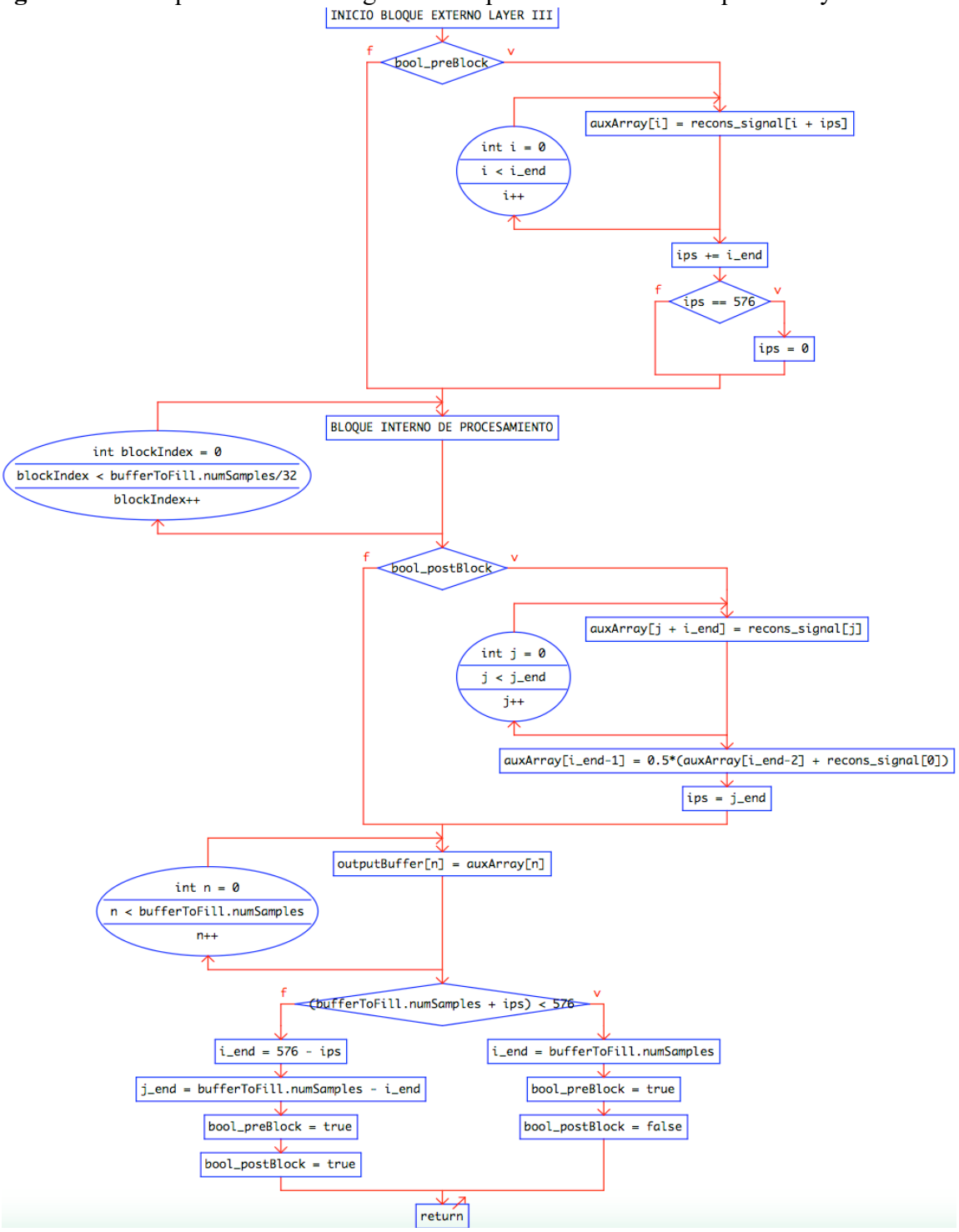
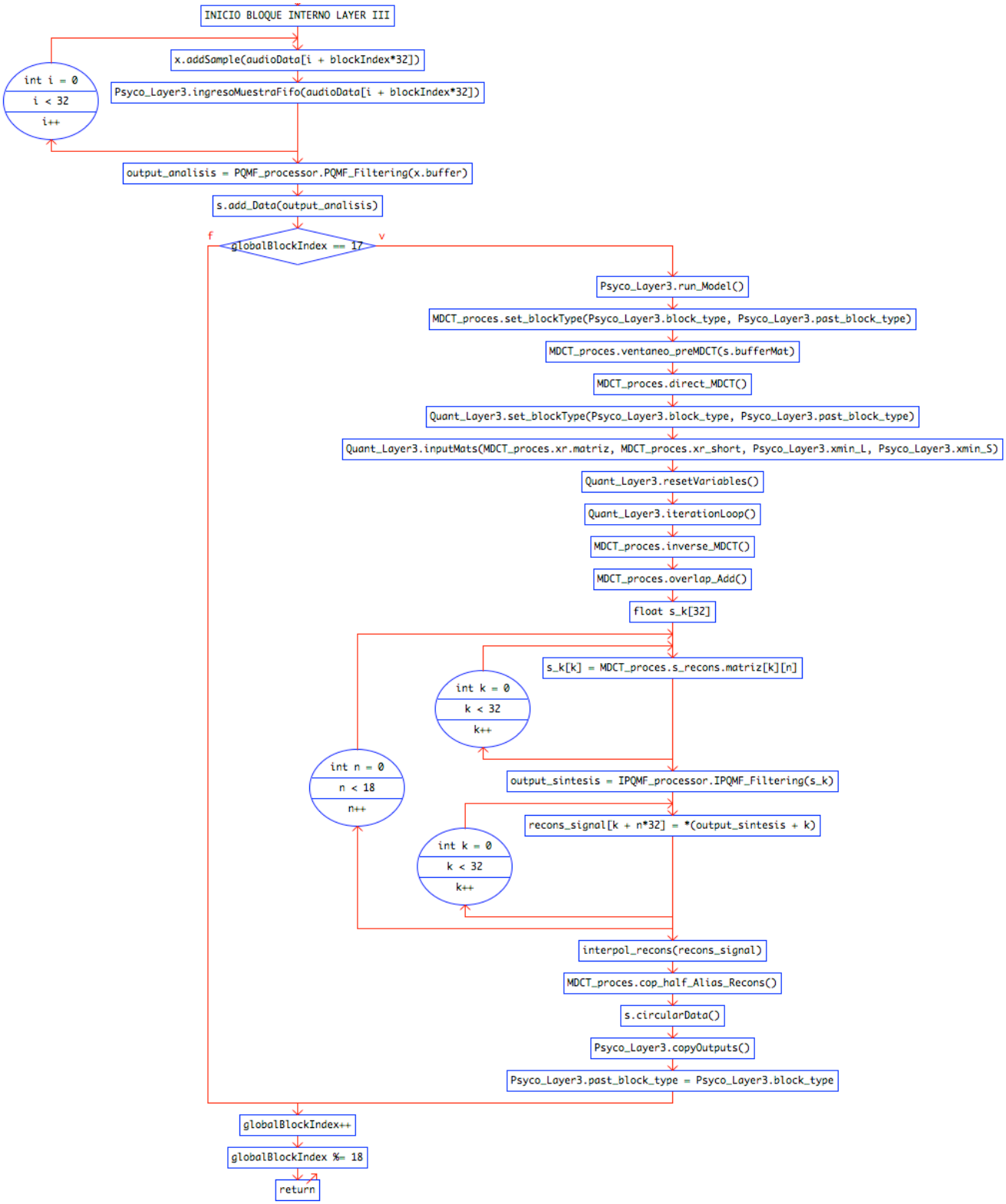


Figura 7–10: Bloque Interno del algoritmo de procesamiento del Esquema Layer III:



Anexo 5: Tablas de flujo de Asignación de muestras de salida en TR

Tabla 7-1: Flujo de la lógica de asignación de muestras de salida, para un buffer de salida de 512 muestras.

		(*), #3 if bool_preBlock		(***), #1 if bool_postBlock
i_end,ips,j_end	$y_\ell \rightarrow$	**for (i = 0; i < i_end; i++) auxArray[i]=recons_signal[i+ips]**	$y_\ell \rightarrow$	**for (j = 0; j < j_end; j++) auxArray[j+i_end]=recons_signal[j]**
null,null,512			$y_0 \rightarrow$	for (i = 0; i < 512) auxArray[i+0]=recons_signal[i]
64,512,448	$y_0 \rightarrow$	for (i = 0; i < 64) auxArray[i]=recons_signal[i+512]	$y_1 \rightarrow$	for (i = 0; i < 448) auxArray[i+64]=recons_signal[i]
128,448,384	$y_1 \rightarrow$	for (i = 0; i < 128) auxArray[i]=recons_signal[i+448]	$y_2 \rightarrow$	for (j = 0; j < 384; j++) auxArray[j+128]=recons_signal[j]
192,384,320	$y_2 \rightarrow$	for (i = 0; i < 192) auxArray[i]=recons_signal[i+384]	$y_3 \rightarrow$	for (j = 0; j < 320; j++) auxArray[j+192]=recons_signal[j]
256,320,256	$y_3 \rightarrow$	for (i = 0; i < 256) auxArray[i]=recons_signal[i+320]	$y_4 \rightarrow$	for (j = 0; j < 256; j++) auxArray[j+256]=recons_signal[j]
320,256,256	$y_4 \rightarrow$	for (i = 0; i < 320) auxArray[i]=recons_signal[i+256]	$y_5 \rightarrow$	for (j = 0; j < 192; j++) auxArray[j+320]=recons_signal[j]
384,192,128	$y_5 \rightarrow$	for (i = 0; i < 384) auxArray[i]=recons_signal[i+192]	$y_6 \rightarrow$	for (j = 0; j < 128; j++) auxArray[j+384]=recons_signal[j]
448,128,64	$y_6 \rightarrow$	for (i = 0; i < 448) auxArray[i]=recons_signal[i+128]	$y_7 \rightarrow$	for (j = 0; j < 64; j++) auxArray[j+448]=recons_signal[j]
512,64,null	$y_7 \rightarrow$	for (i = 0; i < 512) auxArray[i]=recons_signal[i+64]		
		Se repite la misma secuencia de arriba…		
null,null,512			$y_8 \rightarrow$	for (i = 0; i < 512) auxArray[i+0]=recons_signal[i]

Tabla 7-2: Flujo de la lógica de asignación de muestras de salida, para un buffer de salida de 256 muestras.

		(*), #3 if bool_preBlock		(***), #1 if bool_postBlock
i_end,ips,j_end	$y_\ell \rightarrow$	**for (i = 0; i < i_end; i++) auxArray[i]=recons_signal[i+ips]**	$y_\ell \rightarrow$	**for (j = 0; j < j_end; j++) auxArray[j+i_end]=recons_signal[j]**
0,0,128			$y_0 \rightarrow$	for (i = 0; i < 128) auxArray[i+0]=recons_signal[i]
256,256,null	$y_0 \rightarrow$	for (i = 0; i < 256) auxArray[i]=recons_signal[i+256]		
64,512,192	$y_0 \rightarrow$	for (i = 0; i < 64) auxArray[i]=recons_signal[i+512]	$y_1 \rightarrow$	for (j = 0; j < 192; j++) auxArray[j+64]=recons_signal[j]
256,192,null	$y_1 \rightarrow$	for (i = 0; i < 256) auxArray[i]=recons_signal[i+192]		
128,448,128	$y_1 \rightarrow$	for (i = 0; i < 128) auxArray[i]=recons_signal[i+448]	$y_2 \rightarrow$	for (j = 0; j < 128; j++) auxArray[j+128]=recons_signal[j]
256,128,null	$y_2 \rightarrow$	for (i = 0; i < 256) auxArray[i]=recons_signal[i+128]		
192,384,64	$y_2 \rightarrow$	for (i = 0; i < 192) auxArray[i]=recons_signal[i+384]	$y_3 \rightarrow$	for (j = 0; j < 64; j++) auxArray[j+192]=recons_signal[j]
256,64,null	$y_3 \rightarrow$	for (i = 0; i < 256) auxArray[i]=recons_signal[i+64]		
256,320,null	$y_3 \rightarrow$	for (i = 0; i < 256) auxArray[i]=recons_signal[i+320]		
		Se repite la misma secuencia de arriba…		
0,0,256			$y_4 \rightarrow$	for (i = 0; i < 256) auxArray[i]=recons_signal[i+0]

Tabla 7-3: Flujo de la lógica de asignación de muestras de salida, para un buffer de salida de 128 muestras.

		(*), #3 if bool_preBlock		(***), #1 if bool_postBlock
i_end,ips,j_end	$y_\ell \rightarrow$	**for (i = 0; i < i_end; i++) auxArray[i]=recons_signal[i+ips]**	$y_\ell \rightarrow$	**for (j = 0; j < j_end; j++) auxArray[j+i_end]=recons_signal[j]**
0,0,128			$y_0 \rightarrow$	for (i = 0; i < 128) auxArray[i+0]=recons_signal[i]
128,128,null	$y_0 \rightarrow$	for (i = 0; i < 128) auxArray[i]=recons_signal[i+128]		
128,256,null	$y_0 \rightarrow$	for (i = 0; i < 128) auxArray[i]=recons_signal[i+256]		
256,384,null	$y_0 \rightarrow$	for (i = 0; i < 128) auxArray[i]=recons_signal[i+384]		
64,512,64	$y_0 \rightarrow$	for (i = 0; i < 64) auxArray[i]=recons_signal[i+512]	$y_1 \rightarrow$	for (j = 0; j < 64; j++) auxArray[j+64]=recons_signal[j]
128,64,null	$y_1 \rightarrow$	for (i = 0; i < 128) auxArray[i]=recons_signal[i+64]		
128,192,null	$y_1 \rightarrow$	for (i = 0; i < 128) auxArray[i]=recons_signal[i+192]		
256,64,null	$y_1 \rightarrow$	for (i = 0; i < 128) auxArray[i]=recons_signal[i+320]		
128,448,null	$y_1 \rightarrow$	for (i = 0; i < 128) auxArray[i]=recons_signal[i+448]		
		Se repite la misma secuencia de arriba…		
0,0,256			$y_2 \rightarrow$	for (i = 0; i < 128) auxArray[i+0]=recons_signal[i+0]

Tabla 7-4: Flujo de la lógica de asignación de muestras de salida, para un buffer de salida de 64 muestras.

		(*), #3 if bool_preBlock		(***), #1 if bool_postBlock
i_end,ips,j_end	$y_\ell \rightarrow$	**for (i = 0; i < i_end; i++) auxArray[i]=recons_signal[i+ips]**	$y_\ell \rightarrow$	**for (j = 0; j < j_end; j++) auxArray[j+i_end]=recons_signal[j]**
0,0,64			$y_0 \rightarrow$	for (i = 0; i < 64) auxArray[i+0]=recons_signal[i]
64,64,null	$y_0 \rightarrow$	for (i = 0; i < 64) auxArray[i]=recons_signal[i+64]		
64,128,null	$y_0 \rightarrow$	for (i = 0; i < 64) auxArray[i]=recons_signal[i+128]		
64,384,null	$y_0 \rightarrow$	for (i = 0; i < 64) auxArray[i]=recons_signal[i+192]		
64,254,null	$y_0 \rightarrow$	for (i = 0; i < 64) auxArray[i]=recons_signal[i+256]		
64,320,null	$y_0 \rightarrow$	for (i = 0; i < 64) auxArray[i]=recons_signal[i+320]		
64,384,null	$y_0 \rightarrow$	for (i = 0; i < 64) auxArray[i]=recons_signal[i+384]		
64,448,null	$y_0 \rightarrow$	for (i = 0; i < 64) auxArray[i]=recons_signal[i+448]		
64,512,null		for (i = 0; i < 64) auxArray[i]=recons_signal[i+512]		
		Se repite la misma secuencia de arriba…		
0,0,256			$y_1 \rightarrow$	for (i = 0; i < 64) auxArray[i+0]=recons_signal[i+0]

Anexo 6: Códigos del sistema implementado

PQMF_Analysis.h

```
#pragma once
#include <math.h>
#include <cstring>
#include <iostream>

class CircularPQMF_Buffer{
public:
    CircularPQMF_Buffer();
    ~CircularPQMF_Buffer();

    void  clear();
    void  setBufferSize(int size);
    void  addSample(float sample);
    float evaluateBuffer(int indice);

    void internalPrint(){
        for (int i = 0; i < bufferSize; i++)std::cout << "buffer[" << i << "] = " << buffer[i] << std::endl;
    }

    float *buffer;

private:
    int index, i = 0;
    int bufferSize;
};


extern float c[512];

class PQMF_Analysis{
public:
    PQMF_Analysis(){
        memset(z, 0, sizeof(z));
        memset(s, 0, sizeof(s));
        part_sum = 0;
        part_sum2 = 0;
        block_index = 0;
        num_blocks = 16;
        jj = 0;

        //cálculo de la matriz moduladora M para el análisis
        for(int k = 0; k < 32; k++){
            for (int r = 0; r < 64; r++)
                M_k_r[k][r] = cos((k+0.5)*(r-16)*(M_PI/32));
        }
    };
    ~PQMF_Analysis(){};

    void setNumberBlocks(int);
    float *PQMF_Filtering(float x[]);

private:
    int block_index, num_blocks, jj;
    float z[512];
    float s[64];
    float M_k_r[32][64];
    float part_sum, part_sum2;
};


class s_ProcessBandCoef_MAT{
public:
    s_ProcessBandCoef_MAT();
    ~s_ProcessBandCoef_MAT();

    void clear();
    void setBufferDim(int size_Row, int size_Col);
    void add_Data(float s[]);

    //el método "circularData" debe llamarse despues de que bufferMat se llene de 576 nuevos datos
    //y que luego de lo anterior se haga una copia externa de bufferMat para ventanear las filas
```

```
    void circularData();

    void printer(){
        for (int col = 0; col < 36; col++)
            for(int row = 0; row < 32; row++)
                std::cout << "bufferMat[" << row << "][" << col << "] = " << bufferMat[row][col] << std::endl;
    }
    float **bufferMat;
private:
    int index_col;
    const int K, N; //N columnas, K filas
    int col_cont;
};
```

PQMF_Analysis.cpp

```
#include "PQMF_Analysis.h"

CircularPQMF_Buffer::CircularPQMF_Buffer(){
    bufferSize = 0; index = 0; buffer = NULL;
}

CircularPQMF_Buffer::~CircularPQMF_Buffer(){
    if (buffer) {delete [] buffer;} //libera espacio en bytes utilizados anteriormente
}

void CircularPQMF_Buffer::setBufferSize(int size){
    if (buffer) {delete [] buffer;}
    bufferSize = size;
    index = bufferSize - 1;
    buffer = new float[bufferSize]; //buffer es el arreglo
    clear();
}

void CircularPQMF_Buffer::clear(){
    memset(buffer, 0, bufferSize*sizeof(float));
}

void CircularPQMF_Buffer::addSample(float sample){
    buffer[index] = sample;
    if (index == 0){index = bufferSize;}
    index --;
}

float CircularPQMF_Buffer::evaluateBuffer(int indice){
    return buffer[indice];
}


void PQMF_Analysis::setNumberBlocks(int blocks){
    num_blocks = blocks;
}

float *PQMF_Analysis::PQMF_Filtering(float x[]){

    block_index %= 16; //siempre x[] debe tener un tamaño de 512, por eso la función "setNumberBlocks" no se
usa

    jj = 0;
    for(int i = 0; i < 512; i++) {
        jj = i + 480 - block_index*32;
        if (i >= 32*(block_index + 1)){
            jj = i - 32*(block_index + 1);
        }
        z[i] = c[i] * x[jj];
    }

    block_index++;

    // for (int i = 0; i < 512; i++)  z[i] = c[i] * x[i];

    for (auto r = 0; r < 64; r++) {
        part_sum = 0;
        for (auto j = 0; j < 8; j++){
            part_sum = part_sum + z[r + 64*j];
        }
        s[r] = part_sum;
    }

    static float y[32];

    for (auto k = 0; k < 32; k++){
        part_sum2 = 0;
        for (auto r = 0; r < 64; r++){
            part_sum2 += M_k_r[k][r] * s[r];
```

```
        }
        y[k] = part_sum2;
    }

    return y;
}


s_ProcessBandCoef_MAT::s_ProcessBandCoef_MAT(): K(32), N(36){

    index_col = 0;
    bufferMat = NULL;

    col_cont = N/2; //se ingresan los datos de la mitad en adelante, se inicia en la mitad obviamente

    setBufferDim(K, N);
}

s_ProcessBandCoef_MAT::~s_ProcessBandCoef_MAT(){
    clear();
}

void s_ProcessBandCoef_MAT::clear(){
    for (int k = 0; k < K; ++k){
        if (bufferMat[k] != 0)
            delete [] bufferMat[k];
    }
    if (bufferMat != 0)
        delete [] bufferMat;
}

void s_ProcessBandCoef_MAT::setBufferDim(int size_Row, int size_Col){
    bufferMat = new float*[size_Row];
    for (int k = 0; k < size_Row; k++)
        bufferMat[k] = new float[size_Col];

    for (int k = 0; k < K ; k++){
        for (int n = 0; n < N; n++)
            bufferMat[k][n] = 0.0;
    }
}

void s_ProcessBandCoef_MAT::add_Data(float *s){

    //printer(); std::cout << "col_cont = " << col_cont << std::endl; std::cout << "N = " << N << std::endl;

    if (col_cont == N) //si col_cont llegase hasta N-1, nunca ingresarían nuevos datos a la ultima columna de
bufferMat
        col_cont = N/2; //std::cout << "se llenó de 576!" << std::endl;


    for(int k = 0; k < K; k++) //se ingresan 32 datos de s[] a las 32 filas de una sola columna
        bufferMat[k][col_cont] = s[k];

    col_cont++;

    //apenas se ingresen 576 nuevos valores a la matriz y se haga una copia
    //externa de bufferMat, la segunda mitad de bufferMat se copia a la primera mitad de bufferMat en
circularData
}

void s_ProcessBandCoef_MAT::circularData(){

    for (int n = 0; n < N/2; n++)
        for(int k = 0; k < K; k++)
            bufferMat[k][n] = bufferMat[k][n+18];
}
```

MDCT_processing.h

```
#pragma once
#include <stdio.h>
#include <iostream>
#include "Psychoacoustic_Mod.h"

class MDCT_Matrix{ //esta clase se usa internamente para generar varias matrices en la clase MDCT_processing
public:

    float **matriz;

    void setSizes(int sizeFilas, int sizeCols){num_filas = sizeFilas; num_cols = sizeCols;}

    void setMatriz(){
        matriz = new float *[num_filas];
        for (int fila = 0; fila < num_filas; fila++)
```

```
            matriz[fila] = new float[num_cols];

        for (int fila = 0; fila < num_filas; fila++){ //se inician rellenos de 0
            for (int col = 0; col < num_cols; col++)
                matriz[fila][col] = 0.0f;
        }
    }

    ~MDCT_Matrix(){ destroyMatriz(); }

    void destroyMatriz(){

        for (int i = 0; i < num_filas; i++)
            if (matriz[i] != 0)
                delete [] matriz[i];

        if (matriz != 0) delete [] matriz;
    }

private:
    int num_filas, num_cols;
};


class MDCT_processing{
public:

    MDCT_processing();
    ~MDCT_processing();

    void setProcessMatrices(float **matrix, int num_rows, int num_cols);

    void set_blockType(int blockType, int pastBlockType){
        block_type = blockType; past_block_type = pastBlockType;
    }
    //block_type = 0:Long, 1:Start, 2:Short, 3:Stop

    void ventaneo_preMDCT(float **s);

    void direct_MDCT();
    void inverse_MDCT();

    void overlap_Add();
    void cop_half_Alias_Recons();

    /*float Ψ[32][18]; //PRIMERO se ventanea, luego se transforma usando MDCT, debe recuantizarse usando modelo
psicoacústico, por eso debe ser pública para acceder a esta info desde otra clase (clase de modelo
psicoacústico)
     float γ_Aliased[32][36]; //transformada inversa IMDCT de Ψ
     float γ_second_Half[16][18]; //segunda mitad de γ_Aliased
     */

    MDCT_Matrix Ψ, γ_Aliased, γ_Al_second_Half, s_recons;
    float ***Ψ_short; //float Ψ_short[3][32][6]; //Matriz tridimensional de la MDCT para los 3 bloques short.


private:
    //Psychoacoustic_Mod2_Layer3 Psyco_Layer3;

    int block_type, past_block_type;
    const int N, K; //N columnas, K filas
    int index_col;

    float C_A_2[32][36], //Bases de la MDCT.
          C_S_2[36][32]; //Bases de la IMDCT.
    float w_long[36],    //ventanas (LONG, START, STOP y SHORT(solo en 12 valores la función es distinta de
0)).
          w_start[36], w_stop[36], w_short[12];
    float γ[32][36], //matriz s (salidas del PQMF) ventaneado, que será ingresado al método direct_MDCT().
        γ_short[3][32][12], //matriz s tridimensional, para los 3 bloques short cada uno con 12 muestras de
tiempo.
        γ_Aliased_short[3][32][12],
        γ_Al_secHalf_short[3][32][12],
        cs[8], ca[8]; //Coeficientes mariposa antiAliasing
    float xr[576], xar[576];

};
```

MDCT_processing.cpp

```
#include "MDCT_processing.h"
#include <cmath>

MDCT_processing::MDCT_processing(): N(36), K(32){
```

```
    // Transformada directa MDCT:
    Ψ.setSizes(K, N/2); Ψ.setMatriz();
    // Transformada inversa IMDCT:
    γ_Aliased.setSizes(K, N); γ_Aliased.setMatriz();
    // Copia de la segunda mitad de γ_Aliased:
    γ_Al_second_Half.setSizes(K, N/2); γ_Al_second_Half.setMatriz();
    // Matriz de salida, producto del Overlap & Add:
    s_recons.setSizes(K, N/2); s_recons.setMatriz();

    Ψ_short = new float **[3];
    for (int i = 0; i < 3; i++){
        Ψ_short[i] = new float *[K];
        for (int k = 0; k < K; k++)
            Ψ_short[i][k] = new float[6];
    }

    for (int i = 0; i < 3; i++)
        for (int k = 0; k < K; k++)
            for (int ξ = 0; ξ < 6; ξ++)
                Ψ_short[i][k][ξ] = 0.0;

    index_col = 0;

    for (int n = 0; n < N; n++){
        for (int ξ = 0; ξ < N/2; ξ++){
            C_A_2[ξ][n] = cos((2*M_PI/N)*(n+N/4+0.5)*(ξ+0.5));
            C_S_2[n][ξ] = float(4.0/N)*cos((2*M_PI/N)*(n+N/4+0.5)*(ξ+0.5));
        }
    }

    //Definición de los 4 tipos de ventana:
    for (int n = 0; n < N; n++){
        w_long[n] = sin((M_PI/N)*(n+0.5));
        w_start[n] = sin((M_PI/N)*(n+0.5));
        w_stop[n] = 0.0f;

        if(n > 17)
            w_start[n] = 1.0f;
        if(n > 23)
            w_start[n] = sin((M_PI/12.0f)*(n - 18 + 0.5));
        if(n > 29)
            w_start[n] = 0.0f;

        if(n > 5)
            w_stop[n] = sin((M_PI/12.0f)*(n - 6 + 0.5));
        if(n > 11)
            w_stop[n] = 1.0f;
        if(n > 17)
            w_stop[n] = sin((M_PI/N)*(n+0.5));
    }

    for (int n = 0; n < N/3; n++) //n = 0, 1,..., 11
        w_short[n] = sin((M_PI/12.0f)*(n + 0.5));

    float c[8] = {-0.6, -0.535, -0.33, -0.185, -0.095, -0.041, -0.0142, -0.0037};
    for (int i = 0; i < 8; i++) {
        cs[i] = 1/std::sqrt(1 + powf(c[i], 2.0f));
        ca[i] = c[i]/std::sqrt(1 + powf(c[i], 2.0f));
    }

    //memset(xr, 0.0f, sizeof(xr)); memset(xar, 0.0f, sizeof(xar));
    //setProcessMatrices(Ψ, K, N/2); //setProcessMatrix();
}

MDCT_processing::~MDCT_processing(){}

//no se está usando:
void MDCT_processing::setProcessMatrices(float **matrix, int num_rows, int num_cols){

    for (int k = 0; k < num_rows; k++)
        for (int j = 0; j < num_cols; j++)
            matrix[k][j] = 0.0f; //se llena de ceros.
}

void MDCT_processing::ventaneo_preMDCT(float **s){

    switch (block_type){
        case 0: //LONG
            for (int k = 0; k < K; k++)
                for (int n = 0; n < N; n++)
                    γ[k][n] = s[k][n] * w_long[n];
            break;
        case 1: //START
            for (int k = 0; k < K; k++)
                for (int n = 0; n < N; n++)
```

```
                    γ[k][n] = s[k][n] * w_start[n];
            break;
        case 2: //SHORT
            for (int k = 0; k < K; k++)
                for (int n = 0; n < N/3; n++){ //n = 0, 1,..., 11
                    γ_short[0][k][n] = s[k][n + 6] * w_short[n];
                    γ_short[1][k][n] = s[k][n + 12] * w_short[n];
                    γ_short[2][k][n] = s[k][n + 18] * w_short[n];
                }
            break;
        case 3: //STOP
            for (int k = 0; k < K; k++)
                for (int n = 0; n < N; n++)
                    γ[k][n] = s[k][n] * w_stop[n];
            break;
    }
}

void MDCT_processing::direct_MDCT(){

    float part_sum;

    if (block_type == 2){
        for (int bloque = 0; bloque < 3; bloque++)
            for (int k = 0; k < K; k++){ //sub-banda k = 0, 1,..., 31
                for (int ξ = 0; ξ < N/6; ξ++){ //frecuencia ξ = 0, 1,..., 6 (la mitad de n)
                    part_sum = 0;
                    for (int n = 0; n < N/3; n++) //tiempo n = 0, 1,..., 11
                        part_sum += γ_short[bloque][k][n] * C_A_2[ξ][n];
                    //los datos de Ψ_short deben ser recuantizados según el modelo psicoacústico
                    Ψ_short[bloque][k][ξ] = part_sum;
                }
            }
    }else{

        for (int k = 0; k < K; k++)
            for (int ξ = 0; ξ < N/2; ξ++){
                part_sum = 0;
                for (int n = 0; n < N; n++)
                    part_sum += γ[k][n] * C_A_2[ξ][n];
                //los datos de Ψ.matriz deben ser recuantizados según el modelo psicoacústico
                Ψ.matriz[k][ξ] = part_sum;
            }
    }
}

//aquí debería estar como parámetro de entrada Ψ recuantizado, o simplemente manipular Ψ, para que al usarse en
este metodo ya esté recuantizado
void MDCT_processing::inverse_MDCT(){

    float part_sum;

    if (block_type == 2){
        for (int bloque = 0; bloque < 3; bloque++)
            for (int k = 0; k < K; k++){
                for (int n = 0; n < N/3; n++){
                    part_sum = 0;
                    for (int ξ = 0; ξ < N/6; ξ++)
                        part_sum += Ψ.matriz[k][ξ] * C_S_2[n][ξ];
                    γ_Aliased_short[bloque][k][n] = part_sum * w_short[n];
                }
            }
    }else{
        float *window;
        switch (block_type){//incluyendo ventaneo por la misma ventana que se usó en la MDCT DIRECTA
            case 0:
                window = w_long; break;
            case 1:
                window = w_start; break;
            case 3:
                window = w_stop; break;
        }
        for (int k = 0; k < K; k++){
            for (int n = 0; n < N; n++){
                part_sum = 0;
                for (int ξ = 0; ξ < N/2; ξ++)
                    part_sum += Ψ.matriz[k][ξ] * C_S_2[n][ξ];
                γ_Aliased.matriz[k][n] = part_sum * window[n];
            }
        }
    }
}

void MDCT_processing::overlap_Add(){
```

```
    if ((past_block_type == 1) && (block_type == 2)){ //Overlap & Add de ventana Start con primer gránulo
Short.
        for(int k = 0; k < K; k++)
            for(int n = 0; n < N/2; n++){
                if (n < N/6)                    //n < 6
                    s_recons.matriz[k][n] = γ_Al_second_Half.matriz[k][n];
                if ((n > N/6 - 1) && (n < N/3)) //n > 5 && n < 12
                    s_recons.matriz[k][n] = γ_Al_second_Half.matriz[k][n] + γ_Aliased_short[0][k][n - 6];
                if ((n > N/3 - 1) && (n < N/2)) //n > 11 && n < 18
                    s_recons.matriz[k][n] = γ_Aliased_short[0][k][n - 6] + γ_Aliased_short[1][k][n - 12];
            }
    }else if ((past_block_type == 2) && (block_type == 2)){ //Overlap & Add entre bloques Short sucesivos.
        for(int k = 0; k < K; k++)
            for(int n = 0; n < N/2; n++){
                if (n < N/6)                    //n < 6
                    s_recons.matriz[k][n] = γ_Al_secHalf_short[1][k][n + 6] + γ_Al_secHalf_short[2][k][n];
                if ((n > N/6 - 1) && (n < N/3)) //n > 5 && n < 12
                    s_recons.matriz[k][n] = γ_Al_secHalf_short[2][k][n] + γ_Aliased_short[0][k][n - 6];
                if ((n > N/3 - 1) && (n < N/2)) //n > 11 && n < 18
                    s_recons.matriz[k][n] = γ_Aliased_short[0][k][n - 6] + γ_Aliased_short[1][k][n - 12];
            }
    }else if (block_type == 3){ //Overlap & Add de último gránulo Short con ventana Stop actual.
        for(int k = 0; k < K; k++)
            for(int n = 0; n < N/2; n++){
                if (n < N/6)                    //n < 6
                    s_recons.matriz[k][n] = γ_Aliased_short[1][k][n + 6] + γ_Aliased_short[2][k][n];
                if ((n > N/6 - 1) && (n < N/3)) //n > 5 && n < 12
                    s_recons.matriz[k][n] = γ_Aliased_short[2][k][n] + γ_Aliased.matriz[k][n];
                if ((n > N/3 - 1) && (n < N/2)) //n > 11 && n < 18
                    s_recons.matriz[k][n] = γ_Aliased.matriz[k][n];
            }
    }else{ //Overlap & Add entre bloques Long, Long-Start o Stop-Long.
        for(int k = 0; k < K; k++)
            for(int n = 0; n < N/2; n++)
                s_recons.matriz[k][n] = γ_Al_second_Half.matriz[k][n] + γ_Aliased.matriz[k][n];
    }
}

void MDCT_processing::cop_half_Alias_Recons(){

    if ((block_type == 2) && (past_block_type == 2)){
        for (int k = 0; k < K; k++)
            for (int n = 0; n < N/3; n++){
                γ_Al_secHalf_short[0][k][n] = γ_Aliased_short[0][k][n];
                γ_Al_secHalf_short[1][k][n] = γ_Aliased_short[1][k][n];
                γ_Al_secHalf_short[2][k][n] = γ_Aliased_short[2][k][n];
            }
    }else{
        for (int k = 0; k < K; k++)
            for (int n = N/2; n < N; n++)
                γ_Al_second_Half.matriz[k][n - N/2] = γ_Aliased.matriz[k][n];
    }
}
```

IPQMF_Synthesis.h

```
#pragma once

#include <iostream>
#include <stdio.h>
#include <math.h>

extern float d[512];

class IPQMF_Synthesis{
public:

    IPQMF_Synthesis(){
        memset(v, 0.0f, sizeof(v));
        memset(u, 0.0f, sizeof(u));
        memset(w, 0.0f, sizeof(w));
        part_sum = 0;
        part_sum2 = 0;

        //cálculo de la matriz moduladora N para la síntesis
        for(int k = 0; k < 32; k++){
            for (int r = 0; r < 64; r++) {
                N_k_r[k][r] = cos((2*k+1)*(r+16)*(M_PI/64));
            }
        }
    };

    ~IPQMF_Synthesis(){};

    float *IPQMF_Filtering(float y[]);
```

```
private:

    float v[1024];
    float u[512];
    float w[512];
    float N_k_r[32][64];
    float part_sum, part_sum2;
};
```

IPQMF_Synthesis.cpp

```
#include "IPQMF_Synthesis.h"

float *IPQMF_Synthesis::IPQMF_Filtering(float y[]){

    for(int i = 1023; i >= 64; i--)
        v[i] = v[i - 64];

    for (auto i = 0; i < 64; i++) {
        part_sum = 0;
        for (auto k = 0; k < 32; k++) {
            part_sum += N_k_r[k][i] * y[k];
        }
        v[i] = part_sum;
    }

    for(int i = 0; i < 8; i++){
        for(int j = 0; j < 32; j++){
            u[64*i+j]    = v[128*i + j];
            u[64*i+32+j] = v[128*i + 96 + j];
        }
    }

    for (int i = 0; i < 512; i++)
        w[i] = d[i] * u[i];

    static float x[32];

    for (int j = 0; j < 32; j++){
        part_sum2 = 0;
        for (auto i = 0; i < 16; i++) {
            part_sum2 += w[j + 32*i];
        }
        x[j] = part_sum2;
    }

    //Corrección por promediación de la muestra 1
    x[1] = 0.5*(x[0] + x[2]);

    return x;
}
//si se usan los c[n] y d[n] como dicta la norma, las muestras x[1] y x[31] salen distorsionadas
//si se usan los c[n] y d[n] base (normales), solo la muestra x[0] sale distorsionada
```

Psychoacoustic_Mod.h

```
#pragma once
#include <math.h>
#include <iostream>
#include <algorithm>
#include "../JuceLibraryCode/JuceHeader.h"

extern float C_7_48k_Long[6][62], C_8_48k_Long[6][21],
C_7_44k_Long[6][63], C_8_44k_Long[6][21],
C_7_32k_Long[6][59], C_8_32k_Long[6][21],
C_7_48k_Short[6][38], C_8_48k_Short[6][12],
C_7_44k_Short[6][39], C_8_44k_Short[6][12],
C_7_32k_Short[6][42], C_8_32k_Short[6][12];
/* PROCEDIMIENTO
1. setMatrices();
2. fs_select(double fs); -> copyMat(float aux_mat[rows][cols], int opt_mat);
3. calc_sprdngf();
4. ingresoMuestraFifo(float sample); Hacerlo 576 veces
5. run_Model():
    5.1 transformFFT(); Hacerlo inmediatamente después de que se llenen el fifo con 576 muestras
    5.2 unpredictability_cw();
    5.3 calc_umbra_parte1();
    if PE > 1800
        5.4 calc_umbra_ShortBlock();
        5.5 calc_umbra_parte2(); para bloques Short.
        3 veces se ejecuta (cada bloque short) estos 2 métodos.
    else
        5.4 calc_umbra_parte2(); para bloques Long
```

```
*/
//LAYER III
class Psychoacoustic_Mod2_Layer3{
public:
    Psychoacoustic_Mod2_Layer3():
    forward_LongFFT   (fftLongOrder),
    forward_3ShortFFT (fftShortOrder)
    {
        setMatrices();
        index = 0; s_block = 0;
        block_type = 6; past_block_type = 10; //valores iniciales
    }

    ~Psychoacoustic_Mod2_Layer3(){}
    //La definición usada acá (C++ 14) para las matrices dinámicas permite redefinirlos sin tener que
destruirlos antes
    //destruir arreglos dinámicos !!!

    void setMatrices();
    void fs_select(double fs); //Llamar en prepareToPlay(,)

    template <size_t rows, size_t cols>
    void copyMat(float aux_mat[rows][cols], int opt_mat);

    void calc_sprdngf();

    void ingresoMuestraFifo(float sample);
    void transformFFT();

    void unpredictability_cw();
    void calc_umbra_parte1();
    void calc_umbra_ShortBlock();
    void calc_umbra_parte2();
    void copyOutputs();
    void run_Model();

    //Salida del modelo psicoacústico, Relación señal a enmascarador para bloques Long y Short
    float SMR_L[21], SMR_S[3][12],

    // Energía en cada banda de los Scalefactors;
          en_L[21], en_S[3][12],

    //xmin : distorsión permitida en las bandas de los scalefactors. _L: bloques "Long", _S: bloques "Short"
    //xmin_L: 2 filas referentes al gránulo actual y al anterior,
    //xmin_S: 3 filas referentes a los 3 trozos de 2 gránulos,
        **xmin_L, ***xmin_S; //xmin_L[2][21], xmin_S[2][3][12];

    //Entropía Perceptual
    float PE;
    int block_type, past_block_type; //0:LONG, 1:START, 2:SHORT, 3:STOP

private:
    enum{fftLongOrder = 10, fft_L_Size  = 1 << fftLongOrder,
        fftShortOrder = 8, fft_S_Size  = 1 << fftShortOrder};

    dsp::FFT forward_LongFFT; //1 objeto de FFT para una FFT de 1024
    dsp::FFT forward_3ShortFFT; //1 objetos de FFT para 3 FFTs de 256

    //Matrices locales de las tablas C.7 y C.8 según la frecuencia de muestreo seleccionada
    float **C_7_Long, **C_7_Short, **C_8_Long, **C_8_Short; //6 filas, N columnas
    float **aux_mat;
    //Cantidad de particiones b del dominio de Bark: C_7[0][last_index+1]
    int b_L, b_S; //para bloques Long y Short

    //Arreglo para copiar el audio de entrada:
    float fifo[576]; int index;

    //L:Long, S: Short
    //Matrices para las transformaciones en frecuencia del audio de entrada:
    //Cada fila en la matriz Long corresponde a las FFTs de 1024 datos del granulo actual (2), el pasado (1) y
el antepasado (0).
    //float fft_L_Data[3][2*fft_L_Size],
    float fft_L_Data[2*fft_L_Size],
    // FFTs de 256 puntos de 192 datos temporales consecutivos de un solo granulo de 576 muestras.
          fft_3_S_Data[3][2*fft_S_Size]; //las 3 filas contienen la info de 3 bloques short consecutivos del
mismo gránulo.

    int s_block; //índice de la fila de las matrices short, que corresponde a la info de alguno de los 3
bloques short

    //Reinterpretación de las matrices pasadas a formato complejo: std::complex<float> = (Re{}, Im{})
    std::complex<float> ** fft_Complex_L, **fft_Complex_3S;

    //Ventanas Hann, de tamaño del gránulo y del tercio de uno
    float w_L_hann[576], w_S_hann[192];
```

```
    //Arreglo de medidas de impredecibilidad cw (de longitud variable -suma de FFTLines-)
    float *cw; int cw_size;

    //Matriz de la función de esparcimiento sprdngf
    float **sprdngf_L, **sprdngf_S;
    //Arreglos de las energías sin normalizar de las FFTs Long y Short
    float *eb_L, *eb_S;
    //Arreglo de la energía normalizada con la impredecibilidad cw
    float *cb;
    //Arreglos de las energías mapeadas al dominio de bark
    float *ecb_L, *ecb_S, *ctb;
    //Índices de tonalidad, SRN_Long
    float *tbb, *SNR_L;
    //Umbrales en cada partición para bloques Long(matriz) y Short(arreglo)
    float **nbb_L, *nbb_S;
    //Umbrales actuales de los granulos, para bloques Long y Short
    float *thr_L, *thr_S;

    //Umbral en cada banda de los scalefactors
    float thm_L[21], thm_S[3][12]; //, en_L[21], en_S[3][12];
    //

};

//-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-
//LAYER II
//-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-
extern float D_4_32k[3][132], D_4_44k[3][130],D_4_48k[3][126],
D_3_32k[6][49], D_3_44k[6][57], D_3_48k[6][58];
extern int D_5[3][33];

class Psychoacoustic_Mod2_Layer2{
public:

    Psychoacoustic_Mod2_Layer2():
    forward_FFT (fftOrder){
        setMatrices();
    }

    ~Psychoacoustic_Mod2_Layer2(){}

    void setMatrices();
    void fs_select(double fs);
    template <size_t rows, size_t cols>
    void copyMat(float aux_mat[rows][cols], int opt_mat);

    void calc_sprdngf();

    void ingresoMuestraFifo(float sample);
    void transformFFT();

    void unpredictability_cw();
    void calc_umbral();
    void copyOutputs();
    void run_Model();

    float  **SMR;//SMR[2][32]
private:

    enum{fftOrder = 10, fft_Size = 1 << fftOrder};
    dsp::FFT forward_FFT; //1 objeto de FFT para una FFT de 1024

    float **D_4, **D_3,
          **aux_mat,
          **sprdngf,
          *cw,
          *eb, *cb,
          *ecb, *ctb, *cbb, *rnorm, *enb,
          *tbb, *SNR,
          *nbb, *nbw, *thr,
    epart[32], npart[32],

    fifo[576], fft_Data[2*fft_Size], w_hann[576];

    std::complex<float> ** fft_Complex;

    int b_max, index, cw_size, absthr_imax;
};
```

Psychoacoustic_Mod.cpp

```
#include "Psychoacoustic_Mod.h"

void Psychoacoustic_Mod2_Layer3::setMatrices(){
```

```
    for (int n = 0; n < 576; n++)
        w_L_hann[n] = 0.5 - 0.5*cos((2*M_PI/576)*(n - 0.5));
    for (int n = 0; n < 192; n++)
        w_S_hann[n] = 0.5 - 0.5*cos((2*M_PI/192)*(n - 0.5));

    memset(fifo, 0, sizeof(fifo)); //se llenan de ceros las matrices
    memset(fft_L_Data, 0, sizeof(fft_L_Data));
    memset(fft_3_S_Data, 0, sizeof(fft_3_S_Data));
    memset(en_L, 0, sizeof(en_L));
    memset(en_S, 0, sizeof(en_S));

    // se ubican dinámicamente las matrices complejas
    fft_Complex_L = new std::complex<float> *[3];
    fft_Complex_3S = new std::complex<float> *[3];
    for (int fila = 0; fila < 3; fila++){
        fft_Complex_L[fila] = new std::complex <float>[fft_L_Size]; //✓
        fft_Complex_3S[fila] = new std::complex <float>[fft_S_Size]; //✓
    }

    xmin_S = new float **[2];
    for (int i = 0; i < 2; ++i){
        xmin_S[i] = new float *[3];
        for (int j = 0; j < 3; ++j)
            xmin_S[i][j] = new float[12];
    }
    for (int i = 0; i < 2; ++i)
        for (int j = 0; j < 3; ++j)
            for (int k = 0; k < 12; ++k)
                xmin_S[i][j][k] = 0;

    xmin_L = new float *[2];
    for (int fila = 0; fila < 2; fila++) xmin_L[fila] = new float [21];
    for (int i = 0; i < 2; i++) //forma de llenar de ceros unas matriz dinámica:
        std::fill(xmin_L[i], xmin_L[i] + 21, 0);

    //se llenan de 0 las matrices complejas
    for (int fila = 0; fila < 3; fila++){
        std::fill(fft_Complex_L[fila], fft_Complex_L[fila] + fft_L_Size, std::complex<float>{0,0});
        std::fill(fft_Complex_3S[fila], fft_Complex_3S[fila] + fft_S_Size, std::complex<float>{0,0});
    }

}

template <size_t rows, size_t cols>
void Psychoacoustic_Mod2_Layer3::copyMat(float C_Table[rows][cols], int opt_mat){

    aux_mat = new float *[rows];
    for (int i = 0; i < rows; i++)
        aux_mat[i] = new float[cols];

    for (int fila = 0; fila < rows; fila++)
        for (int col = 0; col < cols; col++)
            aux_mat[fila][col] = C_Table[fila][col];

    switch (opt_mat) {
        case 0: C_7_Long = aux_mat; b_L = cols;  break;
        case 1: C_7_Short = aux_mat; b_S = cols; break;
        case 2: C_8_Long = aux_mat;  break;
        case 3: C_8_Short = aux_mat; break;
    }
}

void Psychoacoustic_Mod2_Layer3::fs_select(double fs){
    //llamar método fs_select(sample_rate) desde el getnextaudioblock
    //dependiendo de la frecuencia de muestreo se generan unas nuevas matrices dinámicas para las tablas C.7
Long y Short
    switch (int(fs)) {
        case 48000:
            copyMat<6>(C_7_48k_Long, 0); copyMat<6>(C_7_48k_Short, 1);
            copyMat<6>(C_8_48k_Long, 2); copyMat<6>(C_8_48k_Short, 3); break;
        case 44100:
            copyMat<6>(C_7_44k_Long, 0); copyMat<6>(C_7_44k_Short, 1);
            copyMat<6>(C_8_44k_Long, 2); copyMat<6>(C_8_44k_Short, 3); break;
        case 32000:
            copyMat<6>(C_7_32k_Long, 0); copyMat<6>(C_7_32k_Short, 1);
            copyMat<6>(C_8_32k_Long, 2); copyMat<6>(C_8_32k_Short, 3); break;
        default:
            copyMat<6>(C_7_44k_Long, 0); copyMat<6>(C_7_44k_Short, 1);
            copyMat<6>(C_8_44k_Long, 2); copyMat<6>(C_8_44k_Short, 3); break;
    }
    // copy to: C_7_Short ✓, C_7_Long ✓, C_8_Short ✓, C_8_Long ✓

    cw_size = 0;
    for (int j = 0; j < b_L; j++)
```

```
        cw_size += C_7_Long[1][j]; // se suman los valores de la columna "FFT-Lines"

    if (cw) {delete [] cw;} cw = new float[cw_size];
    if (eb_L) {delete [] eb_L;} eb_L = new float[b_L];
    if (eb_S) {delete [] eb_S;} eb_S = new float[b_S];
    if (cb) {delete [] cb;} cb = new float[b_L];
    if (ecb_L) {delete [] ecb_L;} ecb_L = new float[b_L];
    if (ecb_S) {delete [] ecb_S;} ecb_S = new float[b_S];
    if (ctb) {delete [] ctb;} ctb = new float[b_L];

    sprdngf_L = new float *[b_L];
    for (int fila = 0; fila < b_L; fila++) sprdngf_L[fila] = new float [b_L];

    sprdngf_S = new float *[b_S];
    for (int fila = 0; fila < b_S; fila++) sprdngf_S[fila] = new float [b_S];
    calc_sprdngf();

    if (tbb) {delete [] tbb;} tbb = new float[b_L];
    if (SNR_L) {delete [] SNR_L;} SNR_L = new float[b_L];

    nbb_L = new float *[3];//info filas: (2) gránulo actual, (1) pasado y (0) antepasado
    for (int fila = 0; fila < 3; fila++) nbb_L[fila] = new float [b_L];
    for (int i = 0; i < 3; i++) //forma de llenar de ceros unas matriz dinámica:
        std::fill(nbb_L[i], nbb_L[i] + b_L, 0);

    if (nbb_S) {delete [] nbb_S;} nbb_S = new float[b_S];
    if (thr_L) {delete [] thr_L;} thr_L = new float[b_L];
    if (thr_S) {delete [] thr_S;} thr_S = new float[b_S];
}

void Psychoacoustic_Mod2_Layer3::calc_sprdngf(){
    //Cálculo de la función de Esparcimiento, según la frecuencia de muestreo y tipo de bloque

    float tmpx_L[b_L][b_L], tmpx_S[b_S][b_S],
          x_L[b_L][b_L], x_S[b_S][b_S],
          tmpy_L[b_L][b_L], tmpy_S[b_S][b_S],
          var;

    for (int b = 0; b < b_L; b++)
        for (int bb = 0; bb < b_L; bb++) {

            tmpx_L[b][bb] = 1.05*(C_7_Long[5][bb] - C_7_Long[5][b]);

            var = powf(tmpx_L[b][bb] - 0.5, 2.0f) - 2*(tmpx_L[b][bb]-0.5);

            if (var < 0.0f)
                x_L[b][bb] = 8.0f*var;
            else
                x_L[b][bb] = 0.0f;

            tmpy_L[b][bb] = 15.811389 + 7.5*(tmpx_L[b][bb]+0.474) - 17.5*sqrtf(1.0f+powf(tmpx_L[b][bb] + 0.474,
2.0f));

            if (tmpy_L[b][bb] < -100)
                sprdngf_L[b][bb] = 0.0f;
            else
                sprdngf_L[b][bb] = powf(10.0f, (x_L[b][bb] + tmpy_L[b][bb])/10.0f);
        }

    for (int b = 0; b < b_S; b++)
        for (int bb = 0; bb < b_S; bb++) {
            tmpx_S[b][bb] = 1.05*(C_7_Short[5][bb] - C_7_Short[5][b]);
            var = powf(tmpx_S[b][bb] - 0.5, 2.0f) - 2*(tmpx_S[b][bb]-0.5);

            if (var < 0.0f)
                x_S[b][bb] = 8.0f*var;
            else
                x_S[b][bb] = 0.0f;

            tmpy_S[b][bb] = 15.811389 + 7.5*(tmpx_S[b][bb]+0.474) - 17.5*sqrtf(1.0f + powf(tmpx_S[b][bb] +
0.474, 2.0f));

            if (tmpy_S[b][bb] < -100)
                sprdngf_S[b][bb] = 0.0f;
            else
                sprdngf_S[b][bb] = powf(10.0f, (x_S[b][bb] + tmpy_S[b][bb])/10.0f);
        }
}

void Psychoacoustic_Mod2_Layer3::ingresoMuestraFifo(float sample){
    fifo[index] = sample;
    index ++;
    //Hasta esta etapa los siguientes arreglos contienen muestras temporales que se ventanean aquí
    if (index == 576){
        memset(fft_L_Data, 0.0f, sizeof(fft_L_Data));
```

```
        memset(fft_3_S_Data, 0.0f, sizeof(fft_3_S_Data));
        for (int n = 0; n < 576; n++)
            fft_L_Data[n] = fifo[n]*w_L_hann[n];//fft_L_Data[2][n] = fifo[n]*w_L_hann[n];
        for (int n = 0; n < 192; n++)
            fft_3_S_Data[0][n] = fifo[n]*w_S_hann[n]; //primeras 192 muestras
        for (int n = 192; n < 384; n++)
            fft_3_S_Data[1][n - 192] = fifo[n]*w_S_hann[n - 192]; //segundas 192 muestras
        for (int n = 384; n < 576; n++)
            fft_3_S_Data[2][n - 384] = fifo[n]*w_S_hann[n - 384]; //terceras 192 muestras
    }
    index %= 576;
    //enviar a ceros los arreglos de fft apenas se acabe el modelo psicoacústico
}

void Psychoacoustic_Mod2_Layer3::transformFFT(){

    //forward_LongFFT  .performRealOnlyForwardTransform(fft_L_Data[2]);
    forward_LongFFT  .performRealOnlyForwardTransform(fft_L_Data);
    forward_3ShortFFT.performRealOnlyForwardTransform(fft_3_S_Data[0]);
    forward_3ShortFFT.performRealOnlyForwardTransform(fft_3_S_Data[1]);
    forward_3ShortFFT.performRealOnlyForwardTransform(fft_3_S_Data[2]);

    //Se ordenan los flotantes fftData ya transformados en el dominio de la frecuencia compleja a tipo
std::complex fft_Complex_3S[k] = (Re[k],Im[k]):

    fft_Complex_L[2] = reinterpret_cast < std::complex <float> * > (fft_L_Data);

    for (int fila = 0; fila < 3; fila++)
        fft_Complex_3S[fila] = reinterpret_cast < std::complex <float> * > (fft_3_S_Data[fila]);
}

void Psychoacoustic_Mod2_Layer3::unpredictability_cw(){

    float R_L[6], φ_L[6], //Magnitud y Fase del último gránulo
          R_p_L[6], φ_p_L[6], //Magnitud y fase compuestas de 2 gránulos anteriores
          p1_L[6], p2_L[6], p3_L[6]; //arreglos auxiliares

    //Cálculo de los primeros 6 valores de cw
    for (int k = 0; k < 6; k++) {
        R_L[k] = std::abs(fft_Complex_L[2][k]);
        φ_L[k] = arg(fft_Complex_L[2][k]);
        R_p_L[k] = 2*std::abs(fft_Complex_L[1][k]) - std::abs(fft_Complex_L[0][k]);
        φ_p_L[k] = 2*arg(fft_Complex_L[1][k]) - arg(fft_Complex_L[0][k]);

        p1_L[k] = R_L[k]*cosf(φ_L[k]) - R_p_L[k]*cosf(φ_p_L[k]);
        p2_L[k] = R_L[k]*sinf(φ_L[k]) - R_p_L[k]*sinf(φ_p_L[k]);
        p3_L[k] = R_L[k] + std::abs(R_p_L[k]);

        //primeros 6 valores de cw[k]
        if (p3_L[k] != 0.0f){ //si el numerador es distinto de cero:
            cw[k] = sqrtf(powf(p1_L[k],2) + powf(p2_L[k],2))/p3_L[k];
        }else{
            cw[k] = 0.0f;
        }
    }

    float R_S[50], φ_S2[50], //Magnitud y Fase del trozo 2 de 192 muestras
          R_p_S[50], φ_p_S[50], //Magnitud y fase compuestas de los trozos 1 y 3 de 192 muestras
          p1_S[50], p2_S[50], p3_S[50]; //arreglos auxiliares

    //Cálculo de los siguientes 200 valores de cw
    int k;
    for (int kk = 6; kk < 206; kk += 4) {

        k = (kk+1)/4; //kk = 1,2,3,...,50

        R_S[k] = std::abs(fft_Complex_3S[1][k]);
        φ_S2[k] = arg(fft_Complex_3S[1][k]);
        R_p_S[k] = 2.0f*std::abs(fft_Complex_3S[0][k]) - std::abs(fft_Complex_3S[2][k]);
        φ_p_S[k] = 2.0f*arg(fft_Complex_3S[0][k]) - arg(fft_Complex_3S[2][k]);

        p1_S[k] = R_S[k]*cosf(φ_S2[k]) - R_p_S[k]*cosf(φ_p_S[k]);
        p2_S[k] = R_S[k]*sinf(φ_S2[k]) - R_p_S[k]*sinf(φ_p_S[k]);
        p3_S[k] = R_S[k] + std::abs(R_p_S[k]);

        // se añaden 200 valores más a cw
        if (p3_S[k] != 0.0f) //si el numerador es distinto de cero:
            cw[kk] = sqrtf(powf(p1_S[k], 2.0f) + powf(p2_S[k], 2.0f))/p3_S[k];
        else
            cw[kk] = 0.0f;

        for (int i = 1; i < 4; i++) //se copia 3 veces el valor actual en la posición kk
            cw[kk + i] = cw[kk];
    }
```

```
    //cw[k>=206] = 0.4
    for (int k = 206; k < cw_size; k++)
        cw[k] = 0.4f;
}


void Psychoacoustic_Mod2_Layer3::calc_umbra_parte1(){

    //Cálculo de la energía en las particiones b de Bark

    float sum_eb, sum_cb;
    int k = 0;
    for (int b = 0; b < b_L; b++) {
        sum_eb = 0; sum_cb = 0;
        for (int i = 0; i < int(C_7_Long[1][b]); i++){ //se reunen las energías en cada partición b
            sum_eb += powf(std::abs(fft_Complex_L[2][k]), 2.0f);
            sum_cb += powf(std::abs(fft_Complex_L[2][k]), 2.0f)*cw[k];
            k++;
        }
        eb_L[b] = sum_eb; //✓
        cb[b] = sum_cb; //✓
    }

    //Transformación de las energías al dominio de Bark a través del mapeo de eb_L[b] y cb[b] con la función de
esparcimiento
    float psum_ecb, psum_ctb;
    for (int b = 0; b < b_L; b++) {
        psum_ecb = 0; psum_ctb = 0;
        for (int bb = 0; bb < b_L; bb++) {
            psum_ecb += eb_L[bb]*sprdngf_L[b][bb];
            psum_ctb += cb[bb]*sprdngf_L[b][bb];
            //std::cout << "sprdngf_L[" << b << "][" << bb << "] = " << sprdngf_L[b][bb] << std::endl;
        }
        ecb_L[b] = psum_ecb; //✓
        ctb[b] = psum_ctb; //✓
        //std::cout << "ecb_L[" << b << "] = " << ecb_L[b] << ", ctb[" << b << "] = " << ctb[b] << std::endl;
    }

    //Cálculo de los índices de tonalidad tbb[] y umbrales de enmascaramiento thr[]
    float cbb, aux;
    for (int b = 0; b < b_L; b++){
        if(ecb_L[b] != 0.0f){
            cbb = ctb[b]/ecb_L[b];
            if(cbb < 0.01)
                cbb = 0.01;
            cbb = logf(cbb);
        }else
            cbb = 0.0f;

        // tbb[b] = max(1, min(-0.299 - 0.430*cbb,0))
        tbb[b] = -0.299 - 0.430*cbb; //0 < tbb < 1
        if (tbb[b] < 0) tbb[b] = 0.0f;
        if (tbb[b] > 1) tbb[b] = 1.0f;

        // SNR_L[b] = max(C_7_Long[2][b], 29.0f*tbb[b] + 6.0f*(1.0f - tbb[b]))
        aux = 29.0f*tbb[b] + 6.0f*(1.0f - tbb[b]);
        if (aux > C_7_Long[2][b])
            SNR_L[b] = aux;
        else
            SNR_L[b] = C_7_Long[2][b]; //minval[b]

        nbb_L[2][b] = ecb_L[b] * C_7_Long[4][b] * powf(10.0f, -SNR_L[b]/10.0f);

        //temp_1 = min(nbb_L[2][b], min(2.0f*nbb_L[1][b], 16.0f*nbb_L[0][b]));
        //thr_L[b] = max(qthr_L[b], temp_1);
        thr_L[b] = 16.0f*nbb_L[0][b];

        if (2.0f*nbb_L[1][b] < 16.0f*nbb_L[0][b])
            thr_L[b] = 2.0f*nbb_L[1][b];
        if (thr_L[b] > nbb_L[2][b])
            thr_L[b] = nbb_L[2][b];
        if (thr_L[b] < C_7_Long[3][b])
            thr_L[b] = C_7_Long[3][b]; //qthr_L[b]

    }

    //Cálculo de la Entropía Perceptual PE
    PE = 0.0f;
    for (int b = 0; b < b_L; b++) //no se calcula para bloques Short
        PE += C_7_Long[1][b]*logf(thr_L[b]/(eb_L[b] + 1.0f));
    PE *= -1;

    //if (PE > 1800) //ataque detectado: short block

    if (((PE > 1800) && (past_block_type == 0)) || ((PE > 1800) && (past_block_type == 10))){ //Start
```

```
        block_type = 1;
    }
    else if ((past_block_type == 1) || ((PE > 1800) && (past_block_type == 2)) || ((PE > 1800) &&
(past_block_type == 3))){ //short
        block_type = 2;
    }
    else if ((PE <= 1800) && (past_block_type == 2)){ //Stop
        block_type = 3;
    }
    else if ((PE <= 1800) || ((PE <= 1800) && (past_block_type == 3))){//long
        block_type = 0;
    }
    //past_block_type = block_type;
    //std::cout << "block_type = " << block_type << ", past_block_type = " << past_block_type << ", PE = " <<
PE << std::endl;
}

void Psychoacoustic_Mod2_Layer3::calc_umbra_ShortBlock(){
    //Cálculo de la energía en las particiones b de Bark
    float sum_eb;
    int k = 0;
    for (int b = 0; b < b_S; b++) {
        sum_eb = 0;
        for (int i = 0; i < int(C_7_Short[1][b]); i++){
            sum_eb += powf(std::abs(fft_Complex_3S[s_block][k]), 2);
            k++;
        }
        eb_S[b] = sum_eb;
    }

    //Transformación de la energía eb_S[][] al dominio de Bark a través del mapeo de eb_S[b] con la función de
esparcimiento
    float psum_ecb;
    for (int b = 0; b < b_S; b++) {
        psum_ecb = 0;
        for (int bb = 0; bb < b_S; bb++)
            psum_ecb += eb_S[bb]*sprdngf_S[b][bb];
        ecb_S[b] = psum_ecb;
    }

    for (int b = 0; b < b_S; b++){
        //nbb_S[b] = ecb_S[b] * norm[b] * powf(10.0f, -SMR_S[b]/10.0f)
        nbb_S[b] = ecb_S[b] * C_7_Short[3][b] * powf(10.0f, -C_7_Short[3][b]/10.0f);

        //thr_S[b] = max(nbb_S[b], qthr_S[b])
        thr_S[b] = nbb_S[b];
        if (thr_S[b] < C_7_Short[3][b])
            thr_S[b] = C_7_Short[2][b];
    }
}

void Psychoacoustic_Mod2_Layer3::calc_umbra_parte2(){

    if (block_type == 2){//para bloques Short

        float *w1 = C_8_Short[4], *w2 = C_8_Short[5],
        *bu = C_8_Short[2], *bo = C_8_Short[3], *bw = C_8_Short[1],
        part_sum1, part_sum2;

        for(int sb = 0; sb < 12; sb++){
            part_sum1 = 0; part_sum2 = 0;
            for (int b = int(bu[sb]) + 1; b < (int(bo[sb]) - 1); b++){
                part_sum1 += eb_S[b];
                part_sum2 += thr_S[b];
            }
            en_S[s_block][sb] = w1[sb]*eb_S[int(bu[sb])] + part_sum1 + w2[sb]*eb_S[int(bo[sb])];
            thm_S[s_block][sb] = w1[sb]*thr_S[int(bu[sb])] + part_sum2 + w2[sb]*thr_S[int(bo[sb])];
            SMR_S[s_block][sb] = thm_S[s_block][sb]/en_S[s_block][sb];
            //xmin_S[1][s_block][sb] = (SMR_S[s_block][sb]*en_S[s_block][sb])/bw[sb];

            xmin_S[1][s_block][sb] = float(SMR_S[s_block][sb]/bw[sb]);//se multiplica por la energía en
Quantizaion_Mod.cpp
            //std::cout << "SMR_S[" << s_block << "][" << sb << "] = " << SMR_S[s_block][sb] << std::endl;
        }
        s_block ++; s_block %= 3;
    }else{//para bloques distintos al short

        float *w1 = C_8_Long[4], *w2 = C_8_Long[5],
        *bu = C_8_Long[2], *bo = C_8_Long[3], *bw = C_8_Long[1],
        part_sum1, part_sum2;

        for(int sb = 0; sb < 21; sb++){
            part_sum1 = 0; part_sum2 = 0;
            for (int b = int(bu[sb]) + 1; b < (int(bo[sb]) - 1); b++){
                part_sum1 += eb_L[b];
```

```
                part_sum2 += thr_L[b];
            }
            en_L[sb] = w1[sb]*eb_L[int(bu[sb])] + part_sum1 + w2[sb]*eb_L[int(bo[sb])];
            thm_L[sb] = w1[sb]*thr_L[int(bu[sb])] + part_sum2 + w2[sb]*thr_L[int(bo[sb])];
            SMR_L[sb] = thm_L[sb]/en_L[sb];

            //xmin_L[1][sb] = (SMR_L[sb]*en_L[sb])/bw[sb];
            xmin_L[1][sb] = float(SMR_L[sb]/bw[sb]); //se multiplica por la energía en Quantizaion_Mod.cpp
            //std::cout << "SMR_L[" << sb << "] = " << SMR_L[sb] << std::endl;
        }
    }
}

void Psychoacoustic_Mod2_Layer3::run_Model(){

    transformFFT();
    unpredictability_cw();
    calc_umbra_parte1();

    if (block_type != 2) //PE <= 1800, LONG
        calc_umbra_parte2();
    else{
        std::cout << "Cálculos Psico Short" << std::endl;
        for (int short_block = 0; short_block < 3; short_block++){
            calc_umbra_ShortBlock();
            calc_umbra_parte2();
        }
    }

    //Se copian los datos de FFT de los gránulos actuales a los anteriores
    for (int k = 0; k < fft_L_Size; k++){
        fft_Complex_L[0][k] = fft_Complex_L[1][k];
        fft_Complex_L[1][k] = fft_Complex_L[2][k];
    }

    for (int b = 0; b < b_L; b++){
        nbb_L[0][b] = nbb_L[1][b];
        nbb_L[1][b] = nbb_L[2][b];
    }
}

//Usar este método despues de utilizar los valores RECIEN calculados de en_L y SMR_L
void Psychoacoustic_Mod2_Layer3::copyOutputs(){

    //Se copian las distorisones permitidas actuales del gránulo actual en el la fila del gránulo pasado
    if (block_type == 2)
        for (int i = 0; i < 3; i++)
            for (int sb = 0; sb < 12; sb++)
                xmin_S[0][i][sb] = xmin_S[1][i][sb];
    else
        for (int sb = 0; sb < 21; sb++)
            xmin_L[0][sb] = xmin_L[1][sb];

}
/* ============================================================================================*/
//LAYER 2
/* ============================================================================================*/

void Psychoacoustic_Mod2_Layer2::setMatrices(){

    for (int n = 0; n < 576; n++)
        w_hann[n] = 0.5 - 0.5*cos((2*M_PI/576)*(n - 0.5));

    memset(fifo, 0, sizeof(fifo)); //se llenan de ceros las matrices
    memset(fft_Data, 0, sizeof(fft_Data));

    fft_Complex = new std::complex<float> *[3];
    for (int fila = 0; fila < 3; fila++)
        fft_Complex[fila] = new std::complex <float>[fft_Size];
    for(int fila = 0; fila < 3; fila ++)
        std::fill(fft_Complex[fila], fft_Complex[fila] + fft_Size, std::complex<float>{0,0});

    SMR = new float *[2];
    for (int fila = 0; fila < 2; fila++)
        SMR[fila] = new float[32];

    for(int fila = 0; fila < 2; fila ++)
        std::fill(SMR[fila], SMR[fila] + 32, 0.0);
}

template <size_t rows, size_t cols>
void Psychoacoustic_Mod2_Layer2::copyMat(float C_Table[rows][cols], int opt_mat){

    aux_mat = new float *[rows];
    for (int i = 0; i < rows; i++)
```

```
        aux_mat[i] = new float[cols];

    for (int fila = 0; fila < rows; fila++)
        for (int col = 0; col < cols; col++)
            aux_mat[fila][col] = C_Table[fila][col];

    switch (opt_mat) {
        case 0: D_4 = aux_mat; absthr_imax = int(D_4[1][cols]); break;
        case 1: D_3 = aux_mat; b_max = int(cols); break;
    }
}

void Psychoacoustic_Mod2_Layer2::fs_select(double fs){
    std::cout << "psyco fs = " << fs << std::endl;
    switch (int(fs)) {
        case 48000:
            copyMat<3>(D_4_48k, 0); copyMat<6>(D_3_48k, 1); break;
        case 44100:
            copyMat<3>(D_4_44k, 0); copyMat<6>(D_3_44k, 1); break;
        case 32000:
            copyMat<3>(D_4_32k, 0); copyMat<6>(D_3_32k, 1); break;
        default:
            copyMat<3>(D_4_44k, 0); copyMat<6>(D_3_44k, 1); break;
    }
    std::cout << "b_max = " << b_max << std::endl;
    sprdngf = new float *[b_max];
    for (int fila = 0; fila < b_max; fila++) sprdngf[fila] = new float [b_max];
    calc_sprdngf();

    //std::cout << "D_3[2][b_max] = " << D_3[2][b_max-1] << std::endl;
    cw_size = int(D_3[2][b_max-1]-1);

    if (cw) {delete [] cw;} cw = new float[cw_size];
    if (eb) {delete [] eb;} eb = new float[b_max];
    if (cb) {delete [] cb;} cb = new float[b_max];
    if (ecb) {delete [] ecb;} ecb = new float[b_max];
    if (ctb) {delete [] ctb;} ctb = new float[b_max];
    if (cbb) {delete [] cbb;} cbb = new float[b_max];
    if (tbb) {delete [] tbb;} tbb = new float[b_max];
    if (rnorm) {delete [] rnorm;} rnorm = new float[b_max];
    if (enb) {delete [] enb;} enb = new float[b_max];
    if (SNR) {delete [] SNR;} SNR = new float[b_max];
    if (nbb) {delete [] nbb;} nbb = new float[b_max];
    if (nbw) {delete [] nbw;} nbw = new float[512];
    if (thr) {delete [] thr;} thr = new float[512];
}

void Psychoacoustic_Mod2_Layer2::calc_sprdngf(){

    //Cálculo de la función de Esparcimiento, según la frecuencia de muestreo y tipo de bloque

    float tmpx[b_max][b_max], x[b_max][b_max], tmpy[b_max][b_max], var;

    for (int b = 0; b < b_max; b++)
        for (int bb = 0; bb < b_max; bb++) {

            tmpx[b][bb] = 1.05*(D_3[3][bb] - D_3[3][b]);

            var = powf(tmpx[b][bb] - 0.5, 2.0f) - 2*(tmpx[b][bb]-0.5);

            if (var < 0.0f)
                x[b][bb] = 8.0f*var;
            else
                x[b][bb] = 0.0f;

            tmpy[b][bb] = 15.811389 + 7.5*(tmpx[b][bb]+0.474) - 17.5*sqrtf(1.0f + powf(tmpx[b][bb] + 0.474,
2.0f));

            if (tmpy[b][bb] < -100)
                sprdngf[b][bb] = 0.0f;
            else
                sprdngf[b][bb] = powf(10.0f, (x[b][bb] + tmpy[b][bb])/10.0f);
        }
}

void Psychoacoustic_Mod2_Layer2::ingresoMuestraFifo(float sample){
    fifo[index] = sample;
    index ++;
    //Hasta esta etapa los siguientes arreglos contienen muestras temporales que se ventanean aquí
    if (index == 576){
        memset(fft_Data, 0.0f, sizeof(fft_Data));
        for (int n = 0; n < 576; n++)
            fft_Data[n] = fifo[n]*w_hann[n];
    }
    index %= 576;
```

```
}

void Psychoacoustic_Mod2_Layer2::transformFFT(){

    forward_FFT.performRealOnlyForwardTransform(fft_Data);
    fft_Complex[2] = reinterpret_cast < std::complex <float> * > (fft_Data);

}

void Psychoacoustic_Mod2_Layer2::unpredictability_cw(){
    float R[512], φ[512], //Magnitud y Fase del último gránulo
    R_p[512], φ_p[512], //Magnitud y fase compuestas de 2 gránulos anteriores
    p1[512], p2[512], p3[512]; //arreglos auxiliares

    //Cálculo de los primeros 6 valores de cw
    for (int k = 0; k < cw_size; k++) {
        R[k] = std::abs(fft_Complex[2][k]);
        φ[k] = arg(fft_Complex[2][k]);
        R_p[k] = 2*std::abs(fft_Complex[1][k]) - std::abs(fft_Complex[0][k]);
        φ_p[k] = 2*arg(fft_Complex[1][k]) - arg(fft_Complex[0][k]);

        p1[k] = R[k]*cosf(φ[k]) - R_p[k]*cosf(φ_p[k]);
        p2[k] = R[k]*sinf(φ[k]) - R_p[k]*sinf(φ_p[k]);
        p3[k] = R[k] + std::abs(R_p[k]);

        //primeros 6 valores de cw[k]
        if (p3[k] != 0.0f) //si el numerador es distinto de cero:
            cw[k] = sqrtf(powf(p1[k],2) + powf(p2[k],2))/p3[k];
        else
            cw[k] = 0.0f;
        //std::cout << "cw[" << k << "] = " << cw[k] << std::endl;
    }
}

void Psychoacoustic_Mod2_Layer2::calc_umbral(){
    float sum_eb, sum_cb;

    for (int b = 0; b < b_max; b++) {
        sum_eb = 0.0; sum_cb = 0.0;
        for (int k = int(D_3[1][b]-1); k < int(D_3[2][b]-1); k++){ //se reunen las energías en cada partición b
            sum_eb += powf(std::abs(fft_Complex[2][k]), 2.0f);
            sum_cb += powf(std::abs(fft_Complex[2][k]), 2.0f)*cw[k];
        }
        eb[b] = sum_eb; //✓
        cb[b] = sum_cb; //✓
        //std::cout << "eb [" << b << "] = " << eb[b] << ", cb[" << b << "] = " << cb[b] << std::endl;
    }

    float psum_ecb, psum_ctb, rnorm_sum, aux;

    for (int b = 0; b < b_max; b++) {
        psum_ecb = 0; psum_ctb = 0; rnorm_sum = 0;
        for (int bb = 0; bb < b_max; bb++) {
            psum_ecb += eb[bb]*sprdngf[b][bb];
            psum_ctb += cb[bb]*sprdngf[b][bb];
            rnorm_sum += sprdngf[b][bb];
            //std::cout << "sprdngf[" << b << "][" << bb << "] = " << sprdngf[b][bb] << std::endl;
        }

        ecb[b] = psum_ecb; //✓
        ctb[b] = psum_ctb; //✓
        cbb[b] = float(ctb[b]/ecb[b]);

        //std::cout << "ecb [" << b << "] = " << ecb[b] << ", ctb[" << b << "] = " << ctb[b] << std::endl;

        rnorm[b] = float(1.0f/rnorm_sum);
        enb[b] = ecb[b]*rnorm[b];

        tbb[b] = -0.299 - 0.43*logf(cbb[b]); //0<tbb<1
        if (tbb[b] < 0) tbb[b] = 0.0f;
        if (tbb[b] > 1) tbb[b] = 1.0f;

        //SNR[b] = max(minval[b], TMN[b]*tbb[b] + 5.5(dB)*(1.0f - tbb[b]))
        aux = float(D_3[5][b])*tbb[b] + 5.5f*(1.0f - tbb[b]);
        if (aux > D_3[4][b])
            SNR[b] = aux;
        else
            SNR[b] = D_3[5][b]; //minval[b]

        //std::cout << "SNR [" << b << "] = " << SNR[b] << ", tbb[" << b << "] = " << tbb[b] << std::endl;

        nbb[b] = enb[b] * powf(10.0f, -SNR[b]/10.0f);
        //std::cout << "nbb [" << b << "] = " << nbb[b] << std::endl;

        float k_l = int(D_3[1][b]-1), k_h = int(D_3[2][b]);
```

```
        for (int k = k_l; k < k_h; k++){

            nbw[k] = nbb[b]/(float(D_3[2][b] - D_3[1][b] + 1.0f));

            // thr[k] = max(nbw[k],absthr[k])
            if (k <= absthr_imax){
                if (nbw[k] > D_4[2][k])
                    thr[k] = nbw[k];
                else
                    thr[k] = powf(10.0f, -D_4[2][k]/10.0f);;
            }else{
                thr[k] = nbw[k];
            }
            //std::cout << "thr[" << k << "] = " << thr[k] << std::endl;

        }
    }

    float epartSum, npartSum;

    for (int n = 0; n < 32; n++){
        epartSum = 0; npartSum = 0;
        for (int k = (D_5[1][n] - 1); k < (D_5[1][n+1] - 1); k++){
            epartSum += powf(std::abs(fft_Complex[2][k]), 2.0f);
            npartSum += thr[k];
        }
        epart[n] = epartSum;
        if (D_5[2][n]){
            npart[n] = npartSum;
        }else{
            npart[n] = *std::min_element(thr + 0, thr + 512)*(D_5[1][n+1]-D_5[1][n]+1);
            //std::cout << "npart[" << n << "] = " << npart[n] << ", thr_min = " << *std::min_element(thr + 0,
thr + 512) << std::endl;
        }
        SMR[1][n] = 10*log10f(epart[n]/npart[n]);
        //std::cout << D_5[2][n] << ": SMR [" << n << "] = " << SMR[1][n] << ", npart[" << n << "] = " <<
npart[n] << std::endl;
    }

    for (int k = 0; k < fft_Size; k++){
        fft_Complex[0][k] = fft_Complex[1][k];
        fft_Complex[1][k] = fft_Complex[2][k];
    }
}

void Psychoacoustic_Mod2_Layer2::run_Model(){
    transformFFT();
    unpredictability_cw();
    calc_umbral();
}

/* //en cada fila va la informaciòn de cada gránulo
 fft_Complex_S1 = new std::complex<float> *[3];
 for (int fila = 0; fila < 3; fila++) fft_Complex_S1[fila] = new std::complex<float>[fft_S_Size];

 fft_Complex_S2 = new std::complex<float> *[3];
 for (int fila = 0; fila < 3; fila++) fft_Complex_S2[fila] = new std::complex<float>[fft_S_Size];

 fft_Complex_S3 = new std::complex<float> *[3];
 for (int fila = 0; fila < 3; fila++) fft_Complex_S3[fila] = new std::complex<float>[fft_S_Size];

 */

/*
 float Psychoacoustic_Mod2_Layer3::min(float array[], int size_arr){
 float minim = array[0];
 for (int i = 0; i < size_arr; i++)
 if (minim > array[i])
 minim = array[i];
 return minim;
 }
 float Psychoacoustic_Mod2_Layer3::max(float array[], int size_arr){
 float maxi = array[0];
 for (int i = 0; i < size_arr; i++)
 if (maxi < array[i])
 maxi = array[i];
 return maxi;
 }

 void Psychoacoustic_Mod2_Layer3::copyMat(float fila_mat[], int index_Row,int size_Row, bool opt_mat){

 if (opt_mat){//Caso true-> Matriz Long

 if (index_Row == 0) {//se crea la matriz, y se llena la primera fila
```

```
 b_L = size_Row;
 C_7_Long = new float *[6];
 for (int i = 0; i < 6; i++) C_7_Long[i] = new float[size_Row];
 for (int j = 0; j < size_Row; j++) C_7_Long[0][j] = fila_mat[j];

 }else{//se copia el resto de filas en la matriz
 for (int j = 0; j < size_Row; j++) C_7_Long[index_Row][j] = fila_mat[j];
 }

 }else{//Caso true-> Matriz Short
 if (index_Row == 0) {
 b_S = size_Row;
 C_7_Short = new float *[6];
 for (int i = 0; i < 6; i++) C_7_Short[i] = new float[size_Row];
 for (int j = 0; j < size_Row; j++) C_7_Short[0][j] = fila_mat[j];
 }else{
 for (int j = 0; j < size_Row; j++) C_7_Short[index_Row][j] = fila_mat[j];
 }
 }
 }
 */
```

Quantization_Mod.h

```
#pragma once
#include <math.h>
#include <iostream>
#include <algorithm>

extern int B_8_48k_Long[4][21], B_8_44k_Long[4][21], B_8_32k_Long[4][21],
B_8_48k_Short[4][12], B_8_44k_Short[4][12], B_8_32k_Short[4][12],
pretab[21];

class Quantization_Mod_Layer3{
public:
    Quantization_Mod_Layer3();
    ~Quantization_Mod_Layer3();

    int nint(float var);
    void fs_select(double fs);

    template <size_t rows, size_t cols>
    void copyMat(int aux_mat[rows][cols], int opt_mat);

    void set_blockType(int blockType, int pastBlockType);
    void resetVariables();
    void inputMats(float **MDCT_L, float ***MDCT_S, float **PSYxmin_L, float ***PSYxmin_S);
    void outputMats();

    void iterationLoop();

    void outerLoop();
    void innerLoop();
    void quantize(int bl);
    void ix_max(int bl);

    void preemphasis();
    void amp_scalefactor_bands();
    void calc_dist();

    float scalefac_L[2][21], scalefac_S[2][3][12];
    int ix_L[576], ix_S[3][192];

private:

    // Arreglos punteros, para paso por referencia de las matrices: MDCT directa (xr), y umbrales mínimos
(xmin)
    float **xr_L, ***xr_S, **xmin_L, ***xmin_S,
          xr_array[576], xr_S_array[3][192],
    //sfm: medida de la planitud espectral,
    //ifqstep: factor usado para amplificar las bandas que exceden la distorsión permitida
    //xfsf: distorsión hallada en cada scalefactor band
          sfm, ifqstep, xfsf_L[21], xfsf_S[3][12];

    //scfsi: 0: los scalefactors de cada gránulo se asignan a sus gránulos respectivos, 1: los scalefactors del
gránulo 0 se utilizan para el gránulo 1.
    //endLoop: variable de control, permite verificar si se han amplificado todos las líneas de frecuencia de
la MDCT en sus respectivas scalefactor bands
    bool scfsi[4], endLoop;

    int begIndex[4] = {0, 6, 11, 16}, endIndex[4] = {6, 11, 16, 21},
        copyScalefactor, preventScalefactor, mixed_block_flag;

    int en_tot[2], en_L[2][21], xm[2][21], quantanf, qquant[4], max_ix[4];
```

```
    //Matrices locales de las tablas B.8 según la frecuencia de muestreo seleccionada
    int **B_8_Long, **B_8_Short, //4 filas, 21 o 12 columnas
        **aux_mat,
        block_type, past_block_type, sb_len,
        *lbl, *bw, *sb_beg, *sb_end, preflag[2], scalefac_scale[2],
        gr, iteracion;

};
/*============================================================================*/
extern int B_8_II_32k[2][4], B_8_II_44k[2][4], B_8_II_48k[2][4], pretab_II[32];

class Quantization_Mod_Layer2{
public:
    Quantization_Mod_Layer2();
    ~Quantization_Mod_Layer2();

    int nint(float var);
    void fs_select(double fs);

    template <size_t rows, size_t cols>
    void copyMat(int aux_mat[rows][cols], int opt_mat);

    void set_block();
    void resetVariables();
    void inputMats(float **PQMF_coef, float **PSYxmin);

    void iterationLoop();

    void outerLoop();
    void innerLoop();
    void quantize(int bl);
    void ix_max(int bl);

    void preemphasis();
    void amp_scalefactor_bands();
    void calc_dist();

    float scalefac[2][21];
    int ix[576];

private:

    // Arreglos punteros, para paso por referencia de las matrices: MDCT directa (xr), y umbrales mínimos
(xmin)
    float **xmin,
    xr_array[576],
    //sfm: medida de la planitud espectral,
    //ifqstep: factor usado para amplificar las bandas que exceden la distorsión permitida
    //xfsf: distorsión hallada en cada scalefactor band
    sfm, ifqstep, xfsf[21];

    //scfsi: 0: los scalefactors de cada gránulo se asignan a sus gránulos respectivos, 1: los scalefactors del
gránulo 0 se utilizan para el gránulo 1.
    //endLoop: variable de control, permite verificar si se han amplificado todos las líneas de frecuencia de
la MDCT en sus respectivas scalefactor bands
    bool scfsi[4], endLoop;

    int begIndex[4] = {0, 6, 11, 16}, endIndex[4] = {6, 11, 16, 21},
    copyScalefactor, preventScalefactor, mixed_block_flag;

    int en_tot[2], en_L[2][21], xm[2][21], quantanf, qquant[4], max_ix[4];

    //Matrices locales de las tablas B.8 según la frecuencia de muestreo seleccionada
    int **B_8_Long, //4 filas, 21 columnas
    **B_8_II, **aux_mat,
    sb_len,
    *lbl, *bw, *sb_beg, *sb_end, preflag[2], scalefac_scale[2],
    gr, iteracion;

    float re_W[32][18], **s, xr[32][18], w_hann[36];
};
```

Quantization_Mod.cpp

```
#include "Quantization_Mod.h"

Quantization_Mod_Layer3::Quantization_Mod_Layer3(){}

Quantization_Mod_Layer3::~Quantization_Mod_Layer3(){}

int Quantization_Mod_Layer3::nint(float in){ //1995 MPEG/audio software simulation group
    int temp;

    if(in < 0) temp = (int)(in - 0.5);
    else temp = (int)(in + 0.5);
```

```
    return (temp);
}


template <size_t rows, size_t cols>
void Quantization_Mod_Layer3::copyMat(int C_Table[rows][cols], int opt_mat){

    aux_mat = new int *[rows];
    for (int i = 0; i < rows; i++)
        aux_mat[i] = new int[cols];

    for (int fila = 0; fila < rows; fila++)
        for (int col = 0; col < cols; col++)
            aux_mat[fila][col] = C_Table[fila][col];

    switch (opt_mat) {
        case 0: B_8_Long = aux_mat; break;
        case 1: B_8_Short = aux_mat; break;
    }
}

void Quantization_Mod_Layer3::fs_select(double fs){
    //llamar método fs_select(sample_rate) desde el getnextaudioblock
    //dependiendo de la frecuencia de muestreo se generan unas nuevas matrices dinámicas para las tablas C.7
Long y Short
    switch (int(fs)) {
        case 48000:
            copyMat<4>(B_8_48k_Long, 0); copyMat<4>(B_8_48k_Short, 1);
            break;
        case 44100:
            copyMat<4>(B_8_44k_Long, 0); copyMat<4>(B_8_44k_Short, 1);
            break;
        case 32000:
            copyMat<4>(B_8_32k_Long, 0); copyMat<4>(B_8_32k_Short, 1);
            break;
        default:
            copyMat<4>(B_8_44k_Long, 0); copyMat<4>(B_8_44k_Short, 1);
            break;
    }
    // copy to: B_8_Short , B_8_Long

}

void Quantization_Mod_Layer3::set_blockType(int blockType, int pastBlockType){
    block_type = blockType; past_block_type = pastBlockType;

    if (block_type == 2){
        sb_len = 12;
        lbl = B_8_Short[2]; bw = B_8_Short[1];
        sb_beg = B_8_Short[2]; sb_end = B_8_Short[3];
    }else{
        sb_len = 21;
        lbl = B_8_Long[2]; bw = B_8_Long[1];
        sb_beg = B_8_Long[2]; sb_end = B_8_Long[3];
    }
}

void Quantization_Mod_Layer3::inputMats(float **MDCT_L, float ***MDCT_S, float **PSYxmin_L, float
***PSYxmin_S){

    if (block_type == 2){
        xr_S = MDCT_S;
        for (int bloque = 0; bloque < 3; bloque++)
            for (int k = 0; k < 32; k++)
                for (int ξ = 0; ξ < 6; ξ++)
                    xr_S_array[bloque][k*6 + ξ] = MDCT_S[bloque][k][ξ];

        xmin_S = PSYxmin_S;
        float sum;
        //Terminación del cálculo de xmin
        for (int b = 0; b < 3; b++)
            for (int sb = 0; sb < 12; sb++){
                sum = 0;
                for (int l = sb_beg[sb]; l < (sb_end[sb] + 1); l++)
                    sum += xr_S_array[b][l]*xr_S_array[b][l];
                xmin_S[0][b][sb] *= sum;
                xmin_S[1][b][sb] *= sum;
            }

    }else{
        xr_L = MDCT_L;

        for (int k = 0; k < 32; k++)
            for (int ξ = 0; ξ < 18; ξ++)
```

```
                xr_array[k*18 + ξ] = MDCT_L[k][ξ];

        xmin_L = PSYxmin_L;
        float sum;
        //Terminación del cálculo de xmin
        for (int sb = 0; sb < 21; sb++){
            sum = 0;
            for (int l = sb_beg[sb]; l < (sb_end[sb] + 1); l++)
                sum += xr_array[l]*xr_array[l];
            xmin_L[0][sb] *= sum;
            xmin_L[1][sb] *= sum;
        }
    }

}

void Quantization_Mod_Layer3::outputMats(){

    if (block_type == 2){

        /*for (int bloque = 0; bloque < 3; bloque++)
            for (int ξ = (sb_end[11] + 1); ξ < 192; ξ++)
                xr_S_array[bloque][ξ] = 0; //se envia a cero las bandas por fuera de los límites de las tablas
B.8
        */
        for (int bloque = 0; bloque < 3; bloque++)
            for (int k = 0; k < 32; k++)
                for (int ξ = 0; ξ < 6; ξ++)
                    xr_S[bloque][k][ξ] = xr_S_array[bloque][k*6 + ξ];

    }else{
        //for (int ξ = (sb_end[20] + 1); ξ < 576; ξ++) xr_array[ξ] = 0;

        for (int k = 0; k < 32; k++)
            for (int ξ = 0; ξ < 18; ξ++)
                xr_L[k][ξ] = xr_array[k*18 + ξ];
    }

}

void Quantization_Mod_Layer3::resetVariables(){
    endLoop = false;
    memset(max_ix, 0, sizeof(max_ix));
    memset(ix_S, 0, sizeof(ix_S));
    memset(ix_L, 0, sizeof(ix_L));
    memset(scalefac_L, 0, sizeof(scalefac_L));
    memset(scalefac_S, 0, sizeof(scalefac_S));
    memset(scalefac_scale, 0, sizeof(scalefac_scale));
    memset(preflag, 0, sizeof(preflag));
    iteracion = 0;
    memset(qquant, 0, sizeof(preflag));
    quantanf = 0;

    float sum1 = 0, sum2 = 0;
    for (int k = 0; k < 32; k++)
        for (int ξ = 0; ξ < 18; ξ++){
            sum1 += powf(logf(xr_L[k][ξ]), 2.0f);
            sum2 += powf(xr_L[k][ξ], 2.0f);
        }
    sum1 = float(1.0/576.0)*sum1;
    sum2 = float(1.0/576.0)*sum2;
    sfm = powf(expf(1.0), sum1)/sum2;

    quantanf = nint(8.0*logf(sfm)); //cálculo válido solo 1 vez por cada 2 gránulos

    if (quantanf < -100)
        quantanf = -100;

    quantanf -= 70;

}

void Quantization_Mod_Layer3::iterationLoop(){
    resetVariables();
    //-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=
    //determinación de scfsi (pg. 100 - 101)
    float sum1 = 0, sum2 = 0;
    for (int k = 0; k < 32; k++)
        for (int ξ = 0; ξ < 18; ξ++)
            sum1 += powf(fabs(xr_L[k][ξ]), 2.0f);

    en_tot[1] = int(log2f(sum1));

    for (int sb = 0; sb < sb_len; sb++){
        sum2 = 0;
```

```
        for (int ξ = lbl[sb]; ξ < (lbl[sb] + bw[sb] - 1); ξ++)
            sum2 += powf(fabs(xr_array[ξ]), 2.0f);
        en_L[1][sb] = int(log2f(sum2));
        //xm[0][sb] = int(log2f(xmin_L[0][sb]));
        xm[1][sb] = int(log2f(xmin_L[1][sb]));
    }

    //-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=
    // Cuatro criterios: para el uso de scfsi pg. 101

    bool cond1 = false, cond2 = false, cond3 = false, cond4 = false;

    //a) no todos los valores espectrales son cero.
    for (int ξ = 0; ξ < 576; ξ++)
        if (xr_array[ξ] != 0){
            cond1 = true;
            break;
        }

    //b) ninguno de los dos gránulos presentes es tipo SHORT
    if ((block_type != 2) && (past_block_type != 2))
        cond2 = true;

    //c) |en_tot[0] - en_tot[1]| < en_tot_krit = 10
    if (abs(en_tot[0] - en_tot[1]) < 10)
        cond3 = true;

    en_tot[0] = en_tot[1];

    //d) ∑|en_L[0][sb] - en_L[1][sb]| < en_dif_krit = 100
    float crit = 0;
    for (int sb = 0; sb < 21; sb++)
        crit += std::abs(en_L[0][sb] - en_L[1][sb]);
    if (crit < 100)
        cond4 = true;

    //Revisión del cumplimiento de los 4 criterios, si alguno de estos es falso se niegan los 4 scfsi[i]
    if ((cond1 == false) || (cond2 == false) || (cond3 == false) || (cond4 == false))
        for (int scfsi_band = 0; scfsi_band < 4; scfsi_band++)
            scfsi[scfsi_band] = false;

    //-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.
    // Dos criterios para cada scfsi_band (se hacen las sumas sobre los intervalos de scfsi_band)
    //a) ∑|en_L[0][sb] - en_L[1][sb]| < en(scfsi_band)_krit = 10
    //b) ∑|xm[0][sb] - xm[1][sb]| < xm(scfsi_band)_krit = 10
    int crit_a, crit_b;

    for(int scfsi_band = 0; scfsi_band < 4; scfsi_band++){
        crit_a = 0, crit_b = 0;
        for (int sb = begIndex[scfsi_band]; sb < endIndex[scfsi_band]; sb++){
            crit_a += std::abs(en_L[0][sb] - en_L[1][sb]);
            crit_b += std::abs(xm[0][sb] - xm[1][sb]);
        }
        //Ambos criterios deben cumplirse para habilitar cada scfsi[i]
        if ((crit_a < 10) && (crit_b < 10))
            scfsi[scfsi_band] = true;
        else
            scfsi[scfsi_band] = false;
    }

    //Asignación de los valores actuales a los pasados
        //for (int sb = 0; sb < sb_len; sb++) en_L[0][sb] = en_L[1][sb];

    //-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-
    do{
        outerLoop();
        iteracion++;
    }
    while (endLoop == false);


    gr++; gr %= 2;
    outputMats();
}

void Quantization_Mod_Layer3::outerLoop(){

    innerLoop();
    calc_dist();
    preemphasis();
    amp_scalefactor_bands();
}

void Quantization_Mod_Layer3::calc_dist(){
```

```
    float sum, var1 = float(4.0/3.0), var2;
    if (block_type == 2){
        for (int block = 0; block < 3; block ++){
            var2 = powf(2.0,(qquant[block + 1]+quantanf)*0.25);
            for (int sb = 0; sb < sb_len; sb++) {
                sum = 0;
                for (int ξ = lbl[sb]; ξ < (lbl[sb] + bw[sb] - 1); ξ++)
                    sum += powf(fabsf(xr_S_array[block][ξ]) - powf(ix_S[block][ξ],var1)*var2,2.0)/float(bw[ξ]);
                xfsf_S[block][sb] = sum;
            }
        }
    }else{
        var2 = powf(2.0,(qquant[0] + quantanf)*0.25);
        for (int sb = 0; sb < sb_len; sb++) {
            sum = 0;
            for (int ξ = lbl[sb]; ξ < (lbl[sb] + bw[sb] - 1); ξ++)
                sum += powf(fabsf(xr_array[ξ]) - powf(ix_L[ξ],var1)*var2,2.0)/float(bw[ξ]);
            xfsf_L[sb] = sum;
        }
    }
}

void Quantization_Mod_Layer3::preemphasis(){ // Referencia ISO 11172-3 sección C.1.5.4.3.4, pg. 102

    /*
     Si el segundo gránulo se está procesando y scfsi está activo en
     al menos un scfsi_band, el preemphasis() previo en el segundo gránulo
     se establece igual a la configuración del primer gránulo
    */
    if (gr == 1){
        for (int scfsi_band = 0; scfsi_band < 4; scfsi_band++)
            if (scfsi[scfsi_band] ){
                preflag[1] = preflag[0];
                return;
            }
    }
    int fin = 0;
    /*preemphasis() es activado si en todos las cuatro bandas superiores de los scalefactors
      la distorsión actual supera el umbral después de la primera llamada del innerLoop()*/
    if ((block_type != 2) && (preflag[gr] == 0))
        for (int sb = 17; sb < 21; sb++)
            if (xfsf_L[sb] > xmin_L[1][sb])
                fin++;

    if (scalefac_scale[gr] == 0)
        ifqstep = sqrtf(2.0);
    else
        ifqstep = powf(2.0, 0.5*(1.0 + scalefac_scale[gr]));
    //ifqstep *= 18.0f/(4096.0);
    if (fin == 4) {
        preflag[gr] = 1;

        for (int sb = 0; sb < 21; sb++)
            xmin_L[gr][sb] *= powf(ifqstep, 2.0*float(pretab[sb]));

        //PREEMPHASIS per se:

        for (int scfsi_band = 0; scfsi_band < 4; scfsi_band++)
            if (scfsi[scfsi_band])
                for (int sb = begIndex[scfsi_band]; sb < endIndex[scfsi_band]; sb++)
                    for (int ξ = sb_beg[sb]; ξ < (sb_end[sb]+1); ξ++)
                        xr_array[ξ] *= powf(ifqstep, float(pretab[sb]));
    }
}

void Quantization_Mod_Layer3::amp_scalefactor_bands(){// Referencia ISO 11172-3 sección C.1.5.4.3.5, pg. 103

    copyScalefactor = false; preventScalefactor = false;
    int scfsi_band;
    if (gr == 1){
    //Si el segundo gránulo se está procesando y scfsi es 1 en al menos un scfsi_band (break):
        for (scfsi_band = 0; scfsi_band < 4; scfsi_band++)
            if (scfsi[scfsi_band]){

                //a) ifqstep debe establecerse de manera similar al del primer granulo (gr=0)
                if (scalefac_scale[0])
                    ifqstep = powf(2.0, 0.5*(1.0 + scalefac_scale[gr]));
                else
                    ifqstep = sqrtf(2.0);

                //ifqstep *= 18.0f/(4096.0);

                if (iteracion == 0)
                    /* b) Si es la primera iteración, los scalefactors de las bandas de los
                     scalefactors en las que scfsi está habilitado deben tomarse del primer gránulo */
```

```
                    copyScalefactor = true;
                else
                    /* c) Si no es la primera iteración, la amplificación debe prevenirse para
                     las bandas de los scalefactors en las que scfsi esté habilitado */
                    preventScalefactor = true;
                break;
            }
            else
                ifqstep = 1.0f;
    }

    if (block_type == 2){
        int couter[3] = {0,0,0};
        for (int bloque = 0; bloque < 3; bloque++){
            for (int sb = 0; sb < 12; sb++)
                if (xfsf_S[bloque][sb] > xmin_S[gr][bloque][sb]) {
                    couter[bloque] ++;
                    xmin_S[gr][bloque][sb] *= powf(ifqstep, 2.0);
                    scalefac_S[gr][bloque][sb] ++;
                    for (int ξ = sb_beg[sb]; ξ < (sb_end[sb]+1); ξ++)
                        xr_S_array[bloque][ξ] *= ifqstep;
                }
            if ((couter[0] == 0)&&(couter[1] == 0)&&(couter[2] == 0))
                endLoop = true;//todas las bandas están por debajo del límite de thm[sb]
        }
    }else{
        scfsi_band = 0;
        int counter = 0;
        for (int sb = 0; sb < 21; sb++){
            if (copyScalefactor || preventScalefactor){
                if ((sb == begIndex[scfsi_band + 1]) || (sb == 20))
                    scfsi_band ++;
                if (scfsi[scfsi_band]){
                    if (copyScalefactor)
                        scalefac_L[1][sb] = scalefac_L[0][sb];
                    continue;
                }
            }
            if (xfsf_L[sb] > xmin_L[gr][sb]) {
                counter ++;
                xmin_L[gr][sb] *= powf(ifqstep, 2.0);
                scalefac_L[gr][sb] ++;
                for (int ξ = sb_beg[sb]; ξ < (sb_end[sb]+1); ξ++)
                    xr_array[ξ] *= ifqstep;
            }
            if (counter == 0)
                endLoop = true;//todas las bandas están por debajo del límite de thm[sb]
        }

    }
}

void Quantization_Mod_Layer3::innerLoop(){

    if (block_type == 2){
        for(int block = 0; block < 3; block++){
            do{
                qquant[block + 1] ++; //de pronto hacerlo más bien con quantanf, como en el codigo de los de la
norma (da igual)
                quantize(block);
                ix_max(block);
            }
            while (max_ix[block + 1] > 8206);
        }
    }else{
        do{
            qquant[0] ++; //de pronto hacerlo más bien con quantanf, como en el codigo de los de la norma (da
igual)
            quantize(0);
            ix_max(0);
        }
        while (max_ix[0] > 8206);
        //std::cout << qquant[0] << std::endl;
    }

}

void Quantization_Mod_Layer3::quantize(int bl){
    float deno;

    if (block_type == 2){
        deno = powf(2.0,(qquant[bl + 1] + quantanf)*0.25);
        for (int ξ = 0; ξ < 192; ξ++){
            ix_S[bl][ξ] = nint(powf(fabsf(xr_S_array[bl][ξ])/deno,0.75) - 0.0946);
            //std::cout << "ix_S[" << bl << "][" << ξ << "] = " << ix_S[bl][ξ] << std::endl;
```

```
        }
    }else{
        deno = powf(2.0,(qquant[0] + quantanf)*0.25);
        for (int ξ = 0; ξ < 576; ξ++){
            ix_L[ξ] = nint(powf(fabsf(xr_array[ξ])/deno,0.75) - 0.0946);
            //std::cout << "ix_L[" << ξ << "] = " << ix_L[ξ] << std::endl;
        }
    }
}

void Quantization_Mod_Layer3::ix_max(int bl){

    int x = 0;

    if (block_type == 2){
        max_ix[bl + 1] = 0;
        for (int i = 0; i < 192; i++){
            x = abs(ix_S[bl][i]);
            if (x > max_ix[bl + 1])
                max_ix[bl + 1] = x;
        }
    }else{
        max_ix[0] = 0;
        for (int i = 0; i < 576; i++){
            x = abs(ix_L[i]);
            if (x > max_ix[0])
                max_ix[0] = x;
        }
    }
}

/*do{
 do{
 qquant ++; //de pronto hacerlo más bien con quantanf, como en el codigo de los de la norma
 quantize();
 //std::cout << "qquant = " << qquant << ", ix_max() = " << ix_max() << std::endl;
 }
 while (*ix_max() > 8191 + 14);

 }while (1);*/

/*============================================================================*/
//LAYER II
/*============================================================================*/

Quantization_Mod_Layer2::Quantization_Mod_Layer2(){
    //Factores de Giro reales para la DFT/STFT real
    for (int n = 0; n < 36; n++)
        for (int ξ = 0; ξ < 18; ξ++)
            re_W[ξ][n] = cos((2.0*M_PI/36.0)*n*ξ);

    for (int n = 0; n < 36; n++)
        w_hann[n] = 0.5 - 0.5*cos((2*M_PI/36)*(n - 0.5));


}

Quantization_Mod_Layer2::~Quantization_Mod_Layer2(){}

int Quantization_Mod_Layer2::nint(float in){ //1995 MPEG/audio software simulation group
    int temp;

    if(in < 0) temp = (int)(in - 0.5);
    else temp = (int)(in + 0.5);

    return (temp);
}


template <size_t rows, size_t cols>
void Quantization_Mod_Layer2::copyMat(int table[rows][cols], int opt_mat){

    aux_mat = new int *[rows];
    for (int i = 0; i < rows; i++)
        aux_mat[i] = new int[cols];

    for (int fila = 0; fila < rows; fila++)
        for (int col = 0; col < cols; col++)
            aux_mat[fila][col] = table[fila][col];

    switch (opt_mat) {
        case 0: B_8_Long = aux_mat; break;
        case 1: B_8_II = aux_mat; break;
    }
}
```

```
void Quantization_Mod_Layer2::fs_select(double fs){
    //llamar método fs_select(sample_rate) desde el getnextaudioblock
    //dependiendo de la frecuencia de muestreo se generan unas nuevas matrices dinámicas para las tablas B.8
Long y Short
    switch (int(fs)) {
        case 48000:
            copyMat<4>(B_8_48k_Long, 0); copyMat<2>(B_8_II_48k, 1);
            break;
        case 44100:
            copyMat<4>(B_8_44k_Long, 0); copyMat<2>(B_8_II_44k, 1);
            break;
        case 32000:
            copyMat<4>(B_8_32k_Long, 0); copyMat<2>(B_8_II_32k, 1);
            break;
        default:
            copyMat<4>(B_8_44k_Long, 0); copyMat<2>(B_8_II_44k, 1);
            break;
    }
    set_block();
}

void Quantization_Mod_Layer2::set_block(){

    sb_len = 21;
    lbl = B_8_Long[2]; bw = B_8_Long[1];
    sb_beg = B_8_Long[2]; sb_end = B_8_Long[3];

}

void Quantization_Mod_Layer2::inputMats(float **PQMF_coef, float **PSYxmin){

    s = PQMF_coef;

    float part_sum;

    //Primer cálculo de la STFT real:
    for (int k = 0; k < 32; k++)
        for (int ξ = 0; ξ < 18; ξ++){
            part_sum = 0;
            for (int n = 0; n < 36; n++)
                part_sum += w_hann[n] * re_W[ξ][n] * s[k][n];
            xr_array[k*18 + ξ] = part_sum;
            xr[k][ξ] = part_sum;
        }

    xmin = PSYxmin;
    float sum;
    //Terminación del cálculo de xmin:
    for (int sb = 0; sb < 21; sb++){
        sum = 0;
        for (int l = sb_beg[sb]; l < (sb_end[sb] + 1); l++)
            sum += xr_array[l]*xr_array[l];
        xmin[0][sb] *= sum;
        xmin[1][sb] *= sum;
    }

}

void Quantization_Mod_Layer2::resetVariables(){
    endLoop = false;
    memset(max_ix, 0, sizeof(max_ix));
    memset(ix, 0, sizeof(ix));
    memset(scalefac, 0, sizeof(scalefac));
    memset(scalefac_scale, 0, sizeof(scalefac_scale));
    memset(preflag, 0, sizeof(preflag));
    iteracion = 0;
    memset(qquant, 0, sizeof(preflag));
    quantanf = 0;

    float sum1 = 0, sum2 = 0;
    for (int k = 0; k < 32; k++)
        for (int ξ = 0; ξ < 18; ξ++){
            sum1 += powf(logf(xr[k][ξ]), 2.0f);
            sum2 += powf(xr[k][ξ], 2.0f);
        }

    sum1 = float(1.0/576.0)*sum1;
    sum2 = float(1.0/576.0)*sum2;
    sfm = powf(expf(1.0), sum1)/sum2;

    quantanf = nint(8.0*logf(sfm)); //cálculo válido solo 1 vez por cada 2 gránulos

    if (quantanf < -100)
        quantanf = -100;
```

```
    quantanf -= 70;

}

void Quantization_Mod_Layer2::iterationLoop(){
    resetVariables();
    //-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=
    //determinación de scfsi (pg. 100 - 101)
    float sum1 = 0, sum2 = 0;
    for (int k = 0; k < 32; k++)
        for (int ξ = 0; ξ < 18; ξ++)
            sum1 += powf(fabs(xr[k][ξ]), 2.0f);

    en_tot[1] = int(log2f(sum1));

    for (int sb = 0; sb < sb_len; sb++){
        sum2 = 0;
        for (int ξ = lbl[sb]; ξ < (lbl[sb] + bw[sb] - 1); ξ++)
            sum2 += powf(fabs(xr_array[ξ]), 2.0f);
        en_L[1][sb] = int(log2f(sum2));
        //xm[0][sb] = int(log2f(xmin_L[0][sb]));
        xm[1][sb] = int(log2f(xmin[1][sb]));
    }

    //-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=
    // Cuatro criterios: para el uso de scfsi pg. 101

    bool cond1 = false, cond2 = false, cond3 = false;

    //a) no todos los valores espectrales son cero.
    for (int ξ = 0; ξ < 576; ξ++)
        if (xr_array[ξ] != 0){
            cond1 = true;
            break;
        }

    //c) |en_tot[0] - en_tot[1]| < en_tot_krit = 10
    if (abs(en_tot[0] - en_tot[1]) < 10)
        cond2 = true;

    en_tot[0] = en_tot[1];

    //d) ∑|en_L[0][sb] - en_L[1][sb]| < en_dif_krit = 100
    float crit = 0;
    for (int sb = 0; sb < 21; sb++)
        crit += std::abs(en_L[0][sb] - en_L[1][sb]);
    if (crit < 100)
        cond3 = true;

    //Revisión del cumplimiento de los 4 criterios, si alguno de estos es falso se niegan los 4 scfsi[i]
    if ((cond1 == false) || (cond2 == false) || (cond3 == false))
        for (int scfsi_band = 0; scfsi_band < 4; scfsi_band++)
            scfsi[scfsi_band] = false;

    //-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=
    // Dos criterios para cada scfsi_band (se hacen las sumas sobre los intervalos de scfsi_band)
    //a) ∑|en_L[0][sb] - en_L[1][sb]| < en(scfsi_band)_krit = 10
    //b) ∑|xm[0][sb] - xm[1][sb]| < xm(scfsi_band)_krit = 10
    int crit_a, crit_b;

    for(int scfsi_band = 0; scfsi_band < 4; scfsi_band++){
        crit_a = 0; crit_b = 0;
        for (int sb = begIndex[scfsi_band]; sb < endIndex[scfsi_band]; sb++){
            crit_a += std::abs(en_L[0][sb] - en_L[1][sb]);
            crit_b += std::abs(xm[0][sb] - xm[1][sb]);
        }
        //std::cout << "crit_a = " << crit_a << ", crit_b = " << crit_b << std::endl;
        //Ambos criterios deben cumplirse para habilitar cada scfsi[i]
        if ((crit_a < 100) && (crit_b < 100))
            scfsi[scfsi_band] = true;
        else
            scfsi[scfsi_band] = false;
    }

    //Asignación de los valores actuales a los pasados
    //for (int sb = 0; sb < sb_len; sb++) en_L[0][sb] = en_L[1][sb];

    //-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=-=.=
    do{
        outerLoop();
        iteracion++;
    }
    while (endLoop == false);
```

```
    gr++; gr %= 2;
}

void Quantization_Mod_Layer2::outerLoop(){
    //segundo y enésimo cálculo de la STFT real
    float part_sum;
    for (int k = 0; k < 32; k++)
        for (int ξ = 0; ξ < 18; ξ++){
            part_sum = 0;
            for (int n = 0; n < 36; n++)
                part_sum += s[k][n] * w_hann[n] * re_W[ξ][n];
            xr_array[k*18 + ξ] = part_sum;
            xr[k][ξ] = part_sum;
        }

    innerLoop(); //siempre se cuantiza al menos una vez.
    calc_dist();
    preemphasis();
    amp_scalefactor_bands();
}

void Quantization_Mod_Layer2::calc_dist(){

    float sum,
    var1 = float(4.0/3.0), var2;
    var2 = powf(2.0,(qquant[0] + quantanf)*0.25);

    for (int sb = 0; sb < sb_len; sb++) {
        sum = 0;
        for (int ξ = lbl[sb]; ξ < (lbl[sb] + bw[sb] - 1); ξ++){
            sum += powf(fabsf(xr_array[ξ]) - powf(ix[ξ],var1)*var2,2.0)/float(bw[sb]);
        }
        xfsf[sb] = sum;
    }
}

void Quantization_Mod_Layer2::preemphasis(){ // Referencia ISO 11172-3 sección C.1.5.4.3.4, pg. 102

    /*
     Si el segundo gránulo se está procesando y scfsi está activo en
     al menos un scfsi_band, en el preemphasis() previo en el segundo gránulo
     se establece igual a la configuración del primer gránulo
     */
    if (gr == 1){
        for (int scfsi_band = 0; scfsi_band < 4; scfsi_band++)
            if (scfsi[scfsi_band] ){
                preflag[1] = preflag[0];
                return;
            }
    }
    int fin = 0;
    /*preemphasis() es activado si en todos las cuatro bandas superiores de los scalefactors
     la distorsión actual supera el umbral después de la primera llamada del innerLoop()*/
    if (preflag[gr] == 0)
        for (int sb = 17; sb < 21; sb++)
            if (xfsf[sb] > xmin[1][sb])
                fin++;

    if (scalefac_scale[gr] == 0)
        ifqstep = sqrtf(2.0);
    else
        ifqstep = powf(2.0, 0.5*(1.0 + scalefac_scale[gr]));

    if (fin == 4) {
        preflag[gr] = 1;

        for (int sb = 0; sb < 21; sb++)
            xmin[gr][sb] *= powf(ifqstep, 2.0*float(pretab[sb]));

        //PREEMPHASIS per se:

        for (int scfsi_band = 0; scfsi_band < 4; scfsi_band++)
            if (scfsi[scfsi_band])
                for (int k = B_8_II[0][scfsi_band]; k < B_8_II[1][scfsi_band]; k++)
                    for (int n = 0; n < 18; n++)
                        s[k][n] *= ifqstep/36.0f;
                    //no se debe hacer para las 36 muestras de tiempo n, ya que la segunda mitad para gr=1 se
copia a la primera mitad de s. (Se estaría multiplicando dos veces la mismas muestras de tiempo)
    }
}

void Quantization_Mod_Layer2::amp_scalefactor_bands(){// Referencia ISO 11172-3 sección C.1.5.4.3.5, pg. 103

    copyScalefactor = false; preventScalefactor = false;
    int scfsi_band;
```

```
    if (gr == 1){
        //Si el segundo gránulo se está procesando y scfsi es 1 en al menos un scfsi_band (break):
        for (scfsi_band = 0; scfsi_band < 4; scfsi_band++)
            if (scfsi[scfsi_band]){

                //a) ifqstep debe establecerse de manera similar al del primer granulo (gr=0)
                if (scalefac_scale[0])
                    ifqstep = powf(2.0, 0.5*(1.0 + scalefac_scale[gr]));
                else
                    ifqstep = sqrtf(2.0);

                //ifqstep *= 18.0f/(4096.0);

                if (iteracion == 0)
                /* b) Si es la primera iteración, los scalefactors de las bandas de los
                 scalefactors en las que scfsi está habilitado deben tomarse del primer gránulo */
                    copyScalefactor = true;
                else
                /* c) Si no es la primera iteración, la amplificación debe prevenirse para
                 las bandas de los scalefactors en las que scfsi esté habilitado */
                    preventScalefactor = true;
                break;
            }
            else
                ifqstep = 1.0f;
    }

    int counter = 0;
    endLoop = true; //si no entró a algún scfsi[scfsi_band], se sale del do-while
    for (int scfsi_band = 0; scfsi_band < 4; scfsi_band++){
        if (scfsi[scfsi_band])
            for (int sb = begIndex[scfsi_band]; sb < endIndex[scfsi_band]; sb++){
                endLoop = false;
                if (xfsf[sb] > xmin[gr][sb]) {
                    counter ++;
                    //std::cout << "counter = " << counter << std::endl;
                    xmin[gr][sb] *= powf(ifqstep, 2.0);
                    for (int k = B_8_II[0][scfsi_band]; k < B_8_II[1][scfsi_band]; k++)
                        for (int n = 0; n < 18; n++)
                            s[k][n] *= ifqstep/36.0f;
                }
                if ((counter == 0) || (sb == endIndex[3] - 1))
                    endLoop = true;//todas las bandas están por debajo del límite de thm[sb]
            }
    }
}

void Quantization_Mod_Layer2::innerLoop(){

    do{
        qquant[0] ++; //de pronto hacerlo más bien con quantanf, como en el codigo de los de la norma (da
igual)
        quantize(0);
        ix_max(0);
    }
    while (max_ix[0] > 8206);

}

void Quantization_Mod_Layer2::quantize(int bl){
    float deno;

    deno = powf(2.0,(qquant[0] + quantanf)*0.25);
    for (int ξ = 0; ξ < 576; ξ++){
        ix[ξ] = nint(powf(fabsf(xr_array[ξ])/deno, 0.75) - 0.0946);
    }
}

void Quantization_Mod_Layer2::ix_max(int bl){

    int x = 0;

    max_ix[0] = 0;
    for (int i = 0; i < 576; i++){
        x = abs(ix[i]);
        if (x > max_ix[0])
            max_ix[0] = x;
    }
}
```

PluginProcessor.h

```
#pragma once
```

```
#include <JuceHeader.h>

#include "PQMF_Analysis.h"
#include "IPQMF_Synthesis.h"
#include "MDCT_processing.h"
#include "Psychoacoustic_Mod.h"
#include "Quantization_Mod.h"

//==============================================================================
/**
*/
class Psyco_pixela_pluginAudioProcessor: public AudioProcessor
{
public:
    //==========================================================================
    Psyco_pixela_pluginAudioProcessor();
    ~Psyco_pixela_pluginAudioProcessor();

    //==========================================================================
    void prepareToPlay (double sampleRate, int samplesPerBlock) override;
    void releaseResources() override;

   #ifndef JucePlugin_PreferredChannelConfigurations
    bool isBusesLayoutSupported (const BusesLayout& layouts) const override;
   #endif

    void processBlock (AudioBuffer<float>&, MidiBuffer&) override;

    //==========================================================================
    AudioProcessorEditor* createEditor() override;
    bool hasEditor() const override;

    //==========================================================================
    const String getName() const override;

    bool acceptsMidi() const override;
    bool producesMidi() const override;
    bool isMidiEffect() const override;
    double getTailLengthSeconds() const override;

    //==========================================================================
    int getNumPrograms() override;
    int getCurrentProgram() override;
    void setCurrentProgram (int index) override;
    const String getProgramName (int index) override;
    void changeProgramName (int index, const String& newName) override;

    //==========================================================================
    void getStateInformation (MemoryBlock& destData) override;
    void setStateInformation (const void* data, int sizeInBytes) override;

    //==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.
    //Funciones y variables para el funcionamiento gráfico del analizador de espéctro
    void pushNextSampleIntoFifo (float sample) noexcept;
    void drawNextFrameOfSpectrum();
    enum{fftOrder  = 11, fftSize   = 1 << fftOrder, scopeSize = 512};
    bool buttonState = false;
    int caso_transform;
    bool nextFFTBlockReady = false;
    float scopeData [scopeSize];

    void interpol_recons(float *arr){
        for (int i = 31; i < 544; i+=32)
            arr[i] = 0.5*(arr[i-1] + arr[i+1]);
    }

private:
    //FFT para análisis espectral gráfico
    //dsp::FFT forwardFFT;
    //dsp::WindowingFunction<float> window;

    //Buffer que alberga las muestras de salida "reconSignal" de tamaño 576:
    AudioBuffer<float> reconSignal; float *recons_signal;
    //Buffer de tamaño variable, tamaño igual buffer.getNumSamples()
    AudioBuffer<float> aux_Array; float *auxArray;

    //Variables para la lógica de paso de muestras en tiempo real:
    int globalBlockIndex,i_end, j_end, ips;
    bool bool_preBlock, bool_postBlock;

    //Instancias/objetos de las clases externas:
    CircularPQMF_Buffer x;
    PQMF_Analysis PQMF_processor; float * output_analisis;
    IPQMF_Synthesis IPQMF_processor; float * output_sintesis;
```

```
    s_ProcessBandCoef_MAT s; //para Matriz de 32X36 que alberga las muestras filtradas por PQMF
    MDCT_processing MDCT_processor;

    //En esta implementación, se aplica el modelo psicoacústico 2 del Layer 3 en el Layer 2:
    Psychoacoustic_Mod2_Layer3 Psyco_Layer2;//Psychoacoustic_Mod2_Layer2 Psyco_Layer2;
    Psychoacoustic_Mod2_Layer3 Psyco_Layer3;
    Quantization_Mod_Layer2 Quant_Layer2;
    Quantization_Mod_Layer3 Quant_Layer3;

    float fifo [fftSize];
    float fftData [2 * fftSize];
    int fifoIndex = 0;

    //==============================================================================
    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (Psyco_pixela_pluginAudioProcessor)
};
```

PluginProcessor.cpp

```
#include "PluginProcessor.h"
#include "PluginEditor.h"

//==============================================================================
Psyco_pixela_pluginAudioProcessor::Psyco_pixela_pluginAudioProcessor()
#ifndef JucePlugin_PreferredChannelConfigurations
     : AudioProcessor (BusesProperties()
                     #if ! JucePlugin_IsMidiEffect
                      #if ! JucePlugin_IsSynth
                       .withInput  ("Input",  AudioChannelSet::mono(), true)
                      #endif
                       .withOutput ("Output", AudioChannelSet::mono(), true)
                     #endif
                       )
#endif{}

Psyco_pixela_pluginAudioProcessor::~Psyco_pixela_pluginAudioProcessor()
{
    x.~CircularPQMF_Buffer();
    PQMF_processor.~PQMF_Analysis();
    IPQMF_processor.~IPQMF_Synthesis();
    s.~s_ProcessBandCoef_MAT(); //para Matriz de 32X36 que alberga las muestras filtradas por PQMF
    MDCT_processor.~MDCT_processing();

    //En esta implementación, se aplica el modelo psicoacústico 2 del Layer 3 en el Layer 2:
    Psyco_Layer2.~Psychoacoustic_Mod2_Layer3();//Psychoacoustic_Mod2_Layer2 Psyco_Layer2;
    Psyco_Layer3.~Psychoacoustic_Mod2_Layer3();
    Quant_Layer2.~Quantization_Mod_Layer2();
    Quant_Layer3.~Quantization_Mod_Layer3();
}

//==============================================================================
const String Psyco_pixela_pluginAudioProcessor::getName() const{return JucePlugin_Name;}

bool Psyco_pixela_pluginAudioProcessor::acceptsMidi() const
{
   #if JucePlugin_WantsMidiInput
    return true;
   #else
    return false;
   #endif
}

bool Psyco_pixela_pluginAudioProcessor::producesMidi() const
{
   #if JucePlugin_ProducesMidiOutput
    return true;
   #else
    return false;
   #endif
}

bool Psyco_pixela_pluginAudioProcessor::isMidiEffect() const
{
   #if JucePlugin_IsMidiEffect
    return true;
   #else
    return false;
   #endif
}

double Psyco_pixela_pluginAudioProcessor::getTailLengthSeconds() const{return 0.0;}

int Psyco_pixela_pluginAudioProcessor::getNumPrograms()
{
    return 1;   // NB: some hosts don't cope very well if you tell them there are 0 programs,
```

```
                // so this should be at least 1, even if you're not really implementing programs.
}

int Psyco_pixela_pluginAudioProcessor::getCurrentProgram(){return 0;}

void Psyco_pixela_pluginAudioProcessor::setCurrentProgram (int index){}

const String Psyco_pixela_pluginAudioProcessor::getProgramName (int index){return {};}

void Psyco_pixela_pluginAudioProcessor::changeProgramName (int index, const String& newName){}

//==============================================================================
void Psyco_pixela_pluginAudioProcessor::releaseResources()
{
    //aux_Array.~AudioBuffer();
    //reconSignal.~AudioBuffer();
    // When playback stops, you can use this as an opportunity to free up any
    // spare memory, etc.
}

#ifndef JucePlugin_PreferredChannelConfigurations
bool Psyco_pixela_pluginAudioProcessor::isBusesLayoutSupported (const BusesLayout& layouts) const
{
  #if JucePlugin_IsMidiEffect
    ignoreUnused (layouts);
    return true;
  #else
    // This is the place where you check if the layout is supported.
    // In this template code we only support mono or stereo.
    if (layouts.getMainOutputChannelSet() != AudioChannelSet::mono()
     && layouts.getMainOutputChannelSet() != AudioChannelSet::stereo())
        return false;

    // This checks if the input layout matches the output layout
   #if ! JucePlugin_IsSynth
    if (layouts.getMainOutputChannelSet() != layouts.getMainInputChannelSet())
        return false;
   #endif

    return true;
  #endif
}
#endif

void Psyco_pixela_pluginAudioProcessor::prepareToPlay (double sampleRate, int samplesPerBlock)
{
    // Use this method as the place to do any pre-playback
    // initialisation that you need..

    std::cout << "fs = " << sampleRate << std::endl;
    std::cout << "samplesPerBlock = " << samplesPerBlock << std::endl;

    x.setBufferSize(512);
    x.clear();

    globalBlockIndex = 0;

    i_end = 0;
    j_end = samplesPerBlock;
    ips = 0;
    bool_preBlock = false; bool_postBlock = true;

    reconSignal.setSize(getTotalNumInputChannels(), 576); //reconSignal.clear();
    recons_signal = reconSignal.getWritePointer(0, 0);

    aux_Array.setSize(getTotalNumInputChannels(), samplesPerBlock); //aux_Array.clear();
    auxArray = aux_Array.getWritePointer(0, 0);

    Psyco_Layer2.fs_select(sampleRate);
    Quant_Layer2.fs_select(sampleRate);

    Psyco_Layer3.fs_select(sampleRate);
    Quant_Layer3.fs_select(sampleRate);
}

void Psyco_pixela_pluginAudioProcessor::processBlock (AudioBuffer<float>& buffer, MidiBuffer& midiMessages)
{
    ScopedNoDenormals noDenormals;
    auto totalNumInputChannels  = getTotalNumInputChannels();
    auto totalNumOutputChannels = getTotalNumOutputChannels();
    std::cout << "chan = " << totalNumInputChannels << ", " << totalNumOutputChannels << std::endl;
    //const float *audioData = buffer.getReadPointer(0);

    //for (auto i = totalNumInputChannels; i < totalNumOutputChannels; ++i)
      //  buffer.clear (i, 0, buffer.getNumSamples());
```

```
    if (buttonState == false){
        switch (caso_transform) {
            case 0://PQMF-MDCT LAYER 2
                if (bool_preBlock) {
                    for (int i = 0; i < i_end; i++)
                        auxArray[i] = recons_signal[i + ips];
                    ips += i_end; if (ips == 576){ips = 0;}
                }
                for(int blockIndex = 0; blockIndex < buffer.getNumSamples()/32; blockIndex++){
                    //Indexación de 32 muestras en el buffer circular. Ingresan 32 muestras por cada iteración
de blockIndex.
                    for (int i = 0; i < 32; i++){
                        //x.addSample(buffer.getSample(0, i + blockIndex*32));
                        //Psyco_Layer2.ingresoMuestraFifo(buffer.getSample(0, i + blockIndex*32));
                        //x.addSample(audioData[i + blockIndex*32]);
                        //Psyco_Layer2.ingresoMuestraFifo(audioData[i + blockIndex*32]);
                        x.addSample(*buffer.getReadPointer(0, i + blockIndex*32));
                        Psyco_Layer2.ingresoMuestraFifo(*buffer.getReadPointer(0, i + blockIndex*32));

                    }
                    //Procesamiento del buffer circular con el PQMF, e indexación de datos en la matriz de
32X36.
                    output_analisis = PQMF_processor.PQMF_Filtering(x.buffer);
                    s.add_Data(output_analisis); //se ingresan 32 nuevas muestras

                    if (globalBlockIndex == 17){ //apenas se llene la matriz s de 576 muestras, 32datos*18veces
                        Psyco_Layer2.run_Model();
                        Quant_Layer2.inputMats(s.bufferMat, Psyco_Layer2.xmin_L);
                        Quant_Layer2.iterationLoop();

                        float s_k[32];
                        for (int n = 0; n < 18; n++){
                            for(int k = 0; k < 32; k++) //s_k se llena de 32 subbandas por cada indice de
tiempo n
                                s_k[k] = s.bufferMat[k][n];

                            output_sintesis = IPQMF_processor.IPQMF_Filtering(s_k); //32 subbandas se de-
convolucionan
                            for(int k = 0; k < 32; k++)
                                recons_signal[k + n*32] = *(output_sintesis + k); //paso de matriz a arreglo
unidimensional
                        }
                        interpol_recons(recons_signal); //correción de muestras por promediación
                        s.circularData();

                        //en caso de usarse el modelo psico 2 del Layer 2:
                        //for (int sb = 0; sb < 32; sb++) Psyco_Layer2.SMR[0][sb] = Psyco_Layer2.SMR[1][sb];

                        Psyco_Layer2.copyOutputs();
                        //Aunque no se haga uso de los tipos de bloque en el programa global del Layer 2,
dentro del modelo psico. 2 del Layer 3 sí:
                        Psyco_Layer2.past_block_type = Psyco_Layer2.block_type;
                    }
                    globalBlockIndex++; globalBlockIndex %= 18;
                }

                if (bool_postBlock) {
                    for (int j = 0; j < j_end; j++)
                        auxArray[j + i_end] = recons_signal[j];
                    auxArray[i_end-1] = 0.5*(auxArray[i_end-2] + recons_signal[0]); //corrección por prom. cada
576 muestras
                    ips = j_end;
                }

                for (auto channel = totalNumInputChannels; channel < totalNumOutputChannels; ++channel){
                    auto* channelData = buffer.getWritePointer (channel);
                    for(int n = 0; n < buffer.getNumSamples(); n++)
                        channelData[n] = auxArray[n];
                }

               // for(int n = 0; n < buffer.getNumSamples(); n++) pushNextSampleIntoFifo(auxArray[n]);

                if ((buffer.getNumSamples() + ips) < 576){
                    i_end = buffer.getNumSamples();
                    bool_preBlock = true; bool_postBlock = false;
                }else{
                    bool_preBlock = true; bool_postBlock = true;
                    i_end = 576 - ips;
                    j_end = buffer.getNumSamples() - i_end;
                }
                break;
            //==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.
            //==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.==.
            case 1://PQMF-MDCT_LAYER 3
```

```
                if (bool_preBlock) {
                    for (int i = 0; i < i_end; i++)
                        auxArray[i] = recons_signal[i + ips];
                    ips += i_end; if (ips == 576){ips = 0;}
                }

                for(int blockIndex = 0; blockIndex < buffer.getNumSamples()/32; blockIndex++){
                    //Indexación de 32 muestras en el buffer circular. Ingresan 32 muestras por cada iteración
de blockIndex.

                    for (int i = 0; i < 32; i++){
                        //x.addSample(buffer.getSample(0, i + blockIndex*32));
                        //Psyco_Layer2.ingresoMuestraFifo(buffer.getSample(0, i + blockIndex*32));
                        //x.addSample(audioData[i + blockIndex*32]);
                        //Psyco_Layer2.ingresoMuestraFifo(audioData[i + blockIndex*32]);
                        x.addSample(*buffer.getReadPointer(0, i + blockIndex*32));
                        Psyco_Layer2.ingresoMuestraFifo(*buffer.getReadPointer(0, i + blockIndex*32));
                    }
                    //Procesamiento del buffer circular con el PQMF, e indexación de datos en la matriz de
32X36.
                    output_analisis = PQMF_processor.PQMF_Filtering(x.buffer);
                    s.add_Data(output_analisis);

                    if (globalBlockIndex == 17){ //apenas se llene la matriz s de 576 muestras, 32datos*18veces
                        Psyco_Layer3.run_Model();

                        MDCT_processor.set_blockType(Psyco_Layer3.block_type, Psyco_Layer3.past_block_type);
                        MDCT_processor.ventaneo_preMDCT(s.bufferMat);
                        MDCT_processor.direct_MDCT();

                        Quant_Layer3.set_blockType(Psyco_Layer3.block_type, Psyco_Layer3.past_block_type);
                        Quant_Layer3.inputMats(MDCT_processor.Ψ.matriz,
                                               MDCT_processor.Ψ_short,
                                               Psyco_Layer3.xmin_L,
                                               Psyco_Layer3.xmin_S);
                        Quant_Layer3.iterationLoop();

                        MDCT_processor.inverse_MDCT();
                        MDCT_processor.overlap_Add();

                        float s_k[32];
                        for (int n = 0; n < 18; n++){
                            for(int k = 0; k < 32; k++) //s_k se llena de 32 subbandas por cada indice de
tiempo n

                                s_k[k] = MDCT_processor.s_recons.matriz[k][n];

                            output_sintesis = IPQMF_processor.IPQMF_Filtering(s_k); //32 subbandas se de-
convolucionan

                            for(int k = 0; k < 32; k++)
                                recons_signal[k + n*32] = *(output_sintesis + k); //paso de matriz a arreglo
unidimensional
                        }
                        interpol_recons(recons_signal); //correción de muestras por promediación
                        MDCT_processor.cop_half_Alias_Recons();
                        s.circularData();

                        Psyco_Layer3.copyOutputs();
                        Psyco_Layer3.past_block_type = Psyco_Layer3.block_type;

                    }
                    globalBlockIndex++; globalBlockIndex %= 18;
                }

                if (bool_postBlock) {
                    for (int j = 0; j < j_end; j++)
                        auxArray[j + i_end] = recons_signal[j];
                    auxArray[i_end-1] = 0.5*(auxArray[i_end-2] + recons_signal[0]); //corrección por prom. cada
576 muestras
                    ips = j_end;
                }
                for (int channel = 0; totalNumInputChannels < totalNumOutputChannels; ++channel){
                    auto* channelData = buffer.getWritePointer(channel);
                    for(int n = 0; n < buffer.getNumSamples(); n++)
                        channelData[n] = auxArray[n];
                }

                //for(int n = 0; n < buffer.getNumSamples(); n++)pushNextSampleIntoFifo(auxArray[n]);

                if ((buffer.getNumSamples() + ips) < 576){
                    i_end = buffer.getNumSamples();
                    bool_preBlock = true; bool_postBlock = false;
                }else{
                    bool_preBlock = true; bool_postBlock = true;
                    i_end = 576 - ips;
                    j_end = buffer.getNumSamples() - i_end;
```

```
                }

                break;
        }

    }else{
       // for (int channel = 0; channel < totalNumInputChannels; ++channel){
         //   auto* channelData = buffer.getWritePointer(channel);
           /*for (auto n = 0; n < buffer.getNumSamples(); n++)
               pushNextSampleIntoFifo(channelData[n]);
            */
        //}
        //por defecto: buffer[n] = audioData[n];
    }

}

void Psyco_pixela_pluginAudioProcessor::pushNextSampleIntoFifo(float sample) noexcept{
    // if the fifo contains enough data, set a flag to say
    // that the next frame should now be rendered..
    /*if (fifoIndex == fftSize){
        if (! nextFFTBlockReady){
            zeromem (fftData, sizeof (fftData));
            memcpy (fftData, fifo, sizeof (fifo));
            nextFFTBlockReady = true;
        }
        fifoIndex = 0;
    }
    fifo[fifoIndex++] = sample;
     */
}

void Psyco_pixela_pluginAudioProcessor::drawNextFrameOfSpectrum(){
    /*
    // first apply a windowing function to our data
    window.multiplyWithWindowingTable (fftData, fftSize);

    // then render our FFT data..
    forwardFFT.performFrequencyOnlyForwardTransform (fftData);

    auto mindB = -100.0f;
    auto maxdB = 0.0f;

    for (int i = 0; i < scopeSize; ++i){
        auto skewedProportionX = 1.0f - std::exp (std::log (1.0f - i / (float) scopeSize) * 0.2f);
        auto fftDataIndex = jlimit (0, fftSize / 2, (int) (skewedProportionX * fftSize / 2));
        //std::cout << ((int) (skewedProportionX * fftSize/2)) << std::endl;
        auto level = jmap (jlimit (mindB, maxdB, Decibels::gainToDecibels (fftData[fftDataIndex])
                                   - Decibels::gainToDecibels ((float) fftSize)),
                           mindB, maxdB, 0.0f, 1.0f);
        scopeData[i] = level;
    }
     */
}

//==============================================================================
bool Psyco_pixela_pluginAudioProcessor::hasEditor() const
{
    return true; // (change this to false if you choose to not supply an editor)
}

AudioProcessorEditor* Psyco_pixela_pluginAudioProcessor::createEditor()
{
    return new Psyco_pixela_pluginAudioProcessorEditor (*this);
}

//==============================================================================
void Psyco_pixela_pluginAudioProcessor::getStateInformation (MemoryBlock& destData)
{
    // You should use this method to store your parameters in the memory block.
    // You could do that either as raw data, or use the XML or ValueTree classes
    // as intermediaries to make it easy to save and load complex data.
}

void Psyco_pixela_pluginAudioProcessor::setStateInformation (const void* data, int sizeInBytes)
{
    // You should use this method to restore your parameters from this memory block,
    // whose contents will have been created by the getStateInformation() call.
}

//==============================================================================
// This creates new instances of the plugin..
AudioProcessor* JUCE_CALLTYPE createPluginFilter()
{
    return new Psyco_pixela_pluginAudioProcessor();
```

```
}
```

PluginEditor.h

```
#pragma once

#include <JuceHeader.h>
#include "PluginProcessor.h"

//==============================================================================
/**
*/
class Psyco_pixela_pluginAudioProcessorEditor  : public AudioProcessorEditor//, private Timer
{
public:
    Psyco_pixela_pluginAudioProcessorEditor (Psyco_pixela_pluginAudioProcessor&);
    ~Psyco_pixela_pluginAudioProcessorEditor();

    //==============================================================================
    void resized() override;
    /*void paint (Graphics&) override;

    void drawFrame (Graphics& g);
    void timerCallback() override;
*/
private:
    // This reference is provided as a quick way for your editor to
    // access the processor object that created it.
    Psyco_pixela_pluginAudioProcessor& processor;

    Label PsicoPixelaLabel;
    Label titleLabel;
    //Label _60;Label _125;Label _250;Label _500;Label _1000;Label _2000;Label _4000;Label _8000; Label _Hz;

    ToggleButton bypassTogglebutton;

    Label textLabel { {}, "Procesamiento" }; //ComboBox 1
    Font textFont   { 17.0f };
    ComboBox styleMenu;

    void styleMenuChanged(){
        switch (styleMenu.getSelectedId()){
            case 1:
                processor.caso_transform = 0; break;
            case 2:
                processor.caso_transform = 1; break;
        }
        textLabel.setFont (textFont);
    }

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (Psyco_pixela_pluginAudioProcessorEditor)
};
```

PluginEditor.cpp

```
#include "PluginEditor.h"

//==============================================================================
Psyco_pixela_pluginAudioProcessorEditor::Psyco_pixela_pluginAudioProcessorEditor
(Psyco_pixela_pluginAudioProcessor& p)
    : AudioProcessorEditor (&p), processor (p)
{
    // Make sure that before the constructor has finished, you've set the
    // editor's size to whatever you need it to be.
    // ToggleButton de ByPass
    bypassTogglebutton.setButtonText("ByPass");
    bypassTogglebutton.onStateChange = [this] {processor.buttonState = bypassTogglebutton.getToggleState();};
    addAndMakeVisible(&bypassTogglebutton);

    //ComboBox
    addAndMakeVisible (textLabel); textLabel.setFont (textFont);
    addAndMakeVisible (styleMenu);
    styleMenu.addItem ("PQMF, Layer 2", 1);
    styleMenu.addItem ("PQMF-MDCT, Layer 3", 2);
    styleMenu.onChange = [this] { styleMenuChanged(); };
    styleMenu.setSelectedId (1);

    // Label de PiscoPixela
    PsicoPixelaLabel.setText("Psyco/Pseudo-Pixela", dontSendNotification);
    PsicoPixelaLabel.setFont(Font(25.f, Font::bold));
    addAndMakeVisible(&PsicoPixelaLabel);

    // Label de titulo
    titleLabel.setText("PQMF & IPQMF Process", dontSendNotification);
    titleLabel.setFont(Font(20.f, Font::bold));
    addAndMakeVisible(&titleLabel);
```

```
    /*_125.setText("125", dontSendNotification);_125.setFont(Font(11.f, Font::bold));addAndMakeVisible(&_125);
    _250.setText("250", dontSendNotification);_250.setFont(Font(11.f, Font::bold));addAndMakeVisible(&_250);
    _500.setText("500", dontSendNotification);_500.setFont(Font(11.f, Font::bold));addAndMakeVisible(&_500);
    _1000.setText("1000", dontSendNotification);_1000.setFont(Font(11.f,
Font::bold));addAndMakeVisible(&_1000);
    _2000.setText("2000", dontSendNotification);_2000.setFont(Font(11.f,
Font::bold));addAndMakeVisible(&_2000);
    _4000.setText("4000", dontSendNotification);_4000.setFont(Font(11.f,
Font::bold));addAndMakeVisible(&_4000);
    _8000.setText("8000", dontSendNotification);_8000.setFont(Font(11.f,
Font::bold));addAndMakeVisible(&_8000);
    _Hz.setText("Hz", dontSendNotification);_Hz.setFont(Font(12.f, Font::bold));addAndMakeVisible(&_Hz);
    */

    setSize (800, 600);
    //startTimerHz (30);
}

Psyco_pixela_pluginAudioProcessorEditor::~Psyco_pixela_pluginAudioProcessorEditor(){}
/*
void Psyco_pixela_pluginAudioProcessorEditor::timerCallback(){
    if (processor.nextFFTBlockReady){
        processor.drawNextFrameOfSpectrum();
        processor.nextFFTBlockReady = false;
        repaint();
    }
}*/
//==============================================================================
/*void Psyco_pixela_pluginAudioProcessorEditor::paint (Graphics& g)
{
    // (Our component is opaque, so we must completely fill the background with a solid colour)
    g.fillAll (getLookAndFeel().findColour (ResizableWindow::backgroundColourId));

    g.setColour (Colours::white);
    g.setFont (15.0f);
    //g.drawFittedText ("Hello World!", getLocalBounds(), Justification::centred, 1);


    auto width  = getLocalBounds().getWidth()-100;
    auto height = getLocalBounds().getHeight();

    //g.fillAll (Colours::blanchedalmond);
    g.fillAll (Colours::black);
    g.setOpacity(1.0f); g.setColour (Colours::white);
    drawFrame(g);

    auto min_pos = 0.27f;
    auto max_pos = 0.3f;
    int scopeSize = 512;

    g.drawLine ({
        (float) jmap (0, 0, scopeSize , 100, width),
        jmap (0.285f,0.0f, 1.0f, (float) height, 0.0f),
        (float) jmap (512, 0, scopeSize , 100, width),
        jmap (0.285f,0.0f, 1.0f, (float) height, 0.0f) });

    g.drawLine ({
        (float) jmap (0, 0, scopeSize , 100, width),
        jmap (0.285f,0.0f, 1.0f, (float) height, 0.0f),
        (float) jmap (0, 0, scopeSize , 100, width),
        jmap (0.9f,0.0f, 1.0f, (float) height, 0.0f) });

    int f = 15; //125 Hz
    g.drawLine ({ (float) jmap (f, 0, scopeSize , 100, width),
        jmap (min_pos, 0.0f, 1.0f, (float) height, 0.0f),
        (float) jmap (f,0, scopeSize , 100, width),
        jmap (max_pos,0.0f, 1.0f, (float) height, 0.0f) });
    f = 30; //250 Hz
    g.drawLine ({ (float) jmap (f, 0, scopeSize , 100, width),
        jmap (min_pos, 0.0f, 1.0f, (float) height, 0.0f),
        (float) jmap (f,0, scopeSize, 100, width),
        jmap (max_pos,0.0f, 1.0f, (float) height, 0.0f) });
    f = 56; //500 Hz
    g.drawLine ({ (float) jmap (f, 0, scopeSize , 100, width),
        jmap (min_pos, 0.0f, 1.0f, (float) height, 0.0f),
        (float) jmap (f,0, scopeSize , 100, width),
        jmap (max_pos,0.0f, 1.0f, (float) height, 0.0f) });
    f = 108; //1k Hz
    g.drawLine ({ (float) jmap (f, 0, scopeSize , 100, width),
        jmap (min_pos, 0.0f, 1.0f, (float) height, 0.0f),
        (float) jmap (f,0, scopeSize , 100, width),
        jmap (max_pos,0.0f, 1.0f, (float) height, 0.0f) });
    f = 195;  //2k Hz
    g.drawLine ({ (float) jmap (f, 0, scopeSize , 100, width),
```

```
        jmap (min_pos, 0.0f, 1.0f, (float) height, 0.0f),
        (float) jmap (f,0, scopeSize , 100, width),
        jmap (max_pos,0.0f, 1.0f, (float) height, 0.0f) });
    f = 326; //4k Hz
    g.drawLine ({ (float) jmap (f, 0, scopeSize , 100, width),
        jmap (min_pos, 0.0f, 1.0f, (float) height, 0.0f),
        (float) jmap (f,0, scopeSize , 100, width),
        jmap (max_pos,0.0f, 1.0f, (float) height, 0.0f) });
    f = 459; //8k Hz
    g.drawLine ({ (float) jmap (f, 0, scopeSize , 100, width),
        jmap (min_pos, 0.0f, 1.0f, (float) height, 0.0f),
        (float) jmap (f,0, scopeSize , 100, width),
        jmap (max_pos,0.0f, 1.0f, (float) height, 0.0f) });

}*/
/*
void Psyco_pixela_pluginAudioProcessorEditor::drawFrame (Graphics& g){

    int scopeSizer = processor.scopeSize;
    float *scopeDatar = processor.scopeData;
    for (int i = 1; i < scopeSizer; ++i){
        auto width  = getLocalBounds().getWidth() - 100;
        auto height = getLocalBounds().getHeight();

        g.drawLine ({ (float) jmap (i - 1, 0, scopeSizer - 1, 100, width),
            jmap (scopeDatar[i - 1], -0.4f, 1.0f, (float) height, 0.0f),
            (float) jmap (i,     0, scopeSizer - 1, 100, width),
            jmap (scopeDatar[i],     -0.4f, 1.0f, (float) height, 0.0f) });
    }

}*/

void Psyco_pixela_pluginAudioProcessorEditor::resized()
{
    // ToggleButtons
    //bypassTogglebutton.setBounds(getWidth() *0.77, 200+getHeight()/2, 80, 40);
    bypassTogglebutton.setBounds(100, 230+getHeight()/2, 80, 40);

    // Labels
    PsicoPixelaLabel.setBounds(400, 10, getWidth() - 100, 40);
    titleLabel.setBounds(10, 10, getWidth() - 100, 40);

    //combobox
    int border = 100;
    textLabel.setBounds(getWidth()-750, 490, getWidth() - border, 40);
    styleMenu.setBounds(getWidth()-620, 490, getWidth() - (border+470), 40);

 /*
    auto width  = getLocalBounds().getWidth()-100;
    auto height = getLocalBounds().getHeight();
    int scopeSize = 512; int factoy = 10;
    int f = 6;
    _125.setBounds((float) jmap (f, 0, scopeSize , 100, width), jmap (0.27f, 0.0f, 1.0f, (float) height, 0.0f)-
factoy, 80, 40);
    f = 25;
    _250.setBounds((float) jmap (f, 0, scopeSize , 100, width), jmap (0.27f, 0.0f, 1.0f, (float) height, 0.0f)-
factoy, 80, 40);
    f = 49;
    _500.setBounds((float) jmap (f, 0, scopeSize , 100, width), jmap (0.27f, 0.0f, 1.0f, (float) height, 0.0f)-
factoy, 80, 40);
    f = 100;
    _1000.setBounds((float) jmap (f, 0, scopeSize , 100, width), jmap (0.27f, 0.0f, 1.0f, (float) height,
0.0f)-factoy, 80, 40);
    f = 187;
    _2000.setBounds((float) jmap (f, 0, scopeSize , 100, width), jmap (0.27f, 0.0f, 1.0f, (float) height,
0.0f)-factoy, 80, 40);
    f = 320;
    _4000.setBounds((float) jmap (f, 0, scopeSize , 100, width), jmap (0.27f, 0.0f, 1.0f, (float) height,
0.0f)-factoy, 80, 40);
    f = 453;
    _8000.setBounds((float) jmap (f, 0, scopeSize , 100, width), jmap (0.27f, 0.0f, 1.0f, (float) height,
0.0f)-factoy, 80, 40);
    f = 512;
    _Hz.setBounds((float) jmap (f, 0, scopeSize , 100, width), jmap (0.27f, 0.0f, 1.0f, (float) height, 0.0f)-
factoy, 80, 40);
 */
}
```

TablasDatos.cpp

```
#include <stdio.h>
/*=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.
TABLAS LAYER 3
=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.
```

```
// pg. 82
// C.7(a,b,c) LONG:  índice, FFT-Lines, minval, qthr, norm, bval
// C.7(d,e,f) SHORT: índice, FFT-Lines, qthr, norm, SNR(dBFS), bval
// C.7.a -- fs = 48 kHz bloques Long
float C_7_48k_Long[6][62] = {

{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,
40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,2,2,3,3,3,3,3,4,4,4,4,5,5,5,6,7,7,8,8,8,10,10,10,13,13
,14,18,18,20,25,25,35,67,67},

{24.5,24.5,24.5,24.5,24.5,20,20,20,20,20,20,18,18,18,12,12,6,6,3,3,3,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{4.532,4.532,4.532,0.904,0.904,0.090,0.090,0.029,0.029,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.018,0.018,0.
018,0.018,0.018,0.018,0.018,0.018,0.018,0.018,0.018,0.018,0.018,0.018,0.027,0.027,0.027,0.027,0.027,0.036,0.036,0.036
,0.036,0.036,0.045,0.045,0.045,0.054,0.063,0.063,0.072,0.072,0.072,0.180,0.180,0.180,0.372,0.372,0.400,1.628,1.
628,1.808,22.607,22.607,31.650,605.867,605.867},

{0.970,0.755,0.738,0.730,0.724,0.723,0.723,0.723,0.718,0.690,0.660,0.641,0.600,0.584,0.531,0.537,0.857,0.858,0.
853,0.824,0.778,0.740,0.709,0.676,0.632,0.592,0.553,0.510,0.513,0.608,0.673,0.636,0.586,0.571,0.616,0.640,0.597
,0.538,0.512,0.528,0.516,0.493,0.499,0.525,0.541,0.528,0.510,0.506,0.525,0.536,0.518,0.501,0.496,0.497,0.495,0.
494,0.497,0.494,0.487,0.483,0.482,0.524},

{0.000,0.469,0.937,1.406,1.875,2.344,2.812,3.281,3.750,4.199,4.625,5.047,5.437,5.828,6.187,6.522,7.174,7.800,8.
402,8.966,9.483,9.966,10.426,10.866,11.279,11.669,12.042,12.386,12.721,13.115,13.561,13.983,14.371,14.741,15.14
0,15.562,15.962,16.324,16.665,17.020,17.373,17.708,18.045,18.398,18.762,19.120,19.466,19.807,20.159,20.522,20.8
73,21.214,21.553,21.892,22.231,22.569,22.909,23.248,23.583,23.915,24.246,24.576}
    };
// C.7.b -- fs = 44.1 kHz bloques Long
float C_7_44k_Long[6][63] = {

{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,
40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62},

{1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,4,4,4,5,5,5,6,6,7,7,7,9,9,9,10,10,11,14,
14,15,20,20,21,27,27,36,73,18},

{24.5,24.5,24.5,24.5,24.5,20,20,20,20,20,20,20,18,18,18,12,6,6,3,3,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{4.532,4.532,4.532,0.904,0.904,0.090,0.090,0.029,0.029,0.029,0.009,0.009,0.009,0.009,0.009,0.018,0.018,0.018,0.
018,0.018,0.018,0.018,0.018,0.018,0.018,0.018,0.018,0.018,0.018,0.018,0.027,0.027,0.027,0.027,0.027,0.027,0.027,0.036,0.036
,0.036,0.045,0.045,0.045,0.054,0.054,0.063,0.063,0.063,0.081,0.081,0.081,0.180,0.180,0.198,0.400,0.400,0.429,1.
808,1.808,1.899,24.415,24.415,32.554,660.124,162.770},

{0.951,0.700,0.681,0.675,0.667,0.665,0.664,0.664,0.664,0.655,0.616,0.597,0.578,0.541,0.575,0.856,0.846,0.840,0.
822,0.800,0.753,0.704,0.674,0.640,0.609,0.566,0.535,0.531,0.615,0.686,0.650,0.611,0.567,0.520,0.513,0.557,0.584
,0.570,0.579,0.585,0.548,0.536,0.550,0.532,0.504,0.496,0.516,0.527,0.516,0.497,0.489,0.502,0.502,0.491,0.497,0.
504,0.504,0.495,0.486,0.484,0.483,0.475,0.515},
    {0.000,
0.431,0.861,1.292,1.723,2.153,2.584,3.015,3.445,3.876,4.279,4.670,5.057,5.415,5.774,6.422,7.026,7.609,8.168,8.7
10,9.207,9.662,10.099,10.515,10.917,11.293,11.652,11.997,12.394,12.850,13.277,13.681,14.062,14.411,14.751,15.11
9,15.508,15.883,16.263,16.654,17.020,17.374,17.744,18.104,18.447,18.781,19.130,19.487,19.838,20.179,20.510,20.8
52,21.196,21.531,21.870,22.214,22.558,22.898,23.232,23.569,23.897,24.229,24.542}
    };
// C.7.c -- fs = 32 kHz bloques Long
float C_7_32k_Long[6][59] = {

{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,
40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58},

{2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,4,4,4,5,5,5,5,5,6,6,6,8,8,8,10,10,10,12,12,13,14,14,16,2
0,20,23,27,27,32,37,37,12},

{24.5,24.5,24.5,20,20,20,20,20,18,18,12,12,6,6,6,3,3,3,3,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {9.064, 9.064,
1.808,0.181,0.181,0.0570,0.018,0.018,0.018,0.018,0.018,0.018,0.018,0.018,0.018,0.018,0.018,0.018,0.018,0.027,0.027,0.
027,0.027,0.027,0.027,0.027,0.027,0.027,0.036,0.036,0.036,0.045,0.045,0.045,0.045,0.045,0.054,0.054,0.054,0.072
,0.072,0.072,0.090,0.090,0.090,0.108,0.108,0.117,0.252,0.252,0.289,0.572,0.572,0.658,2.441,2.441,2.894,33.458,3
3.458,10.851},

{0.997,0.893,0.881,0.873,0.872,0.871,0.860,0.839,0.812,0.784,0.741,0.697,0.674,0.651,0.633,0.611,0.589,0.575,0.
654,0.724,0.701,0.673,0.631,0.592,0.553,0.510,0.505,0.562,0.598,0.589,0.607,0.620,0.580,0.532,0.517,0.517,0.509
,0.506,0.522,0.531,0.519,0.512,0.509,0.497,0.494,0.501,0.507,0.502,0.493,0.497,0.506,0.510,0.504,0.496,0.493,0.
490,0.483,0.458,0.500},

{0.312,0.937,1.562,2.187,2.812,3.437,4.045,4.625,5.173,5.173,5.698,6.184,6.634,7.070,7.492,7.905,8.305,8.695,9.
064,9.483,9.966,10.425,10.866,11.669,12.042, 12.386, 12.721, 13.091, 13.488, 13.873, 14.268, 14.679, 15.067,
15.424, 15.771, 16.120, 16.466, 16.807, 17.158, 17.518, 17.869, 18.215, 18.562, 18.902, 19.239, 19.579, 19.925,
20.269, 20.606, 20.944, 21.288, 21.635, 21.979, 22.319, 22.656, 22.993,23.326, 23.656, 23.937}
    };
// C.7.d -- fs = 48 kHz bloques short
```

```
float C_7_48k_Short[6][38] = {
    {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37},
    {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,3,3,3,4,4,5,5,5,7,7,11,17},

{4.532,0.904,0.029,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.018,0.018,0.018,0.018,0.018,0.018,0.018,0.054,0.054,0.054,0.114,0.114,0.452,0.452,0.452,6.330,6.330,9.947,153.727},
    {0.970,0.755,0.738,0.730,0.724,0.723,0.723,0.723,0.718,
0.690,0.660,0.641,0.600,0.584,0.532,0.537,0.857,0.858,0.853,0.824,0.778,0.740,0.709,0.676,0.632,0.592,0.553,0.510,0.513,0.608,0.673,0.637,0.586,0.571,0.616,0.640,0.597,0.538},
    {-8.240,-8.240,-8.240,-8.240,-8.240,-8.240,-8.240,-8.240,-8.240,-8.240,-7.447,-7.447,-7.447,-7.447,-7.447,-7.447,-7.447,-7.447,-7.447,-7.447,-6.990,-6.990,-6.990,-6.990,-6.990,-6.990,-6.990,-6.020,-6.020,-6.020,-6.020,-5.229,-5.229,-5.229,-5.229},
    {0.000, 1.875, 3.750, 5.437, 6.857, 8.109, 9.237, 10.202, 11.083,11.864, 12.553, 13.195, 13.781, 14.309, 14.803, 15.250, 15.667, 16.068, 16.409, 17.044, 17.607, 18.097, 18.528, 18.930, 19.295, 19.636, 20.038, 20.486, 20.900, 21.305, 21.722, 22.128, 22.512, 22.877, 23.241, 23.616, 23.974, 24.312}
    };
// C.7.e -- fs = 44.1 kHz bloques short
float C_7_44k_Short[6][39] = {

{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38},
    {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,3,3,3,3,3,4,4,5,5,5,7,7,7,19},

{4.532,0.904,0.029,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.018,0.018,0.018,0.018,0.018,0.018,0.027,0.027,0.054,0.054,0.054,0.114,0.114,0.452,0.452,0.452,6.33,6.33,6.33,171.813},

{0.952,0.700,0.681,0.675,0.667,0.665,0.664,0.664,0.664,0.655,0.616,0.597,0.578,0.541,0.575,0.856,0.846,0.84,0.822,0.8,0.753,0.704,0.674,0.64,0.609,0.566,0.535,0.531,0.615,0.686,0.650,0.612,0.567,0.520,0.513,0.557,0.584,0.570,0.578},
    {-8.24,-8.24,-8.24,-8.24,-8.24,-8.24,-8.24,-8.24,-8.24,-8.24,-7.447,-7.447,-7.447,-7.447,-7.447,-7.447,-7.447,-7.447,-7.447,-7.447,-6.990,-6.990,-6.990,-6.990,-6.990,-6.990,-6.990,-6.990,-6.990,-6.020,-6.020,-6.020,-6.020,-5.229,-5.229,-5.229,-5.229,-4.559},

{0.000,1.723,3.445,5.057,6.422,7.609,8.71,9.662,10.515,11.293,12.009,12.625,13.21,13.748,14.241,14.695,15.125,15.508,15.891,16.537,17.112,17.62,18.073,18.47,18.849,19.271,19.741,20.177,20.576,20.95,21.316,21.699,22.078,22.438,22.782,23.133,23.484,23.828,24.173}
    };
// C.7.f -- fs = 32 kHz bloques short
float C_7_32k_Short[6][42] = {

{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41},
    {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,3,3,3,3,3,4,4,4,5,5,6,7,7,8,10},

{4.532,0.904,0.09,0.029,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.009,0.018,0.018,0.018,0.018,0.018,0.018,0.018,0.018,0.027,0.027,0.027,0.027,0.027,0.072,0.072,0.072,0.143,0.143,0.172,0.633,0.633,0.723,9.043},
    {0.997, 0.893, 0.881, 0.873, 0.872, 0.871, 0.860, 0.839, 0.812, 0.784, 0.741, 0.697, 0.674, 0.651, 0.633, 0.611, 0.589, 0.575, 0.654, 0.724, 0.701, 0.673, 0.631, 0.592, 0.553, 0.510, 0.506, 0.562, 0.598, 0.589, 0.607, 0.620, 0.580, 0.532, 0.517, 0.517, 0.509, 0.506, 0.522, 0.531, 0.519, 0.512},
    {-8.24,-8.24,-8.24,-8.24,-8.24,-8.24,-8.24,-8.24,-8.24,-7.447,-7.447,-7.447,-7.447,-7.447,-7.447,-7.447,-7.447,-7.447,-7.447,-7.447,-6.99,-6.99,-6.99,-6.99,-6.99,-6.99,-6.99,-6.99,-6.99,-6.02,-6.02,-6.02,-6.02,-5.229,-5.229,-5.229,-5.229,-4.559,-4.559, -3.980, -3.980},

{0.000,1.25,2.5,3.75,4.909,5.958,6.857,7.7,8.5,9.237,9.895,10.5,11.083,11.604,12.107,12.554,13,13.391,13.781,14.474,15.096,15.667,16.177,16.636,17.057,17.429,17.786,18.177,18.597,18.994,19.352,19.693,20.066,20.461,20.841,21.201,21.549,21.911,22.275,22.625,22.971,23.321}
    };
// pg. 88
// C.8(a,b,c) LONG:  índice sb, cbw, bu, bo, w1, w2
// C.8(d,e,f) SHORT: índice sb, cbw, bu, bo, w1, w2
// C.8.a -- fs = 48 kHz bloques Long
float C_8_48k_Long[6][21] = {
    {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20},
    {3,3,4,3,3,2,3,2,3,3,3,3,4,3,3,3,3,2,3,2,1},
    {0,4,7,11,14,17,19,22,24,27,30,33,36,40,43,46,49,52,54,57,59},
    {4,7,11,14,17,19,22,24,27,30,33,36,40,43,46,49,52,54,57,59,60},

{1.000,0.944,0.389,0.833,0.278,0.361,0.583,0.917,0.250,0.583,0.352,0.389,0.375,0.856,0.611,0.840,0.783,0.816,0.114,0.687,0.548},

{0.056,0.611,0.167,0.722,0.639,0.417,0.083,0.750,0.417,0.648,0.611,0.625,0.144,0.389,0.160,0.217,0.184,0.886,0.313,0.452,0.908}
    };
float C_8_44k_Long[6][21] = {
    {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20},
    {3,3,4,3,3,1,3,3,3,3,3,3,4,3,3,3,3,3,2,2,2},
    {0,4,7,11,14,17,18,21,24,27,30,33,36,40,43,46,49,52,55,57,59},
    {4,7,11,14,17,18,21,24,27,30,33,36,40,43,46,49,52,55,57,59,61},

{1.000,0.944,0.389,0.833,0.278,0.861,0.083,0.417,0.750,0.194,0.426,0.463,0.180,0.900,0.532,0.376,0.550,0.448,0.597,0.357,0.278},
```

```
{0.056,0.611,0.167,0.722,0.139,0.917,0.583,0.250,0.805,0.574,0.537,0.819,0.100,0.468,0.623,0.450,0.552,0.403,0.
643,0.722,0.960}
    };
float C_8_32k_Long[6][21] = {
    {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20},
    {1,2,2,1,2,2,3,2,3,3,4,4,4,3,4,3,4,3,3,2,2},
    {0,2,4,6,7,9,11,14,16,19,22,26,30,34,37,41,44,48,51,54,56},
    {2,4,6,7,9,11,14,16,19,22,26,30,34,37,41,44,48,51,54,56,58},

{1.000,0.472,0.694,0.917,0.139,0.361,0.583,0.917,0.250,0.130,0.167,0.611,0.522,0.967,0.083,0.383,0.005,0.726,0.
519,0.739,0.116},

{0.528,0.305,0.083,0.861,0.639,0.417,0.083,0.750,0.870,0.833,0.389,0.478,0.033,0.917,0.617,0.995,0.274,0.480,0.
261,0.884,1.000}
    };
float C_8_48k_Short[6][12] = {
    {0,1,2,3,4,5,6,7,8,9,10,11},
    {2,2,3,3,4,4,3,4,4,2,3,2},
    {0,3,5,8,11,15,19,22,26,30,32,35},
    {3,5,8,11,15,19,22,26,30,32,35,37},
    {1.000,0.833,0.167,0.500,0.833,0.833,0.417,0.083,0.055,0.958,0.433,0.833},
    {0.167,0.833,0.500,0.167,0.167,0.583,0.917,0.944,0.042,0.567,0.167,0.618}
    };
float C_8_44k_Short[6][12] = {
    {0,1,2,3,4,5,6,7,8,9,10,11},
    {2,2,3,3,4,5,3,4,3,3,3,2},
    {0,3,5,8,11,15,20,23,27,30,33,36},
    {3,5,8,11,15,20,23,27,30,33,36,38},
    {1.000,0.833,0.167,0.500,0.833,0.833,0.750,0.417,0.944,0.625,0.700,0.833},
    {0.167,0.833,0.500,0.167,0.167,0.250,0.583,0.055,0.375,0.300,0.167,1.000}
    };
float C_8_32k_Short[6][12] = {
    {0,1,2,3,4,5,6,7,8,9,10,11},
    {2,2,3,3,4,5,4,5,4,4,3,1},
    {0,3,5,8,11,15,20,24,29,33,37,40},
    {3,5,8,11,15,20,24,29,33,37,40,41},
    {1.000,0.833,0.167,0.500,0.833,0.833,0.750,0.750,0.944,0.625,0.528,0.062},
    {0.167,0.833,0.500,0.167,0.167,0.250,0.250,0.055,0.375,0.472,0.937,1.000}
    };

//pg. 53
// B.6 --LAYER 3 preemphasis (pretab)
// cada índice corresponde a un valor para cada scalefactor band
int pretab[21] = {0,0,0,0,0,0,0,0,0,0,1,1,1,1,2,2,3,3,3,2};
// pg. 62
// B.8(a,b,c) LONG/SHORT:  scalefactor band, width of band, index of start, index of end
//
// B.8.a -- fs = 32 kHz bloques Long y Short
int B_8_32k_Long[4][21] = {
    {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20},
    {4,4,4,4,4,4,6,6,8,10,12,16,20,24,30,38,46,56,68,84,102},
    {0,4,8,12,16,20,24,30,36,44,54,66,82,102,126,156,194,240,296,364,448},
    {3,7,11,15,19,23,29,35,43,53,65,81,101,125,155,193,239,295,363,447,549},
};

int B_8_32k_Short[4][12] = {
    {0,1,2,3,4,5,6,7,8,9,10,11},
    {4,4,4,4,6,8,12,16,20,26,34,42},
    {0,4,8,12,16,22,30,42,58,78,104,138},
    {3,7,11,15,21,29,41,57,77,103,137,179},
};
// B.8.b -- fs = 44.1 kHz bloques Long y Short
int B_8_44k_Long[4][21] = {
    {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20},
    {4,4,4,4,4,4,6,6,8,8,10,12,16,20,24,28,34,42,50,54,76},
    {0,4,8,12,16,20,24,30,36,44,52,62,74,90,110,134,162,196,238,288,342},
    {3,7,11,15,19,23,29,35,43,51,61,73,89,109,133,161,195,237,287,341,417},
};
int B_8_44k_Short[4][12] = {
    {0,1,2,3,4,5,6,7,8,9,10,11},
    {4,4,4,4,6,8,10,12,14,18,22,30},
    {0,4,8,12,16,22,30,40,52,66,84,106},
    {3,7,11,15,21,29,39,51,65,83,105,135},
};
// B.8.c -- fs = 48 kHz bloques Long y Short
int B_8_48k_Long[4][21] = {
    {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20},
    {4,4,4,4,4,4,6,6,6,8,10,12,16,18,22,28,34,40,46,54,54},
    {0,4,8,12,16,20,24,30,36,42,50,60,72,88,106,128,156,190,230,276,330},
    {3,7,11,15,19,23,29,35,41,49,59,71,87,105,127,155,189,229,275,329,383},
};
int B_8_48k_Short[4][12] = {
    {0,1,2,3,4,5,6,7,8,9,10,11},
    {4,4,4,4,6,6,10,12,14,16,20,26},
```

```
    {0,4,8,12,16,22,28,38,50,64,80,100},
    {3,7,11,15,21,27,37,49,63,79,99,125},
};
//=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.
//Indices iniciales para Psychoacoustic_Mod
int B8Long[3][22] = {
    {0,4,8,12,16,20,24,30,36,44,54,66,82,102,126,156,194,240,296,364,448,550},
    {0,4,8,12,16,20,24,30,36,44,52,62,74,90,110,134,162,196,238,288,342,418},
    {0,4,8,12,16,20,24,30,36,42,50,60,72,88,106,128,156,190,230,276,330,384}
};
int B8Short[3][13] = {
    {0,4,8,12,16,22,30,42,58,78,104,138,180},
    {0,4,8,12,16,22,30,40,52,66,84,106,136},
    {0,4,8,12,16,22,28,38,50,64,80,100,126}
};
/*=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.
 TABLAS LAYER 2
 =.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.
// pg. 133,
// D.3: Calculation partition tables
// D.3(a,b,c) índice, wlow, whigh, bval, minval, TMN
// D.3.a -- fs = 32 kHz
float D_3_32k[6][49] = {

{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,
40,41,42,43,44,45,46,47,48},

{1,2,5,8,11,14,17,20,23,26,29,32,35,38,42,46,50,54,58,62,66,71,76,82,88,94,100,107,114,121,130,139,149,160,171,
184,197,211,226,241,259,280,301,327,355,383,421,459,497},

{1,4,7,10,13,16,19,22,25,28,31,34,37,41,45,49,53,57,61,65,70,75,81,87,93,99,106,113,120,129,138,148,159,170,183
,196,210,225,240,258,279,300,326,354,382,420,45,496,513},

{0.00,0.63,1.56,2.50,3.44,4.34,5.17,5.94,6.63,7.28,7.90,8.50,9.06,9.65,10.28,10.87,11.41,11.92,12.39,12.83,13.2
9,13.78,14.27,14.76,15.22,15.63,16.06,16.47,16.86,17.25,17.65,18.05,18.42,18.81,19.18,19.55,19.93,20.29,20.65,2
1.02,21.38,21.74,22.10,22.44,22.79,23.14,23.49,23.83,24.07},

{0.0,0.0,20.0,20.0,20.0,20.0,20.0,20.0,17.0,15.0,15.0,10.0,7.0,7.0,4.4,4.4,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,
4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5},

{24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.8,25.4,25.9,26.4,26.9,27.3,27.8,28.3,
28.8,29.3,29.7,30.1,30.6,31.0,31.4,31.8,32.2,32.5,32.9,33.3,33.7,34.1,34.4,34.8,35.2,35.5,35.9,36.2,36.6,36.9,3
7.3,37.6,38.0,38.3,38.6}
};
// D.3.b -- fs = 44.1 kHz
float D_3_44k[6][57] = {

{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,
40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56},

{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,20,23,26,29,32,35,38,41,45,49,53,57,61,65,70,75,81,87,94,101,109,117
,125,135,145,156,167,178,193,208,223,244,265,287,315,343,372,402,432,470},

{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,19,22,25,28,31,34,37,40,44,48,52,56,60,64,69,74,80,86,93,100,108,116,12
4,134,144,155,166,177,192,207,222,243,264,286,314,342,371,401,431,469,513},

{0.00,0.43,0.86,1.29,1.72,2.15,2.58,3.01,3.45,3.88,4.28,4.67,5.06,5.42,5.77,6.11,6.73,7.61,8.44,9.21,9.88,10.51
,11.11,11.65,12.24,12.85,13.41,13.94,14.42,14.86,15.32,15.79,16.26,16.73,17.19,17.62,18.05,18.45,18.83,19.21,19
.60,20.00,20.38,20.74,21.12,21.48,21.84,22.20,22.56,22.91,23.26,23.60,23.96,24.30,24.65,25.00,25.33},

{0.0,0.0,0.0,20.0,20.0,20.0,20.0,20.0,20.0,20.0,20.0,20.0,20.0,20.0,20.0,17.0,17.0,15.0,10.0,7.0,7.0,4.4,4.5,4.
5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4
.5,4.5,4.5,4.5,4.5,3.5},

{24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,25.0,
25.6,26.2,26.7,27.4,27.9,28.4,28.9,29.4,29.8,30.3,30.8,31.2,31.7,32.1,32.5,32.9,33.3,33.7,34.1,34.5,34.9,35.2,3
5.6,36.0,36.3,36.7,37.1,37.4,37.8,38.1,38.4,38.8,39.1,39.5,39.8}
};
// D.3.c -- fs = 48 kHz
float D_3_48k[6][58] = {

{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,
40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57},

{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,21,24,27,30,33,36,39,42,46,50,54,58,63,68,73,78,84,90,96,104,112,
121,130,139,150,161,174,188,202,220,239,258,284,310,336,364,392,424,466,508},

{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,20,23,26,29,32,35,38,41,45,49,53,57,62,67,72,77,83,89,95,103,111,120
,129,138,149,160,173,187,201,219,238,257,283,309,335,363,391,423,465,507,513},

{0.00,0.47,0.94,1.41,1.88,2.34,2.81,3.28,3.75,4.20,4.63,5.05,5.44,5.83,6.19,6.52,6.86,7.49,8.40,9.24,9.97,10.65
,11.28,11.86,12.39,12.96,13.56,14.12,14.62,15.14,15.67,16.15,16.58,17.02,17.44,17.84,18.24,18.66,19.07,19.47,19
.85,20.23,20.63,21.02,21.40,21.76,22.12,22.47,22.83,23.18,23.53,23.88,24.23,24.58,24.93,25.27,25.61,25.81},

{0.0,0.0,0.0,20.0,20.0,20.0,20.0,20.0,20.0,20.0,20.0,20.0,20.0,20.0,20.0,17.0,17.0,15.0,10.0,7.0,7.0,4.4,4.5,4.
```

```
5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4
.5,4.5,4.5,4.5,4.5,3.5,3.5},

{24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,24.5,25.1,
25.8,26.4,26.9,27.5,28.1,28.6,29.1,30.2,30.7,31.1,31.5,31.9,32.3,32.7,33.2,33.6,34.0,34.3,34.7,35.1,35.5,35.9,3
6.3,36.6,37.0,37.3,37.7,38.0,38.4,38.7,39.1,39.1,39.4,39.8,40.1,40.3}
};
//=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.
// pg. 136,
// D.4: Absolute thrshold tables
// D.4(a,b,c) índicesLOW, índicesHIGH, absthr
// D.4.a -- fs = 32 kHz
float D_4_32k[3][132] = {

{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40
,41,42,43,44,45,46,47,48,49,51,53,55,57,59,61,63,65,67,69,71,73,75,77,79,81,83,85,87,89,91,93,95,97,101,105,109
,113,117,121,125,129,133,137,141,145,149,153,157,161,165,169,173,177,181,185,189,193,201,209,217,225,233,241,24
9,257,265,273,281,289,297,305,313,321,329,337,345,353,361,369,377,385,393,401,409,417,425,433,441,449,457,465,4
73},

{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40
,41,42,43,44,45,46,47,48,50,52,54,56,57,60,62,64,66,68,70,72,74,76,78,80,82,84,86,88,90,92,94,96,100,104,108,11
2,116,120,124,128,132,136,140,144,148,152,156,160,164,168,172,176,180,184,188,192,200,208,216,224,232,240,248,2
56,264,272,280,288,296,304,312,320,328,336,344,352,360,368,376,384,392,400,408,416,424,432,440,448,456,464,472,
480},

{58.23,33.44,24.17,19.20,16.05,13.87,12.26,11.01,10.01,9.20,8.52,7.94,7.44,7.00,6.62,6.28,5.97,5.70,5.44,5.21,5
.00,4.80,4.62,4.45,4.29,4.14,4.00,3.86,3.73,3.61,3.49,3.37,3.26,3.15,3.04,2.93,2.83,2.73,2.63,2.53,2.42,2.32,2.
22,2.12,2.02,1.92,1.81,1.71,1.49,1.27,1.04,0.80,0.55,0.29,0.02,-0.25,-0.54,-0.83,-1.12,-1.43,-1.73,-2.04,-
2.34,-2.64,-2.93,-3.22,-3.49,-3.74,-3.98,-4.20,-4.40,-4.57,-4.82,-4.96,-4.97,-4.86,-4.63,-4.29,-3.87,-3.39,-
2.86,-2.31,-1.77,-1.24,-0.74,-
0.29,0.12,0.48,0.79,1.06,1.29,1.49,1.66,1.81,1.95,2.08,2.33,2.59,2.86,3.17,3.51,3.89,4.31,4.79,5.31,5.88,6.50,7
.19,7.93,8.75,9.63,10.58,11.60,12.71,13.90,15.18,16.54,18.01,19.57,21.23,23.01,24.90,26.90,29.03,31.28,33.67,36
.19,38.86,41.67,44.63,47.76,51.03}
};
// D.4.a -- fs = 44.1 kHz
float D_4_44k[3][130] = {

{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40
,41,42,43,44,45,46,47,48,49,51,53,55,57,59,61,63,65,67,69,71,73,75,77,79,81,83,85,87,89,91,93,95,97,101,105,109
,113,117,121,125,129,133,137,141,145,149,153,157,161,165,169,173,177,181,185,189,193,201,209,217,225,233,241,24
9,257,265,273,281,289,297,305,313,321,329,337,345,353,361,369,377,385,393,401,409,417,425,433,441,449,457},

{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40
,41,42,43,44,45,46,47,48,50,52,54,56,58,60,62,64,66,68,70,72,74,76,78,80,82,84,86,88,90,92,94,96,100,104,108,11
2,116,120,124,128,132,136,140,144,148,152,156,160,164,168,172,176,180,184,188,192,200,208,216,224,232,240,248,2
56,264,272,280,288,296,304,312,320,328,336,344,352,360,368,376,384,392,400,408,416,424,432,440,448,456,464},

{45.05,25.87,18.70,14.85,12.41,10.72,9.47,8.50,7.73,7.10,6.56,6.11,5.72,5.37,5.07,4.79,4.55,4.32,4.11,3.92,3.74
,3.57,3.40,3.25,3.10,2.95,2.81,2.67,2.53,2.39,2.25,2.11,1.97,1.83,1.68,1.53,1.38,1.23,1.07,0.90,0.74,0.56,0.39,
0.21,0.02,-0.17,-0.36,-0.56,-0.96,-1.37,-1.79,-2.21,-2.63,-3.03,-3.41,-3.77,-4.09,-4.37,-4.60,-4.78,-4.91,-
4.97,-4.98,-4.92,-4.81,-4.65,-4.43,-4.17,-3.87,-3.54,-3.19,-2.82,-2.06,-1.33,-0.64,-
0.04,0.47,0.89,1.23,1.51,1.74,1.93,2.11,2.28,2.45,2.63,2.82,3.03,3.25,3.49,3.74,4.02,4.32,4.64,4.98,5.35,6.15,7
.07,8.10,9.25,10.54,11.97,13.56,15.30,17.23,19.33,21.64,24.15,26.88,29.84,33.04,36.51,40.24,40.26,48.58,53.21,5
8.17,63.48,69.13,69.13,69.13,69.13,69.13,69.13,69.13,69.13,69.13,69.13,69.13,69.13}
};
// D.4.a -- fs = 48 kHz
float D_4_48k[3][126] = {

{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40
,41,42,43,44,45,46,47,48,49,51,53,55,57,59,61,63,65,67,69,71,73,75,77,79,81,83,85,87,89,91,93,95,97,101,105,109
,113,117,121,125,129,133,137,141,145,149,153,157,161,165,169,173,177,181,185,189,193,201,209,217,225,233,241,24
9,257,265,273,281,289,297,305,313,321,329,333,341,349,357,365,373,381,389,397,405,413,421},

{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40
,41,42,43,44,45,46,47,48,50,52,54,56,58,60,62,64,66,68,70,72,74,76,78,80,82,84,86,88,90,92,94,96,100,104,108,11
2,116,120,124,128,132,136,140,144,148,152,156,160,164,168,172,176,180,184,188,192,200,208,216,224,232,240,248,2
56,264,272,280,288,296,304,312,320,328,332,340,348,356,364,372,380,388,396,404,412,420,428},

{42.10,24.17,17.47,13.87,11.60,10.01,8.84,7.94,7.22,6.62,6.12,5.70,5.33,5.00,4.71,4.45,4.21,4.00,3.79,3.61,3.43
,3.26,3.09,2.93,2.78,2.63,2.47,2.32,2.17,2.02,1.86,1.71,1.55,1.38,1.21,1.04,0.86,0.67,0.49,0.29,0.09,-0.11,-
0.32,-0.54,-0.75,-0.97,-1.20,-1.43,-1.88,-2.34,-2.79,-3.22,-3.62,-3.98,-4.30,-4.57,-4.77,-4.91,-4.98,-4.97,-
4.90,-4.76,-4.55,-4.29,-3.99,-3.64,-3.26,-2.86,-2.45,-2.04,-1.63,-1.24,-
0.51,0.12,0.64,1.06,1.39,1.66,1.88,2.08,2.27,2.46,2.65,2.86,3.09,3.33,3.60,3.89,4.20,4.54,4.91,5.31,5.73,6.18,6
.67,7.19,8.39,9.63,11.08,12.71,14.53,16.54,18.77,21.23,23.94,26.90,30.14,33.67,37.51,41.67,46.17,51.04,56.29,61
.94,68.00,68.00,68.00,68.00,68.00,68.00,68.00,68.00,68.00,68.00,68.00,68.00}
};
/* =.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.
 pg. 139,
 D.5. Layer 1 y Layer 2 tabla de particiones
 D.5 index, ωlow_(n+1) ωhigh_n, width_n
*/
int D_5[3][33] = {
    {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32},
```

```
{1,17,33,49,65,81,97,113,129,145,161,177,193,209,225,241,257,273,289,305,321,337,353,369,385,401,417,433,449,46
5,481,497,513},
    {0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
};
/* =.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.
 pg. 46,
 B.2 Layer 2 tablas de asignación de bits (Posibles niveles de cuantización por Subbanda)
 B.2 columnas: sb, nbal index[16](niveles de cuantización para cada sb)
 */
int B_2_a[18][32]{
    {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31},
    {4,4,4,4,4,4,4,4,4,4,4,3,3,3,3,3,3,3,3,3,3,3,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,0,0,0,0,0,0,0},
    {7,7,7,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,0,0,0,0,0},
    {15,15,15,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,65535,65535,65535,65535,0,0,0,0,0},
    {31,31,31,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,0,0,0,0,0,0,0,0,0},
    {63,63,63,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,0,0,0,0,0,0,0,0,0},
    {127,127,127,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,0,0,0,0,0,0,0,0,0},

{255,255,255,63,63,63,63,63,63,63,65535,65535,65535,65535,65535,65535,65535,65535,65535,65535,65535,65535,0,
0,0,0,0,0,0,0,0,0},
    {511,511,511,127,127,127,127,127,127,127,127,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {1023,1023,1023,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {2047,2047,2047,511,511,511,511,511,511,511,511,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {4095,4095,4095,1023,1023,1023,1023,1023,1023,1023,1023,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {8191,8191,8191,2047,2047,2047,2047,2047,2047,2047,2047,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {16383,16383,16383,4095,4095,4095,4095,4095,4095,4095,4095,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {32767,32767,32767,8191,8191,8191,8191,8191,8191,8191,8191,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{65535,65535,65535,65535,65535,65535,65535,65535,65535,65535,65535,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
};
int B_2_b[18][32]{
    {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31},
    {4,4,4,4,4,4,4,4,4,4,4,3,3,3,3,3,3,3,3,3,3,3,2,2,2,2,2,2,2,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,0,0},
    {7,7,7,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,0,0},
    {15,15,15,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,65535,65535,65535,65535,65535,65535,65535,0,0},
    {31,31,31,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,0,0,0,0,0,0,0,0,0},
    {63,63,63,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,0,0,0,0,0,0,0,0,0},
    {127,127,127,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,31,0,0,0,0,0,0,0,0,0},

{255,255,255,63,63,63,63,63,63,63,65535,65535,65535,65535,65535,65535,65535,65535,65535,65535,65535,65535,0,
0,0,0,0,0,0,0,0,0},
    {511,511,511,127,127,127,127,127,127,127,127,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {1023,1023,1023,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {2047,2047,2047,511,511,511,511,511,511,511,511,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {4095,4095,4095,1023,1023,1023,1023,1023,1023,1023,1023,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {8191,8191,8191,2047,2047,2047,2047,2047,2047,2047,2047,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {16383,16383,16383,4095,4095,4095,4095,4095,4095,4095,4095,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {32767,32767,32767,8191,8191,8191,8191,8191,8191,8191,8191,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{65535,65535,65535,65535,65535,65535,65535,65535,65535,65535,65535,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
};
int B_2_c[18][32]{
    {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31},
    {4,4,3,3,3,3,3,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {3,3,3,3,3,3,3,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {5,5,5,5,5,5,5,5,5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {9,9,9,9,9,9,9,9,9,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {15,15,15,15,15,15,15,15,15,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {31,31,31,31,31,31,31,31,31,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {63,63,63,63,63,63,63,63,63,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {127,127,127,127,127,127,127,127,127,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {511,511,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {1023,1023,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {2047,2047,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {4095,4095,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {8191,8191,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {16383,16383,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {32767,32767,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
};
int B_2_d[18][32]{
    {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31},
    {4,4,3,3,3,3,3,3,3,3,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {3,3,3,3,3,3,3,3,3,3,3,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {5,5,5,5,5,5,5,5,5,5,5,5,5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {9,9,9,9,9,9,9,9,9,9,9,9,9,9,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {31,31,31,31,31,31,31,31,31,31,31,31,31,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
```

```
    {63,63,63,63,63,63,63,63,63,63,63,63,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {127,127,127,127,127,127,127,127,127,127,127,127,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {255,255,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {511,511,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {1023,1023,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {2047,2047,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {4095,4095,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {8191,8191,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {16383,16383,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
    {32767,32767,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
};
/* =.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.
 pg. 45,
 B.1 Layer 2 scalefactors
 */

float II_scalefactors[63] =
{2.0,1.5874010519682,1.25992104989487,1.0,0.7937005259841,0.62996052494744,0.5,0.39685026299205,0.3149802624737
2,0.25,0.19842513149602,0.15749013123686,0.125,0.09921256574801,0.07874506561843,0.0625,0.04960628287401,0.0393
7253280921,0.03125,0.02480314143700,0.01968626640461,0.015625,0.0124015707185,
0.0098431332023,0.0078125,0.00620078535925,0.00492156660115,0.00390625,0.00310039267963,0.00246078330058,0.0019
53125,0.0155019633981,0.00123039165029,0.0009765625,0.00077509816991,0.000615195822514,0.00048828125,0.00038754
908495,0.00030759791257,0.00024414062500,0.00019377454248,0.00015379895629,0.00012207031250,0.00009688727124,0.
00007689947814,0.00006103515625,0.00004844363562,0.00003844973907,0.00003051757813,0.00002422181781,0.000019224
86954,0.00001525878906,0.00001211090890,0.00000961243477,0.00000762939453,0.00000605545445,0.00000480621738,0.0
0000381469727,0.00000302772723,0.00000240310869,0.00000190734863,0.00000151386361,0.00000120155435};
/* =.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.
 pg. 76,
 C.5 Layer 2 Signal to noise ratios
 */
float SNR[17] =
{7.0,11.0,16.0,20.84,25.28,31.59,37.75,43.84,49.89,55.93,61.96,67.98,74.01,80.03,86.05,92.01,98.21};
/* =.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.
 pg. 77,
 C.6 Layer 2 quantization coefficients
 */
static float II_A[17] = {
    0.750000000, 0.625000000, 0.875000000, 0.562500000, 0.937500000,
    0.968750000, 0.984375000, 0.992187500, 0.996093750, 0.998046875,
    0.999023438, 0.999511719, 0.999755859, 0.999877930, 0.999938965,
    0.999969482, 0.999984741 };

static float II_B[17] = {
    -0.250000000, -0.375000000, -0.125000000, -0.437500000, -0.062500000,
    -0.031250000, -0.015625000, -0.007812500, -0.003906250, -0.001953125,
    -0.000976563, -0.000488281, -0.000244141, -0.000122070, -0.000061035,
    -0.000030518, -0.000015259 };
// =.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.=.
// coeficientes de análisis PQMF:
float c[512] = {0,-4.77000000000000e-07,-4.77000000000000e-07,-4.77000000000000e-07,-4.77000000000000e-07,-
4.77000000000000e-07,-4.77000000000000e-07,-9.54000000000000e-07,-9.54000000000000e-07,-9.54000000000000e-07,-
9.54000000000000e-07,-1.43100000000000e-06,-1.43100000000000e-06,-1.90700000000000e-06,-1.90700000000000e-06,-
2.38400000000000e-06,-2.38400000000000e-06,-2.86100000000000e-06,-3.33800000000000e-06,-3.33800000000000e-06,-
3.81500000000000e-06,-4.29200000000000e-06,-4.76800000000000e-06,-5.24500000000000e-06,-6.19900000000000e-06,-
6.67600000000000e-06,-7.62900000000000e-06,-8.10600000000000e-06,-9.06000000000000e-06,-1.00140000000000e-05,-
1.14440000000000e-05,-1.23980000000000e-05,-1.38280000000000e-05,-1.47820000000000e-05,-1.66890000000000e-05,-
1.81200000000000e-05,-1.95500000000000e-05,-2.14580000000000e-05,-2.33650000000000e-05,-2.52720000000000e-05,-
2.76570000000000e-05,-3.00410000000000e-05,-3.24250000000000e-05,-3.48090000000000e-05,-3.76700000000000e-05,-
4.05310000000000e-05,-4.33920000000000e-05,-4.62530000000000e-05,-4.95910000000000e-05,-5.29290000000000e-05,-
5.57900000000000e-05,-5.96050000000000e-05,-6.29430000000000e-05,-6.62800000000000e-05,-7.00950000000000e-05,-
7.34330000000000e-05,-7.67710000000000e-05,-8.05850000000000e-05,-8.39230000000000e-05,-8.72610000000000e-05,-
9.05990000000000e-05,-9.34600000000000e-05,-9.63210000000000e-05,9.91820000000000e-
05,0.000101566000000000,0.000103951000000000,0.000105858000000000,0.000107288000000000,0.000108242000000000,0.0
00108719000000000,0.000108719000000000,0.000108242000000000,0.000106812000000000,0.000105381000000000,0.0001025
20000000000,9.91820000000000e-05,9.53670000000000e-05,9.01220000000000e-05,8.44000000000000e-
05,7.77240000000000e-05,6.96180000000000e-05,6.05580000000000e-05,5.05450000000000e-05,3.95770000000000e-
05,2.71800000000000e-05,1.38280000000000e-05,-9.54000000000000e-07,-1.71660000000000e-05,-3.43320000000000e-
05,-5.29290000000000e-05,-7.29560000000000e-05,-9.39370000000000e-05,-0.000116348000000000,-
0.000140190000000000,-0.000165462000000000,-0.000191212000000000,-0.000218868000000000,-0.000247478000000000,-
0.000277042000000000,-0.000307560000000000,-0.000339031000000000,-0.000371456000000000,-0.000404358000000000,-
0.000438213000000000,-0.000472546000000000,-0.000507355000000000,-0.000542164000000000,-0.000576973000000000,-
0.000611782000000000,-0.000646591000000000,-0.000680923000000000,-0.000714302000000000,-0.000747204000000000,-
0.000779152000000000,-0.000809669000000000,-0.000838757000000000,-0.000866413000000000,-0.000891685000000000,-
0.000915051000000000,-0.000935555000000000,-0.000954151000000000,-0.000968933000000000,-0.000980854000000000,-
0.000989437000000000,-0.000994205000000000,-0.000995159000000000,-
0.000991821000000000,0.000983715000000000,0.000971317000000000,0.000953674000000000,0.000930786000000000,0.0009
02653000000000,0.000868797000000000,0.000829220000000000,0.000783920000000000,0.000731945000000000,0.0006742480
00000000,0.000610352000000000,0.000539303000000000,0.000462532000000000,0.000378609000000000,0.0002884860000000
00,0.000191689000000000,8.82150000000000e-05,-2.14580000000000e-05,-0.000137329000000000,-
0.000259876000000000,-0.000388145000000000,-0.000522137000000000,-0.000661850000000000,-0.000806808000000000,-
0.000956535000000000,-0.00111103100000000,-0.00126981700000000,-0.00143241900000000,-0.00159788100000000,-
0.00176668200000000,-0.00193738900000000,-0.00211000400000000,-0.00228309600000000,-0.00245714200000000,-
0.00263071100000000,-0.00280332600000000,-0.00297403300000000,-0.00314188000000000,-0.00330686600000000,-
0.00346708300000000,-0.00362253200000000,-0.00377178200000000,-0.00391435600000000,-0.00404882400000000,-
0.00417470900000000,-0.00429058100000000,-0.00439596200000000,-0.00448989900000000,-0.00457048400000000,-
```

```
0.00463819500000000,-0.00469112400000000,-0.00472831700000000,-0.00474882100000000,-0.00475215900000000,-
0.00473737700000000,-0.00470304500000000,-0.00464916200000000,-0.00457382200000000,-0.00447702400000000,-
0.00435781500000000,-0.00421524000000000,-0.00404930100000000,-0.00385856600000000,-
0.00364303600000000,0.00340175600000000,0.00313472700000000,0.00284147300000000,0.00252151500000000,0.002174854
00000000,0.00180053700000000,0.00139951700000000,0.00097131700000000,0.00051593800000000,3.33790000000000e-
05,-0.00047588300000000,-0.00101184800000000,-0.00157356300000000,-0.00216150300000000,-0.00277423900000000,-
0.00341129300000000,-0.00407218900000000,-0.00475645100000000,-0.00546217000000000,-0.00618934600000000,-
0.00693702700000000,-0.00770330400000000,-0.00848722500000000,-0.00928783400000000,-0.01013703000000000,-
0.01093339900000000,-0.01175017000000000,-0.01262760200000000,-0.01348924600000000,-0.01435852100000000,-
0.01523351700000000,-0.01611280400000000,-0.01699447600000000,-0.01787614800000000,-
0.01963424700000000,-0.02050685900000000,-0.02137231800000000,-0.02222871800000000,-0.02307418500000000,-
0.02390718500000000,-0.02472543700000000,-0.02552700000000000,-0.02631092100000000,-0.02707386000000000,-
0.02781534200000000,-0.02853298200000000,-0.02922487300000000,-0.02989006000000000,-0.03052663800000000,-
0.03113269800000000,-0.03171068100000000,-0.03227480200000000,-0.03275489800000000,-0.03325536000000000,-
0.03365993500000000,-0.03405571000000000,-0.03436599350000000,-0.03322553600000000,-0.03275489800000000,-
0.03227480200000000,-0.03171068100000000,-0.03113269800000000,-0.03052663800000000,-0.02989006000000000,-
0.02922487300000000,-0.02853298200000000,-0.02781534200000000,-0.02707386000000000,-0.02631092100000000,-
0.02552700000000000,-0.02472543700000000,-0.02390718500000000,-0.02307418500000000,-0.02222871800000000,-
0.02137231800000000,-0.02050685900000000,-0.01963424700000000,-0.01876148000000000,-0.01699447600000000,-
0.01611280400000000,-0.01523351700000000,-0.01435852100000000,-0.01348924600000000,-0.01262760200000000,-
0.01175017000000000,-0.01093339900000000,-0.01013703000000000,-0.00928783400000000,-0.00848722500000000,-0.00770330400000000,-0.00693702700000000,-0.00618934600000000,-0.00547621700000000,-0.00475645100000000,-0.00404721890000000,-0.00341129300000000,-0.00277423900000000,-0.00215615030000000,-0.00157356300000000,-0.00101184800000000,-0.00051593800000000,-3.33790000000000e-05,-0.00097131700000000,-0.00139951700000000,-
0.00180053700000000,-0.00217485400000000,-
0.00252151500000000,0.00284147300000000,0.00313472700000000,0.00340175600000000,0.00364303600000000,0.003858566
00000000,0.00404930100000000,0.00421524000000000,0.00435781500000000,0.00447702400000000,0.00457382200000000,0.
00464916200000000,0.00470304500000000,0.00473737700000000,0.00475215900000000,0.00474882100000000,0.00472831700
000000,0.00469112400000000,0.00463819500000000,0.00457048400000000,0.00448989900000000,0.00439596200000000,0.00
429058100000000,0.00417470900000000,0.00404882400000000,0.00391435600000000,0.00377178200000000,0.00362255320000
0000,0.00346708300000000,0.00330686600000000,0.00314188000000000,0.00297403300000000,0.00280332600000000,0.00262
307110000000,0.00245714200000000,0.00228309600000000,0.00211000400000000,0.00193738900000000,0.00176668200000000
,0.00159788100000000,0.00143241900000000,0.00126981700000000,0.00111103100000000,0.00095635000000000,0.00080
680800000000,0.00066185000000000,0.00052213700000000,0.00038814500000000,0.00025987600000000,0.00013732900
0000000,2.14580000000000e-05,-8.82150000000000e-05,-0.00019168900000000,-0.00028848600000000,-
0.00037860900000000,-0.00046253200000000,-0.00053930300000000,-0.00061035200000000,-0.00067424800000000,-
0.00073194500000000,-0.00078392000000000,-0.00082922000000000,-0.00086879700000000,-0.00090265300000000,-
0.00093078600000000,0.00095367400000000,0.00097131700000000,0.00098371500000000,0.00099182100000000,0.0009
95159000000000,0.00099420500000000,0.00098943700000000,0.00098085400000000,0.00096893300000000,0.0009541510
00000000,0.00093555500000000,0.00091505100000000,0.00089168500000000,0.00086641300000000,0.0008387570000000
00,0.00080966900000000,0.00077915200000000,0.00074724000000000,0.00071430200000000,0.00068092300000000,0.0
0064659100000000,0.00061178200000000,0.00057697300000000,0.00054216400000000,0.00050735500000000,0.0004725
46000000000,0.00043821300000000,0.00040435800000000,0.00037145600000000,0.00033903100000000,0.0003075600000
00000,0.00027704200000000,0.00024747800000000,0.00021886800000000,0.00019121200000000,0.00016546200000000,
0.00014019000000000,0.00011634800000000,9.39370000000000e-05,7.29560000000000e-05,5.29290000000000e-
05,3.43320000000000e-05,1.71660000000000e-05,9.54000000000000e-07,-1.38280000000000e-05,-2.71800000000000e-05,-
3.95770000000000e-05,-5.05450000000000e-05,-6.05580000000000e-05,-6.96180000000000e-05,-7.77240000000000e-05,-
8.44000000000000e-05,-9.01220000000000e-05,-9.53670000000000e-05,-9.91820000000000e-05,-0.00010252000000000,-
0.00010538100000000,-0.00010681200000000,-0.00010824200000000,-0.00010871900000000,-0.00010871900000000,-
0.00010824200000000,-0.00010728800000000,-
0.00010585800000000,0.00010395100000000,0.00010156600000000,9.91820000000000e-05,9.63210000000000e-
05,9.34600000000000e-05,9.05990000000000e-05,8.72610000000000e-05,8.39230000000000e-05,8.05850000000000e-
05,7.67710000000000e-05,7.34330000000000e-05,7.00950000000000e-05,6.62800000000000e-05,6.29430000000000e-
05,5.96050000000000e-05,5.57900000000000e-05,5.29290000000000e-05,4.95910000000000e-05,4.62530000000000e-
05,4.33920000000000e-05,4.05310000000000e-05,3.76700000000000e-05,3.48090000000000e-05,3.24250000000000e-
05,3.00410000000000e-05,2.76570000000000e-05,2.52720000000000e-05,2.33650000000000e-05,2.14580000000000e-
05,1.95500000000000e-05,1.81200000000000e-05,1.66890000000000e-05,1.47820000000000e-05,1.38280000000000e-
05,1.23980000000000e-05,1.14440000000000e-05,1.00140000000000e-05,9.06000000000000e-06,8.10600000000000e-
06,7.62900000000000e-06,6.67600000000000e-06,6.19900000000000e-06,5.24500000000000e-06,4.76800000000000e-
06,4.29200000000000e-06,3.81500000000000e-06,3.33800000000000e-06,3.33800000000000e-06,2.86100000000000e-
06,2.38400000000000e-06,2.38400000000000e-06,1.90700000000000e-06,1.90700000000000e-06,1.43100000000000e-
06,1.43100000000000e-06,9.54000000000000e-07,9.54000000000000e-07,9.54000000000000e-07,9.54000000000000e-
07,4.77000000000000e-07,4.77000000000000e-07,4.77000000000000e-07,4.77000000000000e-07,4.77000000000000e-
07,4.77000000000000e-07};


// coeficientes de síntesis IPQMF:
float d[512] = {0,-1.52590000000000e-05,-1.52590000000000e-05,-1.52590000000000e-05,-1.52590000000000e-05,-
1.52590000000000e-05,-1.52590000000000e-05,-3.05180000000000e-05,-3.05180000000000e-05,-3.05180000000000e-05,-
3.05180000000000e-05,-4.57760000000000e-05,-4.57760000000000e-05,-6.10350000000000e-05,-6.10350000000000e-05,-
7.62940000000000e-05,-7.62940000000000e-05,-9.15530000000000e-05,-0.00010681200000000,-0.00010681200000000,-
0.00012207000000000,-0.00013732900000000,-0.00015258800000000,-0.00016784700000000,-0.00019836400000000,-
0.00021362300000000,-0.00024414100000000,-0.00025939900000000,-0.00028991700000000,-0.00032043500000000,-
0.00036621100000000,-0.00039672900000000,-0.00044250500000000,-0.00047302200000000,-0.00053405800000000,-
0.00057983400000000,-0.00062561000000000,-0.00068664600000000,-0.00074768100000000,-0.00080871600000000,-
0.00088501000000000,-0.00096130400000000,-0.00103759800000000,-0.00111389200000000,-0.00120544400000000,-
0.00129699700000000,-0.00138855000000000,-0.00148010300000000,-0.00158691400000000,-0.00169372600000000,-
0.00178527800000000,-0.00190734900000000,-0.00201416000000000,-0.00212097200000000,-0.00224304200000000,-
0.00234985400000000,-0.00245666500000000,-0.00257873500000000,-0.00268554700000000,-0.00279235800000000,-
0.00289917000000000,-0.00299072300000000,-0.00308227500000000,-
0.00317382800000000,0.00325012200000000,0.00332641600000000,0.00338745100000000,0.00343322800000000,0.003463745
00000000,0.00347900400000000,0.00347900400000000,0.00346374500000000,0.00341796900000000,0.00337219200000000,0.
```

```
00328064000000000,0.00317382800000000,0.00305175800000000,0.00288391100000000,0.00270080600000000,0.00248718300
000000,0.00222778300000000,0.00193786600000000,0.00161743200000000,0.00126647900000000,0.00086975100000000,0.0
00442505000000000,-3.05180000000000e-05,-0.000549316000000000,-0.00109863300000000,-0.00169372600000000,-
0.00233459500000000,-0.00300598100000000,-0.00372314500000000,-0.00448608400000000,-0.00529480000000000,-
0.00611877400000000,-0.00700378400000000,-0.00791931200000000,-0.00886535600000000,-0.00984191900000000,-
0.0108489990000000,-0.0118865970000000,-0.0129394530000000,-0.0140228270000000,-0.0151214600000000,-
0.0162353520000000,-0.0173492430000000,-0.0184631350000000,-0.0195770260000000,-0.0206909180000000,-
0.0217895510000000,-0.0228576660000000,-0.0239105220000000,-0.0249328610000000,-0.0259094240000000,-
0.0268402100000000,-0.0277252200000000,-0.0285339360000000,-0.0292816160000000,-0.0299377440000000,-
0.0305328370000000,-0.0310058590000000,-0.0313873290000000,-0.0316619870000000,-0.0318145750000000,-
0.0318450930000000,-0.0317382810000000,-
0.0314788820000000,0.0310821530000000,0.0305175780000000,0.0297851560000000,0.0288848880000000,0.02780151400000
00,0.0265350340000000,0.0250854490000000,0.0234222410000000,0.0215759280000000,0.0195312500000000,0.01725769000
00000,0.0148010250000000,0.0121154790000000,0.00923156700000000,0.00613403300000000,0.00282287600000000,-
0.000686646000000000,-0.00439453100000000,-0.00831604000000000,-0.0124206540000000,-0.0167083740000000,-
0.0211791990000000,-0.0258178710000000,-0.0306091310000000,-0.0355529790000000,-0.0406341550000000,-
0.0458374020000000,-0.0511322020000000,-0.0565338130000000,-0.0619964600000000,-0.0675201420000000,-
0.0730590820000000,-0.0786285400000000,-0.0841827390000000,-0.0897064210000000,-0.0951690670000000,-
0.100540161000000,-0.105819702000000,-0.110946655000000,-0.115921021000000,-0.120697021000000,-
0.125259399000000,-0.129562378000000,-0.133590698000000,-0.137298584000000,-0.140670776000000,-
0.143676758000000,-0.146255493000000,-0.148422241000000,-0.150115967000000,-0.151306152000000,-
0.151962280000000,-0.152069092000000,-0.151596069000000,-0.150497437000000,-0.148773193000000,-
0.146362305000000,-0.143264771000000,-0.139450073000000,-0.134887695000000,-0.129577637000000,-
0.123474121000000,-0.116577148000000,-
0.108856201000000,0.100311279000000,0.0909271240000000,0.0806884770000000,0.0695953370000000,0.0576171870000000
,0.0447845460000000,0.0310821530000000,0.0165100100000000,0.00106811500000000,-0.0152282710000000,-
0.0323791500000000,-0.0503540040000000,-0.0691680910000000,-0.0887756350000000,-0.109161377000000,-
0.130310059000000,-0.152206421000000,-0.174789429000000,-0.198059082000000,-0.221984863000000,-
0.246505737000000,-0.271591187000000,-0.297210693000000,-0.323318481000000,-0.349868774000000,-
0.376800537000000,-0.404083252000000,-0.431655884000000,-0.459472656000000,-0.487472534000000,-
0.515609741000000,-0.543823242000000,-0.572036743000000,-0.600219727000000,-0.628295898000000,-
0.656219482000000,-0.683914185000000,-0.711318970000000,-0.738372803000000,-0.765029907000000,-
0.791213989000000,-0.816864014000000,-0.841949463000000,-0.866363525000000,-0.890090942000000,-
0.913055420000000,-0.935195923000000,-0.956481934000000,-0.976852417000000,-0.996246338000000,-
1.01461792000000,-1.03193664600000,-1.04815673800000,-1.06321716300000,-1.07711792000000,-1.08978271500000,-
1.10121154800000,-1.11137390100000,-1.12022399900000,-1.12774658200000,-1.13392639200000,-1.13876342800000,-
1.14221191400000,-
1.14428710900000,1.14498901400000,1.14428710900000,1.14221191400000,1.13876342800000,1.13392639200000,1.1277465
8200000,1.12022399900000,1.11137390100000,1.10121154800000,1.08978271500000,1.07711792000000,1.06321716300000,1
.04815673800000,1.03193664600000,1.01461792000000,0.996246338000000,0.976852417000000,0.956481934000000,0.93519
5923000000,0.913055420000000,0.890090942000000,0.866363525000000,0.841949463000000,0.816864014000000,0.79121398
9000000,0.765029907000000,0.738372803000000,0.711318970000000,0.683914185000000,0.656219482000000,0.62829589800
0000,0.600219727000000,0.572036743000000,0.543823242000000,0.515609741000000,0.487472534000000,0.45947265600000
0,0.431655884000000,0.404083252000000,0.376800537000000,0.349868774000000,0.323318481000000,0.297210693000000,0
.271591187000000,0.246505737000000,0.221984863000000,0.198059082000000,0.174789429000000,0.152206421000000,0.13
0310059000000,0.109161377000000,0.0887756350000000,0.0691680910000000,0.0503540040000000,0.0323791500000000,0.0
152282710000000,-0.00106811500000000,-0.0165100100000000,-0.0310821530000000,-0.0447845460000000,-
0.0576171870000000,-0.0695953370000000,-0.0806884770000000,-
0.0909271240000000,0.100311279000000,0.108856201000000,0.116577148000000,0.123474121000000,0.129577637000000,0.
134887695000000,0.139450073000000,0.143264771000000,0.146362305000000,0.148773193000000,0.150497437000000,0.151
596069000000,0.152069092000000,0.151962280000000,0.151306152000000,0.150115967000000,0.148422241000000,0.146255
493000000,0.143676758000000,0.140670776000000,0.137298584000000,0.133590698000000,0.129562378000000,0.125259399
000000,0.120697021000000,0.115921021000000,0.110946655000000,0.105819702000000,0.100540161000000,0.095169067000
0000,0.0897064210000000,0.0841827390000000,0.0786285400000000,0.0730590820000000,0.0675201420000000,0.061996460
0000000,0.0565338130000000,0.0511322020000000,0.0458374020000000,0.0406341550000000,0.0355529790000000,0.030609
1310000000,0.0258178710000000,0.0211791990000000,0.0167083740000000,0.0124206540000000,0.00831604000000000,0.00
439453100000000,0.000686646000000000,-0.00282287600000000,-0.00613403300000000,-0.00923156700000000,-
0.0121154790000000,-0.0148010250000000,-0.0172576900000000,-0.0195312500000000,-0.0215759280000000,-
0.0234222410000000,-0.0250854490000000,-0.0265350340000000,-0.0278015140000000,-0.0288848880000000,-
0.0297851560000000,-
0.0305175780000000,0.0310821530000000,0.0314788820000000,0.0317382810000000,0.0318450930000000,0.03181457500000
00,0.0316619870000000,0.0313873290000000,0.0310058590000000,0.0305328370000000,0.0299377440000000,0.02928161600
00000,0.0285339360000000,0.0277252200000000,0.0268402100000000,0.0259094240000000,0.0249328610000000,0.02391052
20000000,0.0228576660000000,0.0217895510000000,0.0206909180000000,0.0195770260000000,0.0184631350000000,0.01734
92430000000,0.0162353520000000,0.0151214600000000,0.0140228270000000,0.0129394530000000,0.0118865970000000,0.01
08489990000000,0.00984191900000000,0.00886535600000000,0.00791931200000000,0.00700378400000000,0.00611877400000
000,0.00529480000000000,0.00448608400000000,0.00372314500000000,0.00300598100000000,0.00233459500000000,0.00169
372600000000,0.00109863300000000,0.000549316000000000,3.05180000000000e-05,-0.000442505000000000,-
0.000869751000000000,-0.00126647900000000,-0.00161743200000000,-0.00193786600000000,-0.00222778300000000,-
0.00248718300000000,-0.00270080600000000,-0.00288391100000000,-0.00305175800000000,-0.00317382800000000,-
0.00328064000000000,-0.00337219200000000,-0.00341796900000000,-0.00346374500000000,-0.00347900400000000,-
0.00347900400000000,-0.00346374500000000,-0.00343322800000000,-0.00338745100000000,-
0.00332641600000000,0.00325012200000000,0.00317382800000000,0.00308227500000000,0.00299072300000000,0.002899170
00000000,0.00279235800000000,0.00268554700000000,0.00257873500000000,0.00245666500000000,0.00234985400000000,0.
00224304200000000,0.00212097200000000,0.00201416000000000,0.00190734900000000,0.00178527800000000,0.00169372600
000000,0.00158691400000000,0.00148010300000000,0.00138855000000000,0.00129699700000000,0.00120544400000000,0.00
111389200000000,0.00103759800000000,0.000961304000000000,0.000885010000000000,0.000808716000000000,0.0007476810
00000000,0.000686646000000000,0.000625610000000000,0.000579834000000000,0.000534058000000000,0.0004730220000000
00,0.000442505000000000,0.000396729000000000,0.000366211000000000,0.000320435000000000,0.000289917000000000,0.0
00259399000000000,0.000244141000000000,0.000213623000000000,0.000198364000000000,0.000167847000000000,0.0001525
88000000000,0.000137329000000000,0.000122070000000000,0.000106812000000000,0.000106812000000000,9.1553000000000
0e-05,7.62940000000000e-05,7.62940000000000e-05,6.10350000000000e-05,6.10350000000000e-05,4.57760000000000e-
05,4.57760000000000e-05,3.05180000000000e-05,3.05180000000000e-05,3.05180000000000e-05,3.05180000000000e-
05,1.52590000000000e-05,1.52590000000000e-05,1.52590000000000e-05,1.52590000000000e-05,1.52590000000000e-
05,1.52590000000000e-05};
```

Anexo 7: Códigos de diseño de señales de prueba en Matlab

Código para generar las señales de **barrido de frecuencias** (Chirp) desde 0 hasta la frecuencia de Nyquist para obtener la Función de Transferencia de los sistemas:

```
fs = 44100;b %o 48000
duracion = 5; %5 segundos
switch fs
    case 44100
        puntos = duracion*(fs+28); %28 solo funciona con 44100
    case 48000
        puntos = duracion*fs;
end
n = linspace(0, duracion, fs*duracion);
fN = fs/2;
%nu = 2^11+2^9+2^8+2^7+2^6+2^6; %44100 kHz
%nu = 2^11+2^9+2^8+2^7; %48000 kHz
windo = blackmanharris(nu)'; %función para ventanear
x_0 = chirp(n, 0, duracion, fN);
x_0 = x_0.*horzcat(windo(1:nu/2),ones(1,length(x_0)-nu),windo(nu/2+1:end));
plot(n, x_0),
title('$Barrido de frecuencia lineal$', 'Interpreter', 'Latex','fontsize',18);
xlabel('$n$ (Indice/muestra)', 'Interpreter', 'Latex','fontsize',22); grid on
xlim([0,length(x_0)])

%Para ingresar a C++:
[sprintf('%d,', x_0(1:end-1)), sprintf('%d', x_0(end))]
save('chirp_44_1_k_5seg.mat','x_0'); %o save('chirp_48k_5seg.mat','x_0')
```

Código para generar los **32 tonos puros** a las frecuencias de las bandas del PQMF para medir **THD** y **SNR** en cada banda.

```
fs = 44100; %o 48000
delta = (fs/2)/32
freqs = 0:delta:fs/2
n = 0:1/fs:1;
%32 tonos puros de 1152 puntos a cada frecuencia “freqs”:
for i = 1:32
    waves(:,i) = sin(2*pi*n(1:1152)*freqs(i));
end
%Para ingresar a C++ la señal deseada:
i = 1;
[sprintf('%d,', waves(1:end-1,i)), sprintf('%d', waves(end,i))]
```

Código para generar la señal **WNG**, con varianza igual a 0.04, y media igual a 0.

```
%------Diseño de la señal WNG------
mu = 0; sigma = 0.2;
fs = 48000; L = 48000;
X = sigma*randn(L,1)+mu;
fileID = fopen('white_noise_48000Hz.txt','w');
fprintf(fileID,[sprintf('%d,', X(1:end-1)), sprintf('%d', X(end))]);
fclose(fileID);
save('Noise_48000.mat','X')
figure, plot(X); grid on, xlim([0 length(X)])
title(['Ruido Blanco Gaussiano : \mu_x=',num2str(mu),',
\sigma^2=',num2str(sigma^2)],'FontName','Times New Roman','FontSize',16)
ylabel('Amplitud','FontName','Times New Roman','FontSize',20,'Interpreter','latex');
xlabel('Muestras','FontName','Times New Roman','FontSize',20, 'Interpreter','latex');
%------Cálculo de su densidad de potencia espectral------
```

Anexo 8: Códigos de algoritmos de cálculo para los resultados de los sistemas en Matlab

Código para calcular y visualizar la **Función de Transferencia** de las señales de salida ("recons") reconstruidas de los sistemas (Layer II y III)

```
% -----FFT recons, y(n)->|Y_out(f)|------
Y_out = fftshift(fft(recons))/length(recons);
fs = 44100; %o 48000
f = linspace(-fs/2, fs/2, length(Y_out));
mag_Y_out = abs(Y_out); dBmag_Y_out_dB = 20.*log10(mag_Y_out);

figure, semilogx(f,smooth(dBmag_Y_out_dB)); grid on, xlim([5,fs/2])
title('$|Y(f)|$ [dBFS]','FontName','Times New Roman','FontSize',20,'Interpreter','latex');
ylabel('$|Y(f)|$ [dBFS]','FontName','Times New Roman','FontSize',20,'Interpreter','latex');
xlabel('f [Hz]','FontName','Times New Roman','FontSize',20, 'Interpreter','latex');


% ----- FFT señal de entrada, x(n)->|V_in(f)|------
V_in = fftshift(fft(x_0))/length(x_0);
if (length(x_0) < length(recons))
    Y_out = Y_out(1:end-(length(recons)-length(x_0)));
end
if (length(x_0) > length(recons))
    V_in = V_in(1:end-(length(x_0)-length(recons)));
end
len_x_0 = length(V_in), len_recons = length(Y_out)

magV_in = abs(V_in); magY_out = abs(Y_out)';

% ----- FFT FUNCIÓN TRANSFERENCIA, |H(f)|------
H = magY_out./magV_in;
mag_H_dB = 20.*log10(H);
figure, semilogx(f,smooth(mag_H_dB)); grid on, xlim([5,fs/2])
title('$|H(f)|$ [dBFS]','FontName','Times New Roman','FontSize',20,'Interpreter','latex');
ylabel('$|H(f)|$ [dBFS]','FontName','Times New Roman','FontSize',20,'Interpreter','latex');
xlabel('f [Hz]','FontName','Times New Roman','FontSize',20, 'Interpreter','latex');

% ----- Cálculo de las diferencias de las magnitudes de ambos Layer------
load('Output_44100Hz_5seg_Layer3.mat','recons') %o 48000
recons_layer3 = recons;
load('Output_44100Hz_5seg_Layer2.mat','recons') %o 48000
recons_layer2 = recons;

fft_layer3 = fftshift(fft(recons_layer3))/length(recons_layer3);
mag_layer3 = abs(fft_layer3);
fft_layer2 = fftshift(fft(recons_layer2))/length(recons_layer2);
mag_layer2 = abs(fft_layer2);

dif = abs(abs(fft_layer2)-abs(fft_layer3));
fs = 44100; %o 48000
f = linspace(-fs/2, fs/2, length(recons_layer3));

semilogx(f,smooth((mag_layer3))), hold on
semilogx(f,smooth((mag_layer2))), hold on
semilogx(f,smooth((dif))), xlim([5,24e3]), grid on
legend('Layer III: $|FFT(y_{3}[n])|=|Y_{3}(f)|$', 'Layer II:$|FFT(y_{2}[n])|=|Y_{2}(f)|$',
'$\Delta = ||Y_{2}(f)|-|Y_{3}(f)||$','FontName','Times New
Roman','FontSize',17,'Interpreter','latex')

ylabel('$|Y(f)|$ normalizado','FontName','Times New Roman','FontSize',20,'Interpreter','latex');
xlabel('f [Hz]','FontName','Times New Roman','FontSize',20, 'Interpreter','latex');
```

Código para el cálculo y visualización del **THD** en dBc y en porcentaje y **SNR** en dBc en cada una de las 32 bandas del PQMF para los Layer II y III

```
fs = 48000; %o 44100
%thd() al no hallar armónicos, hace harmSum = 0, y log10(0/harmPow(1))=infinito
for i = 1:32
    x = tonos_reconstruidos(:,i);
    [pxx,f] = periodogram(x,hann(length(x)),length(x),fs);
    thd_db(i) = thd(pxx,f,10,'psd','aliased');
    if thd_db(i) > 0
       thd_db(i) = -274.7;%valor mínimo de thd hallado en un tono puro por Matlab
    end
    SNR(i) = snr(x, fs);
end
%------Paso del THD a escala lineal y a porcentaje: ------
for i = 1:32
    thd(i) = 100.*(10.^(thd_db(i)./10));
end

save('thd_dbc_layer3_48k.mat','thd_db') %o 44100
save('thd_layer3_48k.mat','thd') %o 44100
save('snr_layer3_48k.mat','SNR') %o 44100

bar(thd_db), grid on
title('$THD$ [dBc] por banda PQMF','FontName','Times New
Roman','FontSize',20,'Interpreter','latex');
ylabel('$THD$ [dBc]','FontName','Times New Roman','FontSize',20,'Interpreter','latex');
xlabel('Banda','FontName','Times New Roman','FontSize',20, 'Interpreter','latex');
figure,
bar(thd), grid on
title('$THD \%$ por banda PQMF','FontName','Times New
Roman','FontSize',20,'Interpreter','latex');
ylabel('$THD \%$ ','FontName','Times New Roman','FontSize',20,'Interpreter','latex');
xlabel('Banda','FontName','Times New Roman','FontSize',20, 'Interpreter','latex');
figure
bar(SNR), grid on
title('$SNR$[dBc] por banda PQMF','FontName','Times New
Roman','FontSize',20,'Interpreter','latex');
ylabel('$SNR$[dBc] ','FontName','Times New Roman','FontSize',20,'Interpreter','latex');
xlabel('Banda','FontName','Times New Roman','FontSize',20, 'Interpreter','latex');
% ------Comparación THD y SNR layer 2 y 3, 44.1 k y 48k Hz------
load('thd_layer3_48k.mat','thd'), load('snr_layer3_48k.mat','SNR')
thd_layer3 = thd;
snr_layer3 = SNR;
load('thd_layer2_48k.mat','thd'), load('snr_layer2_48k.mat','SNR')
thd_layer2 = thd;
snr_layer2 = SNR;
dif_thd = abs(thd_layer3-thd_layer2) %diferencia de porcentajes
bar(dif_thd), set(gca,'yscale','log'), grid on

% tomar los snr como negativos!:
dif_snr = abs(10.^(-snr_layer3/10)-10.^(-snr_layer2/10))
figure, bar(dif_snr), set(gca,'yscale','log'), grid on
```

Código para generar el la señal **WNG**, hallar su función de **Densidad de Potencia Espectral**, histograma (**Curva de Nomalidad**) y su **Autocorrelación** para probar el espectro de potencia a 48kHz.

```
%------Diseño de la señal WNG------
mu = 0; sigma = 0.2;
fs = 48000; L = 48000;
X = sigma*randn(L,1)+mu;
fileID = fopen('white_noise_48000Hz.txt','w');
fprintf(fileID,[sprintf('%d,', X(1:end-1)), sprintf('%d', X(end))]);
fclose(fileID);
save('Noise_48000.mat','X')
figure, plot(X); grid on, xlim([0 length(X)])
title(['Ruido Blanco Gaussiano : \mu_x=',num2str(mu),',
\sigma^2=',num2str(sigma^2)],'FontName','Times New Roman','FontSize',16)
```

```
ylabel('Amplitud','FontName','Times New Roman','FontSize',20,'Interpreter','latex');
xlabel('Muestras','FontName','Times New Roman','FontSize',20, 'Interpreter','latex');
%------Cálculo de su densidad de potencia espectral------
[pxx, w] = periodogram(X);
figure, plot(w/pi,smooth(10*log10(pxx))); grid on
title('Función de densidad espectral','FontName','Times New Roman','FontSize',17);
ylabel('$S_{xx}[F]$ [dBFS]','FontSize',20,'Interpreter','latex');
xlabel('$F$ [ciclo/muestra]','FontSize',20, 'Interpreter','latex');
%------Cálculo de su histograma/PDF ------
n = 500; %numero de bins del histograma
[f,x] = hist(X,n); figure, bar(x,f/trapz(x,f)); hold on;
%------PDF teórico del ruido aleatorio gaussiano------
g = (1/(sqrt(2*pi)*sigma))*exp(-((x-mu).^2)/(2*sigma^2));
plot(x,g,'r','LineWidth',2), grid on, xlim([-1 1]), ylim([0, 2.3])
title('Histograma, Distribución teórica y actual del WGN','FontName','Times New
Roman','FontSize',16);
legend({'Histogr·ma','Distribución teórica'},'FontName','Times New Roman','FontSize',16);
xlabel('Amplitud','FontName','Times New Roman','FontSize',20);
ylabel('$f_{x}(x)$','Interpreter','latex','FontSize',22);
%------Cálculo de la Auto-correlacion ------
Rxx = 1/L*conv(flipud(X),X);
retraso = (-L+1):1:(L-1);
figure, plot(retraso,Rxx), grid on;
title('Función de autocorrelación del WGN','FontName','Times New Roman','FontSize',16);
xlabel('$\tau$','Interpreter','latex','FontSize',22),
ylabel('$R_{xx}(\tau)$','Interpreter','latex','FontSize',20)

fs = 44100;
%-----Visualización Densidad Potencia de señal de SALIDA de los Layer II y III---
load('Noise_recons_Layer2_44100.mat','recons_noise'), X_2 = recons_noise;
subplot(2,2,1), plot((0:length(X_2)-1)/fs,X_2); grid on,
mu = round(mean(X_2),4); sigma = round(std(X_2),3);
title(['Layer II. Sig WGN: \mu_x=',num2str(mu),', \sigma^2 =
',num2str(sigma^2)],'FontName','Times New Roman','FontSize',15)
ylabel('Amplitud','FontName','Times New Roman','FontSize',20,'Interpreter','latex');
xlabel('$n[s]$','FontName','Times New Roman','FontSize',20, 'Interpreter','latex');

[pxx, w] = periodogram(X_2); dBsig = round(10*log10(sigma^2),1);
subplot(2,2,2), plot(w/pi,smooth(10*log10(pxx))); grid on
title(['Layer II. Sig PSD, 10log10(\sigma^2)=',num2str(dBsig),' dBFS'],'FontName','Times New
Roman','FontSize',15);
ylabel('$S_{xx}[F]$ [dBFS]','FontSize',20,'Interpreter','latex');
xlabel('$F$ [ciclo/muestra]','FontSize',20, 'Interpreter','latex');

load('Noise_recons_Layer3_44100.mat','recons_noise'), X_3 = recons_noise;
subplot(2,2,3), plot((0:length(X_3)-1)/fs,X_3); grid on,
mu = round(mean(X_3),4); sigma = round(std(X_3),3);
title(['Layer III. Sig WGN: \mu_x=',num2str(mu),', \sigma^2 =
',num2str(sigma^2)],'FontName','Times New Roman','FontSize',15)
ylabel('Amplitud','FontName','Times New Roman','FontSize',20,'Interpreter','latex');
xlabel('$n[s]$','FontName','Times New Roman','FontSize',20, 'Interpreter','latex');

[pxx, w] = periodogram(X_3); dBsig = round(10*log10(sigma^2),1);
subplot(2,2,4), plot(w/pi,smooth(10*log10(pxx))); grid on
title(['Layer III. Sig PSD, 10log10(\sigma^2)=',num2str(dBsig),' dBFS'],'FontName','Times New
Roman','FontSize',15);
ylabel('$S_{xx}[F]$ [dBFS]','FontSize',20,'Interpreter','latex');
xlabel('$F$ [ciclo/muestra]','FontSize',20, 'Interpreter','latex');

%verificación de la igualdad entre ambas señales
isequal(X_2,X_3)
dif = X_2-X_3; %se hace la resta de lo que introduce el MDCT en el layer 3 respecto al
Layer II
figure, subplot(2,1,1), plot((0:length(dif)-1)/fs,dif); grid on,
mu = round(mean(dif),4); sigma = round(std(dif),3);
title(['abs(dif). Sig WGN: \mu_x=',num2str(mu),', \sigma^2 = ',num2str(sigma^2)],'FontName','Times New
Roman','FontSize',15)
ylabel('Amplitud','FontName','Times New Roman','FontSize',20,'Interpreter','latex');
xlabel('$n[s]$','FontName','Times New Roman','FontSize',20, 'Interpreter','latex');

[pxx, w] = periodogram(dif); dBsig = round(10*log10(sigma^2),1);
subplot(2,1,2), plot(w/pi,smooth(10*log10(pxx))); grid on
title(['Dif. Sig PSD, 10log10(\sigma^2)=',num2str(dBsig),' dBFS'],'FontName','Times New
Roman','FontSize',15);
ylabel('$S_{xx}[F]$ [dBFS]','FontSize',20,'Interpreter','latex');
xlabel('$F$ [ciclo/muestra]','FontSize',20, 'Interpreter','latex');
```

Anexo 9: Gráficos y Tablas extras de Resultados

Figura 7–11: Función de Transferencia y Espectro de Salida de ambos Layer a 44.1 kHz con frecuencia en escala Lineal.

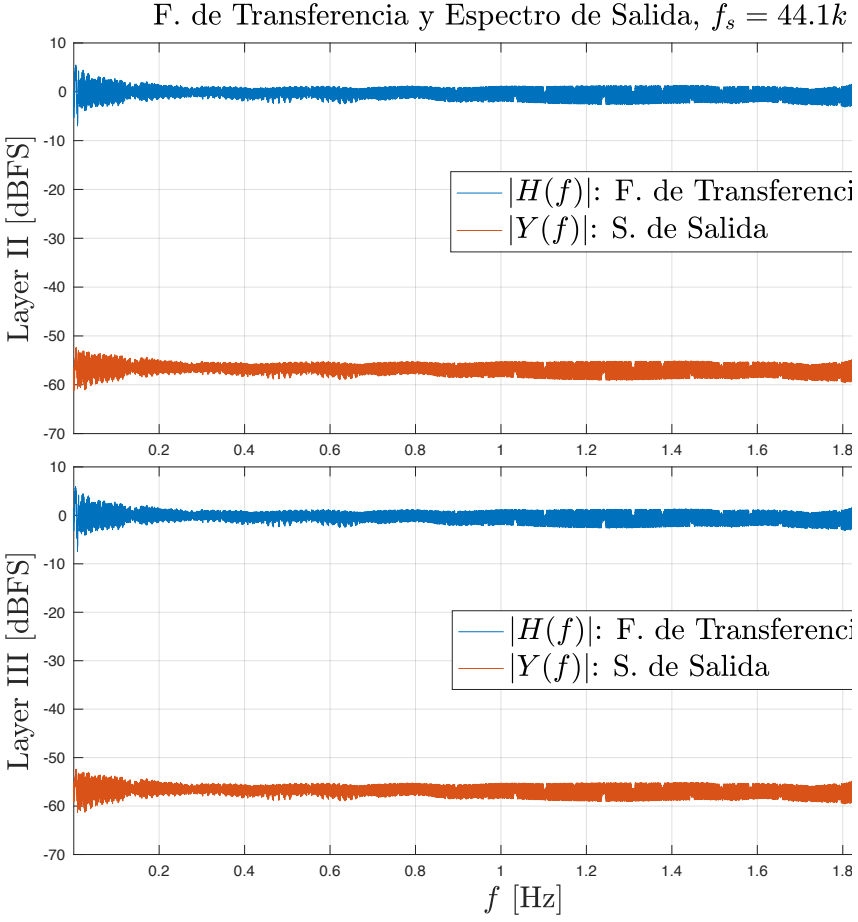


Figura 7–12: Función de Transferencia y Espectro de Salida de ambos Layer a 48 kHz con frecuencia en escala Lineal.

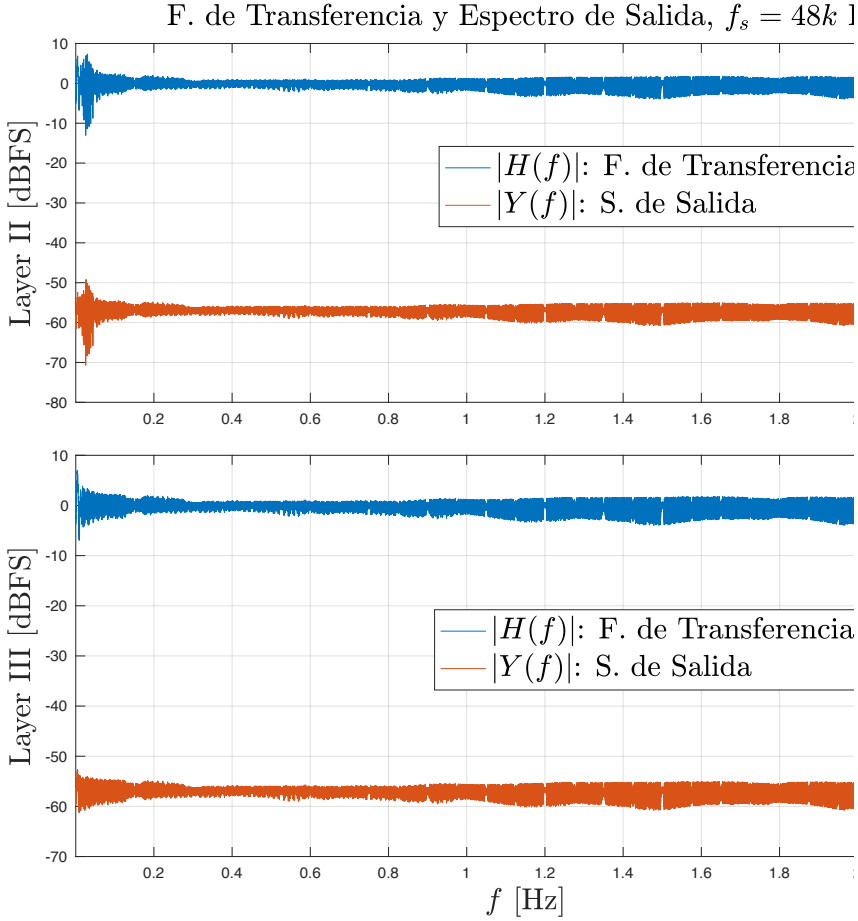


Figura 7–13: Diferencias de las Funciones de Transferencia de ambos Layer a 44.1k y 48k Hz con frecuencia en escala Lineal.

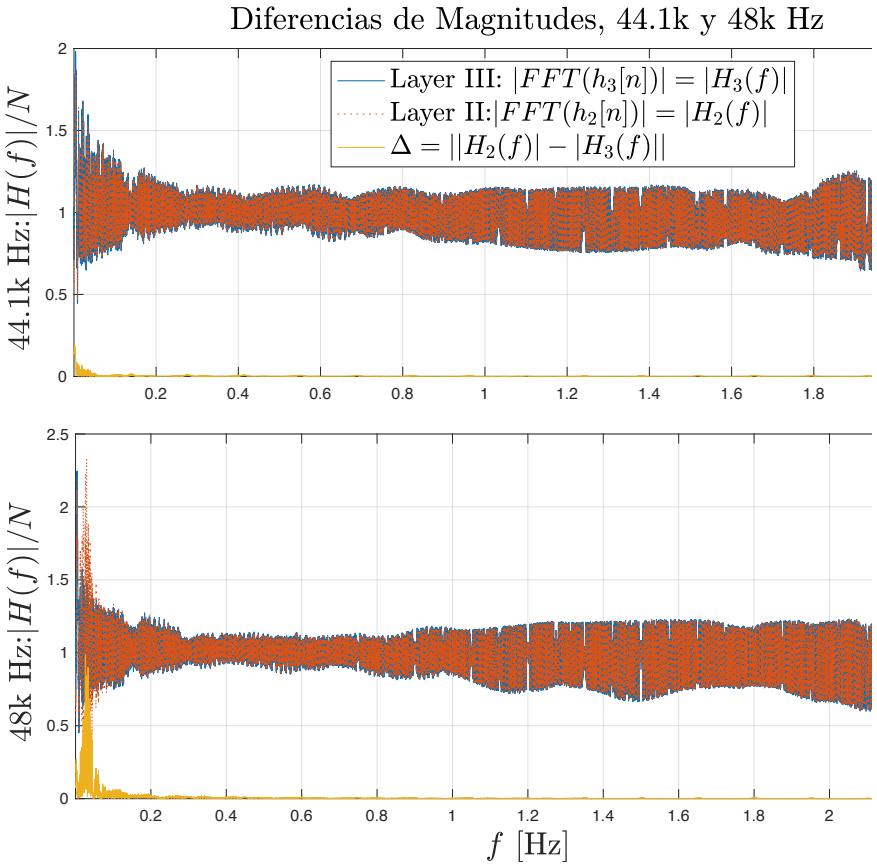


Figura 7–14: Diferencias de las Funciones de Transferencia de ambos Layer a 44.1k y 48k Hz con frecuencia en escala Lineal en dBFS.

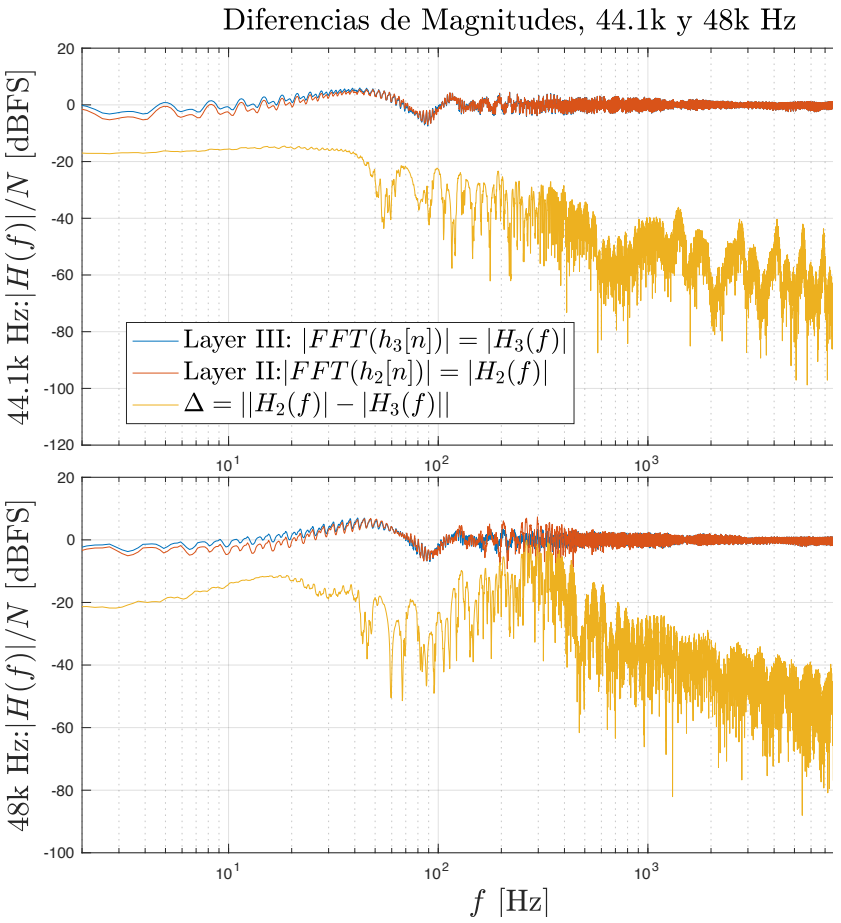


Tabla 7-5: % THD de las 32 bandas de PQMF a 44.1k Hz

THD [%] a fs = 44.1k Hz				
Banda	f [Hz]	Layer 2 [%]	Layer 3 [%]	Diferencia [%]
1	689.06	1.31704e-06	1.31765e-06	6.11483e-10
2	1378.1	0.000190544	0.000190548	3.85716e-09
3	2067.2	0.00089447	0.000894465	4.62151e-09
4	2756.2	0.0096628	0.00966275	5.01912e-08
5	3445.3	0.0152177	0.0152175	2.1929e-07
6	4134.4	0.00913746	0.0867715	0.077634
7	4823.4	0.0796092	0.0796091	8.10931e-08
8	5512.5	3.38844e-26	3.38844e-26	0
9	6201.6	0.210059	0.210059	8.67638e-08
10	6890.6	0.621767	0.621766	1.27247e-06
11	7579.7	0.315987	0.31599	2.95856e-06
12	8268.8	0.657507	0.657507	3.51731e-09
13	8957.8	0.249568	0.249568	1.22466e-07
14	9646.9	0.338393	0.338388	4.89522e-06
15	10336	0.0485757	0.0485759	1.93286e-07
16	11025	3.38844e-26	3.38844e-26	0
17	11714	0.0720581	0.0720583	2.30472e-07
18	12403	0.751169	0.75117	6.46524e-07
19	13092	0.843182	0.843182	2.54005e-07
20	13781	3.46004	3.46004	2.46443e-06
21	14470	2.66152	2.66151	2.2722e-06
22	15159	8.62284	8.62283	1.10848e-05
23	15848	4.93559	4.93559	1.22924e-06
24	16538	3.38844e-26	3.38844e-26	0
25	17227	5.91655	5.91655	1.81402e-06
26	17916	12.3386	12.3386	1.67671e-06
27	18605	4.51728	4.51728	2.97652e-06
28	19294	6.92777	6.92777	6.50486e-08
29	19983	4.51728	4.51728	2.91675e-06
30	20672	6.92777	6.92777	3.35064e-08
31	21361	1.98783	1.98783	1.40538e-06
32	22050	2.09128	2.09128	4.39923e-07

Tabla 7-6: % THD de las 32 bandas de PQMF a 48k Hz

THD [%] a fs = 48k Hz				
Banda	f [Hz]	Layer 2 [%]	Layer 3 [%]	Diferencia [%]
1	750	1.31704e-06	1.31764e-06	6.03733e-10
2	1500	0.000190544	0.000190549	4.82837e-09
3	2250	6.47169e-05	0.000894468	0.000829751
4	3000	0.0096628	0.00966275	5.28745e-08
5	3750	0.00961695	0.0152174	0.0056005
6	4500	0.086772	0.0867715	4.99255e-07
7	5250	0.0796092	0.0796091	8.27222e-08
8	6000	3.38844e-26	3.38844e-26	0
9	6750	0.210059	0.210059	1.20558e-07
10	7500	0.621767	0.621766	1.17246e-06
11	8250	0.0547028	0.31599	0.261287
12	9000	0.657507	0.657507	3.48017e-08
13	9750	0.249568	0.249568	1.28518e-07
14	10500	0.338393	0.338388	4.86451e-06
15	11250	0.0485757	0.0485759	2.07223e-07
16	12000	3.38844e-26	3.38844e-26	0
17	12750	0.0720581	0.751169	0.679111
18	13500	0.751169	0.751169	5.95718e-07
19	14250	0.843182	0.843182	2.66276e-07
20	15000	3.46004	3.46004	2.16583e-06
21	15750	2.66152	2.66151	2.28934e-06
22	16500	8.62284	8.62283	9.63309e-06
23	17250	4.93559	4.93559	9.03167e-07
24	18000	3.38844e-26	3.38844e-26	0
25	18750	5.91655	5.91655	1.76096e-06
26	19500	12.3386	12.3386	2.11869e-06
27	20250	4.51728	4.51728	2.94168e-06
28	21000	6.92777	6.92777	6.11724e-08
29	21750	1.98783	1.98783	1.32741e-06
30	22500	2.09128	2.09128	4.4336e-07
31	23250	0.238838	0.238838	4.8999e-07
32	24000	15.8277	2.37089	13.4568

Tabla 7-7: SNR en dBFS de las 32 bandas de PQMF a 44.1k Hz

SNR [dBFS] a fs = 44.1k Hz				
Banda	f [Hz]	Layer 2 [dBFS]	Layer 3 [dBFS]	Diferencia [dBFS]
1	689.06	73.467331	'73.465499	39.719
2	1378.1	56.975004	56.974999	-2.32469
3	2067.2	45.361286	45.361314	-6.52194
4	2756.2	39.666442	39.666456	-15.294
5	3445.3	33.059678	33.059700	-19.8255
6	4134.4	27.460606	29.946522	26.3397
7	4823.4	25.550696	25.550693	-37.3413
8	5512.5	23.618949	23.618953	-36.0949
9	6201.6	21.336696	21.336696	-52.2707
10	6890.6	20.637363	20.637352	-35.519
11	7579.7	19.262577	19.262573	-41.1407
12	8268.8	19.811875	19.811875	-56.8648
13	8957.8	20.287384	20.287385	-55.1505
14	9646.9	22.944877	22.944876	-45.2301
15	10336	27.395092	27.395090	-34.9806
16	11025	97.332219	97.131038	83.8901
17	11714	25.682256	25.682255	-37.8998
18	12403	19.171305	19.171304	-45.6668
19	13092	15.000291	15.000292	-51.4628
20	13781	12.289704	12.289704	-56.9413
21	14470	10.008270	10.008276	-48.519
22	15159	8.572215	8.572221	-50.1105
23	15848	7.326458	7.326458	-69.4026
24	16538	6.843448	6.843448	-69.0912
25	17227	6.539130	6.539130	-63.8717
26	17916	7.016076	7.016076	-63.7794
27	18605	7.710993	7.710993	-71.8634
28	19294	9.274588	9.274588	-70.8527
29	19983	7.710993	7.710993	-67.9637
30	20672	9.274588	9.274588	-70.8963
31	21361	11.275796	11.275796	-63.1371
32	22050	14.724826	14.724826	-53.3

Tabla 7-8: SNR en dBFS de las 32 bandas de PQMF a 48k Hz

SNR [dBFS] a fs = 48k Hz				
Banda	f [Hz]	Layer 2 [dBFS]	Layer 3 [dBFS]	Diferencia [dBFS]
1	750	73.467331	73.465527	39.6523
2	1500	56.975004	56.974992	1.44335
3	2250	16.156142	45.361317	45.3561
4	3000	39.666442	39.666456	-15.0778
5	3750	25.079270	33.059701	32.3066
6	4500	29.946503	29.946523	-23.2681
7	5250	25.550696	25.550693	-37.3431
8	6000	23.618949	23.618954	-35.6924
9	6750	21.336696	21.336696	-52.4681
10	7500	20.637363	20.637348	-34.0903
11	8250	13.064173	19.262573	18.0709
12	9000	19.811875	19.811875	-58.311
13	9750	20.287384	20.287385	-55.4628
14	10500	22.944877	22.944876	-42.1561
15	11250	27.395092	27.395089	-34.5654
16	12000	97.133698	97.131700	63.7605
17	12750	25.682256	19.171304	24.5847
18	13500	19.171305	19.171304	-46.3009
19	14250	15.000291	15.000292	-51.464
20	15000	12.289704	12.289704	-56.6322
21	15750	10.008270	10.008276	-48.519
22	16500	8.572215	8.572221	-50.1352
23	17250	7.326458	7.326458	-70.3409
24	18000	6.843448	6.843448	-69.3768
25	18750	6.539130	6.539130	-63.8715
26	19500	7.016076	7.016076	-63.5284
27	20250	7.710993	7.710993	-71.7898
28	21000	9.274588	9.274588	-68.0309
29	21750	11.275796	11.275796	-60.8671
30	22500	14.724826	14.724826	-53.3678
31	23250	20.478175	20.478172	-40.3546
32	24000	-1.333825	1.074847	-2.63403

Anexo 10: Plantilla del instrumento de encuesta para la prueba subjetiva tipo MUSHRA

Numero de Prueba _____

Nombre		Firma	
Test Subjetivo del plugin PsycoPixela			

El objetivo de esta prueba es evaluar el rendimiento del plugin de audio "PsycoPixela", el cual está basado en los esquemas de compresión del MPEG en sus capas II y III. Se ha seleccionado al presente evaluador para reconocer en diferentes fragmentos de audio la calidad de compresión presentada, así como el reconocimiento de la presencia de diferentes "artefactos" o degradaciones en el material presentado.

Método de la prueba:
En cada test se presentarán tres versiones de audio, de las cuales podrá conmutar libremente entre «A», «B» o «C» en cualquier instante. Las secuencias de audio pueden repetirse indefinidamente hasta que usted esté seguro de sus evaluaciones. Cuando lo desee puede pasar al experimento siguiente, una vez que quede satisfecho con la evaluación del experimento en curso. En cada experimento se solicita que evalúe la diferencia percibida (caso de existir) entre «B» y «A», por un lado, y la diferencia entre «C» y «A», por el otro, utilizando la escala de cinco notas indicada

Degradación	Nota
Imperceptible	5
Perceptible	4
Ligeramente molesta	3
Molesta	2
Muy molesta	1

_____

Experimento 1
Asigne un nivel de degradación (de acuerdo con la tabla), comparando los fragmentos que va a escuchar.

Prueba 1

Prueba 2

Ingrese un nivel de degradación general a cada fragmento.

Nivel de degradación (nota)			

Experimento 2

Asigne un nivel de degradación (de acuerdo con la tabla), comparando los fragmentos que va a escuchar.

Prueba 1

Prueba 2

Ingrese un nivel de degradación general a cada fragmento.

Nivel de degradación (nota)			

Experimento 3

Asigne un nivel de degradación (de acuerdo con la tabla), comparando los fragmentos que va a escuchar.

Prueba 1

Prueba 2

Ingrese un nivel de degradación general a cada fragmento.

Nivel de degradación (nota)			

Experimento 4

Asigne un nivel de degradación (de acuerdo con la tabla), comparando los fragmentos que va a escuchar.

Prueba 1

Prueba 2

Ingrese un nivel de degradación general a cada fragmento.

Nivel de degradación (nota)			

Experimento 5

Asigne un nivel de degradación (de acuerdo con la tabla), comparando los fragmentos que va a escuchar.

Prueba 1

Prueba 2

Ingrese un nivel de degradación general a cada fragmento.

Nivel de degradación (nota)			

Anexo 11: Copia de consentimiento aprobado por cada participante

Numero de Prueba 1

Nombre	Jose Gomez	Firma Daniel G
Test Subjetivo del plugin PsycoPixela		

El objetivo de esta prueba es evaluar el rendimiento del plugin de audio "PsycoPixela", el cual está basado en los esquemas de compresión del MPEG en sus capas II y III.

Se ha seleccionado al presente evaluador para reconocer en diferentes fragmentos de audio la calidad de compresión presentada, así como el reconocimiento de la presencia de diferentes "artefactos" o degradaciones en el material presentado.

Método de la prueba:
En cada test se presentarán tres versiones de audio, de las cuales podrá conmutar libremente entre «A», «B» o «C» en cualquier instante. Las secuencias de audio pueden repetirse indefinidamente hasta que usted esté seguro de sus evaluaciones. Cuando lo desee puede pasar al experimento siguiente, una vez que quede satisfecho con la evaluación del experimento en curso.

En cada experimento se solicita que evalúe la diferencia percibida (caso de existir) entre «B» y «A», por un lado, y la diferencia entre «C» y «A», por el otro, utilizando la escala de cinco notas indicada

Numero de Prueba 2

Nombre	Jaime Galvis	Firma	

Test Subjetivo del plugin PsycoPixela

El objetivo de esta prueba es evaluar el rendimiento del plugin de audio "PsycoPixela", el cual está basado en los esquemas de compresión del MPEG en sus capas II y III.

Se ha seleccionado al presente evaluador para reconocer en diferentes fragmentos de audio la calidad de compresión presentada, así como el reconocimiento de la presencia de diferentes "artefactos" o degradaciones en el material presentado.

Método de la prueba:
En cada test se presentarán tres versiones de audio, de las cuales podrá conmutar libremente entre «A», «B» o «C» en cualquier instante. Las secuencias de audio pueden repetirse indefinidamente hasta que usted esté seguro de sus evaluaciones. Cuando lo desee puede pasar al experimento siguiente, una vez que quede satisfecho con la evaluación del experimento en curso.

En cada experimento se solicita que evalúe la diferencia percibida (caso de existir) entre «B» y «A», por un lado, y la diferencia entre «C» y «A», por el otro, utilizando la escala de cinco notas indicada

Numero de Prueba 3

Nombre	Fredy Vélez	Firma	

Test Subjetivo del plugin PsycoPixela

El objetivo de esta prueba es evaluar el rendimiento del plugin de audio "PsycoPixela", el cual está basado en los esquemas de compresión del MPEG en sus capas II y III.

Se ha seleccionado al presente evaluador para reconocer en diferentes fragmentos de audio la calidad de compresión presentada, así como el reconocimiento de la presencia de diferentes "artefactos" o degradaciones en el material presentado.

Método de la prueba:
En cada test se presentarán tres versiones de audio, de las cuales podrá conmutar libremente entre «A», «B» o «C» en cualquier instante. Las secuencias de audio pueden repetirse indefinidamente hasta que usted esté seguro de sus evaluaciones. Cuando lo desee puede pasar al experimento siguiente, una vez que quede satisfecho con la evaluación del experimento en curso.

En cada experimento se solicita que evalúe la diferencia percibida (caso de existir) entre «B» y «A», por un lado, y la diferencia entre «C» y «A», por el otro, utilizando la escala de cinco notas indicada

Numero de Prueba 4

Nombre	Juan Sebastian Garcia Mora	Firma	

Test Subjetivo del plugin PsycoPixela

El objetivo de esta prueba es evaluar el rendimiento del plugin de audio "PsycoPixela", el cual está basado en los esquemas de compresión del MPEG en sus capas II y III.

Se ha seleccionado al presente evaluador para reconocer en diferentes fragmentos de audio la calidad de compresión presentada, así como el reconocimiento de la presencia de diferentes "artefactos" o degradaciones en el material presentado.

Método de la prueba:
En cada test se presentarán tres versiones de audio, de las cuales podrá conmutar libremente entre «A», «B» o «C» en cualquier instante. Las secuencias de audio pueden repetirse indefinidamente hasta que usted esté seguro de sus evaluaciones. Cuando lo desee puede pasar al experimento siguiente, una vez que quede satisfecho con la evaluación del experimento en curso.

En cada experimento se solicita que evalúe la diferencia percibida (caso de existir) entre «B» y «A», por un lado, y la diferencia entre «C» y «A», por el otro, utilizando la escala de cinco notas indicada

Numero de Prueba 5

Nombre	Fabian David Castellanos	Firma	

Test Subjetivo del plugin PsycoPixela

El objetivo de esta prueba es evaluar el rendimiento del plugin de audio "PsycoPixela", el cual está basado en los esquemas de compresión del MPEG en sus capas II y III.

Se ha seleccionado al presente evaluador para reconocer en diferentes fragmentos de audio la calidad de compresión presentada, así como el reconocimiento de la presencia de diferentes "artefactos" o degradaciones en el material presentado.

Método de la prueba:
En cada test se presentarán tres versiones de audio, de las cuales podrá conmutar libremente entre «A», «B» o «C» en cualquier instante. Las secuencias de audio pueden repetirse indefinidamente hasta que usted esté seguro de sus evaluaciones. Cuando lo desee puede pasar al experimento siguiente, una vez que quede satisfecho con la evaluación del experimento en curso.

En cada experimento se solicita que evalúe la diferencia percibida (caso de existir) entre «B» y «A», por un lado, y la diferencia entre «C» y «A», por el otro, utilizando la escala de cinco notas indicada

Numero de Prueba 6

Nombre	Santiago Ramirez Sarmiento	Firma	

Test Subjetivo del plugin PsycoPixela

El objetivo de esta prueba es evaluar el rendimiento del plugin de audio "PsycoPixela", el cual está basado en los esquemas de compresión del MPEG en sus capas II y III.

Se ha seleccionado al presente evaluador para reconocer en diferentes fragmentos de audio la calidad de compresión presentada, así como el reconocimiento de la presencia de diferentes "artefactos" o degradaciones en el material presentado.

Método de la prueba:
En cada test se presentarán tres versiones de audio, de las cuales podrá conmutar libremente entre «A», «B» o «C» en cualquier instante. Las secuencias de audio pueden repetirse indefinidamente hasta que usted esté seguro de sus evaluaciones. Cuando lo desee puede pasar al experimento siguiente, una vez que quede satisfecho con la evaluación del experimento en curso.

En cada experimento se solicita que evalúe la diferencia percibida (caso de existir) entre «B» y «A», por un lado, y la diferencia entre «C» y «A», por el otro, utilizando la escala de cinco notas indicada

Numero de Prueba 7

Nombre	Juan Pablo Prada	Firma	

Test Subjetivo del plugin PsycoPixela

El objetivo de esta prueba es evaluar el rendimiento del plugin de audio "PsycoPixela", el cual está basado en los esquemas de compresión del MPEG en sus capas II y III.

Se ha seleccionado al presente evaluador para reconocer en diferentes fragmentos de audio la calidad de compresión presentada, así como el reconocimiento de la presencia de diferentes "artefactos" o degradaciones en el material presentado.

Método de la prueba:
En cada test se presentarán tres versiones de audio, de las cuales podrá conmutar libremente entre «A», «B» o «C» en cualquier instante. Las secuencias de audio pueden repetirse indefinidamente hasta que usted esté seguro de sus evaluaciones. Cuando lo desee puede pasar al experimento siguiente, una vez que quede satisfecho con la evaluación del experimento en curso.

En cada experimento se solicita que evalúe la diferencia percibida (caso de existir) entre «B» y «A», por un lado, y la diferencia entre «C» y «A», por el otro, utilizando la escala de cinco notas indicada

Numero de Prueba 8

Nombre	David F. Aldana	Firma	

Test Subjetivo del plugin PsycoPixela

El objetivo de esta prueba es evaluar el rendimiento del plugin de audio "PsycoPixela", el cual está basado en los esquemas de compresión del MPEG en sus capas II y III.

Se ha seleccionado al presente evaluador para reconocer en diferentes fragmentos de audio la calidad de compresión presentada, así como el reconocimiento de la presencia de diferentes "artefactos" o degradaciones en el material presentado.

Método de la prueba:
En cada test se presentarán tres versiones de audio, de las cuales podrá conmutar libremente entre «A», «B» o «C» en cualquier instante. Las secuencias de audio pueden repetirse indefinidamente hasta que usted esté seguro de sus evaluaciones. Cuando lo desee puede pasar al experimento siguiente, una vez que quede satisfecho con la evaluación del experimento en curso.

En cada experimento se solicita que evalúe la diferencia percibida (caso de existir) entre «B» y «A», por un lado, y la diferencia entre «C» y «A», por el otro, utilizando la escala de cinco notas indicada

Numero de Prueba 9

Nombre	Oscar J. Rodriguez C.	Firma	

Test Subjetivo del plugin PsycoPixela

El objetivo de esta prueba es evaluar el rendimiento del plugin de audio "PsycoPixela", el cual está basado en los esquemas de compresión del MPEG en sus capas II y III.

Se ha seleccionado al presente evaluador para reconocer en diferentes fragmentos de audio la calidad de compresión presentada, así como el reconocimiento de la presencia de diferentes "artefactos" o degradaciones en el material presentado.

Método de la prueba:
En cada test se presentarán tres versiones de audio, de las cuales podrá conmutar libremente entre «A», «B» o «C» en cualquier instante. Las secuencias de audio pueden repetirse indefinidamente hasta que usted esté seguro de sus evaluaciones. Cuando lo desee puede pasar al experimento siguiente, una vez que quede satisfecho con la evaluación del experimento en curso.

En cada experimento se solicita que evalúe la diferencia percibida (caso de existir) entre «B» y «A», por un lado, y la diferencia entre «C» y «A», por el otro, utilizando la escala de cinco notas indicada

Numero de Prueba 10

Nombre	Sebastian Galindo	Firma	

Test Subjetivo del plugin PsycoPixela

El objetivo de esta prueba es evaluar el rendimiento del plugin de audio "PsycoPixela", el cual está basado en los esquemas de compresión del MPEG en sus capas II y III.

Se ha seleccionado al presente evaluador para reconocer en diferentes fragmentos de audio la calidad de compresión presentada, así como el reconocimiento de la presencia de diferentes "artefactos" o degradaciones en el material presentado.

Método de la prueba:
En cada test se presentarán tres versiones de audio, de las cuales podrá conmutar libremente entre «A», «B» o «C» en cualquier instante. Las secuencias de audio pueden repetirse indefinidamente hasta que usted esté seguro de sus evaluaciones. Cuando lo desee puede pasar al experimento siguiente, una vez que quede satisfecho con la evaluación del experimento en curso.

En cada experimento se solicita que evalúe la diferencia percibida (caso de existir) entre «B» y «A», por un lado, y la diferencia entre «C» y «A», por el otro, utilizando la escala de cinco notas indicada

Numero de Prueba 11

Nombre	Santiago Garzón Carrero	Firma	

Test Subjetivo del plugin PsycoPixela

El objetivo de esta prueba es evaluar el rendimiento del plugin de audio "PsycoPixela", el cual está basado en los esquemas de compresión del MPEG en sus capas II y III.

Se ha seleccionado al presente evaluador para reconocer en diferentes fragmentos de audio la calidad de compresión presentada, así como el reconocimiento de la presencia de diferentes "artefactos" o degradaciones en el material presentado.

Método de la prueba:
En cada test se presentarán tres versiones de audio, de las cuales podrá conmutar libremente entre «A», «B» o «C» en cualquier instante. Las secuencias de audio pueden repetirse indefinidamente hasta que usted esté seguro de sus evaluaciones. Cuando lo desee puede pasar al experimento siguiente, una vez que quede satisfecho con la evaluación del experimento en curso.

En cada experimento se solicita que evalúe la diferencia percibida (caso de existir) entre «B» y «A», por un lado, y la diferencia entre «C» y «A», por el otro, utilizando la escala de cinco notas indicada

Numero de Prueba 12

Nombre	Brayan Aguirre	Firma	

Test Subjetivo del plugin PsycoPixela

El objetivo de esta prueba es evaluar el rendimiento del plugin de audio "PsycoPixela", el cual está basado en los esquemas de compresión del MPEG en sus capas II y III.

Se ha seleccionado al presente evaluador para reconocer en diferentes fragmentos de audio la calidad de compresión presentada, así como el reconocimiento de la presencia de diferentes "artefactos" o degradaciones en el material presentado.

Método de la prueba:
En cada test se presentarán tres versiones de audio, de las cuales podrá conmutar libremente entre «A», «B» o «C» en cualquier instante. Las secuencias de audio pueden repetirse indefinidamente hasta que usted esté seguro de sus evaluaciones. Cuando lo desee puede pasar al experimento siguiente, una vez que quede satisfecho con la evaluación del experimento en curso.

En cada experimento se solicita que evalúe la diferencia percibida (caso de existir) entre «B» y «A», por un lado, y la diferencia entre «C» y «A», por el otro, utilizando la escala de cinco notas indicada

Numero de Prueba 13

Nombre	Carlos Daniel Cuervo Rodriguez	Firma	

Test Subjetivo del plugin PsycoPixela

El objetivo de esta prueba es evaluar el rendimiento del plugin de audio "PsycoPixela", el cual está basado en los esquemas de compresión del MPEG en sus capas II y III.

Se ha seleccionado al presente evaluador para reconocer en diferentes fragmentos de audio la calidad de compresión presentada, así como el reconocimiento de la presencia de diferentes "artefactos" o degradaciones en el material presentado.

Método de la prueba:
En cada test se presentarán tres versiones de audio, de las cuales podrá conmutar libremente entre «A», «B» o «C» en cualquier instante. Las secuencias de audio pueden repetirse indefinidamente hasta que usted esté seguro de sus evaluaciones. Cuando lo desee puede pasar al experimento siguiente, una vez que quede satisfecho con la evaluación del experimento en curso.

En cada experimento se solicita que evalúe la diferencia percibida (caso de existir) entre «B» y «A», por un lado, y la diferencia entre «C» y «A», por el otro, utilizando la escala de cinco notas indicada

Numero de Prueba 14

Nombre	Arangurren Baril Juan David	Firma	

Test Subjetivo del plugin PsycoPixela

El objetivo de esta prueba es evaluar el rendimiento del plugin de audio "PsycoPixela", el cual está basado en los esquemas de compresión del MPEG en sus capas II y III.

Se ha seleccionado al presente evaluador para reconocer en diferentes fragmentos de audio la calidad de compresión presentada, así como el reconocimiento de la presencia de diferentes "artefactos" o degradaciones en el material presentado.

Método de la prueba:
En cada test se presentarán tres versiones de audio, de las cuales podrá conmutar libremente entre «A», «B» o «C» en cualquier instante. Las secuencias de audio pueden repetirse indefinidamente hasta que usted esté seguro de sus evaluaciones. Cuando lo desee puede pasar al experimento siguiente, una vez que quede satisfecho con la evaluación del experimento en curso.

En cada experimento se solicita que evalúe la diferencia percibida (caso de existir) entre «B» y «A», por un lado, y la diferencia entre «C» y «A», por el otro, utilizando la escala de cinco notas indicada

Numero de Prueba 15

Nombre	Juan David Bueno Erazo	Firma	

Test Subjetivo del plugin PsycoPixela

El objetivo de esta prueba es evaluar el rendimiento del plugin de audio "PsycoPixela", el cual está basado en los esquemas de compresión del MPEG en sus capas II y III.

Se ha seleccionado al presente evaluador para reconocer en diferentes fragmentos de audio la calidad de compresión presentada, así como el reconocimiento de la presencia de diferentes "artefactos" o degradaciones en el material presentado.

Método de la prueba:
En cada test se presentarán tres versiones de audio, de las cuales podrá conmutar libremente entre «A», «B» o «C» en cualquier instante. Las secuencias de audio pueden repetirse indefinidamente hasta que usted esté seguro de sus evaluaciones. Cuando lo desee puede pasar al experimento siguiente, una vez que quede satisfecho con la evaluación del experimento en curso.

En cada experimento se solicita que evalúe la diferencia percibida (caso de existir) entre «B» y «A», por un lado, y la diferencia entre «C» y «A», por el otro, utilizando la escala de cinco notas indicada

Numero de Prueba 16

Nombre	Juan Diego Rodriguez Chavez	Firma	

Test Subjetivo del plugin PsycoPixela

El objetivo de esta prueba es evaluar el rendimiento del plugin de audio "PsycoPixela", el cual está basado en los esquemas de compresión del MPEG en sus capas II y III.

Se ha seleccionado al presente evaluador para reconocer en diferentes fragmentos de audio la calidad de compresión presentada, así como el reconocimiento de la presencia de diferentes "artefactos" o degradaciones en el material presentado.

Método de la prueba:
En cada test se presentarán tres versiones de audio, de las cuales podrá conmutar libremente entre «A», «B» o «C» en cualquier instante. Las secuencias de audio pueden repetirse indefinidamente hasta que usted esté seguro de sus evaluaciones. Cuando lo desee puede pasar al experimento siguiente, una vez que quede satisfecho con la evaluación del experimento en curso.

En cada experimento se solicita que evalúe la diferencia percibida (caso de existir) entre «B» y «A», por un lado, y la diferencia entre «C» y «A», por el otro, utilizando la escala de cinco notas indicada

Numero de Prueba 17

Nombre	Angie Natalia Molina Serrano	Firma	

Test Subjetivo del plugin PsycoPixela

El objetivo de esta prueba es evaluar el rendimiento del plugin de audio "PsycoPixela", el cual está basado en los esquemas de compresión del MPEG en sus capas II y III.

Se ha seleccionado al presente evaluador para reconocer en diferentes fragmentos de audio la calidad de compresión presentada, así como el reconocimiento de la presencia de diferentes "artefactos" o degradaciones en el material presentado.

Método de la prueba:
En cada test se presentarán tres versiones de audio, de las cuales podrá conmutar libremente entre «A», «B» o «C» en cualquier instante. Las secuencias de audio pueden repetirse indefinidamente hasta que usted esté seguro de sus evaluaciones. Cuando lo desee puede pasar al experimento siguiente, una vez que quede satisfecho con la evaluación del experimento en curso.

En cada experimento se solicita que evalúe la diferencia percibida (caso de existir) entre «B» y «A», por un lado, y la diferencia entre «C» y «A», por el otro, utilizando la escala de cinco notas indicada

Numero de Prueba 18

Nombre	[illegible]	Firma	[illegible]

Test Subjetivo del plugin PsycoPixela

El objetivo de esta prueba es evaluar el rendimiento del plugin de audio "PsycoPixela", el cual está basado en los esquemas de compresión del MPEG en sus capas II y III.

Se ha seleccionado al presente evaluador para reconocer en diferentes fragmentos de audio la calidad de compresión presentada, así como el reconocimiento de la presencia de diferentes "artefactos" o degradaciones en el material presentado.

Método de la prueba:
En cada test se presentarán tres versiones de audio, de las cuales podrá conmutar libremente entre «A», «B» o «C» en cualquier instante. Las secuencias de audio pueden repetirse indefinidamente hasta que usted esté seguro de sus evaluaciones. Cuando lo desee puede pasar al experimento siguiente, una vez que quede satisfecho con la evaluación del experimento en curso.

En cada experimento se solicita que evalúe la diferencia percibida (caso de existir) entre «B» y «A», por un lado, y la diferencia entre «C» y «A», por el otro, utilizando la escala de cinco notas indicada

Numero de Prueba 19

Nombre	[illegible]	Firma	[illegible]

Test Subjetivo del plugin PsycoPixela

El objetivo de esta prueba es evaluar el rendimiento del plugin de audio "PsycoPixela", el cual está basado en los esquemas de compresión del MPEG en sus capas II y III.

Se ha seleccionado al presente evaluador para reconocer en diferentes fragmentos de audio la calidad de compresión presentada, así como el reconocimiento de la presencia de diferentes "artefactos" o degradaciones en el material presentado.

Método de la prueba:
En cada test se presentarán tres versiones de audio, de las cuales podrá conmutar libremente entre «A», «B» o «C» en cualquier instante. Las secuencias de audio pueden repetirse indefinidamente hasta que usted esté seguro de sus evaluaciones. Cuando lo desee puede pasar al experimento siguiente, una vez que quede satisfecho con la evaluación del experimento en curso.

En cada experimento se solicita que evalúe la diferencia percibida (caso de existir) entre «B» y «A», por un lado, y la diferencia entre «C» y «A», por el otro, utilizando la escala de cinco notas indicada

Numero de Prueba 20

Nombre	Christian R. Meca Paya	Firma	[illegible]

Test Subjetivo del plugin PsycoPixela

El objetivo de esta prueba es evaluar el rendimiento del plugin de audio "PsycoPixela", el cual está basado en los esquemas de compresión del MPEG en sus capas II y III.

Se ha seleccionado al presente evaluador para reconocer en diferentes fragmentos de audio la calidad de compresión presentada, así como el reconocimiento de la presencia de diferentes "artefactos" o degradaciones en el material presentado.

Método de la prueba:
En cada test se presentarán tres versiones de audio, de las cuales podrá conmutar libremente entre «A», «B» o «C» en cualquier instante. Las secuencias de audio pueden repetirse indefinidamente hasta que usted esté seguro de sus evaluaciones. Cuando lo desee puede pasar al experimento siguiente, una vez que quede satisfecho con la evaluación del experimento en curso.

En cada experimento se solicita que evalúe la diferencia percibida (caso de existir) entre «B» y «A», por un lado, y la diferencia entre «C» y «A», por el otro, utilizando la escala de cinco notas indicada

Numero de Prueba 21

Nombre	Juan Pablo Roa Amado	Firma	[illegible]

Test Subjetivo del plugin PsycoPixela

El objetivo de esta prueba es evaluar el rendimiento del plugin de audio "PsycoPixela", el cual está basado en los esquemas de compresión del MPEG en sus capas II y III.

Se ha seleccionado al presente evaluador para reconocer en diferentes fragmentos de audio la calidad de compresión presentada, así como el reconocimiento de la presencia de diferentes "artefactos" o degradaciones en el material presentado.

Método de la prueba:
En cada test se presentarán tres versiones de audio, de las cuales podrá conmutar libremente entre «A», «B» o «C» en cualquier instante. Las secuencias de audio pueden repetirse indefinidamente hasta que usted esté seguro de sus evaluaciones. Cuando lo desee puede pasar al experimento siguiente, una vez que quede satisfecho con la evaluación del experimento en curso.

En cada experimento se solicita que evalúe la diferencia percibida (caso de existir) entre «B» y «A», por un lado, y la diferencia entre «C» y «A», por el otro, utilizando la escala de cinco notas indicada

Numero de Prueba 22

Nombre	Andrés Felipe Castillo Patiño	Firma	

Test Subjetivo del plugin PsycoPixela

El objetivo de esta prueba es evaluar el rendimiento del plugin de audio "PsycoPixela", el cual está basado en los esquemas de compresión del MPEG en sus capas II y III.

Se ha seleccionado al presente evaluador para reconocer en diferentes fragmentos de audio la calidad de compresión presentada, así como el reconocimiento de la presencia de diferentes "artefactos" o degradaciones en el material presentado.

Método de la prueba:
En cada test se presentarán tres versiones de audio, de las cuales podrá conmutar libremente entre «A», «B» o «C» en cualquier instante. Las secuencias de audio pueden repetirse indefinidamente hasta que usted esté seguro de sus evaluaciones. Cuando lo desee puede pasar al experimento siguiente, una vez que quede satisfecho con la evaluación del experimento en curso.

En cada experimento se solicita que evalúe la diferencia percibida (caso de existir) entre «B» y «A», por un lado, y la diferencia entre «C» y «A», por el otro, utilizando la escala de cinco notas indicada

Numero de Prueba 23

Nombre	Juan Esteban Gómez H.	Firma	

Test Subjetivo del plugin PsycoPixela

El objetivo de esta prueba es evaluar el rendimiento del plugin de audio "PsycoPixela", el cual está basado en los esquemas de compresión del MPEG en sus capas II y III.

Se ha seleccionado al presente evaluador para reconocer en diferentes fragmentos de audio la calidad de compresión presentada, así como el reconocimiento de la presencia de diferentes "artefactos" o degradaciones en el material presentado.

Método de la prueba:
En cada test se presentarán tres versiones de audio, de las cuales podrá conmutar libremente entre «A», «B» o «C» en cualquier instante. Las secuencias de audio pueden repetirse indefinidamente hasta que usted esté seguro de sus evaluaciones. Cuando lo desee puede pasar al experimento siguiente, una vez que quede satisfecho con la evaluación del experimento en curso.

En cada experimento se solicita que evalúe la diferencia percibida (caso de existir) entre «B» y «A», por un lado, y la diferencia entre «C» y «A», por el otro, utilizando la escala de cinco notas indicada

Numero de Prueba 24

Nombre	Andres Sierra	Firma	

Test Subjetivo del plugin PsycoPixela

El objetivo de esta prueba es evaluar el rendimiento del plugin de audio "PsycoPixela", el cual está basado en los esquemas de compresión del MPEG en sus capas II y III.

Se ha seleccionado al presente evaluador para reconocer en diferentes fragmentos de audio la calidad de compresión presentada, así como el reconocimiento de la presencia de diferentes "artefactos" o degradaciones en el material presentado.

Método de la prueba:
En cada test se presentarán tres versiones de audio, de las cuales podrá conmutar libremente entre «A», «B» o «C» en cualquier instante. Las secuencias de audio pueden repetirse indefinidamente hasta que usted esté seguro de sus evaluaciones. Cuando lo desee puede pasar al experimento siguiente, una vez que quede satisfecho con la evaluación del experimento en curso.

En cada experimento se solicita que evalúe la diferencia percibida (caso de existir) entre «B» y «A», por un lado, y la diferencia entre «C» y «A», por el otro, utilizando la escala de cinco notas indicada

Numero de Prueba 25 Carlos Monroy

Nombre		Firma	

Test Subjetivo del plugin PsycoPixela

El objetivo de esta prueba es evaluar el rendimiento del plugin de audio "PsycoPixela", el cual está basado en los esquemas de compresión del MPEG en sus capas II y III.

Se ha seleccionado al presente evaluador para reconocer en diferentes fragmentos de audio la calidad de compresión presentada, así como el reconocimiento de la presencia de diferentes "artefactos" o degradaciones en el material presentado.

Método de la prueba:
En cada test se presentarán tres versiones de audio, de las cuales podrá conmutar libremente entre «A», «B» o «C» en cualquier instante. Las secuencias de audio pueden repetirse indefinidamente hasta que usted esté seguro de sus evaluaciones. Cuando lo desee puede pasar al experimento siguiente, una vez que quede satisfecho con la evaluación del experimento en curso.

En cada experimento se solicita que evalúe la diferencia percibida (caso de existir) entre «B» y «A», por un lado, y la diferencia entre «C» y «A», por el otro, utilizando la escala de cinco notas indicada