UNIVERSIDAD AUTÓNOMA DE SAN LUIS POTOSI

FACULTAD DE CIENCIAS

ENCRIPTACIÓN IMPLEMENTADA EN UN FPGA DE INFORMACIÓN COMPRIMIDA CON LA TRANSFORMADA ONDELETA

TESIS
QUE PARA OBTENER EL GRADO DE
MAESTRO EN CIENCIAS APLICADAS

PRESENTA

MARIO ALBERTO ALMAZÁN MONTELONGO

ASESORES DE TESIS:

DRA. MARCELA MEJÍA CARLOS
DR. JOSÉ SALOMÉ MURGUÍA IBARRA

SAN LUIS POTOSI, SLP. AGOSTO 2007

UNIVERSIDAD AUTÓNOMA DE SAN LUIS POTOSI

FACULTAD DE CIENCIAS

ENCRIPTACIÓN IMPLEMENTADA EN UN FPGA DE INFORMACIÓN COMPRIMIDA CON LA TRANSFORMADA ONDELETA

I. E. MARIO ALBERTO ALMAZÁN MONTELONGO

SINODALES

DRA. MARCELA MEJÍA CARLOS (ASESOR)

DR. JOSÉ SALOMÉ MURGUÍA IBARRA (ASESOR)

DR. RAUL BALDERAS NAVARRO

DR. LUIS FELIPE LASTRAS MARTÍNEZ

SAN LUIS POTOSI, SLP. AGOSTO 2007

Gracias a mis padres y por sus apoyos y consejos quienes desde primaria siempre me alentaron a seguir estudiando.

Gracias a mis hermanos Lalo, Iliana Y Miguel por los buenos y malos momentos que pasamos durante tanto tiempo.

Y gracias a mi novia Jeovana por su apoyo durante todo este tiempo.

Agradecimientos

A mis asesores, la Dra. Marcela Mejía Carlos y el Dr. José Salomé Murguía Ibarra, que gracias a su paciencia y apoyo ha sido posible el desarrollo de este trabajo.

Al director del Instituto de Investigación en Comunicación Óptica Dr.Alfonso Lastras Martínez, y al secretario académico Dr. Gustavo Ramírez Flores por todas las facilidades.

Agradezco el apoyo otorgado por CONACyT mediante la beca otorgada para la realización de los estudios de Maestría.

Al laboratorio de Comunicaciones del IICO-UASLP donde se realizó el trabajo para esta tesis, con financiamiento parcial de los proyectos de PROMEP/103.5/03/1118 PTC-63 y PTC-66, asi como del proyecto denominado Integración y Fortalecimiento del CA de Matematicas Aplicadas P/CA-21 2066-24-14.

Por otro lado agradezco a los profesores que han ido guiando mi desarrollo académico y personal.

Agradezco el apoyo de todos mis amigos me han ayudado directa e indirectamente desde la licenciatura hasta la realización de este trabajo.

Encriptación implementada en un FPGA de información comprimida con la Transformada Ondeleta

Mario Alberto Almazán Montelongo
Instituto de Investigación en Comunicación Óptica

Agosto del 2007

Resumen

En este trabajo se realiza la implementación numérica y experimental de un proceso de encriptación de información de voz comprimida con la Transformada Ondeleta (TO). El esquema de compresión basado en la herramienta estándar de la TO es presentado e implementado numéricamente en dos programas diferentes. Así mismo, un algoritmo de encriptación basado en autónomas celulares, el cual pasó todos los estándares de seguridad de la NIST, es presentado e implementado de manera numérica y experimental. Para la etapa experimental se utilizó y configuró un FPGA, logrando una rapidez para la encriptación y su síntesis.

Los resultados numéricos y experimentales del proceso completo de encriptación de información comprimida se ilustran usando diferentes tipos de señales de voz.

Índice general

Índice de figuras

Índice de Tablas

1

Introducción

En la actualidad han surgido un gran número de aplicaciones relacionadas con el manejo y procesamiento de señales multimedia. Una característica en dichas aplicaciones es el gran uso de recursos de memoria y cómputo para su funcionamiento. De ahí que se tenga la necesidad de utilizar algún esquema de compresión, que nos permita tener flexibilidad para almacenar y transmitir información. Algunas de las ventajas de comprimir información son, por ejemplo, 1. La miniaturización de información, es decir, almacenamiento de mayor información en el mismo espacio, 2. Transmisión mas rápida de datos, 3.Utilizar menor ancho de banda en la transmisión de datos, entre otras.

En este ámbito, la transformada ondeleta ha resultado ser una herramienta muy potente para procesar de manera eficiente, señales que involucran grandes cantidades de información. Las ondeletas son un descubrimiento relativamente nuevo en las matemáticas aplicadas. El interés por las ondeletas ha crecido en las últimas dos décadas debido a que las ondeletas proveen una herramienta matemática muy sencilla con una gran variedad de aplicaciones y su implementación es en muchas ocasiones fácil de realizar. Por ejemplo, en la referencia [8] se implemento y aplicó un esquema de compresión basado en ondeletas a señales de voz, mostrando su implementación y buenas tasas de compresión para diferentes umbrales y ondeletas.

Sin embargo hoy en día en algunos casos es necesario tener la confiabilidad de que nadie ajeno pueda tener o hacer uso de la información comprimida ya sea almacenada o transmitida, es por eso que no solo se debe de tener un sistema de compresión, si no que, sera necesario aplicar algún tipo de clave secreta a la información comprimida para que esto no suceda. Esto se puede hacer mediante el uso de un sistema de encriptación. Actualmente existe un gran número de sistemas de encriptación, donde su principal objetivo es el de proteger información por medio de un algoritmo que hace uso de una o más llaves. Muchos de estos sistemas sacrifican el tiempo de procesamiento para tener un encriptador mas confiable o viceversa, el encriptador es menos confiable pero se logra un menor tiempo en el proceso de encriptado y desencriptado, y en algunos casos la información se encripta de manera parcial. Un sistema de encriptación que ha resultado confiable y fácil de implementar de manera digital es el sistema de encriptación ESAC [14] basado en la sincronización en Autómatas Celulares, donde el generador de llaves de dicho sistema paso las pruebas de la NIST [2, 20] resultando ser confiable para usos criptográficos.

En este trabajo de tesis se propone implementar de manera conjunta las etapas de compresión y encriptación de información de voz. Dicha implementación se probará tanto de manera numérica, como experimental, donde la etapa de compresión se basa en la transformada ondeleta, mientras que la encriptación en el sistema ESAC.

La estructura de este trabajo es de la siguiente manera. El Capitulo 2 presenta las bases necesarias para describir de manera general la Transformada Ondeleta (TO). Se revisan las versiones continua y discreta, haciendo una mayor referencia al caso discreto basándose en el Análisis Multi-Resolución para su implementación numérica [18]. Se detalla el esquema de compresión en términos de la energía de los coeficientes de la TO como se realizo en [8], siendo una parte fundamental para el actual trabajo. El Capitulo 3 describe la estructura e implementación del sistema de encriptación ESAC, y se explica de manera general las partes fundamentales de dicho sistema como lo son el generador de llaves y las funciones que realizan la encriptación y desencriptación. La descripción de la implementación numérica y experimental del sistema completo, el cual comprende las etapas de compresión y encriptación, se muestra en el Capítulo 4, donde además, se muestran los resultados obtenidos para diferentes señales de voz. Las conclusiones finales que se obtuvieron de la aplicación del sistema completo a información de voz, así como el posible trabajo a futuro se discute en el Capitulo 5. Finalmente, se anexan los algoritmos de un solo tiempo que se emplearon en el sistema ESAC, los **listados** de los programas utilizados en las implementaciones numérica y experimental, así como las especificaciones generales de la tarjeta FPGA utilizada para la implementación experimental.

2

Teoría Básica Ondeleta

Para el procesamiento de señales existen diferentes técnicas para obtener "información" de las señales y en muchas ocasiones son complejas, pero que dependiendo del enfoque resultan ser muy útiles. La teoría de Fourier ha resultado ser muy eficiente para el procesamiento de señales. En particular, es una herramienta adecuada para varios tipos de señales cuyas propiedades estadísticas no varien en el tiempo. A través de la teoría de Fourier se puede obtener información espectral de la señal, es decir, las frecuencias de las que se compone la señal.

Sin embargo, la Transformada de Fourier (TF) para cierta clase de señales, como señales no estacionarias, carece de localización en el tiempo y no proporciona una información adecuada. Una alternativa para el análisis de este tipo de señales es la Transformada Ondeleta (TO), la cual es ideal para trabajar con señales no estacionarias y transitorias, debido a que brinda información en tiempo y frecuencia de la información analizada, en contraste con el análisis en Fourier. Además, desde el punto de vista ingenieril, surge como una herramienta poderosa para procesar señales que involucran grandes cantidades de información, ya que ha sido eficiente en la compresión de datos, permitiéndonos manipular y almacenar dicha información de manera más adecuada.

2.1 Bases matemáticas

En este trabajo se analizarán con mayor frecuencia datos de tiempo discreto, por lo que resulta conveniente describir algunos conceptos y notación involucrados en este dominio, pero considerando en algún momento teoría cuando el tiempo es continuo.

Denotamos a $\mathbb{R}$ como el conjunto de números reales y a $\mathbb{Z}$ como el conjunto de números enteros. Por brevedad, llamaremos a $f(t)$ como función continua, mientras que a $f[n]$ como función discreta o secuencia.

Una función continua $f(t)$ pertenece al espacio $L^2(\mathbb{R})$, $f(t) \in L^2(\mathbb{R})$, si cumple con

$$\int_{-\infty}^{\infty} |f(t)|^2 dt < \infty, \tag{2.1}$$

y una secuencia $f[n]$ pertenece a $l^2(\mathbb{Z})$ si satisface

$$\sum_{n} |f[n]|^2 dt < \infty. \tag{2.2}$$

De hecho, en la literatura de procesamiento de señales, tales espacios son conocidos como espacios de energía ya que por ejemplo la energía de $f[n]$ se define como

$$E_f = \sum_{n} |f[n]|^2. \tag{2.3}$$

Además, otro concepto que nos será útil, la energía acumulativa de $f[n]$ se define como

$$Ec_f = \left[\frac{f[1]^2}{E_f}, \frac{f[1]^2 + f[2]^2}{E_f}, \frac{f[1]^2 + f[2]^2 + f[3]^2}{E_f}, \ldots, 1\right], \tag{2.4}$$

donde $E_f \neq 0$ y es definido por (2.3).

El *producto punto* o escalar en el espacio de $L^2(\mathbb{R})$ entre las funciones continuas f y g se define como

$$\langle f, g\rangle = \int_{-\infty}^{\infty} f(t)g^*(t)dt, \tag{2.5}$$

mientras que la misma operación para $l^2(\mathbb{Z})$ entre dos secuencias f y g se define como

$$\langle f, g\rangle = \sum_{n\in\mathbb{Z}} f[n]g^*[n]dt, \tag{2.6}$$

donde el simbolo " $*$ " significa el complejo conjugado de la función g. Sin embargo, en este trabajo se consideran sólo señales reales por lo que resulta que $g^* = g$. Otra ventaja de trabajar en estos espacios se debe a que las normas de las funciones $f(t)$ y $f[n]$, esta en términos del producto punto (2.5)-(2.6),

$$||f(t)|| = \sqrt{\langle f, f\rangle} = \sqrt{\int_{-\infty}^{\infty} |f(t)|^2 dt}, \tag{2.7}$$

$$||f[n]|| = \sqrt{\langle f, f\rangle} = \sqrt{\sum_{n\in\mathbb{Z}} |f[n]^2|}. \tag{2.8}$$

Otra operación, quizás la más, útil en el procesamiento continuo o discreto de señales resulta ser la convolución entre dos funciones, la cual se define como

$$\textbf{Caso Continuo} \qquad (f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau, \tag{2.9}$$

$$\textbf{Caso Discreto} \qquad (f * g)[n] = \sum_{m} f[m]g[n-m] \tag{2.10}$$

2.2 Teoría Ondeleta

Empezaremos discutiendo lo más elemental de la teoría ondeleta en tiempo continua, pero se hará más enfásis para el tiempo discreto.

Una función de energia finita, $\psi(t) \in L^2(\mathbb{R})$, se denomina ondeleta si cumple con las propiedades

1. ψ debe tener promedio cero

$$\int_{-\infty}^{\infty} \psi(t)dt = 0, \tag{2.11}$$

lo cual implica que es una función oscilatoria.

2. La función debe decaer con respecto al tiempo,

$$\lim_{t\to\infty} |\psi(t)| = 0. \tag{2.12}$$

Dependiendo de la aplicación en la que se utilice la función $\psi(t)$, debe de cumplir con otras propiedades. Al aplicar la operaciones de escalamiento y traslación a la función $\psi(t)$ se puede generar una familia de funciones definidas como

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right), \tag{2.13}$$

donde a es el parámetro real de escala que debe cumplir con $a > 0$, mientras que b corresponde al parámetro real de traslación. La función que cumple con (2.11) y genera una familia de (2.13) es llamada ondeleta madre, (figura 2.1) [8].

Dada una función de energía finita $f(t) \in L^2(\mathbb{R})$, podemos definir la **Transformada Ondeleta Continua**(TOC) de f como

$$W_{\psi}f(a,b) = \int_{-\infty}^{\infty} f(t)\psi^*_{a,b}(t)dt, \tag{2.14}$$

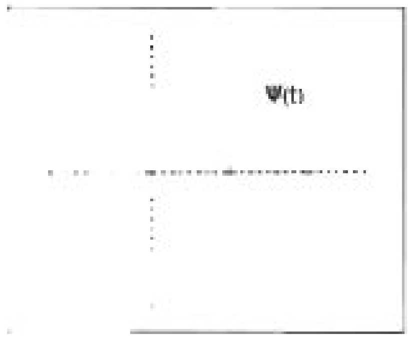


Figura 2.1 Representación de la ondeleta madre de Haar

donde "*" denota conjugación compleja. Cabe mencionar que la TOC se puede escribir en términos del producto escalar o de la convolución [13, 18], operaciones definidas en $L^2(\mathbb{R})$ y descritas en la Sección 2.1.

Además la TOC es una transformación reversible. Para reconstruir la función f, se considera la siguiente formula [8, 1]

$$f(t) = \frac{1}{C_\psi}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} W_\psi f(a,b)\psi_{a,b}(t)\frac{da\,db}{a^2}, \tag{2.15}$$

donde C_ψ es conocida como constante de admisibilidad, que depende de la función ondeleta que se utilice, y debe satisfacer

$$C_\psi = \int_{-\infty}^{\infty}\frac{|\hat{\psi}(\omega)|^2}{|\omega|}d\omega < \infty. \tag{2.16}$$

donde $\hat{\psi}(\omega) = \int_{-\infty}^{\infty} f(t)e^{-i\omega t}dt$, es la transformada de Fourier de $\psi(t)$. La función (2.16) implica que $\hat{\psi}(0) = 0$, es decir, cumple con (2.11). A este proceso de reconstrucción (2.15) se le conoce como síntesis de señal [8].

En la actualidad, la mayoría de los cálculos se realizan mediante computador. Por tanto, resulta útil discretizar la TOC (2.14). Una de las maneras o formas de lograr una eficiencia en el cálculo de la TOC es la de discretizar los parámetros de escala y traslación (a, b) de la ecuación (2.13) de la siguiente manera,

$$a = 2^{-j} \quad \text{y} \quad b = k2^{-j}, \tag{2.17}$$

donde $j, k \in \mathbb{Z}$. Así la familia de funciones ondeletas queda de la siguiente manera,

$$\psi_{j,k}(t) = 2^{j/2}\psi(2^j t - k), \tag{2.18}$$

que al sustituirla en la ecuación (2.14), tenemos que ahora los coeficientes son dados como

$$d_{j,k} = \int_{-\infty}^{\infty} f(t)\psi_{j,k}(t)dt. \tag{2.19}$$

Para la reconstrucción de la señal $f(t)$ a través de los coeficientes $d_{j,k}$ se tiene

$$f(t) = \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} d_{j,k}\psi_{j,k}(t). \tag{2.20}$$

El conjunto de ecuaciones (2.19) y (2.20) forman la llamada Serie de Ondeletas [8].

2.3 Análisis Multi-Resolución

Aunque la discretización de los parámetros de escala y traslación nos "facilita" el cálculo de la transformada ondeleta, no es realmente una transformada discreta, es decir, que las series de ondeletas son simplemente una versión muestreada de la TOC y que la información que aporta todavía resulta ser muy redundante. Esta redundancia, además, requiere una cantidad importante de tiempo y recursos de computación. Para afrontar estas dificultades, Mallat [18] propuso un algoritmo conocido como la Transformada Rápida Ondeleta (TRO), la cual está basada en el Análisis Multi-Resolución (AMR).

De manera general, el AMR utiliza un algoritmo para descomponer una señal $f(t)$ en elementos más simples, los cuales son llamados promedios y detalles, siendo los promedios donde se concentra la mayor información de la señal original $f(t)$ [18].

Después de que se le aplica a una señal $f(t)$ la TRO nos queda una aproximación de un promedio y varios detalles, como se muestra en la figura (2.2).

De manera más formal, un AMR consiste de una secuencia de subespacios cerrados anidados,

$$\ldots \subset V_{-1} \subset V_0 \subset V_1 \subset V_j \ldots$$

que cumple con las siguientes propiedades:

1. $\bigcup_{j\in\mathbb{Z}} V_j$ es denso en $L^2(\mathbb{R})$.
2. $\bigcap_{j\in\mathbb{Z}} V_j = 0$.
3. Invariabilidad en escala. Para cada $j \in \mathbb{Z}$, $f(t) \in V_j$ es equivalente a $f(2t) \in V_{j+1}$.

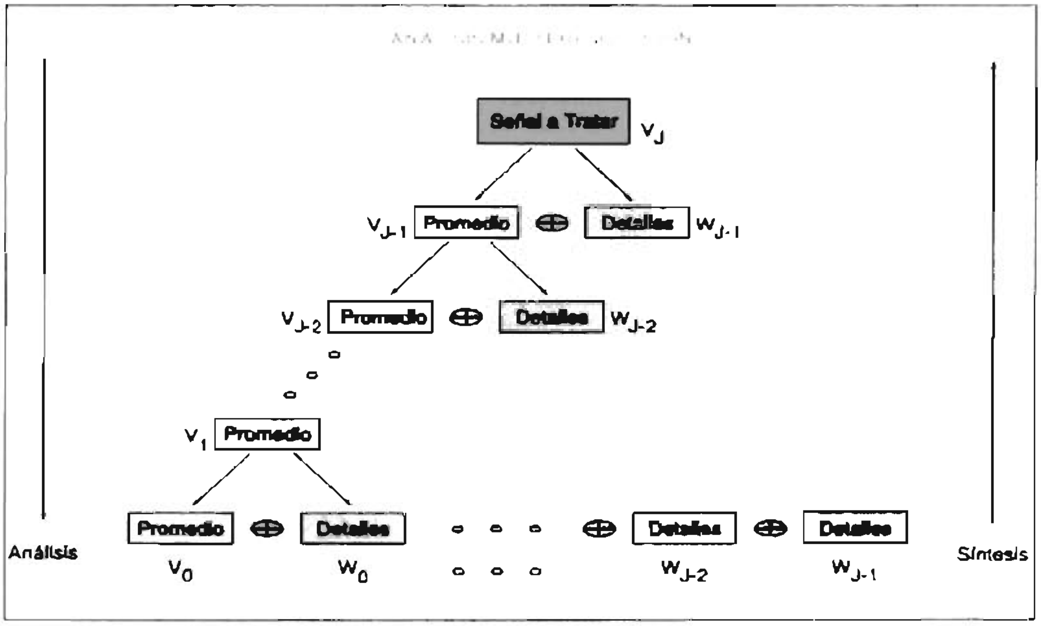


Figura 2.2: Representación del Análisis Multi-Resolución.

4. Invariabilidad bajo corrimiento. Para cada $f(t) \in V_0$ y cada $k \in \mathbb{Z}$, $f(t-k) \in V_0$

5. Existencia de una base. Existe $\varphi(t) \in V_0$ tal que $\varphi(t) = \varphi(t-k) : k \in \mathbb{Z}$ es una base para V_0, donde φ es llamada función escala.

La función de escala φ, mediante las operaciones de escalamiento y traslación, generan una base $\{\varphi_{j,k}(t) = 2^{j/2}\varphi(2^j t - k) : k \in \mathbb{Z}\}$ para V_j. Mientras que para la información que no se puede representar por medio de la función de escala $\varphi(t)$, se consideran la funciones ondeletas, $\psi(t)$, las cuales representan los detalles y generan los espacios denotados como los espacios W_j. De hecho, los espacios W_j son el complemento ortogonal de los espacios V_j, es decir, $V_{j+1} = V_j \bigoplus W_j$, donde $V_j \perp W_j$, y el símbolo $\bigoplus$ denota la operación de suma directa. En base a nuestra aplicación, se considerará la función ondeleta $\psi(t)$ para el caso cuando genera una base ortonormal $\{\psi_{j,k}(t) = 2^{j/2}\psi(2^j t - k) : k \in \mathbb{Z}\}$ para W_j [8, 18].

De lo anterior tenemos que la función $\varphi(t)$ es la que nos representa el promedio de nuestra información, mientras que la función $\psi(t)$ representará nuestro detalle. A continuación se da una representación clave en el AMR [18]. Debido a que $V_0 \subset V_1$, y $\varphi(t) \in V_0$, entonces $\varphi(t) \in V_1$, y también como $W_0 \subset V_1$ y $\psi(t) \in W_0$ entonces la la función de escala $\psi(t) \in V_1$, así , $\varphi(t)$, y la función ondeleta, $\psi(t)$, se pueden escribir como una combinación lineal

$$\varphi(t) = \sqrt{2}\sum_{k\in\mathbb{Z}} h[k]\varphi(2t-k), \tag{2.21}$$

$$\psi(t) = \sqrt{2}\sum_{k\in\mathbb{Z}} g[k]\varphi(2t-k), \tag{2.22}$$

donde los coeficientes $h[k]$ de $\varphi(t)$ y $g[k]$ de $\psi(t)$ están dados por

$$h[k] = \sqrt{2}\int_{-\infty}^{\infty} \varphi(t)\varphi(2t-k)dt. \tag{2.23}$$

$$g[k] = (-1)^k h[1-k] \tag{2.24}$$

y $h[k]$ cumple con

$$\sum_{k\in\mathbb{Z}} |h[k]|^2 = 1.$$

En base al AMR cualquier función $f(t) \in L^2(\mathbb{R})$ puede ser escrita mediante:

$$f(t) = \sum_{k=0}^{2^{j_0}-1} c_{j_0}[k]\varphi_{j_0,k}(t) + \sum_{j=j_0}^{J-1}\sum_{k=0}^{2^j-1} d_j[k]\psi_{j,k}(t), \tag{2.25}$$

donde los coeficientes son

$$c_j[k] = \int_{-\infty}^{\infty} f(t)\varphi_{j,k}(t)dt, \tag{2.26}$$

$$d_j[k] = \int_{-\infty}^{\infty} f(t)\psi_{k,j}(t)dt, \tag{2.27}$$

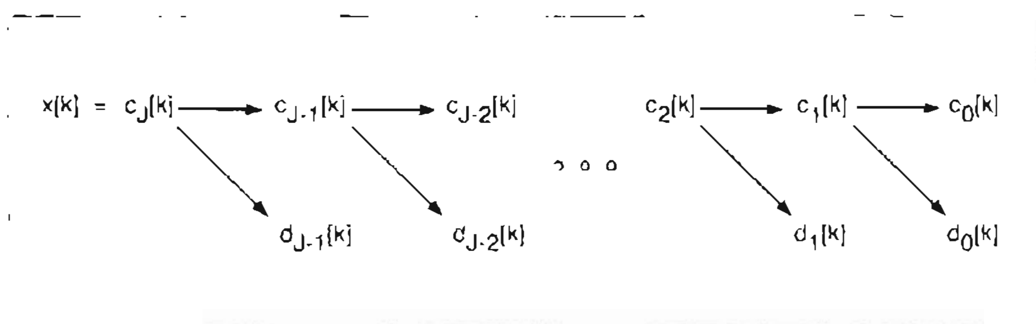


Figura 2.3: Representación Esquemática del Algoritmo de Mallat.

A la ecuación (2.25) se le conoce como la **Transformada Ondeleta Discreta (TOD)** de la función $f(t)$.

Por otra parte, la Transformada Ondeleta en sus versiones continua y discreta, tienen una relación de conservación de energía del tipo de Parseval. Para el caso discreto en el que las funciones $\varphi_{k,j}(t)$ y $\psi_{k,j}(t)$ formen bases ortonormales, la energía de f en (2.25) está dada como

$$E_f = \sum_{k=1}^{2^m} |c_m[k]|^2 + \sum_{j=j_0}^{J-1} \sum_{k=1}^{2^j} |d_j[k]|^2 \tag{2.28}$$

esto es, la energía de f está en términos de los coeficientes de la función escala y ondeleta. Tal propiedad hace atractiva tal transformación y será útil para el proceso de compresión, tema que se discutirá más adelante.

Para una implementación numérica eficiente de (2.26) y (2.27), se considera el Algoritmo de Mallat (figura 2.3), el cual se basa en

$$c_j[k] = \sum_{l=0}^{m-1} h[l-2k]c_{j+1}[l], \quad d_j[k] = \sum_{l=0}^{m-1} g[l-2k]c_{j+1}[l] \tag{2.29}$$

para el análisis de la señal a tratar, mientras que para la síntesis se considera

$$c_{j+1}[k] = \sum_{l \in \mathbb{Z}} (h[k-2l]c_j[l] + g[k-2l]d_j[l]). \tag{2.30}$$

Dicho algoritmo es conocido también como la **Transformada Rápida Ondeleta (TRO)**, que básicamente nos permite determinar de manera recursiva los coeficientes $c_j[k]$ y $d_j[k]$. Los términos $h[n]$ y $g[n]$ utilizados en (2.29) y (2.30) son típicamente llamados filtros pasa-bajas y pasa-altas de manera respectiva. Lo sorprendente de la TRO es que todo el algoritmo sólo depende de las dos secuencias $h[n]$ y $g[n]$, sin tener que conocer la función de escala ni la función ondeleta [21]. El algoritmo de la TRO utiliza el AMR y realiza la conexión entre las ondeletas y la teoría de bancos de filtros [21, 24].

La descripción general de la TRO se empieza al dividir o partir la señal en una versión de aproximación y una de detalles que juntas nos reproduce la señal original. Con la subdivisión se tiene que la información de la señal de aproximación contiene el contenido de las bajas frecuencias, mientras que la señal de detalles tiene las componentes de alta frecuencia. La aplicación repetitiva de dicho procedimiento sobre la señal de aproximación, los detalles de mayor resolución son separados y la aproximación se hace cada vez más burda. En términos de la teoría de filtros, (2.29) son el resultado de la operación de convolución discreta de $c_{j+1}[k]$ con los filtros $h[k]$ y $g[k]$ seguidos por la operación de "downsampling" con un factor de 2 [13, 18, 21]. El resultado del "downsampling" con un factor k a una señal, da las muestras de la señal cada k términos.

Mientras que la reconstrucción de los coeficientes correspondientes a (2.30), la inversa de la TRO, se puede considerar como la convolución discreta entre la señal $c_j[k]$, a la cual se le aplica la operación "upsampling" con un factor de 2, con los filtros $h[k]$ y $g[k]$.

El número de iteraciones que se puede aplicar la TRO está relacionado con la longitud de la señal. En la literatura se prefiere utilizar el término de niveles. Así, una señal con 2^j valores puede tener una descomposición de hasta $(j+1)$ niveles.

Para empezar la TRO consideramos la primera aplicación de (2.29) comenzando con $c_{j+1}[k] = X[k]$, donde la señal de tiempo discreto $X[k]$ a procesar, tiene como muestras $\{X[1], X[2], \ldots, X[J]\}$, con $J = 2^j$. Esto define el primer nivel de la TRO de X. El proceso se aplica de manera iterativa, considerando siempre a los "$j+1$" coeficientes de escala para calcular los "j" coeficientes de escala y de ondeleta. La J-ésima iteración de (2.29), la señal transformada consiste de J conjuntos de coeficientes de ondeleta en escalas de resolución $j = 1, \ldots, J$, y un sólo conjunto de la señal coeficientes de escala en la escala J. Al final se tienen exactamente $2^{(n-j)}$ coeficientes de ondeleta $d_j[k]$ en cada escala de resolución j, y $2^{(k-J)}$ coeficientes de escala $c_J[k]$. El número máximo de iteraciones es $J_{máx} = j$. De manera ilustrativa la figura 2.4 muestra el proceso de análisis del algoritmo de la TRO.

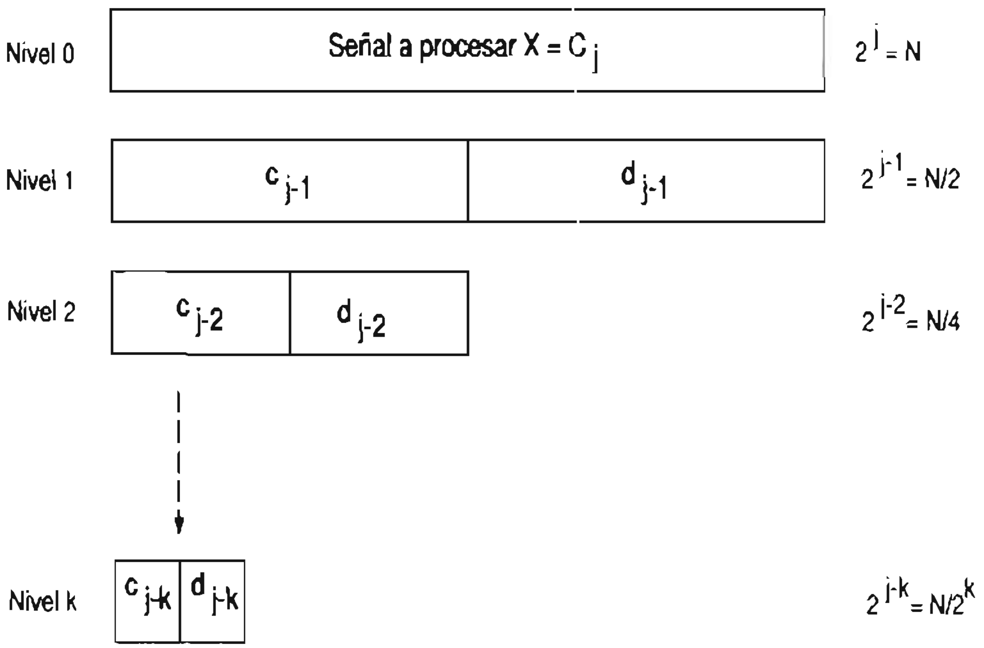


Figura 2.4: Ilustración de la TRO.

2.4 Ejemplo utilizando la Transformada Ondeleta de Haar (TOH)

Se hará un fácil ejemplo utilizando una secuencia de datos, los cuales se elegirán arbitrariamente para una mayor facilidad como puros números enteros, quedando la secuencia como a continuación se muestra:

$$f = [\mathbf{2} \quad \mathbf{4} \quad \mathbf{6} \quad \mathbf{8} \quad \mathbf{10} \quad \mathbf{12} \quad \mathbf{14} \quad \mathbf{16}] \tag{2.31}$$

La secuencia de (2.31) consta de ocho datos, por lo que $2^J = 8$ implica que $J = 3$, resultando que la secuencia de (2.31) se puede expresar en el espacio V_3.

La descomposición de (2.31) se puede hacer mediante (2.25), utilizando (2.29), para el uso particular de la TOH se tiene que los coeficientes $h[k]$ y $g[k]$ están definidos como se muestra a continucaicon,

$$h[k] = \begin{cases} \frac{1}{\sqrt{2}}, & k = 0; \\ 1, & k \neq 0. \end{cases} \tag{2.32}$$

$$g[k] = \begin{cases} \frac{1}{\sqrt{2}}, & k = 0; \\ -\frac{1}{\sqrt{2}}, & k = 1; \\ 0, & k \neq 0, 1. \end{cases} \tag{2.33}$$

Por la que la función de escala se escribe de la siguiente manera,

$$\varphi(t) = \begin{cases} 1, & 0 \leq t < 1; \\ 0, & \text{si no.} \end{cases} \tag{2.34}$$

Mientras que la función ondeleta es de la siguiente manera,

$$\psi(t) = \begin{cases} 1, & 0 \leq t < \frac{1}{2}; \\ -1, & \frac{1}{2} \leq t < 1; \\ 0, & \text{si no.} \end{cases} \tag{2.35}$$

De esto se tiene como resultado para V_2 y W_2 lo que se muestra en la figura 2.5.

De igual forma la descomposición para obtener los valores de V_1, W_1, V_0 y W_0, se muestra en la figura 2.6.

La secuencia (2.31) después de aplicar la TOH la tenemos de la siguiente manera,

$$\mathbf{25.455} \quad \mathbf{-11.313} \quad \mathbf{-4} \quad \mathbf{-4} \quad \mathbf{-1.414} \quad \mathbf{-1.414} \quad \mathbf{-1.414} \quad \mathbf{-1.414} \tag{2.36}$$

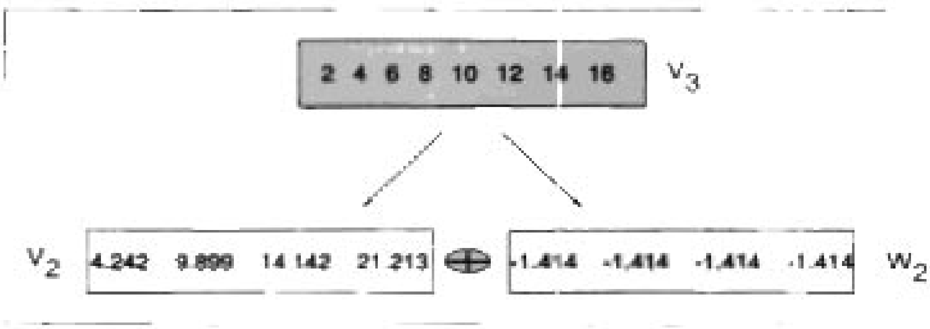


Figura 2.5: Espacios V_2 y W_2 para la secuencia (2.31).

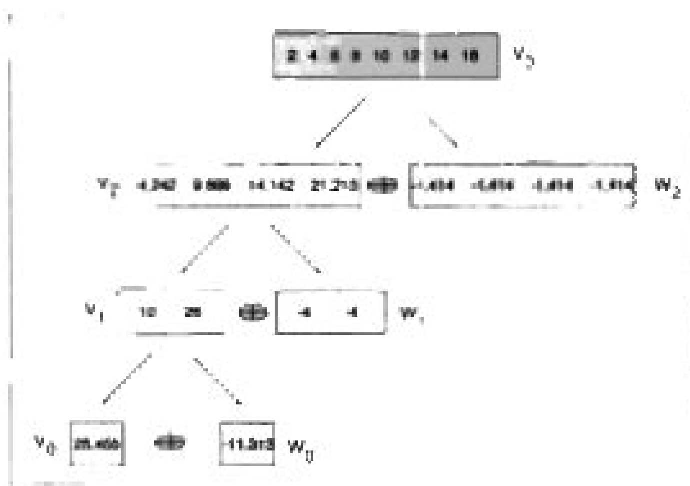


Figura 2.6 Aplicación del AMR con la ondeleta de Haar a la secuencia (2.31)

como se menciono antes, la Transformada Ondeleta tiene la característica de que conserva la energía de la señal a transformada, si calculamos la energía de la secuencia (2.31) tenemos lo siguiente,

$$E_1 = [(2)^2 + (4)^2 + (6)^2 + (8)^2 + (10)^2 + (12)^2 + (14)^2 + (16)^2]$$

$$E_1 = 816 \tag{2.37}$$

Y si también lo hacemos para la secuencia (2.36) tenemos lo siguiente.

$$E_2 = [(25.455)^2 + (-11.313)^2 + (-4)^2 + (-4)^2 + (-1.414)^2 + (-1.414)^2 + (-1.414)^2 + (-1.414)^2$$

$$E_2 = 816 \tag{2.38}$$

De esto tenemos que la energía en (2.37) y (2.38) son iguales.

Para la reconstrucción de la secuencia a partir de (2.36) utilizaremos (2.30), de lo cual se llegara a V_1 utilizando V_0 y W_0 así sucesivamente hasta llegar al nivel de V_3 mediante V_2 y W_2

2.5 Esquema de compresión de información con la TO

Aunque existen diferentes maneras de utilizar la TO como herramienta para realizar la compresión de información, en este trabajo se considera el esquema empleado en [8], el cual se basa en determinar el valor de umbral en términos de la energía. Por lo que se describirá básicamente la manera de determinar el calor del umbral.

Determinar el Umbral:

Considerando los coeficientes ondeleta $\mathcal{X}$ y mediante (2.3) se calcula la energía de $\mathcal{X}$, $E_{\mathcal{X}}$. De hecho, por la propiedad de conservación de energía podríamos considerar la energía de la señal a procesar E_f en vez de $E_{\mathcal{X}}$.

Se forma una nueva secuencia $\mathcal{X}_o[k]$ al ordenar las magnitudes de los coeficientes ondeleta en orden descendiente, es decir,

$$M_1 \geq M_2 \geq M_3 \ldots \geq M_N \tag{2.39}$$

donde M_i, $i = 1, \ldots, N$, son los valores absolutos de los coeficientes ondeleta. Obviamente M_1 es el valor absoluto más grande de $\mathcal{X}$, M_2 es el siguiente más grande, y así sucesivamente. Por lo que tenemos

$$\mathcal{X}_o = [M_1, M_2, \ldots, M_N]. \tag{2.40}$$

Ahora se calcula la energía acumulativa (2.4) de $\mathcal{X}_o$:

$$E_{\text{cum}} = \left[\frac{M_1^2}{E_{\mathcal{X}}}, \frac{M_1^2 + M_2^2}{E_{\mathcal{X}}}, \frac{M_1^2 + M_2^2 + M_3^2}{E_{\mathcal{X}}}, \ldots, 1\right]. \tag{2.41}$$

A la ecuación (2.41) la denominaremos como *perfil de energía* y a su gráfica *mapa de energía*. Se podrá observar que en el mapa de energía se alcanza rápidamente su valor máximo de 1, el cual equivale al 100% de energía. En muchas ocasiones el mapa de energía es útil para determinar aproximadamente la cantidad de coeficientes que se tomarán en cuenta, dependiendo de la energía que se considere en la aplicación [8]. Teniendo seleccionado el porcentaje de energía, el umbral se obtiene al determinar la posición del primer coeficiente para el cual el perfil de energía obtiene tal porcentaje. En sí, el valor del umbral corresponde al valor de tal coeficiente. Por ejemplo, si se esta interesado en conservar el 95 % de la energía de una señal dada y con su perfil de energía correspondiente se tiene

$$\frac{M_1^2 + M_2^2 + \cdots + M_{70}^2}{E_X} = 0.9501,$$

entonces, el umbral tendría que ser igual al valor de M_{70}. De ahí que se considere el umbral en términos de la energía.

Teniendo el valor del umbral se procede a igualar a cero los coeficientes de la señal transformada que sean menor a dicho valor. De lo anterior se tendrá una señal con demasiados ceros y resulta práctico tener dos vectores, uno de ceros y unos, que da información de las posiciones con la misma longitud de la señal de entrada pero fácil de procesar, y otro de coeficientes con una longitud mucho menor a la señal de entrada. De manera general tendríamos el siguiente esquema de compresión basado en la transformada ondeleta, y se ilustra en la figura 2.7.

Procedimiento para compresión basado en la TO

1. Dada una señal de información, evaluar numéricamente la TRO con la ondeleta de Haar.
2. Determinar el umbral ε seleccionado en base a el perfil de la energía y hacer cero los valores de los coeficientes ondeleta que sean menor que ε.
3. Codificar los coeficientes ondeleta.
4. Transmisión de información.
5. Aplicar la transformación inversa a la información recibida.

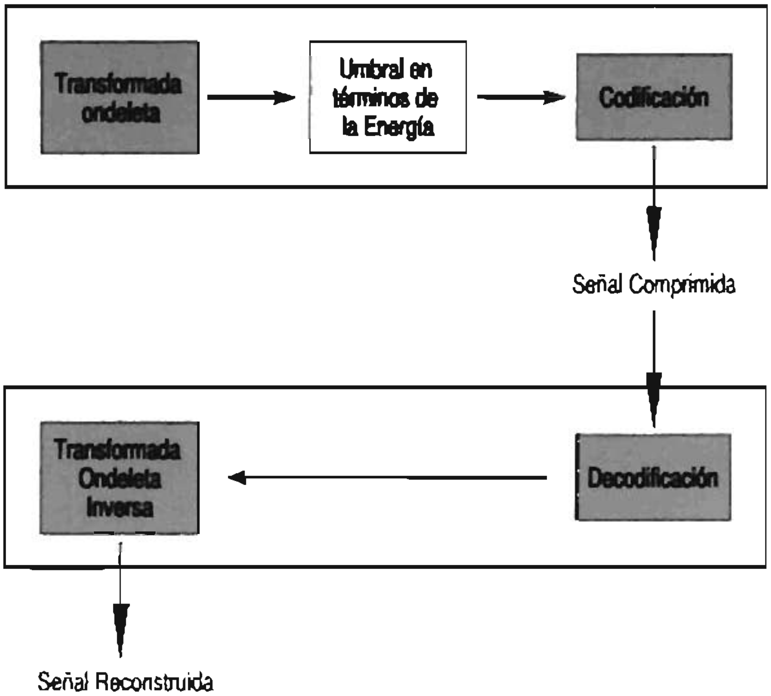


Figura 2.7: Esquema de compresión basado en la transformada ondeleta.

3

Criptología

Desde que el hombre ha tenido la necesidad de comunicarse con los demás, ha querido que solo cierto tipo de personas puedan leer o entender los mensajes que se envían, este tipo de necesidad hizo que se pusieran códigos a estos mensajes, a este proceso se le conoce como encriptar o cifrar. Y para poder leer los mensajes se tenia que seguir un proceso de descifrado.

Los sistemas de cifrado se fueron modificando, pasando de ser un arte a ser toda una ciencia. La criptología es, por tanto, un requisito esencial en la sociedad de hoy en día.

3.1 Introducción

La palabra Criptología agrupa la *Criptografía* y el *Criptoanálisis* [12, 11].

La criptografía hace referencia al uso de códigos para ocultar, enmascarar o transformar algún tipo de información, mientras que la palabra criptoanálisis engloba a las técnicas que se usan para romper esos códigos, ambas técnicas están íntimamente ligadas.

Definiremos la palabra de *Texto Claro* o *Texto Plano* a la información original con la que se cuente (ya sea texto, audio, etc.).

Las palabras *encriptar, encriptación, cifrar y codificar* se refieren al hecho de transformar el *Texto Plano* en *Texto Cifrado* o *Texto Encriptado*, esto es, cualquier persona puede ver la información contenida en el *Texto Cifrado*, pero no la podrá entender o relacionar con algún tipo de dato.

Mientras que usaremos las palabras *desencriptar, descifrar, decodificar* para referirnos al hecho de transformar el *Texto Cifrado* en *Texto Plano*.

3.2 Criptografía

La palabra criptografía proviene del griego *κρυπτς*, que significa oculto y *γραφειν*, escribir, es decir, escritura escondida o Arte de escribir con clave secreta [12, 11].

La criptografía se puede clasificar en criptografía clásica y criptografía moderna [12, 11]. La criptografía clásica abarca desde tiempos inmemoriales hasta los años de la posguerra, es decir, hasta la mitad del siglo XX. La criptografía moderna nace al mismo tiempo que las computadoras, es decir, durante la segunda guerra mundial.

Cuando hablamos de criptografía clásica nos referimos a esta por las técnicas utilizadas, prácticamente operaciones de sustitución y transposición de caracteres, dentro de la gran cantidad de cifradores existentes dentro de la criptografía clásica podemos dividirlos en diferentes tipos: *Cifradores Monoalfabéticos, Cifradores Polialfabéticos, Cifradores por Sustitución Homofónica* y *Cifradores de Transposición*. En la criptografía moderna además de utilizar la sustitución y transposición de sus caracteres se hace uso de las propiedades matemáticas.

3.2.1 Criptografía Moderna

La criptografía moderna se puede clasificar *según el tratamiento del mensaje* y *según el tipo de claves*.

Según el Tratamiento del Mensaje: Tenemos cifradores de bloques, en donde se divide el texto plano en n bloques de igual longitud, la longitud del bloque esta previamente definida en los algoritmos de cifrado (figura 3.1).

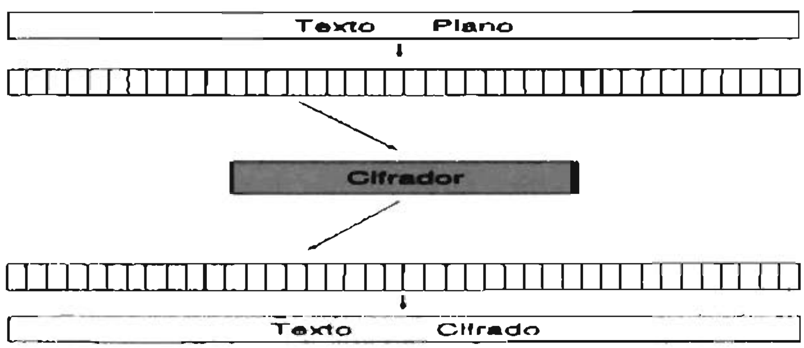


Figura 3.1: Cifrador de bloques.

También se tiene el cifrador en flujo, este tipo de cifradores puede cifrar bit a bit, habitualmente esta operación es un simple xor-exclusivo (figura 3.2), este tipo de cifrado es mas rápido que el cifrado de bloques.

Según el Tipo de Claves: Se clasifican en Criptosistemas Simétricos y Asimétricos. Los

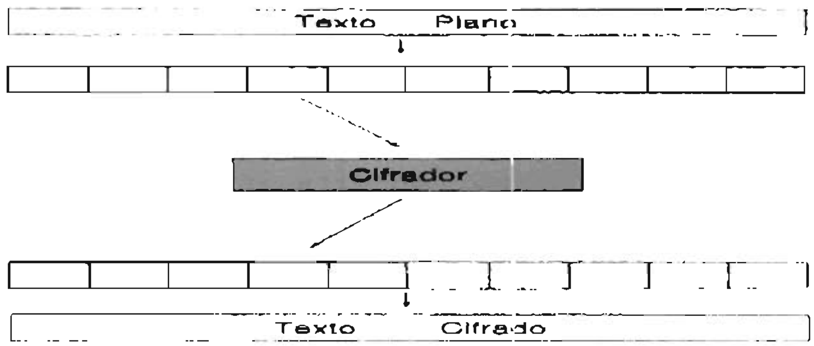


Figura 3.2: Cifrador de flujo.

Criptosistemas Simétricos, conocidos también como de clave o llave privada, constan una sola llave, que es usada tanto para encriptar como para desencriptar.

Los Criptosistemas Asimétricos, se les conoce también como criptosistemas de clave publica. Constan de dos llaves, una para cifrar que se llama clave pública y una para descifrar que es la clave privada [3]. A diferencia de los criptosistemas simétricos, la llave publica se puede transmitir por un medio no seguro. Se debe estar seguro de que mediante la llave publica no se pueda obtener o adivinar la llave privada.

Los criptosistemas más modernos se basan en la seguridad computacional. La idea es considerar un criptosistema computacionalmente seguro que aun cuando haya un algoritmo que rompa el sistema, éste requiera un tiempo de computación tan grande que sea inviable llevarlo a la práctica.

3.3 Criptoanálisis

El criptoanálisis consiste en comprometer la seguridad de un criptosistema, esto es tratar de desencriptar un documento encriptado sin conocer la llave.

El criptoanálisis se lleva acabo estudiando una gran cantidad de mensajes de texto plano-texto encriptado. Obviamente, mientras mayor sea el numero de mensajes estudiados, mayor probabilidad de éxito tendrá el criptoanálisis.

También se puede tratar de criptoanalizar un sistema aplicando el algoritmo de descifrado con cada una de las posibles claves, a este método se le denomina *ataques por la fuerza bruta*

3.4 Criptografía Basada en Autómatas Celulares (AC)

Un tipo de sistemas de encriptación, son los llamados criptosistemas iterados, en donde una transformación criptográficamente débil es aplicada repetidamente a un mensaje, resultando una transformación fuerte, como el caso del sistema de encriptación DES [16]. Este tipo de criptosistemas se acercan mucho a los sistemas dinámicos.

El estado futuro de un sistema dinámico depende sensiblemente de su estado inicial. Ya que el sistema es determinista, la misma trayectoria siempre será trazada de las mismas condiciones iniciales. Entonces, la llave de un criptosistema pudiera ser el estado inicial de un conocido sistema dinámico. Una colección de usuarios que comparten esta llave secreta, pueden mandar mensajes secretos entre ellos combinando sus mensajes con alguna parte de la trayectoria trazada por el estado inicial secreto bajo la acción del sistema dinámico.

Usando sistemas dinámicos reversibles un mensaje puede ser encriptado codificándolo como un estado del sistema y entonces se hace correr el sistema en el tiempo hacia adelante, durante un cierto tiempo. El estado resultante es el texto cifrado. Para desencriptar el texto cifrado el sistema es iterado inversamente el mismo número de pasos en el tiempo que fueron usados en la encriptación, recobrando el texto franco como un estado del sistema.

3.4.1 Autómatas Celulares

Los autómatas celulares son sistemas dinámicos definidos de manera completamente discreta, tanto espacial como temporalmente.

- El espacio de estados de un autómata celular es el conjunto $\mathbb{Z}_K^{\mathbb{Z}}$, donde $\mathbb{Z}$ denota el conjunto de los números enteros.
- Un estado del autómata celular es representado por una secuencia doblemente infinita de valores tomados de $\mathbb{Z}_K = \{0, 1, \ldots K - 1\}$. Para *autómatas celulares binarios* $K = 2$ y denotamos su espacio de estados como $\Omega = \mathbb{Z}_2^{\mathbb{Z}}$. Por ejemplo.

$$\begin{array}{rcccccccccl} x = (\ldots, & 1, & 0, & 0, & 1, & 0, & 0, & 1, & 1, & \ldots) \\ \mathbb{Z} = \{\ldots, -4, & -3, & -2, & -1, & 0, & 1, & 2, & 3, & \ldots\} \end{array} \tag{3.1}$$

- La i-ésima coordenada de un elemento $x \in \Omega$ es denotada por $x_i \in \mathbb{Z}_2 = \{0, 1\}, i \in \mathbb{Z}$.
- La evolución es definida por la iteración repetida de un operador de evolución $\mathcal{A} : \mathbb{Z}_K^{\mathbb{Z}} \to \mathbb{Z}_K^{\mathbb{Z}}$.
- La acción de $\mathcal{A}$ sobre un estado del autómata x se especifica mediante una regla local $\mathcal{A}_{\mathcal{L}} : \mathbb{Z}_2^3 \to \mathbb{Z}_2$, tal que $(\mathcal{A}(\underline{x}^t))_i = \mathcal{A}_{\mathcal{L}}(x_{i-1}^t, x_i^t, x_{i+1}^t)$.

  Los algoritmos de encriptación utilizados en la tesis se basan en el autómata celular con la regla local
  $$\mathcal{A}_{\mathcal{L}}(x_{i-1}, x_i, x_{i+1}) = x_{i-1} + x_{i+1} \in \mathbb{Z}_2.$$

  La figura 3.3 muestra la regla local donde las coordenadas son representadas por circulos y el valor de cada coordenada es la suma módulo 2 de los valores presentados por las flechas entrantes al círculo. Y las flechas salientes representan el valor tomado por la coordenada, en la figura 3.3 el valor de la coordenada $(i, t + 1)$ es $x_i^{t+1} = x_{i-1}^t + x_{i+1}^t$.

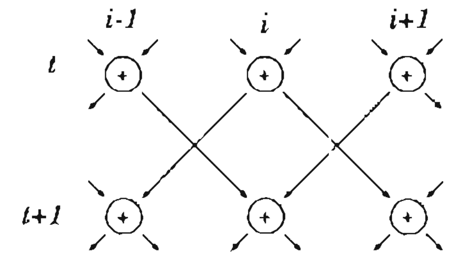


Figura 3.3: Regla local $A_L(x_{i-1}, x_i, x_{i+1}) = x_{i-1} + x_{i+1} \in Z_2, Z_2 = \{0, 1\}$.

Métodos y sistemas de encriptación y desencriptación que utilizan autómatas celulares han sido realizados y patentados [4, 10, 19, 5].

3.4.2 Sincronización en Automatas celulares

Dos sistemas dinámicos acoplados sincronizan si, después de un largo periodo de tiempo, sus comportamientos consiguen estar arbitrariamente cerca Esto es, en cada paso de tiempo ambos sistemas evolucionan de acuerdo a la misma regla.

En el caso de autómatas celulares existe la sincronización como el resultado de un acoplamiento no trivial[23]. El acoplamiento en autómatas celulares sucede cuando un conjunto determinado de coordenadas (*coordenadas acopladas*) es copiada de uno de los sistemas el cual es el *autómata celular manejador*, al sistema de respuesta que será el *autómata celular de respuesta*. Esto es, en cada paso de tiempo ambos sistemas evolucionan de acuerdo a la misma regla, y las coordenadas acopladas del autómata celular manejador son copiadas a las correspondientes coordenadas del autómata celular de respuesta. En la figura 3.4 se muestra un ejemplo que ilustra el acoplamiento unidireccional en autómatas celulares.

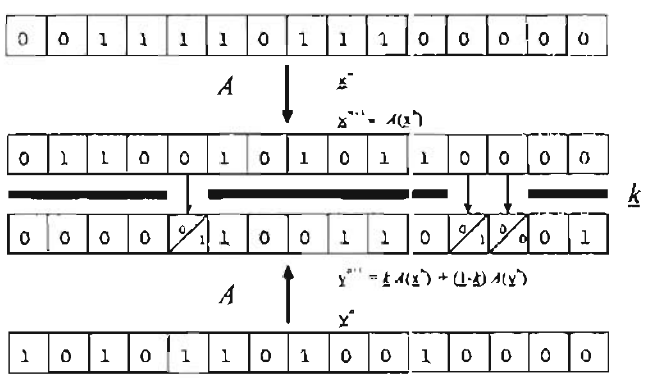


Figura 3.4: Ejemplo de acoplamiento unidireccional

En el ejemplo de la figura 3.4 la secuencia de acoplamiento es,

$$\underline{\kappa} = (0,0,0,0,1,0,0,0,0,0,0,1,1,0,0)$$

y las coordenadas acopladas son κ_4, κ_{11} y κ_{12}. Los estados en el tiempo t son $\underline{x}^t$ y $\underline{y}^t$,

$$\begin{aligned}\underline{x}^t &= (0,0,1,1,1,1,0,1,1,1,0,0,0,0,0)\\ \underline{y}^t &= (1,0,1,0,1,1,0,1,0,0,1,0,0,0,0)\end{aligned}$$

entonces el estado del autómata manejador en el tiempo $t+1$ es

$$\underline{x}^{t+1} = \mathcal{A}(\underline{x}^t) = (0,1,1,0,0,1,0,1,0,1,1,0,0,0,0)$$

y el estado del autómata respuesta en el tiempo $t+1$ es obtenido de la siguiente manera

$$\begin{aligned}\mathcal{A}(\underline{y}^t) &= (0,0,0,0,1,1,0,0,1,1,0,1,0,0,1)\\ \underline{\kappa} &= (0,0,0,0,1,0,0,0,0,0,0,1,1,0,0)\\ 1-\underline{\kappa} &= (1,1,1,1,0,1,1,1,1,1,1,0,0,1,1)\\ \underline{\kappa}\mathcal{A}(\underline{x}^t) &= (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)\\ (1-\underline{k})\mathcal{A}(\underline{y}^t) &= (0,0,0,0,0,1,0,0,1,1,0,0,0,0,1)\\ (\underline{1}-\underline{\kappa})\mathcal{A}(\underline{y}^t)+\underline{\kappa}\mathcal{A}(\underline{x}^t) &= (0,0,0,0,0,1,0,0,1,1,0,0,0,0,1) \qquad (3.2)\end{aligned}$$

Un par acoplado $(\mathcal{A},\underline{\kappa})$ sincroniza cuando la diferencia $\underline{z}^t = \underline{y}^t - \underline{x}^t$ de los vectores de estado $\underline{x}^t$, $\underline{y}^t$ (correspondientes al autómata manejador y al autómata respuesta, respectivamente) iguala al vector nulo $\underline{0} = (\ldots,0,0,0,\ldots)$ después de un cierto número de pasos en el tiempo t.

3.5 Sistema de Encriptación ESAC

El sistema de Encriptación ESAC, es un sistema de encriptación basado en la sincronización en Autómatas Celulares el cual fue realizado en [14].

El fenómeno de sincronización en autómatas celulares es usado para construir 2 familias de permutaciones, Ψ y Φ, de palabras binarias de longitud finita de 2^k-1, $k\in\mathbb{Z}$. Estas permutaciones son usadas para encriptar y desencriptar N bloques de longitud 2^k-1, $k\in\mathbb{Z}$.

Este fenómeno de sincronización también permite la construcción de una función llamada *función h* la cual se utiliza para la generación de llaves.

3.5.1 Encriptación

Una de las familias de permutaciones la cual logra la encriptación esta definida como:

$$\Psi = \psi_{\mathrm{x}} : M \longrightarrow C \mid \mathrm{x} \in X, \tag{3.3}$$

siendo cada uno de los tres conjuntos, $M = C = X = 2^k - 1, k \in \mathbb{Z}$

Las palabras M son los bloques antes de encriptar (texto plano), las palabras C son los bloques después de encriptarlos (texto encriptado) y las palabras X son los bloques llamados llaves, utilizados para encriptar.

Para cada $\mathrm{x} \in X$ existe una $\mathrm{m} \in M$ tal que tiene una expresión dada por $\psi_{\mathrm{x}}(\mathrm{m})$

3.5.2 Desencriptación

La familia de permutaciones inversa a (3.3), la cual lograra desencriptar el texto encriptado esta dada por

$$\Phi = \phi_{\mathbf{x}} : C \longrightarrow M \mid \mathbf{x} \in X, \tag{3.4}$$

tal que para toda $\mathbf{x} \in X$, $\mathrm{m} = \phi_{\mathbf{x}}(\psi_{\mathbf{x}}(\mathrm{m}))$ para toda m.

3.5.3 Unidad Encriptadora Básica

En [23] se muestra que un par de autómatas celulares lineales elementales con la regla local $\mathcal{A}_{\mathcal{L}}(x_{-1}, x_0, x_1) = x_{-1} + x_1 \pmod 2$ sincroniza si cada par de coordenadas consecutivas acopladas están separadas por un bloque de $2^k - 1$ sitios desacoplados. Utilizando esto se define en [14] una Unidad Encriptadora Básica a partir de la cual se construyen las permutaciones ψ y ϕ y la función h. A continuación se describe como se obtienen cada uno de ellas.

Considere una secuencia inicial infinita:

$$\underline{x}^0 = (\ldots, x_{-1}^0, x_0^0, x_1^0, x_2^0, \ldots, x_i^0, \ldots, x_{N-1}^0, x_N^0, x_{N+1}^0, x_{N+2}^0 \ldots)$$

que evoluciona del tiempo $t = 0$ al tiempo $t = N = 2^k - 1$, siguiendo la regla local $\mathcal{A}_{\mathcal{L}}(x_{i-1}, x_i, x_{i+1}) = x_{i-1} + x_{i+1}$ para las coordenadas $i \neq 0$ e $i \neq N+1$. Las coordenadas 0 y $N+1$ son acopladas, es decir que los valores x_0^t y x_{N+1}^t son asignados externamente a cada tiempo t. Esta evolución genera la configuración espacio temporal del *autómata celular de respuesta*, la cual es mostrada en la figura 3.5 .

En la evolución de la figura 3.5 se muestra un cuadro, formado con líneas punteadas, a partir de las coordenadas acopladas x_0^0 y x_{N+1}^0. Este cuadro se define como la Unidad Encriptadora mediante la cual se definen las permutaciones ψ y ϕ y la función h. Dentro de la unidad encriptadora se identifican las palabras $\mathrm{x}, \mathrm{y}, \mathrm{m}, \mathrm{c}$ y t.

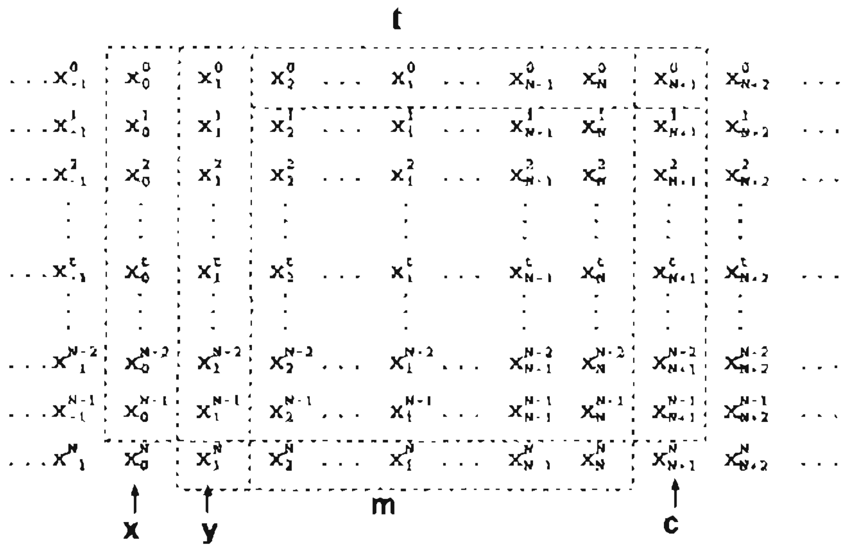


Figura 3.5: Evolución de una secuencia infinita donde las coordenadas $i = 0$ e $i = N + 1$ están dadas externamente.

- **Permutación** $\mathbf{m} = \phi_{\mathbf{x}}(\mathbf{c})$

  La palabra m, que se identifica como un bloque de texto franco, es la palabra $\mathbf{m} = (x_1^N, x_2^N, ..., x_i^N, ..., x_N^N)$, la cual se encuentra en la parte inferior del bloque básico en la figura 3.5.

  La forma evidente de generar la permutación $\mathbf{m} = \phi_{\mathbf{x}}(\mathbf{c})$, es haciendo evolucionar el autómata hacia adelante en el tiempo, usando como entradas las palabras c y x, ver figura 3.3. En [23, 15, 22] se demuestra que m solo depende de x y de c. La independencia en t es resultado de la sincronización.

- **Permutación** $\mathbf{c} = \psi_{\mathbf{x}}(\mathbf{m})$

  La palabra c, la cual es identificada como el bloque encriptado, es la palabra $\mathbf{c} = (x_{N+1}^0, x_{N+1}^1, ..., x_{N+1}^n, ..., x_{N+1}^{N-1})$ situada al lado derecho en el bloque básico mostrado en la figura 3.5. Para desencriptar este bloque se utiliza la permutación inversa ϕ de manera que se tiene $\mathbf{m} = \phi_{\mathbf{x}}(\mathbf{c}) = \phi_{\mathbf{x}}(\psi_{\mathbf{x}}(\mathbf{m}))$.

  Para generar la permutación $\mathbf{c} = \psi_{\mathbf{x}}(\mathbf{m})$, se hace evolucionar el autómata hacia atrás en el tiempo, ver figura 3.6, utilizando como entradas las palabras x y m. Operativamente es necesario proporcionar algún valor para y pero no afecta el valor de $\mathbf{c} = \psi_{\mathbf{x}}(\mathbf{m})$. De igual manera esta independencia es resultado de la sincronización, ver [22, 15, 23].

- **Función** $\mathbf{t} = h(\mathbf{x}, \mathbf{y})$

  En la parte izquierda de la unidad encriptadora mostrada en la figura 3.5 se encuentran las dos partes de la semilla x y y, utilizadas por la función $\mathbf{t} = h(\mathbf{x}, \mathbf{y})$. Estas palabras son $\mathbf{x} = (x_0^0, x_0^1, ..., x_0^n, ...x_0^{N-1})$ y $\mathbf{y} = (x_1^0, x_1^1, ..., x_1^n, ...x_1^{N-1}, x_1^N)$. En la parte superior

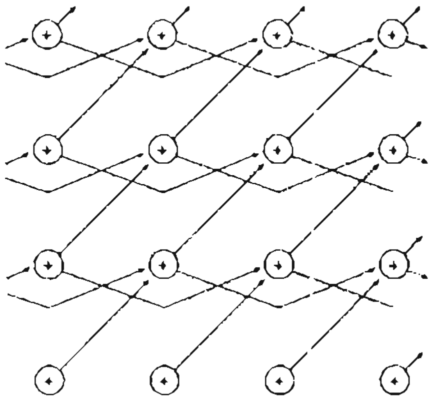


Figura 3.6: Ejecución del autómata hacia atrás

está la palabra t que es el resultado de la función h, esta palabra es identificada como $\mathbf{t} = (x_2^0, x_3^0, x_4^0, \ldots, x_i^0, \ldots, x_{N+1}^0)$.

Para generar esta función h, se hace evolucionar el autómata hacia atrás, ver figura 3.7, utilizando como entradas las palabras $\mathbf{x}$ y $\mathbf{y}$.

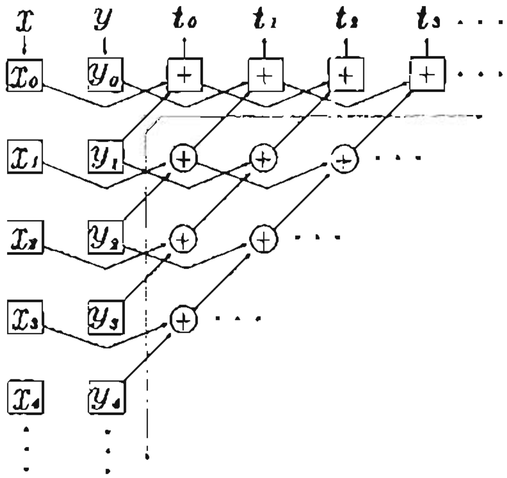


Figura 3.7: Esquina superior izquierda de la unidad básica

El sistema más simple que uno puede idear para encriptar un texto largo, es dividirlo en bloques $\mathbf{m}^0, \mathbf{m}^1, \mathbf{m}^2, \ldots$ que son transformados consecutivamente a $\psi_{\mathbf{x}^0}(\mathbf{m}^0), \psi_{\mathbf{x}^1}(\mathbf{m}^1), \psi_{\mathbf{x}^2}(\mathbf{m}^2), \ldots$, siguiendo la secuencia de llaves $\mathbf{x}^0, \mathbf{x}^1, \mathbf{x}^2, \ldots$ generadas por un generador pseudoaleatorio.

Para reconstruir el texto plano a partir del texto encriptado, se necesita conocer la semilla que fue usada para generar la secuencia de llaves pseudoaleatoria y tener acceso a la familia de permutaciones inversas $\Phi = \{\phi_{\mathrm{x}} : C \to M \mid \mathrm{x} \in X\}$ tal que para toda $\mathrm{x} \in X$, $\mathrm{m} = \phi_{\mathrm{x}}(\psi_{\mathrm{x}}(\mathrm{m}))$ para toda m.

3.5.4 Funciones de un solo tiempo

Para las permutaciones ϕ (3.3), ψ (3.4) y la *función h* se hace uso de las leyes y teoremas del álgebra booleanas para reducir estas funciones a un algoritmo de un solo tiempo de manera que cada ciclo de reloj se ejecute una función completa, y no una vez la regla del automata.

La palabra m corresponde a la permutación $\mathrm{m} = \phi_{\mathrm{x}}(\mathrm{c})$, la cual se genera haciendo evolucionar el autómata hacia adelante. En la figura 3.8 se muestra como se obtiene el bit $m1$ de la palabra m de 7 bits.

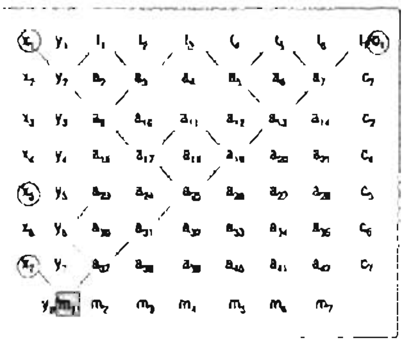


Figura 3.8: Reducción del bit $m1$ de M.

En la ecuación (3.5) se muestra el valor del bit $m1$ al pasar por todas la compuertas xor y su reducción utilizando las reglas del algebra booleana.

$$\begin{aligned} m1 &= x7 \oplus a37 = x7 \oplus y6 \oplus a31 = x7 \oplus x5 \oplus a23 \oplus a23 \oplus a25 \\ &= x7 \oplus x5 \oplus a25 = x7 \oplus x5 \oplus a17 \oplus a19 = x7 \oplus x5 \oplus a9 \oplus a13 \\ &= x7 \oplus x5 \oplus y2 \oplus a3 \oplus a5 \oplus a7 \\ &= x7 \oplus x5 \oplus x1 \oplus t1 \oplus t3 \oplus t3 \oplus t5 \oplus t5 \oplus c1 \\ m1 &= x7 \oplus x5 \oplus x1 \oplus c1 \end{aligned} \tag{3.5}$$

Este mismo procedimiento se aplicaria para cada uno de los bits de palabras de tamaño $2^k - 1$. En el apéndice A se muestran las ecuaciones reducidas de los bits de las palabras m y c de tamaño de 15 bits.

En la figura 3.9 se muestra como obtener el bit $t3$ de la palabra t de tamaño de 15 bits. Como se comentó en la sección anterior el valor de la palabra t se obtiene haciendo evolucionar el autómata hacia atras, utilizando los valores de las palabras x y y.

En (3.6) se muestra el valor del bit t_3 al pasar por todas las compuertas xor, y su reducción utilizando las reglas de algebra booleana.

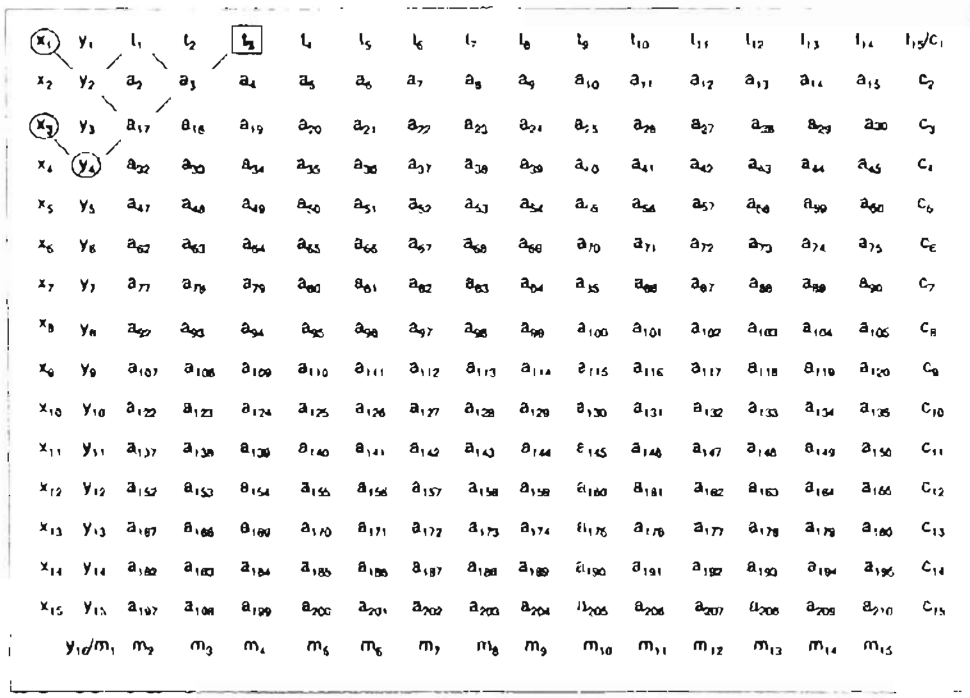


Figura 3.9: Se hace evolucionar el automata para obtener $t3$ de la palabra t.

$$\begin{aligned} t3 &= t1 \oplus a3 \\ &= x1 \oplus y2 \oplus y2 \oplus a17 \\ &= x1 \oplus a17 \\ &= x1 \oplus x3 \oplus y4 \end{aligned} \tag{3.6}$$

Haciendo el mismo procedimiento para cada bit de la palabra t, se obtienen las ecuaciones de un solo tiempo de la función h las cuales se muestran a continuación.

$$t1 = x1 \oplus y2$$

$$t2 = x2 \oplus y1 \oplus y3$$

$$t3 = x1 \oplus x3 \oplus y4$$

$$t4 = x4 \oplus y1 \oplus y3 \oplus y5$$

$$t5 = x5 \oplus x3 \oplus x1 \oplus y2 \oplus y6$$

$$t6 = x6 \oplus x2 \oplus y1 \oplus y5 \oplus y7$$

$$t7 = x7 \oplus x5 \oplus x1 \oplus y8$$

$$t8 = x8 \oplus y1 \oplus y5 \oplus y7 \oplus y9$$

$$
\begin{aligned}
t9 &= x9 \oplus x7 \oplus x5 \oplus x1 \oplus y2 \oplus y6 \oplus y10 \\
t10 &= x10 \oplus x6 \oplus x2 \oplus y1 \oplus y3 \oplus y5 \oplus y9 \oplus y11 \\
t11 &= x11 \oplus x9 \oplus x5 \oplus x3 \oplus x1 \oplus y4 \oplus y12 \\
t12 &= x12 \oplus x4 \oplus y1 \oplus y3 \oplus y9 \oplus y11 \oplus y13 \\
t13 &= x13 \oplus x11 \oplus x9 \oplus x3 \oplus x1 \oplus y2 \oplus y10 \oplus y14 \\
t14 &= x14 \oplus x10 \oplus x2 \oplus y1 \oplus y9 \oplus y13 \oplus y15 \\
t15 &= x15 \oplus x13 \oplus x9 \oplus x1 \oplus y16
\end{aligned}
$$

3.5.5 Generador Pseudoaleatorio de Llaves

Para poder encriptar nuestro texto plano es necesario contar con un generador de llaves lo suficientemente seguro. El generador utilizado en esta tesis fue probado en [2] utilizando las pruebas de la NIST [20] resultando ser un generador confiable para usos en criptografía.

El generador hace uso de la función $t = h(x, y)$ para generar las llaves. Las llaves generadas son bloques de tamaño de 15 bits, las cuales se utilizaran para encriptar y desencriptar utilizando las funciones ψ y ϕ respectivamente.

Inicialmente el generador se alimenta con 2 semillas, la *semilla 1* que sera de 15 bits y la *semilla 2* que sera de 16 bits el algoritmo se ilustra en la figura 3.10.

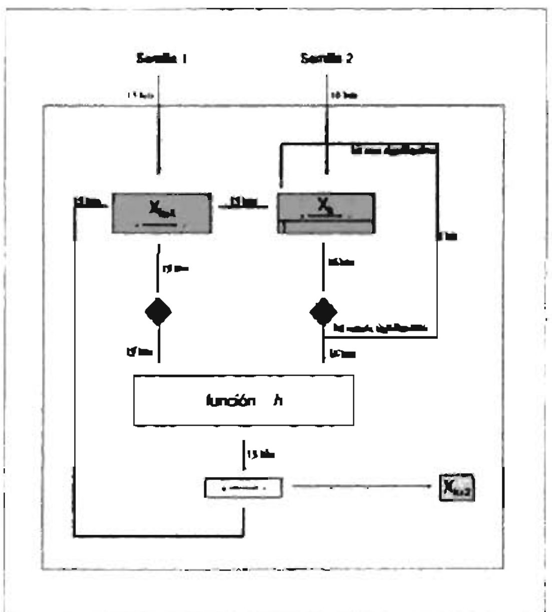


Figura 3.10: Generador pseudoaleatorio de llaves.

Una vez que se introducen las 2 semillas se genera la primer llave x_0^1 utilizando la ecuacion 3.7.

$$x_0^1 = h(semilla1, semilla2) \tag{3.7}$$

A partir de este valor empieza la recursión, donde x_0^1 pasa a ser el siguiente valor de x en la función $h(x, y)$ y el valor inicial de x el cual es x_0^0 que corresponde a la *semilla 1* se convierte en el nuevo valor de y. Debido a que y es de 16 bits, el bit menos significativo del valor anterior de y pasa a ser el bit mas significativo del nuevo valor y. Con los nuevos valores de x y y se calcula $x_0^2 = h(x_0^1, x_0^0)$ y así sucesivamente se va generando la secuencia pseudoaleatoria, según la ecuación 3.8.

$$x_{k+2} = h(x_{k+1}, x_k). \tag{3.8}$$

3.5.6 Funcionamiento del sistema ESAC

El sistema ESAC de manera general se muestra en la figura 3.11. Dicho sistema comprende de dos partes, la que realiza la encriptación de bloques de longitud de 15 bits y otra que se encarga del proceso inverso, desencriptar bloques de 15 bits utilizando la misma semilla inicial, utilizada en la encriptación, y así poder recuperar la información original.

Figura 3.11: Diagrama a bloques de la encriptación y desencriptación.

La parte correspondiente a ENCRIPTAR de la figura 3.11 es mostrada en la figura 3.12, en el cual podemos ver que el texto original es dividido en n bloques de 15 bits, estos 15 bits al igual que la llave creada por la función h pasan al encriptador y crean un bloque de 15 bits de texto encriptado, esto se repite hasta tener n bloques de texto encriptado, posteriormente se agrupan en un solo bloque.

Para la parte correspondiente a DESENCRIPTAR de la figura 3.11 se hace algo similar, se divide el texto encriptado en n bloques de 15 bits, cada bloque es desencriptado con la llave creada por la función h y cuando se han desencriptado los n bloques se agrupan formando el Texto Original, esta parte es mostrada en la figura 3.13.

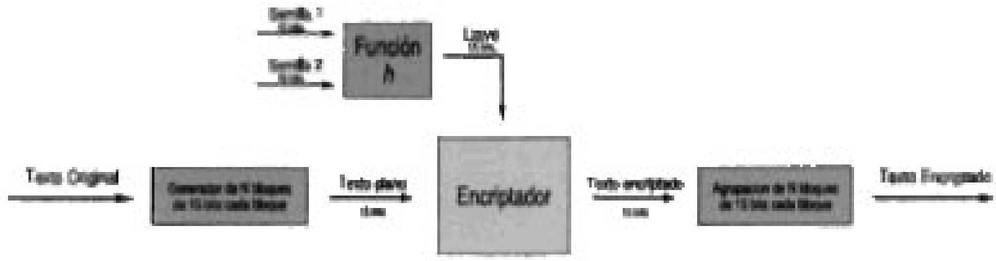


Figura 3.12: Diagrama a bloques del sistema de encriptación.

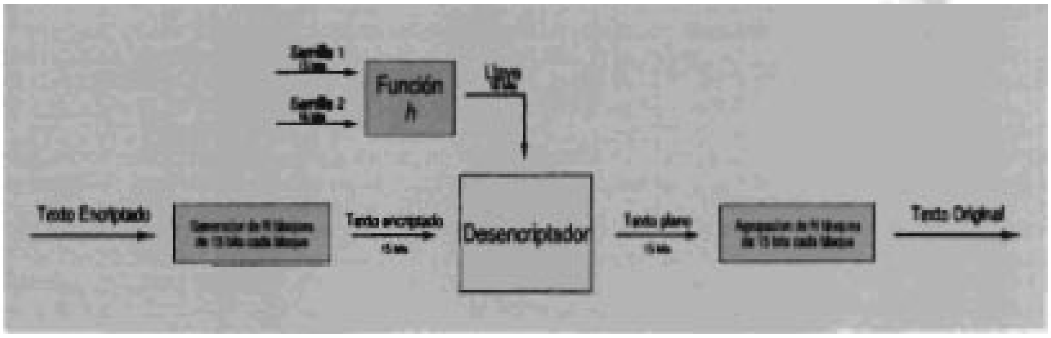


Figura 3.13: Diagrama a bloques del sistema de desencriptación.

Para entender un poco mas esto se mostrara un ejemplo de encriptación de bloques basado en autómatas celulares, se considerara la siguiente secuencia binaria.

$$1101101111000011110110101001000111011111 \qquad (3.9)$$

La secuencia (3.9) es introducida al diagrama de la figura 3.12 donde se dividirá en bloques de longitud de 15 bits, obteniendo los bloques mostrados en la ecuación (3.10)

$$\begin{aligned} m^0 &= 110110111100001 \\ m^1 &= 111011010100100 \\ m^2 &= 0111011111 \end{aligned} \qquad (3.10)$$

Al no ser la secuencia (3.9) múltiplo de 15, uno de los bloques es menor a 15 bits, por lo que se completa la palabra agregando ceros en las posiciones de los bits menos significativos. Los bloques resultantes se muestran en la ecuación (3.11)

$$
\begin{aligned}
m^0 &= 1\,1\,0\,1\,1\,0\,1\,1\,1\,1\,0\,0\,0\,0\,1 \\
m^1 &= 1\,1\,1\,0\,1\,1\,0\,1\,0\,1\,0\,0\,1\,0\,0 \\
m^2 &= 0\,1\,1\,1\,0\,1\,1\,1\,1\,1\,0\,0\,0\,0\,0
\end{aligned}
\tag{3.11}
$$

Para poder encriptar los bloques m^0, m^1 y m^2 se deben generar 3 de llaves.

Para generar las llaves se utilizan las semillas siguientes

$$\textit{semilla 1} = 1\,0\,0\,1\,1\,0\,0\,0\,1\,1\,1\,0\,0\,0\,0 \tag{3.12}$$

$$\textit{semilla 2} = 1\,1\,1\,1\,0\,1\,1\,1\,0\,0\,1\,1\,0\,0\,0\,1 \tag{3.13}$$

La llaves generadas utilizando estas semillas son

$$
\begin{aligned}
x^0 &= 0\,0\,1\,1\,0\,1\,0\,1\,0\,1\,0\,0\,0\,1\,0 \\
x^1 &= 0\,1\,1\,0\,1\,1\,1\,1\,0\,0\,1\,1\,0\,1\,0 \\
x^2 &= 0\,0\,1\,0\,0\,0\,1\,1\,0\,1\,1\,1\,0\,1\,1
\end{aligned}
\tag{3.14}
$$

Introduciendo las semillas y los bloques de texto plano en el encriptador como se muestra en la figura 3.12, obtenemos los bloques encriptados c^0,c^1 y c^2 mostrados en la ecuación (3.15).

$$
\begin{aligned}
c^0 &= 1\,0\,0\,0\,1\,0\,0\,0\,0\,0\,1\,1\,1\,0\,0 \\
c^1 &= 0\,0\,0\,0\,0\,1\,1\,0\,1\,0\,1\,1\,0\,1\,0 \\
c^2 &= 1\,1\,0\,1\,0\,0\,1\,0\,1\,0\,0\,1\,0\,1\,1
\end{aligned}
\tag{3.15}
$$

El texto encriptado es agrupado para formar una sola secuencia como se muestra en (3.16). Esta secuencia podrá estar lista para poder ser transmitida por un medio de comunicación.

$$1\,0\,0\,0\,1\,0\,0\,0\,0\,0\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,1\,1\,0\,1\,0\,1\,1\,0\,1\,0\,1\,1\,0\,1\,0\,0\,1\,0\,1\,0\,0\,1\,0\,1\,1 \tag{3.16}$$

Para su recuperación se tendrá que procesar mediante el diagrama de la figura 3.13, donde primero (3.16) es seccionado en bloques de 15 bits. Y con ayuda de las mismas *semilla*1 y *semilla*2 utilizadas para la encriptación se generan las mismas llaves para la desencriptación. Al introducir los bloques encriptados y las llaves al desencriptador obtenemos los bloques originales (3.11), por ultimo se le tienen que quitar los ceros que se le agregaron antes de ser encriptado.

$$\begin{matrix} 1\,1\,0\,1\,1\,0\,1\,1\,1\,1\,0\,0\,0\,0\,1 \\ 1\,1\,1\,0\,1\,1\,0\,1\,0\,1\,0\,0\,1\,0\,0 \\ 0\,1\,1\,1\,0\,1\,1\,1\,1 \end{matrix} \tag{3.17}$$

Los bloques (3.17) se agrupan en una sola secuencia obteniendo la secuencia inicial (3.18).

$$1\,1\,0\,1\,1\,0\,1\,1\,1\,1\,0\,0\,0\,0\,1\,1\,1\,1\,0\,1\,1\,0\,1\,0\,1\,0\,0\,1\,0\,0\,0\,1\,1\,1\,0\,1\,1\,1\,1 \tag{3.18}$$

Se debe tener en cuenta que el proceso de encriptación y desencriptación no pierde datos, por lo que la secuencia original (3.9) es igual a (3.18).

4

Implementación Numérica y Experimental del Sistema de Encriptación

En este capítulo se describe la implementación numérica y experimental del sistema de encriptación ESAC aplicado a información comprimida por la TOH. Dicha implementación comprende la integración completa de las etapas de compresión y encriptación de información de voz.

La implementación numérica de tal sistema es realizada en los ambientes de Matlab y de LabVIEW, mientras que para la versión experimental se utilizo una tarjeta de adquisición de datos cuya principal característica es que cuenta con un FPGA.

4.1 Descripción del Sistema

El sistema completo que se propone en esta tesis se muestra en la figura 4.1. El sistema se conforma de dos etapas, las cuales son etiquetadas como los módulos A y B. El módulo A, al cual denominaremos como Análisis del sistema, realiza la compresión y la encriptación de la información. Mientras que el módulo B se encarga de realizar el proceso inverso, es decir, la información que recibe la desencripta, después la condiciona para aplicarse la transformación inversa ondeleta y así tener una aproximación fiable de la señal de acuerdo a los criterios empleados en la compresión. Tal bloque es nombrado Síntesis del sistema.

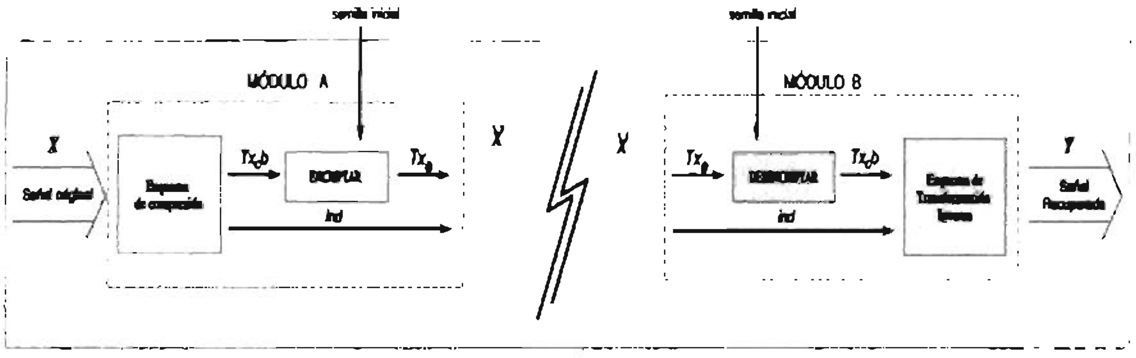


Figura 4.1: Diagrama a bloques del sistema de compresión y encriptación.

4.1.1 Análisis del sistema

El módulo A realiza dos tareas principales, la compresión y la encriptación. Para el procedimiento del esquema de compresión basado en TO se explico en el capítulo 2 y consiste de tres etapas. La primera de ellas consiste en la aplicación de la TOH a la información denotada como **X**, mientras que en la segunda se proporciona el porcentaje de energía a considerar para comprimir. En base a lo anterior se realiza la última etapa, la determinación del umbral para el cual los valores de los coeficientes que sean menores a dicho valor se igualan a cero. Como resultado nos proporciona dos vectores de salida, el de los coeficientes Tx_c que sobrevivieron al umbral y un vector *ind* de ceros y unos que nos indica con unos las posiciones originales de los coeficientes Tx_c. Antes de poder encriptar Tx_c con el sistema ESAC , a los valores de Tx_c que se encuentran en decimal se les aplicara un cambio de escala y se deben poner en su representación en binaria (15 bits), teniendo Tx_cb.

Mientras que el bloque de encriptación, el cual esta basado en el sistema ESAC, encripta los coeficientes cuya longitud individual es de 15 bits utilizando una semilla inicial elegida por el usuario. Con dicha semilla el generador de llaves del sistema ESAC generará un número de llaves igual al número de coeficientes Tx_cb y con estas llaves se encriptarán cada uno de ellos obteniendo la información encriptada Tx_e. Por último, la información de salida $\ddot{x}$ del módulo A se compone por la señal encriptada Tx_e y el vector de posiciones *ind*. La figura 4.2 muestra las etapas correspondientes al Análisis del sistema.

4.1.2 Síntesis del sistema

La recuperación de la señal se realiza a través del Módulo B, el cual consiste de dos etapas, la desencriptación y el esquema de transformación inversa. Para la desencriptación de Tx_cb se debe de utilizar la misma semilla inicial que se empleo en el módulo A para la encriptación, así el generador del sistema ESAC generara las mismas llaves que se utilizaron en la encriptación y poder desencriptar los coeficientes Tx_c.

En el bloque de recuperación de la señal como se dijo en el capítulo 2, consiste de 2 etapas,

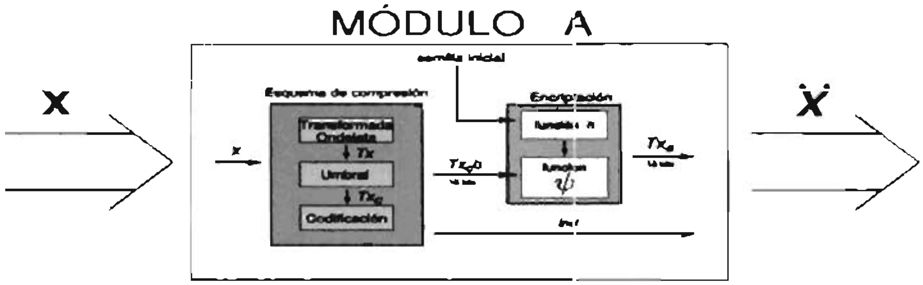


Figura 4.2: Diagrama correspondiente al Análisis.

en la primera se acondiciona la señal de los coeficientes Tx_c con *ind* para formar una señal de longitud igual a *ind* pero con los valores de Tx_c y en la segunda se le aplica la transformada inversa ondeleta a la señal acondicionada obteniendo una señal denotada como **Y** la cual es una aproximacion a **X**. Esto se muestra en la figura 4.3

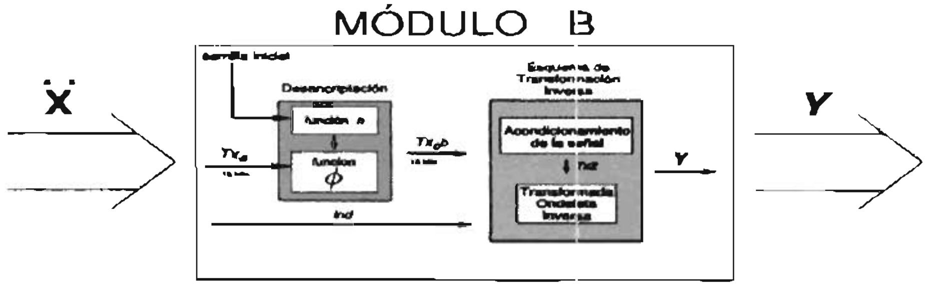


Figura 4.3: Diagrama correspondiente al Síntesis.

4.2 Implementación Numérica

La implementación numérica del sistema completo que comprende la compresión y encriptación del esquema de la figura 4.1 se realizo en los ambientes de programación de Matlab y LabVIEW. En tal implementación, se analizaron tres tipos diferentes de señales de voz que corresponden a tres personas diferentes, las cuales serán denotadas como $s1$, $s2$ y $s3$. Las tres señales tienen formato de audio .wav y son grabaciones adquiridas a las frecuencias de muestreo de 8 kHz para la primer señal y de 44.1 kHz para las restantes, con una resolución de 16 bits. El número de muestras a considerar fue de 65536, 1048576 y 4194304, respectivamente. El proceso de adquisición para $s1$ se realizó mediante un micrófono conectado a la PC, donde una joven pronuncia "*jueves 16 de agosto, esto es una prueba ... esto es una prueba para compresión de datos*" y tiene una duración de aproximadamente 8 segundos. A través del software `All2WAV Recorder` se adquirieron las señales $s2$ y $s3$. La información de $s2$ es un pequeño comentario realizado por un adulto en un video y tiene una duración aproximada de 24 segundos, mientras que para $s3$ corresponde a una conversación entre una niña y una persona adulta con duración aproximada de 95 segundos. La figura 4.4 muestra la gráfica correspondiente a tales señales de voz.

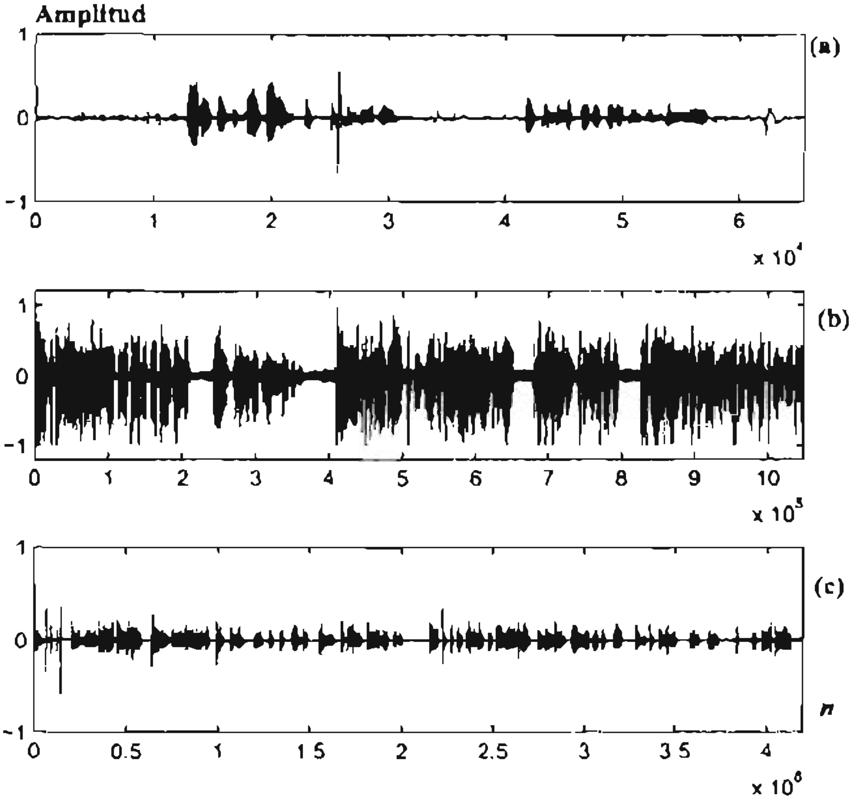


Figura 4.4: Señales de prueba de voz (a) $s1$, (b) $s2$ y (c) $s3$.

4.2.1 *Implementación en Matlab*

Los programas utilizados para la implementación numérica del sistema completo de encriptación en Matlab fueron alrededor de once, los cuales serán descritos de manera general dando una pequeña sintaxis y la relación con los parámetros de entrada y salida. Dicha implementación se realiza de la siguiente manera.

Etapa de compresión

En principio se carga la información de los archivos de voz en Matlab mediante la instrucción `X = waveread('s1.wav')`. En este punto se trata de ajustar que la longitud de la señal a analizar tenga un número de datos múltiplo de 2^n, con $n \geq 16$. A la primer señal de voz la denotaremos como **X**.

Posteriormente se le aplica la TOH a la señal **X** y se obtiene una señal Transformada Tx. La aplicación de la TOH a $s1$ se muestra en la figura 4.5 (a). Para aplicar la TOH se utilizo el programa titulado `Thaar.m` y recibe de entrada la señal a procesar. La sintaxis de la TOH en dicho software es `Tx = Thaar(X)`

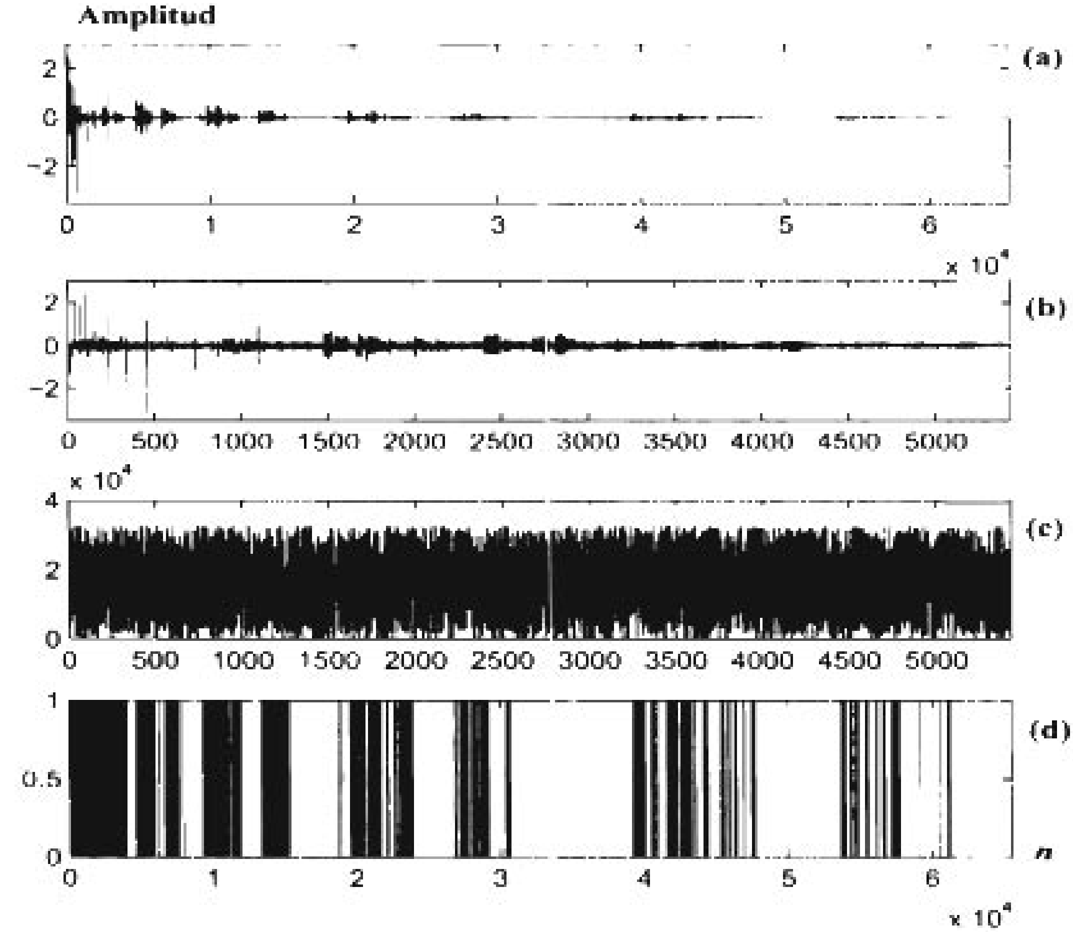


Figura 4.5: Representación del análisis para la señal s1 considerando un porcentaje del 95 % de energía. (a) Señal Tx correspondiente a la TOH de $s1$, (b) señal de los coeficientes que sobrevivieron al umbral, Tx_c, (c) representación decimal de señal encriptada, Tx_c y (d) indices de posición de la señal Tx_c.

A continuación se calcula el valor del umbral en términos del porcentaje de energía que se desea considerar. Para su determinación se considero del programa `umbrales.m`, el cual requiere de dos parámetros de entrada, la señal transformada y el porcentaje de energía. Su sintaxis es `umbral = umbrales(Tx, porcentaje)`. Cabe mencionar que para determinar cual es el porcentaje de energía, se realizaron varias pruebas para diferentes valores de energía y se seleccionaron las que permitió una reconstrucción "satisfactoria" debido al análisis de las señales que se consideraron. En [8] consideran la gráfica de la energía acumulativa como auxiliar para determinar el porcentaje de energía. Con el valor obtenido del umbral se procede a determinar que valores de la señal transformada Tx son menores que dicho valor y cuales son sus posiciones en Tx. Para esto se utilizo el programa `indice.m`. Tal programa recibe dos parámetros de entrada y nos proporciona un vector de menor longitud Tx_c y otro de ceros y unos, donde el "1" significa la posición de los datos de Tx_c. La sintaxis del programa es `[Txc, ind] = indice(Tx, umbral);`

Al considerar un porcentaje de energía del 95 %, se obtuvo la señal comprimida Tx_c con un número de datos de 5431, lo cual representa una compresión de la señal $s1$ por un factor de compresión de 12 : 1. Esta señal resultante para $s1$ es mostrada en la figura 4.5 (b). Mientras que la señal de información ind, compuesta de ceros y unos, tiene una longitud igual a la señal original y para $s1$ es mostrada en la figura 4.5 (d).

Por último, en la etapa de compresión se realiza la codificación de la información de Tx_c. La operación de codificación se realiza al aplicarles un escalamiento a los valores de Tx_c y posteriormente se realiza la conversión de decimal a binario a cada elemento escalado de la señal a encriptar con el programa `texto_binario.m`. Tal programa recibe sólo de entrada la información que se quiere codificar y nos proporciona cinco parámetros de salida, Tx_cb que es la representación en binario de Tx_c, $n1$ es la longitud de Tx_c e indica el número de llaves necesarias para la encriptación, $n3$ y $n4$ son los valores mínimo y máximo de la señal Tx_c, respectivamente. Su sintaxis correspondiente es `[Txcb,n1,n3,n4] = texto_binario(Txc);`.

Así se tienen dos señales de salida de la etapa de compresión, la señal comprimida Tx_cb que se va a encriptar y la señal ind que no será encriptada, junto con las constantes $n3$ y $n4$.

Etapa de encriptación

Para la encriptación de la información Tx_cb, es necesario primero la generación de las llaves. El programa `t_llaves.m` realiza tal generación con ayuda de la semilla inicial, la cual es proporcionada por el usuario. El programa requiere sólo la longitud de la señal a encriptar y así generar el número necesario de llaves requeridas para la encriptación. La salida del programa nos proporciona un arreglo matricial de tamaño igual a la señal a encriptar, donde cada fila corresponde a la representación binaria de cada llave de longitud de 15 bits y su sintaxis es `llaves = t_llaves(n1);` .

Después de tener las llaves se realiza la encriptación de Tx_cb utilizando el programa `encriptar.m`. Los parámetros que requiere de entrada son la información a encriptar Tx_cb y las llaves, obteniendo como resultado la señal encriptada Tx_e. La sintaxis del programa es `Txe = encriptar(Txcb, llaves);` . La representación decimal de Tx_e para la señal $s1$ puede verse en la figura 4.5(c).

Así, el Análisis del sistema completo tiene como salida la señal $\ddot{x}$, que se compone de la señal encriptada Tx_e, el vector de posiciones ind y las constantes $n3$ y $n4$, las cuales pueden agregarse al vector ind. Con esto $\ddot{x}$ está disponible para ser transmitida por algún medio de comunicación, tarea que no se realizó en este trabajo.

Etapa de Desencriptación

Para la desencriptación de la señal de entrada x, primero se tienen que generar las llaves con la misma semilla utilizada en la encriptación. Por lo que se utiliza nuevamente el programa `t_llaves.m` con sus respectivos parámetros. Teniendo las llaves se procede a desencriptar la señal de los coeficientes codificados Tx_e con el programa `desencriptar.m`, cuya sintaxis es `Txcb = desencriptar(Txe, llaves);` proporcionándonos la señal de coeficientes Tx_cb.

Etapa de transformación inversa

El esquema de transformación inversa consta de dos módulos principales, el acondicionamiento de la señal y el de la aplicación de la transformada ondeleta inversa. En el acondicionamiento de la señal es necesario tener los coeficientes desencriptados en su versión binaria y el vector de posiciones ind, en el cual se agruparon las constantes $n3$ y $n4$. En base a esto se procede a realizar la conversión de binario a decimal a cada elemento de la señal desencriptada y posteriormente a su decodificación. El programa `texto_decimal.m` con sintaxis `Txc=texto_decimal(Txcb,n3,n4);` realiza la conversión y la decodificación, donde Tx_c es la señal de coeficientes recuperados. Con esta señal y el vector de posiciones ind se procede al acondicionamiento final de la información, el cual consiste en el reacomodo de los coeficientes en su posición original. El programa `acondicionamiento_señal.m` realiza dicha adecuación y la sintaxis es `Tx2=acondicionamiento_señal(Txc,ind);`, donde se tiene a la señal $Tx2$ con misma longitud que el de la señal original analizada. La representación de $Tx2$ para la señal $s1$ se observa en la figura 4.6 (a). Finalmente, se le aplica la transformada ondeleta inversa a la señal $Tx2$ con el programa `thaarinv.m`, con sintaxis `Y = thaarinv(Tx2)`. Se observa que el programa requiere sólo de la señal a la cual se le aplicará la transformación inversa, mientras que la señal de salida, **Y**, es la señal de voz recuperada. La figura 4.6 (b) muestra la versión de la señal recuperada para la señal $s1$. Como una herramienta de eficiencia en el proceso se considera calcular el error cuadrático medio entre la señal original **X** y la señal recuperada **Y**. Para la señal $s1$ con un porcentaje de energía del $95\,\%$, el error en la recuperación de la señal es de aproximadamente 0.00013191. La figura 4.6 (c) muestra la gráfica del error obtenido al restar la señal recuperada de la original $s1$, donde se puede observar que la amplitud del error es muy pequeña, de hecho la amplitud se encuentra limitada en el intervalo $[-0.069, 0.065]$.

En el Apéndice B se muestra el código en lenguaje de Matlab de los programas utilizados en el sistema completo de encriptación.

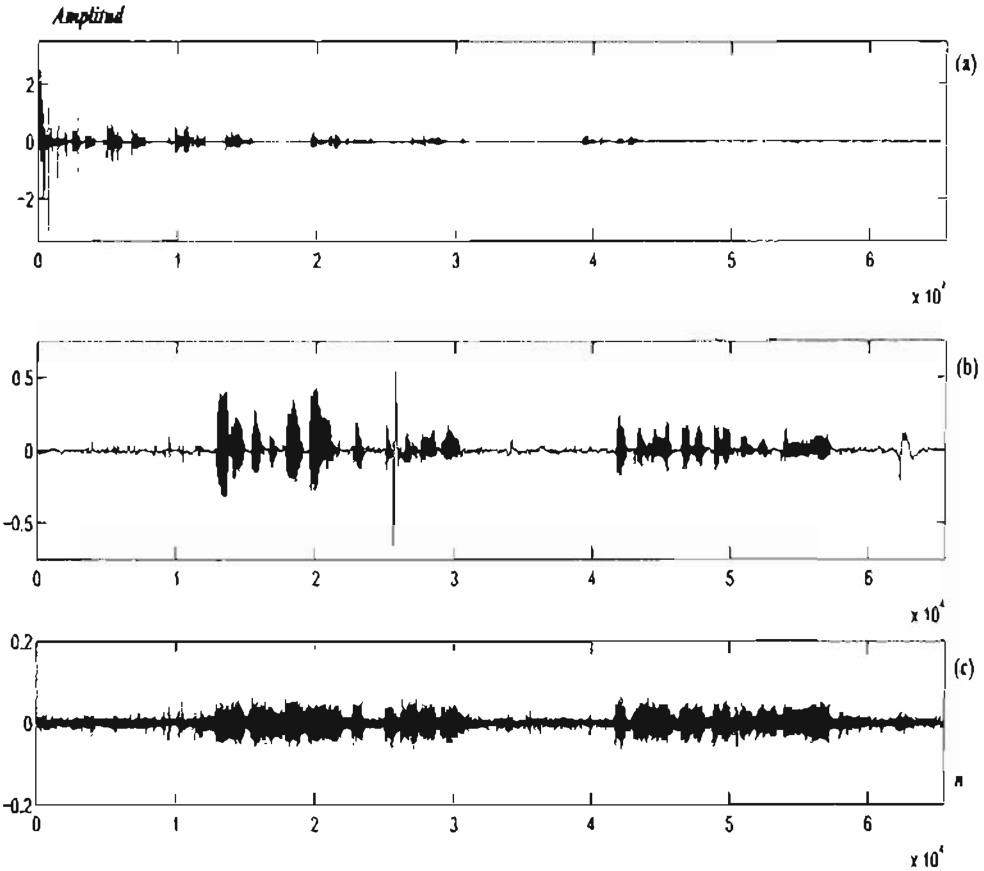


Figura 4.6: Representación de la síntesis para la señal $s1$ considerando un porcentaje del 95 % de energía. (a) Señal $Tx2$ correspondiente a la señal de coeficientes desencriptados de $s1$, (b) señal recuperada **Y** de $s1$ y (c) gráfica de errores obtenida al restar a la señal original **X** la señal recuperada **Y** para $s1$.

4.2.2 Implementación en LabVIEW

El software LabVIEW de National Instruments es una herramienta gráfica que tiene la flexibilidad de un lenguaje de programación combinado con herramientas integradas diseñadas específicamente para pruebas, medidas y control, LabVIEW es generalmente usado para la adquisición de datos, control de instrumentos y automatización industrial. Los programas hechos o realizados con LabVIEW son llamados Instrumentos Virtuales(VI's). Cada VI consta de dos interfaces gráficas, un panel frontal y un diagrama de bloques. El panel frontal puede estar constituido por controles e indicadores, los cuales representan las entradas y salidas del VI, mientras que en el diagrama de bloques es donde se implementa el código fuente gráfico.

En esta implementación numérica se realizó de manera general los mismos pasos y etapas realizadas en Matlab. En el Análisis del sistema completo de encriptación, se consideraron las etapas de compresión con la TOH y encriptación con el ESAC. Para el esquema de compresión se utilizaron los SUBVI's mostrados en la figura 4.7, en donde se puede observar como se va realizando esta etapa para una señal de entrada de voz x dándonos como resultado la señal transformada y comprimida en versión decimal $Tx_c c$ y la señal de índices de posición ind. Cabe mencionar que la amplitud de la información de audio y a su vez de voz no esta limitada al intervalo de -1 a 1, situación diferente a la de Matlab. Además, para la conversión de números binarios a decimal y viceversa, se emplea una función determinada de LabVIEW.

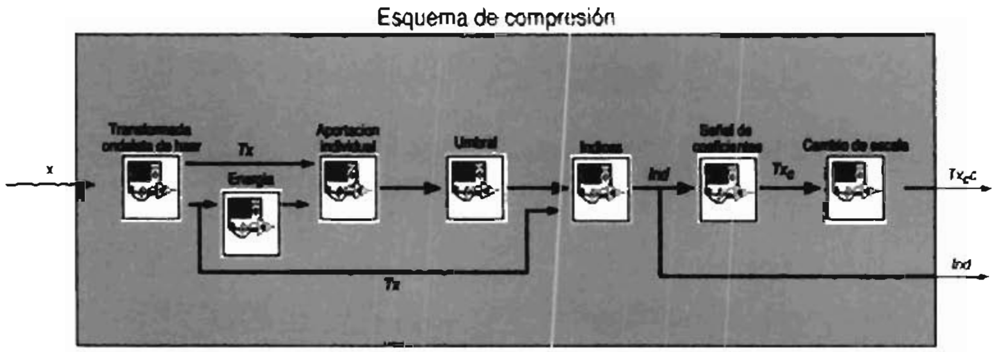


Figura 4.7: Representación del esquema de compresión utilizado en LabVIEW.

Mientras en la encriptación de $Tx_c c$ se utilizaron los SUBVI's mostrados en la figura 4.8, dando como resultado los coeficientes encriptados Tx_e los cuales están en versión decimal. Por lo que se tiene la señal encriptada y la señal de posiciones.

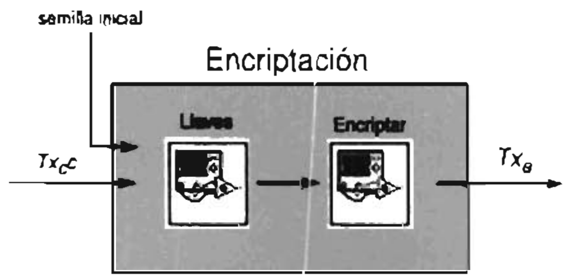


Figura 4.8: Representación del esquema de encriptación en LabVIEW.

Para la operación de desencriptar Tx_e se considera la representación en LabVIEW mostrada en la figura 4.9. Primero se utiliza el SUBVI `llaves.vi`, para generar las llaves necesarias y posteriormente con el SUBVI `desencriptar.vi`, se desencripta la información dándonos como resultado la señal $Tx_c c$.

Por último, para el esquema de transformación inversa se utilizaron los SUBVI's mostrados en la fig 4.10, siguiendo el mismo procedimiento de matlab. Los SUBVI's realizados en esta

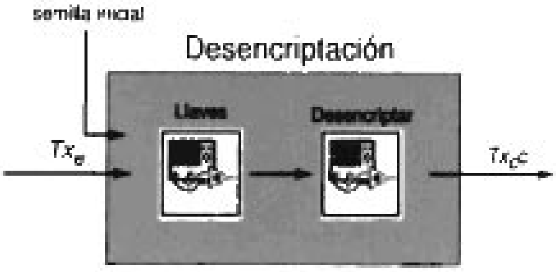


Figura 4.9: Representación del esquema de desencriptación utilizado en LabVIEW.

implementación se muestran en el Apéndice C.

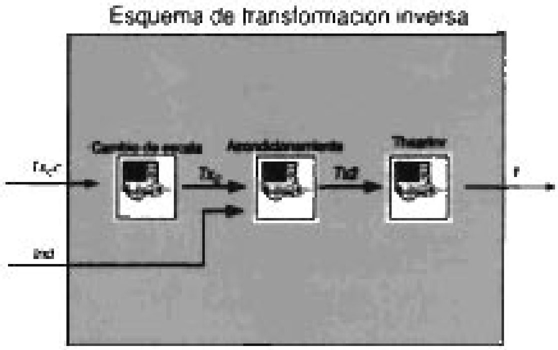


Figura 4.10. Representación en LabVIEW del esquema de transformación inversa.

4.3 Implementación experimental del sistema de encriptación

Para esta implementación se utilizó el LabVIEW junto con una tarjeta de adquisición de datos PCI-7811R de National Instruments, la cual tiene integrado un FPGA.

Dada la gran cantidad de compuertas con las que cuenta un FPGA, es posible implementar sistemas digitales complejos. Una característica importante es que procesan la información que adquieren (para el caso de tarjetas de adquisición de datos con FPGA) de forma paralela.

Para programar el FPGA de la tarjeta se requirió del módulo LabVIEW FPGA. Gracias a este módulo no se requiere un conocimiento de herramientas de diseño de hardware complejo, ya que precisamente el módulo LabVIEW FPGA trabaja en un ambiente gráfico, la manera de trabajar con LabVIEW FPGA es crear VI´s y emularlos en la computadora y cuando ya estén funcionando correctamente se compilan en la tarjeta con el FPGA, posteriormente estos programas en el FPGA interactúan con diversas aplicaciones que se encuentran en la computadora (si se requiere). Desafortunadamente tal módulo está limitado en cuanto al uso de diversas herramientas de programación existentes en el software LabVIEW, un ejemplo es la restricción en las operaciones con números flotantes, lo cual es requerido en el esquema completo de compresión. En el Apéndice D se da una breve descripción del FPGA.

La implementación en el FPGA se realizó a los bloques correspondientes de "ENCRIPTAR" y "DESENCRIPTAR" de la figura 4.1. Dicha tarea consistió en implementar y trasladar los mismos SUBVI's realizados en LabVIEW en el FPGA.

4.4 Resultados de la aplicación a las señales de voz

En esta sección se presentan los resultados obtenidos de la aplicación del sistema completo de encriptación.

Las gráficas mostradas en la figura 4.11 muestran los resultados obtenidos en LabVIEW para la señal $s1$ en las diferentes etapas del Análisis del sistema, (a) TOH de $s1$, (b) coeficientes diferentes de cero considerando el 95 % de energía, (c) coeficientes encriptados en versión decimal y (d) la señal de posiciones de los coeficientes diferentes de cero.

Mientras que en la fig 4.12 se pueden observar los resultados obtenidos de la Sintesis del sistema para la señal $s1$.

Cabe mencionar que en las implementaciones numéricas, en Matlab y LabVIEW, del sistema completo de la señal $s1$, se puede ver una diferencia en la escala de amplitud de la señal de audio entre tales implementaciones, y por lo tanto en el proceso de encriptar y comprimir, pero de manera cuantitativa y cualitativamente se lograron los mismos resultados.

Sin embargo, una diferencia muy notable radica en el tiempo de ejecución que realizan los dos programas al momento de procesar las diferentes señales de voz, donde el LabVIEW es más eficiente que el Matlab. En la tabla 4.1 se muestran los resultados obtenidos para la señal $s1$ considerando diferentes porcentajes de energía en la compresión en sus diferentes implementaciones. Se puede observar que numéricamente el LabVIEW realiza la aplicación del sistema completo de encriptación que el Matlab, y que a su vez la implementación experimental de la etapa de encriptación es más rápido que el LabVIEW.

		Señal s1			
% de energía	No. de Coeficientes	Tasa de Compresión	$t(s)$ en Matlab	$t(s)$ en LabVIEW	$t(s)$ en LabVIEW-FPGA
99 %	13913	7.71:1	124.129	1.227	0.545
95 %	5431	12.06:1	9.064	0.575	0.320
90 %	3231	20.28:1	4.586	0.420	0.287
85 %	2246	29.17:1	4.071	0.350	0.375
80 %	1627	40.28:1	3.697	0.320	0.268

Tabla 4.1: Resultados comparativos de las tasas de compresión y tiempo en ejecución para la señal $s1$ en las diferentes implementaciones.

Continuando con el análisis de las señales, se tiene que para la señal $s2$ el Matlab resultó ser ineficiente para procesarla debido al tamaño de la señal, longitud de 2^{20}, por lo que se optó por

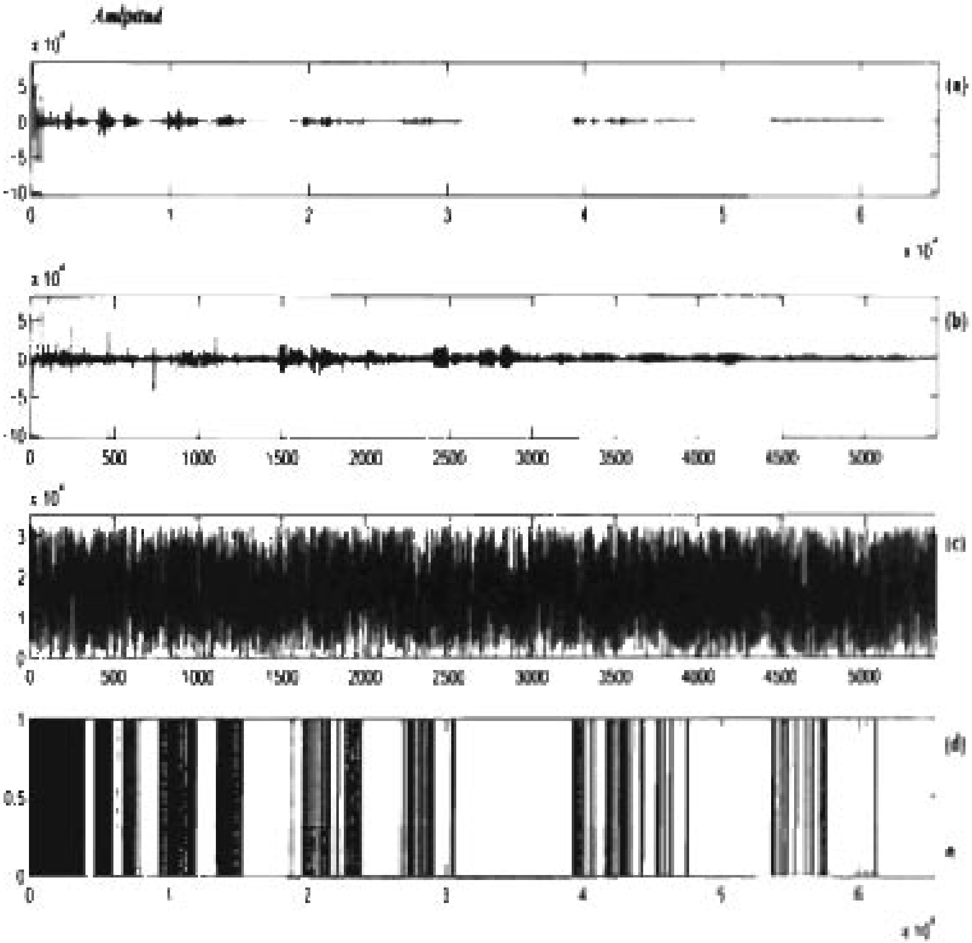


Figura 4.11: Representación del análisis para la señal $s1$ considerando un porcentaje del 95 % de energía. (a) Señal Tx correspondiente a la TOH de $s1$, (b) señal de los coeficientes que sobrevivieron al umbral, Tx_c, (c) representación decimal de señal encriptada, Tx_e y (d) índices de posición de la señal Tx_c.

realizar los cálculos numéricos con el LabVIEW. La figura 4.13 muestra el proceso de Análisis de la señal $s2$, considerando el 95 % de su energía para comprimir, mientras que la figura 4.14 muestra el proceso de Síntesis realizado para dicha señal.

La tabla 4.2 muestra el desempeño en las implementaciones numérica(LabVIEW) y experimental(LabVIEW FPGA) para diferentes tasas de compresión. Se puede observar nuevamente que la implementación experimental es más eficiente en tiempo de ejecución que la numérica. De hecho, entre más grande sea la señal a procesar se mejora el tiempo de ejecución.

Por último, el Análisis del sistema para la señal $s3$ se muestra en la figura 4.15, con la misma tasa inicial de compresión del 95 % de su energía, y en la figura 4.16 se muestran los resultados

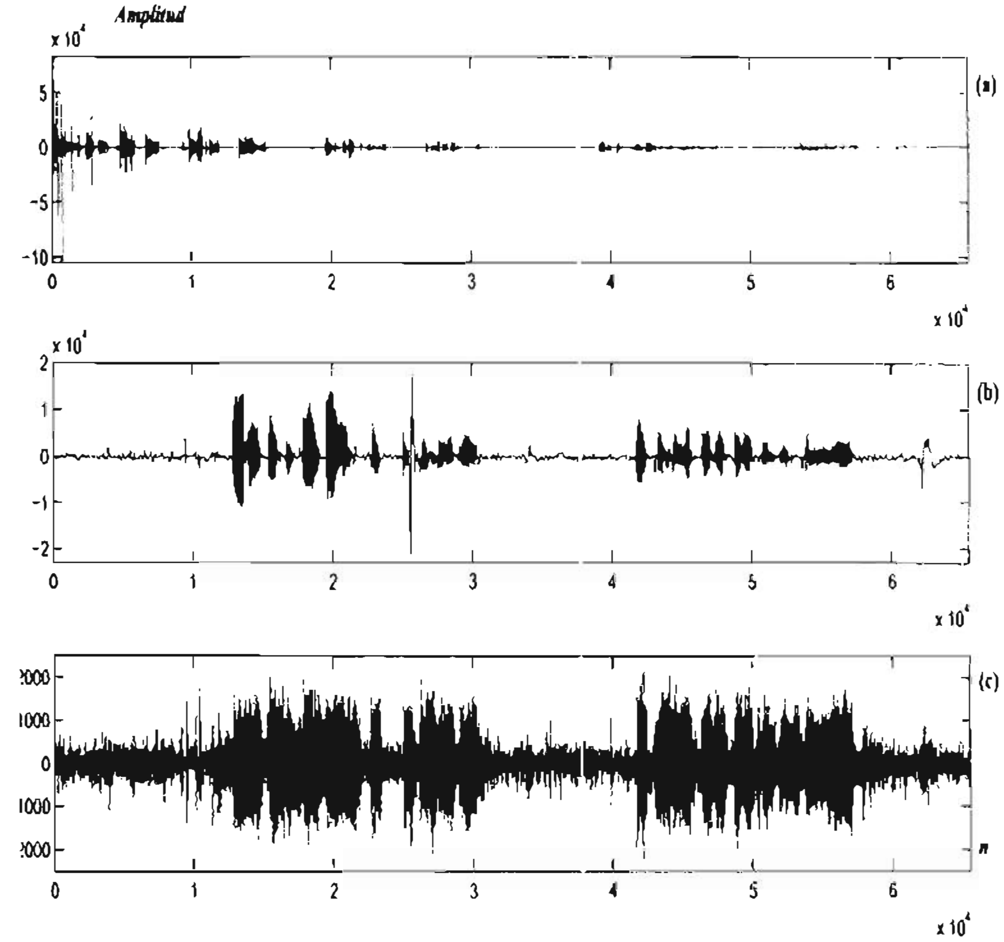


Figura 4.12: Representación de la síntesis para la señal $s1$ considerando un porcentaje del 95 % de energía. (a) Señal $Tx2$ correspondiente a la señal de coeficientes desencriptados de $s1$, (b) señal recuperada **Y** de $s1$ y (c) gráfica de errores obtenida al restar a la señal original **X** la señal recuperada **Y** para $s1$.

de la Síntesis.

La tabla 4.3 muestra los resultados obtenidos para diferentes tasa de compresión de la señal $s3$ y sus diferentes implementaciones.

Nuevamente se tiene un mejor desempeño en tiempo de ejecución con la implementación experimental, aunque en algunos casos es muy cercano el tiempo de ejecución. Sin embargo, al realizar la generación de $50,000,000$ de llaves se pudo observar que el LabVIEW tarda aproximadamente 459918 milisegundos, mientras que con ayuda de la tarjeta FPGA se tarda 377767 milisegundos. Con esto se tiene que la combinación del software y hardware se logro un ahorro de aproximadamente 82.151 segundos, sobre el caso de utilizar únicamente software.

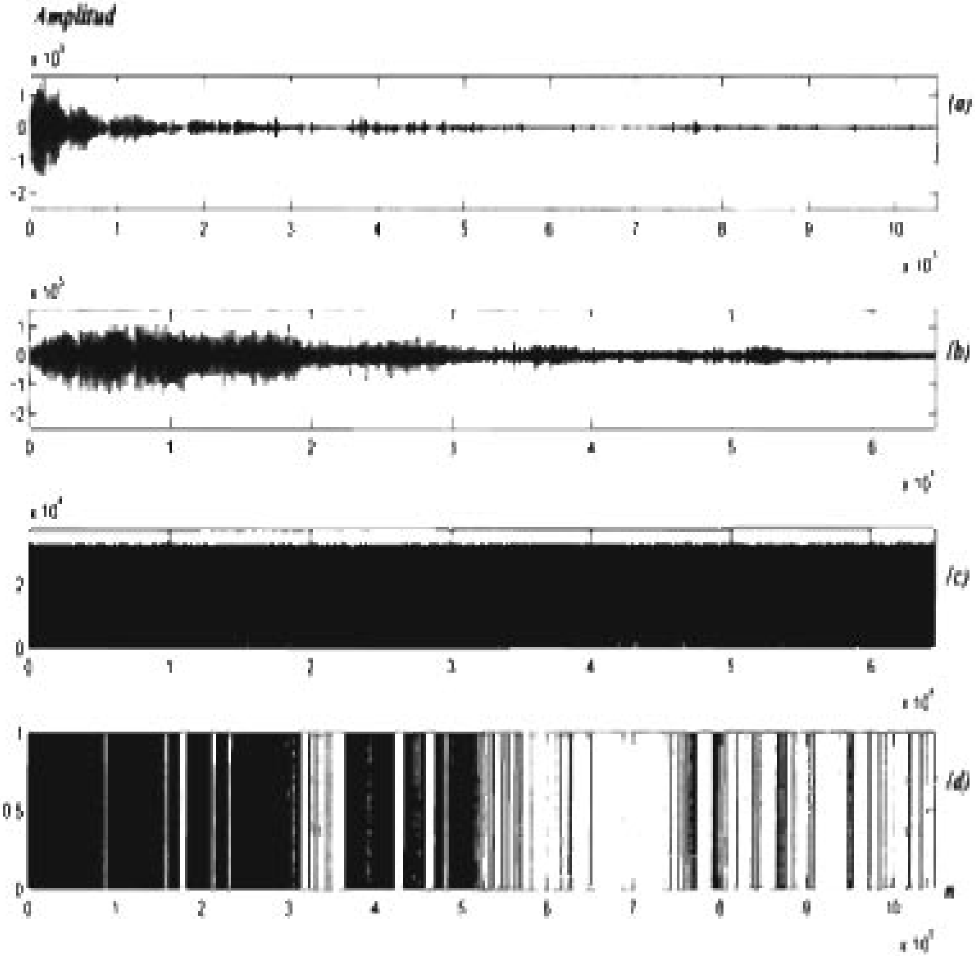


Figura 4.13 Representación del análisis para la señal s2 considerando un porcentaje del 95 % de energía (a) Señal Tx correspondiente a la TOH de s2, (b) señal de los coeficientes que sobrevivieron al umbral, $Tx_{r}c$, (c) representación decimal de señal encriptada, Tx_{e} y (d) índices de posición de la señal Tx_{r}

		Señal s2		
% de energía	No. de Coeficientes	Tasa de Compresión	t(s) en LabVIEW	t(s) en LabVIEW-FPGA
99 %	169272	6.19:1	16.352	7.967
95 %	64572	16.23:1	8.644	5.649
90 %	35308	29.69:1	6.728	5.333
85 %	22620	46.35:1	5.873	5.203
80 %	15701	66.78:1	5.575	5.081

Tabla 4.2 Resultados comparativos de las tasas de compresión y tiempo en ejecución para la señal s2 en las diferentes implementaciones.

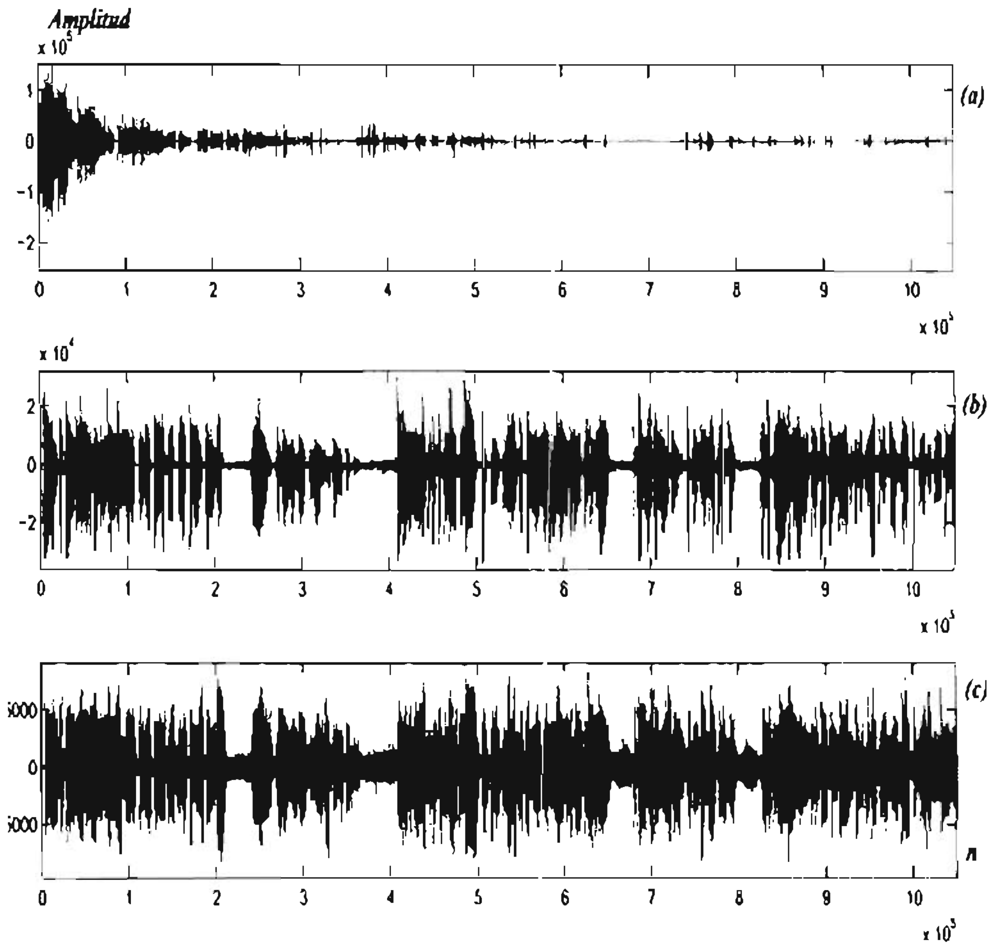


Figura 4.14: Representación de la síntesis para la señal $s2$ considerando un porcentaje del 95 % de energía. (a) Señal $Tx2$ correspondiente a la señal de coeficientes desencriptados de $s2$, (b) señal recuperada $\mathbf{Y}$ de $s2$ y (c) gráfica de errores obtenida al restar a la señal original $\mathbf{X}$ la señal recuperada $\mathbf{Y}$ para $s2$.

		Señal s3		
% de energía	No. de Coeficientes	Tasa de Compresión	t(s) en LabVIEW	t(s) en LabVIEW-FPGA
99 %	706703	5.93:1	67.652	32.751
95 %	271998	15.42:1	35.906	23.176
90 %	148281	28.28:1	27.141	21.837
85 %	94629	44.32:1	24.035	21.213
80 %	64963	64.56:1	22.697	20.829

Tabla 4.3: Resultados comparativos de las tasas de compresión y el tiempo de ejecución para la señal $s3$ en las diferentes implementaciones.

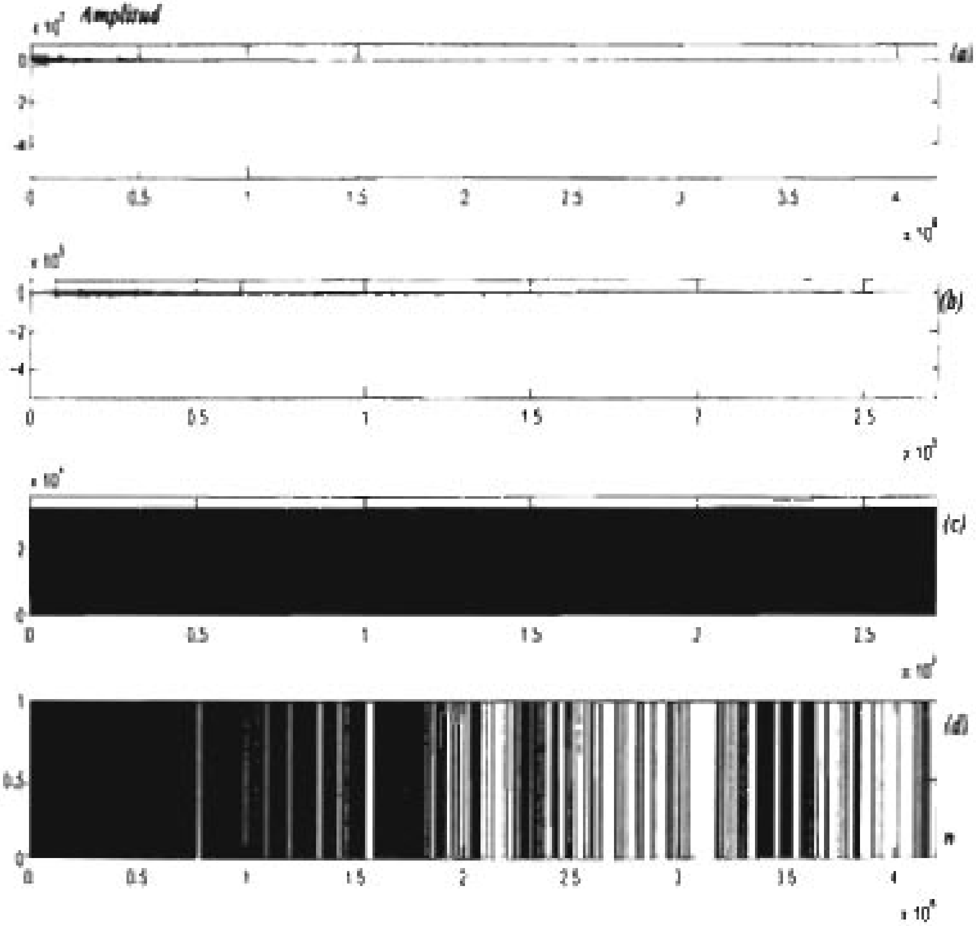


Figura 4.15: Representación del análisis para la señal $s3$ considerando un porcentaje del 95 % de energía. (a) Señal Tx correspondiente a la TOH de $s3$, (b) señal de los coeficientes que sobrevivieron al umbral, Tx_cc, (c) representación decimal de señal encriptada, Tx_r y (d) índices de posición de la señal Tx_r.

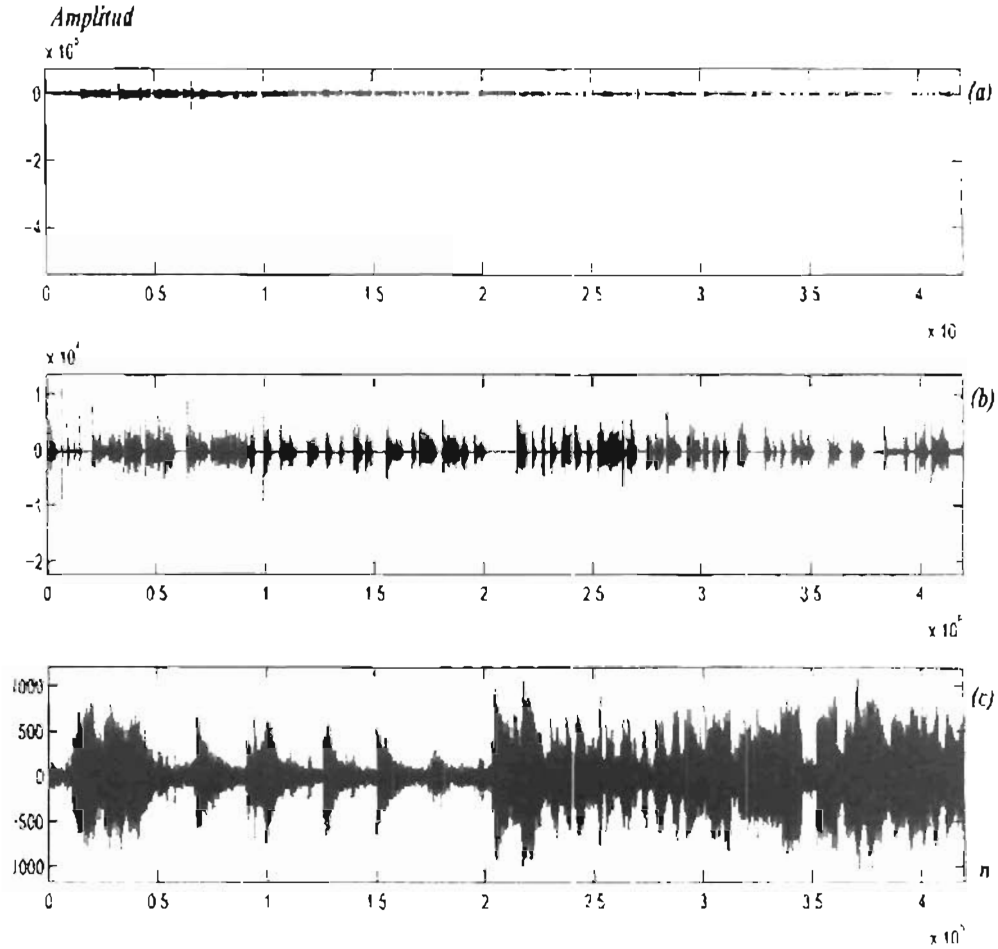


Figura 4.16: Representación de la síntesis para la señal $s3$ considerando un porcentaje del 95% de energía. (a) Señal $Tx2$ correspondiente a la señal de coeficientes desencriptados de $s3$, (b) señal recuperada **Y** de $s3$ y (c) gráfica de errores obtenida al restar a la señal original **X** la señal recuperada **Y** para $s3$.

5

Conclusiones

En este trabajo se realizó la implementación numérica y experimental (de manera parcial) de un sistema que integra las etapas de compresión y encriptación de señales de voz. La etapa de compresión está basada primordialmente en la aplicación de la herramienta conocida como transformada ondeleta, mientras que para el proceso de encriptación se consideró el sistema ESAC propuesto en [14].

Primeramente se presentaron de manera básica y general las bases necesarias para describir el análisis de ondeletas en sus versiones continua y discreta, considerando en mayor medida el enfoque discreto presentado en [8, 18]. Para la implementación numérica de la transformada ondeleta de Haar se utilizó el algoritmo de la transformada rápida ondeleta(basado en el análisis multi-resolución), el cual resulta fácil y rápido de implementar. Posteriormente, se presentó un esquema de compresión basado en la transformada ondeleta discreta de Haar, en el cual la preservación de la energía jugó un papel importante para la elección del umbral.

De la misma manera, se describió el proceso de encriptación y desencriptación del sistema denominado ESAC, el cual fue realizado en [14] basado en autómatas celulares. La descripción del sistema comprendió la explicación de la función generadora de llaves, así como de las familias de permutación para realizar las operaciones de encriptación y desencriptación, las cuales procesaron bloques de información de 15 bits. La encriptación se realizó de manera parcial a la información comprimida, ya que sólo se aplicó dicha operación a los coeficientes con magnitud mayor a cero, lo cual nos permitió un manejo más flexible de la información a encriptar debido a la reducción del tamaño o longitud de la señal.

Posteriormente se detalla paso por paso la manera como se realizó la implementación numérica del sistema completo, que incluye las etapas de compresión basada en ondeletas y de encriptación / desencriptación del ESAC, así como el caso experimental que considera sólo la etapa de encriptación. Cabe mencionar que la implementación numérica del sistema completo se realizo en dos lenguajes de programación, Matlab y LabVIEW, mientras que para el caso experimental se utilizo la tarjeta de adquisición de datos PCI-7811R de National Instruments la cual tiene integrado un FPGA. Por último, se presentaron los resultados obtenidos tanto de manera numérica como experimental para diferentes señales de voz. Se pudo observar que la transformada ondeleta discreta de Haar es una herramienta útil para procesar la información de manera eficiente, ya que se obtuvieron buenas tasas de compresión debido a la gran concentración de energía que se presentó en pocos coeficientes de la señal transformada. Mientras que para la parte de encriptación se realizaron pruebas numéricas y experimentales para las señales comprimidas, mostrando un buen desempeño al encriptar parcialmente la información. Además se observó que con la implementación del ESAC en el FPGA se logró tener una mayor rapidez, a diferencia del caso numérico, en cuanto a la generación de las llaves utilizadas por el ESAC, así como la encriptación y desencriptación de la información comprimida.

Una vez concluido este trabajo, consideramos que se tiene un atractivo sistema que integra la compresión y encriptación de información de voz, y resultaría ser una herramienta útil para el desarrollo y uso de las aplicaciones actuales de multimedia. Creemos que el sistema implementado es simple, rápido y podría incrustarse de manera fácil en un sistema de comunicación existente con mínimos requerimientos. Consideramos que el sistema propuesto podría tener un mejor desempeño si se realizan o desarrollan las siguientes propuestas: a) optimización de la selección del umbral permitiéndonos mejores tasas de compresión, b) implementación eficiente de la transformada ondeleta discreta de Haar en el FPGA, c) involucrar el valor del umbral con las semillas iniciales para la generación de las llaves, entre otras.

A

Algoritmos de un solo tiempo

$\boxed{A.1}$ Expresiones booleanas de un solo tiempo de la función $\mathrm{m} = \phi_{\mathrm{x}}(\mathrm{c})$ de tamaño 15 de la unidad encriptadora.

$$
\begin{aligned}
m1 &= x15 \oplus x13 \oplus x9 \oplus x1 \oplus c1 \\
m2 &= x14 \oplus x10 \oplus x2 \oplus c2 \\
m3 &= x13 \oplus x11 \oplus x9 \oplus x3 \oplus x1 \oplus c1 \oplus c3 \\
m4 &= x12 \oplus x4 \oplus c4 \\
m5 &= x11 \oplus x9 \oplus x5 \oplus x3 \oplus x1 \oplus c1 \oplus c3 \oplus c5 \\
m6 &= x10 \oplus x6 \oplus x2 \oplus c2 \oplus c6 \\
m7 &= x9 \oplus x7 \oplus x5 \oplus x1 \oplus c1 \oplus c5 \oplus c7 \\
m8 &= x8 \oplus c8 \\
m9 &= x7 \oplus x5 \oplus x1 \oplus c1 \oplus c5 \oplus c7 \oplus c9 \\
m10 &= x6 \oplus x2 \oplus c2 \oplus c6 \oplus c10 \\
m11 &= x5 \oplus x3 \oplus x1 \oplus c1 \oplus c3 \oplus c5 \oplus c9 \oplus c11 \\
m12 &= x4 \oplus c4 \oplus c12 \\
m13 &= x3 \oplus x3 \oplus c1 \oplus c3 \oplus c9 \oplus c11 \oplus c13 \\
m14 &= x2 \oplus c2 \oplus c10 \oplus c14
\end{aligned}
$$

$m15 = x1 \oplus c1 \oplus c9 \oplus c13 \oplus c15$

$\boxed{A.2}$ Expresiones booleanas de un solo tiempo de la función $\mathrm{c} = \psi_{\mathrm{x}}(\mathrm{m})$ de tamaño 15 de la unidad encriptadora.

$c1 = x15 \oplus x13 \oplus x9 \oplus x1 \oplus m1$

$c2 = x14 \oplus x13 \oplus x2 \oplus m2$

$c3 = x15 \oplus x11 \oplus x3 \oplus m1 \oplus m3$

$c4 = x12 \oplus x4 \oplus m4$

$c5 = x13 \oplus x5 \oplus m3 \oplus m5$

$c6 = x14 \oplus x6 \oplus m2 \oplus m6$

$c7 = x15 \oplus x7 \oplus m1 \oplus m3 \oplus m5 \oplus m7$

$c8 = x8 \oplus m8$

$c9 = x9 \oplus m7 \oplus m9$

$c10 = x10 \oplus m6 \oplus m10$

$c11 = x11 \oplus m5 \oplus m9 \oplus m7 \oplus m11$

$c12 = x12 \oplus m4 \oplus m12$

$c13 = x13 \oplus m3 \oplus m5 \oplus m11 \oplus m13$

$c14 = x14 \oplus m2 \oplus m6 \oplus m10 \oplus m14$

$c15 = x15 \oplus m1 \oplus m3 \oplus m5 \oplus m7 \oplus m9 \oplus m10 \oplus m13 \oplus m15$

$\boxed{A.3}$ Ecuaciones de un solo tiempo de las permutaciones y la función *h* para bloques de 15 bits

$t1 = x1 \oplus y2$

$t2 = x2 \oplus y1 \oplus y3$

$t3 = x1 \oplus x3 \oplus y4$

$t4 = x4 \oplus y1 \oplus y3 \oplus y5$

$t5 = x5 \oplus x3 \oplus x1 \oplus y2 \oplus y6$

$t6 = x6 \oplus x2 \oplus y1 \oplus y5 \oplus y7$

$$
\begin{aligned}
t7 &= x7 \oplus x5 \oplus x1 \oplus y8 \\
t8 &= x8 \oplus y1 \oplus y5 \oplus y7 \oplus y9 \\
t9 &= x9 \oplus x7 \oplus x5 \oplus x1 \oplus y2 \oplus y6 \oplus y10 \\
t10 &= x10 \oplus x6 \oplus x2 \oplus y1 \oplus y3 \oplus y5 \oplus y9 \oplus y11 \\
t11 &= x11 \oplus x9 \oplus x5 \oplus x3 \oplus x1 \oplus y4 \oplus y12 \\
t12 &= x12 \oplus x4 \oplus y1 \oplus y3 \oplus y9 \oplus y11 \oplus y13 \\
t13 &= x13 \oplus x11 \oplus x9 \oplus x3 \oplus x1 \oplus y2 \oplus y10 \oplus y14 \\
t14 &= x14 \oplus x10 \oplus x2 \oplus y1 \oplus y9 \oplus y13 \oplus y15 \\
t15 &= x15 \oplus x13 \oplus x9 \oplus x1 \oplus y16
\end{aligned}
$$

B

Apéndice de listado de programas en Matlab

En este apéndice se listan los principales programas que se utilizaron para implementar numéricamente en Matlab el sistema de encriptación completo.

B.1 Programas del Análisis del sistema

Etapa de compresión

TRANSFORMADA ONDELETA DE HAAR

```
function y=Thaar(X)

L = length(X);
cte = sqrt(2);

while L > 1
    for a = 1 : L/2
        V(a) = ( X(2*a - 1) + X(2*a) )/ cte;
        V(L/2 + a) = ( X(2*a - 1) - X(2*a) )/ cte;
    end
    L = L/2;
    X = V;
end

y = X;
```

CÁLCULO DEL UMBRAL

```
function umbral = umbrales(Tx, porcentaje)

N = length(Tx);
r1 = porcentaje/100;
Ex = sum(Tx.^2);

xm = sort(abs(Tx));
Lm = fliplr(xm);

if porcentaje == 100,
   umbral=xm(1);
else
   ETx=cumsum((Lm.^2)/Ex);
   ind = find(ETx >= r1 );
   umbral = Lm(ind(1));
end
```

PROGRAMA PARA ENCONTRAR LOS INDICES DONDE ESTARÁN LOS DATOS A ENCRIPTAR

```
function [Txc, ind] = indice(Tx,umbral)

n = length(Tx);
y2 = zeros(1, n);
indi = find(abs(Tx) >= umbral);
y2(indi) = Tx(indi);
nn =length(indi);
y3 = zeros(1, nn);
for i=1:nn
   y3(i) = y2(indi(i));
end
if umbral == 0
   ind = (indi~=0);
else
   ind =(y2~=0);
end

Txc = y3;
```

CONVERSIÓN DECIMAL A BINARIO

```
function [Txcb,n1,n3,n4] = Texto_binario(Txc)

bits=15;
n1=length(Txc);

n3=max(Txc);
n4=min(Txc);

c2=32767;
c1=128;

for i=1:n1
    vn(i)=(c2-c1)*((Txc(i)-n4)/(n3-n4))+c1;
end

s=vn;

  b=double(dec2bin(s,bits));
Txcb=(b==49);
```

Etapa de Encriptación

Programa Para la Generación de Llaves

```
function [llaves] = [illegible] llaves(s1,s2)

bits = 15;

s1 = [0 1 0 0 0 0 0 1 1 0 1 1 1 0 0 1];     %semilla1 escogida por el usuario
s2 = [0 1 0 1 1 1 1 0 0 1 0 0 0 1 1 0];     %semilla2 escogida por el usuario

s1 = fliplr(s1);
s2 = fliplr(s2);

x = 0;

for i = 1:bits
    x(1) = xor(s1(1),s2(2));
    x(2) = xor(xor(s2(1),s1(2)),s2(3));
    x(3) = xor(xor(s1(1),s1(3)),s2(4));
    x(4) = xor(xor(xor(s2(1),s2(3)),s1(4)),s2(5));
    x(5) = xor(xor(xor(xor(s1(1),s2(2)),s1(3)),s1(5)),s2(6));
    x(6) = xor(xor(xor(xor(s2(1),s1(2)),s2(5)),s1(6)),s2(7));
    x(7) = xor(xor(xor(s1(1),s1(5)),s1(7)),s2(8));
    x(8) = xor(xor(xor(xor(s2(1),s2(5)),s2(7)),s1(8)),s2(9));
    x(9) = xor(xor(xor(xor(xor(xor(s1(1),s2(2)),s1(5)),s2(6)),s1(7)),s1(9)),s2(10));
    x(10) = xor(xor(xor(xor(xor(xor(xor(s2(1),s1(2)),s2(3)),s2(5)),s1(6)),s2(9)),s1(10)),s2(11));
    x(11) = xor(xor(xor(xor(xor(xor(s1(1),s1(3)),s2(4)),s1(5)),s1(9)),s1(11)),s2(12));
    x(12) = xor(xor(xor(xor(xor(xor(s2(1),s2(3)),s1(4)),s2(9)),s2(11)),s1(12)),s2(13));
    x(13) = xor(xor(xor(xor(xor(xor(xor(s1(1),s2(2)),s1(3)),s1(9)),s2(10)),s1(11)),s1(13)),s2(14));
    x(14) = xor(xor(xor(xor(xor(xor(s2(1),s1(2)),s2(9)),s1(10)),s2(13)),s1(14)),s2(15));
    x(15) = xor(xor(xor(xor(s1(1),s2(16)),s1(9)),s1(13)),s1(15));

    llave(i,:) = x;
    s2 = [s1 s2(:)];
    s1 = x;
end

llaves = fliplr(llave);
```

PROGRAMA PARA LA ENCRIPTACIÓN

```
function [Txe]=encriptar(Txcb,llaves)

t=Txcb;
x=llaves;
n=length(t(:,1));
m=0;
x=fliplr(x);
t=fliplr(t);

for i=1:n
m(i,1)=xor(xor(xor(xor(x(i,15),x(i,13)),x(i,9)),x(i,1)),t(i,1));
m(i,2)=xor(xor(xor(x(i,14),x(i,10)),x(i,2)),t(i,2));
m(i,3)=xor(xor(xor(xor(x(i,15),x(i,11)),x(i,3)),t(i,1)),t(i,3));
m(i,4)=xor(xor(x(i,12),x(i,4)),t(i,4));
m(i,5)=xor(xor(xor(x(i,5),x(i,13)),t(i,3)),t(i,5));
m(i,6)=xor(xor(xor(x(i,14),x(i,6)),t(i,2)),t(i,6));
m(i,7)=xor(xor(xor(xor(xor(x(i,7),x(i,15)),t(i,1)),t(i,3)),t(i,5)),t(i,7));
m(i,8)=xor(x(i,8),t(i,8)); m(i,9)=xor(xor(x(i,9),t(i,7)),t(i,9));
m(i,10)=xor(xor(x(i,10),t(i,6)),t(i,10));
m(i,11)=xor(xor(xor(xor(x(i,11),t(i,5)),t(i,7)),t(i,9)),t(i,11));
m(i,12)=xor(xor(x(i,12),t(i,4)),t(i,12));
m(i,13)=xor(xor(xor(xor(x(i,13),t(i,3)),t(i,5)),t(i,11)),t(i,13));
m(i,14)=xor(xor(xor(xor(x(i,14),t(i,2)),t(i,6)),t(i,10)),t(i,14));
m(i,15)=xor(xor(xor(xor(xor(xor(xor(xor(x(i,15),t(i,1)),t(i,3)),t(i,5)),t(i,9)),t(i,11)),t(i,13)),
        t(i,15)),t(i,7));
end

Txe=fliplr(m);
```

B.2 Programas de la Síntesis del sistema

Etapa de desencriptación

PROGRAMA PARA LA DESENCRIPTACIÓN

```
function [Txcb] = desencriptar(Txe,llaves)


m=Txe;
x=llaves;
n=length(m(:,1));
t=0;

x=fliplr(x);
m=fliplr(m);

for i=1:n
t(i,1)=xor(xor(xor(xor(x(i,1),x(i,9)),x(i,13)),x(i,15)),m(i,1));
t(i,2)=xor(xor(xor(x(i,2),x(i,10)),x(i,14)),m(i,2));
t(i,3)=xor(xor(xor(xor(xor(xor(x(i,1),x(i,3)),x(i,9)),x(i,11)),x(i,13)),m(i,1)),m(i,3));
t(i,4)=xor(xor(x(i,4),x(i,12)),m(i,4));
t(i,5)=xor(xor(xor(xor(xor(xor(xor(x(i,1),x(i,3)),x(i,5)),x(i,9)),x(i,11)),m(i,1)),m(i,3)),m(i,5));
t(i,6)=xor(xor(xor(xor(x(i,2),x(i,6)),x(i,10)),m(i,2)),m(i,6));
t(i,7)=xor(xor(xor(xor(xor(xor(x(i,1),x(i,5)),x(i,7)),x(i,9)),m(i,1)),m(i,5)),m(i,7));
t(i,8)=xor(x(i,8),m(i,8));
t(i,9)=xor(xor(xor(xor(xor(x(i,1),x(i,5)),x(i,7)),m(i,1)),m(i,5)),m(i,7)),m(i,9));
t(i,10)=xor(xor(xor(xor(x(i,2),x(i,6)),m(i,2)),m(i,6)),m(i,10));
t(i,11)=xor(xor(xor(xor(xor(xor(xor(x(i,1),x(i,3)),x(i,5)),m(i,1)),m(i,3)),m(i,5)),m(i,9)),m(i,11));
t(i,12)=xor(xor(x(i,4),m(i,4)),m(i,12));
t(i,13)=xor(xor(xor(xor(xor(xor(x(i,1),x(i,3)),m(i,1)),m(i,3)),m(i,9)),m(i,11)),m(i,13));
t(i,14)=xor(xor(xor(x(i,2),m(i,2)),m(i,10)),m(i,14));
t(i,15)=xor(xor(xor(xor(x(i,1),m(i,1)),m(i,9)),m(i,13)),m(i,15));
end

Txcb=fliplr(t);
```

Etapa de Transformación Inversa

CONVERSIÓN BINARIO A DECIMAL

```
function [Txc]=texto_decimal(Txcb,n3,n4)

b=Txcb;
n=length(b(:,1));
bits=15;

for i=1:n
    for j=1:15
        if b(i,j)==0
            c(i,j)=48;
        elseif b(i,j)==1
            c(i,j)=49;
        end
    end
end

bb=char(c);
add=(2^(bits-1));

for i=1:n
    c(i,:)=( bin2dec(bb(i,:)) );
end

ww=transpose(c(:,1));
maxww=max(ww);
minww=min(ww);
n3;
n4;
n1=length(ww);

for i=1:n1
    www(i)=(n3-n4)*((ww(i)-minww)/(maxww-minww))+n4;
end

Txc=www;
```

ACONDICIONAMIENTO DE LA SEÑAL

```
function x =acondicionamiento_señal(Txc,ind)

n = length(ind);
ind2=find(ind == 1);
nn =length(Txc);
y4=zeros(1,n);

for i = 1:nn
    y4(ind2(i))=Txc(i);
end

x = y4;
```

TRANSFORMADA ONDELETA DE HAAR INVERSA

```
function y = Thaarinv(Tx2)

n = length(Tx2);
L = 1;
V = Tx2;
cte = 1/sqrt(2);

while L <= n/2
    for k = 1 : L
        V(2*k - 1) = ( Tx2(k) + Tx2(k + L) ) * cte;
        V(2*k) = ( Tx2(k) - Tx2(k + L) ) * cte;
    end
    L = L*2;
    Tx2 = V;
end

y = Tx2;
```

Programa Para Calcular el Error Cuadrático Medio

```
function E=errores2(x,y)

n0=length(x(:,1));


if n0~=1
   z=transpose(x);
   n0=length(z(:,1));
else
   z=x;
end

nn=length(y(:,1));

if nn~=n0
   w=transpose(y);
else
   w=y;
end

E= sum(abs(z-w).^2)/n0;
```

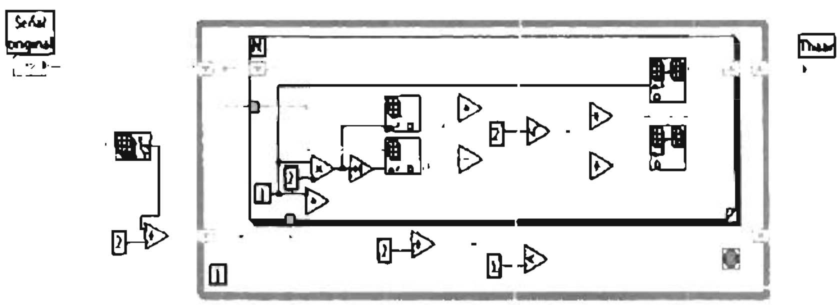
C

Apéndice de listado de programas en LabVIEW

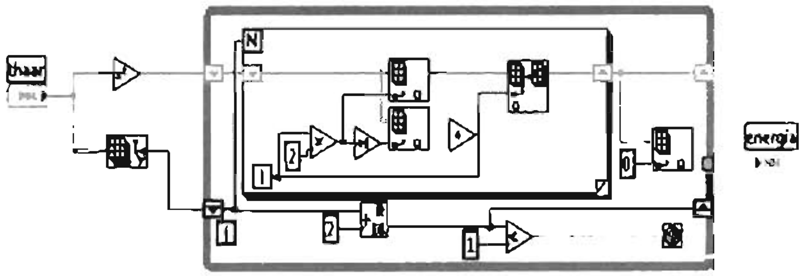
C.1 Programas del Análisis del sistema

Etapa de compresión

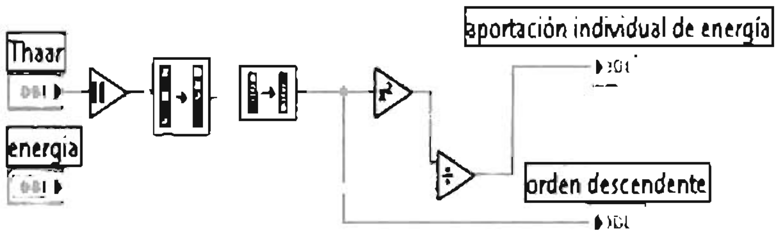
TRANSFORMADA ONDELETA DE HAAR



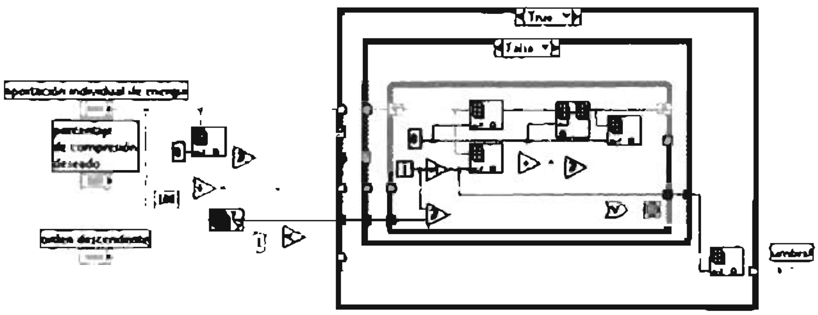
CALCULO DE LA ENERGÍA



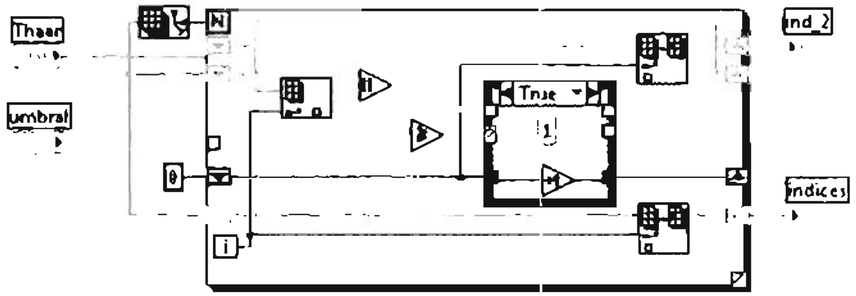
APORTACIÓN INDIVIDUAL DE DE LA ENERGÍA



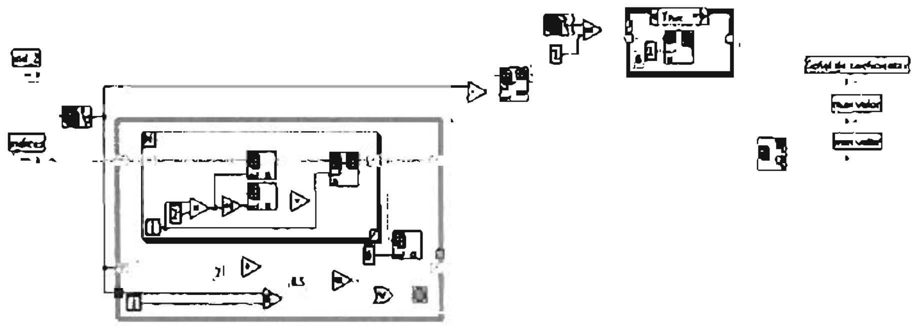
CALCULO DEL UMBRAL



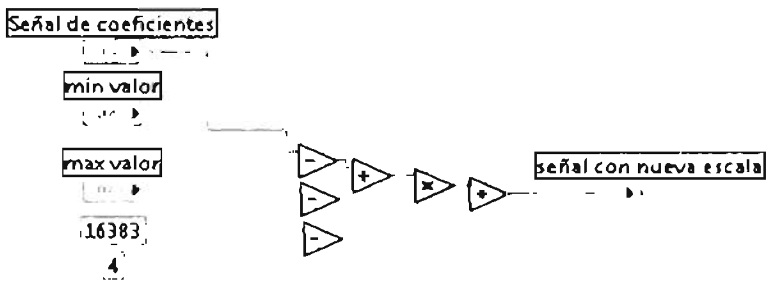
CALCULO DE LOS INDICES



SEÑAL DE COEFICIENTES

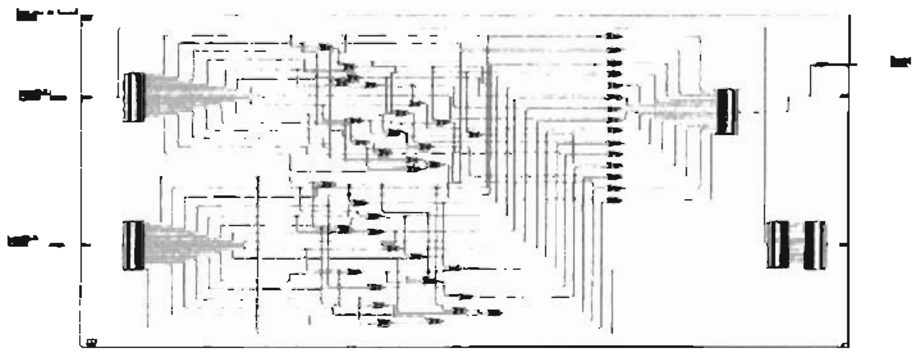


CAMBIO DE ESCALA

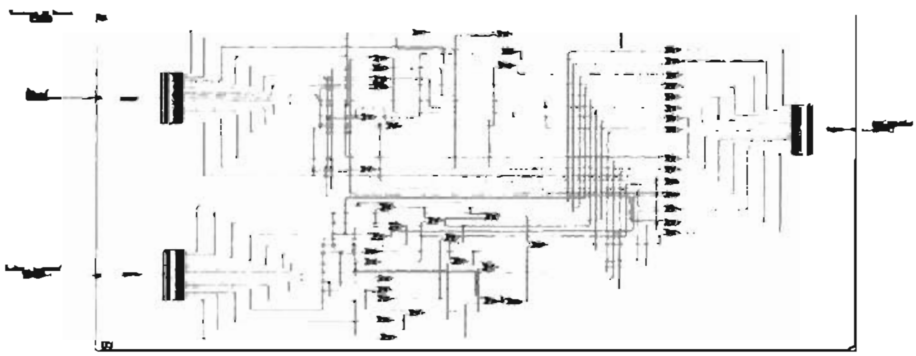


Etapa de encriptación

LLAVES



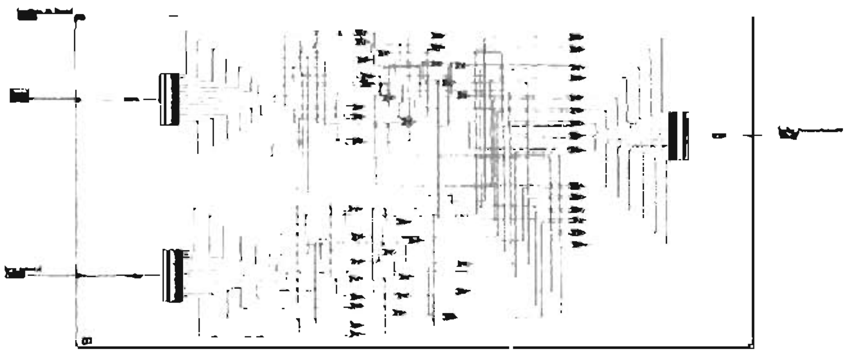
ENCRIPTADOR



C.2 Programas de la Síntesis del sistema

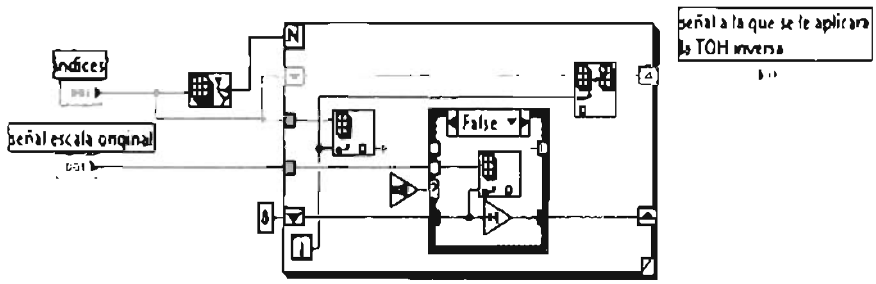
Etapa de desencriptación

DESENCRIPTADOR

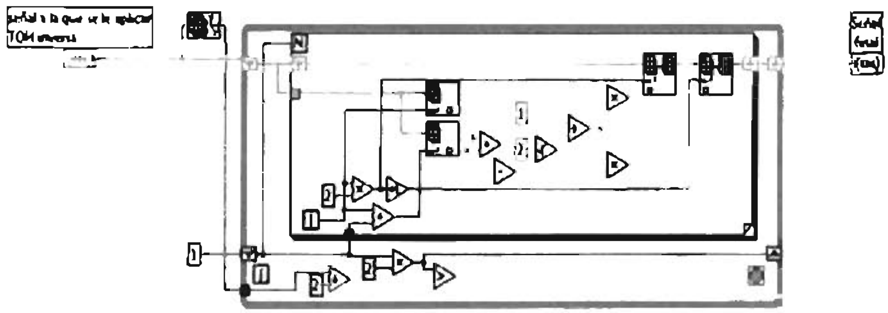


Etapa de transformación inversa

ACONDICIONAMIENTO



TRANSFORMADA ONDELETA INVERSA DE HAAR



D

FPGA

D.1 FPGA (Field Programmable Gate Array)

Un FPGA es un dispositivo que contiene componentes lógicos programables cuyas características pueden ser modificadas, manipuladas o almacenadas mediante programación. Los componentes lógicos programables pueden ser programados para duplicar la funcionalidad de compuertas lógicas básicas o funciones combinacionales más complejas tales como decodificadores o funciones matemáticas simples. La arquitectura de un FPGA consiste en arreglos de varias celdas lógicas las cuales se comunican unas con otras mediante canales de conexiones verticales y horizontales. Cada celda lógica contiene arreglos de compuertas lógicas AND y OR, así como un número definido de registros de multiplexores. La figura D.1 muestra el esquema básico de una FPGA con los bloques lógicos configurables (CLBs), las interconexiones programables y los bloques de entrada/salida (IOB).

Una tendencia reciente ha sido combinar los bloques lógicos e interconexiones de los FPGA con microprocesadores y periféricos relacionados para formar un "Sistema programable en un chip". Para la programación de los FPGA el diseñador cuenta con la ayuda de lenguajes de programación especiales conocidos como HDL o Hardware Description Language (lenguajes de descripción de hardware). Los HDLs más utilizados son:

- VHDL
- Verilog
- ABEL

En un intento de reducir la complejidad y el tiempo de desarrollo en fases de prototipaje rápido,

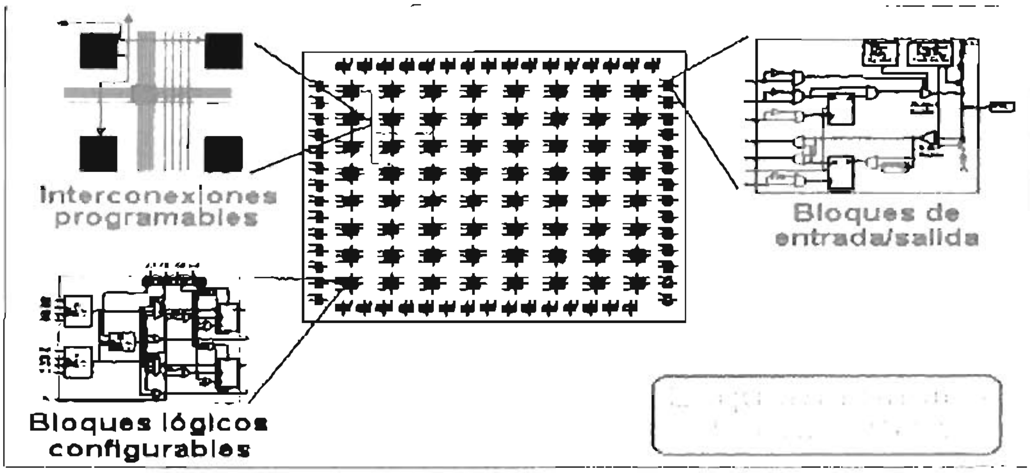


Figura D.1: Esquema básico de una FPGA.

existen varias propuestas y niveles de abstracción del diseño, National Instruments por medio de LabVIEW FPGA propone un acercamiento de programación gráfica de alto nivel.

El FPGA utilizado en esta tesis se encuentra en la tarjeta PCI-7811R de National Instruments, la cual es mostrada en la figura D.2. Esta tarjeta cuenta con 160 líneas configurables como entradas, salidas, contadores o lógica custom con velocidades de hasta 40 MHz, además con 1 millon de de posibles combinaciones de entradas o salidas digitales para procesado paralelo. El chip FPGA de la tarjeta es configurado en LabVIEW para crear diagramas de bloque con el módulo LabVIEW FPGA. Los diagramas de bloque se ejecuta en el hardware, teniendo un control inmediato sobre todas las señales entrada-salida.

La tabla D.1 muestra algunas especificaciones de la tarjeta PCI-7811R.

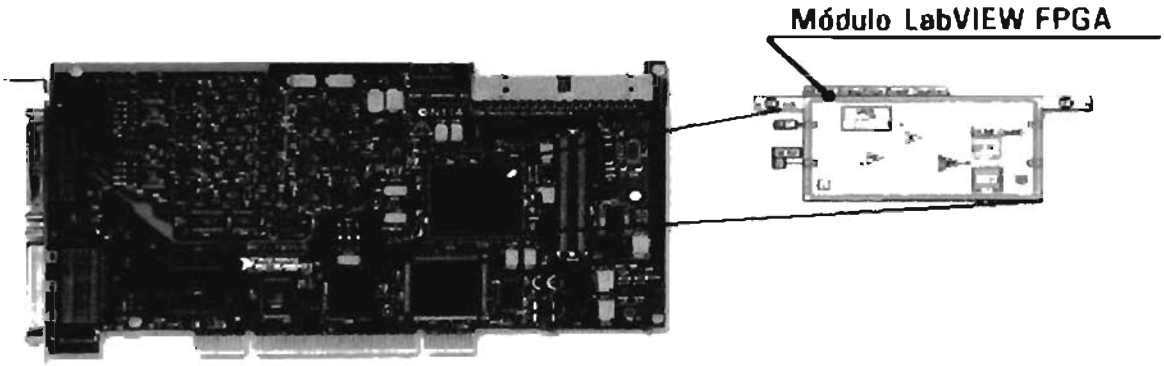


Figura D.2: Tarjeta de adquisición de datos PCI-7811R de National Instruments.

General	
Tipo de bus	pci
Soporte para SO	Windows, Real-Time, RTX
Familia de productos	Serie R
Compatible con tiempo real	Control determinístico de un solo punto Prueba robusta y crítica
Disparo	Digital
E/S Digitales	
Número de Canales	160 DIO
Temporización	Estático, Temporizado por hardware (< 10 MHz) Temporizado por hardware (>= 10 MHz)
Niveles Lógicos	TTL
Máximo Rango de Entrada	0.5 V
Máximo Rango de Salida	0.5 V
Entrada de Flujo de Corriente	Sinking, Sourcing
Filtros de Entrada Programables	Si
Salida de Flujo de Corriente	Sinking, Sourcing
Capacidad de Corriente (Canal/Total)	5 mA/0.8 A
Temporizador Watchdog	Si
Estados de Encendido Programables	Si
Protocolo de Sincronización de E/S	Si
Patrón de E/S	Si
Contadores/Temporizadores	
Número de Contadores/Temporizadores	160
Resolución	64 bits
Frecuencia Máxima de la Fuente	80 MHz
Entrada Mínima de Ancho de Pulso	12.5 ns
Niveles Lógicos	TTL
Rango Maximo	0.5 V
Estabilidad de Tiempo	100 ppm
Sincronización GPS	No
Generación de Pulso	Si
Operaciones a Búfer	Si
Eliminación de Rebotes	No
Número de Canales DMA	3

Tabla D.1: Resumen de Especificaciones de la Tarjeta PCI-7811R.

Bibliografía

[1] José Manuel Alarcón Roldán, *Compresión de Imágenes y Video Mediante la Transformada Wavelet.*(2005).

[2] Carlos Almaguer López, Tesis de Maestría *Construcción y evaluación de un sistema digital para encriptación.*(2007).

[3] José de Jesús Angel Angel, *Criptografía Para Principiantes.*Versión 1.0.SeguriDATA (México).*http : //www.criptored.upm.es/guiateoria/gt_m117e.htm*

[4] M. Bianco and D. Reed , *Encryption system based on chaos theory*, US patent number 5,048,086.

[5] Howard A. Gutowitz, *Method and apparatus for encryption and authentication using dynamical systems*, US patent number 5,365,589.

[6] Luis Hernández Encinas, Angel Martín del Rey e Isabel Visus ruiz, *Cifrador de imagenes en tonos de grises mediante autonomas celulares.*

[7] Sara Hoya White, *Aplicaciones de los Autómatas Celulares a los Criptosistemas de Cifrado en Flujo.*Tercera edicion (2002).

[8] Huriel Hurtado Partida, *Compresión de Señales de Voz con la Transformada Ondeleta.*(2006).

[9] Song-Ju Kim and Ken Umeno, *Randomness Evaluation and Hardware Implementation of Nonadditive CA-Based Stream Cipher*, J. Signal Process (2005), vol 9 No.1, page 71-77.

[10] LAFE, Olurinde, E., *Method and apparatus for information processing using cellular automata transform*, International Patent number WO 97/12330.

[11] D. David Liñán Zayas *Tutorial de PGP.*Madrid, Julio de (1999).*http : //www.criptored.upm.es/software/sw_m001g.htm*

[12] Manuel José Lucena López, *Criptografía y Seguridad en Computadores.*Tercera edicion (marzo de 2002).

[13] Stéphane Mallat, A Wavelet Tour of Signal Processing, 2nd. Edition, Academic Press, 1999.

[14] Marcela Mejía Carlos, *Encriptación por Sincronización en Autómatas Celulares*.(2001)

[15] M. Mejía and J. Urías, *An Asymptotically Perfect Pseudorandom Generator*, Discrete and Continuos Dynamical Systems, **7**, 115-126 (2001).

[16] A. J. Menezes, Paul C. van Oorschot and Scott A. Vanstone, *Handbook of applied cryptography*, CRC Press, NY 1997.

[17] Miguel Morales Sandoval, *Arquitectura Hardware de un Criptosistema de Curva Elíptica con Compresión de Datos*.

[18] José Salomé Murguía Ibarra, Tesis de Maestría *Tratamiento Multiresolución de Señales e Imágenes con Ondeletas de Haar*.(1999).

[19] Vladimir A. Protopopescu, Robert T. Santoro and S. Tolliver, *Fast and secure encryption-decryption method based on chaotic dynamics*, US patent number 5,479,513.

[20] Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, Elaine Barker, Stefan Leigh, Mark Levenson, Mark Vangel, David Banks, Alan Heckert, James Dray, San Vo, *A statistical test suite for random and pseudorandom number generators for cryptographic applications*. (2001).

[21] Gilbert Strang and T. Nyugen, *Wavelets and Filter Banks*, Prentice Hall, Englewood Cliffs NJ, 1996.

[22] J. Urías, *Cryptography primitives based on a cellular automaton*, appeared in Coding Theory, cryptography and related areas (J. Buchmann *et al.*; eds.), Springer, NY (2000).

[23] J. Urías, G. Salazar and E. Ugalde, *Synchronization of cellular automaton pairs*, Chaos, **8**, 814–818 (1998).

[24] Martin Vetterli and Jelena Kovacevic, *Wavelets and subband coding*, Prentice Hall, Englewood Cliffs NJ, 1995.