Informe Proyecto de Investigación 222

Implementación en DSPs de Técnicas de Comunicación basadas en Wavelets

Jorge Iván Marín Hurtado
Investigador Principal

Alexander López Parrado
Coinvestigador

Universidad del Quindío
Facultad de Ingeniería
Grupo de Procesamiento Digital de Señales
y Procesadores - GDSPROC

2 de diciembre de 2004

Resumen

Se presentan los resultados del proyecto de investigación "*Implementación en DSPs de Técnicas de Comunicación basadas en Wavelets*", cuyo propósito es la construcción de sistemas en tiempo real que hacen uso de tecnología DSP (Procesador Digital de Señales) para aplicaciones en sistemas de comunicaciones. Para el diseño de los algoritmos se hace uso de técnicas derivadas del tratamiento wavelet. Para tal efecto se propone: un esquema de compresión híbrido basado en los paquetes wavelet y los coeficientes de predicción LPC; un sistema de sincronización para la técnica de multiplexación por división de paquetes wavelet (WPDM); y un sistema de encriptación híbrido basado en wavelets y sistemas caóticos. Se presentan diferentes pruebas de desempeño de los sistemas y se analizan sus respectivas complejidad computacionales.

Palabras Claves: Transformada Wavelet, Paquetes Wavelets, Compresión, Multiplexación, Encriptación, Procesador Digital de Señales.

Abstract

This paper presents the results of the research project: "*Implementation on DSPs of wavelet-based communication techniques*". We propose a new wavelet-based compression and encryption methods, and a modified version or the wavelet packet division multiplexing (WPDM). The compression system is based wavelet packet, using dynamic and static tree, and the linear prediction coefficients (LPC) to sub-band coding. Besides, a new synchronization technique is proposed for the WPDM system. The encryption system uses over-complete wavelet transform and chaos like mother function. These systems were implemented on an digital signal processor (DSP) in order to be used on real-time applications. Performance, computational costs and implementation details of these techniques was analyzed.

Keywords: Wavelet Transform, Wavelet Packets, Compression, Multiplexing, Encryption, Digital Signal Processor.

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

En este documento se presentan los resultados del Proyecto de Investigación 222: "*Implementación en DSPs de Técnicas de Comunicación basadas en Wavelets*". Este proyecto fue realizado por los integrantes del Grupo de Procesamiento Digital de Señales y Procesadores (GDSPROC) de la Universidad del Quindío, los docentes Jorge Iván Marín Hurtado y Alexander López Parrado, y fueron vinculados en calidad de jóvenes investigadores los estudiantes de Ingeniería Electrónica: Alex García Quinchía, Antonio Ramos Murillo, Carlos Andrés Giraldo Castañeda, Germán Auguso Ramírez Alzate, Lilian Johanny Certuche Alzate y Oscar Hernán Ocampo Hernández.

El propósito de este proyecto es el estudio de las diferentes aplicaciones de las wavelets en los sistemas de comunicación de señales de voz, tales como: compresión, multiplexación y encriptación de señales, con el ánimo de diseñar algoritmos computacionalmente eficientes para su implementación sobre procesadores digitales de señales (DSPs). Para ello, se estudiaron las características de los diferentes sistemas de comunicaciones, basados en wavelets, reportados en la literatura [AM99, SN97, Teo98, Het, HSD95, Sal01, RY01, WWDJ97, Won98, CF04, Cui03, SAA98], y se efectuaron una serie de simulaciones en Matlab y algoritmos en C con el fin de determinar las condiciones y esquemas más apropiados para la implementación de dichos sistemas, así mismo, verificar la eficiencia de los técnicas reportadas y proponer nuevas variantes y mejoras.

Dado a que existen diferentes versiones de la Transformada Wavelet (WT) [RB98, Teo98, Wic94], en este trabajo se emplearon dos esquemas, la Transformada Wavelet Sobre-completa (OCWT) [Teo98] y los Paquetes Wavelet (WP) [Wic94]. La primera fue usada en diseño del sistema de encriptación, y la segunda en desarrollo de los sistemas de multiplexación y compresión. Para cada uno de estas variantes de la WT se elaboraron algoritmos optimizados para un DSP de punto flotante de Texas Instruments, el TMS320C6701. Tanto los aspectos teóricos sobre cada una de estas transformadas, y los respectivos algoritmos y optimizaciones desarrollados en este proyecto, se describen en detalle en el capítulo 3.

Para facilitar la presentación de los resultados de este proyecto, cada uno de las técnicas implementadas, compresión, multiplexación y encriptación, se presenta en un

capítulo diferente (Capítulos 4-6). En cada uno de éstos, se hace una presentación del estado del arte respecto a la aplicación de las wavelets a la técnica en particular, el esquema propuesto en este trabajo y sus respectivas pruebas de desempeño y análisis comparativo con los sistemas reportados en la literatura, y por último las conclusiones.

Finalmente, en el capítulo 7 se presentan las conclusiones generales de esta investigación.

Capítulo 2

Presentación en Eventos y Publicaciones

Como resultado de este proyecto de investigación, se han elaborado cuatro artículos y ponencias:

- *Sistema de encriptación de señales basado en wavelets y caos.* Ponencia en el VIII Simposio de Tratamiento de Señales, Imágenes y Visión Artificial. Universidad Pontificia Bolivariana de Medellín del 5 al 7 de Noviembre de 2003.
- *Compresión de señales de voz en tiempo real usando paquetes wavelet.* Ponencia en el IX Simposio de Tratamiento de Señales, Imágenes y Visión Artificial. Universidad Nacional de Colombia sede Manizales. 15 al 17 de Septiembre de 2004.
- *Implementación de un sistema de multiplexación por división de paquetes wavelet (WPDM).* Ponencia en el IX Simposio de Tratamiento de Señales, Imágenes y Visión Artificial. Universidad Nacional de Colombia sede Manizales. 15 al 17 de Septiembre de 2004.
- *Sistema en tiempo real para la encriptación basado en wavelets pseudocaóticas.* Ponencia en el IX Simposio de Tratamiento de Señales, Imágenes y Visión Artificial. Universidad Nacional de Colombia sede Manizales. 15 al 17 de Septiembre de 2004.

Y se participó también en el siguiente evento de divulgación científica:

- Aplicación de la transformada wavelet y los DSPs en el reconocimiento y en los sistemas de comunicación. Ponencia la II Feria de la Ciencia, el Arte y la Tecnología del Eje Cafetero. Universidad Tecnológica de Pereira del 27 al 29 de Agosto de 2003.

Capítulo 3

Las Wavelets. Implementación de algoritmos

La Transformada Wavelet (WT) es una técnica matemática que permite el estudio de señales no estacionarias, y a diferencia con la Transformada Corta de Fourier (STFT), ofrece una mejor resolución y localización frecuencia-tiempo de las componentes de frecuencia, y la posibilidad de trabajar con funciones base (wavelet madre) más semejantes a la señal a analizar, en lugar de hacerlo con exponenciales complejas [Teo98, SN97]. Aunque ha sido usada ampliamente en la construcción de sistemas de compresión de imágenes [Teo98, SN97, RB98, Sal01], se ha utilizado también en el diseño de sistemas de encriptación y multiplexación de señales[AM99, Het, HSD95, WWDJ97, WWD$^+$00, LCL02, Lin97, CF04].

Existen diferentes variantes para la transformada wavelet: la Transformada Wavelet Continua (CWT), su versión discreta, la Transformada Wavelet Sobre-completa (OCWT o SCWT), la Transformada Wavelet Discreta (DWT) y los Paquetes Wavelet (WP). Para conocer sus principales diferencias se hará inicialmente una breve descripción de las características de cada una de estas variantes, y posteriormente se enfatizará en los métodos de cómputo de la OCWT y los Paquetes Wavelet, dado a que estas técnicas fueron las empleadas en el diseño de los sistemas de compresión, multiplexación y encriptación de señales de voz que se discuten en este trabajo. Finalmente, se presentarán los algoritmos propuestos y las optimizaciones para su ejecución sobre un DSP TMS320C6701.

3.1 Tipos de Transformada Wavelet

El propósito de la WT es la descomposición de una señal $f(\tau)$ en una combinación lineal de versiones dilatadas y desplazadas de la función madre $g(\tau)$, lo cual se denota a través del operador producto interno como [Teo98]:

$$(\mathcal{W}_g f)(t,s) = \langle f\,,\, \tau_t D_s g\rangle = \left\langle f(\tau)\,,\, s^{1/2} g(s(\tau - t))\right\rangle \tag{3.1}$$

donde $\mathcal{W}_g$ representa el operador transformada wavelet, τ_t el operador desplazamien-

to y D_s el operador dilatación. Nótese que la WT de una señal unidimensional es una función bidimensional, en la cual los ejes son t (tiempo) y s (escala), esta última está asociada al inverso de la frecuencia. Por estas particularidades, la WT es por excelencia, una transformación frecuencia-tiempo de la señal.

Para que la transformación sea posible, se requiere que la función madre cumpla ciertas propiedades como son: ser una función de energía finita (pertenecer al espacio de Hilbert $L^2(\mathbf{R})$), oscilar en el tiempo y tener media cero. Por estas características, las funciones base reciben el nombre de onditas o *wavelet (wave + little)*. Es importante indicar que g no necesariamente tiene que formar una base ortonormal en el espacio $L^2(\mathbf{R})$ [Teo98].

La ecuación (3.1) se puede llevar a la forma de una notación en convolución [Teo98]:

$$(\mathcal{W}_g f)(t,s) = (f \star D_s \widetilde{g})(t,s) \tag{3.2}$$

siendo $\widetilde{g}$ el operador de involución que retorna la versión reflejada del complejo conjugado $\overline{g(-\tau)}$; lo cual permite visualizar la transformación como un conjunto infinito de banco de filtros, donde cada filtro tiene una diferente respuesta al impulso igual a la versión dilatada o comprimida de la función base $(D_s\widetilde{g})$.

Para las ecs. (3.1) y (3.2) tanto t como s pueden tomar cualquier valor en el dominio de $\mathbf{R}$, por lo tanto, estas representaciones conforman la que se denomina la Transformada Wavelet Continua o CWT, y su cálculo implica el empleo de funciones continuas.

En la práctica la CWT se puede calcular haciendo una discretización y restricción a la región de evaluación $\{t, s\}$. Algunos autores denominan habitualmente a esta transformada como la CWT, sin embargo, un término más apropiado es el de Transformada Wavelet Sobre-completa (*OCWT: Overcomplete Wavelet Transform*) [Teo98] o Transformada Wavelet Continua Muestreada (*SCWT: Sampled CWT*) [GKMM89]. Con la OCWT se puede usar una discretización uniforme para t y s o alguna cuantización arbitraria.

Solamente cuando la discretización de la escala $s = 2^k$ y tiempo $t = 2^k n$ es en potencias de dos o diádica, la transformada recibe el nombre de Transformada Wavelet Discreta (*DWT: Discrete Wavelet Transform*). En la Figura 3.1 se muestra la distribución de escalas y tiempo para una DWT. Nótese que para altas frecuencias (baja escala) se asocian muchos instantes de tiempo, en cambio para bajas frecuencias (alta escala), el número de muestras de tiempo es menor.

A diferencia con la OCWT, para calcular la DWT se requiere que la wavelet madre sea la base de un espacio ortogonal, de allí que para el empleo de la DWT no se pueda usar cualquier tipo de función base de descomposición. Por otro lado, el esquema de la DWT permite realizar lo que se denomina Análisis Multirresolución (MRA) e implementar la DWT, en forma eficiente, a través de un banco de filtros de octavas [RB98, Fli94].

En el MRA, a cada uno de los niveles de descomposición (o diferentes escalas) se le asocia un subespacio V_k generado a partir de la dilatación y translación de una función de escalamiento $\phi \subset V_0$[Teo98, RB98, Fli94]:

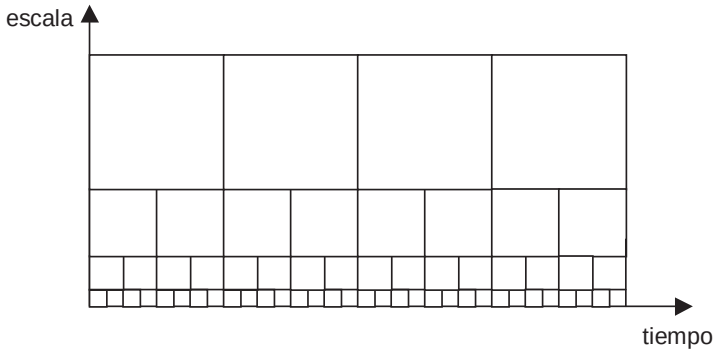


Figura 3.1: Distribución de escala-tiempo para la Transformada Wavelet Discreta.

$$V_k = \{D_{2^k}\tau_n\phi\} \tag{3.3}$$

teniéndose que los subespacios cumplen la propiedad:

$$\{0\} \subset ... \subset V_{-1} \subset V_0 \subset V_1 \subset ... \subset \mathcal{H} \tag{3.4}$$

siendo $\mathcal{H}$ el espacio de Hilbert completo.

Cuando se representa una señal continua f en el espacio V_k (llamemos a esta representación f_k), ésta tendrá un nivel de aproximación mayor a la señal original que si se representara en el espacio V_{k-1}, es decir, V_{k-1} es un espacio de menor resolución, y f_k se aproxima más a f que la versión f_{k-1}.

Por otra parte, lo que le resta a la señal f_{k-1} para convertirse en la señal f_k es lo que se denomina detalle $g_k = f_k - f_{k-1}$. Los detalles son señales que pertenecen a un espacio W_k, el cual es formado por medio de las denominadas funciones wavelet ψ[Teo98, RB98, Fli94]:

$$W_k = \{D_{2^k}\tau_n\psi\} \tag{3.5}$$

Dos espacios W_{k_1} y W_{k_2} cualquiera son ortogonales entre sí, lo que garantiza que las funciones wavelet de dos escalas diferentes sean ortogonales entre sí $\langle\psi_{k_1}, \psi_{k_2}\rangle = \delta_{k_1,k_2}$, así mismo, se cumplen las siguientes relaciones entre los subespacios V_k y W_k[Teo98, RB98, Fli94]:

$$V_{k+1} = V_k \oplus W_k \tag{3.6}$$

$$V_k \perp W_k \tag{3.7}$$

Estas propiedades entre espacios, permiten que la DWT pueda ser calculada por medio de un banco de filtros de octavas (Figura 3.2), en el cual, la función de escalamiento ϕ genera un filtro FIR pasa-bajo con respuesta al impulso[Teo98, RB98, Fli94]:

$$h_0[n] = \langle \phi, D_2 \tau_n \phi \rangle \tag{3.8}$$

y la función wavelet ψ uno pasa-alto con respuesta al impulso

$$h_1[n] = \langle \psi, D_2 \tau_n \phi \rangle \tag{3.9}$$

La salida de los filtros pasa-alto se denomina detalles (d), y la de los pasa-bajo (a), aproximaciones. Como puede verse de la Figura 3.2, tanto la salida del filtro pasa-bajo como la del pasa-alto se diezman por un factor de 2, de esta forma, la DWT entrega L detalles y 1 aproximación, siendo L el número de niveles de descomposición; y la longitud de los datos de salida en cada uno de los niveles de detalles d_{-k} y aproximaciones a_{-k} es 2^k veces más pequeño que la longitud de datos de entrada en la base del árbol de análisis. Esta estructura se denomina la Transformada Wavelet Rápida (FWT) o algoritmo de Mallat [RB98, Teo98].

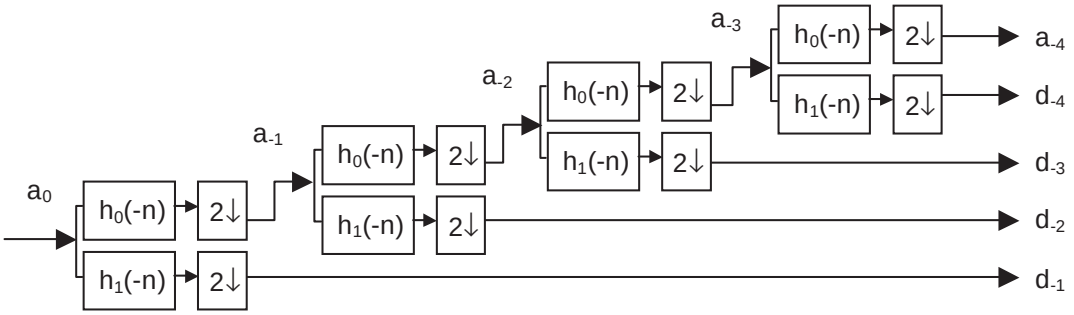


Figura 3.2: Transformada Wavelet Rápida (FWT) o Algoritmo de Mallat

Matemáticamente, las aproximaciones y detalles se pueden expresar por medio del operador diezmado ($\downarrow$) y la convolución con las funciones base escaladora (ϕ) y wavelet (ψ) de la siguiente forma [Teo98, RB98, Fli94]:

$$\begin{aligned} a_{-k}[m] &= \langle a_{-k+1}[m]\,,\, D_{2^k}\tau_m\phi \rangle \\ &= \downarrow a_{-k+1}[m] \star \widetilde{h}_0[m] = \sum_n a_{-k+1}[n] h_0[n-2m] \end{aligned} \tag{3.10}$$

y

$$\begin{aligned} d_{-k}[m] &= \langle a_{-k+1}[m]\,,\, D_{2^k}\tau_m\psi \rangle \\ &= \downarrow a_{-k+1}[m] \star \widetilde{h}_1[m] = \sum_n a_{-k+1}[n] h_1[n-2m] \end{aligned} \tag{3.11}$$

A partir del esquema de la Figura 3.2 se aprecia una característica importante de los detalles y las aproximaciones para una DWT, y en general del análisis multirresolución,

consistente en que la información correspondiente al detalle de mayor resolución, d_{-1}, se asocia a la región de frecuencia comprendida entre $\frac{\pi}{2} \leq \Omega \leq \pi$, d_{-2} brinda la información en el rango de $\frac{\pi}{4} \leq \Omega \leq \frac{\pi}{2}$, d_{-3} lo hace para $\frac{\pi}{8} \leq \Omega \leq \frac{\pi}{4}$, y así sucesivamente (Figura 3.3). En otras palabras, el aumento del número de niveles trae consigo un análisis más detallado de la información concerniente a baja frecuencia, y el conjunto filtro pasa-bajo/diezmador y filtro pasa-alto/diezmador, particiona a la mitad, o diádicamente, el espectro de la etapa anterior.

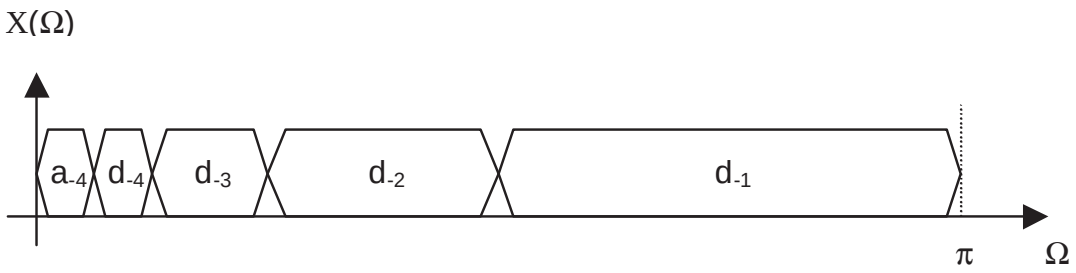


Figura 3.3: Particionamiento del espectro en una DWT

Otra de las variantes de la transformada wavelet y de reciente aplicación en el tratamiento digital de señales, son los paquetes wavelet (WP) [Wic94, RB98]. Estos pueden entenderse como una versión mejorada de la DWT, ya que la partición del espectro de la señal no se hace exclusivamente sobre la regiones de baja frecuencia, como se mostró en la Figura 3.3, sino también sobre las de alta frecuencia asociadas a los detalles. Esto se consigue filtrando y diezmando los detalles, tal como se ilustra en el ejemplo de un paquete wavelet de la Figura 3.4.

Es importante resaltar que, de todas las variantes de la Transformada Wavelet, la DWT ofrece el menor costo computacional, pues para la OCWT no existe una estructura rápida parecida al algoritmo de Mallat, y los paquetes wavelet, al derivarse de la DWT, tienen una complejidad ligeramente superior a la transformada wavelet discreta. Lo anterior puede analizarse mejor al calcular el número de operaciones multiplicación-acumulación (MAC) promedio que son necesarias para cada transformada.

En la FWT, el número de operaciones involucradas es igual a $N_h \times \left\{ N_x + \frac{N_x}{2} + \frac{N_x}{4} + ... \right\}$ y en el peor de los casos, para un número muy alto de niveles N_l, implica una complejidad de $\mathcal{O}(2N_hN_x)$, con N_x la longitud de la señal de entrada y N_h la longitud de la respuesta al impulso.

En los paquetes wavelet, el mejor de los casos se presenta cuando el árbol de descomposición corresponde a la forma piramidal del algoritmo de Mallat, y el peor cuando el árbol de descomposición es uniforme, es decir, todos los detalles y aproximaciones se subdividen hasta un cierto nivel N_l, proporcionando una partición uniformemente equiespaciada del espectro de la señal. De esta forma, el costo computacional oscila entre $\mathcal{O}(2N_hN_x)$ y $\mathcal{O}(N_lN_hN_x)$.

En lo que respecta a la OCWT, al asumir un muestreo uniforme en el tiempo, el

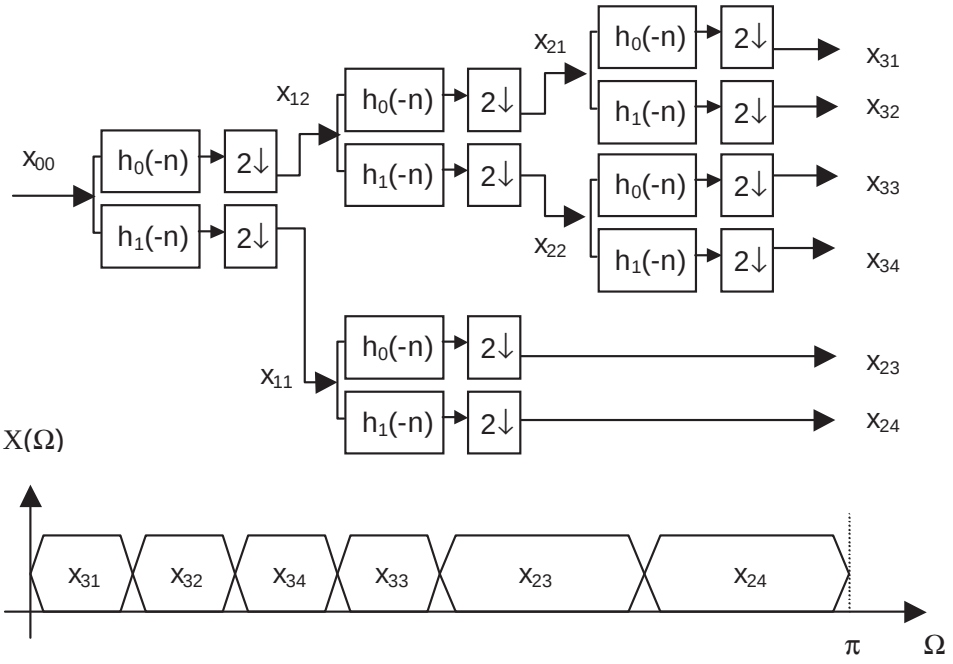


Figura 3.4: Ejemplo de un Paquete Wavelet

número de operaciones MAC promedio necesarias es del orden de $\mathcal{O}(N_l N_h N_x)$. Lo cual indica que el tiempo de ejecución de una OCWT crece linealmente con el aumento del número de niveles y la longitud de la señal de entrada, y se aproxima en el mejor de los casos, al costo del WP. Cuando la longitud de la respuesta al impulso es alta y comparable a N_x, la OCWT calculada por medio de la convolución directa en el tiempo (ec. 3.2) resulta ineficiente, por lo cual se recurre al método de la convolución rápida [PM92, LF94], que hace uso de la Transformada Rápida de Fourier (FFT), dando un costo computacional de $\mathcal{O}(N_l N_x log_2 N_x)$ [Teo98]. Estos esquemas de implementación serán abordados en las secciones 3.3 y 3.4.

3.2 Filtros QMF y Paquetes Wavelet

En la sección precedente se indicó que el cálculo de la DWT y los paquetes wavelet implica un proceso de filtrado y diezmado por un factor de dos, obteniéndose a la salida de un filtro pasa-bajo, las aproximaciones, y por el pasa-alto, los detalles. Para el proceso de síntesis de la señal, o lo que es equivalente, el cálculo de la DWT y WP inversa, se hace necesario invertir el proceso, por medio de la interpolación y posterior filtrado de los coeficientes de aproximaciones y detalles, tal como se indica en la Figura 3.5. Esta operación se expresa matemáticamente como:

$$\begin{aligned} a_{-k+1}[m] &= (\uparrow a_{-k}[m]) \star g_0[m] + (\uparrow d_{-k}[m]) \star g_1[m] \\ &= \sum_n g_0[n] a_{-k}[\frac{m-n}{2}] + \sum_n g_1[n] d_{-k}[\frac{m-n}{2}] \end{aligned} \tag{3.12}$$

siendo $g_0$un filtro pasa-bajo y $g_1$un filtro pasa-alto.

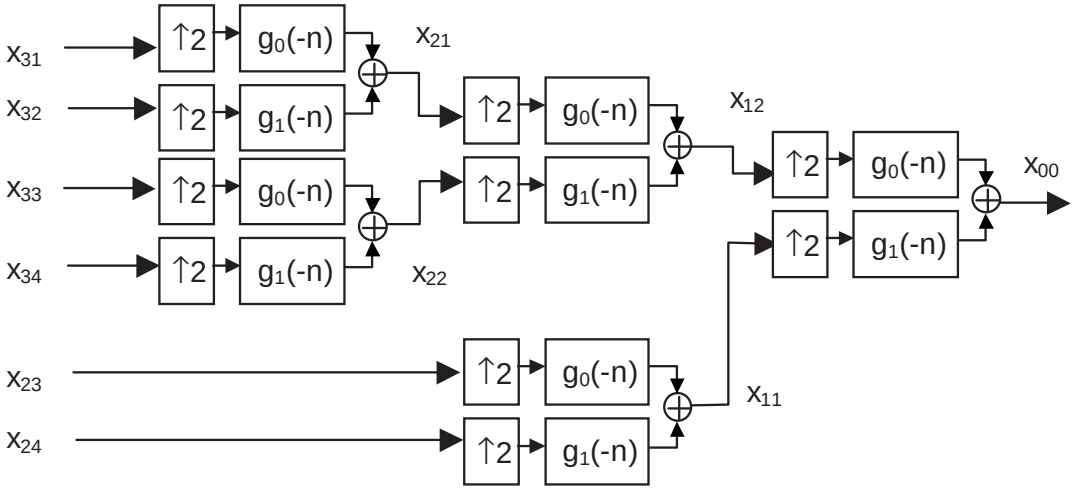


Figura 3.5: Paquete Wavelet de Síntesis para el árbol de descomposición de la Figura 3.4

Al igual que los filtros h_0y h_1, g_0y g_1 están relacionados con la funciones escaladora ϕ y wavelet ψ. Sin embargo, una relación más simple entre las cuatro respuestas al impulso, y ciertas propiedades de estos filtros, se pueden establecer más fácilmente por medio de la representación del paquete wavelet en forma de filtros de espejo en cuadratura (*QMF: Quadrature Mirror Filter*). Un filtro QMF de dos canales se muestra en la Figura 3.6. Nótese que la etapa de análisis y síntesis, tienen la misma estructura para el cálculo de cualquier nodo del paquete wavelet directo e inverso, respectivamente. En una estructura de filtrado QMF como la mostrada en la figura es necesario garantizar la invertibilidad del proceso, es decir, que la señal a la salida banco de filtros de síntesis sea una copia idéntica a la señal de entrada al banco de análisis, lo cual se consigue empleando filtros pasa-bajo (h_0 y g_0) y pasa-alto (h_1y g_1) con frecuencias de corte $\Omega_c = \frac{\pi}{2}$ y espectralmente complementarios[1], de allí el nombre de filtros de espejo en cuadratura. Esta invertibilidad del proceso, permite que el banco de filtros QMF calcule una transformada con funciones base ortogonales.

Para conseguirlo, las respuestas al impulso de los filtros QMF deben cumplir las condiciones de reconstrucción perfecta conjugadas, dadas por [Fli94]:

[1]la respuesta en frecuencia de $H_1(\Omega)$ es complementaria a $H_0(\Omega)$ y $G_1(\Omega)$ a $G_0(\Omega)$, lo que matemáticamente se representa por medio de $|H_0(\Omega)|^2 + |H_1(\Omega)|^2 = 1$

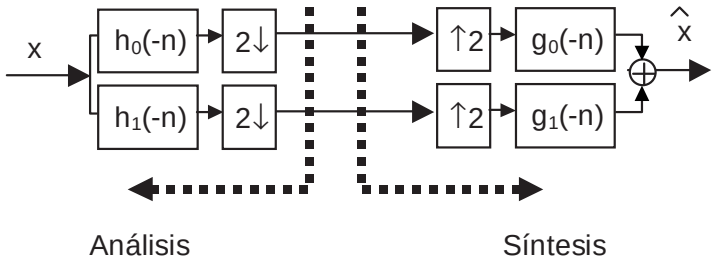


Figura 3.6: Filtro de espejo en cuadratura (QMF)

$$\begin{aligned} h_0[n] = h[n] &\Leftrightarrow H_0(z) = H(z) \\ h_1[n] = (-1)^{N-1-n} h[N-1-n] &\Leftrightarrow H_1(z) = z^{-(N-1)} H(-z^{-1}) \\ g_0[n] = 2h[N-1-n] &\Leftrightarrow G_0(z) = 2z^{-(N-1)} H(z^{-1}) \\ g_1[n] = 2(-1)^n h[n] &\Leftrightarrow G_1(z) = 2H(-z) \end{aligned} \tag{3.13}$$

Estas expresiones muestran que los cuatro filtros se derivan de un único filtro FIR pasa-bajo $h[n]$, causal y de longitud N. La respuesta al impulso de este filtro no es arbitraria, pues se deben garantizar también las siguientes relaciones espectrales [Fli94]:

$$H_0(z)G_0(z) + H_1(z)G_1(z) = 2z^{-k} \tag{3.14}$$

$$H_0(-z)G_0(z) + H_0(-z)G_1(z) = 0 \tag{3.15}$$

La ec. (3.14) implica que la señal a la salida del banco de filtro de síntesis sea una versión desplazada de la señal de entrada en k muestras, donde $k = N - 1$, y la ec. (3.15) garantiza la no presencia de efectos de aliasing. Por otra parte, las relaciones dadas en la ec. (3.13) traen consigo el hecho de que todos los filtros sean causales, que justifican el desplazamiento de fase de la señal de salida que muestra la ec. (3.14).

Otra expresión útil, que permite mostrar el carácter complementario de los filtros pasa-bajo y pasa-alto empleados en la estructura QMF resulta de la sustitución de las expresiones (3.13) en la ec. (3.14):

$$H^2(z) + H^2(-z) = 1 \tag{3.16}$$

3.3 Implementación de los Paquetes Wavelet (WP)

Es necesario precisar que la implementación de los algoritmos de este proyecto están enfocados al procesamiento por bloques, dado a que el sistema de adquisición de señales está basado en el esquema de entrada/salida por acceso directo a memoria (DMA), así

mismo, con el fin de generar algoritmos para procesamiento en tiempo real, las estructuras a plantear deben ser causales. Por lo anterior se decidió emplear el formalismo de los filtros QMF para el desarrollo de los algoritmos, pues éstos permiten obtener versiones causales de la transformada a costa de la presencia de un corrimiento de fase. Para la implementación de los paquetes wavelet se proponen dos esquemas, uno de ellos basado en el cálculo de células básicas y el otro en un esquema de filtros equivalentes o combinados.

3.3.1 Implementación por células básicas

Dado a que en el proceso de análisis, la estructura básica de descomposición está compuesta por un filtro pasa-bajo y pasa-alto y sus respectivos diezmadores, y para la síntesis, interpoladores y filtros pasa-bajo y pasa-alto, se escribieron dos rutinas que permiten una fácil adaptación del sistema a otros niveles y árboles de descomposición, denominadas Wave_StageD y Wave_StageI cuyos prototipos son:

```
Wave_StageD(
    entrada,          ▷vector entrada
    lentrada,         ▷longitud del vector entrada
    h0, h1,           ▷coeficientes de los filtros
    lh,               ▷longitud de coeficientes de los filtros
    aproximaciones,   ▷salida aproximaciones
    detalles          ▷salida detalles
)

Wave_StageI(
    aproximaciones,   ▷entrada de aproximaciones
    detalles          ▷entrada de detalles
    lentrada,         ▷longitud del vector entrada
    g0, g1,           ▷coeficientes de los filtros
    lh,               ▷longitud de coeficientes de los filtros
    salida            ▷vector de salida
)
```

La implementación de cada una de las etapas del banco de filtros wavelet se llevó a cabo usando la estructura polifásica [Fli94], con la cual se evita la realización de cálculos innecesarios en los bloques de filtrado-diezmado e interpolación-filtrado. La estructura polifásica consiste en particionar cada filtro $h[n]$ en D filtros con respuesta al impulso $h^{(k)}[n] = h[nD + k]$ $0 \leq k \leq D - 1$, donde D es el nivel de diezmado o interpolación[Fli94, PM92], conduciéndonos a una transformación de las células básicas de

cálculo en las estructuras mostradas en las Figuras 3.7 y 3.8. A partir de estos diagramas de bloques fueron escritas las funciones Wave_StageD y Wave_StageI cuyo código completo se presenta en el Apéndice A.

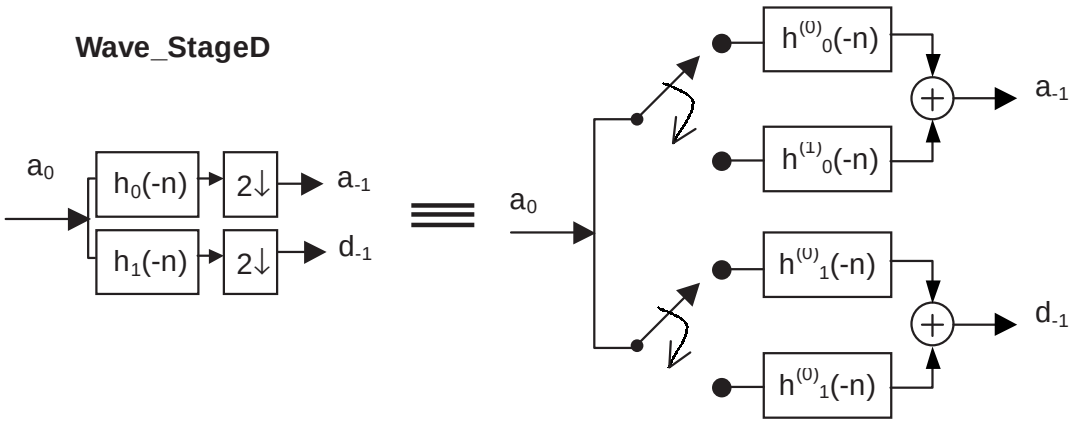


Figura 3.7: Célula básica para la descomposición en un paquete wavelet Función Wave_StageD.

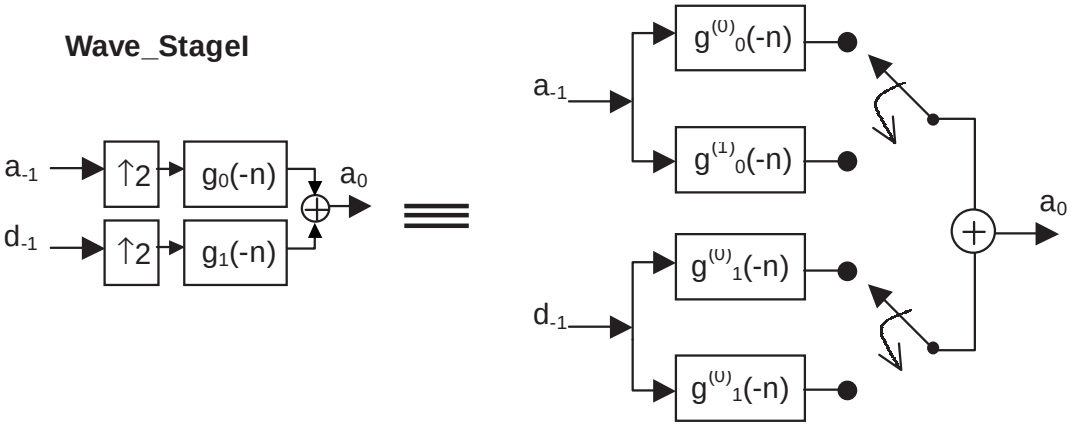


Figura 3.8: Célula básica para la síntesis en un paquete wavelet. Función Wave_StageI.

Dado a que el procesamiento de la señal de entrada se realiza a un bloque de datos capturado por DMA, para el cálculo de estas células básicas se hizo necesario asegurar una continuidad en la señal de entrada en el procesamiento de los bancos de filtros. Esto se consiguió empleando como condiciones iniciales para cada uno de los bancos de filtros wavelet muestras pasadas que habían sido entregadas por la etapa anterior. En particular se deben tener en cuenta las $N-1$ muestras pasadas, donde N es la longitud de la respuesta al impulso de los filtros asociados a la wavelet usada. La Figura 3.9

ilustra el esquema en la memoria de los bloques de entrada a cada una de las etapas del banco de filtros wavelet.

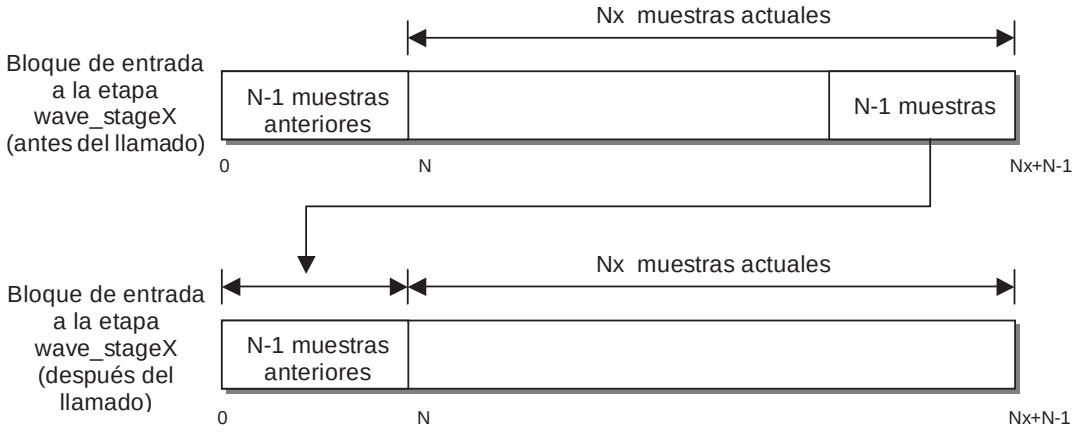


Figura 3.9: Esquema en memoria del vector de muestras a filtrar

Con lo expuesto anteriormente, la versión en pseudocódigo de las rutinas Wave_StageD y Wave_StageI son como siguen:

---

convolución(entrada, hn)

▷ Cálculo de la convolución orientada a bloque
salida←0
Para k←0 hasta longitud(hn)-1 en paso de 1
    salida← salida + entrada[-k]*hn[k]
retornar salida

Wave_StageD(entrada, lentrada, h0, h1, lh, aproximaciones, detalles)

▷ Cálculo de la estructura polifase
Para i ←0 hasta lentrada-1 en paso de 2
    ▷ Calcula las aproximaciones
    aproximación[i/2] ←convolución(entrada[i+lh-1], h0)
    ▷ Calcula los detalles
    detalle[i/2] ←convolución(entrada[i+lh-1], h1)
▷ Actualiza las condiciones iniciales
copiar lh-1 términos de &entrada[lentrada] a &entrada[0]

Wave_StageI(aproximaciones, detalles, lentrada, g0, g1, lh, salida)

▷ Cálculo de la estructura polifase
Para i ←0 hasta lentrada-1

```
        ▷ Calcula los cuatro filtros polifase
        t1 ←convolución( aproximaciones[i+lh/2-1], g_0^(0) )
        t2 ←convolución( aproximaciones[i+lh/2-1], g_0^(1) )
        t3 ←convolución( detalles[i+lh/2-1], g_1^(0) )
        t4 ←convolución( detalles[i+lh/2-1], g_1^(1) )
        ▷ Calcula las dos muestras de salida
        salida[2*i] ←t1 + t3;
        salida[2*i+1] ←t2 + t4;
    ▷ Actualiza las condiciones iniciales
    copiar lh-1 términos de &aprox.[lentrada] a &aprox.[0]
    copiar lh-1 términos de &detalles[lentrada] a &detalles[0]
```

Nótese que la función Wave_StageD no realiza el cálculo de los 4 filtros polifase indicados en la estructura de la Figura 3.7, esto se debe a que la suma de las señales entregadas por los filtros polifase ${h_{0,1}}^{(0)}$ y ${h_{0,1}}^{(1)}$ es equivalente a la convolución entre la señal de entrada y la respuesta al impulso $h_{0,1}$ cada $D = 2$ muestras de la señal de entrada. Esta es la razón por la cual el ciclo de cálculo de la función Wave_StageD se realiza en pasos de 2. Por otra parte, las convoluciones están orientadas al procesamiento por bloques y asumen que la respuesta al impulso es causal, de allí que los índices k del arreglo de entrada se tomen negativos y los de la respuesta al impulso, positivos.

Finalmente, para el cálculo del paquete wavelet directo (PWD) e inverso (IPW) se proponen las siguientes funciones descritas en pseudocódigo, que hacen uso de las células básicas Wave_StageD y Wave_StageI. El código completo en C de estas funciones se incluye en el apéndice A.

```
pwd ( árbol,  entrada, lentrada, salidas )
    salidas[0,0] ← entrada
    Para i←0 hasta árbol.número_niveles-2
        Para k←0 hasta 2^i − 1
            Si árbol.nodo[i, k] = 1
              Wave_StageD(
                salidas[i, k],
                lentrada / 2^i,
                árbol.h0,
                árbol.h1,
                árbol.lh,
                salidas[i+1, 2k],
                salidas[i+1, 2k+1]
              )
ipw ( árbol, entradas, salida, lsalida )
```

```
Para i←árhol.número_niveles-1 hasta 0
    Para k←0 hasta 2^i − 1
        Si árbol.nodo[i,k] = 1
          Wave_StageI(
            entradas[i+1, 2k],
            entradas[i+1, 2k+1],
            lsalida / 2^i,
            árbol.g0,
            árbol.g1,
            árbol.lh,
            entradas[i,k]
          )
salida ← entradas[0,0]
```

Dado que la funciones básicas Wave_StageD y Wave_StageI son la pieza fundamental de los algoritmos de cálculo del paquete wavelet, y como su versión definitiva se realiza sobre un procesador digital de señales TMS320C6701; se realizaron diferentes versiones en lenguaje C para estas funciones, haciendo uso de las sugerencias del manual de optimización del compilador CodeComposer[Ins00]. No se empleó codificación en lenguaje ensamblador, ya que en un proyecto anterior [MHAG+03a, MHAG+03b] se determinó que el compilador de C genera código lo suficientemente optimizado. Este DSP es de punto flotante, por lo cual los algoritmos se codificaron para operar con cantidades flotantes de 32bits (tipo *float*). Todas las optimizaciones están dirigidas hacia el aprovechamiento de las ocho unidades en paralelo con que cuenta el DSP'6701 y a la captación de operandos de 64bits en un único ciclo de reloj [Tex00]. Dichas estrategias de optimización consistieron en el alineamiento a 32bits de los punteros a los arreglos de entrada y coeficientes, el desenrollado de los ciclos y la captación eficiente de operandos de 64bits.

El desenrollado (*unrolled*) consiste en el reordenamiento de las instrucciones según su grado de dependencia y la asignación más eficiente de las unidades del procesador que ejecutarán cada una de las instrucciones de máquina, con el fin de permitir la ejecución en paralelo de más de una instrucción de máquina por periodo de reloj, reducir el número de instrucciones NOP que se insertan para sincronizar el *pipeline*. Este procedimiento se puede realizar en forma automática, mediante directivas específicas del compilador (#pragma MUST_ITERATE), o "ayudarle" a éste mediante la replicación manual de código. Esto último se consigue introduciendo redundancia de líneas de código de lenguaje C que se encuentran al interior de los ciclos, dejándole como tareas al optimizador, el reordenamiento y asignación de las unidades a las instrucciones.

La técnica de optimización con instrucciones de acceso a 64 bits consiste en que el DSP puede mover dos veces más rápido cantidades de 64 bits desde de la memoria hacia los registros internos del procesador, que no de cantidades de 32 bits, como el caso de los

tipos de datos float. Esto permite reducir considerablemente los tiempos de ejecución de los algoritmos, pues se optimizan los movimientos de datos necesarios para el cálculo de las convoluciones.

Cabe resaltar que el desenrollado manual, se puede conseguir calculando dos o cuatro elementos de salida por cada iteración del ciclo, o aprovechando el hecho de que todos los cálculos se realizan sobre cantidades de 32bits, y el DSP al contar con la capacidad de captación de operandos a 64bits, permite calcular dos iteraciones por ciclo en forma eficiente. Es importante indicar que el desenrollado de los ciclos solamente es posible conseguirlo si los ciclos se ejecutan un número de veces divisible por alguna potencia de dos.

De esta forma se realizaron diferentes versiones empleando el desenrollado automático y manual. Para la evaluación del desempeño se hicieron mediciones del tiempo promedio que tarda en la calcularse la transformada directa e inversa, empleado el conjunto de herramientas que posee el entorno de desarrollo CodeComposer para la medición del número de ciclos de máquina. La longitud del bloque de entrada al nodo base de la transformada directa se escogió de 8192 muestras, y se usó un árbol de descomposición uniforme (el peor de los casos según lo analizado en la sección 3.1) y 3 niveles de descomposición. En todos los casos, se forzó la alineación a 32bits de los buffers de entrada y coeficientes, y se usaron respuestas al impulso de 32 coeficientes. En la Tabla 3.1 se presentan los tiempos de ejecución para el DSP TMS320C6701 funcionando a una frecuencia de reloj de 133MHz.

A continuación se indican los lineamientos que se emplearon en cada una de las optimizaciones de la tabla. Cabe indicar que todas ellas se restringieron a la parte más crítica del cálculo del paquete wavelet, en el ciclo for más interno que calcula la convolución en las funciones wave_stagex.

- **Optimización 1**. Activación de la directiva #pragma MUST_ITERATE en el ciclo for, con el fin de que el optimizador desenrolle automáticamente el ciclo.

- **Optimización 2**. Empleo de la directiva #pragma MUST_ITERATE y duplicación del código manualmente. De esta forma se calculan dos iteraciones por ciclo, y la longitud de la respuesta al impulso debe ser como mínimo divisible por dos.

- **Optimización 3**. Empleo de la directiva #pragma MUST_ITERATE y uso de instrucciones de carga a 64bits para los coeficientes de las respuestas al impulso. En esta versión, la longitud de la respuesta al impulso debe ser divisible por dos.

- **Optimización 4**. Uso de la directiva #pragma MUST_ITERATE y carga a 64bits tanto para el vector de la señal de entrada y los coeficientes de las respuestas al impulso. Esta versión solamente funciona si la longitud de la respuesta al impulso es divisible por cuatro.

.

Tabla 3.1: Tiempos de Ejecución de las funciones PWD e IPW basada en la implementación de células básicas para un árbol uniforme, tres niveles de descomposición y una longitud de entrada de 1024 datos

Estrategia de optimización	PWD		IPW	
	Número de ciclos	Tiempo (ms)	Número de ciclos	Tiempo (ms)
Versión sin optimizar	32.279.000	242	29.065.000	218
Optimización 1	22.643.000	170	27.412.000	206
Optimización 2	23.054.000	173	26.849.000	201
Optimización 3	22.623.000	170	26.940.000	202
Optimización 4	15.023.000	113	19.971.000	150

Como se ilustra en la tabla 3.1, el desenrollado manual por medio de la replicación del código al interior del ciclo (optimización 2) no ofrece mejoras en la ejecución del algoritmo respecto al desenrollado automático, debido a que se generan demasiadas instrucciones de máquina con las cuales el optimizador es incapaz de establecer una dependencia, reordenamiento y asignación de unidades más eficiente que la versión desenrollada automáticamente. Sin embargo, el desenrollado manual de los ciclos por medio de instrucciones de carga a 64bits tanto para la señal de entrada como para los coeficientes, es la versión que ofrece mejores resultados (optimización 4). Es importante aclarar que la versión más optimizada solamente puede usarse si la longitud de la respuesta al impulso es divisible por cuatro. Los algoritmos en lenguaje C de las versión sin optimizar y la más optimizada se incluyen en el Apéndice A.

El hecho que se consigan eliminar cerca de siete millones de ciclos con esta última rutina, se explica por la reducción considerable del número de cargas de operandos necesarias en las funciones wave_stageX. Para entenderlo mejor, se presenta en la Figura 3.10, el esquema de cálculo de la función wave_stage1. En esta función, el ciclo for más interno que calcula las convoluciones, permite calcular simultáneamente 4 valores de salida en lugar de 2, como lo hace la versión sin optimizar. En esta figura, se asume que las respuestas al impulso g_0 y g_1 tienen una longitud de 12 coeficientes, a es el vector de entrada correspondiente a las aproximaciones y d a los detalles. Por ejemplo, para calcular la salida $sal[0]$ es necesario realizar el producto y acumulación de los siguientes términos:

$$\begin{aligned} sal[0] &= (a[0]G_0[1] + d[0]G_1[1]) + \\ &\quad (a[1]G_0[3] + d[1]G_1[3]) + \\ &\quad (a[2]G_0[5] + d[2]G_1[5]) + \\ &\quad ... \\ &\quad (a[5]G_0[11] + d[5]G_1[11]) \end{aligned}$$

	Primera Captación a 64bits		Segunda Captación a 64bits		Tercera Captación a 64bits		Última Captación a 64bits	
sal[0] =	$a[0]G_0[1]+$ $d[0]G_1[1]$	$a[1]G_0[3]+$ $d[1]G_1[3]$	$a[2]G_0[5]+$ $d[2]G_1[5]$	$a[3]G_0[7]+$ $d[3]G_1[7]$	$a[4]G_0[9]+$ $d[4]G_1[9]$	$a[5]G_0[11]+$ $d[5]G_1[11]$		
sal[1] =	$a[0]G_0[0]+$ $d[0]G_1[0]$	$a[1]G_0[2]+$ $d[1]G_1[2]$	$a[2]G_0[4]+$ $d[2]G_1[4]$	$a[3]G_0[6]+$ $d[3]G_1[6]$	$a[4]G_0[8]+$ $d[4]G_1[8]$	$a[5]G_0[10]+$ $d[5]G_1[10]$		
sal[2] =		$a[1]G_0[1]+$ $d[1]G_1[1]$	$a[2]G_0[3]+$ $d[2]G_1[3]$	$a[3]G_0[5]+$ $d[3]G_1[5]$	$a[4]G_0[7]+$ $d[4]G_1[7]$	$a[5]G_0[9]+$ $d[5]G_1[9]$	$a[6]G_0[11]+$ $d[6]G_1[11]$	
sal[3] =		$a[1]G_0[0]+$ $d[1]G_1[0]$	$a[2]G_0[2]+$ $d[2]G_1[2]$	$a[3]G_0[4]+$ $d[3]G_1[4]$	$a[4]G_0[6]+$ $d[4]G_1[6]$	$a[5]G_0[8]+$ $d[5]G_1[8]$	$a[6]G_0[10]+$ $d[6]G_1[10]$	

	1era Captación a 64bits		2da Captación a 64bits		3era Captación a 64bits		4ta Captación a 64bits		5ta Captación a 64bits		6ta Captación a 64bits	
G_0	0	1	2	3	4	5	6	7	8	9	10	11
G_1	0	1	2	3	4	5	6	7	8	9	10	11

Figura 3.10: Esquema de cálculo de la función wave_stageI para la versión más optimizada

Gráficamente se resalta en forma sombreada en la figura, las instrucciones que se calculan por primera vez en el ciclo for más interno de la función wave_stageI optimizada. Nótese que solamente son necesarias dos instrucciones de carga a 64bits para capturar simultáneamente cuatro elementos de los vectores de entrada a y d: $a[2]$, $a[3]$, $d[2]$ y $d[3]$; y cuatro instrucciones de carga a 64bits para los coeficientes g_0y g_1: $g_0[4]$, $g_0[5]$, $g_1[4]$, $g_1[5]$, $g_0[6]$, $g_0[7]$, $g_1[6]$ y $g_1[7]$. Así mismo, cada uno de los valores de entrada a y b captados a 64bits se emplean 4 veces en cada iteración, los coeficientes $g_{0,1}[4]$, $g_{0,1}[5]$, dos veces, mientras que los $g_{0,1}[6]$ y $g_{0,1}[7]$ solo una vez, pues en la siguiente iteración estos valores se usarán una vez más. Esta reutilización de las captaciones, es la que le otorga la alta eficiencia al algoritmo, pues en esta versión optimizada, son necesarias tan solo 6 captaciones a 64bits por iteración, frente a 32 captaciones a 32bits que requiere la versión sin optimizar para ejecutar la misma cantidad de MACs.

Es importante resaltar que el ciclo for más interno solamente se puede ejecutar $Nh/4 - 1$ veces, ya que en la primera captación de las señales de entrada solamente es posible ejecutar 12 MACs, y con la última captación tan solo 4 (ver Figura 3.10).

3.3.2 Implementación en filtros equivalentes

Otra forma eficiente y simple de calcular un nodo terminal de un árbol de descomposición wavelet es por medio del filtrado con una respuesta al impulso $f_{lm}[n]$ y un diezmador por un factor de 2^l, con l el número del nivel y m la posición en dicho nivel, en lugar de la aplicación iterativa de los procesos de filtrado y diezmado por dos que sugiere la implementación en células básicas descritas en la sección anterior (Figura 3.11b). El filtro $f_{lm}[n]$, al que denominamos *filtro equivalente*, es el resultado de combinar los filtros h_0, h_1 y los diezmadores según la secuencia de ramas que dan lugar al nodo terminal

del árbol, por ejemplo, como se muestra en la Figura 3.11a, el filtro $f_{2,3}$, es el resultado de combinar los filtros h_1, el diezmador y h_0; $f_{2,4}$, en cambio combina h_1, el diezmador y h_1. Para el caso de la síntesis, la estructura equivalente se consigue por medio de un proceso de interpolación por un factor 2^l y el subsecuente filtrado (Figura 3.11c).

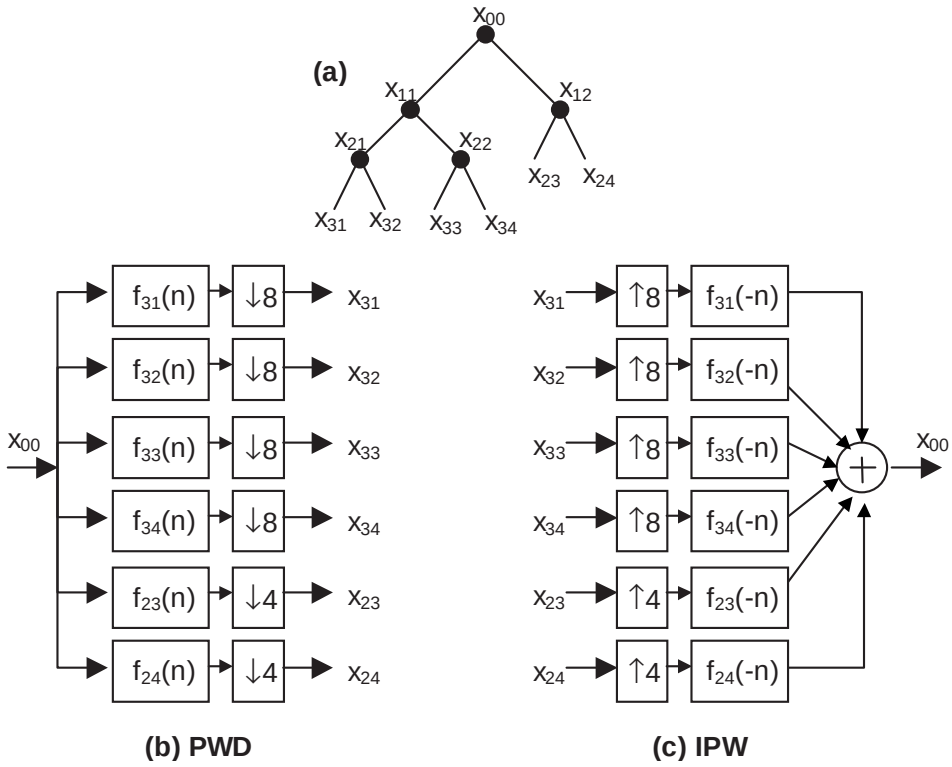


Figura 3.11: Estructura de cómputo del paquete wavelet (a): (b) directo y (c) inverso por medio de filtros equivalentes

Cada uno de los filtros equivalentes que requieren los nodos terminales en una descomposición, se calculan por medio de la iteración recursiva de las siguientes ecuaciones en diferencia:

$$f^{(l)}[n] = \sum_{k=0}^{N-1} h^{(l-1)}[k] f^{(l-1)}[n - 2^{l-1}k] \quad l \geq 2 \tag{3.17}$$

$$f^{(1)}[n] = h^{(0)}[n] \tag{3.18}$$

con $0 \leq n \leq (2^l - 1)(N - 1)$, $l \geq 2$, N la longitud de las secuencias $h^{(k)}$, y $h^{(0)}, h^{(1)}, h^{(2)}, ...$, las respuestas al impulso de los filtros QMF pasa-alto y pasa-alto necesarios para llegar desde la base del árbol ($l = 0$) hasta el nodo terminal ubicado en el nivel l. Los superíndices indican la posición del nivel.

Por ejemplo, en el esquema de la Figura 3.11b, para determinar el filtro equivalente f_{23} es necesario iterar dos veces la expresión 3.17, tomando $l = 2$, y los filtros $h^{(0)} = h_1$ y $h^{(1)} = h_0$.

Para la síntesis, la respuesta al impulso está dada por:

$$f^{(-l)}[n] = \sum_{k=0}^{N-1} h^{(l-2)}[k] f^{(-l+1)} \left[\frac{n-k}{2}\right] \qquad l \geq 2 \tag{3.19}$$

y,

$$f^{(-1)}[n] = h^{(1)}[n] \tag{3.20}$$

con $0 \leq n \leq (2^l - 1)(N - 1)$ y las mismas condiciones para $h^{(l)}$ y $f^{(l)}$ de la descomposición. La demostración de estas expresiones se incluye en el Apéndice B.

En apariencia, esta estructura sugiere un menor número de cálculos, por lo que se propuso llevar a cabo una implementación en lenguaje C de dicho esquema, y hacer uso de las mismas técnicas de optimización empleadas en la sección anterior, como son el desenrollado de ciclos y cargas a 64bits de los vectores de entrada y coeficientes, con el fin de establecer un análisis comparativo del desempeño de estos algoritmos y el basado en células básicas. Es importante resaltar que este esquema de cálculo favorece la implementación en versiones hardware del paquete wavelet, dado que hace posible explotar las características de ejecución en paralelo que ofrecen los dispositivos FPGA (*Field Programmable Gate Array*) de nueva generación. Este aspecto está siendo abordado actualmente por el grupo de investigación en un proyecto consistente en la implementación en hardware del sistema de compresión de voz que se propone en este proyecto[MHGQO04].

Respecto al costo computacional de esta implementación, debemos recordar que la implementación de los WP por medio de las células básicas tiene un tiempo de ejecución promedio que oscila entre $\mathcal{O}(2N_hN_x)$, para un árbol de descomposición igual a la DWT, y $\mathcal{O}(N_lN_hN_x)$, para un árbol de descomposición uniforme, con N_h la longitud de la respuesta al impulso de los filtros h_0y h_1, N_x la longitud del bloque de entrada y N_l el número de niveles de descomposición. Para la estructura de implementación en filtros equivalentes, el peor de los casos se presenta también cuando el árbol de descomposición es uniforme, lo cual implica por cada nodo terminal un total de $N^{(N_l)} \times \frac{N_x}{2^{N_l}}$ operaciones, y como son 2^{N_l} nodos terminales, se tiene un costo de $\mathcal{O}(N^{(N_l)}N_x) = \mathcal{O}(\,[(2^{N_l} - 1)(N_h - 1)]N_x\,) = \mathcal{O}(\,(2^{N_l} - 1)N_hN_x - (2^{N_l} - 1)N_x\,)$. De esta forma, desde el punto de vista del costo computacional, esta estructura implica una mayor cantidad de operaciones que la basada en células básicas, sin embargo, merece la pena analizarla desde el punto de vista de implementación en un DSP, pues implica menor cantidad de movimientos de datos y posibilidad de sacar el mejor provecho a la optimización por desenrollado de ciclos.

Los algoritmos que se desarrollaron para el cálculo de cada nodo terminal están basados en la estructura polifase de un diezmador por un factor $D = 2^l$, para el caso

de la descomposición, y en un interpolador polifásico con factor de interpolación $I = 2^l$, para la síntesis. Estos algoritmos son similares a los presentados anteriormente para las rutinas Wave_StageD y Wave_StageI. La notación en pseudocódigo de estos algoritmos, cuyos programas en extenso en lenguaje C se incluyen en el Apéndice C, son como siguen:

---

diezmadorD(entrada, lentrada, hn, lh, D, salida)

▷ Calcula los filtros polifase
j ←0
Para i ←0 hasta lentrada-1 en paso de D
    salida[j] ←convolución(entrada[i], hn)
    j ←j + 1
▷ Actualiza las condiciones iniciales
copiar lh-1 términos de &entrada[lentrada] a &entrada[0]

inrterpoladorI(entrada, lentrada, gn, lh, I, salida)

Para j ←0 hasta lentrada-1
    ▷ Calcula los I filtros polifase
    Para i ←0 hasta I-1
        salida[j*I+i] ←convolución(entrada[j], $g_n^{(i)}$)
▷ Actualiza las condiciones iniciales
copiar lh-1 términos de &entrada[lentrada] a &entrada[0]

Los desempeños de esta implementación de los paquetes wavelets se presentan en la Tabla 3.2, al igual que el tiempo de ejecución de la versión más optimizada de la implementación basada en células básicas con la mismas condiciones de cálculo: $N_x = 1024$, $N_h = 32$, $N_l = 3$ y estructura de árbol uniforme. Las diferentes optimizaciones en la tabla se refieren a:

- **Optimización 1**. Se empleó la directiva #pragma MUST_ITERATE en el ciclo for más interno con el fin de forzar al optimizador para que desenrrolle automáticamente el ciclo.
- **Optimización 2**. Empleo de la directiva #pragma MUST_ITERATE y el código del ciclo for más interno fue replicado 4 veces en forma manual. De esta forma se calculan cuatro iteraciones por ciclo, lo que implica que la longitud de la respuesta al impulso debe ser divisible por cuatro.
- **Optimización 3**. Uso de la directiva #pragma MUST_ITERATE y carga a 64bits tanto para el vector de la señal de entrada y los coeficientes de las respuestas al impulso. En esta versión, el código del ciclo for más interno fue replicado cuatro veces. Esta versión solamente funciona si la longitud de la respuesta al impulso es divisible por cuatro.

Tabla 3.2: Tiempos de Ejecución de las funciones PWD e IPW basada en la implementación de filtros equivalentes para un árbol uniforme, tres niveles de descomposición y una longitud de entrada de 1024 datos

Estrategia de optimización	PWD		IPW	
	Número de ciclos	Tiempo (ms)	Número de ciclos	Tiempo (ms)
Versión sin optimizar	172.001.000	1290	273.529.000	2051
Optimización 1	171.438.000	1286	270.948.000	2032
Optimización 2	171.683.000	1288	99.469.000	746
Optimización 3	86.592.000	649	80.021.000	600
Implementación por células básicas	15.023.000	113	19.971.000	150

Nótese que solamente la versión que hace uso de instrucciones de carga de 64bits y la cuadruplicación de código, es la más eficiente de todas, pero no tanto como lo es la implementación por células básicas. Esto se debe principalmente al hecho de que las respuestas al impulso de los filtros equivalentes f_{lm} no crece linealmente con el aumento del número de niveles sino en forma diádica, ya que $\mathcal{O}(\,[(2^{N_l}-1)(N_h-1)]N_x\,)$, así mismo, la estructura de diezmado e interpolación por un factor potencia de dos, no permite que se puedan reutilizar las cargas del vector de entrada tan eficientemente como se lo podía hacer en la estructura de células básicas que se presentó en la sección anterior. Esto puede corroborase al calcular el número de operaciones MAC que involucra cada estructura:

$$\begin{aligned} N_{MAC}(celulas\,basicas) &= N_l N_h N_x = 98304 \\ N_{MAC}(filtro\,equivalente) &= [(2^{N_l}-1)(N_h-1)]N_x = 222208 \end{aligned}$$

teniéndose que el tiempo de ejecución de la implementación por filtros equivalente debería crecer teóricamente en proporción a $N_{MAC}(celulas)/N_{MAC}(celulas\,basicas) = 2.26$ veces, sin embargo, el tiempo de cómputo fue realmente cerca de cuatro veces superior.

Cabe resaltar que la versión más optimizada la estructura en filtros equivalentes asume que la longitud de la respuesta al impulso del filtro combinado es divisible por cuatro. Esto generalmente no se garantiza en todos los casos, de allí que sea necesario realizar una extensión de la respuesta al impulso a una longitud apropiada, mediante la adición de ceros en los últimos elementos de la respuesta al impulso. Esto conlleva a un número más alto de instrucciones MAC que las calculadas teóricamente.

3.4 Implementación de la Transformada Wavelet Sobre-completa (OCWT)

Como se indicó en la sección 3.1, la Transformada Wavelet Sobre-completa (OCWT) consiste en la discretización de la Transformada Wavelet en Tiempo Continuo (CWT), y se puede calcular por medio de la aplicación de un banco de filtros de la forma (ec. 3.2):

$$F(s,\tau) = f(\tau) \star \psi_s(-\tau) \tag{3.21}$$

con $\psi_s(t)$ la versión escalada de la wavelet madre y $f(\tau)$ la señal a descomponer, y cada filtro tiene una respuesta al impulso igual a $\psi_s(-t)$. Es importante resaltar la necesidad de introducir redundancia en la información, que facilite el proceso de cálculo de la transformada inversa, para ello se eligen escalas de la forma $s_m = a_0^{-m}$ con $m \in \mathbb{Z}$.

Por otra parte, la transformada wavelet sobre-completa inversa (IOCWT) se puede calcular a partir de la representación en marcos de la OCWT [Teo98], sin embargo, los algoritmos resultantes son computacionalmente muy costosos y exigen la existencia de la inversa de la matriz de correlación de la función wavelet madre. Como alternativas de solución, de menor precisión pero de factible implementación, está la discretización de la expresión para la Transformada Wavelet Continua Inversa [Teo98]:

$$\hat{f}(\gamma) = C^{-1} \sum_m \hat{F}(s_m,\tau)\hat{\psi}_{s_m}(\gamma) \tag{3.22}$$

con $\hat{f}(\gamma)$, $F(s_m,\tau)$, y $\hat{\psi}_{s_m}(\gamma)$ las transformadas de Fourier de: la señal, coeficientes de la transformada wavelet, y wavelet madre respectivamente, y C calculado a través de:

$$C = \sum_m s_m \left|\hat{\psi}_{s_m}(\gamma)\right|^2 \tag{3.23}$$

Desde el punto de vista de implementación, y teniendo en cuenta que en la práctica el procesamiento de la información se realiza por bloques, a un flujo continuo de datos capturados por medio de DMA, la OCWT (ec. 3.21) se puede calcular fácilmente mediante el empleo del algoritmo de la convolución rápida, basada en la FFT, haciendo uso de la técnica de solapamiento y almacenamiento[2] [PM92]; en cambio para la IOCWT, se hace uso de la ec. (3.22), implementada por medio de la FFT, y para garantizar un procesamiento por bloques se emplea la técnica de solapamiento y suma[3] [PM92].

Respecto a la complejidad computacional, se indicó en la sección 3.1, que la OCWT implementada por medio de la convolución en el tiempo, dada por la ec. (3.21) implica $\mathcal{O}(N_l N_h N_x)$ operaciones, y su versión mejorada por medio de la FFT, $\mathcal{O}(N_l N_x log_2 N_x)$.

[2] los últimos $N_\psi - 1$ datos de cada bloque de entrada se almacenan para ser introducidos como los primeros datos del siguiente bloque y de la solución se descartan los primeros $N_\psi - 1$ datos

[3] los últimos $N_\psi - 1$ datos de cada solución se suman a la siguiente trama de salida

Se indicó también, que la estructura basada en FFTs es útil solamente en el caso en el que N_h se hace comparable a N_x, de allí que resulte importante realizar un análisis comparativo entre estas dos estructuras para longitudes de la respuesta al impulso y entrada semejantes y su implementación sobre un DSP.

Para el cálculo de la transformada inversa (IOCWT), la ec. (3.22) sugiere la existencia de un algoritmo basado en convoluciones en el tiempo para su cálculo, este algoritmo, al igual que el de la IOCWT calculado por medio de la FFT, tienen costos computacionales idénticos al los de la OCWT.

Los algoritmos en pseudocódigo para el cálculo de la OCWT e IOCWT por medio de la transformada de Fourier se describen a continuación:

OCWT(entrada, lentrada, psi, lpsi, escalas, lescalas, salida)
▷ Extensión de la señal de entrada

TFentrada[0:NFFT-lentrada-1] ←ocwt_pasadas;
TFentrada[NFFT-lentrada:NFFT-1]←entrada;
TFentrada←FFT(TFentrada, NFFT)

▷ Cálculo de la salida en cada escala

Para i ←0 hasta lescalas-1

TFpsi ←FFT(escalar(psi, lpsi, escalas[i]))
TFsalidai←IFFT(TFentrada * TFpsi)
salida[i]←TFsalidai[lpsi:NFFT-1]

▷ Actualiza las condiciones iniciales

copiar NFFT-lentrada términos de &entrada[2*lentrada-NFFT-1)] a ocwt_pasadas

IOCWT(entrada, lentrada, psi, lpsi, escalas, lescalas, salida)

TFsalida[0:NFFT-1]←0
Para i ←0 hasta lescalas-1

▷ Calcula los transformadas de Fourier intermedias
TFcoef[0:lentrada-1]←entrada[i]
TFcoef[lentrada:NFFT-1]←0
TFcoef←FFT(TFcoef)
TFpsi ←FFT(escalar(psi, lpsi, escalas[i]))
TFsalida ←TFsalida + TFcoef×TFpsi

▷ Finaliza el cálculo de la IOCWT
salidat ←IFFT(TFsalida)
salida[0:NFFT-lentrada-1]←salidat[0:NFFT-lentrada-1] + iocwt_pasadas
salida[NFFT-lentrada:lentrada-1]←salidat[NFFT-lentrada:lentrada-1]
▷ Actualiza las condiciones iniciales

copiar NFFT-
lentrada términos de &salidat[lentrada] a iocwt_pasadas

El algoritmo de la OCWT se puede resumir como el cálculo de lescalas convoluciones rápidas entre la señal de entrada y la versión escalada de la wavelet madre en la respectiva escala. En una convolución rápida, la longitud de la FFT debe cumplir la condición $N_{FFT} \geq N_x + N_h - 1$[PM92, LF94], con el fin que asegurar que el algoritmo calcule una convolución lineal y no circular. Como se indicó anteriormente, en este algoritmo se hace uso de la técnica de segmentación de solapamiento y almacenamiento, la cual explica el por qué de la extensión de la señal de entrada, anteponiendo muestras pasadas, que se realiza al inicio del algoritmo, y el descarte de los primeros $N_h - 1$ datos cuando se almacenan los coeficientes de descomposición en los vectores salida[i]. Nótese que si se hubiera empleado la segmentación por solapamiento y suma, serían necesarios una mayor cantidad de cálculos y espacio en memoria, pues por cada una de las escalas habría sido necesario almacenar las muestras de salida no usadas, y se aumentaría el número de operaciones por bloque debido a las sumas adicionales que involucra dicho esquema.

En el caso de la IOCWT, la técnica de segmentación más eficiente y simple de implementar, en este caso, es solapamiento y suma, pues solamente se hace necesario almacenar los últimos datos de salida no usados que entrega la IFFT.

Dado a que los algoritmos de cómputo de la OCWT e IOCWT presentados, muestran una alta dependencia del algoritmo de la FFT, se realizaron diferentes optimizaciones a esta última rutina, teniendo en cuenta diferentes aspectos de optimización:

1. La función wavelet base de cada escala es fija, lo cual permite precalcularla. De esta forma, y dado a que tanto en la OCWT como la IOCWT se hace necesario calcular la FFT de la función wavelet base, los vectores realmente precalculados en la implementación final son las transformadas discretas de Fourier de la función base de cada escala.

2. Las transformadas rápidas de Fourier involucradas en el cálculo de la OCWT e IOCWT asumen datos de entrada completamente reales, lo cual permite emplear un algoritmo eficiente de la FFT para magnitudes de entrada/salida completamente reales, algoritmo conocido con el nombre de descomposición trigonométrica [Emb99].

3. El DSP sobre el cual se implementó el sistema, un TMS320C6701, no posee instrucciones especializadas para *bit-reversal*, por lo cual fue necesario una rutina optimizada que se puede encontrar en la Ref. [Emb99].

4. La FFT implica el llamado de las funciones *sin* y *cos* de la librería matemática *math.h*, las cuales consumen demasiados ciclos de reloj, por lo cual, se decidió generar una tabla precalculada para dichas funciones.

5. Con el fin de aprovechar la capacidad de ejecución en paralelo de las instrucciones del DSP, se hizo necesario desenrollar los ciclos, por medio de la escritura redundante de líneas de código de lenguaje C.

Para este último aspecto, se analizaron dos estrategias de codificación para la FFT: cálculo de mariposas de 2 puntos al inicio de la rutina y la posterior mezcla de FFTs desenrollando manualmente el ciclo por medio de la duplicación de código; y modificación de la técnica anterior a partir del cálculo inicial de mariposas de 4 puntos en lugar de mariposas de 2 puntos. Así mismo, se adaptaron dos métodos de cómputo diferentes para la FFT real indicados en las Refs. [Emb99] y [PTVF92], haciendo uso de las estrategias de optimización 4 y 5.

El desempeño de estos algoritmos, sobre el DSP'6701 a 133MHz, para un vector de entrada de 1024 datos se presentan en la Tabla 3.3. Para todos los casos se emplearon tablas precalculadas de las funciones seno y coseno, y se indican únicamente las versiones optimizadas obtenidas por medio del desenrollado manual de los ciclos. Así mismo, para el algoritmo de la Ref. [Emb99], denominado reconstrucción trigonométrica, se emplearon los algoritmos de la FFT compleja indicados en la fila 1 y 2 de la tabla.

Tabla 3.3: Tiempos de Ejecución de las diferentes optimizaciones de la FFT

	Algoritmo	PWD	
		Número de ciclos	Tiempo (ms)
1	FFT compleja. Mariposas iniciales de 2 puntos	2.245.000	16.8
2	FFT compleja. Mariposas iniciales de 4 puntos	2.055.000	15.4
3	FFT real adaptada de la Ref. [Emb99]. Mariposas iniciales de 2 puntos	1.395.000	10.5
4	FFT real adaptada de la Ref. [Emb99]. Mariposas iniciales de 4 puntos	1.302.000	9.8
5	FFT real adaptada de la Ref. [PTVF92]	618.000	4.6

De la tabla se concluye que la versión que mejor saca provecho de los recursos del DSP es la FFT calculada para secuencias reales adaptada de la Ref. [PTVF92] por medio del empleo de tablas precalculadas para las funciones seno y coseno, y el desenrollado de ciclos.

Finalmente, con el fin de establecer una comparación entre los dos esquemas de cómputo de la OCWT, la forma basada en FFT y la que emplea convolución directa en el tiempo, se midieron los tiempos promedio de ejecución de la OCWT e IOCWT, para un vector de datos de entrada de 1024 puntos e igual longitud para la respuesta al

impulso[4], y un conjunto de 64 escalas. Estos resultados se relacionan en la Tabla 3.4.

Tabla 3.4: Tiempos de Ejecución de las diferentes versiones de la OCWT e IOCWT. $N_x = N_h = 1024$ y $N_{escalas} = 64$

Algoritmo	OCWT		IOCWT		Tiempo promedio por escala (ms)
	Número de ciclos	Tiempo (ms)	Número de ciclos	Tiempo (ms)	
Implementación basada en FFT	$195{,}7\times10^6$	1.467	$210{,}1\times10^6$	1.576	23
Implementación basada en convolución en el tiempo	1922×10^6	14.415	1.929×10^6	14.469	225

A partir de la tabla es evidente, la alta eficiencia que ofrece la implementación por medio de la transformada de Fourier para longitudes similares de los vectores de entrada y coeficientes. Por esta razón, la implementación basada en FFT será la técnica empleada para el cómputo de la OCWT e IOCWT que se emplea en algunas de las técnicas de comunicación que se proponen en este trabajo, y su código en lenguaje C se incluye en extenso en el apéndice D.

[4]con estas condiciones la longitud de la FFT es de 2048 datos

Capítulo 4

Sistema de compresión de voz

4.1 Generalidades de los sistemas de compresión de voz

La compresión de información es el proceso de reducción del volumen de datos necesarios para representar la información, sin perder en la nueva representación la legibilidad y la calidad. Esta reducción no se consigue suprimiendo datos al azar, sino eliminado la redundancia, entidad matemática cuantificable, que está presente en los datos[Sal01]. Existen dos esquemas básicos para la eliminación de la redundancia: la *compresión sin pérdidas* (*lossless*) y la *compresión con pérdidas* (*lossy*). El primer tipo es reversible, razón por la cual se emplea en los compresores de datos comerciales, mientras que en el segundo, se elimina información irrelevante basándose en la respuesta perceptiva de los sentidos, consiguiendo de esta forma mayores tasas de compresión que los sistemas sin pérdidas[Sal01, RY01]. En este último esquema se basan los compresores de audio, voz y vídeo.

Respecto a los sistemas de compresión de voz, los métodos de compresión más difundidos emplean la codificación lineal predictiva (LPC: *Linear Prediction Coding*), la codificación sub-banda o la transformada discreta del coseno (DCT: *Discrete Cosine Transform*)[RY01, AM99]. La codificación lineal predictiva es ampliamente usada en sistemas de telefonía celular tales como CELP (*Code Excited Linear Prediction*), VSELP (*Vector Sum Excited Linear Predictive*) y GSM (*Global System for Mobile*). Estas técnicas adaptativas permiten conseguir buenas tasas de compresión manteniendo un buen compromiso entre razón de compresión y la calidad de la voz, pues se fundamentan en el almacenamiento de los parámetros de un modelo, típicamente el tracto vocal, que representa la señal[DHP00]. Por otra parte, ciertos autores[NRPS03, HERES96, Cui03, GK94, GLOB95, SAA98, AM99] han mostrado la posibilidad de emplear, para la codificación sub-banda, la Transformada Wavelet, mostrando que ofrece mejores resultados que la DCT.

Entre los métodos de compresión de voz por wavelets se destacan dos tipos: los basados en la DWT[NRPS03, SAA98, GLOB95, Cui03] y los que hacen uso de los

paquetes wavelet[HERES96, AM99]; estos últimos ofrecen mejor calidad que los basados en la DWT[HERES96]. Para el proceso de codificación de los coeficientes wavelet se suele emplear los códigos Huffman o técnicas similares.

La codificación de voz que se propone en este trabajo se basa en un sistema híbrido que combina la transformación por paquetes wavelet con la técnica de codificación lineal predictiva. Se emplean los paquetes wavelet debido a que en comparación con la DWT ofrecen una mayor flexibilidad, dado a que permiten una descomposición subbanda completa, permitiendo escoger una forma de árbol que contenga los coeficientes más relevantes y necesarios para la representación de la señal.

4.2 Sistema propuesto para la compresión de señales de voz

En esta sección se describen los diferentes componentes del sistema de compresión de señales de voz desarrollado, el cual hace uso de los paquetes wavelet. Como se comentó en la sección *3.1*, los paquetes ofrecen la versatilidad de particionamiento del espectro en diferentes subbandas, que pueden adaptarse según las características de la señal. Adicionalmente, existe un método automático para el cálculo de la estructura del árbol que permite codificar la señal en su mínima representación. Esta técnica conocida con el nombre de método de la selección de la mejor fue propuesta por Coifman y Wickerhauser[Wic94, CW92].

El sistema en diagrama de bloques, que se propone en este trabajo e ilustrado en la Figura 4.1, consta de cuatro etapas que son: adquisición, descomposición, codificación y cuantización.

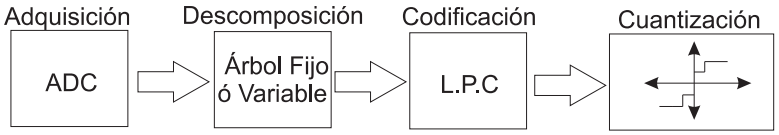


Figura 4.1: Esquema propuesto para la compresión de señales de voz

Para la adquisición y reconstrucción de la señal, se emplea una frecuencia de muestreo de $8kHz$, debido a que el ancho de banda del espectro de la voz está contenido en una región aproximada a $4kHz$. La captura está orientada a bloques haciendo uso del controlador DMA, y la técnica de doble buffer. Como ADC y DAC se emplea el CODEC presente en la tarjeta de desarrollo EVM TMS320C6701 de Texas Instruments.

La señal de voz, una vez capturada, se descompone realizando transformaciones por paquetes wavelet, empleando la estructura de descomposición y las rutinas optimizadas que se presentaron en la sección *3.3.*

Con el fin de valorar dos estrategias de compresión, se proponen dos esquemas, uno basado en árbol estático y otro en árbol dinámico. El primero, es útil en aplicaciones en

las cuales se deba garantizar una tasa de compresión fija que permita una transmisión a una razón constante de bits, mientras que en el segundo esquema, la tasa de compresión es variable y depende de la señal en particular.

4.2.1 Estructura de árbol estático

En el sistema de compresión basado en árbol estático, la forma del árbol de descomposición permanece fija e inalterable durante todo el proceso de compresión. Para determinar dicha estructura se realizaron una serie de pruebas que consistieron en el análisis de 10 voces diferentes (5 hombres y 5 mujeres), pronunciando palabras que contienen una gran cantidad de vocales y fonemas sordos. Las voces se grabaron a una frecuencia de muestreo de $8kHz$, y se particionaron en bloques de 1024 datos, que fueron posteriormente analizados con el *GUI* del *toolbox Wavelet* de Matlab. En estudios anteriores se concluyó que las funciones daubechies db8 y db4 son las más apropiadas para el análisis de señales de voz, debido a su similitud con la mayoría de los fonemas *[MHAG+03b, MHAG+03a].*

Se realizó la estimación del mejor árbol para cada uno de los bloques de la señal de prueba, con diferente wavelet madre (*db4* y *db8*). Como los bloques de entrada son de 1024 elementos, el máximo nivel de profundización usado fue de 10. Los resultados mostraron que el árbol de descomposición que más se repite es el de la Figura 4.2, encontrándose en más del 15% de los bloques analizados. Este árbol de descomposición es similar al reportado en la literatura para análisis sub-banda de señales de voz que reportan otros autores[DHP00].

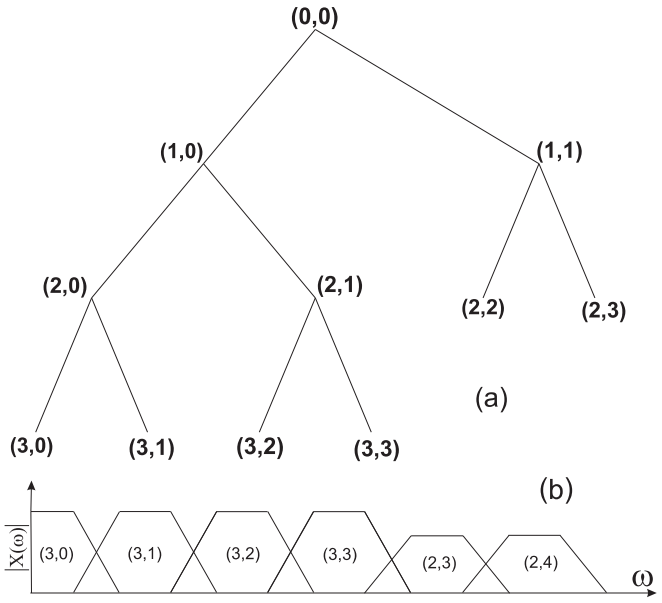


Figura 4.2: (a) Mejor árbol, (b) Distribución en bandas de frecuencia.

4.2.2 Estructura de árbol dinámico

Por otra parte, en la estructura dinámica, el cálculo del WP se realizó con un árbol de descomposición wavelet variable, en otras palabras, para cada segmento de voz capturado se determina su mejor representación o mejor árbol. Esto se logra empleando el algoritmo de Coifman y Wickerhauser[Wic94] para la selección de la mejor base. El algoritmo consiste en un filtrado adaptativo en el que el árbol de descomposición va cambiando en función del comportamiento temporal de la señal. Desde el punto de vista de calidad en la reconstrucción, este esquema conduce a mejores resultados que un árbol estático, debido a que el algoritmo busca la representación más óptima de la señal, aunque implica mayores requerimientos de hardware.

El algoritmo de la selección de la mejor base emplea una función de coste, típicamente la entropía, definida para secuencia X_i como:

$$\mu(\{X_I\}) = -\sum P_n ln P_n \tag{4.1}$$

donde

$$P_n = \frac{\|X_n\|^2}{\sum \|X_n\|^2} \tag{4.2}$$

Según la teoría de la información, la entropía se define como el cálculo de la probabilidad de una señal, por lo que, para el caso de la compresión, permite determinar la representación que ofrece el menor número de bits[Sal01]. Por otra parte, la mejor base se consigue cuando la transformada de la señal posee la mínima función de coste.

El procedimiento para calcular la mejor base del Paquete Wavelet es el siguiente[Wic94, RB98]:

1. Escoger L como número máximo de niveles de descomposición.
2. Mientras el nivel de descomposición sea menor que L, se realizan los siguientes pasos :
   (a) Se calcula la función de coste para los coeficientes de la transformada en el nodo[1] i, n.
   (b) Se descompone el nodo i, n y se aplica la función de coste a los nodos $i + 1, 2n + 1$ y $i + 1, 2n$.

[1] iCorresponde al nivel de descomposición.
nEs un nodo del nivel i

(c) Si $\mu(i,n) > \mu(i+1,2n) + \mu(i+1,2n+1)$ permanecen los nodos $i+1,2n$ y $i+1,2n+1$, de lo contrario sólo se mantiene el nodo i,n o también conocido como nodo padre.

Cabe mencionar que este algoritmo es aplicado a cada trama de entrada de 1024 datos, lo que implica que cada bloque tiene su propia estructura de descomposición o representación mínima.

4.2.3 Codificación

Para codificar la información se emplean dos estrategias, una codificación diferencial para el nodo de mayor resolución y la codificación lineal predictiva (LPC) para los restantes nodos del árbol. La codificación lineal predictiva consiste en almacenar los parámetros de un filtro todo polos representado por medio de la función de transferencia:

$$H(z) = \frac{G}{1 - \sum_{k=1}^{p} \alpha_k Z^{-k}} \tag{4.3}$$

En nuestro caso, los LPC se emplearon para predecir la forma de onda de los coeficientes wavelet de los nodos terminales de menor relevancia. Para la realización del algoritmo de predicción lineal se resuelven las ecuaciones normales de Yule-Walker[DHP00]:

$$\begin{bmatrix} \alpha_1 \\ ... \\ \alpha_p \end{bmatrix} = \begin{bmatrix} r_0 & ... & r_{p-1} \\ ... & & ... \\ r_{p-1} & ... & r_0 \end{bmatrix}^{-1} \begin{bmatrix} r_1 \\ ... \\ r_p \end{bmatrix} \tag{4.4}$$

donde $r_k = \sum_n s(n)s(n-k)$, son los elementos de la matriz de autocorrelación de la señal de entrada, $n = k$, ..., $N-1$y $k = 0$,..., p. Esta matriz puede calcularse ya sea por medio de la transformada de Fourier o por la convolución en el tiempo. En este trabajo se utilizó la segunda forma debido a que implicaba un menor costo computacional.

Por otra parte, dado a que del orden del filtro depende el tamaño de la matriz, para su elección se empleó el siguiente criterio $P = 2 * BW + [2,3,4]$, donde BW es el ancho de banda de la señal. Para nuestro caso P puede ser igual 10, 11 o 12. Finalmente se optó por 10, debido a que implicaba menos cálculos y según simulaciones en Matlab no había una diferencia significativa con los predictores de orden superior. Al realizar la autocorrelación obtenemos una matriz Toeplitz, que permite reducir el problema de cálculo de los pesos del filtro IIR a la determinación de la inversa de esta matriz. Un método computacionalmente eficiente para resolver la ec. (**??**), es el algoritmo de *Levinson-Durbin [DHP00].* El código del algoritmo se presenta en el anexo *E.*

4.2.4 Cuantización

Para realizar el empaquetamiento de cada bloque procesado se utilizó el esquema de la Figura 4.3, el cual cuenta con los siguientes parámetros:

1. Cabecera (número de nodos, índices de los nodos, número de coeficientes wavelet).
2. Coeficientes wavelet
3. Coeficientes LPC y ganancia de los filtros LPC.

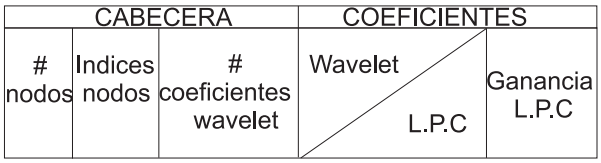


Figura 4.3: Esquema de cuantización para cada uno de los bloques de datos

El parámetro 1, solamente se hace necesario para la estructura de árbol dinámico, pues se compone del número de nodos terminales y los índices de estos nodos. La ranura de los Coeficientes contiene tanto los parámetros LPC como los coeficientes del WP no codificados. Estos se diferencian entre sí dado a que el número asociado al nodo terminal n de cierto nivel i es diferente de cero en los parámetros predecidos linealmente, en otras palabras, los coeficientes del WP no codificados corresponden al nodo terminal que está más a la izquierda. A estos coeficientes no se les aplica el filtro LPC ya que además de tener una baja resolución contienen gran parte de la energía de la señal y no es deseable perder información que puede ser importante en la reconstrucción. Sin embargo, se determinó que, la máxima diferencia entre dos coeficientes wavelets vecinos cuantizados por medio de la ley μ[PS94], es de 15, lo cual permite emplear una cuantización diferencial que consiste en almacenar únicamente las diferencias entre los valores cuantizados de los coeficientes wavelet, empleando 4 en lugar de 8 bits, lo cual aumenta considerablemente la razón

4.3 Descompresor

Para la descompresión se interpreta la información empaquetada con el formato de la Figura 4.3 y se usa un paquete wavelet inverso para reconstruir la señal (Figura 4.4). Como la gran mayoría de los nodos terminales del árbol de descomposición se representaron por medio de coeficientes de predicción lineal, es necesario aplicar a dichos nodos de entrada del árbol de síntesis las muestras generadas por medio de un conjunto de filtros IIR todo polos excitados con ruido blanco, en los que, sus coeficientes son los parámetros LPC.

Cabe resaltar que en la implementación final del sistema, en el empaquetado de datos no se almacenan realmente los coeficientes LPC, sino los coeficientes de autocorrelación parcial o *parcor*[DHP00], pues éstos se encuentran normalizados, lo cual facilita su cuantización. Estos coeficientes son una etapa intermedia de cálculo de los LPC, por

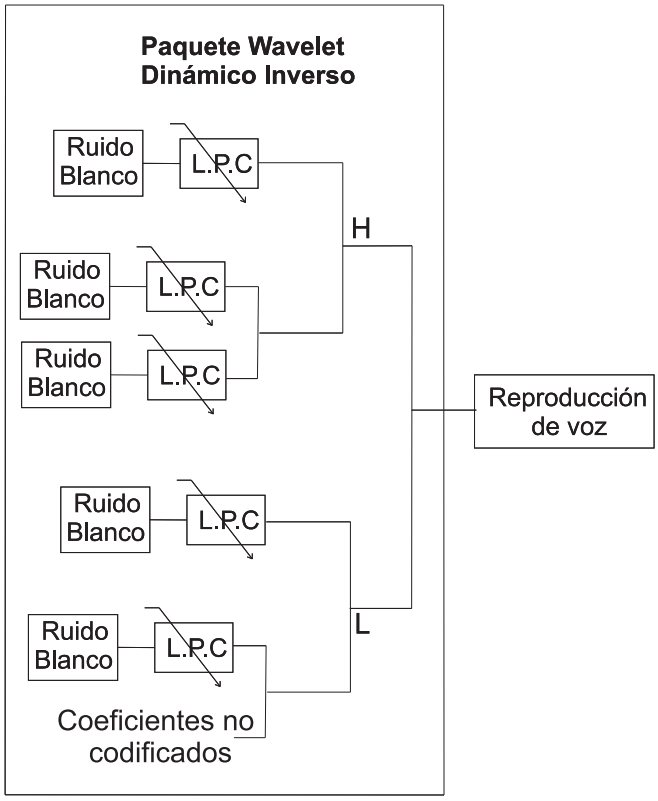


Figura 4.4: Estructura del descompresión propuesto

consiguiente, en el receptor se deben realizar además de los procesos de filtrado y cálculo del paquete wavelet inverso, concluir el cálculo de los coeficientes LPC que requieren los filtros digitales todo polos.

4.4 Implementación en el DSP

Para la implementación en tiempo real sobre el DSP, se empleó un procesador TMS-320C6701 disponible en la tarjeta de desarrollo EVM6x, la cual incorpora un CODEC de audio empleado para la adquisición de la señal. En la captura de datos se usa una frecuencia de muestreo de 8KHz, una resolución de 16bits, y la técnica de entrada/salida por Acceso Directo a Memoria (DMA) empleando buffer doble de 1024 muestras. En la implementación con árbol estático, se empleó la estructura de implementación optimizada de los paquetes wavelet por células básicas que se presentó en la sección *3.3* y el apéndice *A*, la cual considera una continuidad entre los de datos capturados. Por su parte, en el esquema dinámico, por cada uno de los bloques capturados se determina el mejor árbol, por medio del cálculo de la entropía y el algoritmo de la selección de la mejor base. Por cuestiones de optimización, este algoritmo se encuentra embebido en la rutina de cómputo de la célula básica (ver apéndice E). Además, se incluyó dentro del sistema de compresión un esquema de reducción del ruido que emplea la técnica *Wavelet Denoising* con umbral fuerte [RB98].

Dado a que en el cálculo de la entropía se requiere evaluar de la función logaritmo, que involucra muchos ciclos de máquina si se usa la función log de la librería math.h, se hizo necesario crear una función optimizada que hiciera uso de instrucciones MAC. Para tal efecto, se empleó la aproximación de Padé, usando una ecuación de tercer orden:

$$log(x) = \frac{b_0 + b_1 x + b_2 x^2 + b_3 x^3}{a_0 + a_1 x + a_2 x^2 + a_3 x^3} \tag{4.5}$$

.

Los coeficientes de la aproximación de Padé que mejor se adaptan al rango de argumentos de entrada se listan en la Tabla 4.1.

Tabla 4.1: Coeficientes para la aproximación de Padé de la función log

k	a_k	b_k
0	0.00000189502225	-0.00003485477543
1	0.01356191615502	-0.07607351608817
2	0.61316751842795	-1.31783184604965
3	1.0	1.37152375842232

Para comprimir y descomprimir las señales de voz en el DSP y almacenar los resultados en un archivo de datos en el computador, se desarrolló una interfaz gráfica de

usuario en Builder C++ 5.0, que permite transformar la tarjeta experimental EVM6x en un compresor/descompresor por hardware (Figura 4.5). Para esto fue necesario empelar la librerías que permiten manipular el *HPI* (*Host Port Interface*) tanto en el DSP como en el PC.

La interface ubicada en el Host es muy simple, pues se encarga de enviar los códigos de máquina que conforman el programa compresor o descompresor a la tarjeta EVM, poner el programa en ejecución y almacenar los datos comprimidos en un archivo de datos en el PC o bien, leer el archivo comprimido y enviar los fragmentos de datos a la tarjeta EVM para su respectiva descompresión. Para sincronizar ambos sistemas se codificaron las rutinas representadas en diagramas de estados que se muestran en la Figura 4.5. En esta figura se presentan únicamente los diagramas correspondientes al compresor. Para el descompresor si cuenta con una estructura semejante.

El sistema funcionó perfectamente en tiempo real, pues el tiempo promedio que tardó la rutina del compresor encargada de realizar el cálculo del paquete wavelet de descomposición, incluido la búsqueda de la mejor base y cálculo de los LPC, llegó a tener, en el peor de los casos (un árbol dinámico de cuatro niveles de profundidad) fue de 1.548.000 ciclos de máquina que corresponden a 11.6ms, y el tiempo máximo permisible es de 128ms.

4.5 Desempeño de los sistemas

Del conjunto de factores reportados en la literatura para para medir el rendimiento de la compresión, se analizaron en este trabajo los factores de razón de compresión, el error relativo y el test MOS[Sal01, RY01]:

1. La razón de compresión está definida como [Sal01]: $RC = \frac{Tamano\,de\,entrada}{Tamano\,de\,salida}$. Un valor de 0.6 indica que después de la compresión los datos ocupan el 60% del tamaño original. Valores mayores que 1 indican que los datos de salida son más mayores que los datos de entrada[2] . El inverso de la razón de compresión es el factor de compresión y en este caso valores mayores que 1 indican compresión , y menores que 1 expansión.

2. Error relativo[Teo98]: Calcula el error relativo en la reconstrucción de la los datos comprimidos y se denota según la siguiente ecuación : $ERR = \frac{\|f_n - f_{rec}\|}{\|f_n\|}$, donde f_n es la señal original y f_{rec} la reconstruida, el denominador implica la norma de f_n.

3. Test MOS (*Mean Opinion Score*) [Jim02]. El test MOS consiste en una evaluación de la calidad de la voz de un sistema. Fue normalizado por el Comité Consultivo Internacional de Telefonía y Telegrafía (*CCITT*) a principio de los años 80 y se ha utilizado principalmente para medir la calidad en sistemas de comunicación

[2]Compresión negativa

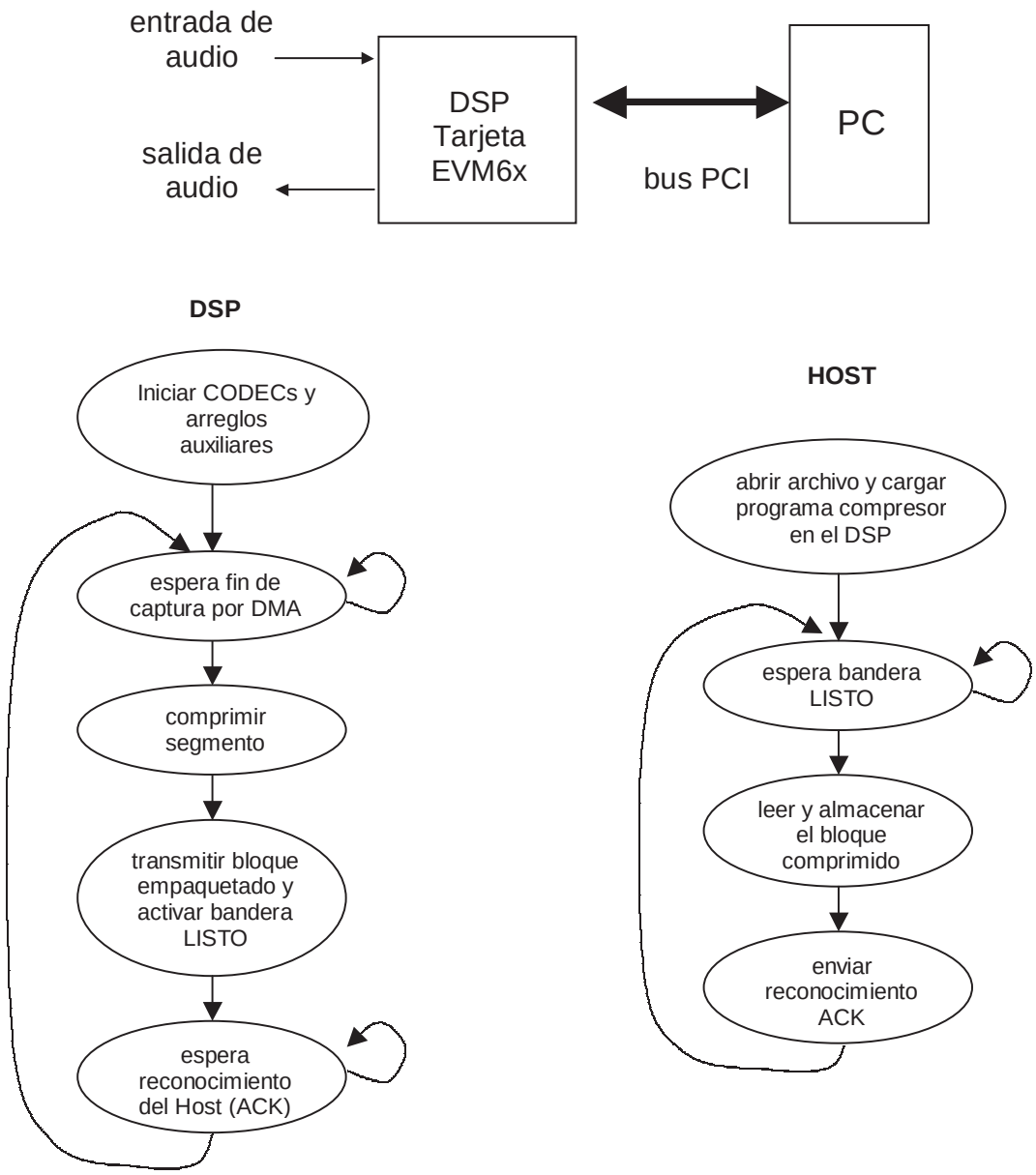


Figura 4.5: Interface entre el DSP y el PC para el sistema de compresión/descompresión

digital celular. El test consiste en realizar una encuesta de opinión a un conjunto de individuos de prueba los cuales deben evaluar la señal descomprimida según la Tabla 4.2

Tabla 4.2: Test MOS

NOTA	CALIDAD	ESFUERZO DE ESCUCHA	DEGRADACIÓN
5	Excelente	No requiere ningún esfuerzo	Inaudible
4	Buena	No requiere esfuerzo apreciable	Audible Pero no Molesta
3	Aceptable	Se necesita esfuerzo moderado	Ligeramente Molesta
2	Mediocre	Se necesita esfuerzo considerable	Molesta
1	Mala	Cualquier esfuerzo no permite comprender	Muy Molesta

En la Tabla 4.3 se muestra el desempeño del compresor dinámico y estático implementado en el DSP. Estas pruebas fueron efectuadas a un conjunto de 10 hablantes quienes pronunciaron el mismo texto.

Tabla 4.3: Desempeño del sistema de compresión

Hablante	ERR		RC		Test MOS
	dinámico	estático	dinámico	estático	
1	13.11	14.2	0.21	0.136	4
2	12.63	13.2	0.20	0.136	4
3	13.70	15.9	0.18	0.136	4
4	17.12	12.9	0.09	0.136	2
5	20.10	23.4	0.10	0.136	2
6	12.83	15.7	0.15	0.136	4
7	15.13	10.5	0.09	0.136	3
8	13.40	12.9	0.21	0.136	3
9	12.53	15.3	0.19	0.136	4
10	10.38	12.1	0.24	0.136	5
11	16.66	15.7	0.19	0.136	4

Como puede observarse en la tabla los resultados obtenidos son aceptables, con una nota promedio de 3.9, similar al reportado por Hosny et al. [HERES96] para un compresor basado en DWT. Respecto a la tasa de transmisión tenemos que, nuestro sistema de compresión de voz con árbol estático está muestreado a $8KHz$, pero con una resolución de $16bits$, lo que implica una capacidad máxima de almacenamiento de $8KHZ \times 16bits = 128Kbps$, y en promedio los datos comprimidos se almacenan con una velocidad de $8.7Kbps$, valor que está ligeramente por debajo de los valores reportados por Hosny [HERES96] ($9.93Kbps$) y Singh [SAA98] ($11.23Kbps$) para compresores tipo wavelet, y mejora en cerca de dos veces la norma G.728 [SAA98], que establece una tasa de transferencia fija de $16Kbps$.

Finalmente, hay que resaltar el hecho de que el árbol dinámico arroja en promedio para la razón de compresión un valor de 0.185, el cual es superior al árbol estático, sin embargo, es importante destacar que para algunas voces particulares, el árbol dinámico ofrece una mejor razón de compresión.

4.6 Conclusiones

Se propuso un sistema de compresión de señales de voz híbrido que hace uso de la transformada wavelet por paquetes y la codificación LPC para los nodos terminales de menor relevancia. El sistema ofrece un desempeño superior a los sistemas de compresión basados en wavelets reportados en la literatura. Se mostró que con el empleo de una estructura de árbol dinámico se consigue, en promedio, una razón de compresión más alta que con el árbol estático, pero para ciertas voces particulares, la razón de compresión es mejor que la del árbol estático.

Capítulo 5

Sistema de multiplexación de señales de voz

5.1 Panorama de los Sistemas de Multiplexación de Señales basados en Wavelets

Desde su aparición, la transformada wavelet ha sido usada en una amplia variedad de aplicaciones [RB98, SN97, Teo98] entre las cuales se encuentran los sistemas de modulación y multiplexación[AM99]. Al respecto, se han propuesto diferentes esquemas de modulación y multiplexación de señales [AM99, HSD95, ATJ04, WWDJ97, Won98, Wu98, Lin97]. El más exitoso de ellos y de aplicación en las comunicaciones móviles, es el empleo de un árbol de síntesis wavelet para la generación de señales de espectro esparcido y/o la generación de códigos CDMA [AM99, Het, HSD95, ZFL02]; otro esquema de poca divulgación es la Multiplexación de Señales por División de Paquetes Wavelet (WPDM) propuesta por Wong et al. [WWDJ97, WWD$^+$00, Wu98]. Esta última al igual que las técnicas convencionales TDM y FDM, tiene como propósito el transmitir en forma simultanea múltiples señales por un mismo canal, recurriendo a la codificación de cada una de las señales-mensaje con las funciones base de un paquete wavelet [WWDJ97]. A diferencia de las técnicas convencionales, WPDM ofrece un traslape frecuencia-tiempo entre los canales multiplexados, similar como lo hace la Multiplexación por División de Frecuencia Ortogonal (OFDM)[AM99].

La WPDM está basada en los conceptos de la transformada wavelet discreta DWT, y consiste en utilizar la transformada de paquetes wavelet inversa como esquema de multiplexación, en el que los coeficientes a la entrada en las ramas del árbol, son reemplazados por las señales-mensaje, con el fin de codificarlas con las funciones bases (wavelet), y conformar una única señal que contiene la información suficiente y necesaria para recuperar en el receptor las señales transmitidas. La WPDM, por el hecho de estar basada en la DWT, tiene la propiedad de contener información en el dominio tiempo-frecuencia, generando un solapamiento de las señales, que a su vez, produce una mejora del ancho de banda, en comparación con la multiplexación por división de frecuencia

FDM[WWDJ97, Wu98]; la propiedad de ortogonalidad garantiza que las señales traslapadas puedan ser recuperadas en el receptor, además esta codificación le brinda a la WPDM una mayor inmunidad ante el ruido blanco aditivo gausiano (AWGN) que la multiplexación por división de tiempo TDM[WWDJ97, WWD$^+$00, Wu98].

Adicionalmente, se han reportado en la literatura estudios de este sistema ante ruidos impulsivos[WWD$^+$00, Wu98], errores de temporización[WWDJ97, Wu98] y canales descendentes[Wu98], mostrando excelentes desempeños; incluso se han propuesto nuevos coeficientes para los filtros de descomposición wavelet madre que satisfacen criterios de optimización referidos a la reducción de errores de temporización[WWD$^+$00].

Sin embargo, la principal desventaja de esta técnica es la no linealidad de la amplitud de la señal multiplexada, característica que limita aplicación en sistemas de baja potencia[ATJ04, TWW00]. Para mejorar estas deficiencias, Aicha et al. [ATJ04], basados en el hecho de que las señales a transmitir son binarias, proponen emplear la función wavelet madre Haar como función base de descomposición en el paquete wavelet, sin embargo, esta propuesta debe ser cuestionada, pues dicha función wavelet no permite sacar provecho de todas las potencialidades de la WPDM como son, la alta inmunidad al ruido y la mejor administración del espectro. A pesar de las deficiencias de WPDM respecto al manejo de potencia, se han propuesto variantes a dicho sistema tales como la Multiplexación por División de Paquetes Wavelet de Salto de Rama (*BH-WPDM: Branch Hopping Wavelet Packet Division Multiplexing*)[DSW98, Won98].

Cabe resaltar que la totalidad de los reportes sobre WPDM, se refieren a simulaciones, que hacen uso de señales-mensaje del tipo binarias, en las que los coeficientes de entrada toman el valor -1 o 1 [WWDJ97, Wu98]. Por tal motivo, se propone como alternativa en este trabajo realizar un estudio para señales de entrada multinivel y mostrar la factibilidad de implementación en arquitecturas digitales. Así mismo, y dado a que la técnica de WPDM es altamente sensible a la sincronización, se propone también una técnica de sincronización que permita la ejecución en tiempo real del sistema de multiplexación.

5.2 Diagrama esquemático de la multiplexación por división de paquetes wavelet (WPDM)

Si las versiones desplazadas en el tiempo de la función base $\phi_{lm}(t)$, forma el espacio de señales que permite representar los coeficientes wavelet en un nodo particular (l, m)[1] del árbol de descomposición, la señal sintetizada en la base del árbol (paquete inverso) deberá calcularse por medio de[WWDJ97]:

$$s(t) = \sum_{l\in\mathcal{L}, m\in\mathcal{M}_l} \sum_{n} x_{lm}[n]\phi_{lm}(t - nT_l) \tag{5.1}$$

[1] l= nivel; m=posición dentro del nivel

con $\mathcal{L}$ el conjunto de niveles que contienen los nodos terminales de un árbol dado, $\mathcal{M}_l$ el conjunto de índices de los terminales en el nivel l; $x_{lm}[n]$ los bits de la señal-mensaje a transmitir, que deben tomar el valor ± 1, y $s(t)$ es la señal multiplexada.

Como las funciones wavelet madre de un espacio de menor resolución se pueden obtener a partir del filtrado y diezmado de la función madre del espacio de mayor resolución (ϕ_{00})[Wic94], la expresión anterior se puede reducir a[WWDJ97, Wu98]:

$$s(t) = \sum_k x_{00}[k]\phi_{00}(t - kT_l) \tag{5.2}$$

donde ϕ_{00} es la función escaladora del espacio de mayor resolución o nodo base del árbol y x_{00} es la secuencia equivalente calculada en la base del árbol por medio del filtrado e interpolado en tiempo discreto de los coeficientes que constituyen la señal-mensaje. En la Figura 5.1 se muestra el diagrama de bloques de implementación de la ec. (5.2). Nótese que, como consecuencia directa de dicha ecuación, el esquema multiplexor emplea un modulador PAM (*Pulse Amplitute Modulation*) a la salida del nodo raíz del paquete wavelet inverso. Este modulador, al igual que en la técnica de TDM, tiene por objeto transformar la señal multiplexada pulsante a una señal extensa en el tiempo, ofrecer mayor inmunidad al ruido y reducir la interferencia intersímbolo (*ISI: Inter-Symbol Interference*) [Str89, PS94].

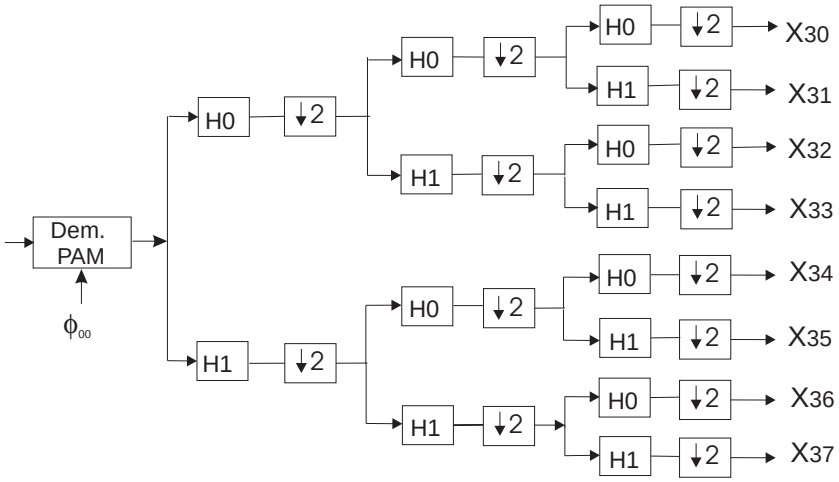


Figura 5.1: Multiplexación por división de paquetes wavelet

En el receptor, la señal se demultiplexa por medio del empleo de un correlacionar seguido de un muestreador (demodulador PAM) y el respectivo paquete de descomposición wavelet (Figura 5.2).

Por otra parte, como la WPDM está basada en la DWT, la señal multiplexada estará traslapada tanto en tiempo como en frecuencia y por esta razón puede hacerse un mejor uso de los recursos del canal. Por ejemplo, al emplear una estructura del árbol asimétrica (Figura 5.3) es posible transmitir señales a diferentes ratas de muestreo, ya

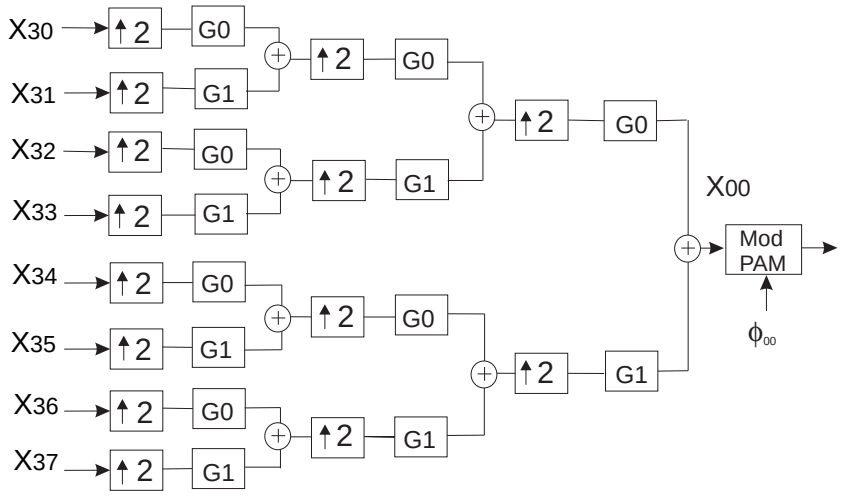


Figura 5.2: Demultiplexación por división de paquetes wavelet

que los canales x_{30},x_{31}, x_{32}, x_{33},x_{36},x_{37} tienen el mismo período de muestreo, mientras x_{22} tiene una tasa de muestreo equivalente al doble de los otros canales. De esta forma se puede asignar mayor o menor ancho de banda a los usuarios según los requerimientos.

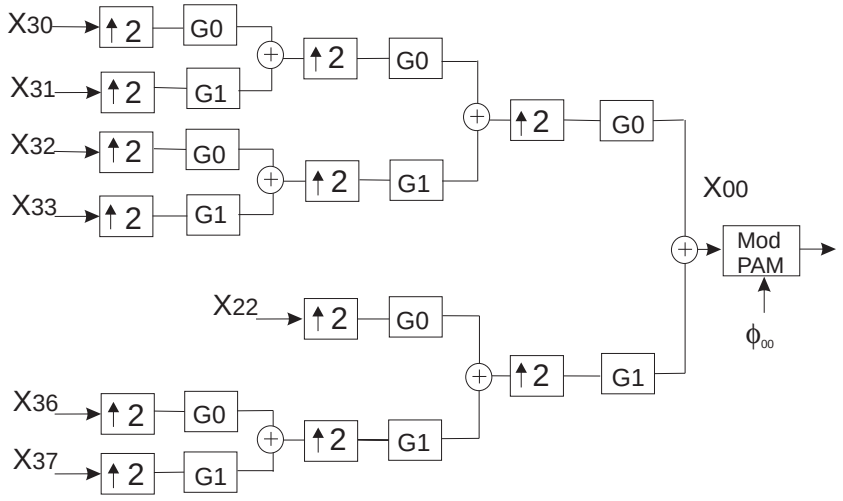


Figura 5.3: Árbol asimétrico

5.3 Análisis de la sincronización del sistema WPDM

La demultiplexación de las señales WPDM requiere de una buena sincronización entre el transmisor y el receptor[WWDJ97]. De no garantizarse esta condición, aparecen en el

receptor problemas de interferencia intersímbolo ISI. Esto se puede verificar al calcular las condiciones de reconstrucción perfecta de un sistema WPDM de 2, 4 y M-canales (apéndice F), encontrándose que la señal recuperada en cualquiera de los canales es una combinación lineal de las señales de los canales restantes, inclusive bajo el supuesto de que se cumplan las condiciones de reconstrucción perfecta de los filtros QMF presentadas en la sección 3.2.

Para comprender mejor estas características del sistema WPDM e ilustrar el principio básico de la técnica de sincronización que se propone en este trabajo, en la Figura 5.4 se muestra la estructura de un sistema WPDM simplificado de dos y cuatro canales. En estos esquemas se han eliminado el modulador y demodulador PAM, pues son estructuras que para efectos de análisis se pueden combinar y modelar en conjunto por medio de una línea de retardos.

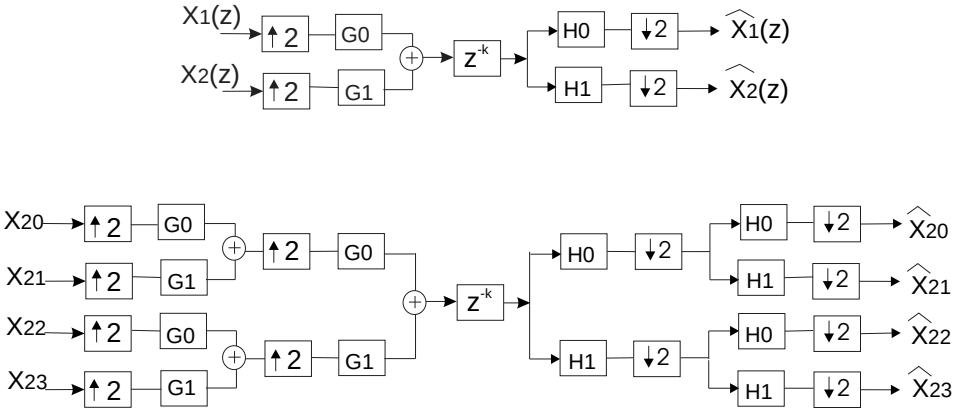


Figura 5.4: Sistema WPDM simplificado de dos y cuatro canales

Si en estas estructuras, a las que denominaremos en adelante como filtros *QMF inverso*s, se eliminan los elementos de la línea de retardos que se muestran en la figura, se tiene para el sistema de dos canales las siguientes condiciones de reconstrucción perfecta (ver anexo F):

$$\begin{aligned} H_0(z)G_0(z) + H_0(-z)G_0(-z) &= 2z^{-2k} \\ H_1(z)G_1(z) + H_1(-z)G_1(-z) &= 2z^{-2k} \end{aligned} \tag{5.3}$$

Dado a que los filtros H_0, H_1, G_0 y G_1 se obtienen a partir de las ecuaciones de los filtros QMF conjugados dados por la ec. (3.13), se encuentra al verificar las condiciones de reconstrucción antes indicadas, la siguiente relación:

$$[H^2(z) - H^2(-z)]z^{-(N-1)} = z^{-2k}$$

que solamente es válida si el término entre corchetes es igual a la unidad, lo cual no es cierto, pues si se emplean las respuestas al impulso de los filtros QMF conjugados,

la complementariedad espectral se consigue solo si $H^2(z) + H^2(-z) = 1$ (ec. 3.16). Se puede demostrar que (apéndice F), para el filtro QMF inverso de dos canales, la reconstrucción perfecta se consigue solamente si se introduce una línea de retardos de un número impar de muestras entre los árboles de síntesis y descomposición wavelet.

Para el caso del filtro QMF inverso de 4 o más canales, el número de muestras de la línea de retardo que se hacen necesarias para compensar la fase y garantizar la reconstrucción perfecta de los canales, está dada por la relación:

$$k = 2^l i + 2^l \left\lceil \frac{(Nh-1)\left(2^l-1\right)}{2^l} \right\rceil - (Nh-1)\left(2^l-1\right) \tag{5.4}$$

con $i = 0, 1, 2, 3, ..., l$ el nivel máximo de descomposición, Nh el tamaño de la respuesta al impulso de los filtros QMF, y el operador $\lceil x \rceil$ denota el redondeo a la cantidad entera más próxima por encima. En esta ecuación, el término de la izquierda expresa un comportamiento cíclico para la compensación de fase y los términos restantes denotan el valor del corrimiento mínimo necesario para evitar la interferencia intersímbolo. De esta forma, para eliminar la ISI en un sistema WPDM es necesario introducir una línea de retardos, de longitud adaptable, a la entrada del árbol de descomposición wavelet (Figura 5.5), y realizar una búsqueda de la fase de compensación en el rango entre $0 \leq k \leq 2^l - 1$. Esta propiedad se aprovecha para el diseño del sistema de sincronización propuesto en este trabajo.

Cabe resaltar que dicha compensación de fase es condición indispensable para la implementación en tiempo real de la técnica WPDM, pues los algoritmos que se proponen para el cálculo de los paquetes wavelet directo e inverso están orientados a operaciones causales sobre un flujo continuo de datos. En la mayoría de las librerías comerciales y gratuitas que se encuentran para el tratamiento de los paquetes wavelet, incluido Matlab, el cálculo de las transformadas se realiza en forma no causal que no requieren compensación de fase. Por tal motivo, las simulaciones y pruebas que se presentan en este documento fueron realizadas por medio de los algoritmos de cómputo de la IPW e PWD que se presentaron en la sección 3.3.

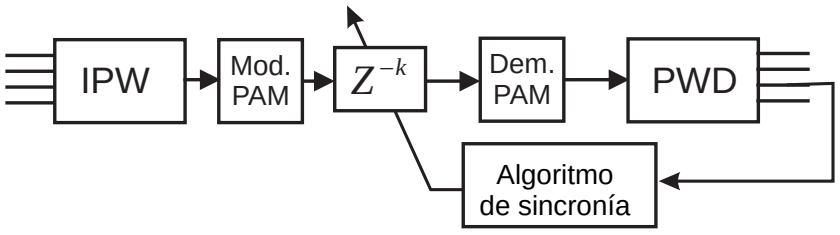


Figura 5.5: Esquema del sistema de sincronización propuesto

5.4 Sistema propuesto

Para la implementación del multiplexor y demultiplexor WPDM se empleó la estructura de implementación en células básicas de los paquetes wavelet que se describió en la sección 3.3.1, pues esta ofrece una menor cantidad de instrucciones MAC, mayor velocidad y la ventaja de implementar árboles no simétricos para la transmisión de señales-mensaje a diferentes anchos de banda.

Respecto al modulador PAM, su algoritmo de cómputo se obtiene a partir de la discretización de la ecuación de un modulador PAM (ec. 5.2):

$$s\left[n\right] = \sum_{k} x\left[\frac{n-u}{I_{PAM}}\right] h\left[u\right] \tag{5.5}$$

donde $x[n]$ será la señal pulsante multiplexada y $h[n]$ la versión discretizada de la función base ϕ. Esta ecuación sugiere que el modulador PAM se implementa por medio de la interpolación por el factor I_{PAM} seguido de un filtrado con respuesta al impulso de la forma $h\left[n\right] = \phi\left(n\frac{T_s}{I_{PAM}}\right)$. Este interpolador, y el respectivo diezmador correspondiente al demodulador PAM, hacen uso del algoritmo descrito en la sección 3.3.2 y el apéndice C.

Aunque la ec. (5.2) indica que la respuesta al impulso empleada en el cálculo del modulador PAM debe ser la función base de descomposición, en este trabajo se exploró la posibilidad de implementación de un sistema híbrido basado en el cálculo de los paquetes wavelet y el empleo en el modulador PAM de la función coseno elevado[Str89]:

$$\phi(t) = \left(\frac{sen\omega t}{\omega t}\right)\left(\frac{cos\alpha\omega t}{1-(\frac{2\alpha\omega t}{\pi})^2}\right) \tag{5.6}$$

que es la forma de pulso más ampliamente usada en los sistemas TDM convencionales. En este caso, el factor de interpolación I_{PAM} se escogió igual a 9, pues con éste valor se garantizan cruces por cero en diferentes instantes de muestreo, lo cual garantiza la reducción de la interferencia intersímbolo (ISI). El parámetro α, toma valores en el rango $0 \leq \alpha \leq 1$, y permite controlar las posiciones de los cruces por cero y reducir o aumentar el ancho de banda (BW) de la señal multiplexada. Cuando α se hace cercano a cero, el BW requerido para la transmisión se reduce, de lo contrario si α se aproxima a uno, éste se hace mayor. La ecuación (5.7) permite predecir dicho comportamiento:

$$BW = \frac{(1+\alpha)}{2T_x} \tag{5.7}$$

De forma experimental se seleccionó un factor de transición de $\alpha = 0.4$, ya que éste proporciona una mejor reconstrucción de los canales multiplexados.

5.4.1 Sincronización

Dado a que el multiplexor WPDM emplea un modulador PAM a la salida del nodo base del árbol, Wu [Wu98] propone emplear como técnica de sincronización en el receptor el mismo principio de la recuperacón del reloj en un sistema TDM[PS94], que consiste en filtrar el valor absoluto de la señal multiplexada por medio de un filtro pasa-banda de banda angosta con frecuencia central igual a $1/T_s$. Sin embargo, tal como se demostró anteriormente, aunque se pueda recuperar la señal de reloj, es necesario introducir una fase de compensación para la recuperación correcta de los canales. Es así como, se propone un nuevo método, que consiste en emplear uno de los canales como *canal de sincronización*. El canal elegido para ello es el que presenta menor inmunidad al ruido y mayor susceptibilidad a la interferencia intersímbolo, y puede ser determinado teóricamente con las expresiones dadas en la Ref.[WWDJ97].

Por dicho canal se envía un tren de impulsos *sinc*, señal elegida por poseer un espectro en frecuencia de banda limitada y de amplitud uniforme en un rango amplio de frecuencias. En el receptor, se calcula la correlación cruzada entre la señal recuperada en el canal de sincronización y el tono *sinc* transmitido; de esta forma, si el sistema se encuentra perfectamente sincronizado, es de esperarse que la correlación cruzada sea igual a la función de autocorrelación de la señal sinc, en caso contrario se obtendría una señal que es la combinación lineal de todos los canales transmitidos.

Cuando se inicia el algoritmo del receptor WPDM, se realiza la búsqueda del desplazamiento más óptimo para la línea de retardos, empleando como función de coste, la distancia entre la correlación cruzada, estimada en el receptor, y la autocorrelación ideal del tono sinc. De esta forma, se calculan $2^l \times I_{PAM}$ distancias por cada uno de los posibles $k = 0, 1, 2, ..., 2^l \times I_{PAM} - 1$ elementos en la línea de retardos (Figura 5.6), teniéndose que la fase de compensación es el valor del retardo para el cual se encuentra la mínima distancia. En las expresiones anteriores, l es el número máximo de niveles de descomposición e I_{PAM} el factor de interpolación usado en la implementación del modulador PAM.

Es preciso indicar que este procedimiento se repite continuamente por cada trama de datos procesada en el receptor. Si la distancia entre las correlaciones es ligeramente superior a la distancia obtenida en el caso óptimo, la cantidad de elementos de retardo involucrados, se reduce o aumenta, hasta conseguir un nuevo mínimo en la distancia.

En la Figura 5.6 se muestra el diagrama de bloques del sistema de sincronización propuesto, cuyo algoritmo en notación de pseudocódigo es la siguiente:

---

sincro(senalsincr, tonosinc, autocorr)
▷ Calcula la correlación cruzada

    ytest←correlación_cruzada(senalsincr, tonosinc)

▷ Busca los valores mínimos y máximos a partir de
▷ los cuales debe calcular la distancia

    cenmasa←centro_masa(ytest)

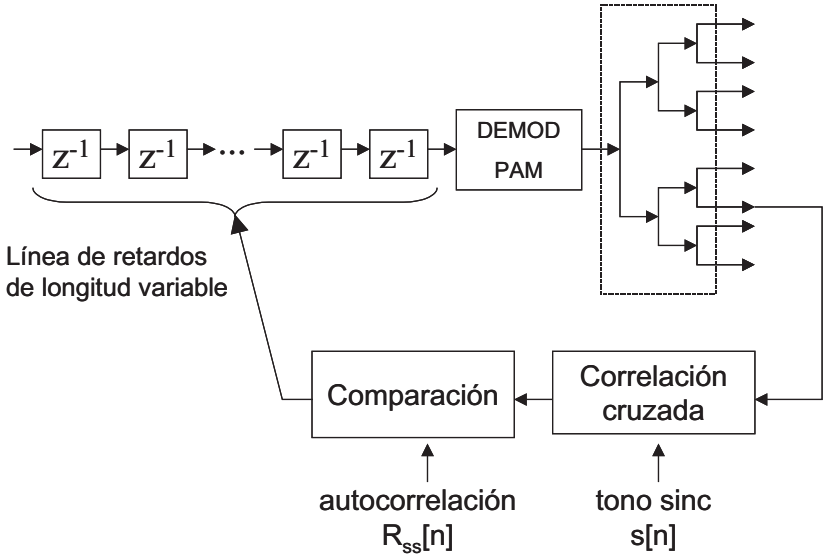


Figura 5.6: Sistema de sincronización

cmin←cenmasa-largo(tonosinc)/2;
cmax←cenmasa+largo(tonosinc)/2;
cy←posición_máximo(ytest[cmin:cmax])
cmin←cx-largo(tonosinc)/2;
cmax←cy+largo(tonosinc)/2;

▷ Calcula la distancia entre la correlación cruzada y la autocorrelación del tono sinc

dist←distancia(ytest[cmin:cmax], autocorr)
retornar dist

Para efectos de implementación, la correlación cruzada se calcula por medio del filtrado lineal entre la señal recuperada por el canal de sincronía y la versión reflejada del tono sinc transmitido idealmente. La búsqueda de los puntos extremos de la correlación cruzada tiene por objeto garantizar un cálculo más preciso de la distancia de esta señal y la función de autocorrelación.

El programa principal que se encarga de la rutina de cálculo del receptor WPDM debe invocar la función sincro de la siguiente forma:

▷ k es igual a la fase de compensación. Off tiene un valor inicial de 1

x[Off:largo(x)+Off]←**capturar_bloque_datos**()
y1←**demodulador_PAM**(x)
IPW(y1, coefs)

```
dist←sincro(
    coefs[canal_sincronización],
    tonosinc,
    autocorr
    )
Si sincronizado = 0
    distancias[k]←dist
    Si k = 2^l × I_PAM − 1
        k←posición_mínimo( distancias )
        dist_optima←distancias[k]
        Off ←k
        sincronizado←1
    k←k + 1
de lo contrario
    Off←0
    Si dist > dist_optima
        Off←paso
        k←0
        sincronizado←0
```

5.5 Pruebas de Desempeño

Para evaluar el comportamiento de los sistemas implementados se hace un estudio comparativo con las ténicas de multiplexación más comunes, TDM y FDM. Todas las pruebas que se presentan en esta sección fueron realizadas con las rutinas en lenguaje C que se desarrollaron para las implementaciones en tiempo real, dadas las características de causalidad y procesamiento a un flujo continuo de datos que ofrecen. Fue usado en todos las casos un árbol de descomposición uniforme de tres niveles, el cual permite la transmisión simultánea de ocho canales.

5.5.1 Sincronización

Para mostrar el efecto de la interferencia intersímbolo ante condiciones de de-sincronía e ilustrar el funcionamiento del sistema de sincronización desarrollado en este trabajo, se diseñaron una series de pruebas que discutirán a lo largo de esta sección.

La primera de ellas consistió en añadir líneas de retardo a la entrada al demodulador PAM, con el fin de simular el efecto del desplazamiento entero de muestras en un escenario de perfecta sincronía entre las señales de reloj del DAC del transmisor y el ADC del receptor. Con esta prueba se observa que a diferentes valores del desplazamiento de fase de la señal, el sistema produce una interferencia intersímbolo considerable, pero para ciertas fases específicas, se demostró de manera experimental, el comportamiento

cíclico que predice teóricamente la ec. (5.4). La Figura 5.7, muestra la ISI que se presenta con una fase arbitraria que no garantiza las condiciones de reconstrucción perfecta del banco de filtros QMF inverso, en cambio, para la fase de compensación calculada con la ec. (5.4), la Figura 5.8 muestra el comportamiento óptimo del sistema.

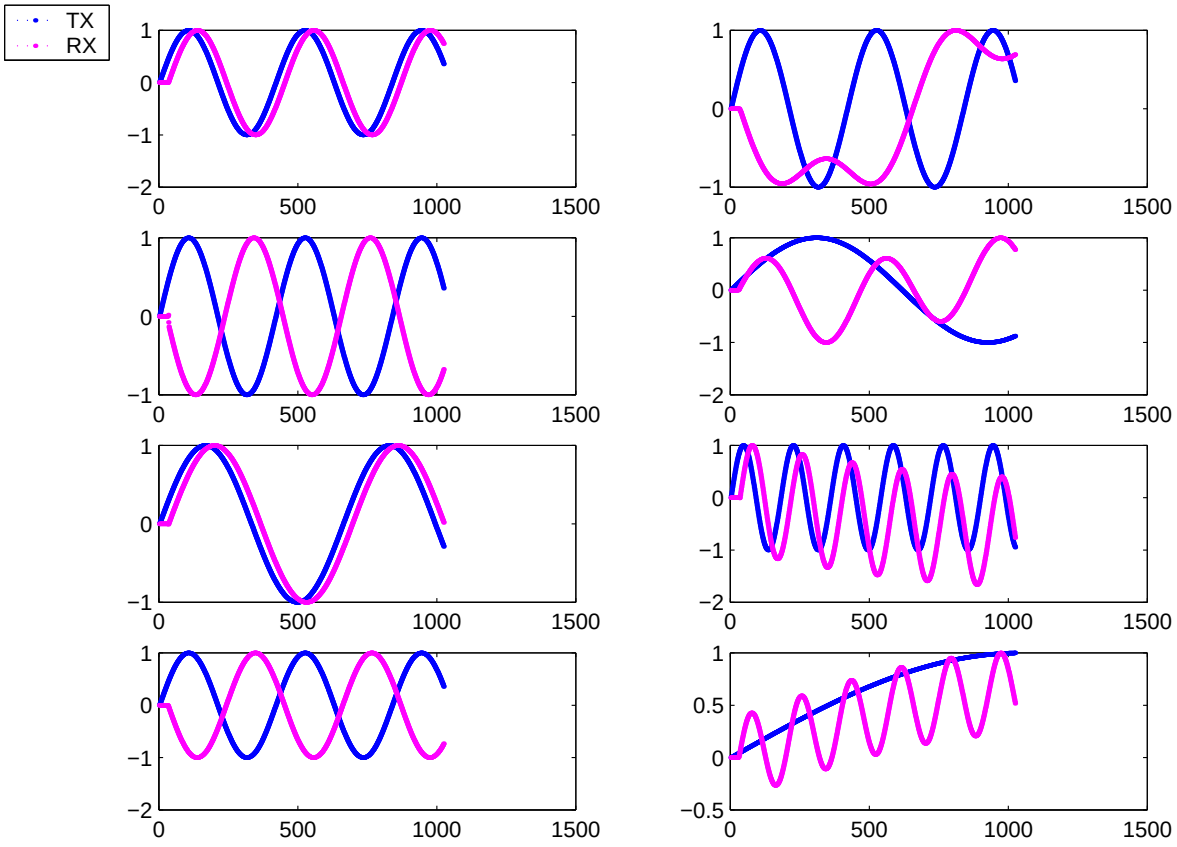


Figura 5.7: Recuperación de los canales en WPDM con fase arbitraria.

Para estudiar el efecto de la de-sincronización entre los relojes del DAC y ADC, del transmisor y receptor, respectivamente, se diseñó una prueba que permite introducir una fase no entera a la señal multiplexada, por medio la adición de un interpolador, desplazador de fase y diezmador, que hacen las veces del canal de comunicación. Al recuperar las señales se observó un comportamiento aceptable, pues todos los canales se ven afectados en mínima proporción (Figura 5.9).

Para ilustrar el adecuado funcionamiento del sistema de sincronización propuesto, se muestra en la Figura 5.10 la forma como evolucionan en el tiempo, las señales de salida en cada uno de los canales, una vez se inicia el receptor de WPDM. Nótese que solo unos instantes de tiempo después de haberse iniciado el sistema, se lleva a cabo la reconstrucción perfecta de todos los canales. Para facilitar la su visualización, lo resultados presentados en la figura fueron con señales de entrada senoidal, sin embargo, el

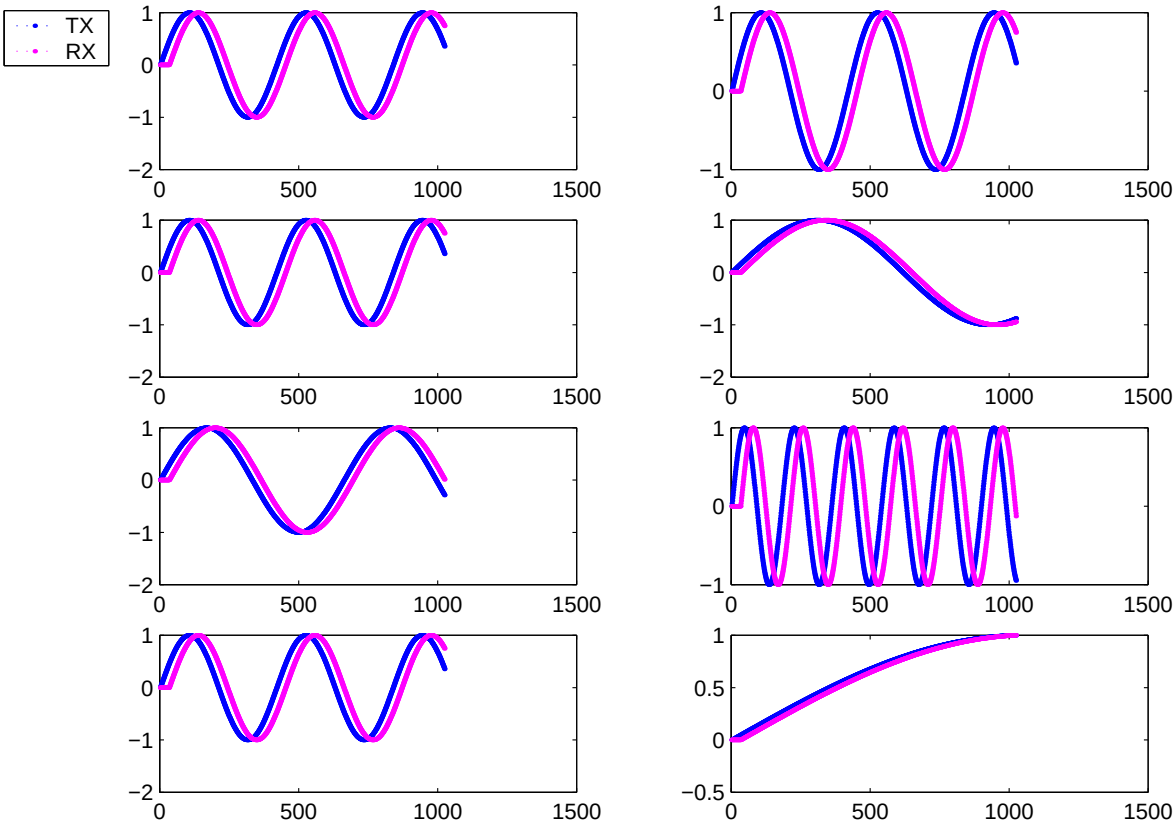


Figura 5.8: Recuperación de los canales en WPDM con fase de compensación

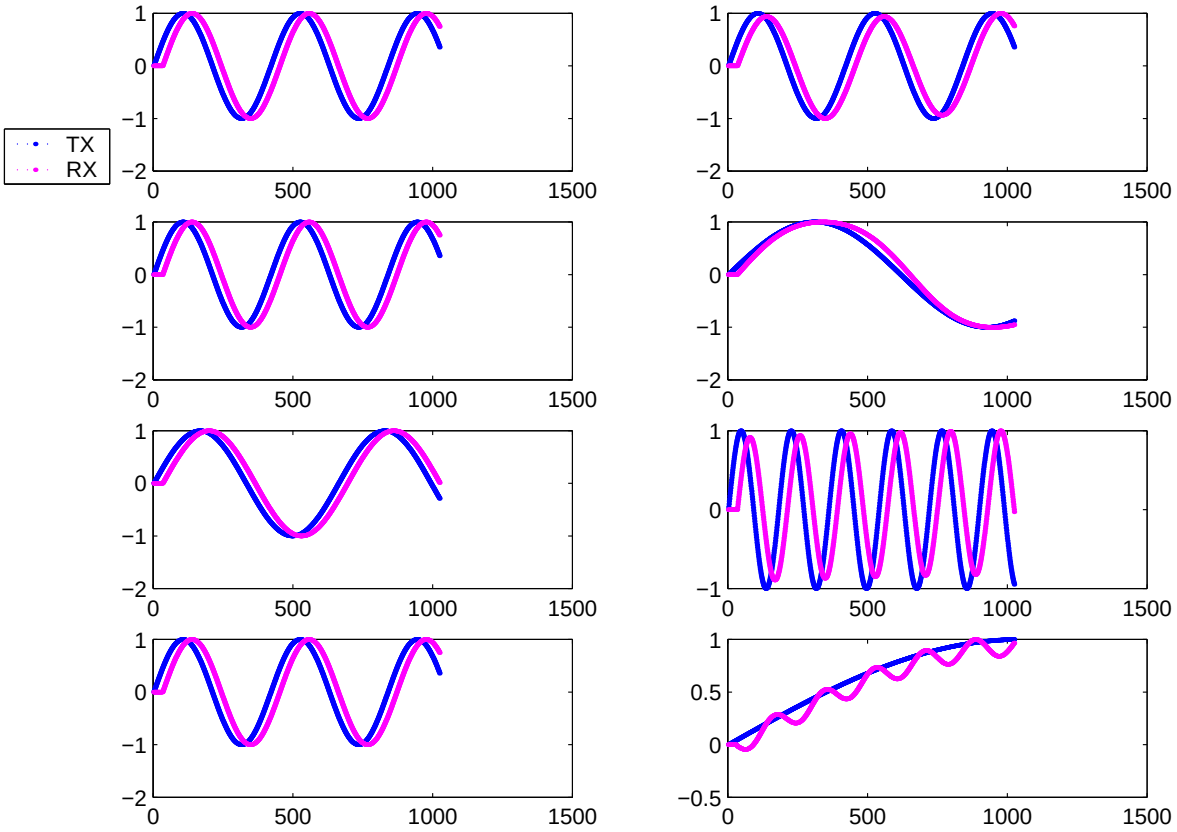


Figura 5.9: Recuperación de los canales WPDM con fase no entera

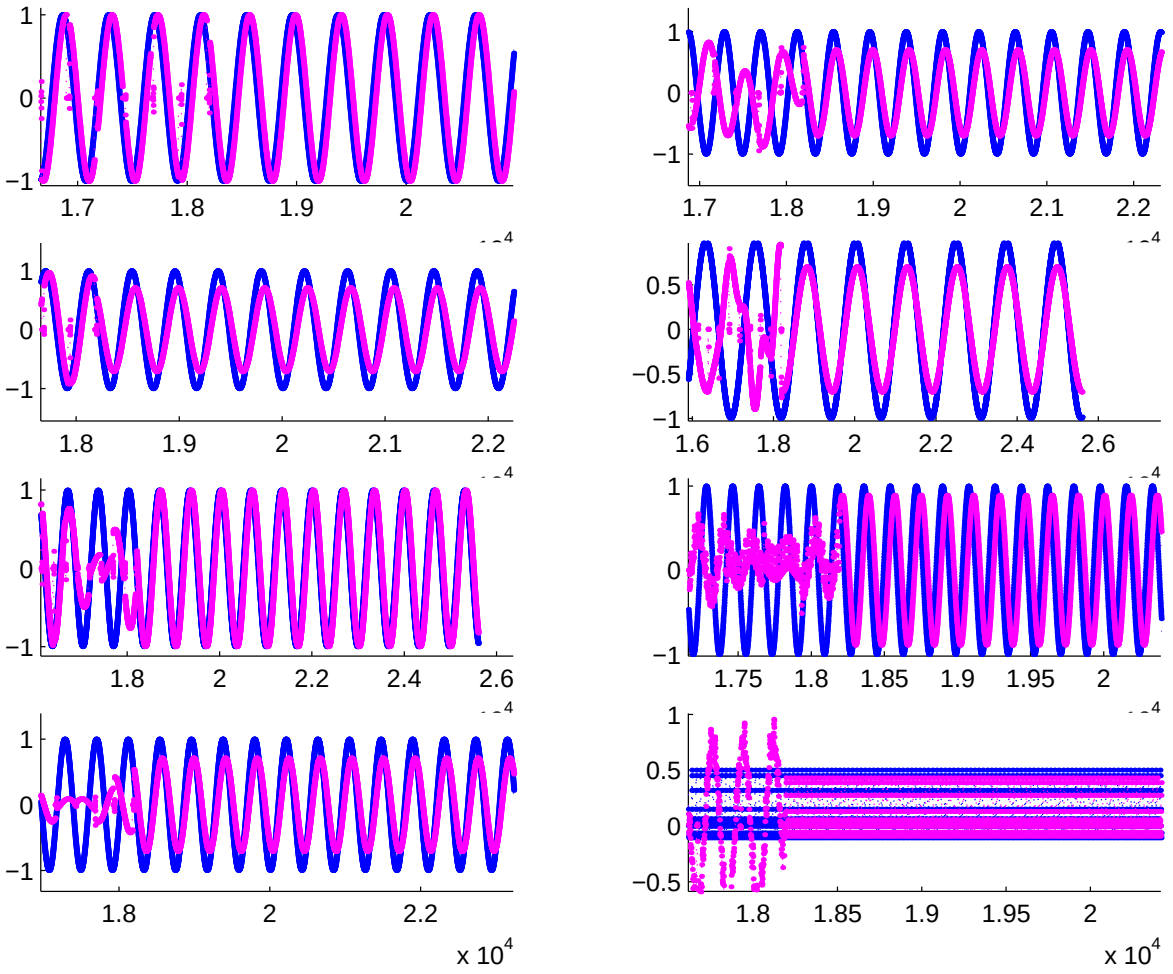


Figura 5.10: Funcionamiento del sistema de sincronización

sistema de sincronización fue sometido a pruebas empleando señales de voz y diferentes relaciones señal a ruido (SNR) para la señal multiplexada. Aunque la inmunidad al ruido está ligada a la función wavelet madre como se verá más adelante, se encontró en promedio una sincronización aceptable del sistema hasta SNR superiores a $2dB$.

5.5.2 Inmunidad al Ruido

Para hacer el estudio comparativo de la influencia del ruido aditivo gausiano (AWGN), se hizo un estudio comparativo entre los sistemas de multiplexación TDM y WPDM, dado que el sistema TDM ofrece una mayor inmunidad al ruido que FDM[Str89, PS94]. Las señales a transmitir por cada uno de los canales son multinivel, por lo tanto se estableció como criterio de comparación, la medida de la distancia entre la señal transmitida y la recuperada en cada uno de los canales. Se definió esta unidad de medida ya que en la totalidad de los reportes para WPDM se emplea como unidad de criterio la tasa de bit erróneo (*BER: Bit Error Rate*), pues lo aplican a la transmisión de señales binarias [WWDJ97, WWD$^+$00, Wu98]. Dicha distancia se calcula con la ecuación (5.8):

$$D = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ...} \tag{5.8}$$

Para garantizar su adecuada medición, se eliminó, de las señales recuperadas, el corrimiento de fase que introducen los filtros QMF. Para WPDM, se realizaron pruebas de inmunidad al ruido empleando para los paquetes wavelet diferentes funciones base como son Coiflet (coif), Symlet (sym) y Daubechies (db), a diferentes órdenes, dado a que estas satisfacen las condiciones de admisibilidad en la DWT. Para el modulador y demodulador PAM del esquema WPDM y TDM, se empleó la función coseno elevado.

En la Tabla 5.1 se muestra el desempeño del sistema WPDM ante una $SNR = 0dB$ y usando voz como señal-mensaje. A partir de esta tabla se seleccionaron las funciones base con mejor adaptación a una condición de ruido excesiva, como lo es la SNR igual a $0dB$. Las cinco mejores funciones base con un buen desempeño a estas condiciones de ruido fueron la coiflet 5, la daubechies 13-17 y la symlet 14-17. Las figuras 5.11, 5.12 y 5.13, muestran las gráficas de distancia para cada uno de los canales con las wavelet madre elegidas, así como los resultados obtenidos para TDM. Todas estas gráficas están referidas a $SNR = 0dB$.

Nótese, que para el caso de TDM, los canales inferiores presentan una inmunidad al ruido menor que los canales superiores, en cambio, para el caso de WPDM con las funciones base daubechies 17 y symlets 17, el sistema ofrece una inmunidad al ruido semejante en todos los canales. Wong et al.[WWDJ97, Wu98] han encontrado resultados similares, pues proponen trabajar con funciones daubechies de órdenes superiores a 14 para realizar la multiplexación por WPDM a señales binarias.

Para obtener resultados más confiables se realizaron 100 pruebas con diferentes relaciones de señal a ruido para la mejor wavelet madre de cada función base, es decir coiflet 5, daubechies 17 y symlet 17, y se incluye también el desempeño del sistema TDM. El valor medio de la distancia obtenida para cada una de las SNR se presenta en la Tabla

Tabla 5.1: Distancia promedio de las funciones base

	Distancia promedio $SNR = 0dB$		Distancia promedio $SNR = 0dB$
coif 1	9.80	db 14	9.77
coif 2	9.76	db17	9.35
coif 3	9.60	db18	9.99
coif 4	9.95	sym 2	9.67
coif 5	9.51	sym 3	9.82
db 2	9.72	sym 5	10.02
db3	9.87	sym 6	9.79
db5	9.65	sym 9	9.84
db8	9.85	sym 10	9.92
db9	9.97	sym 13	9.83
db10	10.08	sym 14	9.56
db13	9.46	sym 17	9.33

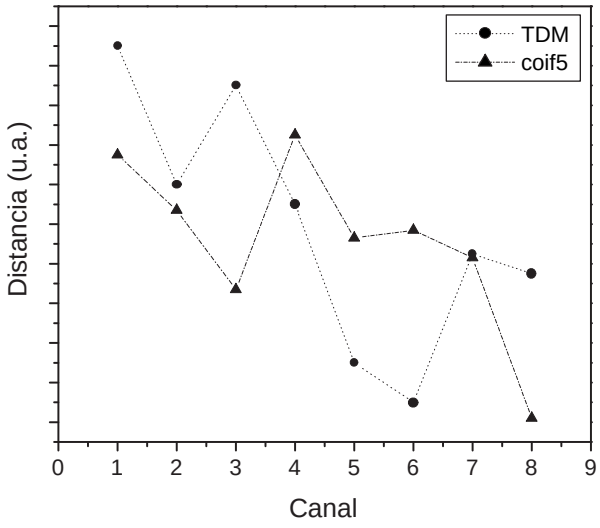


Figura 5.11: Análisis de ruido TDM y WPDM con función wavelet coiflet

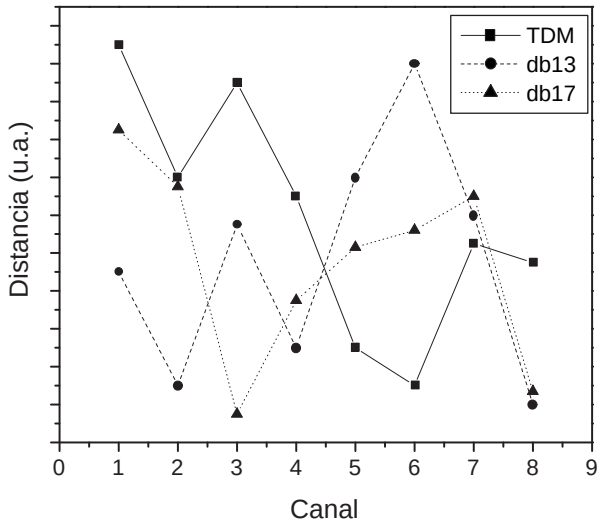


Figura 5.12: Análisis de ruido TDM y WPDM con función wavelet daubechies

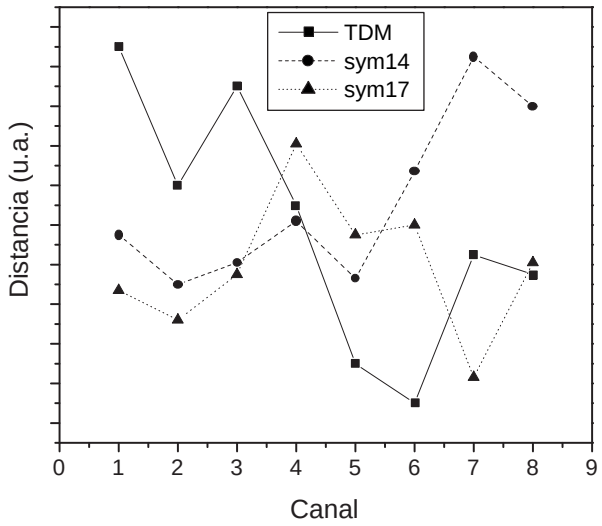


Figura 5.13: Análisis de ruido TDM y WPDM con función wavelet symlet

5.2. En las Figuras 5.14, 5.15, 5.16 y 5.17, se muestran algunos comportamientos de las señales ante el ruido aditivo a diferentes SNR.

Tabla 5.2: Distancia promedio con diferentes SNR

SNR	Distancia promedio coif 5	Distancia promedio db 17	Distancia promedio Sym 17	Distancia promedio TDM
$0dB$	9.73	9.72	9.74	9.87
$3dB$	8.72	8.64	8.67	8.79
$10dB$	6.41	6.42	6.44	6.58
$20dB$	3.96	3.96	3.97	4.06
$45dB$	0.98	0.98	0.98	1.09

De la Tabla 5.2 se concluye que todas las variantes del sistema WPDM tienen una distancia promedio menor a TDM, y por lo tanto ofrecen mayor inmunidad al ruido.

Dado a que las pruebas anteriores están referidas al comportamiento del sistema WPDM que involucra un árbol wavelet y un modulador PAM que hace uso de la función coseno elevado, se analizó el efecto de emplear tanto en el árbol como en el modulador PAM la misma función base wavelet. Los valores de las distancias obtenidas por esta variante se muestran en la Tabla 5.3.

Tabla 5.3: Distancia promedio con modulador wavelet

	Distancia promedio $SNR = 0dB$		Distancia promedio $SNR = 0dB$
coif 1	10.09	db 18	9.66
coif 2	9.82	Sym 2	9.76
coif 3	10.01	Sym 3	10.00
coif 4	9.74	Sym 5	9.97
coif 5	9.90	Sym6	9.85
db 2	9.95	Sym 9	9.89
db3	9.63	Sym 10	9.67
db5	9.67	Sym 13	9.92
db8	9.91	Sym 14	9.69
db 13	9.83	Sym 17	9.94
db 14	9.92	Sym 18	9.80
db 17	9.88	TDM	10.36

Al hacer un análisis comparativo de las Tablas 5.1 y 5.3, se observa una menor

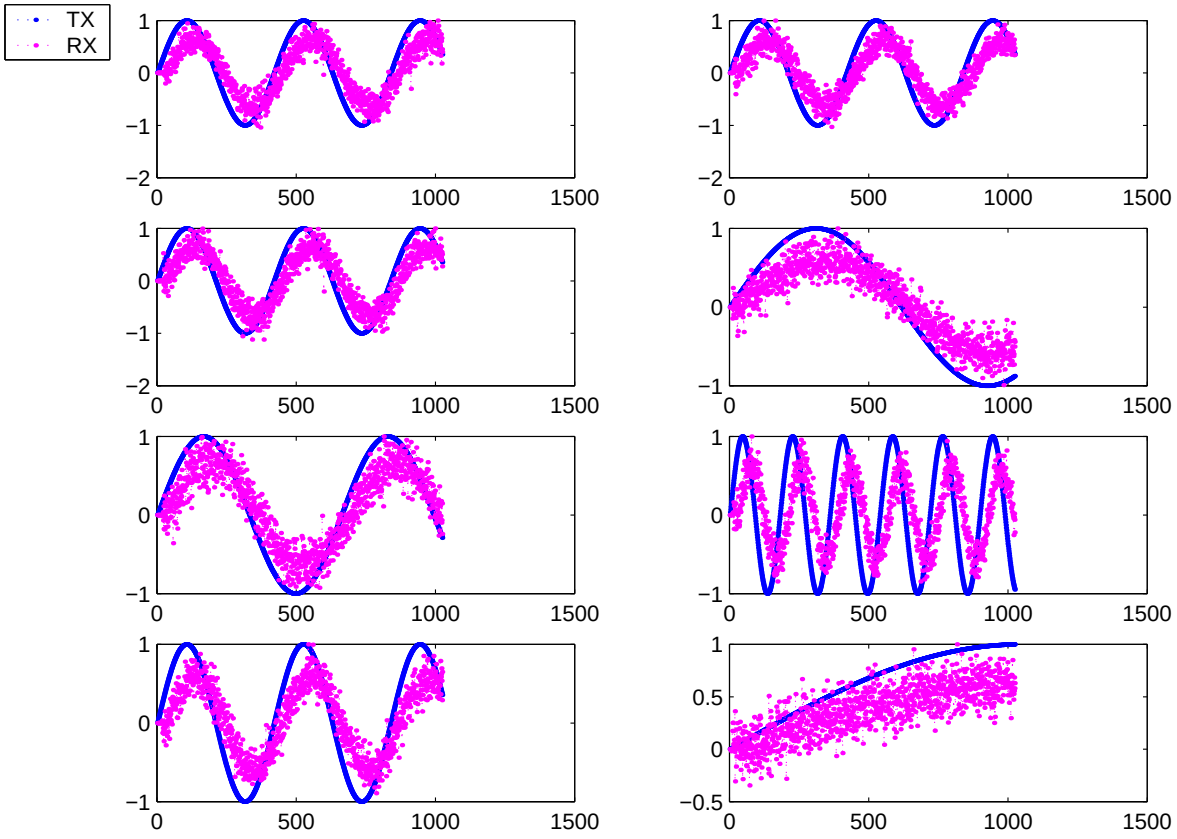


Figura 5.14: Recuperación de los canales en WPDM a 0dB

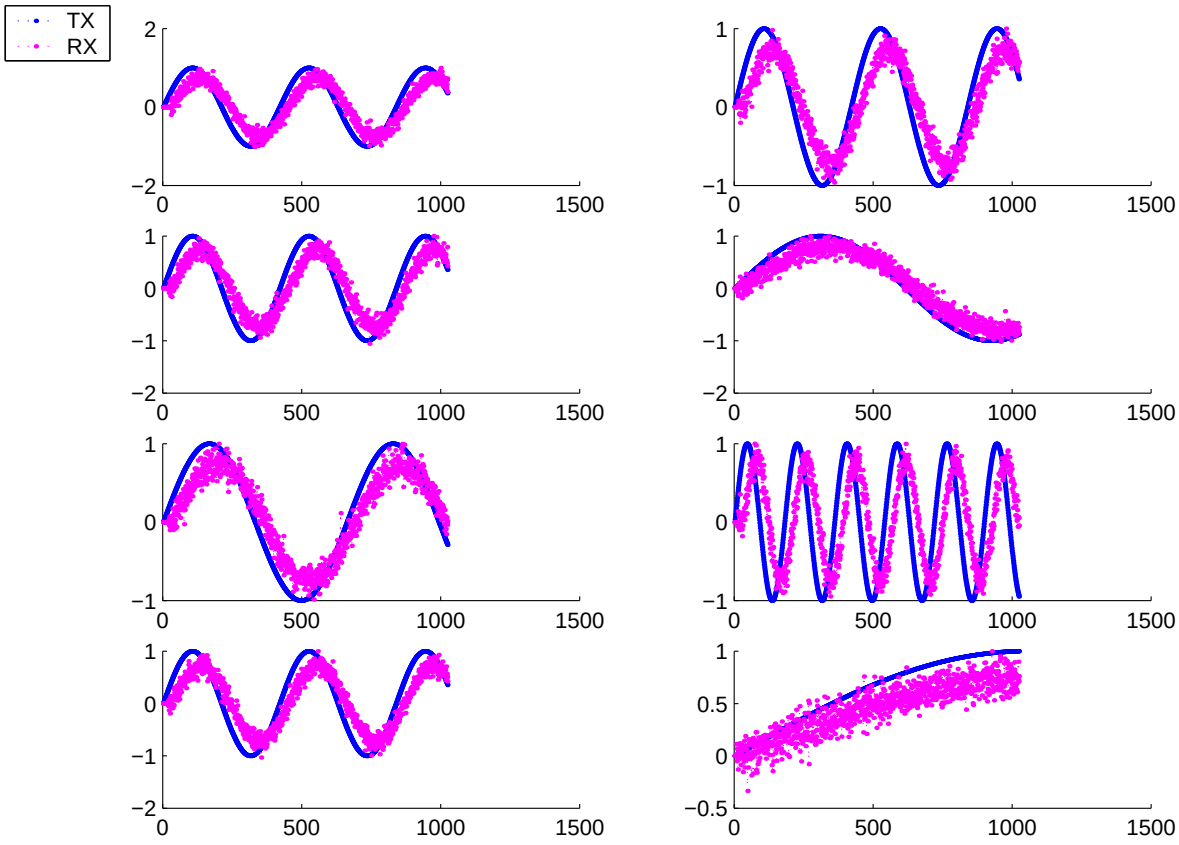


Figura 5.15: Recuperación de los canales en WPDM a 10dB

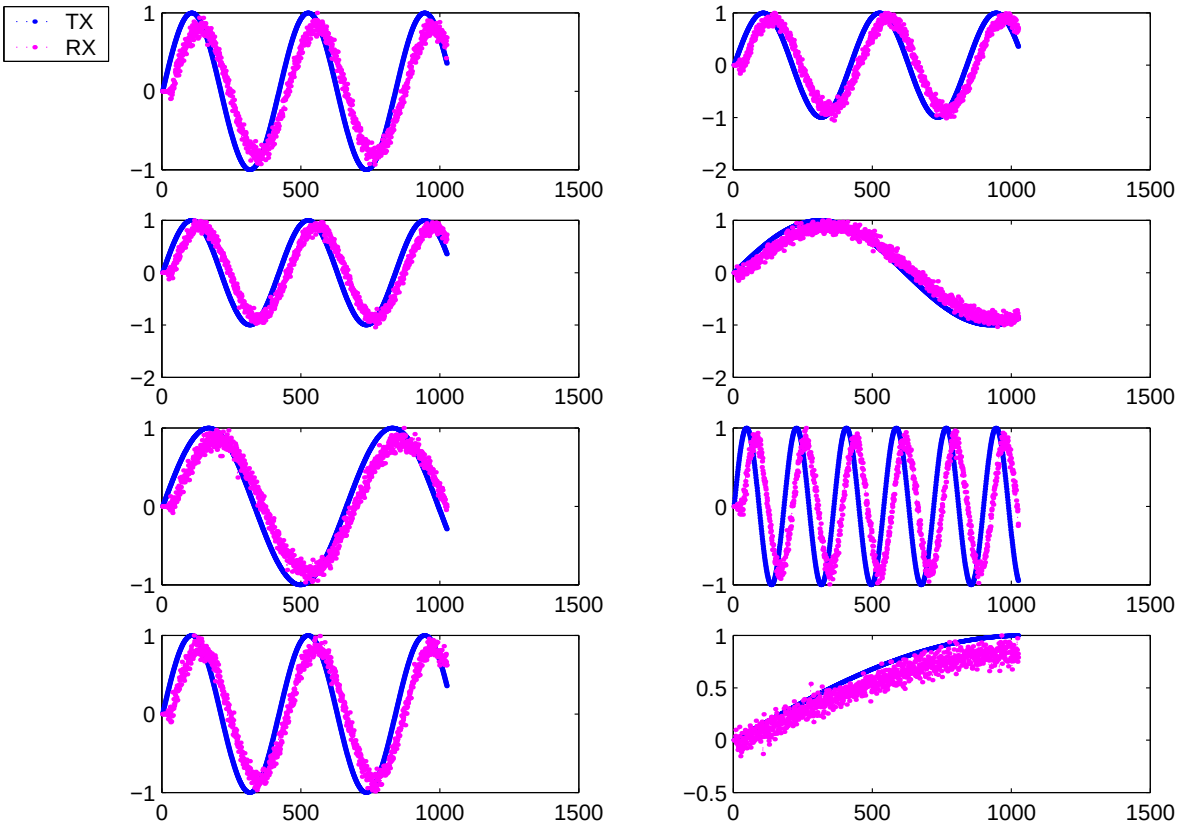


Figura 5.16: Recuperación de los canales en WPDM a 20dB

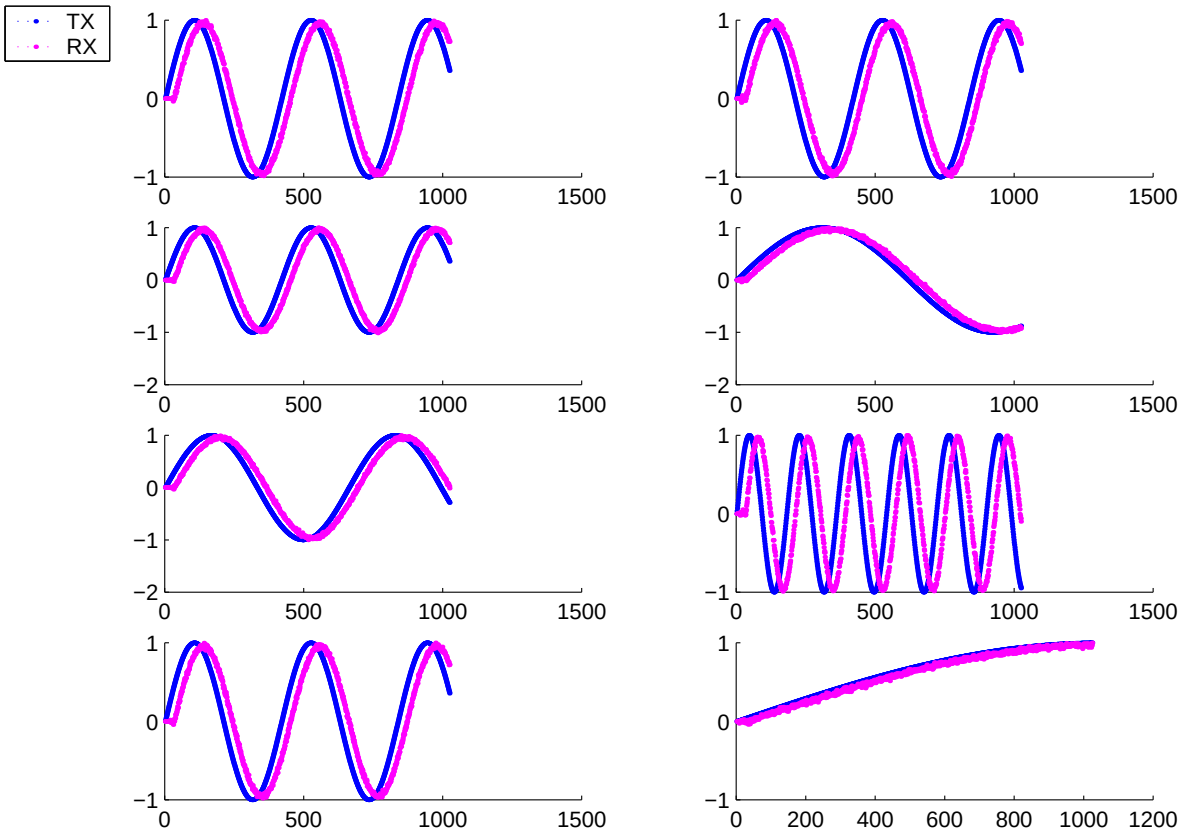


Figura 5.17: Recuperación de los canales en WPDM a 45dB

inmunidad al ruido para el esquema WPDM que emplea un modulador PAM con la función coseno elevado.

5.5.3 Análisis de Ancho de Banda

Para evaluar el desempeño de los sistemas en cuanto al ancho de banda requerido para la transmisión, se seleccionó el sistema FDM como base comparativa con WPDM, porque en éste el ancho de banda total requerido para transmitir los canales multiplexados es igual a $BW = \frac{N_l(1+\delta_g)}{T_l}$, con N_l el número de canales transmitidos, δ_g el ancho de la banda de guarda y T_l la frecuencia de muestreo de cada canal. En esta prueba se utilizaron señales de prueba del tipo $sinc^2(t)$. Se escogió esta forma particular de señal, dado a el espectro de dicha señal es una forma triangular con componentes de frecuencia hasta $f_l/2$.

La prueba consistió en calcular el espectro de la señal multiplexada de WPDM para las diferentes wavelets madres seleccionadas en la sección anterior y analizar que tan eficiente resultra la distribución espectral de los canales. Como puede verse en la Figura 5.18, no existen diferencias apreciables para el ancho de banda de la señal multiplexada ante diferentes funciones wavelet del árbol de síntesis. Si se compara el espectro de la señal multiplexada con el de una señal FDM con banda de guarda y modulación AM-SSB, se encuentra que, aunque FDM hace un uso más eficiente del ancho de banda que el sistema WPDM, el primer presenta menor inmunidad al ruido como consecuencia de que cada uno de los canales se encuentra espectralmente más localizado.

Por otra parte, en la Figura 5.19 se muestra la distribución en frecuencia de cada uno de los canales, para un árbol de síntesis que hace uso de la función base daubechies 17. En esta gráfica se observa claramente el solapamiento de los canales.

Cuantitativamente, en la Tabla 5.4 se presentan los valores de los ancho de banda efectivos para cada una de las variantes de WPDM y FDM. Para las medidas del ancho de banda del sistema WPDM, se utilizó el criterio de seleccionar la región de frecuencias que concentra el 99% de la energía total de la señal multiplexada (Figura 5.18). Los valores del ancho de banda presentados difieren de lo planteado en la Ref. [Wu98, WWDJ97], quienes muestran un ahorro del 30% cuando se utiliza WPDM y señales binarias de entrada.

Tabla 5.4: Ancho de Banda de los sistemas

	Ancho de Banda BW kHz
coif 5	40.25
db 17	40.16
sym 17	39.80
FDM	39

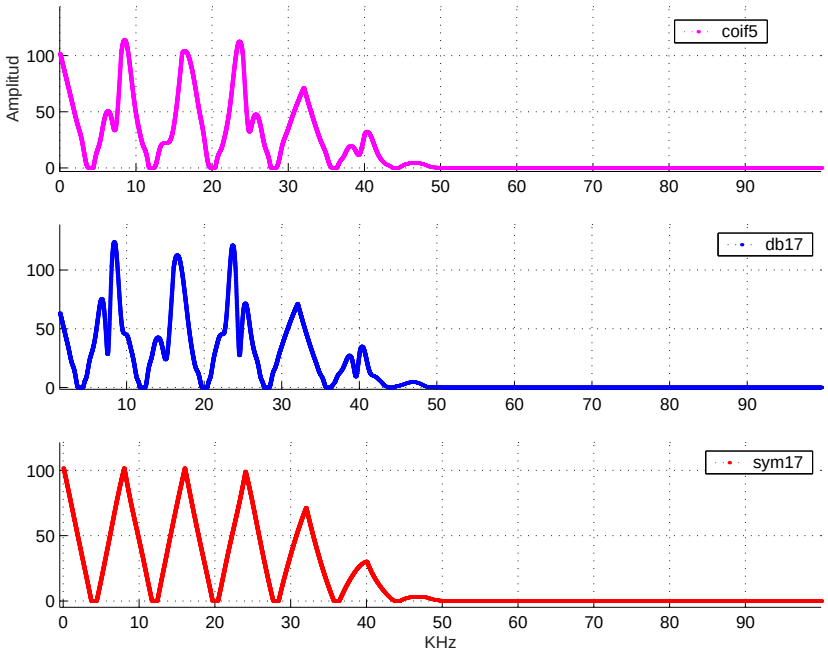


Figura 5.18: Ancho de banda WPDM con las wavelet madre seleccionadas

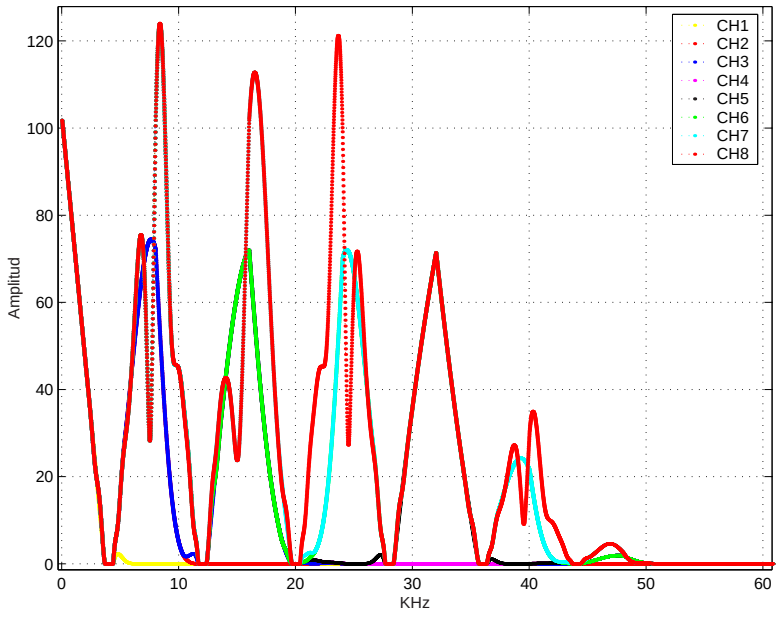


Figura 5.19: Distribución en frecuencia de los canales en WPDM

Experimentalmente se comprobó que la reducción del ancho de banda se logra disminuyendo el factor de interpolación del modulador PAM, sin embargo, esto conlleva a un submuestreo de la señal base PAM (coseno elevado) que no garantiza los cruces por cero que evitan la interferencia intersímbolo. De esta forma, en un sistema WPDM basado en señales multinivel, no es posible conseguir una ahorro del ancho de banda del espectro superior a FDM, ya que de hacerlo se produciría interferencia intersímbolo y una menor inmunidad al ruido

Finalmente, en la Figura 5.20 se presenta el ancho de banda de una estructura WPDM de 4 niveles de descomposición (16 canales uniformes) que hace uso de la wavelet madre db17. Se encuentra que el ancho de banda se duplica, lo cual coincide con lo esperado teóricamente.

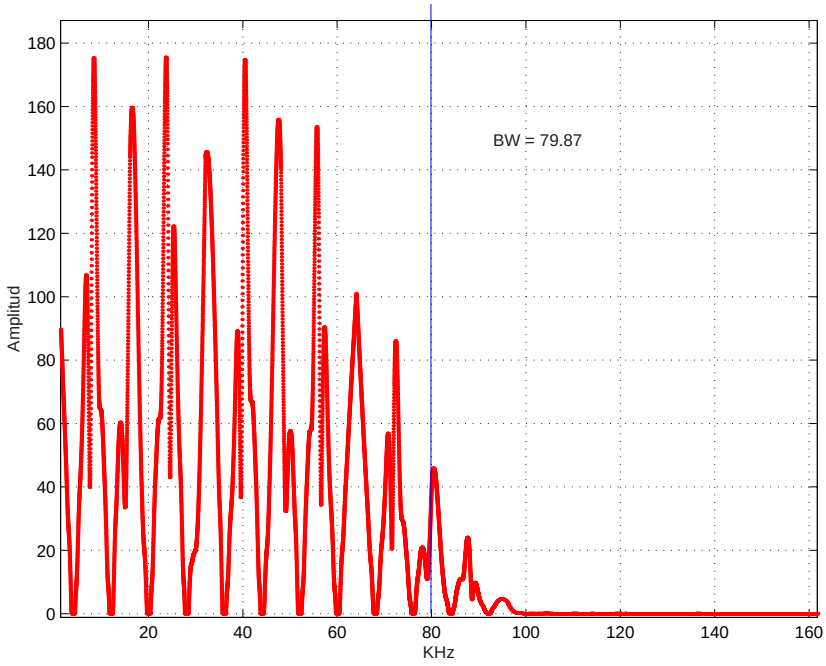


Figura 5.20: Ancho de banda de la WPDM para 16 canales realizada con la Db 17

5.6 Implementación en el DSP

Para la adquisición y reproducción de las señales multiplexadas fue necesario diseñar una tarjeta de adquisición de datos que contara con varios canales simultáneos de entrada y salida, y una frecuencia de muestreo mínima de 250KHz. Para ello se escogieron los circuitos integrados ADS8364, un ADC de 6 entradas de muestreo simultáneo y salida paralela de datos; y el DSD1608, un DAC de 8 canales simultáneos de salida y entrada serial de datos con protocolo compatible con el puerto serial del DSP (McBSP).

Las tarjetas diseñadas, cuyos diagramas esquemáticos se presentan en el anexo G, se conectan a los buses de expansión del DSP. Para la interface con las tarjetas fue necesario escribir una serie de rutinas en lenguaje C que hacen uso de las librerías para captura de datos por DMA que ofrece el entorno de desarrollo del CodeComposer.

Debido a las características del ADC, las pruebas de desempeño presentadas en esta sección están restringidas a la multiplexación de cuatro canales de igual frecuencia de muestreo, es decir, dos niveles de descomposición. Aunque se podría aumentar a 6 canales por medio del empleo de un árbol wavelet asimétrico, el ADC utilizado en la tarjeta no permite hacerlo, puesto que todos los canales deben tener la misma frecuencia de muestreo.

Por otra parte, para determinar si el sistema implementado funciona en tiempo real, fue necesario realizar una medición de los tiempos de ejecución de los algoritmos sobre el DSP, empleando las herramientas de depuración que ofrece el entorno de desarrollo CodeComposer. Cabe resaltar que el DSP trabaja a una frecuencia de reloj de $133MHz$ que implica un tiempo de $t_{clk} = 7.5ns$ por cada ciclo de reloj. Para conocer el número máximo de operaciones que se pueden realizar en el sistema para garantizar un comportamiento en tiempo real, hace necesario determinar inicialmente, la frecuencia de muestreo del DAC empleado para la transmisión de la señal multiplexada. Para lo cual se parte de que:

$$fs = n_{canales} \times fs_{canal} \times I_{PAM} \tag{5.9}$$

$n_{canales}$ e I_{PAM} son conocidos y el DAC DS1608 impone una restricción máxima de frecuencia de $200KHz$. Con estos parámetros se seleccionó para el sistema de multiplexación una $f_s = 180KHz$ que limita la frecuencia de muestreo de cada canal a $5KHz$. De esta forma, el tiempo máximo permisible en el cual se debe sintetizar la señal multiplexada y realizar el respectivo proceso de demultiplexación, debe ser igual a

$$T_{max}(N) = \frac{n_{canales} \times I_{PAM} \times N}{fs} = \frac{N}{fs_{canal}} \tag{5.10}$$

que expresado en número de ciclos de máquina es:

$$N_{ciclos}(N) = \frac{T_{max}(N)}{t_{clk}} = \frac{N}{fs_{canal} \times t_{clk}} \tag{5.11}$$

Si consideramos que $N = 1024$, el número de ciclos máximo en el que debe ejecutarse la rutina debe ser $N_{ciclos} = 27$ millones de ciclos.

Para el cómputo de los paquetes wavelet se puede hacer uso de las rutinas optimizadas que se describieron en la sección *3.3.1* y el anexo *A*, que se basan en el cálculo por células básicas. Sin embargo, es importante indicar que la versión más optimizada requiere una longitud de la respuesta al impulso divisible por cuatro, lo cual no se garantiza con la función que mejor desempeño arrojó, la daubechies 17. Para hacer uso de este algoritmo, fue necesario llevar a cabo una extensión del vector de la respuesta al impulso

a 36 coeficientes, donde los primeros 34 son la versión reflejada del filtro wavelet[2] y los dos últimos son ceros.

Respecto al modulador PAM, en la sección 5.2 se indicó que éste se puede implementar por medio de la rutina optimizada de la sección *3.3.2* y el anexo *C*. Estas rutinas están optimizadas para longitudes de la respuesta al impulso y factores de interpolación y diezmado divisibles por 4. El primer aspecto se puede garantizar fácilmente por medio de la extensión con ceros de los vectores de la respuesta al impulso, mientras que el segundo no, pues el factor de interpolado/diezmado es 9. Por tal motivo, se hizo necesario adaptar dichas rutinas para soportar factores de interpolación y diezmado por 9. Sin embargo, no fue posible conseguir una ejecución tan optimizada como la del interpolador/diezmador por factor D múltiplo por cuatro.

Con las modificaciones realizadas a los algoritmos, y asumiendo un árbol de descomposición wavelet de dos niveles, función wavelet madre daubechies 17 y modulador PAM con la función coseno-elevado, se encontraron los tiempos reportados en las Tablas 5.6 y 5.5. Nótese que, a pesar de las optimizaciones, el demultiplexor WPDM no es posible implementarlo en tiempo real, lo cual solamente es factible con el empleo de un procesador más veloz.

Tabla 5.5: Tiempo de ejecución del transmisor WPDM

Rutina	Número de ciclos	Tiempo (ms)
IPW	6.796.000	51
Modulador PAM	17.004.000	128
TOTAL	23.801.000	179

Tabla 5.6: Tiempo de ejecución del receptor WPDM

Rutina	Número de ciclos	Tiempo (ms)
PWD	7.084.000	53
Demodulador PAM	30.799.000	231
Sincronización	747.000	6
TOTAL	38.630.000	290

[2]otra condición del algoritmo de cómputo de los paquetes wavelet por células básicas es que los vectores con los coeficientes wavelet deben escribirse en forma reflejada

5.7 Conclusiones

Se mostró la factibilidad de empleo de la técnica WPDM para la multiplexación de señales multinivel, encontrándose que, para transmitir señales de voz, se puede usar un sistema híbrido que involucre un paquete wavelet en conjunto con un modulador PAM basado en la función coseno elevado.

A partir del estudio realizado bajo las diferentes condiciones de relación señal a ruido SNR, se evalúo el desempeño de las técnicas WPDM y TDM, encontrándose que el sistema WPDM, ofrece un mejor desempeño para todas las funciones madre (coif5, db17 y sym 17) empleadas en el paquete wavelet de síntesis y descomposición. De acuerdo a las pruebas de ruido y ancho de banda realizadas en el sistema WPDM, se determinó que la función wavelet madre con mejor desempeño en el árbol de descomposición, es la db17.

De acuerdo a los reportes de la literatura en WPDM para señales binarias, se consigue un ahorro considerable de ancho de banda de la señal multiplexada, sin embargo, para señales multinivel no fue posible obtenerlo, lo que nos condujo, en todos los casos, a un desempeño comparable al ancho de banda de la señal FDM.

Se determinó teórica y experimentalmente que para el correcto funcionamiento de un sistema WPDM, en tiempo real, se requiere de una fase de compensación que garantice las condiciones de reconstrucción perfecta del filtro QMF inverso; para lo cual se propuso un nuevo esquema de sincronización, que emplea uno de los canales de comunicación como canal de sincronización.

Capítulo 6

Sistema de encriptación de señales

6.1 Panorama de los Sistemas de Encriptación de Señales

Actualmente, los sistemas de mayor difusión y amplio uso para encriptar la información se basan en la criptografía digital. Dicha masificación se debe al aumento en la prestación de servicios y oferta de productos distribuidos por las redes de datos, en especial Internet, y a su facilidad de implementación en arquitecturas digitales. Entre los métodos de criptografía digital, existen dos tendencias, los basados en clave pública y clave privada. Del primer tipo, se destacan el RSA y el EDS, y del segundo tipo el DES e IDES[LCL02]. Algunos de estos, han mostrado ser vulnerables, y con el aumento en la velocidad y capacidad de cómputo de los procesadores actuales, ciertos métodos de criptografía digital que se pensaban incorruptibles, lo pueden llegar a ser. Es así como se realizan esfuerzos por generar sistemas de criptográfica alternativos, basados en señales análogas, los cuales prometen tener una mayor seguridad, debido a la alta redundancia, y a la posibilidad de transmisión por medios diferentes a las redes de datos convencionales.

Entre los sistemas de encriptación analógica se destacan los basados en caos [PC90, COS93, GM99, SY00, Sil96] y los que emplean transformadas [Teo98, AM99].

Los sistemas de encriptación con caos fueron propuestos inicialmente por Pecora y Carroll en 1990 [PC90], como resultado del descubrimiento de la posibilidad de sincronización de dos sistemas caóticos. El caos ha sido atractivo para la encriptación de información, debido a que las señales caóticas tienen un ancho de banda infinito y son determinísticas, pues dependen fuertemente del sistema caótico y su condición inicial. A partir del trabajo de Pécora y Carroll, se han propuesto diversos esquemas de encriptación[COS93, SY00, GM99], cuyo principio básico es la adición a la señal-mensaje de una señal caótica producida por un sistema dinámico (Figura 6.1). En esta técnica, conocida con el nombre de *enmascaramiento* se debe garantizar una relación señal a ruido (SNR) baja, con el fin de conseguir una alta confidencialidad en la transmisión. Por otra parte, para llevar a cabo la sincronización se hace necesario transmitir una variable de estado del sistema, de allí que la confiabilidad y seguridad de estos esque-

mas haya sido debatida por ciertos autores [DO97, AMRP00a, AMRP00b], quienes han demostrado que es posible deducir las propiedades del sistema dinámico a partir de la señal encriptada, empleando técnicas de filtrado adaptativo. A pesar de ello, han aparecido recientes esquemas de encriptación basados en hipercaos y observadores lineales [GM99, PCK$^+$92, **?**] que vislumbran ser más robustos y confiables que sus predecesores. Una visión sobre lo que es el caos y sus principales aplicaciones en los sistemas de comunicación puede encontrarse en la Ref.[SY00].

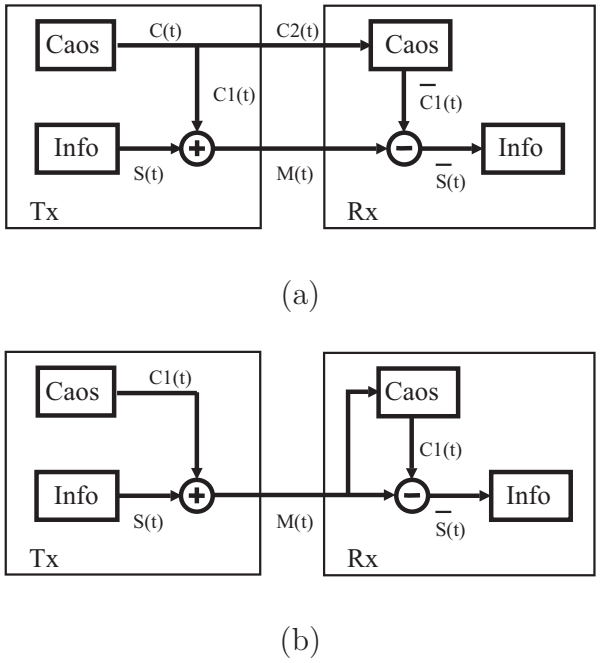


Figura 6.1: Diferentes esquemas para encriptación con caos. Propuestos por: (a) Pécora y Carroll[PC90] y (b) Cuomo et al.[COS93]

Por otra parte, en lo referente a los sistemas basados en transformadas, se ha mostrado que es posible emplear la transformada wavelet discreta (DWT) para generar una señal en espectro esparcido [AM99, HSD95], o la transformada wavelet sobre-completa (OCWT) [Teo98]. Ambos sistemas están orientados a la transmisión de bits, y sus principales ventajas frente a los esquemas de encriptación con caos, son la mayor facilidad de implementación y la sincronización.

El primer esquema, ha tenido buena aceptación en aplicaciones militares, y se fundamenta en que los coeficientes de los filtros de síntesis y reconstrucción se calculan a partir de métodos de optimización que emplean criterios de ortogonalidad y baja probabilidad de intercepción (*LPI: Low Probability of Intercept*). En este esquema de encriptación, mostrado en la Figura 6.2, se hace uso de la transformada wavelet discreta inversa para generar la señal encriptada, empleando como coeficientes de entrada, el mensaje codificado por medio de un modulador Walsh y una secuencia pseudoaleatoria (PN). En

el receptor, la información se recupera a partir de una transformada wavelet discreta directa, y un demodulador Walsh, que recupera la información a partir del producto entre los coeficientes entregados por la DWT y la la secuencia pseudoaleatoria.

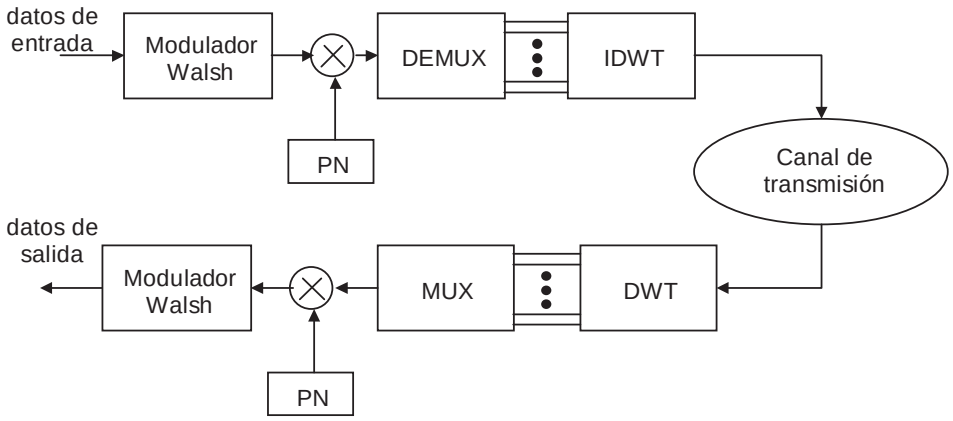


Figura 6.2: Sistema de encriptación propuesto por Orr et al.[AM99]

El segundo método tiene la flexibilidad de poder emplear cualquier función wavelet madre, sin la necesidad de satisfacer condiciones de ortogonalidad de las funciones base. Al igual que el primer sistema, se emplea la transformada inversa para sintetizar la señal encriptada y la transformada directa para recuperar la información. En este esquema (Figura 6.3), los bits a transmitir se ubican en ciertas posiciones preferenciales de la matriz de coeficientes que se le proporciona como entrada a la IOCWT, y para la detección de los bits de información a partir de la señal de salida de la OCWT, se determina la energía en la región frecuencia-tiempo asociada a la dispersión del bit transmitido, si esta toma un valor superior a cierto umbral, el bit se identifica como un uno, en caso contrario es un cero.

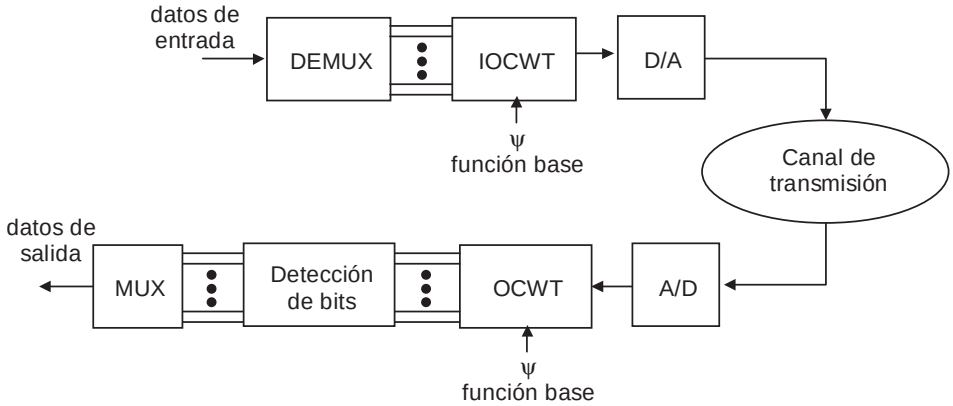


Figura 6.3: Sistema de encriptación propuesto por Teolis[Teo98]

En este trabajo se propone combinar las técnicas de encriptación de señales análogas basadas en caos y transformadas, por medio del empleo de la transformada wavelet sobre-completa como núcleo básico de síntesis de la señal encriptada y una función wavelet pseudocaótica como función base. Se emplea la OCWT dado a que la función wavelet pseudocaótica que se propone no forma una base ortogonal, además, el término pseudocaótico se debe a que la señal wavelet madre se genera a partir de un sistema caótico en tiempo discreto.

En la literatura se encuentran algunos ejemplos de sistemas híbridos que hacen uso de wavelets y señales caóticas para encriptar imágenes [LCL02, XJZW02]. Estos sistemas han sido diseñados para la transmisión segura a través de una red de datos o para la autenticación y el firmado digital de imágenes (*watermarking*). Estos y otros sistemas, emplean la técnica de enmascaramiento para la codificación de los coeficientes wavelets, idéntico al esquema de encriptación de señales unidimensionales propuesto por Pécora y Carroll (Figura 6.1). De esta forma, en el transmisor, a los coeficientes wavelet que entrega la transformada wavelet directa se les suma una señal caótica empleando un valor de relación señal a ruido pequeño, y en el receptor se procede a la substracción entre los coeficientes wavelet recibidos y la señal caótica, y el posterior cálculo de la transformada wavelet inversa. Por otra parte, sistemas como el de Luo et al.[LCL02], por ejemplo, no emplea la transformada wavelet como tal, sino de una matriz de transformación construida a partir de la función wavelet Haar.

Existen reportados otros sistemas más sofisticados de encriptación de imágenes basados en wavelets y caos, los cuales hacen uso de la transformada wavelet sobre campos finitos[CF04], o la codificación de los coeficientes wavelet de una imagen por medio de técnicas de criptografía digital convencional[CFW01, HW03].

Este esquema de encriptación propuesto en este trabajo, difiere de los esquemas híbridos reportados en la literatura en cuanto a que no hace uso del método de enmascaramiento o la transformada wavelet sobre campos finitos, sino de la síntesis de una señal encriptada a partir de la IOCWT y una función base generada por medio de un sistema caótico. Es así como, el sistema aquí propuesto resulta ser una alternativa que aún no ha sido explorada.

6.2 Sistema de Encriptación Propuesto

El esquema de encriptación propuesto en este trabajo se ilustra en la Figura 6.4. En éste, la información a encriptar se introduce al sistema como los coeficientes wavelet, y para facilitar el proceso de decodificación, dicha información debe ser una trama de bits. Con este sistema, a la salida de la IOCWT se obtiene una señal unidimensional que puede ser fácilmente transmitida por algún método análogo y/o digital, y en el receptor se emplea la OCWT para recuperar los bits de información. El demultiplexor que se muestran en el diagrama de bloques tienen por objeto ubicar los bits de la señal unidimensional de entrada en las posiciones designadas en la matriz 2D de coeficientes wavelet para su respectiva transmisión, y el multiplexor, se encarga de realizar el procedimiento contrario

en el receptor.

Aunque este sistema es similar al planteado por Teolis[Teo98] (Figura 6.3), difiere de éste en cuanto a que dicho autor realizó únicamente simulaciones al sistema de encriptación empleando las funciones wavelet madre de Morlet, BSpline y PBL, y no propone ningún tipo de esquema de sincronización que permita implementarlo en tiempo real, tal como se presenta en este trabajo.

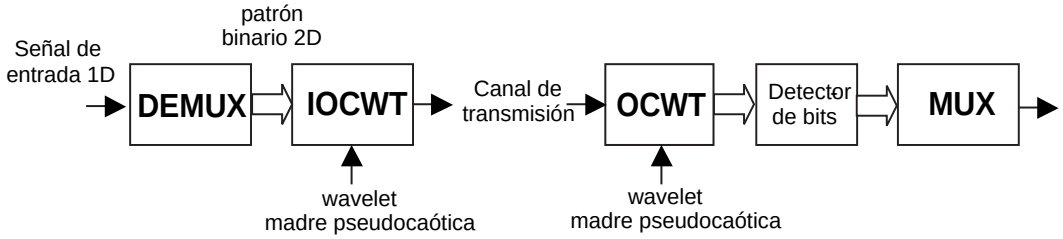


Figura 6.4: Diagrama de bloques genérico del sistema de encriptación propuesto

Dependiendo de la forma de transmisión de la señal encriptada proponemos dos variantes para el sistema. La primera, consiste en la generación de una señal análoga a partir de la conversión digital-análoga de la señal sintetizada con la IOCWT, con el fin de ser enviada por un radiotransmisor, y el empleo de un radiorreceptor y conversor análogo-digital en el receptor. En esta propuesta, los bits a transmitir se ubican en posiciones fijas y predeterminadas de la matriz de coeficientes 2D. Para garantizar la adecuada recuperación, es necesario garantizar una alta inmunidad al ruido y la perfecta sincronización entre el transmisor y receptor (Figura 6.5). Hacia este último aspecto están orientadas la mayoría de las pruebas descritas en la sección 6.3 y el sistema de sincronización presentado en el numeral 6.2.4.

En esta propuesta la clave de encriptación está compuesta por: la serie caótica que se emplee para la generación de la función base $\psi_0[n, \psi_0(0)]$, la condición inicial $\psi_0(0)$ y los parámetros de preprocesamiento $\mathbf{w}_0$ de dicha serie, y la posición escala-tiempo en la cual se ubican los bits de entrada a la IOCWT $(\mathbf{s}, \tau)$. Cada uno de estos términos será ampliado en detalle en la siguiente sección.

La segunda alternativa, está orientada hacia la transmisión de paquetes por una red de datos, por ejemplo, Internet. En este esquema, no es necesario incluir algún tipo de sincronización y se puede asegurar una mayor inmunidad ante ataques, por medio de la reasignación dinámica, en cada paquete transmitido, de las posiciones de los bits en la matriz 2D que entrada a la IOCWT (Figura 6.6). De esta forma, cuando se transmite un paquete, tanto el receptor como el transmisor modifican las posiciones en las cuales transmiten los bits, lo cual se puede realizar por medio una segunda serie caótica tal como se describirá en la sección6.2.5.

Por medio de este esquema, la clave de encriptación se hace más compleja, y difícil de determinar, pues además de la serie caótica $\psi_0[n, \psi_0(0)]$, su condición inicial $\psi_0(0)$ y los parámetros de preprocesamiento $\mathbf{w}_0$, se requiere el conocimiento de la segunda serie

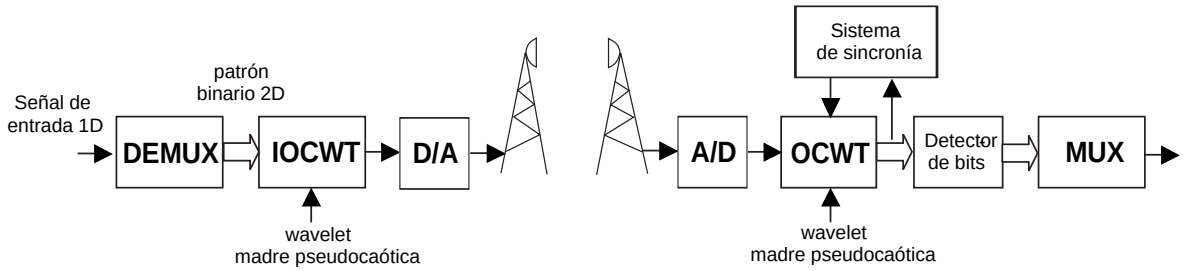


Figura 6.5: Diagrama de bloques del sistema de encriptación orientado a la transmisión análoga

$\mathbf{p}(n, p(0))$ y su condición inicial $p(0)$ que genera las posiciones escala-tiempo en la cual se ubican los bits de entrada a la IOCWT.

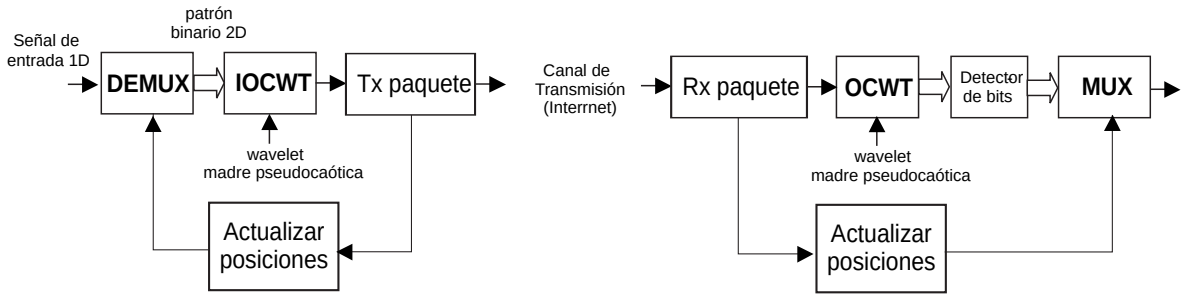


Figura 6.6: Diagrama de bloques del sistema de encriptación orientado a la transmisión digital

Las diferentes optimizaciones y en especial énfasis la implementación en tiempo real sobre el DSP, están orientadas a la primera variante, en cambio, la segunda, puede ser implementada sobre un PC, independiente de un DSP, y haciendo uso de sockets para la transmisión de los datos. Por lo anterior, las pruebas de desempeño de tiempos indicadas en este trabajo corresponden a la primera variante del sistema de encriptación.

6.2.1 Generación de la wavelet madre pseudocaótica

La wavelet madre pseudocaótica se generó con el sistema dinámico en tiempo discreto

$$x[k] = x[k-1]^2 + c \tag{6.1}$$

tomando $c = -1.95$ y diferentes condiciones iniciales. El valor de c se determinó a partir del diagrama de bifurcación y los coeficientes de Lyapunov. Se escogió esta expresión en particular dado a que se trata de un sistema dinámico en tiempo discreto y el conjunto de datos de salida oscilan alrededor del cero, requisito indispensable para ser una función wavelet (ver sección 3.1).

A partir de esta función se generan 128 elementos, que se someten a un proceso de preprocesado, consistente en la interpolación por un factor de 8, que permite obtener un vector con una longitud de 1024 elementos; paso siguiente, se enventana y filtra con un filtro pasa-banda IIR de segundo orden. El tomar tan solo 128 elementos se debe a que el contenido de frecuencia de dos series caóticas con diferente condición inicial tienen a ser similares para una longitud grande. Por otra parte, el objeto del enventanamiento es el de concentrar la función wavelet madre en el tiempo, con el fin de garantizar que la función madre sea una señal de energía; y el propósito del filtrado es reducir el ancho de banda de la serie, puesto que de no hacerlo se introduce una alta redundancia en la información de todas las escalas que imposibilita el proceso de recuperación de la información. La frecuencia central y ancho de banda de dicho filtro se calculan a partir del valor del último elemento de la serie caótica original.

Con el método descrito anteriormente se genera la wavelet madre para la escala más baja ψ $(m = 0)$, y para las restantes escalas, las diferentes wavelet ψ_m, se obtienen por medio del diezmado por un factor no entero $s_m = a_0^{m1}$:

$$\psi_m[n] = \sqrt{s_m}\psi[s_m n] \tag{6.2}$$

y la extensión a un nuevo vector de 1024 datos, por medio de la adición de ceros. En la Figura 6.7 se muestran las funciones wavelets base para tres diferentes escalas, así como sus respectivas transformadas de Fourier. En particular, para todas las pruebas que se presentan en este documento, se escogió un factor de escala $a_0 = 1.04$, sin embargo puede tomar cualquier otro valor, y sirve como parámetro para conformar la *clave de encriptación*. Se recomienda el empleo de un $1 \leq a_0 \leq 2$, pues con este rango de valores se garantiza una alta redundancia en las escalas.

Debido al principio de incertidumbre, cada una de las funciones base presenta una dispersión en frecuencia y tiempo que reduce dramáticamente el número de bits que se pueden transmitir por cada bloque de datos, comparado con el límite máximo ideal de 1024 por el número de escalas. Estas dispersiones (σ_n: dispersión en el tiempo y σ_Ω: dispersión en la frecuencia) se pueden calcular con las expresiones dadas en la Ref. [AM99]:

$$\begin{aligned} \sigma_n^2 &= \frac{1}{E}\sum_n (n-\bar{n})\,|\psi_m[n]|^2 \\ E &= \sum_n |\psi_m[n]|^2 \\ \bar{n} &= \frac{1}{E}\sum_n n\,|\psi_m[n]|^2 \end{aligned} \tag{6.3}$$

y

[1]Con esta forma de definir las escalas, baja escala significa una wavelet muy dispersa en el tiempo y altamente concentrada en la frecuencia, y alta escala, una wavelet angosta en el tiempo y altamente dispersa en la frecuencia.

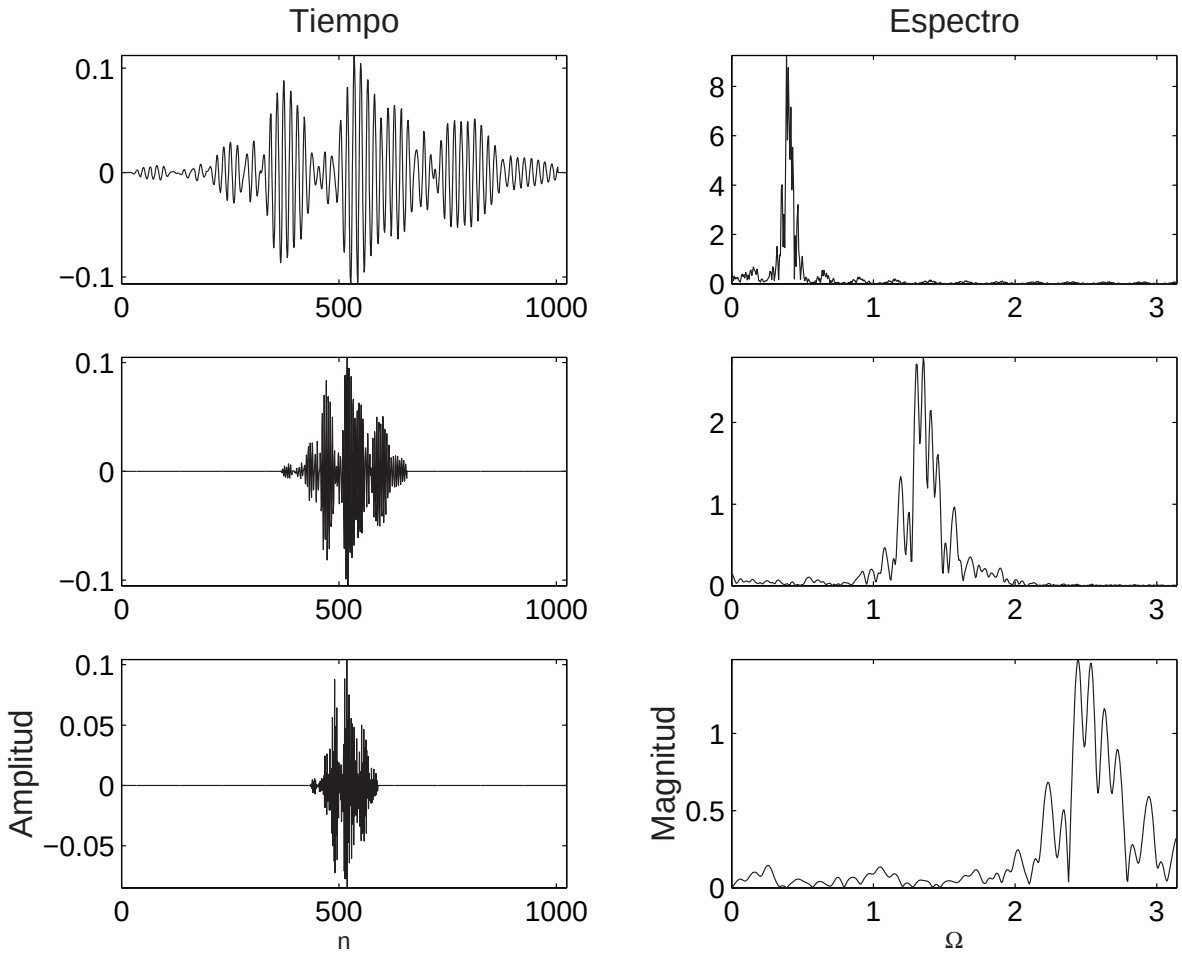


Figura 6.7: Wavelets pseudocaóticas para diferentes escalas: $m = 0$ (superior), $m = 25$ (centro) y $m = 64$ (inferior)

$$\sigma_{\Omega}^{2} = \frac{1}{2\pi E}\int_{-\pi}^{\pi}(\Omega-\bar{\Omega})\left|\Psi_m(\Omega)\right|^2 d\Omega \tag{6.4}$$
$$\bar{\Omega} = \frac{1}{2\pi E}\sum_{n}\Omega\left|\Psi_m(\Omega)\right|^2$$

Las expresiones anteriores se emplean para identificar las posiciones en escala y tiempo en las cuales es posible transmitir los bits, es decir, las posiciones en la matriz 2D que requiere la IOCWT, pues no todas las posiciones de la matriz son posibles, ya que se producirían traslapes en la descomposición que realiza la OCWT del receptor, imposibilitando el proceso de desencriptación con el método que se describirá más adelante.

Aunque el número de bits por trama se puede aumentar con la selección de wavelets madre concentradas en el tiempo y escala, su uso no es práctico en un sistema de encriptación, en el cual se requiere alta redundancia y traslape entre escalas y tiempo, con el fin asegurar una buena confiabilidad del sistema. Por esta razón, para generar la señal pseudocaótica, el filtro pasa-banda se escoge con una baja selectividad y se seleccionan las escalas más bajas para transmitir la información, debido a que espectralmente estas funciones base se encuentran más dispersas en frecuencia y más concentradas en el tiempo (ver figura 6.7). Se empleó también como criterio de selección de las escalas, la menor razón de bits erróneos (BER) ante diferentes relaciones señal/ruido en la señal encriptada.

Para ilustrar el efecto de la dispersión en las escalas y tiempos, se presenta en la Figura 6.8 un ejemplo de una señal encriptada con el esquema propuesto y el respectivo mapa escala-tiempo que se detecta en el receptor.

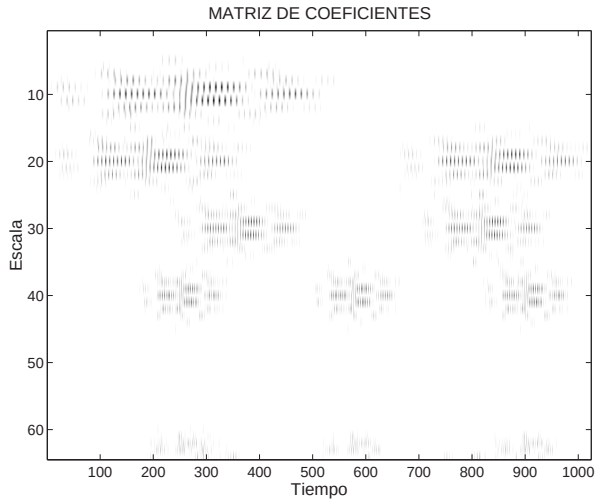


Figura 6.8: Efecto de la dispersión en frecuencia-tiempo de una cadena binaria encriptada con el sistema propuesto

6.2.2 Generación de la señal encriptada

Para generar la señal encriptada, se escriben los bits a transmitir en una matriz bidimensional de 1024 elementos por el número de escalas seleccionadas, que se usa como entrada al algoritmo de cálculo de la IOCWT (ec. 3.22). En esta matriz, inicializada con ceros, los bits se ubican en las posiciones indicadas por la dispersión en el tiempo (ec. 6.3) de la función wavelet madre asociada a la escala. Por cada una de las matrices se obtiene un vector unidimensional de 1024 datos que, se envía al conversor D/A o se empaqueta para ser transmitido por una red de datos. Este proceso se repite en forma indefinida.

Tanto para el cálculo de la IOCWT como la OCWT se empleó el algoritmo de cómputo optimizado descrito en la sección 3.4.

Por otra parte, para llevar a cabo el proceso de sincronización de la transmisión, que requiere la variante para la transmisión de señales análogas, por cada uno de los bloques de 1024 datos calculados, se transmite un único bit en una escala dada, denominada *escala de señalización*, que permite determinar en el receptor el inicio del bloque de datos, haciendo uso del algoritmo que presenta en la sección 6.2.4. Se escoge como escala de señalización, la más baja, pues al ofrecer una mayor dispersión en el tiempo no sería práctica para la transmisión de una alta densidad de bits.

6.2.3 Des-encriptación de la señal

Tal como se ilustra en la Figura 6.4, la recuperación de la información se consigue por medio de una OCWT. Para el cálculo de la OCWT se incluyen algunas de las escalas que no fueron tenidas en cuenta en el proceso de generación de la señal caótica, con el fin de considerar la dispersión en frecuencia que sufren los coeficientes wavelet, favoreciendo el proceso de detección de bits en el decodificador y aumentando la confiabilidad del sistema. La matriz recuperada se somete a un proceso de reducción de ruido por medio de la técnica de *wavelet denoising* de umbral fuerte[RB98].

Una vez el sistema está sincronizado, los bits de información ($r(s_i, \tau_i)$) se recuperan por medio del cálculo de la energía en la región de dispersión del bit. Las dimensiones de dicha región de dispersión pueden calcularse por medio de las ec. (6.3) y (6.4). Si dicha energía es superior al umbral de detección, el bit se considerará como uno, en caso contrario se considera cero. Esta forma de detección de los bits se conoce con el nombre de *detección radiométrica*[AM99] y se denota matemáticamente como:

$$r(s_i, \tau_i) = \begin{cases} 1 & E_i \geq \delta_n \\ 0 & E_i < \delta_n \end{cases} \tag{6.5}$$

con δ_n el umbral de detección y E_i, la energía en la región de dispersión:

$$E_i = \sum_{s=s_i-\sigma_{\Omega i}/2}^{s_i+\sigma_{\Omega i}/2} \sum_{\tau=\tau_i-\sigma_{ni}/2}^{\tau_i+\sigma_{ni}/2} c[s,\tau]^2 \tag{6.6}$$

6.2.4 Sincronización

Como se indicó en la sección 6.2, en el sistema de encriptación basado en la transmisión analógica de la señal, es necesario sincronizar el receptor y transmisor, con el fin de poder interpretar correctamente la matriz de salida de la OCWT, para lo cual se hace necesario identificar el inicio y fin de la trama. Esto se consigue por medio de la detección del bit de sincronía que se transmite en la escala de señalización.

Gracias a que la OCWT es una transformada invariante a la translación, es de suponer que, aún bajo condiciones de de-sincronización, es posible identificar la posición del bit de sincronía.

Dado que el bit transmitido se identifica en el receptor por medio de una forma de onda dispersada en el tiempo, el proceso de búsqueda de dicho bit se realiza por medio del cálculo del centro de masa de los coeficientes wavelet de la escala de señalización para dos bloques consecutivos de 1024 elementos. Una vez se identifica el centro de masa, el inicio de la trama se encuentra localizado un medio de la longitud de dispersión en el tiempo de la función base de dicha escala, y a partir de dicho punto se calculan las energías en cada una de la regiones de dispersión previamente conocidas.

Para ilustrar el funcionamiento de dicha técnica se muestra en la Figura 6.9 dos tramas consecutivas de 1024 elementos calculados en la OCWT del receptor. Si el sistema está desincronizado, es de esperarse que el centro de masa de la señal recibida en la escala de señalización esté ubicado en algún punto intermedio entre las dos tramas. Una vez se identifica el centro de masa, las regiones de dispersión en las cuales se calcula la energía para determinar cada uno de los bits es una tarea muy simple, pues los límites de dichas regiones se encuentran precalculados en una plantilla con coordenadas relativas al bit de sincronización.

El algoritmo del sistema de sincronización en notación de pseudocódigo se presenta a continuación:

```
receptor(
    sig,                  ▷ Señal de entrada
    lentrada,             ▷ Longitud en el tiempo de los coeficientes wavelet
    escalas,              ▷ Escalas en las cuales se calcula la OCWT
    lescalas,             ▷ Número máximo de escalas
    dispersión_tiempo     ▷ Vector que contiene las dispersiones
                            en el tiempo para cada una de las escalas
    bitsrecibidos         ▷ Vector en el que se retornan los bits recibidos
)
▷ Calcula los coeficientes wavelet
    c←OCWT( sig, escalas, lescalas )
▷ Concatena la trama de entrada con la trama de entrada anterior
    ctotal[lentrada:2*lentrada] ←c;
```

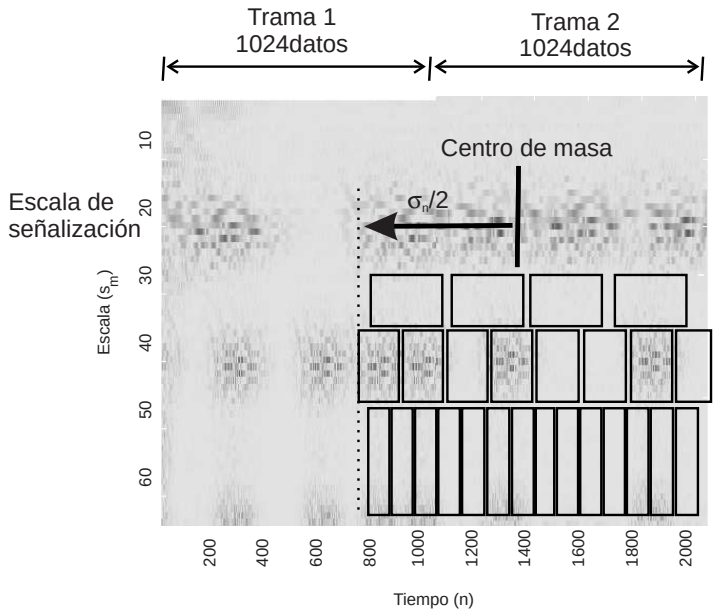


Figura 6.9: Ejemplo de la estrategia de sincronización en el desencriptador pseudocaótico

▷ Busca el máximo en la escala de señalización

iOff ← posición_máximo(ctotal[escala_señalización][0:lentrada])

▷ Busca el centro de masa de la escala de señalización

iOff ← iOff - dispersión_tiempo[escala_señalización]
Si iOff<0

iOff ←0

Si iOff + lentrada < 2*lentrada

iOff ←centro_masa(ctotal[escala_señalización][iOff:iOff+lentrada])
. . .
Recupera los bits a partir de la matriz ctotal[...][iOff:iOff+lentrada]
. . .
▷ Actualiza la matriz de tramas

ctotal[...][0:lentrada]←ctotal[...][lentrada:2*lentrada]

6.2.5 Reasignación dinámica de posiciones

Como se comentó anteriormente, el sistema de encriptación que se propone puede ser usado para la transmisión de información por medio de redes de datos, con la ampliación de la complejidad de la clave por medio de una segunda serie caótica que define las posiciones en las cuales se transmiten los bits en la matriz 2D que entra a la IOCWT.

Esta reasignación dinámica de posiciones se realiza de forma muy simple por medio de un algoritmo que realiza un reordenamiento de los bits que ingresan al algoritmo de cómputo de la señal encriptada. Si las funciones de transmisión y recepción de señales tienen los siguientes formatos:

---

transmisor(

bits_a_transmitir ▷ Bits de información a transmitir
escalas, ▷ Escalas en las cuales se calcula la OCWT
lescalas, ▷ Número máximo de escalas
dispersión_tiempo ▷ Vector que contiene las dispersiones
en el tiempo para cada una de las escalas
sig, ▷ Señal de salida

)
receptor(

sig, ▷ Señal de entrada
lentrada, ▷ Longitud en el tiempo de los coeficientes wavelet
escalas, ▷ Escalas en las cuales se calcula la OCWT
lescalas, ▷ Número máximo de escalas
dispersión_tiempo ▷ Vector que contiene las dispersiones
en el tiempo para cada una de las escalas
bitsrecibidos ▷ Vector en el que se retornan los bits recibidos

)

---

para incluir la reasignación dinámica de posiciones solamente se deben adicionar las siguientes líneas antes y después de invocar al transmisor y receptor, respectivamente:

---

reasignar_bits(bits_a_transmitirin, bits_a_transmitirout)
transmisor(bits_a_transmitirout, escalas, lescalas, dispersión_tiempo, sig)
empaquetar_y_transmitir(sig)
desempaquetar_y_recibir(sig)
receptor(sig, lentrada, escalas, lescalas, dispersión_tiempo, bitsrecibidosin)
reasignar_bits(bits_recibidosin, bits_recibidosout)

Bajo este principio, el algoritmo de reasignación dinámica de posiciones consiste en la copia de los elementos del arreglo de entrada al de salida, usando como índice de salida, un valor estimado a partir de una secuencia pseudoaleatoria calculada por medio de una serie caótica. En las pruebas realizadas se empleó el un mapa logístico de Matthews, reportado en la Ref. [SY00], que genera valores entre $[0, 1]$:

$$x[k+1] = (\beta+1)(1+\frac{1}{\beta})^{\beta}x[k](1-x[k])^{\beta}$$

Estos valores son escalados por un factor de 1000 y posteriormente se calcula el residuo resultante de dividir dicho producto entre el número de bits por trama. Por lo anteriormente expuesto, el algoritmo de reasignación de bits en notación de pseudocódigo es como sigue:

---

reasignar_bits(bitsin, bitsout)
▷ Índice secuencial del bit
ipos←0
▷ inicializa las posiciones
Para i ←0 hasta Nbits-1
    bitpos[i] ← -1
▷ realiza la asignación de posiciones
Para i ←0 hasta NX-1
    xn←**sistema_dinámico**(xn)
    rpos←(xn*1000) mod Nbits
    Si bitpos[rpos] = -1
        bitpos[rpos] ←ipos ▷ Actualiza el listado de asignación de bits
        bitsout[rpos] ←bitsin[ipos] ▷ Intercambia los bits
        ipos←ipos+1
▷ Garantiza que todos las posiciones de bits estén inicializadas
SI ipos ≠pNbits
    Para i ←0 hasta Nbits-1
        Si bitpos[i] = -1
            bitsout[i] ← bitsin[ipos];▷ Intercambia los bits
            ipos←ipos+1

6.2.6 Costo computacional

Aparte de las optimizaciones de la OCWT e IOCWT que se indicaron en la sección 3.4, es posible reducir el número de cálculos considerando únicamente en el transmisor caótico las escalas más relevantes, ya que la entrada a la IOCWT, se trata de un patrón

de datos de entrada binarios, que permite reducir el número de términos involucrados de la sumatoria de la ec. (3.22).

Aunque en el receptor se puede usar una estrategia similar, el número de escalas que involucra la OCWT del receptor siempre será superior, pues para garantizar una detección más confiable del bit es necesario otras escalas en las cuales se dispersa el bit.

Por esta razón, la tasa máxima de transmisión de bits del sistema en tiempo real, depende únicamente de lo optimizado del algoritmo de cálculo de la OCWT.

En este trabajo se encontró que con el valor del factor de escala a_0 empleado, el número de escalas necesarias para la identificación correcta del bit son ± 1, es decir, si el bit se transmite en la escala $m = 30$, es necesario analizar en el receptor, los coeficientes de las escalas $m = 29$, 30 y 31. De esta forma, si m_T es el número de escalas que se emplean para la transmisión de los bits en la IOCWT, el receptor requerirá calcular y analizar $3m_T + 1$ escalas, con 1 que corresponde al cálculo de la escala de la señalización.

6.3 Resultados y Pruebas de Desempeño

Se realizaron diferentes pruebas de desempeño al sistema de encriptación, que consistieron en el análisis de la razón de bit errados (*BER: Bit Error Rate*) ante diferentes condiciones del medio de transmisión tales como, ruido aditivo y de-sincronización del sistema, y la confiabilidad del sistema, empleando para la des-encriptación funciones wavelets estándares y pseudocaóticas con diferentes condiciones iniciales y atractores. Los valores cuantitativos fueron obtenidos por medio de simulaciones del sistema en Matlab, resultados que fueron apreciados cualitativamente por medio de la implementación en tiempo real del sistema sobre un DSP TMS320C6701.

6.3.1 Efecto del medio de transmisión

Para la sincronización, se realizaron diferentes simulaciones que consistieron en desplazar la señal encriptada un número entero y no entero de muestras de la señal entregada por la IOCWT, con el fin de identificar la viabilidad de implementación del sistema en tiempo real, pues en un escenario práctico, el ADC del sistema transmisor y el DAC del receptor no se encuentran perfectamente sincronizados. Dichos resultados fueron verificados posteriormente con la ejecución del sistema sobre el DSP, mostrando que, bajo condiciones de una relación señal a ruido aceptable, el sistema es capaz de sincronizarse perfectamente y determinar en forma correcta los bits.

Respecto a la inmunidad al ruido se encontró que el sistema de encriptación permite una recuperación de la información para relaciones señal a ruido (SNR) superiores a $8dB$. Estos resultados se muestran gráficamente en la Figura 6.10. Al compararlos con los valores reportados por Teolis[Teo98], quien emplea como wavelet madre la función de Morlet, se encuentra que el sistema basado en wavelets pseudocaóticas es más sensible al ruido, pues al usar la función base de Morlet se encuentra una recuperación perfecta para SNR superiores a $2dB$, y a una relación señal a ruido igual a $0dB$ el BER del

sistema es de 0.1% frente al 14% del sistema propuesto. Esto se debe a que la forma altamente regular de la wavelet Morlet favorece el proceso de recuperación de bits más fácilmente que con la función madre pseudocaótica que se propone en este trabajo.

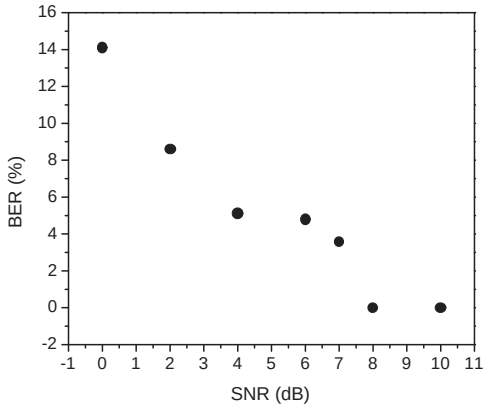


Figura 6.10: Dependencia de la BER para diferentes SNR

Para identificar el conjunto de escalas que ofrecen una mayor inmunidad al ruido, se realizaron una serie de simulaciones que consistieron en el cálculo de la BER por cada una de las escalas ante diferentes relaciones señal a ruido. En la Figura se presentan los resultados para dos escalas. En términos generales se encontró que para una SNR de hasta $4dB$, las escalas menores tienen un comportamiento superior a las escalas superiores, sin embargo, para SNR inferiores, el porcentaje de error presentado en las escalas bajas aumenta considerablemente. Así mismo, se encontró que los bits transmitidos en las escalas altas ($m \geq 40$) presentan una alta dispersión e interferencia sobre las escalas más bajas.

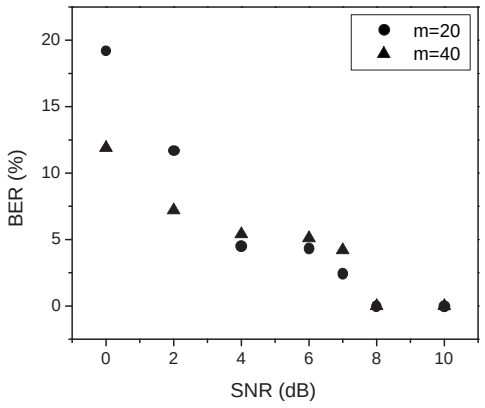


Figura 6.11: Dependencia de la BER para dos escalas diferentes respecto a la distintas SNR

6.3.2 Confiabilidad del sistema

Para verificar la seguridad del sistema se propuso desencriptar la señal con: una wavelet pseudocaótica generada con el sistema caótico de la ec. (**??**) y con diferentes condiciones iniciales, una wavelet pseudocaótica logística, una wavelet de Morlet [Teo98] y una wavelet generada con el esquema presentado en este trabajo pero al que se introduce un vector aleatorio con distribución normal en lugar de una serie caótica. Esta última prueba permite mostrar que las características espectrales de la wavelet madre no dependen del preprocesamiento que se realiza a la función base por medio del interpolado, filtrado y enventanado de la secuencia caótica.

La wavelet madre logística se origina en una serie caótica generada a partir de la ecuación en diferencias dada por la ec. (6.7), conocida como mapa logístico:

$$x[k] \quad = \quad 3.89 * x[k-1] * (1 - x[k-1]) \tag{6.7}$$

En la Tabla 6.1 se presentan las diferentes BER obtenidas a condiciones favorables del sistema (SNR = $100dB$) y con la presencia de ruido (SNR=$10dB$). En todos los casos, la wavelet madre empleada en el sistema transmisor es una serie caótica con condición inicial 0.5. Nótese que aún con la misma wavelet pseudocaótica, para valores de la condición inicial ligeramente alejados, no se logra desencriptar correctamente la señal. Comparativamente con los resultados de Teolis (Ref. [Teo98]) se encuentra que el sistema propuesto es más seguro contra ataques, pues para todos los casos, la BER se mantiene en valores cercanos la 50%, mientras que los valores reportados para el sistema de encriptación con wavelets de Morlet, se consiguen BER que oscilan entre 2% y 50%.

En la Figura 6.12 se presenta la señal transmitida y el espectrograma de los coeficientes wavelet entregados por la OCWT del receptor, cuando éste emplea la misma función madre pseudocaótica del transmisor con igual condición inicial. En esta figura se pueden identificar perfectamente las regiones donde se localizan los bits transmitidos.

En la Figura 6.13 se presentan las matrices de coeficientes en el receptor cuando la condición inicial en el transmisor es 0.5 y en el receptor es 0.1, 0.499 y 0,51, respectivamente. Puede apreciarse que ante un cambio en la condición inicial del sistema caótico que genera la wavelet madre en el receptor, el conjunto de coeficientes generados se dispersarán por toda la matriz o se desplazarán de escala.

Por su parte, la Figura 6.14 corresponde a los espectrogramas de los coeficientes wavelet cuando se intenta la desencriptación de la información con las wavelet de Morlet, aleatoria y logística. Se encuentra que, al igual que con la variación de la condición inicial, no es posible identificar con precisión las regiones de dispersión en las cuales se encuentran localizados los bits.

Para estudiar más a fondo la confiabilidad del sistema, es necesario analizar los tres escenarios prácticos de ataque que considera la criptología convencional[Sch96]: el ataque a partir del *texto cifrado exclusivamente*, el *ataque con el conocimiento del texto plano*, y el *ataque con la selección del texto plano*.

En el ataque a partir del texto cifrado exclusivamente, el intruso conoce únicamente la señal encriptada y tiene poca o ninguna información respecto de la fuente. Bajo este

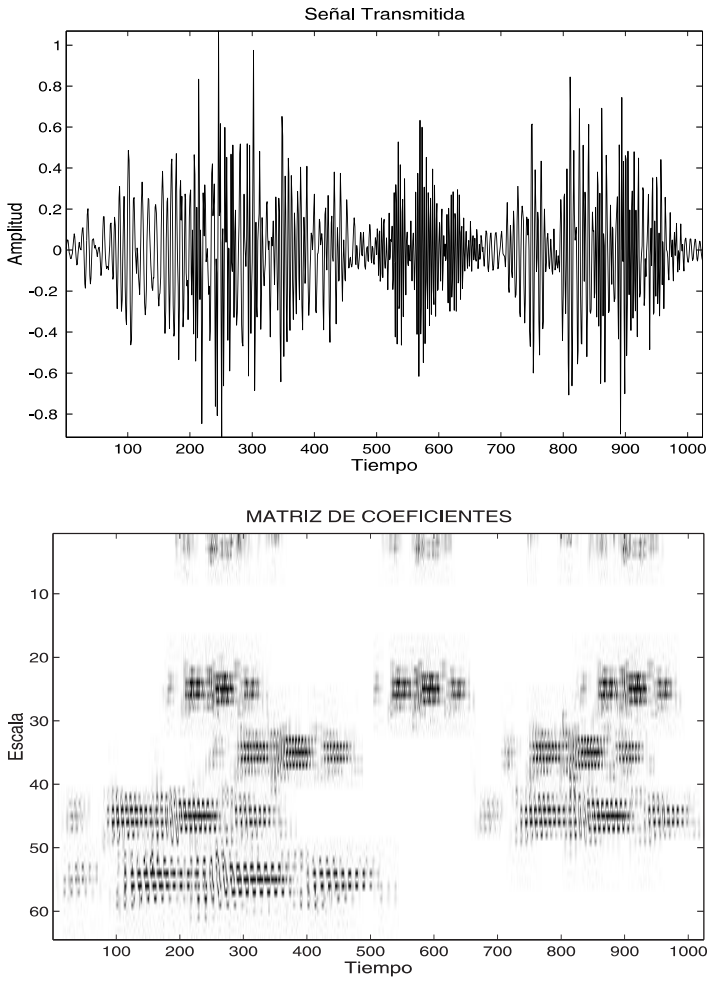


Figura 6.12: Señal transmitida y espectrograma de los coeficientes wavelet para la misma función madre pseudocaótica del transmisor

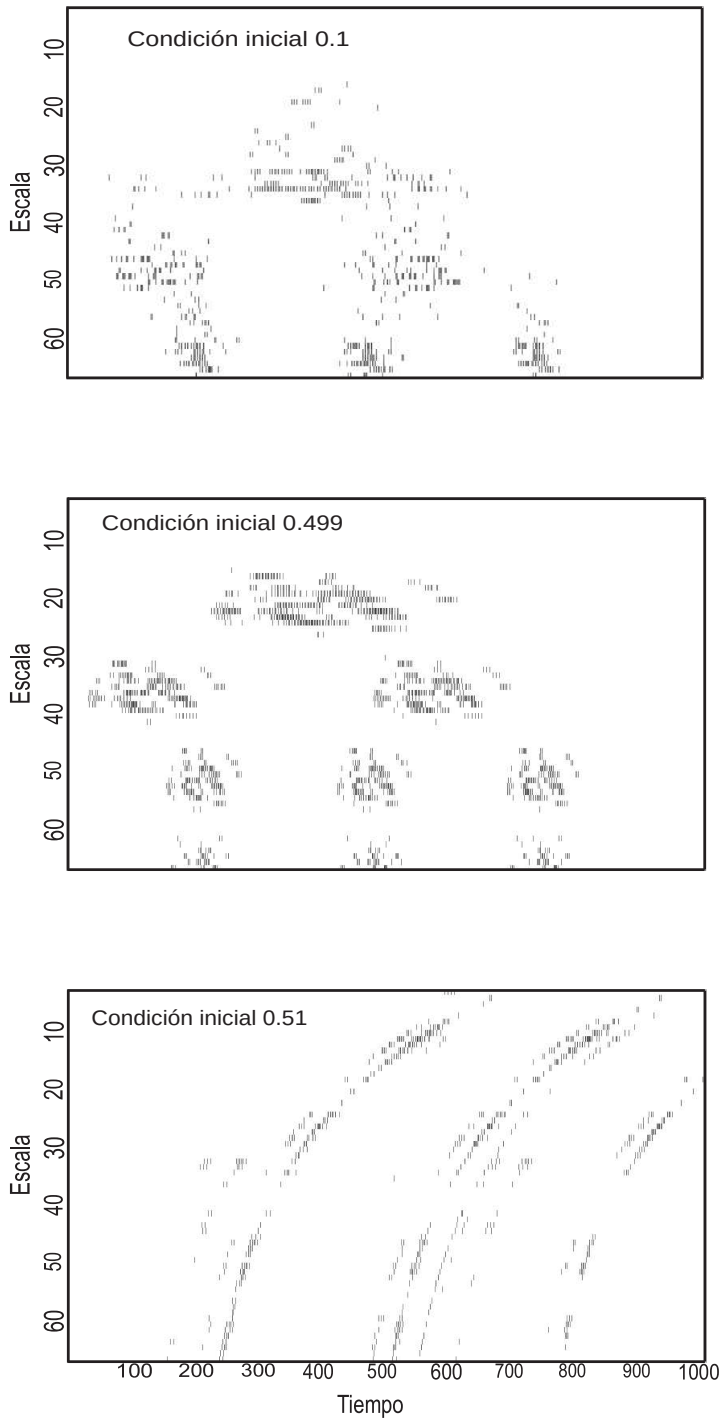


Figura 6.13: Espectrogramas de los coeficientes wavelet para distintas condiciones iniciales de la función madre pseudocaótica

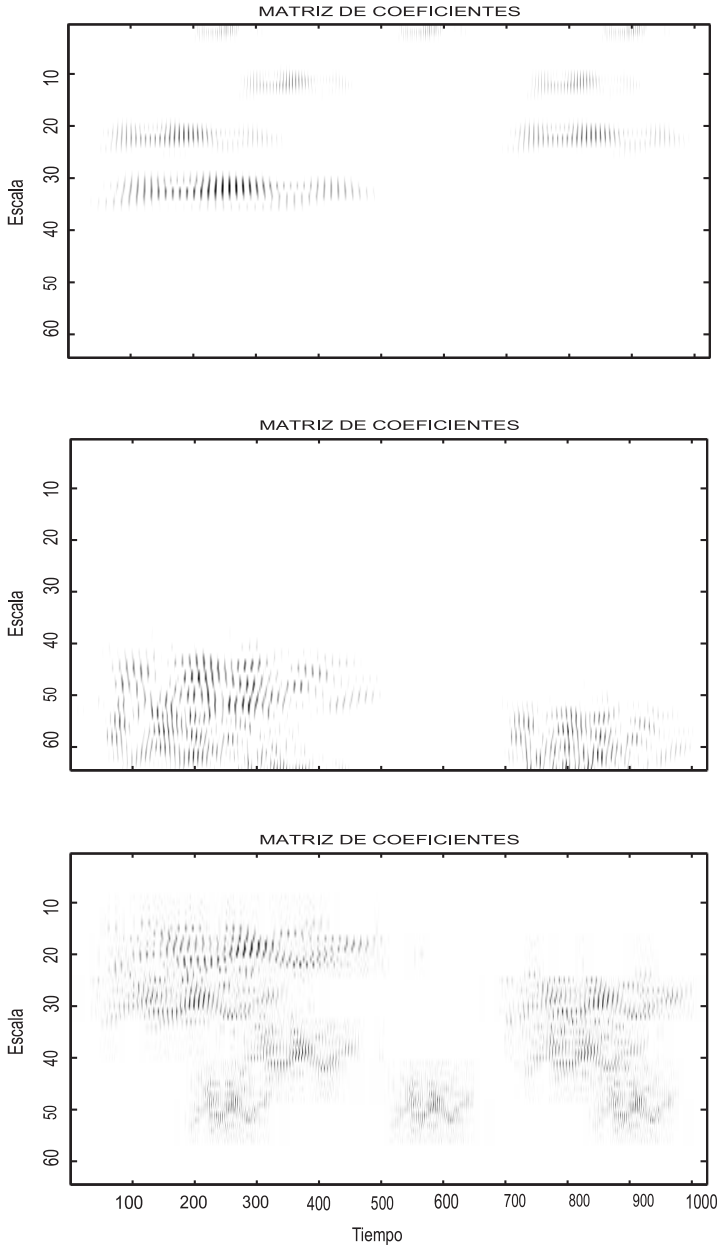


Figura 6.14: Espectrograma de los coeficientes wavelet para diferentes funciones madre: Morlet, aleatoria y logística.

Tabla 6.1:

Wavelet madre	Condición inicial	BER (%) SNR=10dB	BER (%) SNR=100dB
Original	0.75	37	42
	0.55	44.4	48.2
	0.51	43.5	43.1
	0.49	34.6	20
	0.50	0	0
	0.45	51.8	51.3
	0.25	39.7	40.2
Logística	0.55	49.8	42.1
	0.51	50.5	49.6
	0.5	48.8	50.3
	0.49	48.6	49.7
	0.45	49.8	48.8
Morlet		48.1	47.4
Aleatoria		47.5	45.4

tipo de ataque, el sistema aquí propuesto es casi inquebrantable, pues para recuperar la información binaria se hace necesario conocer la clave en su totalidad, conformada por: la serie caótica $\psi_0[n, \psi_0(0)]$, la condición inicial $\psi_0(0)$, los parámetros de preprocesamiento $\mathbf{w}_0$, y la posición escala-tiempo en la cual se ubican los bits de entrada a la IOCWT $(\mathbf{s}, \tau)$. Aunque el bit de sincronización sea fácilmente reconocido, pues se trata de una señal periódica que se envía por cada trama transmitida, la recuperación y determinación de las regiones de cada uno de los bits es casi imposible, pues como se mostró en las Figuras 6.13 y 6.14, un cambio en la condición inicial y tipo de sistema dinámico altera considerablemente los coeficientes wavelet. Por otra parte, para la identificación del sistema dinámico caótico, los parámetros de preprocesamiento y el conjunto de escalas, se haría necesario el almacenamiento de una cantidad apreciable de espectrogramas de los coeficientes wavelet, lo cual implica algoritmos altamente complejos desde el punto de vista del almacenamiento, y el cálculo de un número elevado de instrucciones MAC. De esta forma, la complejidad computacional del proceso de identificación de la clave, es una tarea altamente compleja para este tipo de ataque.

Respecto al ataque con el conocimiento del texto plano, en el que el intruso tiene un conocimiento mínimo de la codificación de ciertos fragmentos de la fuente; el sistema propuesto sigue teniendo una alta invulnerabilidad, pues en este caso, aunque sea posible emplear técnicas de filtrado adaptativo o redes neuronales para la identificación del sistema dinámico caótico y/o las regiones de dispersión de cada bit, el costo de dichos algoritmos es bastante elevado, pues la OCWT involucra un número muy alto de instrucciones MAC. Así mismo, la identificación de las regiones de dispersión de cada

bit no garantiza la detección completa de la clave, pues se hace necesario identificar la asociación entre cada bit transmitido y las su respectiva región de dispersión. Para este aspecto, se tienen $nbit!$ combinaciones posibles, con $nbit$ el número de bits transmitidos por trama.

Finalmente, la vulnerabilidad más crítica se presenta con el escenario de ataque con la selección del texto plano. En este esquema el intruso tiene un conocimiento mínimo del sistema de encriptación y tiene la posibilidad de aplicar señales al sistema de encriptación. Es factible con el sistema propuesto, detectar la clave o proponer estrategias alternativas para la decodificación de la información, pues aunque resulta altamente costoso identificar el sistema dinámico, la condición inicial y las condiciones de preprocesamiento, el reconocimiento de la posición escala-tiempo de cada uno de los bits transmitidos es bastante simple.

Por consiguiente, el sistema de encriptación propuesto debe ser empleado únicamente en escenarios en los cuales el intruso solamente pueda tener conocimiento de la señal encriptada y en casos excepcionales, de ciertas asociaciones entre la información fuente y la señal encriptada.

6.3.3 Implementación sobre el DSP

Las primeras pruebas sobre el DSP no permitieron efectuar la transmisión en tiempo real, pues como se indicó en la sección 3.4, el algoritmo de cómputo de la OCWT e IOCWT involucra un número elevado de instrucciones MAC, y por cada escala, en la versión más optimizada, son necesarios cerca de 23 milisegundos.

Debido a que en el receptor se deben calcular tres veces más escalas que en el transmisor (ver sección 6.2.6), el número máximo de escalas que se emplearon en la implementación final, fue de dos. Con este número de escalas, al transmisor le toma cerca de 96ms generar un fragmento de 1024 elementos, y al receptor, 192ms en procesarlo (OCWT+sincronización+detección), lo cual implica una frecuencia de muestreo máxima para la ejecución en tiempo real de $1024/196ms \approx 5.33KHz$. Como en la tarjeta experimental EVM6701 la frecuencia mínima de muestreo configurable en sus codecs es de $8KHz$, se hizo necesario para la ejecución en tiempo real la adición de un interpolador a la salida de la IOCWT que permitiera aumentar el número de muestras por segundo, y de un diezmador a la entrada de la OCWT. Para las pruebas preliminares, el factor de interpolación y diezmado escogido fue de 8, sin embargo, con optimizaciones realizadas al algoritmos se pudo descender hasta 2. De esta forma, la tasa de bits por segundo que se pueden transmitir con el sistema de encriptación es bastante baja:

$$N_{bits \times trama} \times \frac{fs}{N_{trama} \times I_{factor}} \tag{6.8}$$

pues, $N_{bits \times trama} = 12$, $N_{trama} = 1024$ e $I_{factor} = 2$, conduciéndonos a 46.8 bits por segundo. Esta cantidad solamente puede ser mejorada con el empleo de otro tipo de procesador o el cómputo en paralelo de la OCWT e IOCWT en sistemas multiprocesador. Aspectos que se dejan como propuestas de trabajos futuros. Las pruebas en

tiempo real de los algoritmos se realizaron empleando la transmisión en banda base de la señal encriptada, y como canal de comunicación se usó una transmisión por cable, encontrándose que el mensaje puede recuperarse para valores de la amplitud de la señal encriptada del cerca del 2% de la excursión máxima. Así mismo, para valores de amplitud menores a 10% de la excursión máxima, se hace necesario modificar el umbral de supresión de ruido de la etapa de *wavelet denoising*. Se propone como trabajo futuro el diseño de una técnica de control automático del nivel de umbral de detección.

6.4 Conclusiones

Se mostró que es posible usar una serie caótica, sometida a varios procesos de filtrado y enventanamiento, como función wavelet madre, para la construcción de un sistema de encriptación de información.

Se propusieron dos esquemas de aplicación según la forma de transmisión de la señal encriptada: un sistema orientado a señales análogas y otro orientado a paquetes de datos. Para el sistema orientado a la transmisión de señales análogas se propuso un sistema de sincronización basado en la transmisión de un bit de señalización por cada una de las tramas de datos.

La seguridad del sistema fue probada ante diferentes funciones wavelet madre pseudocaóticas y la función wavelet de Morlet, y con distintas relaciones señal a ruido (SNR). Se encontró que no es posible desencriptar la información por medio del uso de otra función madre, aún tratándose del mismo sistema caótico pero diferente condición inicial. Se mostró también que el valor mínimo de SNR bajo el cual el sistema trabaja en forma óptima es $8dB$.

Como resultado del análisis de vulnerabilidad se determinó que el sistema propuesto es resistente a ataques basados en los escenarios de texto cifrado exclusivamente y ataque con el conocimiento del texto plano, pero no ante un ataque con el esquema de la selección del texto plano.

Capítulo 7

Conclusiones

Se propusieron en este trabajo tres diferentes sistemas para la compresión, multiplexación y encriptación de señales. A excepción del sistema de multiplexación, los restantes fueron versiones completamente funcionales en tiempo real, nuestro principal limitante fue la frecuencia de reloj del procesador con que se cuente, sin embargo, los algoritmos y técnicas para WPDM pueden ser migradas fácilmente a DSPs de nueva generación Aunque estos tres sistemas fueron desarrollados en forma separada, pueden ser combinados en conjunto, por ejemplo, para aumentar la tasa de transmisión en el sistema de encriptación se puede emplear el sistema de compresión, dado a éste último no demanda demasiados recursos de cómputo.

Se mostró también que, introduciendo pequeñas variantes a los sistemas de compresión, multiplexación y encriptación, es posible conseguir mejoras significativas. Por ejemplo, para el sistema compresor de señales, el empleo de LPCs para la codificación de los coeficientes de ciertos nodos del árbol mejora la tasa de compresión y la razón de transferencia de bits con respecto a otras técnicas de compresión basadas en wavelets, así mismo, sólo para ciertos hablantes, el empleo de un árbol dinámico mejora la tasa de compresión frente a un sistema de árbol estático. En el sistema de multiplexación, se mostró la factibilidad de emplear señales multinivel, sin embargo, es necesario incluir una etapa de compensación de fase, para la cual se propuso un método de sincronización, y sustituyendo el modulador wavelet por uno PAM basado en la función coseno elevado, se consigue una mayor inmunidad al ruido. Y por último, en un esquema simple de encriptación como lo es el basado en la transformada wavelet sobre-completa, es posible aumentar la seguridad del sistema empleando una función madre caótica a costa de la disminución en la velocidad de transmisión.

.

Bibliografía

[AM99] A. N. Akansu and M. J. Medley. *Wavelet, Subband and Block Transforms in Communications and Multimedia*. Kluwer Academic Publishers, Boston, 1999.

[AMRP00a] G. Alvarez, F. Montoya, M. Romera, and G. Pastor. Criptoanálisis del sistema criptográfico basado en la sincronización de osciladores caóticos. *Mundo Electrónico*, 307:56–58, 2000.

[AMRP00b] G. Alvarez, F. Montoya, M. Romera, and G. Pastor. Cryptanalysis of a chaotic encryption system. *Physics Letters A*, 276:191–196, 2000.

[ATJ04] A.B. Aicha, F. Tlili, and S.B. Jebara. Papr analysis and reduction in wpdm systems. *Proc. of ICASSP*, pages 315–318, 2004.

[CF04] K.S. Chan and F. Fekri. A block cipher cryptosystem using wavelet transforms over finite fields. *IEEE Trans. on Signal Processing*, 52:2975–2991, 2004.

[CFW01] C.C. Chen, K.C. Fan, and S.W. Wang. A secure and robust digital watermarking technique by the block cipher rc6 and secure hash algorithm. *Proc. of International Conference on Image Processing*, 2:518–521, 2001.

[COS93] K. M. Cuomo, A. V. Oppenheim, and S. H. Strogatz. Synchronization of lorenz-based chaotic circuits with applications to communications. *IEEE Trans. Circuits Syst.*, 40:626–633, 1993.

[Cui03] L. Cui. The application of wavelet analysis and audio compression technology in digital audio watermarking. *Proc. of IEEE Int. Conf. on Neural Networks and Signal Processing*, 2:1533–1537, 2003.

[CW92] R.R. Coifman and M.L. Wickerhauser. Entropy based algorithms for best-basis selection. *IEEE Transactions on Information Theory*, 38:713–718, 1992.

[DHP00] John R. Deller, John H. L. Hansen, and John G. Proakis. *Discrete–Time Processing of Speech Signals*. IEEE Press, 2000.

[DO97] H. Dedieu and M. J. Ogorzalek. Identifiability and identification of chaotic systems based on adaptative synchronization. *IEEE Trans. Circuit Syst. I*, 44:948–962, 1997.

[DSW98] T.N. Davidson, A.J. Schott, and K.M. Wong. Branch–hopped wavelet packet division multiplexing. *Proc. of ICASSP*, 4:3233–3236, 1998.

[Emb99] Paul M. Emberee. *C++ Algorithms for Digital Signal Processing*. Pretince Hall, 1999.

[Fli94] N. J. Fliege. *Multirate Digital Signal Processing*. John Wiley and Sons, Chichester, 1994.

[GK94] Z. Gah and S. N. Kah. Speech coding by wavelet representation of residual signal. *Proc. of ICCS*, 2:860–864, 1994.

[GKMM89] A. Grossmann, R. Kronland-Martinet, and J. Morlet. *Reading and understanding continous wavelet transforms*, pages 2–20. Springer-Verlag, Berlin, 1989.

[GLOB95] H. Guo, M. Lang, J.E. Odegard, and C.S. Burrus. Nonlinear shrinkage of undecimated dwt for noise reduction and data compression. *Proc. of Int. Conf. on Digital Signal Processing*, 1995.

[GM99] G. Grassi and S. Mascoio. A system theory approach for designing cryptosystems based on hyperchaos. *IEEE Tran. on Circuits and Systems I: Fundamental Theory and Aplications*, 46:1135–1138, 1999.

[HERES96] N.M. Hosny, S.H. El-Ramly, and M.H. El-Said. Novel techniques for speech compression using wavelet transform. *Proc. of 11th Int. Con. on microelectronics*, pages 225–229, 1996.

[Het] Kenneth Hetling. A pr-qmf (wavelet) based spread spectrum communications system. *URL:citeseer.ist.psu.edu/172163.html*.

[HSD95] K. Hetling, G. Saulnier, and P. Das. Optimized filter design for pr-qmf based spread spectrum communications. *Proc. of the IEEE International Conference on Communications*, pages 1350–1354, 1995.

[HW03] Q. Huo and Y. Wang. Security traffic image transmission based on ezw and aes. *Proc. of the IEEE Intelligent Transportation Systems*, 1:86–89, 2003.

[Ins00] Texas Instruments. *TMS320C6000 Optimizing C/C++ Compiler User's Guide*. January 2000.

[Jim02] Javier Alejandro Bustos Jiménez. Estudio de sistemas de compresión de voz digital orientado a telefonía celular. *URL:http://www.inf.udec.cl/revista/edición7/jbustos.htm*, 2002.

[LCL02] R.C. Luo, L.Y. Chung, and C.H. Lien. A novel symmetric cryptography based on the hybrid haar wavelets encoder and chaotic masking scheme. *IEEE Transactions on Industrial Electronics*, 49:933–944, 2002.

[LF94] Paul A. Lynn and Wolfgang Fuerst. *Digital Signal Processing with Computer Applications*. John Wiley and Sons, 1994.

[Lin97] A.R. Lindsey. Wavelet packet modulation for orthogonally multiplexed communication. *IEEE Trans. on Signal Processing*, 45:1336–1339, 1997.

[MHAG+03a] J.I. Marín-Hurtado, R. Arango, J.E. Gutiérrez, A. López-Parrado, and D.F. González. *Reconocimiento de voz en tiempo real por medio de la transformada wavelet y procesadores digitales de señales*. Universidad del Quindío, Armenia, 2003.

[MHAG+03b] J.I. Marín-Hurtado, R. Arango, J.E. Gutiérrez, A. López-Parrado, and D.F. González. Sistema para el reconocimiento de fonemas en tiempo real usando la transformada wavelet discreta y un procesador digital de señales. *Revista de investigaciones de la Universidad del Quindío*, 12:42–46, 2003.

[MHGQO04] J.I. Marín-Hurtado, A. García-Quinchía, and O.H. Ocampo. Compresión de señales de voz en tiempo real usando paquetes wavelet. *Memorias IX Simposio de Tratamiento de Señales, Imágenes y Visión Artificial*, 2004.

[NRPS03] A.M Najih, A. R. Ramli, V. Prakash, and A.R. Syed. Speech compression using discret wavelet transform. *Proc. of 4th Nat. Conference on Telecommunication Technology*, pages 1–4, 2003.

[PC90] L. M. Pecora and T. L. Caroll. Synchronization in chaotic systems. *Phys. Rev. Letters*, 64:821, 1990.

[PCK+92] U. Parlitz, L.O. Chua, L.J. Kocarev, K.S. Halle, and A. Shang. Transmission of digital signals by chaotic synchronization. *Inter. J. of Bifurcation and Chaos*, 2:973–977, 1992.

[PM92] J. G. Proakis and D.G. Manolakis. *Tratamiento Digital de Señales*. Pretince–Hall, 1992.

[PS94] J. G. Proakis and M. Salehi. *Communication Systems Engineering*. Pretince Hall Int., 1994.

[PTVF92] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C*. Cambridge Unversity Press, Cambridge, 1992.

[RB98] Raghuveer M. Rao and Ajit S. Bopardikar. *Wavelet Transforms*. Addison Wesley, 1998.

[RY01] K.R Rao and P.C. YIP. *The Transform and Data Compression Handbook*. CRC Press LLC, Boca Raton, 2001.

[SAA98] I. Singh, P. Agathoklis, and A. Antoniou. Wavelet–based compression of speech signals on the tms320c30 digital signal processor. *Proc. of IEEE Sym. on advanced in digital signal filtering and signal processing*, pages 178–182, 1998.

[Sal01] David Salomon. *Data Compression Methods*. Springer Verlag, New York, 2001.

[Sch96] B. Schncier. *Applied Cryptography*. Wiley, New York, 1996.

[Sil96] C.P. Silva. A survey of chaos and its applications. *IEEE MTT-S Digest*, 49:1871–1874, 1996.

[SN97] Gilber Strang and Troung Nguyen. *Wavelets and Filter Banks*. Wellesley–Cambridge Press, Wellesley, 1997.

[Str89] F. G. Stremler. *Sistemas de Comunicación*. Alfaomega, 1989.

[SY00] C.P. Silva and A.M. Young. Introduction to chaos-based communications and signal processing. *IEEE Trans. Circuits Syst.*, 49:279–299, 2000.

[Teo98] Anthony Teolis. *Computational Signal Processing with Wavelets*. Birkhauser, Boston, 1998.

[Tex00] Texas Instruments. *TMS320C6000 CPU And Instruction Set Reference Guide*. January 2000.

[TWW00] K.F. To, K.T. Wong, and K.M. Wong. Analysis of amplifier nonlinearities on wavelet packet division multiplexing. *Proc. of ICASSP*, 5:2813–2816, 2000.

[Wic94] Mladen Victor Wickerhauser. *Adapted Wavelet Analysis From Theory to Software*. A K Peters, Massachusetts, 1994.

[Won98] K.M. Wong. Design of branch–hopped wavelet packet division multiplexing schemes. *Proc. of ICSP*, pages 257–262, 1998.

[Wu98] Jiangfeng Wu. *Wavelet Packet Division Multiplexing*. Phd thesis, McMaster University, Hamilton, Ontario, 1998.

[WWD+00] K.M. Wong, J. Wu, T.N. Davidson, Q. Jin, and P.C. Ching. Peformance of wavelet packet–division multiplexing in impulsive and gaussian noise. *IEEE Trans. on Communications*, 48:1083–1086, 2000.

[WWDJ97] K.M. Wong, J. Wu, T.N. Davidson, and W. Jin. Wavelet packet division multiplexing and wavelet packet design under timing error effects. *IEEE Trans. on Signal Processing*, 45:2877–2890, 1997.

[XJZW02] W. Xiao, Z. Ji, J. Zhang, and W. Wu. A watermarking algorithm based on chaotic encryption. *Proc. of IEEE TENCON*, pages 545–548, 2002.

[ZFL02] Hongbing Zhang, Howard Fan, and Alan Lindsey. Receiver design for wavelet based multicarrier cdma communications. 2002.

Apéndice A

Programas en C para el cálculo de los Paquetes Wavelet

En este apéndice se presenta el código de las rutinas de

A.1 Función WaveStageD

A.1.1 Versión sin optimizar

```
void Wave_StageD(
    float *entrada, // Coeficientes de la entrada
    const int lentrada,   // Longitud del bloque entrada
    const float *h0,      // Coeficientes filtro pasa-bajo (análisis)
    const float *h1,      // Coeficientes filtro pasa-alto (análisis)
    const int lh,         // Longitud respuesta al impulso
    float *fi0,     // Coeficientes salida filtro pasa-bajo
    float *fi1      // Coeficientes saliada filtro pasa-alto
) {
    int i,j,ind=0;
    float aux0,aux1;
    _nassert(((int)(entrada) & 0x3) == 0);
    _nassert(((int)(h0) & 0x3) == 0);
    _nassert(((int)(h1) & 0x3) == 0);
/*Convolución y subsecuente submuestreo de manera eficiente
usando filtros polifásicos con un conmutador a la entrada*/
    for (i=0; i< lentrada; i+=2)  {
        aux0 = 0.0;
        aux1 = 0.0;
        #pragma MUST_ITERATE(8,,2)
```

```
            for (j=0; j<lh; j++)  {
                aux0 += entrada[i+j] * h0[j];
                aux1 += entrada[i+j] * h1[j];
            }
            fi0[ind] = aux0;
            fi1[ind] = aux1;
            ind++;
        }
    /*Actualizacón de las condiciones iniciales. Se
        copian las últimas N-1 muestras.*/
        for (i=0; i<lh; i++)  {
            entrada[i]=entrada[lentrada+i];
        }
}
```

A.1.2 Versión optimizada

Esta versión hace uso del alineamiento a 32bits para el vector de la señal de entrada y los coeficientes de las respuestas al impulso, así mismo de las estrategias de desenrollado automático y manual, este último por medio de la captación de operandos (entrada y coeficientes) a 64bits. Esta versión solamente funciona si la longitud de la respuesta al impulso es divisible por cuatro. Así mismo, para el correcto funcionamiento de la rutina, los coeficientes de la respuesta al impulso deben pasarse en forma reflejada.

```
#define LOFLOAT(x)          _itof(_lo(x))
#define HIFLOAT(x)          _itof(_hi(x))
void Wave_StageD(
    float *entrada, // Coeficientes de la entrada
    const int lentrada,   // Longitud del bloque entrada
    const float *h0,      // Coeficientes filtro pasa-bajo (análisis)
    const float *h1,      // Coeficientes filtro pasa-alto (análisis)
    const int lh,         // Longitud respuesta al impulso
    float *fi0,     // Coeficientes salida filtro pasa-bajo
    float *fi1      // Coeficientes salida filtro pasa-alto
) {
    int i,j,ind=0;
    float aux0,aux1;
    double *h0d, *h1d, *entradad;
    _nassert(((int)(entrada) & 0x3) == 0);
    _nassert(((int)(h0) & 0x3) == 0);
```

```
        _nassert(((int)(h1) & 0x3) == 0);
        h0d = (double*) LOR; //OJO
        h1d = (double*) HIR;
        entradad = (double*) entrada;

    /*Convolución y subsecuente submuestreo de manera eficiente
      usando filtros polifásicos con un conmutador a la entrada*/

        for (i=0; i<lentrada/2; i++)  {
            aux0 = 0.0;
            aux1 = 0.0;
            #pragma MUST_ITERATE(4)
            for (j=0; j<lh/2; j++) {

                aux0 += LOFLOAT(entradad[i+j]) * LOFLOAT(h0d[j]);
                aux0 += HIFLOAT(entradad[i+j]) * HI-
                FLOAT(h0d[j]);
                aux1 += LOFLOAT(entradad[i+j]) * LOFLOAT(h1d[j]);
                aux1 += HIFLOAT(entradad[i+j]) * HI-
                FLOAT(h1d[j]);
            }
            fi0[ind] = aux0;
            fi1[ind] = aux1;
            ind++;
        }
     /*Actualización de las condiciones iniciales. Se
       copian las últimas N-1 muestras.*/
        for (i=0; i<lh; i++)  {
            entrada[i]=entrada[lentrada+i];
        }
    }
```

A.2 Función WaveStageI

A.2.1 Versión sin optimizar

```
void Wave_StageI(
    float *entrada0, // Coeficientes de la primera entrada
    float *entrada1, // Coeficientes de la seguenda entrada
    const int lentrada,    // Longitud de bloque
    const float *g0,       // Coeficientes filtro pasa-bajo (síntesis)
```

```
        const float *g1,       // Coeficientes filtro pasa-alto (síntesis)
        const int lh,          // Longitud respuestsa al impulso
        float *sal       // Coeficientes de salida
) {
    int i,j;
    int ind=0;
    float auxp0, auxp1,auxi0,auxi1;
    _nassert(((int)(entrada0) & 0x3) == 0);
    _nassert(((int)(entrada1) & 0x3) == 0);
    _nassert(((int)(g0) & 0x3) == 0);
    _nassert(((int)(g1) & 0x3) == 0);
    g0 = LOD;    //OJO
    g1 = HID;
    /*Convolución y subsequente sobremuestreo de manera eficiente
    usando filtros polifásicos con un conmutador a la salida*/
        for (i=0; i<lentrada; i++)  {
            auxp0 = 0.0;
            auxp1 = 0.0;
            auxi0 = 0.0;
            auxi1 = 0.0;
            #pragma MUST_ITERATE(4)
            for (j=0; j<(lh/2); j++)  {
                auxi0 += entrada0[i+(lh/2)+j]*g0[2*j+1];
                auxp0 += entrada0[i+(lh/2)+j]*g0[2*j];
                auxi1 += entrada1[i+(lh/2)+j]*g1[2*j+1];
                auxp1 += entrada1[i+(lh/2)+j]*g1[2*j];
            }
            sal[ind++] = auxi0+auxi1;
            sal[ind++] = auxp0+auxp1;
        }
        for (i=0; i<lh; i++) {
            entrada0[i] = entrada0[lentrada+i];
            entrada1[i] = entrada1[lentrada+i];
        }
}
```

A.2.2 Versión optimizada

Esta versión hace uso del alineamiento a 32bits para el vector de la señal de entrada y los coeficientes de las respuestas al impulso, así mismo de las estrategias de desenrollado automático y manual, este último por medio de la captación de operandos (entrada y

coeficientes) a 64bits. Esta versión solamente funciona si la longitud de la respuesta al impulso es divisible por cuatro. Así mismo, para el correcto funcionamiento de la rutina, los coeficientes de la respuesta al impulso deben pasarse en forma reflejada.

```
#define LOFLOAT(x)          _itof(_lo(x))
#define HIFLOAT(x)          _itof(_hi(x))
void Wave_StageI(
    float *entrada0, // Coeficientes de la primera entrada
    float *entrada1, // Coeficientes de la seguenda entrada
    const int lentrada,    // Longitud de bloque
    const float *g0,       // Coeficientes filtro pasa-bajo (síntesis)
    const float *g1,       // Coeficientes filtro pasa-alto (síntesis)
    const int lh,          // Longitud respuestsa al impulso
    float *sal       // Coeficientes de salida
) {
    int i,j;
    int ind=0;
    float aux0, aux1,aux2,aux3;
    double *g0d, *g1d, *entrada0d, *entrada1d;
    double g0coef, g1coef;
    _nassert(((int)(entrada0) & 0x3) == 0);
    _nassert(((int)(entrada1) & 0x3) == 0);
    _nassert(((int)(g0) & 0x3) == 0);
    _nassert(((int)(g1) & 0x3) == 0);
    g0d = (double*) LOD; //OJO!!
    g1d = (double*) HID;
  /*Convolución y subsecuente sobremuestreo de manera eficiente
  usando filtros polifásicos con un conmutador a la salida*/
    for (i=0; i<lentrada; i+=2)  {
        aux0 = 0.0;
        aux1 = 0.0;
        aux2 = 0.0;
        aux3 = 0.0;
        entrada0d = (double*) (entrada0+(lh/2)+i);
        entrada1d = (double*) (entrada1+(lh/2)+i);
        g0coef = g0d[0];
        g1coef = g1d[0];
        aux0 += LOFLOAT(*entrada0d) * HIFLOAT(g0coef)
              + LOFLOAT(*entrada1d) * HIFLOAT(g1coef);
        aux1 += LOFLOAT(*entrada0d) * LOFLOAT(g0coef)
             + LOFLOAT(*entrada1d) * LOFLOAT(g1coef);
```

```
aux2 += HIFLOAT(*entrada0d) * HIFLOAT(g0coef)
      + HIFLOAT(*entrada1d) * HIFLOAT(g1coef);
aux3 += HIFLOAT(*entrada0d) * LOFLOAT(g0coef)
      + HIFLOAT(*entrada1d) * LOFLOAT(g1coef);
g0coef = g0d[1];
g1coef = g1d[1];
aux0 += HIFLOAT(*entrada0d) * HIFLOAT(g0coef)
      + HIFLOAT(*entrada1d) * HIFLOAT(g1coef);
aux1 += HIFLOAT(*entrada0d) * LOFLOAT(g0coef)
      + HIFLOAT(*entrada1d) * LOFLOAT(g1coef);
entrada0d++;
entrada1d++;
#pragma MUST_ITERATE(2)
for (j=2; j<(lh/2); j+=2)  {

    aux2 += LOFLOAT(*entrada0d) * HI-
    FLOAT(g0coef)
          + LOFLOAT(*entrada1d) * HI-
    FLOAT(g1coef);
    aux3 += LOFLOAT(*entrada0d) * LOFLOAT(g0coef)
         + LOFLOAT(*entrada1d) * LOFLOAT(g1coef);
    g0coef = g0d[j];
    g1coef = g1d[j];
    aux0 += LOFLOAT(*entrada0d) * HI-
    FLOAT(g0coef)
          + LOFLOAT(*entrada1d) * HI-
    FLOAT(g1coef);
    aux1 += LOFLOAT(*entrada0d) * LOFLOAT(g0coef)
        + LOFLOAT(*entrada1d) * LOFLOAT(g1coef);
    aux2 += HIFLOAT(*entrada0d) * HI-
    FLOAT(g0coef)
        + HIFLOAT(*entrada1d) * HIFLOAT(g1coef);
    aux3 += HI-
    FLOAT(*entrada0d) * LOFLOAT(g0coef)
          + HI-
    FLOAT(*entrada1d) * LOFLOAT(g1coef);
    g0coef = g0d[j+1];
    g1coef = g1d[j+1];
    aux0 += HIFLOAT(*entrada0d) * HI-
    FLOAT(g0coef)
        + HIFLOAT(*entrada1d) * HIFLOAT(g1coef);
    aux1 += HI-
    FLOAT(*entrada0d) * LOFLOAT(g0coef)
```

```
                    + HI-
                FLOAT(*entrada1d) * LOFLOAT(g1coef);
                entrada0d++;
                entrada1d++;
            }
            aux2 += LOFLOAT(*entrada0d) * HIFLOAT(g0coef)
                    + LOFLOAT(*entrada1d) * HIFLOAT(g1coef);
            aux3 += LOFLOAT(*entrada0d) * LOFLOAT(g0coef)
                    + LOFLOAT(*entrada1d) * LOFLOAT(g1coef);
            sal[ind++] = aux0;
            sal[ind++] = aux1;
            sal[ind++] = aux2;
            sal[ind++] = aux3;
        }
        for (i=0; i< lh; i++)     {
            entrada0[i] = entrada0[lentrada+i];
            entrada1[i] = entrada1[lentrada+i];
        }
    }
```

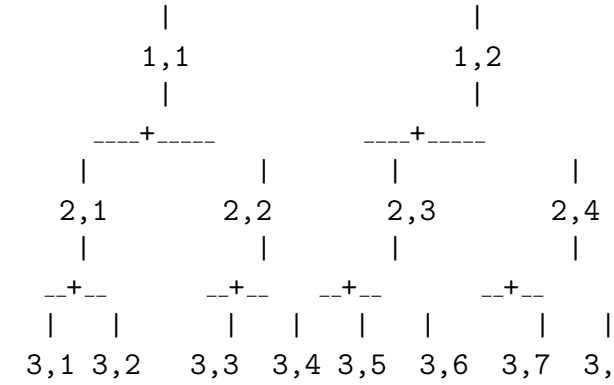
A.3 Funciones PWD e IPW

Para la transformada directa e inversa, los datos de entrada deben estar desplazados $lh - 1$ muestras, con lh la longitud de la respuesta al impulso de los filtros wavelet. La estructura del árbol se crea con la función CreaPaqueteWavelet, cuyo formato es:

```
WPTree* CreaPaqueteWavelet(
    int *iArbol,
    int nNodos,
    float* h0, float* h1,
    float* g0, float* g1,
    int lh
)
```

Para realizar el árbol de descomposición mostrado en la siguiente figura, el arreglo iArbol debe contener los siguientes elementos: `iArbol = [3, 1, 3, 2, 3, 3, 3, 4, 3, 5, 3, 6, 3, 7, 3, 8]` y `nNodos = 8`.

```
                    0,1
                    |
          _________+_________
```




```
void PWD(WPTree* wptree, float *entrada, int lentrada, float **sal) {
    int i, k;
    int imax;
    int n, pr;
    int lh;
    lh = wptree->lh;
    memcpy(wptree->pwd_nodos[0][0]+lh-
    1, entrada, lentrada*sizeof(float));
    /* Calcula el árbol de descomposición */
    for (i=0; i<wptree->nMaxNivel-1; i++)  {
        imax = lentrada >> i;                          //longi-
        tud de el nodo a descomponer
        for(k = 0; k<(1<<i); k++) {
            if(wptree->nodo[i][k])  {
            Wave_StageD(
              wptree->pwd_nodos[i][k],
              imax,
              wptree->h0, wptree->h1, lh,
              wptree->pwd_nodos[i+1][2*k] + lh-1,
              wptree->pwd_nodos[i+1][2*k+1] + lh-1
            );
        }
    }
  }
}
```

```
void IPW(WPTree* wptree, float **entrada, float *sal, int lsalida) {
    int i,imax,k;
    int n, pr;
    int lh;
    lh = wptree->lh;
```

```
        /* Introduce las señales de entrada en el árbol de descomposición */
        for (i = 0; i<wptree->nNodos; i++)  {
            n = wptree->iArbol[2*i];
            pr = wptree->iArbol[2*i+1]-1;
            wptree->ipw_nodos[n][pr] = entrada[i];
        }
        wptree->ipw_nodos[0][0] = sal;
        /* Calcula el árbol de reconstrucción */
        imax = lsalida>>wptree->nMaxNivel;
        for (i=wptree->nMaxNivel-1;i>=0;i–){
            for(k = 0; k < (1<<i); k++)  {
                if (wptree->nodo[i][k]) {
                  Wave_StageI(
                    wptree->ipw_nodos[i+1][2*k],
                    wptree->ipw_nodos[i+1][2*k+1],
                    imax, wptree->g0, wptree->g1, lh,
                    wptree->ipw_nodos[i][k]+(lh>>1)-1
                  );
                }
            }
            imax = (imax<<1);
        }
    }
```

Apéndice B

Filtros equivalentes para el cálculo de un paquete wavelet

Una forma eficiente y simple calcular los árboles wavelet de descomposición y síntesis consiste en el empleo de un equivalente $f_{lm}[n]$ y un diezmador por 2^l, con l el número del nivel y m la posición en dicho nivel. En este apéndice se incluyen las demostraciones de las expresiones que permiten calcular las respuestas al impulso de los filtros equivalentes de descomposición y síntesis, así como los respectivos programas en Matlab que hacen uso de estas expresiones.

B.1 Filtros equivalentes de descomposición

Para calcular los filtros equivalentes de descomposición hacemos uso de la siguiente notación (Figura B.1):

$$x_{l+1}[n] = \sum_{k=0}^{N-1} h^{(l)}[k] x_l[2n-k]$$

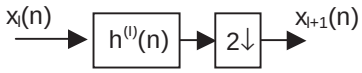


Figura B.1: Notación empleada para el cálculo de los filtros equivalentes de descomposición

Con $h^{(l)}[n]$ el filtro en el nivel l, que puede ser un pasa-alto o pasa-bajo, dependiendo del nodo m de cálculo. Para 2 niveles de descomposición (Figura B.2):

$$x_{l+2}[n] = \sum_{k=0}^{N-1} h^{(l+1)}[k] \sum_{k'=0}^{N-1} h^{(l)}[k'] x_l[4n-2k-k']$$

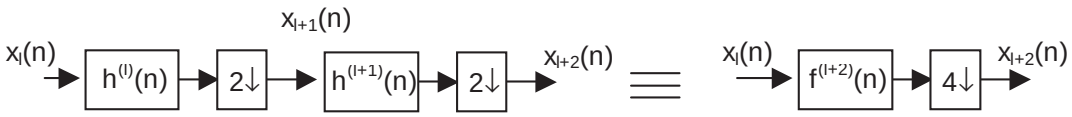


Figura B.2: Esquema de cálculo para el filtro equivalente de dos niveles de descomposición

Como esta ecuación describe la salida total en el diezmador por 4, y debemos encontrar la respuesta al impulso en la iteración $f^{(l+2)}$, interpolamos la ecuación anterior por 4, lo cual se consigue con el cambio de variable $n \to \frac{n}{4}$, y aplicamos una función delta como entrada $x_l[n]$, con el fin de obtener la respuesta al impulso:

$$f^{l+2}[n] = \sum_{k=0}^{N-1} h^{(l+1)}[k] \sum_{k'=0}^{N-1} h^{(l)}[k'] \delta[n - 2k - k']$$

Como la sumatoria más interna abarca un elemento para el cual el argumento de $\delta[n]$ produce $\delta[n] = 1$, lo cual se lleva a cabo en $k' = n - 2k$, de esta forma tenemos que:

$$f^{(l+2)}[n] = \sum_{k=0}^{N-1} h^{(l+1)}[k] h^{(l)}[n - 2k]$$

Para 3 niveles de descomposición se tiene la siguiente expresión:

$$x_{l+3}[n] = \sum_{k=0}^{N} h^{(l+2)}[k] \sum_{k'=0}^{N-1} h^{(l+1)}[k'] \sum_{k''=0}^{N-1} h^{(l)}[k''] x_l[8n - 4k - 2k' - k'']$$

En este caso se interpola por 8, y al introducir una función delta como entrada:

$$f^{(l+3)}[n] = \sum_{k=0}^{N-1} h^{(l+2)}[k] \sum_{k'=0}^{N-1} h^{(l+1)}[k'] h^{(l)}[n - 4k - 2k']$$

Nótese que la sumatoria más interna tiene la misma forma de $f^{(l+2)}$, de tal forma que integrando esta ecuación con la planteada para 2 niveles se induce que:

$$f^{(l+2)}[n] = \sum_{k=0}^{N-1} h^{(l+1)}[k] f^{(l+1)}[n - 2^{l+1}k] \quad l \geq 0$$

de la cual se derivan las siguientes expresiones finales:

$$f^{(l)}[n] = \sum_{k=0}^{N-1} h^{(l-1)}[k] f^{(l-1)}[n - 2^{l-1}k] \qquad l \geq 2 \tag{B.1}$$

$$f^{(1)}[n] = h^{(0)}[n] \tag{B.2}$$

El filtro equivalente, se consigue entonces calculando recursivamente la ecuación 3.17, y la posición particular del nodo terminal define la cadena de filtros $h^{(0)}$, $h^{(1)}$, ... que se usan en cada iteración.

Para el tamaño del filtro tenemos que $f^{(l-1)}$ tiene una longitud $N^{(l-1)}$ y su argumento es válido en el rango $f^{(l-1)}[m]\ \forall\, m = 0 ... N^{(l-1)} - 1$. Por consiguiente:

$$0 \ \leq \ n \leq (N^{(l-1)} - 1) + 2^{l-1}(N-1) \ \ l \geq 2 \tag{B.3}$$

$$0 \ \leq \ n \leq N - 1 \ \ l = 0, 1 \tag{B.4}$$

Siendo N la longitud de la respuesta al impulso de los filtros h_0y $h_1$originales.

La longitud de la respuesta al impulso del filtro equivalente en el nivel $l \geq 2$ es:

$$N^{(l)} \ = \ (N^{(l-1)} - 1) + 2^{l-1}(N-1) + 1 \ \ l \geq 2 \tag{B.5}$$

$$N^{(l)} \ = \ N \ \ l = 1 \tag{B.6}$$

Como los filtros $h^{(0)}$, $h^{(1)}$, ... tienen igual longitud de la respuesta al impulso (N), la ecuación anterior se puede reducir, por simple inducción a:

$$N^{(l)} = (2^l - 1)(N-1) + 1 \ \ l \geq 2 \tag{B.7}$$

B.2 Filtro equivalente de síntesis

El filtro equivalente para la síntesis se obtiene de la siguiente forma (Figura B.3):

$$\begin{aligned} x_l[n] \ &= \ \sum_{k=0}^{N-1} h^{(l)}[k] x_{l-1}\left[\frac{n-k}{2}\right] \quad \textit{primer nivel};\ l = 0 \\ &= \ \sum_{k=0}^{N-1} h^{(l)}[k] \sum_{k'=0}^{N-1} h^{(l-1)}[k] x_{l-2}\left[\frac{\frac{n-k}{2} - k'}{2}\right] \quad \textit{segundo nivel};\ l = 1 \end{aligned}$$

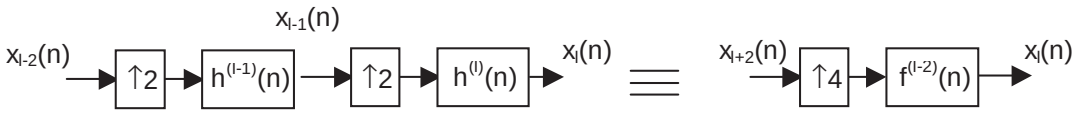


Figura B.3: Notación empleada para el cálculo de los filtros equivalentes de síntesis

En este caso vemos que es necesario diezmar la señal de entrada por 4 y cambiar x_{l-2} por δ. Ahora, si introducimos un nuevo nivel, se puede apreciar que el argumento de x_{l-2}entra en la nueva etapa afectado por $(n-k)/2$, de esta forma, por inducción se llega a:

$$f^{(-l)}[n] = \sum_{k=0}^{N-1} h^{(l-2)}[k] f^{-l+1}\left[\frac{n-k}{2}\right] \quad l \geq 2 \tag{B.8}$$

y,

$$f^{(-1)}[n] = h^{(1)}[n] \tag{B.9}$$

teniéndose en este caso que:

$$0 \leq n \leq 2(N-1) + (N^{(l-1)} - 1) \;\; l \geq 2 \tag{B.10}$$

$$0 \leq n \leq N-1 \;\; l = 0, 1 \tag{B.11}$$

lo cual implica una longitud de la respuesta al impulso igual a:

$$N^{(l)} = 2(N-1) + (N^{(l-1)} - 1) + 1 \;\; l \geq 2 \tag{B.12}$$

que en términos más simples puede expresarse como:

$$N^{(l)} = (2^l - 1)(N-1) + 1 \;\; l \geq 2 \tag{B.13}$$

B.3 Programa en Matlab para el cálculo de los filtros equivalentes de descomposición

```
function f = filtrocombinadoD(hLP, tree)
%f = filtrocombinadoD(hLP, tree)
%
% hLP respuesta al impulso del filtro prototipo pasabajo
%
% tree: estructura del árbol, leida de izquierda a derecha
% un elemento cero indica un filtro pasabajo y un 1 un pasaalto
%
% El filtro combinado resultante se retorna en forma causal
%
% ejemplo: Para devolver el filtro combinado en la posición X de la gráfica
%         o
%    h0/ \h1       f = filtrocombinado( h, [1 0 0] );
%   h0/\h1
% h0/
%   X
%Ajusta las respuesta al impulso de los filtros originales a sus ver-
siones no causales
hHP = -((-1).^(0:(length(hLP)-1))).*fliplr(hLP);
%Determina los filtros de descomposición iniciales
if ( tree(1) == 0 )
      h0 = hLP;
else
      h0 = hHP;
end
if ( length(tree) == 1 )
      f = fliplr(h0);
      return
end
if ( tree(2) == 0 )
      h1 = hLP;
else
      h1 = hHP;
end
treepos = 3;
l = 1;
```

```
% Calcula recursivamente los filtros combinados
while ( treepos-1 <= length(tree) )
    Nh0 = length(h0);
    Nh1 = length(h1);
    %Calcula la dimensión del nuevo filtro
    Nf = (Nh0-1) + (2^l)*(Nh1-1) + 1;
    f = zeros(1,Nf);
    n2 = 1;
    for n = 0:Nf-1
        for k=0:Nh1-1
            h0index = n-(2^l)*k;
            if (h0index >= 0) & (h0index < Nh0)
               f(n2) = f(n2) + h0(h0index + 1)*h1(k + 1);
            end
        end
        n2 = n2+1;
    end
    h0 = f;
    if ( treepos <= length(tree) )
        if ( tree(treepos) == 0 )
            h1 = hLP;
        else
            h1 = hHP;
        end
    end
    treepos = treepos+1;
    l = l+1;
end
```

B.4 Programa en matlab para el cálculo de los filtros equivalentes de síntesis

```
function f = filtrocombinadol(hLP, tree)
%f = filtrocombinadol(hLP, tree)
%
% hLP respuesta al impulso del filtro prototipo pasabajo
%
% tree: estructura del árbol, el primer elemento corresponde al filtro
% ubicado en la base del árbol
```

```
% un elemento cero indica un filtro pasabajo y un 1 un pasaalto
%
% El filtro combinado resultante se retorna en forma causal.
%
% ejemplo: Para devolver el filtro combinado en la posición X d la gráfica
%         o
%    h0/ \h1       f = filtrocombinado( h, [1 0 0] );
%   h0/\h1
% h0/
%   X
%Ajusta las respuesta al impulso de los filtros originales a sus ver-
siones no causales


    hLP = fliplr(hLP);
    hHP = ((-1).^(0:(length(hLP)-1))).*fliplr(hLP);
%Determina los filtros de descomposición iniciales
if ( tree(length(tree)) == 0 )
    h1 = hLP;
else
    h1 = hHP;
end
if ( length(tree) == 1 )
    f = h1;
    return
end
if ( tree( length(tree)-1 ) == 0 )
    h0 = hLP;
else
    h0 = hHP;
end
treepos = length(tree)-2;
% Calcula recursivamente los filtros combinados
while ( treepos >= 0 )
    Nh0 = length(h0);
    Nh1 = length(h1);
    %Calcula la dimensión del nuevo filtro
    Nf = Nh0 + 2*(Nh1-1);
    f = zeros(1,Nf);
```

```
        n2 = 1;
        for n = 0:Nf-1
            for k=0:Nh0-1
                h1index = (n-k)/2;
                if ( (mod(n-
                k,2) == 0) & (h1index >= 0) & (h1index < Nh1) )

                  f(n2) = f(n2) + h0(k+1)*h1(h1index+1);
                end
            end
            n2 = n2+1;
        end
        h1 = f;
        if ( treepos > 0 )
            if ( tree(treepos) == 0 )
                h0 = hLP;
            else
                h0 = hHP;
            end
        end
        treepos = treepos-1;
    end
```

Apéndice C

Programas en C para el cálculo de diezmadores e interpoladores por un factor D

Las versiones del interpolador y diezmador que se presentan a continuación requieren que la longitud de la respuesta al impulso del filtro equivalente sea divisible por cuatro. Como dichas longitudes del filtro equivalente no cumplen en todos los casos esta condición, se hace necesario realizar una extensión de la respuesta al impulso a una longitud apropiada, mediante la adición de ceros en los últimos elementos de la respuesta al impulso. Así mismo, para el correcto funcionamiento de la rutina, los coeficientes de la respuesta al impulso deben pasarse en forma reflejada y el factor ID debe ser un múltiplo de 4 lo cual se garantiza para todos filtros equivalentes, pero no en general para cualquier aplicación.

```
#define LOFLOAT(x)          _itof(_lo(x))
#define HIFLOAT(x)          _itof(_hi(x))
void interpolador(
    float *entrada, /*Entrada*/
    const int lon, /*Longitud de la entrada*/
    const float *hs, /*Coeficientes de la respuesta al impulso*/
    const int lh, /*Tamaño de la respuesta al impulso*/
    const int ID, /*Factor de interpolación*/
    float *salida /*Salida*/
) {
    int i, j, k, k1, z;
    int lhoffset;
    float aux0;
    float aux1;
    float aux2;
```

```
        float aux3;
        float in1, in2;
        double *hs1;
        double *hs2;
        _nassert(((int)(entrada) & 0x3) == 0);
        _nassert(((int)(hs) & 0x3) == 0);
        hs1 = (double*) hs;
        hs2 = (double*) (hs+2);
        lhoffset = lhmax - (lh/ID);
        z=0;
        for(j=0; j<lon; j++) {
            #pragma MUST_ITERATE(2,,2)
            for(i=ID-4; i>=0; i-=4) {
                aux0=0.0;
                aux1=0.0;
                aux2=0.0;
                aux3=0.0;
                k1 = (i>>1);
                for(k=0; k<lh/ID; k++) {
                  in1 = entrada[j+k+lhoffset];
                  in2 = entrada[j+k+lhoffset];
                  aux0+=in1*LOFLOAT(hs1[k1]);
                  aux2+=in2*LOFLOAT(hs2[k1]);
                  aux1+=in1*HIFLOAT(hs1[k1]);
                  aux3+=in2*HIFLOAT(hs2[k1]);
                  k1 += (ID>>1);
                }
                salida[z++] += aux3;
                salida[z++] += aux2;
                salida[z++] += aux1;
                salida[z++] += aux0;
            }
        }
        #pragma MUST_ITERATE(4,,4)
        for(i=0; i<lhmax; i++) {
            entrada[i] = entrada[lon+i];
        }
}
void diezmador(
        float *entrada, /*Entrada*/
        const int lon, /*Longitud de la entrada*/
```

```
        const float *hd, /*Coeficientes de la respuesta al impulso*/
        const int lh, /*Tamaño de la respuesta al impulso*/
        const int ID, /*Factor de interpolación*/
        float *salida, /*Salida*/
        int fMoverdatos /* Bandera que indica si se deben mover
        los datos en el vector de entrada */

    ) {

        int i, j, k, k1, z=0;
        int lhoffset;
        float aux0;
        float aux1;
        double *e0d1;
        double *e0d2;
        double *hd1;
        double *hd2;
        _nassert(((int)(entrada) & 0x3) == 0);
        _nassert(((int)(hd) & 0x3) == 0);
        lhoffset = lhmax - lh;
        hd1 = (double*) hd;
        hd2 = (double*) (hd+2);
        e0d1 = (double*) (entrada+lhoffset);
        e0d2 = (double*) (entrada+lhoffset+2);
        for(j=0; j<lon; j+=ID) {
            aux0=0;
            aux1=0;
            k1 = 0;
            #pragma MUST_ITERATE(8,,4)
            for(k=0; k<lh/4; k++) {

                aux0 += LOFLOAT(e0d1[k1]) * LOFLOAT(hd1[k1]);
                aux1 += LOFLOAT(e0d2[k1]) * LOFLOAT(hd2[k1]);
                aux0 += HIFLOAT(e0d1[k1]) * HI-
                FLOAT(hd1[k1]);
                aux1 += HIFLOAT(e0d2[k1]) * HIFLOAT(hd2[k1]);
                k1+=2;
            }
            salida[z++] = aux0 + aux1;
            e0d1 += ID>>1;
            e0d2 += ID>>1;
        }
        if (fMoverdatos) {
```

```
            #pragma MUST_ITERATE(4,,4)
            for (i=0; i<lhmax; i++) {
                entrada[i]=entrada[lon+i];
            }
        }
    }
```

Apéndice D

Programas en C para el cálculo de la OCWT e IOCWT

D.1 FFT real adaptada de la Ref. [PTVF92]

La rutina de cálculo de la FFT que se lista a continuación está optimizada para longitudes del vector de entrada hasta 4096 datos.

```
#define dsp_real_fft(data, N)   realft((data)-1, N,  1)
#define dsp_real_ifft(data, N)  realft((data)-1, N, -1)
# define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr
const float sinwtemp[] = {
        0.0000000000000000, //sin(pi/2^0)
        1.0000000000000000, //sin(pi/2^1)
        0.7071067811865475, //sin(pi/2^2)
        0.3826834323650898, //sin(pi/2^3)
        0.1950903220161282, //sin(pi/2^4)
        0.0980171403295606, //sin(pi/2^5)
        0.0490676743274180, //sin(pi/2^6)
        0.0245412285229123, //sin(pi/2^7)
        0.0122715382857199, //sin(pi/2^8)
        0.0061358846491545, //sin(pi/2^9)
        0.0030679567629660, //sin(pi/2^10)
        0.0015339801862848, //sin(pi/2^11)
        7.66990318742e-004, //sin(pi/2^12)
};
void four1 (float *data, unsigned int const nn, int const isign) {
        unsigned int n,mmax,m,j,istep,i;
        float wtemp,wr,wpr,wpi,wi;
        float tempr,tempi;
```

```
int indexsintable;
n=nn << 1;
j=1;
for (i=1;i<n;i+=2) {
    if (j > i) {
        SWAP(data[j],data[i]);
        SWAP(data[j+1],data[i+1]);
    }
    m=n >> 1;
    while (m >= 2 && j > m) {
        j -= m;
        m >>= 1;
    }
    j += m;
}
mmax=2;
indexsintable = 0;
while (n > mmax) {
    istep=mmax << 1;
    wtemp=isign * sinwtemp [indexsintable+1];
    wpr = -2.0*wtemp*wtemp;
    wpi = isign * sinwtemp [indexsintable];
    wr = 1.0;
    wi = 0.0;
    for (m = 1; m < mmax; m+=2){
        #pragma MUST_ITERATE(2,,2);
        for (i=m;i<=n;i+=istep) {
          j=i+mmax;
          tempr=wr*data[j]-wi*data[j+1];
          tempi=wr*data[j+1]+wi*data[j];
          data[j]=data[i]-tempr;
          data[j+1]=data[i+1]-tempi;
          data[i] += tempr;
          data[i+1] += tempi;
        }
        wr=(wtemp=wr)*wpr-wi*wpi+wr;
        wi=wi*wpr+wtemp*wpi+wi;
    }
    mmax = istep;
    indexsintable = indexsintable + 1;
}
```

```
}
void realft (float data [], unsigned int const n, int const isign)
{
    unsigned int i, i1, i2, i3, i4, np3;
    float c2, h1r, h1i, h2r, h2i;
    float wr, wi, wpr, wpi, wtemp;
    int indexsintable;
    indexsintable = 11;
    if (isign == 1) {
        c2 = -0.5;
        four1(data, n>>1, 1);
    }else{
        c2 = 0.5;
    }
    wtemp = isign * sinwtemp [indexsintable];
    wpr = -2.0 *wtemp*wtemp;
    wpi = isign * sinwtemp [indexsintable-1];
    wr = 1.0 + wpr;
    wi = wpi;
    np3 = n+3;
    #pragma MUST_ITERATE (2,,2);
    for (i = 2; i<= (n>>2); i++){
        i4 = 1+(i3 = np3 - (i2 = 1 + (i1 = i+i-1)));
        h1r = 0.5 * (data[i1] + data [i3]);
        h1i = 0.5 * (data[i2] - data [i4]);
        h2r = -c2 * (data [i2] + data [i4]);
        h2i = c2 * (data [i1] - data [i3]);
        data [i1] = h1r + wr * h2r - wi * h2i;
        data [i2] = h1i + wr * h2i + wi * h2r;
        data [i3] = h1r - wr * h2r + wi * h2i;
        data [i4] = -h1i + wr * h2i + wi * h2r;
        wr = (wtemp = wr) * wpr - wi * wpi + wr;
        wi = wi * wpr + wtemp * wpi + wi;
    }
    if (isign == 1) {
        data [1] = (h1r = data [1]) + data [2];
        data [2] = h1r - data [2];
    }
    else {
        data [1] = 0.5 * ((h1r = data [1]) + data [2]);
        data [2] = 0.5 * (h1r - data [2]);
```

```
            four1 (data, n>>1, -1);
        }
    }
```

D.2 OCWT basada en FFT

```
float **ghat; //Transformadas de Fourier precalculadas de
              //la función base
float *in;    //Arreglo auxiliar para el cáculo de las FFTs
float x_ocwt_pasadas[lenpsi+lensig];
void ocwt(

    float **c,  //matriz donde se devuelven los coeficientes calculados
    float *sig, //vector con la señal de entrada a descomponer.
    float *sms, //vector con los valores de las escalas
    int nes     //número de escalas que se deben procesar

) {

    int i, m, Nfft = lensig + lenpsi;
    float sm;
    // completar la señal anterior con la señal de entrada
        #pragma MUST_ITERATE (64,,8);
        for (i=lenpsi; i < Nfft; i++)
            x_ocwt_pasadas[i] = sig[i-lenpsi];
        dsp_real_fft(x_ocwt_pasadas,Nfft);
    // Calcular la convolución entre la señal y la wavelet dilatada y es-
    calada
    for (m = 0; m < nes; m++){
        sm = sms[m];
        // cálculo de la convolución entre la señal y el átomo wavelet


            #pragma MUST_ITERATE (64,,2);
            for(i = 0; i < Nfft; i += 2)
            {
              in[i] = x_ocwt_pasadas[i]*ghat[m][i]
                      - x_ocwt_pasadas[i+1]*ghat[m][i+1];
              in[i+1] = x_ocwt_pasadas[i]*ghat[m][i+1]
                       + x_ocwt_pasadas[i+1]*ghat[m][i];
            }
            dsp_real_ifft(in,Nfft);
```

```
            // Mete el vector calculado en la matriz de coefi-
            cientes de la OCWT
            // descartando los primeros Nh elementos
                sm = sqrt(sm) * (2.0/Nfft);
                #pragma MUST_ITERATE (64,,2);
                for (i = 0; i < lensig; i++)
                  c[m][i] = in[lenpsi+i] * sm;
        }
        #pragma MUST_ITERATE (64,,4);
        for(i = 0; i < lenpsi; i++)
            x_ocwt_pasadas[i] = sig[i];
}
```

D.3 IOCWT basada en FFT

```
float *sighat; //Transformadas de Fourier de la señal unidimensional
float *chat; //Arreglo auxiliar para el cálculo de la FFT de los coeficientes
// variable global con el contenido de la salida anterior
float x_iocwt_pasadas[lenpsi];
void iocwt(
    float *sig,  //arreglo donde se almacena la señal de retorno
    float **coefs,  //matriz de coeficientes de entrada
    float *sms, //vector de escalas
    int nes   //número de escalas que se deben procesar
) {
    int i, m, Nfft = lensig + lenpsi;
    // inicializar los vectores para las transformadas en cero
        #pragma MUST_ITERATE (64,,8);
        for (i = 0; i < Nfft; i++) {
            sighat [i] = 0;
        }
    // Calcular la suma de los aportes del átomo para cada escala.
    for (m = 0; m < nes; m++) {
        // Calcula la fft de los coeficientes de entrada para
        // cada una de las escalas
        #pragma MUST_ITERATE (64,,8);
        for (i = 0; i < lensig; i++)
            chat[i] = coefs[m][i];
```

```
            #pragma MUST_ITERATE (64,,8);
            for (; i < Nfft; i++)
                chat[i] = 0;
            dsp_real_fft(chat,Nfft);
            // calcular la convolución en el dominio de la frecuencia
            // y acumular su suma para todas las escalas
            #pragma MUST_ITERATE (64,,8);
            for (i = 0; i < Nfft; i += 2) {
                sighat[i] += chat[i]*ghat[m][i] -
                chat [i+1]*ghat[m][i+1];
                sighat[i+1] += chat[i]*ghat[m][i+1] + chat[i+1]*ghat[m][i];
            }
        }
        // calcular la señal en el dominio del tiempo por medio de la ifft
            dsp_real_ifft(sighat, Nfft);
        // Aplicar el solapamiento y suma
        // sumar los ultimos datos de la salida pasada de la ifft
        #pragma MUST_ITERATE (64,,8);
        for(i = 0; i < Nfft-lensig; i++)
            sig[i] = sighat[i] * (2.0/Nfft) + x_iocwt_pasadas[i];
        // guardar la entrada actual como la salida anterior
        #pragma MUST_ITERATE (64,,8);
        for(i = 0; i < lensig; i++)
            x_iocwt_pasadas[i] = sighat[Nfft - lensig + i] * (2.0/Nfft);
}
```

Apéndice E

Algoritmos para el sistema de compresión de señales

E.1 Algoritmo para el cálculo de la mejor base

```
void Wave_Stage(float *entrada, int lentra-
da,float *h0, float *h1, int lh,float *apr, float *det, int *en_opt)
{
    int i,j,ind;
    float entroph0,entroph1,entropp;
    float aux0,aux1;
    /*Convolución y subsequente submuestreo de manera eficiente
    usando filtros polifásicos con un conmutador a la entrada*/
    ind = 0;
    entroph0 = 0;
    entroph1 = 0;
    entropp = 0;
    for (i = 0; i < (lentrada/2); i++)
    {
      aux0 = 0;
      aux1 = 0;
      // entropía de shannon para el nodo padre
      if(entrada[i+lh] != 0)entropp =
            entropp + pow(entrada[i+lh],2) * log(pow(entrada[i+lh],2));
      if(entrada[(lentrada/2) + lh + i] != 0)
          entropp = entropp +  pow(entrada[(lentrada/2) + lh + i],2)
                      * log(pow(entrada[(lentrada/2) + lh + i],2));
      for (j = 0; j < lh; j++)
      {
         aux0 = aux0 + entrada[2*i+lh+1-j] * h0[j];
```

```
            aux1 = aux1 + entrada[2*i+lh+1-j] * h1[j];
          }
          apr[ind] = aux0;
          det[ind++] = aux1;
          //entropía de shannon para nodos hijos
          if(aux0 != 0)entroph0 = entroph0 + pow(aux0,2) * log(pow(aux0,2));
          if(aux1 != 0)entroph1 = entroph1 + pow(aux1,2) * log(pow(aux1,2));
        }
        entroph0 = entroph0 + entroph1;
        if (-entroph0 < -entropp) // si se cumple,
                                  quiere decir que el nodo
                                  se puede subdividir.
        {
          *(en_opt) = 1;
          *(en_opt + 1) = 1;
        }
      /*Actualización de las condiciones iniciales. Se
      copian las últimas N-1 muestras.*/
      for (i = 0;i < lh;i++)
      *(entrada + i) = entrada[lentrada+i];
}
```

E.2 Algoritmo de cálculo del LPC

```
void LPC_quin(float *entrada,float *A,int lentrada,int ordenlpc)
{
      int i,j;
      float numerador,denominador,g;
      float *err,*k,*B,*x,*R;
      err = (float *)malloc(ordenlpc*sizeof(float));
      k = (float *)malloc(ordenlpc*sizeof(float));
      B = (float *)malloc(ordenlpc*sizeof(float));
      R = (float *)malloc(ordenlpc*sizeof(float));
      x = (float *)malloc((lentrada + ordenlpc)*sizeof(float));
      for (i = 0; i<ordenlpc;i++) //inicializacion del LPC
      {
            A[i] = 0.0;
            x[i+lentrada] = 0.0;
      }
      memcpy(x,entrada,lentrada*sizeof(float));
      R[0] = 0;
```

```
        for (j = 0;j < ordenlpc + 1; j++)
        {
            for (i = 0;i < lentrada; i++)
                R[j] += x[i]*x[i+j]/lentrada; //Autocor-
                relación normalizada
        }
        err[0] = R[0];
        k[0] = 0;
        //algoritmo de Levinson-Durbin
        for (int index = 0;index < ordenlpc;index++)
        {
            numerador = R[index+1];
            for (i = index; i>=1; i–)
                numerador += A[index-i]*R[i];
            denominador = -1*err[index];
            k[index] = numerador/denominador;
                        //coeficientes PARCOR
            for (j = 0 ; j <index ; j++)
                B[j] = A[j]+k[index]*A[index-j-1];
            B[j] = k[index];
            for (int k = 0; k <= index; k++)
                A[k] = B[k];
            err[index+1] = (1.0-pow(k[index],2))*err[index];
        }
        g = 0;
        for (i = 0; i < ordenlpc; i++)
            g += A[i]*R[i+1];
        g = sqrt(R[0]+g); //calculo de la ganancia
        A[ordenlpc] = g;
    }
```

Apéndice F

Condiciones de reconstrucción perfecta de un QMF inverso

Sea la estructura de la Figura F.1, un banco de filtros QMF inverso:

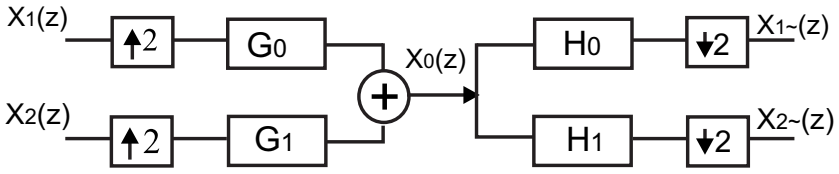


Figura F.1: Filtros QMF inverso de dos canales

Debido al carácter no LTI de los diezmadores e interpoladores, es posible que los filtros H_0, H_1, G_0 y G_1 no cumplan con las ecuaciones de reconstrucción para esta estructura. lo cual se verifica a continuación :

$$X_0(z) = X_1(z^2)G_0(z) + X_2(z^2)G_1(z) \tag{F.1}$$

$$\tilde{X}_1(z) = X_1(z)z^{-k} = \frac{1}{2}X_0(z^{\frac{1}{2}})H_0(z^{\frac{1}{2}}) + \frac{1}{2}X_0(-z^{\frac{1}{2}})H_0(-z^{\frac{1}{2}}) \tag{F.2}$$

$$\tilde{X}_2(z) = X_2(z)z^{-k} = \frac{1}{2}X_0(z^{\frac{1}{2}})H_1(z^{\frac{1}{2}}) + \frac{1}{2}X_0(-z^{\frac{1}{2}})H_1(-z^{\frac{1}{2}}) \tag{F.3}$$

Expandiendo para $\tilde{X}_1(z)$:

$$\tilde{X}_1(z) = \frac{1}{2}[X_1(z)G_0(z^{\frac{1}{2}}) + X_2(z)G_1(z^{\frac{1}{2}})]H_0(z^{\frac{1}{2}})$$

$$+\frac{1}{2}[X_1(z)G_0(-z^{\frac{1}{2}}) + X_2(z)G_1(-z^{\frac{1}{2}})]H_0(-z^{\frac{1}{2}})$$

$$= X_1(z)\left[\frac{1}{2}G_0(z^{\frac{1}{2}})H_0(z^{\frac{1}{2}}) + \frac{1}{2}G_0(-z^{\frac{1}{2}})H_0(-z^{\frac{1}{2}})\right]$$

$$+ X_2(z)\left[\frac{1}{2}G_1(z^{\frac{1}{2}})H_0(z^{\frac{1}{2}}) + \frac{1}{2}G_1(-z^{\frac{1}{2}})H_0(-z^{\frac{1}{2}})\right] \tag{F.4}$$

Para recuperar el canal $X_1(z)$ y evitar interferencia intersímbolo, el canal $X_2(z)$ debe ser eliminado de la expresión anterior. Se tienen entonces dos condiciones:

$$\frac{1}{2}G_0(z^{\frac{1}{2}})H_0(z^{\frac{1}{2}}) + \frac{1}{2}G_0(-z^{\frac{1}{2}})H_0(-z^{\frac{1}{2}}) = z^{-k}$$

$$G_0(z)H_0(z) + G_0(-z)H_0(-z) = 2z^{-2k}$$

ó

$$G_0(\omega)H_0(\omega) + G_0(\omega-\pi)H_0(\omega-\pi) = 2exp(-j2\omega k) \tag{F.5}$$

y la segunda condición:

$$G_1(z)H_0(z) + G_1(-z)H_0(-z) = 0 \tag{F.6}$$

Al analizar el canal $\tilde{X}_2(z)$, se tiene expresiones similares, salvo que cambia H_0 por H_1y G_0 por G_1. En general:

$$H_{0,1}(z)G_{0,1}(z) + H_{0,1}(-z)G_{0,1}(-z) = 2z^{-2k}$$

$$H_{0,1}(\omega)G_{0,1}(\omega) + H_{0,1}(\omega-\pi)G_{0,1}(\omega-\pi) = 2exp(-j2\omega k) \tag{F.7}$$

$$H_{0,1}(\omega)G_{1,0}(\omega) + H_{0,1}(\omega-\pi)G_{1,0}(\omega-\pi) = 0 \tag{F.8}$$

Al verificar las condiciones anteriores con las formas del filtro QMF conjugado tenemos (tomando sólo los índices 0):

$$H(z)\left[2z^{-(N-1)}H(z^{-1})\right] + H(-z)\left[-2z^{-(N-1)}H(-z^{-1})\right] = 2z^{-2k} \tag{F.9}$$

Como N es de longitud par. $(-1)^{-(N-1)} = -1$, lo que finalmente permite reducir la expresión anterior a:

$$[H^2(z) - H^2(-z)]z^{-(N-1)} = z^{-2k} \tag{F.10}$$

De esta forma se tendría que: $k = \frac{N-1}{2}$ y $H^2(z) - H^2(-z) = 1$, lo cual es falso, según la ec. (3.16), lo cual implica que los filtros no cumplen las condiciones de reconstrucción perfecta del banco de filtros QMF inverso.

Como no hay reconstrucción perfecta, y los filtros no se pueden cambiar, es necesario modificar la estructura del sistema, lo cual se consigue analizando el diezmado:

$$\tilde{X}_1(z) = \frac{1}{2}\tilde{X}_0(z^{\frac{1}{2}})H_0(z^{\frac{1}{2}}) + \frac{1}{2}\tilde{X}_0(-z^{\frac{1}{2}})H_0(-z^{\frac{1}{2}}) \tag{F.11}$$

En la ecuación (F.11), se observa que el segundo término debe ser negativo, lo cual se consigue asumiendo $\tilde{X}_0(z) = X_0(z)z^{-1}$, que en diagrama de bloque, se representa en la Figura F.2.

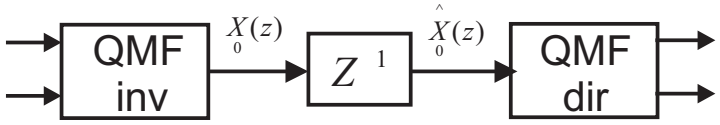


Figura F.2: Filtros QMF inverso de dos canales y con reconstrucción perfecta

F.1 QMF inverso de cuatro canales

Para identificar las características y dependencias del corrimiento entre la etapa QMF inversa a la etapa QMF directa de cuatro canales, asumimos que en un árbol uniforme, se introduce la señal por un único canal. Se puede apreciar en las ecuaciones del QMF inverso de dos canales que esta suposición produce las mismas ecuaciones de reconstrucción perfecta. El modelo simplificado de cómputo se muestra en la Figura F.3.

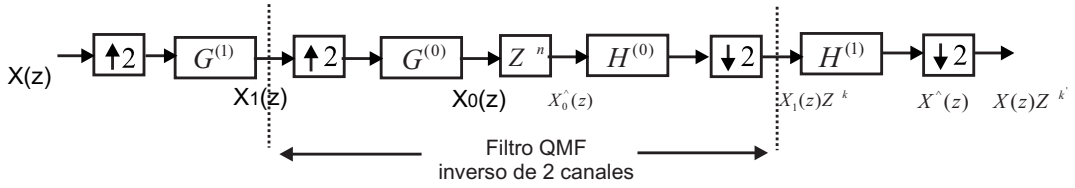


Figura F.3: Estructura de filtros QMF de 4 canales

Notese que el filtro QMF inverso de dos canales ubicado entre las señales$X_1(z)$ y $X_1(z)Z^{-k}$ debe cumplir las ecuaciones de reconstrucción perfecta anteriormente indicadas y calculadas, por lo que la Figura F.3 se transforma en el esquema de la Figura F.4.

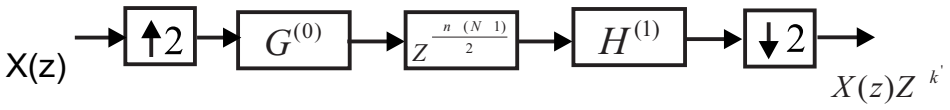


Figura F.4: Estructura simplificada de filtros QMF de 4 canales

Con $G^{(l)}(z)$; $H^{(l)}(z)$: filtros de síntesis/reconstrucción en el nivel l.

De nuevo, este filtro debe satisfacer la condición de un QMF inverso de dos canales, por lo tanto:

$$k' = \frac{k+(N+1)}{2} = \frac{n}{4} + \frac{3(N-1)}{4} = \frac{3N}{4} + \frac{n-3}{4} \tag{F.12}$$

Si la respuesta al impulso N tiene una longitud múltiplo de 4 : $4, 8, 12, 16....$, se tiene que, para reconstrucción perfecta, $k^{'}$debe ser entero, lo cual es posible solo sí $\frac{n-3}{4} \in \mathbb{Z} = i$, lo que se verifica para $n = 3, 7, 11, 15....3 + 4i$. Lo anterior significa que el sistema se sincroniza (reconstrucción perfecta) solo si el mínimo corrimiento n es 3 ó múltiplo aditivos de 4 en 4 ($4 = \#de$canales)

La ecuación anterior solo es válida para las Daubechies 2,4,6.. pero cuando N es múltiplo de 2, no divisible por 4, la reconstrucción se consigue solo sí:

$$N = 2j, j = 1, 2, 3, 4 \in N$$

$$k^{'} = \frac{3.2j}{4} + \frac{n-3}{4} = \frac{3}{2}j + \frac{n-3}{4} \tag{F.13}$$

Excluyendo los N múltiplos de 4, se tiene $j = 1, 3, 5, 7...1 + 2i$

$$k^{'} = \frac{3}{2}(1 + 2i) + \frac{n-3}{4} = \frac{3}{2} + 3i + \frac{n-3}{4} \tag{F.14}$$

Como $3i$ siempre es entero, interesa eliminar el término $\frac{3}{2}$, lo cual se consigue si

$$\frac{3}{2} + \frac{n-3}{4} \quad = \quad l, (l \in \mathbb{Z} = 1, 2, 3, 4...) \tag{F.15}$$

entonces

$$n = 4\left(l - \frac{3}{2}\right) + 3 = 4l - 3 = 1, 2, 9, ... \tag{F.16}$$

En general, para 4 canales, el corrimiento entre la etapa inversa y directa debe ser:

Si N es múltiplo de 4: $n = 3 + 4i, i = 0, 1, 2...$

Si N es múltiplo de 2 no divisible por 4: $n = 1 + 4i$

F.2 QMF inverso de M-canales

Generalizando los resultados para $M-canales$ a partir del banco de filtros de 4 canales, tenemos en forma recursiva que el corrimiento que sufre un canal es:

2 canales; 1 nivel: $k_1 = 2^{-1}n + (N-1).2^{-1}$

4 canales; 2 niveles: $k_2 = 2^{-1}k_1 + (N-1).2^{-1}$
$= 2^{-2}n + (N-1)(2^2-1).2^{-2}$

8 canales; 3 niveles: $k_3 = 2^{-1}k_2 + (N-1).2^{-1}$
$= 2^{-3}n + (N-1)(2^3-1).2^{-3}$

2^l canales; l niveles: $k_l = 2^{-l}n + (N-1)(2^l-1).2^{-l}$
$\frac{n+(N-1)(2^l-1)}{2^l}$

Como k_l debe ser entero; $k_l = 1, 2, 3, 4, 5, 6, 7....$ tenemos que:

$$n = 2^l k_l - (N-1)(2^l - 1) \tag{F.17}$$

lo cual implica que una vez sincronizado el sistema, este seguirá sincronizado si la señal transmitida se desplaza 2^l muestras.

Para determinar el mínimo n, será necesario evaluar la ecuación (F.17) para diferentes $k_l = 1, 2, 3, 4$ y tomar solo n positivos (de hecho el primero), una forma rápida de encontrarlo es:

$$2^l k_l - (N-1)(2^l - 1) > 0 \tag{F.18}$$

entonces

$$k_l > \frac{(N-1)(2^l - 1)}{2^l} \tag{F.19}$$

El método es el siguiente: Se evalúa k_l con la desigualdad anterior, debido a que este da un valor no entero, se redondea al entero siguiente más grande, este valor denotada por $[k_l]$ se usa para evaluar la ecuación de n, así:

$$n_{minimo} = 2^l [k_l] - (N-1)(2^l - 1) \tag{F.20}$$

con

$$\lceil k_l \rceil = \left\lceil \frac{(N-1)(2^l - 1)}{2^l} \right\rceil \tag{F.21}$$

finalmente:

$$n = 2^l i + 2^l \left\lceil \frac{(N-1)(2^l - 1)}{2^l} \right\rceil - (N-1)(2^l - 1) \tag{F.22}$$

$$i = 0, 1, 2, 3, 4...$$

Apéndice G

Tarjeta de adquisición de datos

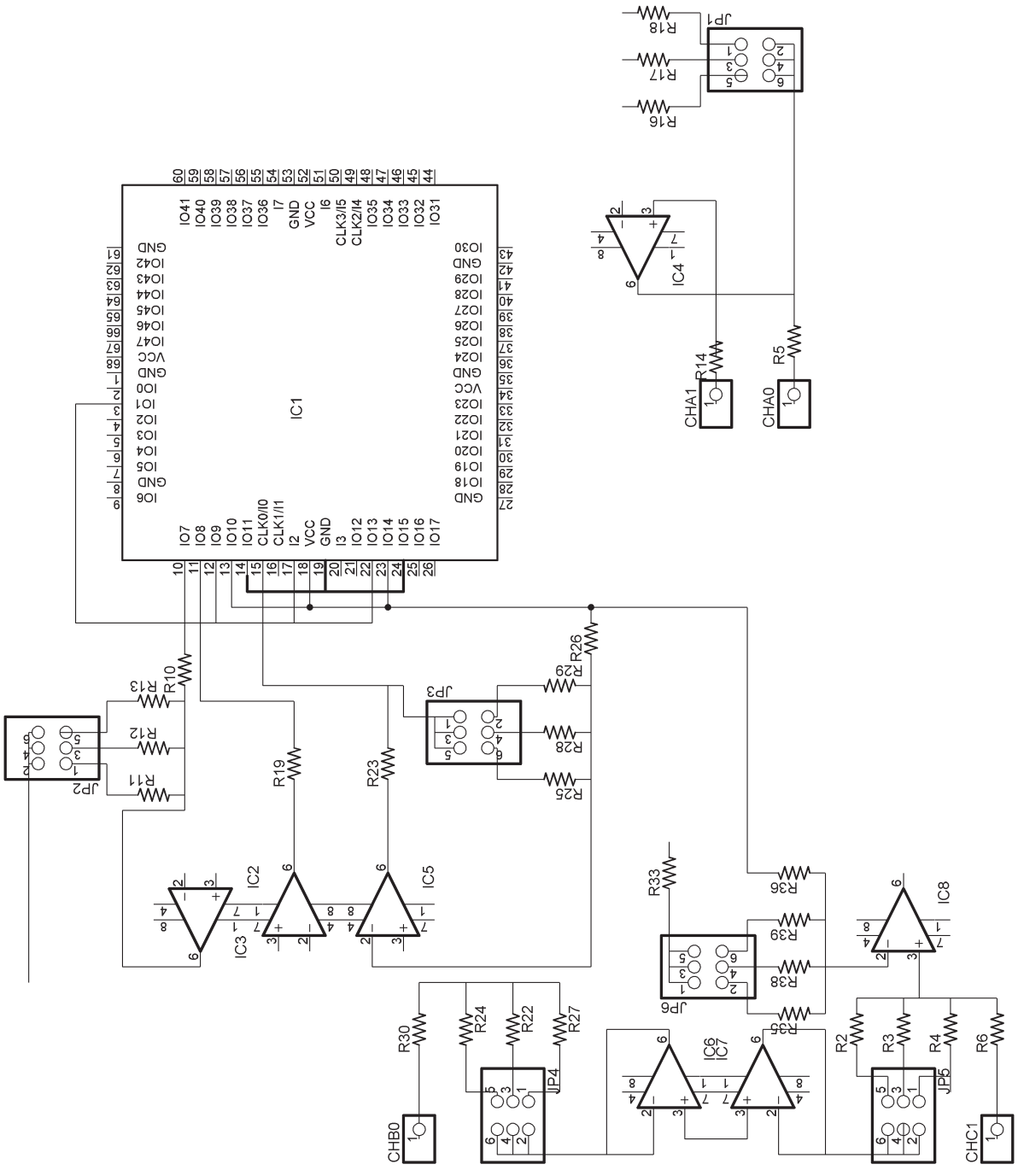


Figura G.1: Diagrama esquemático del sistema de adquisición de 6 canales simultáneos

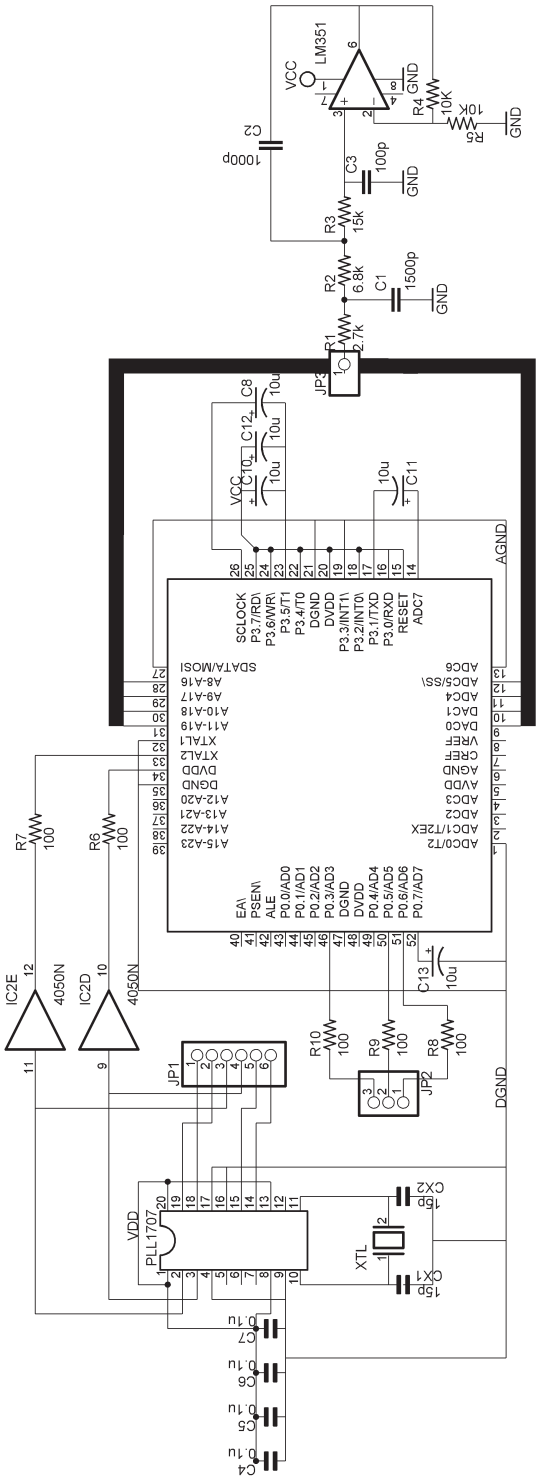


Figura G.2: Diagrama esquemático del sistema de reproducción de 8 canales simultáneos

Apéndice H

Glosario de Términos

Algoritmo de Mallat. Término equivalente a la *FWT*. [RB98]

CWT (*Continuos Wavelet Transform*). Transformada Wavelet Continua. Versión de la WT aplicable a señales continuas que hace uso de una wavelet madre continua. [Teo98]

DCT (*Discrete Cosine Transform*). Transformada Discreta del Coseno. Transformada especial que usa la función coseno como base ortogonal. [SN97]

DFT (*Discrete Fourier Transform*). Transformada de Fourier que usa una exponencial discreta compleja como base ortogonal. [PM92]

DSP (*Digital Signal Processor*). Procesador Digital de Señales. Tipo especial de microprocesador que cuenta con una arquitectura y conjunto de instrucciones optimizadas para la implementación de algoritmos de tratamiento digital de señales. [LF94]

DWT (*Discrete Wavelet Transform*). Transformada Wavelet Discreta. Versión especial de la OCWT que emplea para el muestreo del mapa frecuencia-tiempo un conjunto de escalas en potencias de dos o diádicas. Puede ser implementada eficientemente a través del algoritmo de Mallat. [Teo98, RB98]

Estructura Polifase. Forma computacionalmente eficiente que permite implementar diezmadores e interpoladores por medio de un conjunto de filtros digitales FIR de corta longitud. [Fli94, PM92].

FFT (*Fast Fourier Transform*). Transformada Rápida de Fourier. Algoritmo que permite calcular eficientemente la *DFT* con una complejidad computacional de $\Theta(n) = n \log_2 n$.[PM92]

FWT (*Fast Wavelet Transform*). Transformada Wavelet Rápida. Algoritmo que

permite calcular la DWT con un costo computacional de $\Theta(n)$. También conocido con el nombre de algoritmo de Mallat. [Teo98, RB98]

IDFT. (*Inverse Discrete Fourier Transform*) Transformada Discreta de Fourier Inversa. Versión inversa de la *DFT*. [PM92]

LPC (*Linear Prediction Coefficients*). Coeficientes de Predicción Lineal. Coeficientes de un tipo especial de filtros digitales que habitualmente se usan para el modelamiento de fonemas. [DHP00]

MRA (*Multiresolution Analysis*). Análisis Multiresolución. Término referido al análisis de una señal discreta en diferentes subespacios con diferentes niveles de resolución. Ver sección 3.1 y [Fli94, RB98].

OCWT (*Overcomplete Wavelet Transform*). Transformada Wavelet Sobrecompleta. Versión de la *CWT* aplicable a señales discretas en la cual se realiza un muestreo al mapa frecuencia-tiempo [Teo98]. Se denomina también *SCWT*[GKMM89].

Paquete Wavelet. Versión mejorada de la DWT en el que se realizan particiones por medio un conjunto de filtros pasabajo/diezmador y pasaalto/diezmador no solo a las aproximaciones sino también a los detalles[Wic94].

QMF (Quadrature Mirror Filter). Filtro de espejo en cuadratura.

SCWT (*Sampled CWT*). Transformada Wavelet Muestreada. Término empleado por ciertos autores para referirse a la *OCWT*. [GKMM89]

STFT (*Short Time Fourier Transform*). Transformada Corta de Fourier. Una versión especial de la Transformada de Fourier que permite la localización frecuencia-tiempo de las componentes. Se define como:

$$(\mathcal{S}f)(\omega, T) = \langle f(t)\,,\, \exp(j\omega t) w(t-T) \rangle$$

donde $w(t)$ es una función ventana de longitud fija. [SN97]

Transformada Wavelet. Transformada que permite la localización frecuencia-tiempo de las componentes, para lo cual hace uso de una función ventana de longitud variable (*wavelet madre*) como función base. Se define como:

$$(\mathcal{W}_g f)(s, T) = \left\langle f(t)\,,\, s^{1/2} g(s(t-T)) \right\rangle$$

donde $g(t)$ es la wavelet madre.[SN97, Teo98]

WT (*Wavelet Transform*). Transformada Wavelet

Wavelet madre. Función de energía y media cero usada como función base para el cálculo de la transformada wavelet. [SN97, Teo98]

WP (*Wavelet Packet*). Paquete Wavelet [Wic94].

Apéndice I

Artículos Publicados