Manipulación de imágenes médicas sobre campos finitos

Tesis que presenta
José Francisco Rodríguez arellano
Para obtener el grado de
Maestro en Ciencias en Ingeniería Biomédica

Asesores:
M. en C. Óscar Yáñez Suárez
Dr. Ricardo Marcelín Jiménez

Jurado Calificador:
Presindete: M. en C. Óscar Yáñez Suárez
Secretario: Dr. Adriano de Luca Penachia
Vocal: Dr. Miguel Ángel Ruiz Sánchez

14 de enero de 2011

“Existen cosas que simplemente se tienen que madurar.”

Dr.Ricardo Marcelin Jimenez

Agradecimientos

Quiero agradecer, a conacyt por el apoyo brindado en mis estudios de maestria, a mi familia y a Alejandra Guillen.

Igualmente quiero agradecer a la comunidad T-330, en especial a mis amigos Aldo Mejia y Ronald Arias.

Francisco Rodríguez

Resumen

Este trabajo propone un esquema de almacenamiento y recuperación de imágenes basado en un sistema distribuido tolerante a fallas en los dispositivos de almacenamiento.

Consta de un algoritmo de compresión sin pérdida de información basado en aritmética entera, el cual aprovecha la reducción de entropía generada por filtros wavelets descritos en aritmética de campo finito, así como un algoritmo de dispersión de datos también sobre campo finito, todo ello implementado en hardware reconfigurable (FPGA).

El resultado es un sistema de hardware y software aplicado a imágenes, el cual realiza de manera rápida, segura y eficiente el almacenamiento y recuperación de imágenes en un sistema distribuido. La rapidez de este esquema se debe a la implementación en hardware reconfigurable de los diferentes algoritmos que lo conforman. La seguridad y la eficiencia se deben a que la información almacenada no consiste simplemente en replicas de los archivos originales comprimidos, sino en nuevos archivos generados a partir de una transformación matemática aplicada sobre los datos resultantes de la compresión sin pérdida, mejorada a su vez por la reducción de entropía debida al previo filtrado con wavelets. De esta manera, los archivos almacenados no pueden simplemente abrirse y leerse como cualquier archivo, es necesario emplear el esquema de reconstrucción y descompresión para obtener información que tenga sentido.

Contenido

Lista de Figuras

Lista de Tablas

Capítulo 1
Introducción

1.1. Contexto

La introducción de tecnología digital y de comunicaciones al ambiente hospitalario aporta importantes ventajas, como la manipulación de imágenes digitales para un mejor diagnóstico o la capacidad de consultar remotamente los archivos de los pacientes [1]. La disponibilidad de equipo médico digital ha dado lugar a la generación de un nuevo concepto denominado PACS (Picture Archiving and Communication System), que es un sistema de comunicación y almacenamiento de imágenes. Esta tecnología consiste de dispositivos de adquisición de imágenes, unidades de almacenamiento, estaciones de despliegue, procesadores y bases de datos.

Sin embargo, con la imagenología digital surgen también nuevos retos. Un estudio de ultrasonido Doppler de 10 segundos de duración, por ejemplo, puede producir un archivo de imagen de 255 MB. La cantidad de espacio de almacenamiento requerido es muy alta si se considera el número de estudios de ultrasonido que se genera a diario en un hospital. La inclusión del resto de las imágenes que se producen en todo el departamento de imagenología en este recuento lleva a tomar consideraciones especiales para la transmisión y el almacenamiento de toda esta información. Una posible solución a estos problemas es el uso de técnicas de compresión de datos e imágenes. Puesto que esta información es utilizada para el diagnóstico, tratamiento y seguimiento de la salud de seres humanos, su calidad y confiabilidad son cruciales. Por lo tanto, es necesario garantizar que la manipulación de las imágenes médicas no alterará de ninguna manera la información original.

En el ámbito de la imagenología médica es vital asegurar que los procesos de compresión-descompresión, almacenamiento y distribución de las imágenes se realicen sin pérdidas. La compresión de imágenes sin pérdida se puede lograr al transformar o descomponer la imagen en subbandas mediante un banco de filtros y posteriormente codificar aritméticamente los componentes resultantes. Es posible encontrar una estructura de descomposición de imágenes cuya implementación en hardware dedicado sea muy eficiente tanto en tiempo de compresión como en área utilizada [2].

Para conseguir que la compresión sea sin pérdida se debe garantizar que la inversión no sea afectada por errores de redondeo o de saturación. Lo anterior sugiere utilizar campos y aritméticas distintos a los números reales o complejos. Dichos campos se deben

seleccionar cuidadosamente, ya que sus características pueden conducir a problemas como la necesidad de utilizar cada vez más bits para representar los valores de la imagen en pasos sucesivos de descomposición, como sucede con los anillos de característica cero [3]. Una forma de evitar este problema es emplear aritmética de campos finitos. Estas estructuras computacionales son ampliamente utilizadas para códigos de corrección de errores o criptografía [4]. La compresión de imágenes binarias con la ayuda de bancos de filtros sobre el campo finito con dos elementos fue propuesta por Swanson y Tewfik [5].

La aritmética en un campo finito de característica 2 es fundamentalmente una aritmética modular, es decir, la adición se realiza bajo la aritmética módulo 2, por lo que la suma de dos polinomios se convierte simplemente en la operación XOR bit a bit de sus representaciones binarias. La sustracción es exactamente igual que la adición en la aritmética módulo 2. La multiplicación es la operación aritmética más importante -puesto que operaciones avanzadas como la exponenciación o la inversión están basadas en ellapero no la más sencilla, por lo que existen diferentes algoritmos que resuelven el problema de la implementación.

Además de garantizar que la calidad de la información no se pierda con la compresión, es necesario asegurar que los archivos se encuentran almacenados de manera confiable, es decir, que no se perderán si ocurre una falla en el sistema. Una de las estrategias más comunes consiste en la generación de múltiples copias o respaldos que se guardan en varios sitios diferentes. De esta manera, si alguno falla se puede recuperar la información completa de otro. Sin embargo, este esquema genera otro problema: por cada copia respaldada, se multiplica el espacio de almacenamiento requerido, efecto conocido como redundancia. Aunque el archivo haya sido comprimido antes de generar las copias de respaldo, la ganancia conseguida con la tasa de compresión se ve opacada por la redundancia en el almacenamiento.

1.2. Contribución

En el presente trabajo se propone una solución eficiente al problema de la compresión sin pérdida de imágenes médicas, mediante la implantación física en lógica reconfigurable de un algoritmo de compresión que utiliza filtros basados en wavelets sobre campos finitos. Asimismo, para resolver el problema del exceso de redundancia que plantea el esquema empleado tradicionalmete para el almacenamiento confiable, se propone que el archivo comprimido sea dispersado, mediante el algoritmo de Rabin sobre campos finitos [6], en subarchivos que serán almacenados en un sistema distribuido. Este tipo de sistemas consiste en una colección de computadoras independientes que aparecen ante los usuarios del sistema como una única computadora. Las principales ventajas que se obtienen de trabajar con estos sistemas son que permiten la ejecución concurrente de procesos (lo que agiliza el almacenamiento y transmisión de archivos pesados) y son poco propensos a fallas [7].

El dispositivo resultante (Manipulador) ejecuta todas las acciones involucradas en el almacenamiento y recuperación de la imagen, así como el enlace directo con el sistema

de almacenamiento distribuido instalado en el hospital. En la figura [1.1] se muestran las etapas de que constan la compresión, la dispersión y el almacenamiento distribuido.

La innovación de la presente propuesta radica en que, al almacenar cada pieza en una terminal -computadora- diferente dentro del sistema distribuido, se asegura que el almacenamiento sea tolerante a fallos. Si por alguna razón ajena a la transmisión suceden dos fallos, es decir, se pierden cualesquiera 2 de los dispersos, la imagen se puede recuperar sin pérdida de información a partir de las 3 piezas restantes.

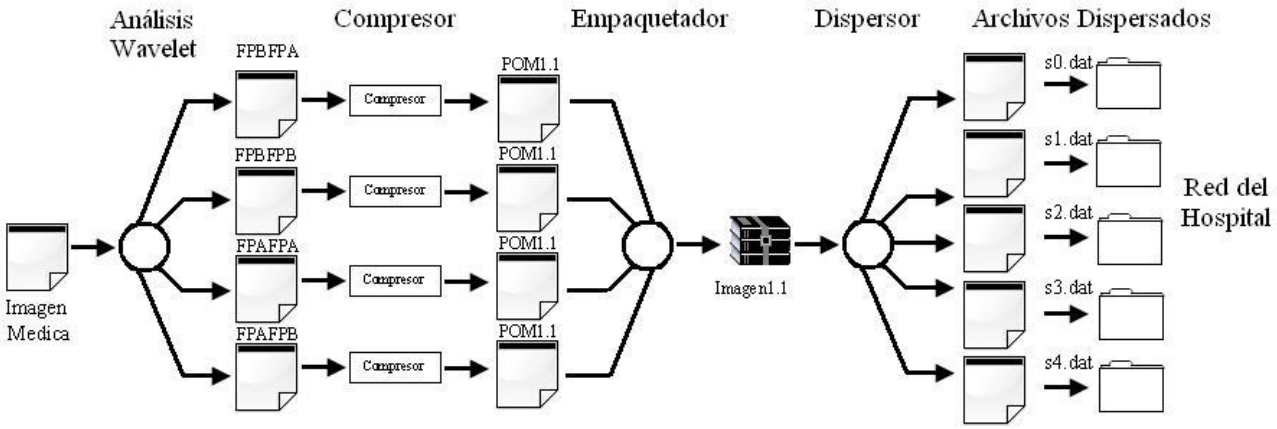


Figura 1.1: *Secuencia propuesta para la compresión y almacenamiento. La imagen es transformada en cuatro subbandas, las cuales son comprimidas y concatenadas en un sólo archivo que es dispersado en cinco subarchivos llamados dispersos. Los dispersos se almacenan en diferentes computadoras del sistema distribuido del hospital.*

En la figura [1.2] encontramos las etapas de que consta la recuperación de la imagen almacenada. En primer lugar es necesario recuperar cualesquiera tres de los cinco archivos dispersos almacenados en el sistema distribuido del hospital. A continuación, se aplica el algoritmo reconstructor de Rabin, a partir del cual se obtiene un solo archivo. El siguiente paso es desempaquetar este archivo para obtener las cuatro subbandas comprimidas, las cuales deben descomprimirse antes de ser sintetizadas con el banco de filtros para formar la imagen original.

1.3. Metodología

El desarrollo del presente proyecto requirió del manejo de múltiples conceptos y herramientas matemáticas, tales como álgebra de campos finitos, filtrado digital, compresión basada en probabilidades y álgebra lineal, involucrados en los diferentes algoritmos de que está compuesto el manipulador propuesto.

Dichos algoritmos fueron programados en lenguaje C++ para someterlos a pruebas y simulaciones que proporcionaron una visión panorámica de su comportamiento individ-

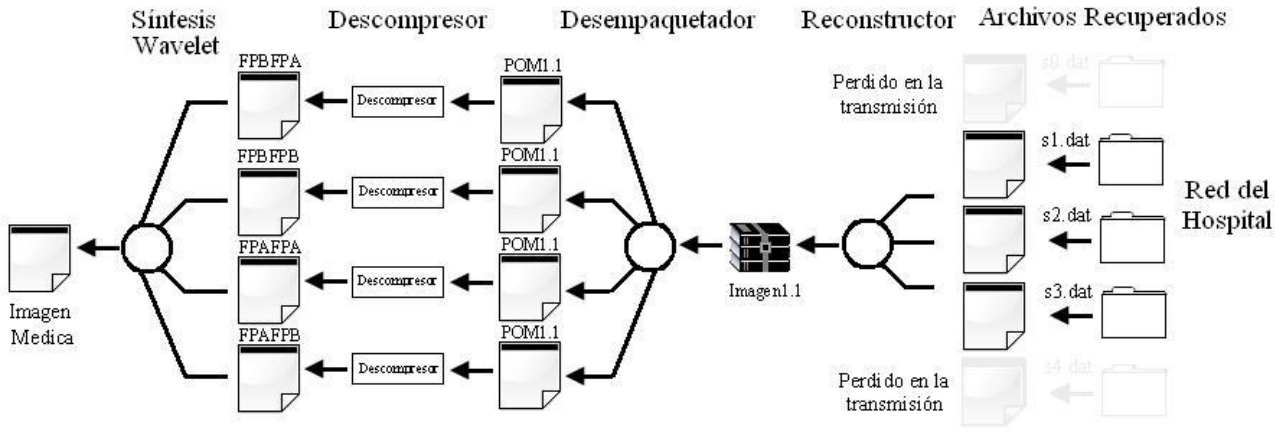


Figura 1.2: *Secuencia propuesta para la recuperación de la imagen original. Se recuperan tres dispersos, con los cuales se reconstruye el archivo empaquetado, desempaquetados, descomprimidos y sintetizados para formar la imagen final.*

ual y en conjunto. Al conocer cómo se comportarían los diferentes algoritmos al trabajar sobre imágenes médicas reales, fue posible proponer una arquitectura en hardware eficiente.

Para implementar en hardware los algoritmos seleccionados y probados fue necesario traducirlos a un lenguaje de descripción de alto nivel. En este proyecto empleamos el lenguaje VHDL. El hardware sobre el cual se construyó el manipulador es un FPGA (Field Programmable Gate Array) Xilinx de la familia Spartan 3E, con un equivalente de 500,000 compuertas, que se encuentra incorporado en una tarjeta de prueba fabricada por Digilent[8].

Dado que el dispositivo es autónomo, lo que le da independencia y portabilidad, se necesita una interfaz rápida y confiable para enlazarlo con cualquier computadora que contenga la imagen a manipular y en la que será almacenado el resultado del procesamiento (dispersos). La interfaz empleada originalmente fue el protocolo USB 1 (Universal Serial Bus 1.0), gestionado por el microcontrolador PIC16C765 de Microchip. Debido a la demanda de mayor velocidad, se escaló a USB 2, por lo que fue necesario cambiar al microcontrolador PIC18F4550.

Para que el personal médico capacitado controle el dispositivo, se diseño una interfaz de usuario en lenguaje C++ utilizando, el entorno de desarrollo C++ Builder[9].

Se evaluó el desempeño del sistema manipulando imágenes de resonancia magnética cerebral de 4kb de tamaño y en escala de grises (64x64 pixeles de 8 bits). Las variables consideradas para el análisis del funcionamiento fueron el espacio de almacenamiento requerido para los dispersos y la concordancia byte a byte entre la imagen original y la imagen recuperada después de la manipulación.

1.4. Estructura de la tesis

- **Capítulo 1: Introducción.** En el que se presenta de manera resumida el tema de la investigación, su contexto, importancia y contribución, así como la metodología seguida para su desarrollo y evaluación.
- **Capítulo 2: Objetivos.** Contiene el listado de los objetivos general y específicos a cumplir.
- **Capítulo 3: Fundamentos.** En él se encuentran los principales conceptos teóricos en los cuales se basan los diferentes algoritmos y técnicas de procesamiento de imágenes empleados durante el desarrollo del presente trabajo.
- **Capítulo 4: Manipulador de Imágenes Médicas.** En este capítulo se describe el proceso de diseño, programación y construcción del dispositivo propuesto. Se detallan las diferentes partes que lo constituyen, el funcionamiento esperado de cada una y las interconexiones entre ellas. Además, se ofrecen las simulaciones que avalan su correcto desempeño, tanto a nivel individual como integrado.
- **Capítulo 5: Desempeño del Manipulador de Imágenes Médicas.** Contiene los resultados de la manipulación, empleando el dispositivo desarrollado, de un banco de imágenes de resonancia magnética craneal, así como la evaluación de los factores cuantitativos y cualitativos relacionados con el desempeño.
- **Capítulo 6: Conclusiones.** Donde se presentan las principales contribuciones de la investigación, sus limitaciones y perspectivas para trabajo futuro.

Capítulo 2
Objetivos

2.1. Objetivos del proyecto

2.1.1. Objetivo general

Diseñar y construir en hardware reconfigurable un sistema digital (manipulador) que realice la compresión y dispersión sin pérdidas de imágenes médicas, así como la reconstrucción y descompresión de las mismas, a partir de la consideración de que éstas son muestras aleatorias de un campo finito.

2.1.2. Objetivos Particulares

- Diseñar y describir combinacionalmente en hardware reconfigurable:
  - Una unidad aritmética lógica de campo finito $GF(2^8)$, basada en el campo generado por el polinomio primitivo $f(\alpha) = \alpha^8 + \alpha^6 + \alpha^5 + \alpha^4 + 1$.
  - El banco de filtros digitales requerido por la transformada wavelet y su antitransformada, utilizando las operaciones aritméticas sobre el mismo campo finito, para descomponer la imagen en subbandas con menor entropía y así maximizar su tasa de compresión.
  - Un compresor y su correspondiente descompresor basado en la codificación aritmética con números enteros.
  - Un dispersor y un reconstructor de Rabin, empleando la aritmética asociada al mismo campo finito $GF(2^8)$.
- Evaluar el desempeño del manipulador de imágenes médicas en función del espacio de almacenamiento requerido para los dispersos y de la integridad de la información recuperada.

Capítulo 3
Fundamentos

3.1. Campos Finitos

Se denomina *campo finito* al conjunto de números enteros módulo p cerrado bajo las operaciones de suma, resta, multiplicación y división (esta última siempre que el divisor sea distinto de cero), denotado por $\mathbb{Z}_p$ [10]

$$\mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z} = \{\overline{0}, \overline{1}, \overline{2}, \cdots, \overline{p-1}\} \tag{3.1}$$

donde p es un número primo.

Cada elemento del campo es representante de una clase [11], la cual es inducida por una relación de equivalencia:

$$a \sim b \ \ (a \ esta \ \ relacionado \ \ con \ \ b) \ \ si \ \ a - b = kp, \ \ donde \ \ a, b, k, p \epsilon \mathbb{Z} \tag{3.2}$$

Se acostumbra denotar la relación de equivalencia como :

$$a \equiv b \ \ mod \ \ p \iff a - b = kp \tag{3.3}$$

que se lee:

a **es congruente con** b **módulo** p **si y sólo si su diferencia es un múltiplo de** p. Es importante notar que se genera un campo de cardinalidad p, es decir, que contiene p elementos.

Definición. *Para un primo* p*, sea* $\mathrm{F}_p = 0, 1, 2, \cdots, p-1$ *y*

$$\psi : \mathbb{Z}/p\mathbb{Z} \mapsto \mathrm{F}_p, \quad \psi(\overline{a}) = a \quad para \quad a = 0, 1, 2, \cdots, p-1 \tag{3.4}$$

$\mathbb{F}_p$ obtiene la estructura de campo finito através del mapeo ψ, y es llamado **campo de Galois de orden p** en honor a Evariste Galois. Dicho campo suele denotarse $GF(p)$ (Galois Field, por sus siglas en inglés).

Existen tantos campos finitos como números primos. Es posible generar nuevos conjuntos de números enteros que satisfagan todas las propiedades anteriores, pero con un mayor número de elementos, conocidos como *extensiones de campos finitos*. La elección del campo depende enteramente del problema a resolver, siendo las extensiones del campo F_2, representadas como F_{2^n}, las que tienen mayor importancia en las aplicaciones informáticas.

3.1.1. Extensión del campo F_2

Durante los últimos años las extensiones del campo F_2 se han utilizado en un gran número de aplicaciones. Entre ellas se pueden mencionar la detección y corrección de errores en la transmisión de información [12], los esquemas criptográficos [13] y el procesamiento de señales [14], etc.

Representaremos por $\mathrm{F}_2[x]$ al conjunto de todos los polinomios de la forma $\mathrm{a}_0 + \mathrm{a}_1 x + \cdots + \mathrm{a}_n x^n$, donde n puede ser cualquier entero no negativo y donde los coeficientes $\mathrm{a}_0, \mathrm{a}_1, \mathrm{a}_2, \cdots, \mathrm{a}_n$ están todos en F_2. Estos polinomios son elementos del campo $\mathbb{F}_2$ en la indeterminada x. Se sabe que si encontramos un polinomio $f(x)$ irreducible de grado n en el campo F_2 podemos construir el campo de las clases residuales módulo el polinomio $f(x)$, que por esta razón se conoce como **polinomio generador de campo**.

$$\frac{\mathrm{F}_2[x]}{< f(x) >} = \mathrm{a}_0 + \mathrm{a}_1 x + \cdots + \mathrm{a}_{n-1} x^{n-1} + < f(x) >: \mathrm{a}_i \epsilon \mathrm{F}_2 \tag{3.5}$$

Con ello podemos generar una extensión K de F_2, en donde cada elemento distinto de cero tiene un inverso multiplicativo y que contiene 2^n elementos. Es importante aclarar que, en esta extensión del campo, el polinomio $f(x)$ tiene una raíz; es decir, existe un elemento en el campo K que es algebraico sobre el campo F_2 [15].

A este campo de clases residuales se le llama **extensión del campo binario**.

$$\mathrm{F}_{2^n} = \{\mathrm{a}_0 + \mathrm{a}_1 x + \cdots + \mathrm{a}_{n-1} x^{n-1}\}, \mathrm{a}_i \epsilon \mathrm{F}_2 \tag{3.6}$$

La extensión generada es isomorfa a F_2^n, el producto cartesiano de F_2 consigo mismo n veces. Por lo tanto, se le puede dar la estructura de campo vectorial, lo que se traduce en que el nuevo campo contenga todos los vectores con n componentes.

$$\mathbb{F}_2^n = \{\mathbf{a} = (\mathrm{a}_{\mathbf{0}}, \mathrm{a}_{\mathbf{1}}, \cdots, \mathrm{a}_{\mathbf{n-1}}) : \mathrm{a}_{\mathbf{i}} \epsilon \mathbb{F}_{\mathbf{2}}\} \tag{3.7}$$

Si consideramos a los pixeles de una imagen como muestras aleatorias de un campo finito de elementos, podemos determinar con certeza la extensión del campo base necesaria. En las imágenes monocromáticas cada pixel es representando por un byte (8 bits), es decir, cada pixel puede tomar un valor de entre 256 elementos de la escala de grises, por lo que requieren un campo F_{2^8} para su representación.

3.1.2. Construcción del campo extensión F_{2^8}

Para construir un campo F_{2^8} es necesario proponer un polinomio generador. La elección del mismo determinará la complejidad de la multiplicación e impactará sobre la descripción en hardware. Por cuestiones de diseño se optó por utilizar el polinomio primitivo $f(x) = x^8 + x^6 + x^5 + x^4 + 1$ sobre el campo F_2 publicado en [16], ya que nos permite recuperar todos los elementos distintos de cero a partir de las potencias del elemento primitivo α, que es raíz de dicho polinomio.

Si establecemos la relación de equivalencia

$$\alpha \equiv x \ \ mod \ \ f(x) \tag{3.8}$$

podemos escribir:

$$f(\alpha) = \alpha^8 + \alpha^6 + \alpha^5 + \alpha^4 + 1 \tag{3.9}$$

Obtenemos las clases de equivalencia módulo $f(\alpha)$

$$\frac{\mathbb{F}_2[\alpha]}{< f(\alpha) >} = \mathrm{a}_0 + \mathrm{a}_1\alpha + \cdots + \mathrm{a}_7\alpha^7 + < f(\alpha) >: \mathrm{a}_i \epsilon \mathbb{F}_2 \tag{3.10}$$

Explícitamente el campo F_{2^8} es isomorfo como espacio vectorial a F_2^8 a través de

$$\varphi : \mathrm{F}_{2^8} \mapsto \mathrm{F}_2^8, \varphi(\mathrm{a}_0 + \mathrm{a}_1\alpha + \cdots + \mathrm{a}_7\alpha^7) = (\mathrm{a}_0, \mathrm{a}_1, \mathrm{a}_2, \mathrm{a}_3, \mathrm{a}_4, \mathrm{a}_5, \mathrm{a}_6, \mathrm{a}_7) \tag{3.11}$$

El campo F_{2^8} contiene 256 elementos, es decir, tiene cardinalidad 256. Los elementos del campo, menos el cero, forman un grupo cíclico de orden 255 bajo la operación de multiplicación. En la siguiente tabla mostramos sólo una sección del campo generado (por razones de espacio).

Representación vectorial F_2^8		Representación polinomial F_{2^8}		Representación exponencial
(0,0,0,0,0,0,0,0)	$\leftrightarrow$	0	$\leftrightarrow$	$-$
(0,0,0,0,0,0,0,1)	$\leftrightarrow$	1	$\leftrightarrow$	α^0
(0,0,0,0,0,0,1,0)	$\leftrightarrow$	α	$\leftrightarrow$	α^1
(0,0,0,0,0,1,0,0)	$\leftrightarrow$	α^2	$\leftrightarrow$	α^2
(0,0,0,0,1,0,0,0)	$\leftrightarrow$	α^3	$\leftrightarrow$	α^3
(0,0,0,1,0,0,0,0)	$\leftrightarrow$	α^4	$\leftrightarrow$	α^4
(0,0,1,0,0,0,0,0)	$\leftrightarrow$	α^5	$\leftrightarrow$	α^5
(0,1,0,0,0,0,0,0)	$\leftrightarrow$	α^6	$\leftrightarrow$	α^6
(1,0,0,0,0,0,0,0)	$\leftrightarrow$	α^7	$\leftrightarrow$	α^7
(0,1,1,1,0,0,0,1)	$\leftrightarrow$	$\alpha^6 + \alpha^5 + \alpha^4 + 1$	$\leftrightarrow$	α^8
(1,1,1,0,0,0,1,0)	$\leftrightarrow$	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha$	$\leftrightarrow$	α^9
(1,0,1,1,0,1,0,1)	$\leftrightarrow$	$\alpha^7 + \alpha^5 + \alpha^4 + \alpha^2 + 1$	$\leftrightarrow$	α^{10}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
(0,0,1,0,1,1,1,0)	$\leftrightarrow$	$\alpha^5 + \alpha^3 + \alpha^2 + \alpha$	$\leftrightarrow$	α^{252}
(0,1,0,1,1,1,0,0)	$\leftrightarrow$	$\alpha^6 + \alpha^4 + \alpha^3 + \alpha^2$	$\leftrightarrow$	α^{253}
(1,0,1,1,1,0,0,0)	$\leftrightarrow$	$\alpha^7 + \alpha^5 + \alpha^4 + \alpha^3$	$\leftrightarrow$	α^{254}
(0,0,0,0,0,0,0,1)	$\leftrightarrow$	1	$\leftrightarrow$	α^{255}

Cuadro 3.1: *Representación vectorial, polinomial y exponencial del campo* F_{2^8}

La representación vectorial es muy útil para almacenar y simplificar el manejo de los elementos del campo finito. La representación exponencial se utiliza para aritmética rápida.

Sobre este campo se realizará la manipulación de imágenes médicas en escalas de gris.

3.1.3. Operaciones en el campo F_{2^8}

Por la definición del campo, sobre él deben existir dos operaciones: suma y multiplicación. Debido a la complejidad que conlleva implementar en hardware la operación de multiplicación, se han propuesto varios algoritmos que difieren entre sí por el manejo que hacen del compromiso generado entre velocidad y espacio de hardware empleado [17].

Multiplicación

La operación de multiplicación está definida de la siguiente manera:

Si $p(\alpha) = \mathrm{a}_0 + \mathrm{a}_1\alpha + \cdots + \mathrm{a}_7\alpha^7$ y $q(\alpha) = \mathrm{b}_0 + \mathrm{b}_1\alpha + \cdots + \mathrm{b}_7\alpha^7$ están ambos en F_{2^8} entonces $p(\alpha)q(\alpha) = \mathrm{c}_0 + \mathrm{c}_1\alpha + \cdots + \mathrm{c}_7\alpha^7$ donde $\mathrm{c}_t = \mathrm{a}_t\mathrm{b}_0 + \mathrm{a}_{t-1}\mathrm{b}_1 + \mathrm{a}_{t-2}\mathrm{b}_2 + \cdots + \mathrm{a}_0\mathrm{b}_t$, donde t va de 0 a 7.

Dado que el campo $\mathrm{F}_{2^8} - \{0\}$ es un grupo cíclico de orden 255 generado por $< \alpha >$, podemos representar a los polinomios $p(\alpha)$ y $q(\alpha)$ como potencias de α, convirtiendo la operación de multiplicación en una suma de exponentes.

Ejemplo 3.1

	Representación polinomial	Representación exponencial
$p(\alpha)$	$\alpha^6 + \alpha^5 + \alpha^4 + 1$	$\alpha^8 \cdot \alpha^1 = \alpha^9$
$q(\alpha)$	$\times \qquad \alpha$	
$p(\alpha) \cdot q(\alpha)$	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha$	

Cuadro 3.2: *Multiplicación polinomial y exponencial en el campo* F_{2^8}

En ocasiones esta operación genera como resultado un elemento que no pertenece al campo, es decir, elementos polinomiales de grado mayor al grado más alto del polinomio primitivo.

Para que el campo sea cerrado bajo la multiplicación, es necesario que el polinomio primitivo satisfaga una relación algebraica: si $\alpha^8 + \alpha^6 + \alpha^5 + \alpha^4 + 1 = 0$, entonces es posible decir que $\alpha^8 = \alpha^6 + \alpha^5 + \alpha^4 + 1$, por lo que podemos sustituir los elementos de mayor grado por su equivalente suma de exponentes menores. En el ejemplo anterior, tenemos que

$$\alpha^9 = (\alpha^8)(\alpha) = (\alpha^6 + \alpha^5 + \alpha^4 + 1)(\alpha) = \alpha^7 + \alpha^6 + \alpha^5 + \alpha \tag{3.12}$$

Otra alternativa es obtener el logaritmo base α de $p(\alpha)$ y $q(\alpha)$, sumar los exponentes módulo 255 y después tomar el antilogaritmo del exponente resultante.

Ejemplo 3.2

$$\log_\alpha(11100010) = 9$$

$$\log_\alpha(01100100) = 247$$

lo cual implica que

$$p(\alpha) \cdot q(\alpha) = \alpha^{9+247} = \alpha^{256} \tag{3.13}$$

$$\alpha^{256} \qquad {}^{\%255=1} = \alpha^1 \tag{3.14}$$

$$\text{Antilog}_\alpha\ (1) = 00000010$$

Por razones prácticas se omitieron las comas en la representación vectorial del ejemplo anterior.

Suma

La operación de suma es la que conlleva menos complejidad, ya que consiste solamente en una suma modular. Se define de la siguiente manera:

Si $p(\alpha) = \mathrm{a}_0 + \mathrm{a}_1\alpha + \cdots + \mathrm{a}_7\alpha^7$ y $q(\alpha) = \mathrm{b}_0 + \mathrm{b}_1\alpha + \cdots + \mathrm{b}_7\alpha^7$ están ambos en $\mathbb{F}_2[\alpha]$ entonces $p(\alpha) + q(\alpha) = \mathrm{c}_0 + \mathrm{c}_1 x + \cdots + \mathrm{c}_7\alpha^7$, donde para cada i, $\mathrm{c}_i = \mathrm{a}_i + \mathrm{b}_i$.

Lo anterior sugiere que para sumar dos polinomios sólo tenemos que efectuar término a término la operación lógica **or exclusiva**, descrita de la siguiente manera:

$\bigoplus$	0	1
0	0	1
1	1	0

Cuadro 3.3: *Or exclusiva*

Por ejemplo, para sumar los polinomios del ejemplo 3.1.

Ejemplo 3.3

	Representación vectorial	Representación polinomial
$p(\alpha)$	(0,1,1,1,0,0,0,1)	$\alpha^6 + \alpha^5 + \alpha^4 + 1$
$q(\alpha)$	$\bigoplus$(0,0,0,0,0,0,1,0)	$\bigoplus$ α
$p(\alpha) \bigoplus q(\alpha)$	$(0, 1, 1, 1, 0, 0, 1, 1)$	$\alpha^6 + \alpha^5 + \alpha^4 + \alpha + 1$

Cuadro 3.4: *Suma vectorial y polinomial en el campo* F_{2^8}

Operaciones inversas

Las operaciones inversas, resta y división, se implementan de manera similar. En el caso de la resta, la implementación es idéntica a la suma, mientras que la división se transforma en una resta de exponentes.

Ejemplo 3.4

	Representación polinomial	Representación exponencial
Dividendo	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha$	$\alpha^9 \cdot \alpha^{-8} = \alpha^1$
Divisor	$\div\alpha^6 + \alpha^5 + \alpha^4 + 1$	
Cociente	α^1	

Cuadro 3.5: *División polinomial y exponencial en el campo* F_{2^8}

Para aprovechar la estructura empleada en el multiplicador, podemos realizar la división como una suma de exponentes, siempre y cuando uno de los exponentes sea previamente convertido a su inverso multiplicativo, es decir, un α^m que sumado al α^n original dé como resultado α^{255}.

$$\frac{\alpha^9}{\alpha^8} = \alpha^9 \cdot (\alpha^8)^{-1} \tag{3.15}$$

$$(\alpha^8)^{-1} = \alpha^{255-8} = \alpha^{247} \tag{3.16}$$

	Representación polinomial	Representación exponencial
Dividendo	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha$	α^9
Divisor	$\div \quad \alpha^6 + \alpha^5 + \alpha^2$	$+\alpha^{247}$
Cociente	α^1	$\alpha^{256\,\%255=1} = \alpha^1$

Cuadro 3.6: *División polinomial y exponencial en el campo* F_{2^8}*. En la representación exponencial* α^{256} *se encuentra fuera del campo, por lo que es necesario aplicar la operación de módulo* 255 *al exponente cuyo resultado es* α^1

3.2. Conceptos Básicos de Teoría de la Información

Puesto que la presente tesis plantea una manipulación de imágenes médicas sin pérdida de información, es necesario definir este último concepto. La Teoría de la Información provee el marco teórico y las herramientas matemáticas fundamentales para el desarrollo del esquema de manipulación sin pérdida.

3.2.1. Cantidad de Información

El concepto de cantidad de información fue acuñado por Claude Shannon [18] para resolver problemas de transmisión de mensajes en sistemas de comunicación. Sea A un evento que pertenece a un conjunto de resultados en un experimento aleatorio. Definimos entonces que la información asociada al evento A está dada por:

$$I(A) = \log_2 \frac{1}{P(A)} \qquad [bits] \tag{3.17}$$

donde $P(A)$ es la probabilidad del evento A. Como el valor del logaritmo de una fracción decrece si el denominador se incrementa, de la ecuación anterior podemos deducir que, si la probabilidad de un evento es baja, su cantidad de información asociada será alta, pero si su probabilidad es alta, su cantidad de información asociada es baja.

Ejemplo 3.5

Supongamos que tenemos una caja con diez canicas, nueve negras y una blanca. Si el experimento consiste en sacar una canica al azar, la probabilidad de que salga una canica negra (evento **N**) será de $9/10$ y de que salga una blanca (evento **B**) de $1/10$. La cantidad de información asociada a cada evento será entonces:

$$I(N) = \log_2(\frac{1}{P(N)}) = \log_2(\frac{1}{0.9}) = 0.15 \tag{3.18}$$

$$I(B) = \log_2(\frac{1}{P(B)}) = \log_2(\frac{1}{0.1}) = 3.32 \tag{3.19}$$

Claramente, la información asociada al evento **B** es mucho mayor que la asociada al evento **N**. Una manera de interpretar este resultado es que si sacamos una canica blanca, la única en nuestro conjunto original, sabremos con certeza que el resto de las canicas es negra, por lo cual habremos anulado la incertidumbre por completo. Es decir, obtenemos mucha información respecto al conjunto original. En cambio, si sacamos una canica negra, la incertidumbre se reduce mucho menos: todavía no es posible saber con seguridad de qué color será la siguiente canica extraída.

Una propiedad de la definición matemática de cantidad de información es que la información obtenida de la ocurrencia de dos eventos independientes es la suma de la información obtenida de la ocurrencia de cada evento por separado.

Si tenemos un sistema con 2^N eventos equiprobables, cada evento aportará N bits de información:

$$I = \log_2 \frac{1}{\frac{1}{2^N}} = \log_2 2^N = N \quad [bits] \tag{3.20}$$

3.2.2. Entropía

Si tenemos un conjunto M de eventos independientes A_i, que son salidas de un experimento aleatorio S, entonces la cantidad promedio de información asociada con dicho experimento estará dada por:

$$H = \mathrm{p}_1 \ \log_2 \frac{1}{\mathrm{p}_1} + \mathrm{p}_2 \ \log_2 \frac{1}{\mathrm{p}_2} + \cdots = \sum_{k=1}^{M} \mathrm{p}_k \ \log_2 \frac{1}{\mathrm{p}_k} \tag{3.21}$$

donde H se denomina entropía, p_i son las probabilidades de que aparezcan los diferentes eventos y M, el número total de eventos. La entropía entonces se refiere a la cantidad promedio de información asociada con el conjunto de eventos aleatorios, y satisface las siguientes afirmaciones:

- Un cambio pequeño en alguna de las probabilidades de aparición de uno de los eventos debe cambiar poco la entropía.
- Si la aparición de todos los eventos es equiprobable, entonces la entropía será máxima.

Shannon demostró que, si el experimento corresponde a una fuente que entrega símbolos A_i de un conjunto A, entonces la entropía es una medida del número promedio de símbolos binarios necesarios para codificar la salida de la fuente. También demostró que la máxima compresión sin pérdida teóricamente posible equivale a codificar la salida de una fuente con un número promedio de bits igual a la entropía de la fuente. Esto repercute directamente en la compresión de imágenes ya que si la entropía es baja, mejor tasa de compresión se logra, puesto que serán necesarios menos bits para codificar cada salida de la fuente.

3.2.3. Entropía aplicada a imágenes

Podemos aplicar el concepto de entropía a una imagen, ya que ésta se puede considerar como una matriz de valores enteros que indican la tonalidad de gris (para imágenes monocromáticas) de cada pixel. Por ejemplo:

$$Imagen = \begin{bmatrix} 253 & 253 & 151 & 151 & 151 & 68 & 32 & 32 \\ 253 & 253 & 151 & 151 & 79 & 68 & 32 & 32 \\ 253 & 253 & 151 & 151 & 79 & 68 & 32 & 30 \\ 253 & 230 & 151 & 130 & 60 & 40 & 30 & 15 \end{bmatrix}$$

Si consideramos al valor de cada pixel como un evento aleatorio y a la imagen completa como el conjunto de eventos aleatorios, podemos aproximar la probabilidad de ocurrencia de cada nivel de gris al calcular el histograma de la imagen:

Nivel de gris	Probabilidades
15	1/32
30	2/32
32	5/32
40	1/32
60	1/32
68	3/32
79	2/32
130	1/32
151	8/32
230	1/32
253	7/32

Cuadro 3.7: Probabilidad de ocurrencia de los pixeles

La entropía de la imagen sería entonces:

$$H = \frac{1}{32}\ \log_2 32 + \frac{2}{32}\ \log_2 \frac{32}{2} + \frac{5}{32}\ \log_2 \frac{32}{5} + \frac{1}{32}\ \log_2 32 + \frac{1}{32}\ \log_2 32 + \frac{3}{32}\ \log_2 \frac{32}{3} + \frac{2}{32}\ \log_2 \frac{32}{2} +$$

$$\frac{1}{32}\ \log_2 32 + \frac{8}{32}\ \log_2 \frac{32}{8} + \frac{1}{32}\ \log_2 32 + \frac{7}{32}\ \log_2 \frac{32}{7} = 2.997\ \ bits\ \ por\ \ pixel \tag{3.22}$$

En promedio, cada pixel puede ser codificado con tres bits sin perder información. Así, la entropía nos indica el límite máximo de compresión de la imagen sin pérdida, razón que podemos expresar de la siguiente manera:

$$Razón\ \ de\ \ Compresión\ \ = \frac{Número\ \ de\ \ bits\ \ de\ \ la\ \ imágen\ \ original}{Número\ \ de\ \ bits\ \ de\ \ la\ \ imágen\ \ comprimida} \tag{3.23}$$

Substituyendo valores obtenemos:

$$Razón\ \ de\ \ Compresión\ \ = \frac{8\ \ bits}{3\ \ bits} = 2.66 \tag{3.24}$$

La mayoría de las imágenes poseen una característica en común: los pixeles cercanos están correlacionados, y por lo tanto contienen información redundante. En las imágenes tenemos dos tipos de redundancia:

- Redundancia espacial (o correlación entre pixeles cercanos).
- Redundancia espectral (o correlación entre las bandas espectrales).

Entre mayor redundancia exista en la imagen, mejor razón de compresión será obtenida, ya que la entropía será menor. Las investigaciones en compresión de imágenes buscan reducir el número de bits necesarios para representar una imagen por medio del aumento de la redundancia espacial y espectral tanto como sea posible [19].

3.3. Transformada Wavelet

Es una técnica de procesamiento de señales e imágenes que permite conocer su contenido espectral y cómo cambia en el tiempo o en el espacio. Está basada en ondas (wavelet) que generalmente son irregulares, asimétricas y de duración limitada cuyo valor medio es cero (fig. 3.1). La transformada se define en una dimensión como la suma sobre todo el tiempo de la señal $f(t)$ multiplicada por versiones trasladadas (desplazadas en tiempo) y escaladas de la wavelet original o madre (fig. 3.2).

$$C(escala, posición) = \int_{-\infty}^{\infty} f(t)\phi(escala, posición, t)dt$$

El resultado de la transformación son coeficientes Wavelet $C(escala, posición)$, que determinan el nivel de correlación existente entre la wavelet y la señal $f(t)$, por tanto estos coeficientes están en función del tipo de wavelet madre utilizada, así como de la escala y posición de la misma.

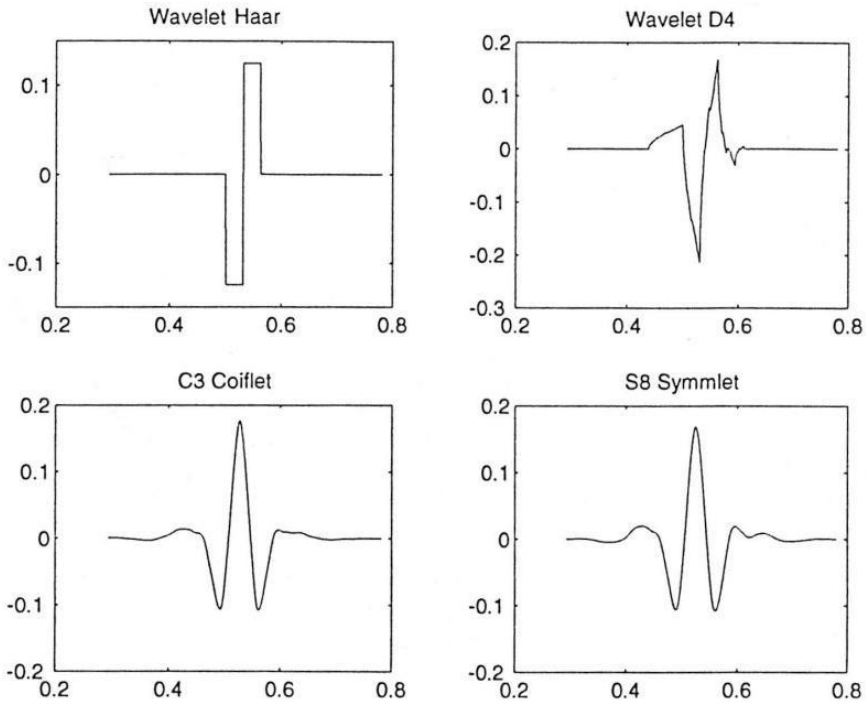


Figura 3.1: *Gráficos de varios tipos distintos de wavelets.*

La idea general del análisis de señales mediante wavelets es la de correlacionar la wavelet en tiempo 0 con un valor de escala determinado, por ejemplo de 1, obtener el valor del coeficiente, posteriormente desplazar la wavelet un tiempo τ y calcular nuevamente la correlación entre las señales. Este desplazamiento se repite hasta terminar con la

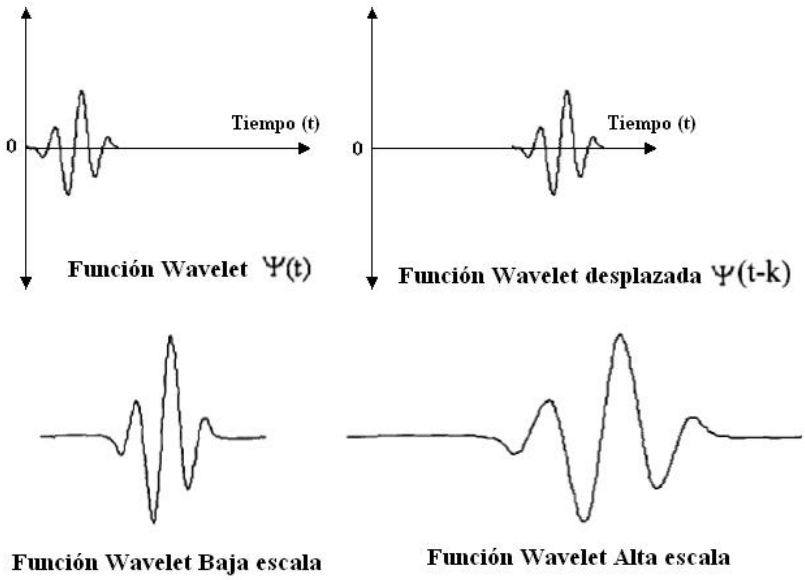


Figura 3.2: *Desplazamiento y escalamiento de las wavelets.*

señal a analizar. Una vez finalizado el desplazamiento se repite el análisis con un valor de escala diferente.

El proceso se puede apreciar gráficamente en la figura 3.3. A partir de la forma de la wavelet se puede deducir la relación que existe entre las diferentes escalas y el contenido en frecuencia de la señal analizada:

- Baja escala $\alpha \longrightarrow$ Wavelet comprimida $\longrightarrow$ cambio rápido de los detalles$\longrightarrow$ alta frecuencia ω.
- Alta escala $\alpha \longrightarrow$ Wavelet alargada $\longrightarrow$ cambio lento de los detalles$\longrightarrow$ baja frecuencia ω.

Es decir, entre más estrecha sea la wavelet, mayor será su correlación con las altas frecuencias, y conforme se vaya ampliando, aumentará la correlación con las bajas frecuencias.

De esta manera se pueden emplear los coeficientes wavelet para generar una representación tiempo-frecuencia del contenido espectral de la información transformada.

Con algunas modificaciones, esta herramienta matemática también puede utilizarse para analizar señales discretas.

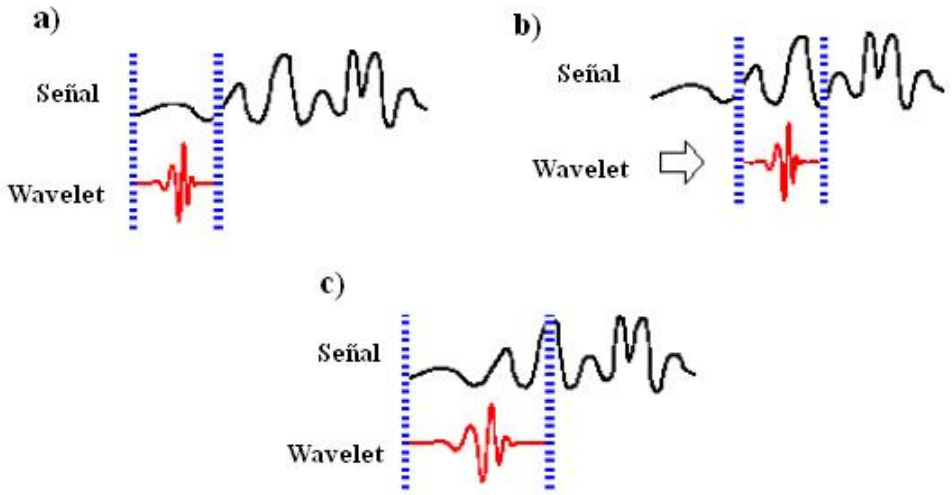


Figura 3.3: *Funcionamiento gráfico de la transformada wavelet. a)wavelet tiempo = 0 y escala = x, b) wavelet tiempo =τ y escala = x, c)wavelet tiempo =0 y escala = 2x.*

Transformada Wavelet Discreta

Es un algoritmo introducido por Mallat en 1988[20], basado en un banco de filtros que permite la separación de la señal de interés en componentes de bajas (Aproximaciones) y altas frecuencias (Detalles).

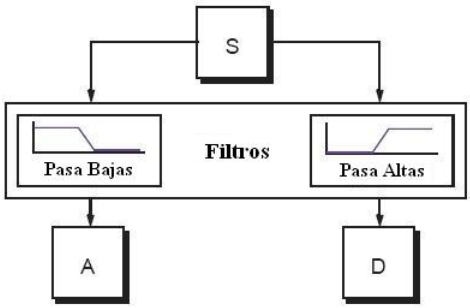


Figura 3.4: *La descomposición de la señal S, en componentes de Bajas y Altas frecuencias, donde A es la salida del filtro de bajas frecuencias(aproximación) y D la salida del filtro de altas frecuencias (detalle).*

donde S es la señal de interés, A es el resultado de aplicar el filtro de aproximación y D la salida del filtro de detalles.

Los filtros deben ser diseñados de tal forma que sean complemetarios, es decir, la suma de A y D debe ser S. Al procesar la señal, cada filtro elimina componentes espectrales de la señal S haciendo que la mitad de las muestras sean redundantes, es decir, la mitad de las muestras pueden ser descartadas por una operación de submuestreo sin que se pierda información [21].

En la figura 3.5 se muestra el efecto de submuestreo aplicado a una señal que tiene 1000 muestras.

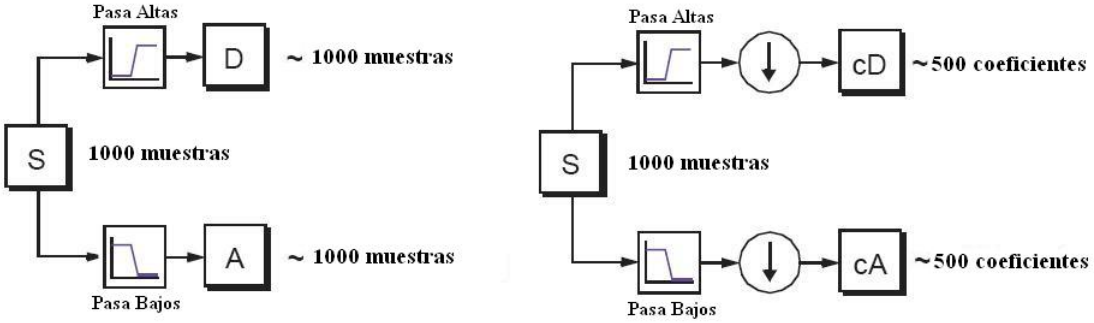


Figura 3.5: *Al submuestrear las salidas de los filtros de aproximación y detalle se logra que el conjunto de A y D sea del mismo tamaño que la imagen original.*

Descomposición Multinivel

Existen aplicaciones en las que se necesitan más de dos bandas de frecuencias de descomposición para poder separar las características de la señal, es decir, se necesita una descomposición multinivel [22]. Ésta consiste en la iteración del proceso de filtrado: repetir el mismo procedimiento sobre la señal de salida de la etapa anterior. En la figura 3.6 se muestra un ejemplo de descomposición Wavelet multinivel.

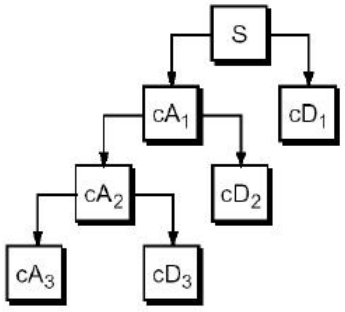


Figura 3.6: *Descomposición wavelet multinivel. La salida de la primera etapa de filtrado se convierte en la entrada de la siguiente etapa.*

cD1 es el componente de más alta frecuencia de la señal, y cA3 el de menor frecuencia.

Reconstrucción Wavelet

Para regresar a la señal original S a partir de los coeficientes cA_i y cD_i se sigue el mismo razonamiento pero en dirección contraria.

En este caso se debe realizar un remuestreo de los datos para compensar el submuestreo realizado en el proceso de descomposición, luego pasar por un proceso de filtrado de reconstrucción y finalmente reconstruir S (fig. 3.7), donde H' y L' son los filtros de reconstrucción.

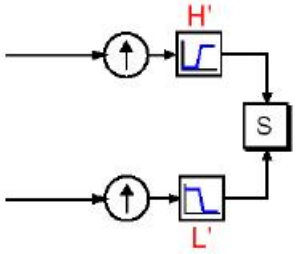


Figura 3.7: *Proceso de reconstrucción de la señal original a partir de los componentes obtenidos en la descomposición. H' y L' son los filtros de reconstrucción.*

3.3.1. Wavelet 2D

La extensión Wavelets a dos dimensiones permite hacer un análisis multirresolución de la variabilidad de la imagen en las distintas direcciones: horizontal, vertical y diagonal.

De la misma forma que para señales en 1-D, las descomposiciones pueden basarse en diferentes familias de funciones. Finalmente el resultado dependerá de las imágenes y de la wavelet madre elegida. La implementación se hace a través de dos filtros: un pasa-bajos y un pasa-altos, con los que se pueden generar diferentes esquemas de filtrado.

Se considera que una imagen es una función bidimensional que puede ser representada como una matriz de información **N** $\times$ M, en donde **N** representa las filas y **M** las columnas de la imagen (una definición formal de imagen digital se puede consultar en el apéndice A.).

En la figura 3.8 se muestra el esquema utilizado por el manipulador para realizar el filtrado de la imagen. Esto se realiza en dos fases, primero se filtran y submuestrean las columnas, generando dos matrices de $N \times \frac{M}{2}$ a las cuales se les filtran y submuestrean las filas en una segunda fase, dando como resultado 4 imágenes (o matrices de información) de $\frac{N}{2} \times \frac{M}{2}$, una sería la aproximación y las demás tendrán los detalles en las direcciones mencionadas.

En la figura 3.9 se puede observar un ejemplo de como una imagen de $\frac{N}{2} \times \frac{M}{2}$ es filtrada en sus distintas direcciones.

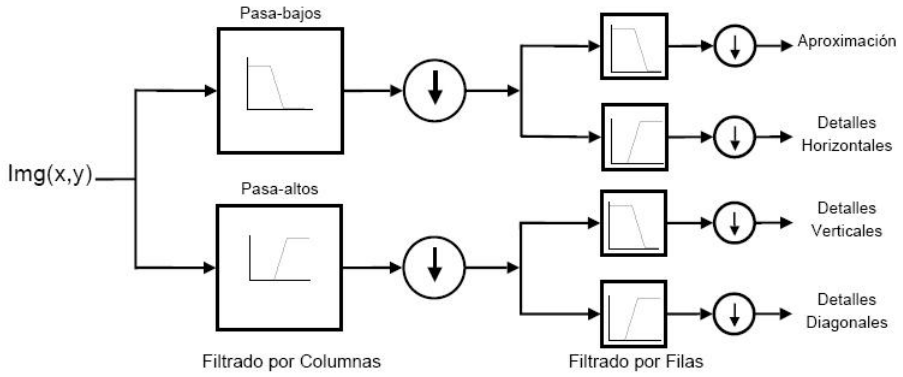


Figura 3.8: *Árbol de descomposición wavelet en 2-D. Cada rama del árbol es una secuencia de filtrado distinta que permite hacer un análisis multirresolución de la imagen.*

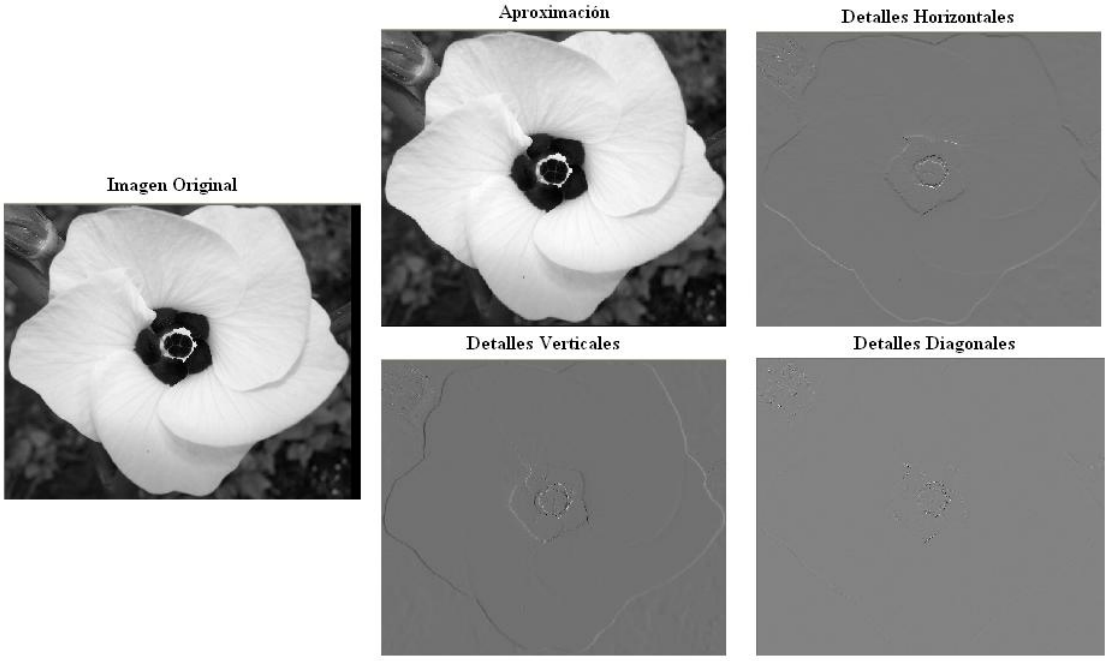


Figura 3.9: *Ejemplo de la transformada wavelet en 2-D. La imagen mostrada a la izquierda es descompuesta en sus cuatro direcciones: aproximación, detalles horizontales, detalles verticales y detalles diagonales. Imagen tomada de www.originlab.com.*

3.3.2. Banco de Filtros sobre el campo finito F_{2^8}

Los bancos de filtros que utilizan aritmética sobre anillos de característica cero (números reales $\mathbb{R}$, enteros $\mathbb{Z}$ y racionales $\mathbb{Q}$) presentan el problema llamado **intermediate coefficient swelling**, que consiste en el crecimiento descontrolado del número de bits necesarios para representar la precisión de los resultados intermedios del proceso de filtrado [24]. Una forma de evitar este problema es la utilización de aritmética sobre anillos finitos. Uno de los anillos más utilizados para este tipo de aplicación son los anillos diádicos, con los cuales se puede controlar el número de bits (amplitud de palabra) para representar los coeficientes intermedios.

A. Klappenecker [25] demostró, para el anillo diádico módulo $Z/256Z$, que si se emplean bancos de filtros cuyas matrices de polifase tanto de análisis $\mathrm{H}_p(z)$ como de síntesis $\mathrm{G}_p(z)$ son elementos del grupo general $GL(2, A[z, z^{-1}])$ y cumplen la siguiente condición:

$$\mathrm{H}_p(z)\mathrm{G}_p(z)^t = I$$

se puede realizar una reconstrucción perfecta. Las matrices de polifase están formadas por los coeficientes pares e impares de los filtros empleados. Así, si $\tilde{\alpha}(z) = \tilde{\alpha}_e(z^2) + z^{-1}\tilde{\alpha}_o(z^2)$, y $\tilde{\beta}(z) = \tilde{\beta}_e(z^2) + z^{-1}\tilde{\beta}_o(z^2)$ entonces:

$$\mathrm{H}_{\mathbf{p}}(\mathbf{z}) = \begin{pmatrix} \tilde{\alpha}_e(z) & \tilde{\beta}_e(z) \\ \tilde{\alpha}_o(z) & \tilde{\beta}_o(z) \end{pmatrix}$$

G_p se define de la misma manera.

La figura 3.10 comprende a un sencillo banco de filtros para emplearse en aplicaciones de compresión.

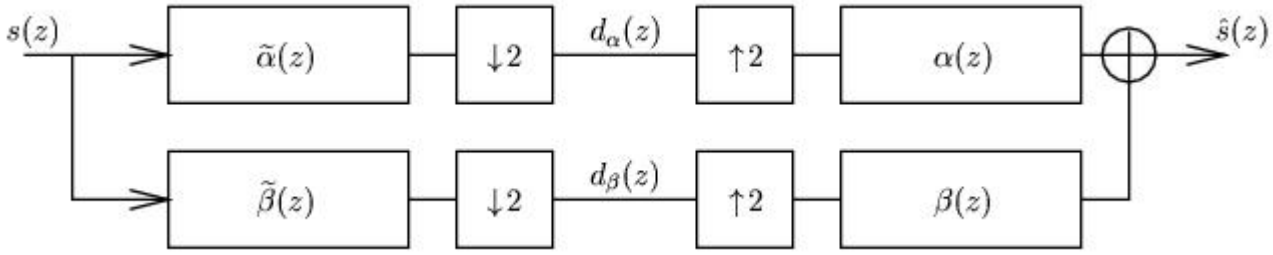


Figura 3.10: *Banco de filtros de dos canales para señales con valores en el anillo diádico módulo* $Z/256Z$.

Para obtener los filtros de análisis es necesario multiplicar la matriz de polifase $\mathrm{H}_p(z)$, por el vector de desplazamiento.

$$(\tilde{\alpha(z)}, \tilde{\beta(z)}) = (1, z^{-1})\mathrm{H}_p(z^2) = (1, z^{-1}) \begin{pmatrix} 0 & 1 \\ 1 & \text{-1} \end{pmatrix} = (z^{-1}, 1 - z^{-1})$$

Los filtros de síntesis son calculados de la misma manera, por lo que es necesario conocer $\mathrm{G}_p(z)$, para lo cual se debe resolver la siguiente ecuación:

$$\mathrm{H}_p(z)^{-1}\mathrm{H}_p(z)\mathrm{G}_p(z)^t = \mathrm{H}_p(z)^{-1}I$$

de lo que resulta

$$\mathrm{G}_p(z)^t = \mathrm{H}_p(z)^{-1}$$

La matriz inversa de $\mathrm{H}_p(z)$ es

$$\mathrm{H}_p(z) = \begin{pmatrix} 0 & 1 \\ 1 & \text{-1} \end{pmatrix} \qquad \mathrm{H}_p(z)^{-1} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

por tanto

$$\mathrm{G}_p(z)^t = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \qquad \mathrm{G}_p(z) = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

Finalmente, se calculan los filtros de síntesis

$$(\alpha(z), \beta(z)) = (1, z^{-1})\mathrm{G}_p(z^2) = (1, z^{-1}) \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = (1 + z^{-1}, 1)$$

Dadas su sencillez y capacidad de reconstrucción sin pérdida, éstos fueron los filtros seleccionados para este proyecto.

Para implementar uno de estos bancos de filtros empleando aritmética de campo finito fue necesario realizar un homomorfismo entre el campo finito F_{2^8} y el grupo aditivo formado por el anillo diádico módulo $Z/256Z$. Esto es necesario ya que el anillo diádico contiene elementos positivos y negativos, a diferencia del campo finito, el cual sólo contiene elementos positivos.

Este homomorfismo es realizado al substituir las operaciones de suma y resta del anillo por la operación or exclusiva en campo finito, que en la aritmética de campo equivale a realizar tanto la suma como la resta.

3.4. Codificación Aritmética

Es una técnica de compresión basada en un modelo estadístico ideada por Elias en 1963[26] que genera una etiqueta o marca para representar la secuencia que se desea comprimir o codificar. Produce por lo general excelente razón de compresión y es uno de los codificadores mas utilizados en la compresión sin perdida.

Existen dos versiones de este codificador:

- Codificador Aritmético para números reales
- Codificador Aritmético para números enteros.

3.4.1. Algoritmo de codificación para números reales

En esta versión, la etiqueta es un número que aumenta su precisión a medida que recibe los elementos de la secuencia a codificar.

Para comenzar con el proceso de codificación es necesario definir los símbolos del alfabeto $|A| = \{\mathrm{a}_1, ..., \mathrm{a}_n\}$, sobre el que se basa la secuencia $S = (\mathrm{s}_1, \mathrm{s}_2...)$ a ser codificada, así como la probabilidad de a_i en S :

$$P(\mathrm{a}_i) := \frac{|\mathrm{S}_{\mathrm{a}_i}|}{n}$$
$$n = |S|$$

De la ecuación anterior se puede concluir que $P(\mathrm{a}_i)$ esta siempre contenida en el intervalo $[0, 1)$ para cualquier símbolo, y que la suma de todas las probabilidades es $\sum_{i=1}^{n} P(\mathrm{a}_i) = 1$.

Esto permite proyectar las probabilidades del alfabeto A como subintervalos consecutivos dentro del rango $[0, 1)$, generando una recta llamada **recta de convergencia**. La distribución de las probabilidades se realizan con la función de distribución de probabilidad.

$$F(n) = \sum_{i=1}^{n} \mathrm{p}_{\mathrm{a}_i} \quad y \quad F(0) = 0,$$

Lo que resulta en tantos subintervalos como símbolos en el alfabeto.

Ejemplo 3.6: Creación de intervalos

$$A = a, b, c$$
$$P(a) = 0{,}2 \quad P(b) = 0{,}4 \quad P(c) = 0{,}4$$

Función de distribución	Las probabilidades acumuladas
F(0)	0.0
F(1)	0.2
F(2)	0.6
F(3)	1

En la figura 3.11 se observa cómo los símbolos del alfabeto son proyectados sobre el intervalo $[0, 1)$.

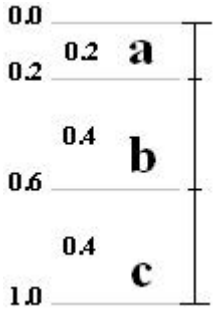


Figura 3.11: *En la figura se observa un ejemplo de cómo se proyectan los símbolos dentro del intervalo [0,1).*

Codificación

Este algoritmo de codificación está fuertemente basado en la densidad de orden existente entre los números racionales, es decir, entre dos números racionales (**Límite Inferior** y **Límite Superior**) siempre existe otro número racional (**etiqueta**) de mayor precisión. En el ejemplo anterior el **Límite Inferior** es cero y el **Límite Superior** es uno, lo que quiere decir, que la **etiqueta** estará contenida dentro de este intervalo.

El primer símbolo de la secuencia a codificar será el que determine la región de convergencia de la **etiqueta**, si en la figura 3.11 el primer símbolo a codificar fuera **b**, la **etiqueta** estará contenida dentro del rango $[0{,}2, 0{,}6)$ y los nuevos límites inferior y superior serían 0.2 y 0.6. Para asegurar que el siguiente símbolo a codificar converja dentro de este nuevo intervalo es necesario replantear las probabilidades del alfabeto de tal forma que ahora se encuentren contenidas entre los nuevos límites inferior y supeior. Este proceso se repite por cada símbolo a ser codificado.

En la figura 3.12 se muestra un ejemplo de cómo opera un codificador aritmético para números reales. Es de notar que conforme se realiza cada codificación el rango que contiene a la **etiqueta** se vuelve mas preciso, una vez terminado el proceso de codificación se puede tomar como **etiqueta** cualquier valor que se encuentre dentro del intervalo final, se acostumbra tomar el valor intermedio, posteriormente se convierte a binario y es almacenado o transmitido.

$$0{,}2128 \approx 0{,}0001101100111101$$

$$Etiqueta = 0001101100111101$$

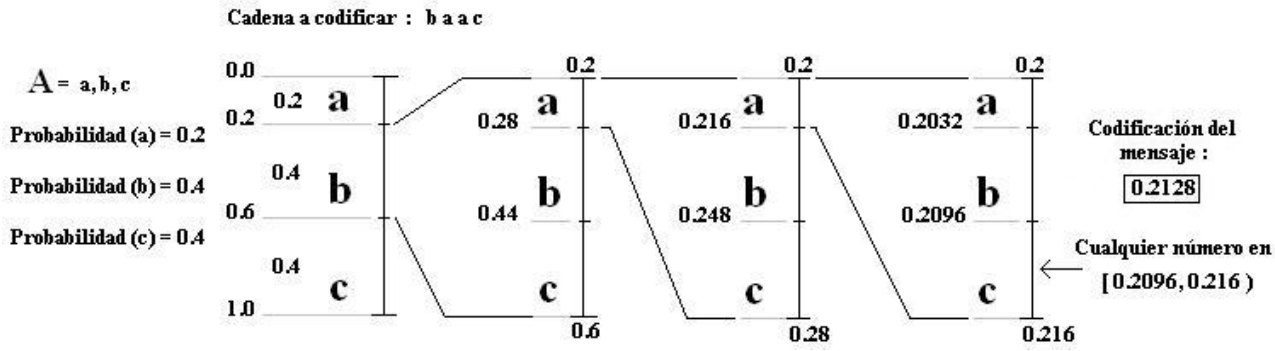


Figura 3.12: *Ejemplo de un Codificador Aritmético para números reales.*

Decodificación

Para la recuperación de los símbolos a partir de la **etiqueta** generada por la codificación, es esencialmente el mismo proceso, necesitamos conocer la distribución de probabilidades que dió origen a la **etiqueta** para poder saber en qué intervalo se encuentra contenida en cada acotamiento.

Observamos en la figura 3.13, el mismo esquema que en la codificación, salvo que ahora por cada acotamiento se decodifica la letra que corresponde al intervalo en el cual se encuentra contenida la **etiqueta**.

El hecho de que sólo se pueda obtener la **etiqueta** cuando todos los símbolos estén codificados supone un gran inconveniente, sobre todo en aplicaciones que requieren velocidad, como la compresión de vídeo para videoconferencia. Esta característica, aunada a la alta precisión requerida para representar la **etiqueta**, hace que este algoritmo sea impráctico.

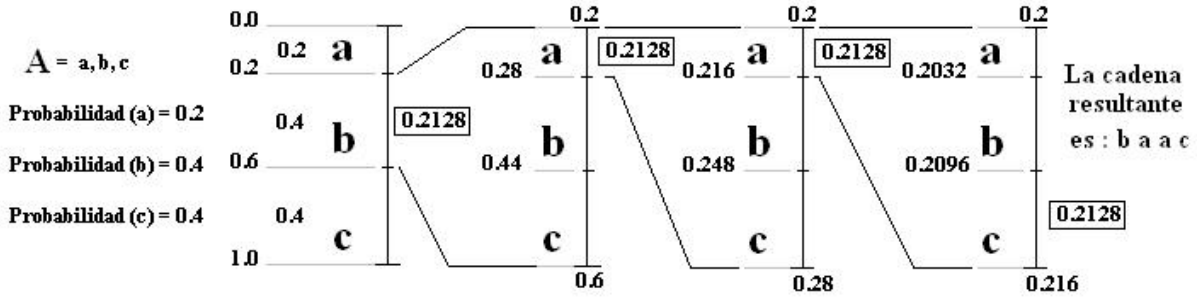


Figura 3.13: *Ejemplo de un Decodificador Aritmético para números reales.*

3.4.2. Algoritmo de codificación con aritmética entera

Este algoritmo opera únicamente con números enteros, evitando así las pérdidas por redondeo (y la necesidad de gran cantidad de bits para mantener la precisión), lo que lo convierte en un excelente candidato para la compresión sin pérdida. Al ser una adaptación del algoritmo descrito en la sección anterior opera con la misma lógica. Sin embargo al sólo utilizar enteros las probabilidades de aparición $P(\mathrm{a}_i)$ no pueden ser representadas como fracciones de 1. Por lo tanto, necesitamos definir un intervalo inicial sobre el que se proyectarán las frecuencias de aparición $Count(\mathrm{a}_i)$ de cada uno de los símbolos, cuya longitud se acotará conforme avance el proceso de codificación.

El intervalo propuesto (inicial), debe ser mínimo de longitud $4 * Total\ \ Count$, donde **Total Count** es el tamaño total de la secuencia a codificar, esto asegura que en el proceso de codificación los límites superior e inferior del intervalo no se crucen.

Ejemplo 3.7: Creación del Intervalo Inicial

$$A = \{a, b\}$$

$$Secuencia\ \ a\ \ codificar\ \ :\ \ b \rightarrow b \rightarrow a$$

$$Total\ \ Count = 3$$

$$por\ \ tanto\ \ el\ \ intervalo\ \ propuesto\ \ debe\ \ ser\ \ >\ \ 12$$

Por razones prácticas el intervalo se mapea en binario de la siguiente manera:

$$\acute{L}imite\ \ Inferior \longleftrightarrow \underbrace{0 \cdots\cdots 0}_{m}$$

$$1\underbrace{0 \cdots\cdots 0}_{m-1}$$

$$\acute{L}imite\ \ Superior \longleftrightarrow \underbrace{1 \cdots\cdots 1}_{m}$$

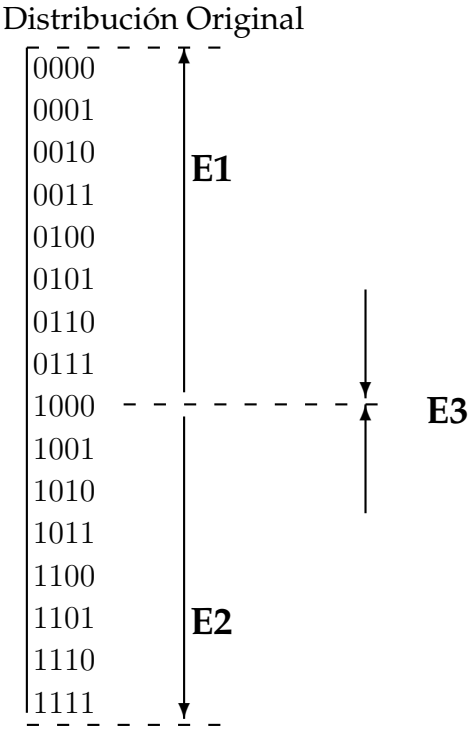
donde:

$$\lceil m \rceil > \mathrm{Log}_2(4 * Total\ \ Count)$$

Utilizando los datos del ejemplo anterior se puede calcular m:

$$\lceil m \rceil > \mathrm{Log}_2(4 * 3) = 3{,}58$$

por tanto $m = 4$.



Cuadro 3.8: *Intervalo propuesto para la codificacióna aritmética con números enteros, así como los segmentos correspondientes a cada condición.*

El intervalo resultante se muestra en el cuadro 3.8.

En este mismo cuadro se observan tres intervalos de convergencia **E1**, **E2** y **E3**, intuitivamente se entiende que el nuevo intervalo (delimitado por los límites inferior y superior), resultado de la codificación de cualquier elemento del alfabeto convergerá a alguna de estas tres regiones, es decir, habrá bits de la **etiqueta** que serán fijos, los cuales pueden ser transmitidos o almacenados antes de terminar el proceso de codificación.

Ejemplo 3.8: Convergencia de la etiqueta

Si,

$$Límite\ \ Inferior = 0000$$

y

$$Límite\ \ Superior = 0100$$

la **etiqueta** convergerá a la región **E1**

Si,

$$Límite\ \ Inferior = 0111$$

y

$$Límite\ \ Superior = 1000$$

la **etiqueta** convergerá a la región **E3**

Si la **etiqueta** converge a la región **E1** o **E2** implica que los bits más significativos (MSB) de ambos límites son iguales y no cambiarán con el siguiente acotamiento, lo que lo hace irrelevante para el siguiente paso de codificación. Sin embargo forma parte importante en el proceso de decodificación, por lo que es almacenado en la **etiqueta**. Terminado este proceso se desplazan ambos límites hacia la izquierda generando un espacio en el bit menos significativo (LSB) que, en el caso del **Límite Inferior** (L_I) se rellena con un 0 y en el caso del **Límite Superior** (L_U) con un 1.

La región **E3** indica que los límites convergen al centro del intervalo, es decir, el límite inferior y superior nunca convergen a algun valor, lo que ocasiona que sea imposible la codificación. Para solucionar este problema, se piensa que por ahora no se puede determinar si la **etiqueta** converge a **E1** o **E2**, pero esto se puede determinar en el siguiente paso de codificación. En el apéndice A encotrará el algoritmo completo de codificación.

Una vez determinado el intervalo inicial, lo siguiente es conocer las frecuencias de aparición de cada uno de los símbolos del alfabeto para acumularlas y mapearlas de forma proporcional sobre el intervalo propuesto.

$$A = \{\mathrm{a}_1, \mathrm{a}_2, \mathrm{a}_3, ..., \mathrm{a}_k\}$$

$$Total\ \ Count = Count(\mathrm{a}_1) + Count(\mathrm{a}_2) + \cdots + Count(\mathrm{a}_k), \quad (k \geq 1)\ \ y\ \ Count(0) = 0$$

Donde $Count(\mathrm{a}_k)$ es la frecuencia de aparición del símbolo a_k y $Cum\_Count(\mathrm{a}_k)$ es la frecuencia acumulada del símbolo a_k.

$$Cum\_Count(\mathrm{a}_k) = \sum_{i=1}^{k} Count(\mathrm{a}_i)$$

Ejemplo 3.9

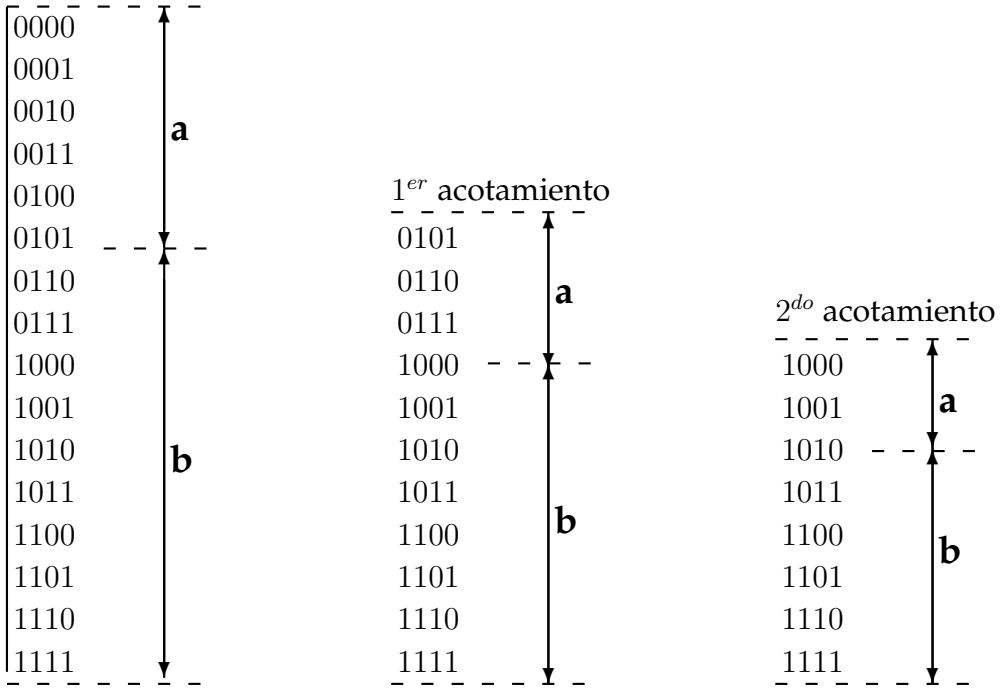
$A =$ [a,b,b]

$\sum = a, b$

$Count(a) = 1$
$Count(b) = 2$

$CumCount(0) = 0$
$CumCount(a) = 1$
$CumCount(b) = 3$

$Secuencia\ \ a\ \ codificar\ \ :\ \ b \rightarrow b \rightarrow a$



Etiqueta = 1000

En el ejemplo anterior podemos observar cómo se delimita el intervalo a medida que se avanza en la codificación. La etiqueta que representa a la secuencia se encuentra dentro del intervalo 1000 y 1001, es decir, cualquier número contenido entre estos valores podría utilizarse para codificar esta serie de datos.

Es importante resaltar que en el primer acotamiento se presenta la condición **E2**, lo que significa que el bit más significativo es 1.

La secuencia de acotamientos se puede determinar matemáticamente mediante las siguientes ecuaciones que calculan los límites superior e inferior del intervalo correspondiente al símbolo a codificar.

$$\mathrm{L}_{I(n+1)} = \mathrm{L}_{In} + \lfloor \frac{(\mathrm{L}_{Un} - \mathrm{L}_{In} + 1) Cum\_Count(\mathrm{a}_{i-1})}{Total\ \ Count} \rfloor \tag{3.25}$$

$$\mathrm{L}_{U(n+1)} = \mathrm{L}_{In} + \lfloor \frac{(\mathrm{L}_{Un} - \mathrm{L}_{In} + 1) Cum\_Count(\mathrm{a}_{i})}{Total\ \ Count} \rfloor - 1 \tag{3.26}$$

Donde:

a_i : es el símbolo que se acaba de recibir.

$\lfloor \rfloor$: denota la parte entera de la operación.

L_{In} : límite inferior inicial.

L_{Un} : límite superior inicial.

$\mathrm{L}_{I(n+1)}$: límite inferior final.

$\mathrm{L}_{U(n+1)}$: límite superior final.

Ejemplo 3.10

Se utilizarán las ecuaciones 3.25 y 3.26 con los datos utilizados en ejemplo 3.9.
Como condiciones iniciales:

$m = 4$.

El intervalo inicial es de $\mathrm{L}_{I0} = 0000_2 = 0$ a $\mathrm{L}_{U0} = 1111_2 = 15$.

Las frecuencias de aparición son:

símbolo	Count	Cum_Count	Intervalo
a	1	1	$0 \leq 1$
b	2	3	$1 \leq 3$

Cuadro 3.9: Tabla de distribución de frecuencias acumuladas.

El mensaje a codificar es $\mathbf{b} \rightarrow \mathbf{b} \rightarrow \mathbf{a}$. El proceso comienza con el primer símbolo de la secuencia, **b**.

$$\mathrm{L}_{I1} = 0 + \lfloor \frac{(15 - 0 + 1) \cdot 1}{3} \rfloor = 5 = 0101_2$$

$$\mathrm{L}_{U1} = 0 + \lfloor \frac{(15 - 0 + 1) \cdot 3}{3} \rfloor - 1 = 15 = 1111_2$$

El siguiente símbolo a codificar es de nuevo **b**.

$$\mathrm{L}_{I2} = 5 + \lfloor \frac{(15 - 5 + 1) \cdot 1}{3} \rfloor = 8 = 1000_2$$

$$\mathrm{L}_{U2} = 5 + \lfloor \frac{(15 - 5 + 1) \cdot 3}{3} \rfloor - 1 = 1111_2$$

El bit más significativo de ambos límites es 1, es decir, se cumple la condición **E2**. Guardamos el bit más significativo en la etiqueta y desplazamos ambos límites hacia la izquierda, generando un espacio que, en el caso de L_I se llena con un 0 y en el caso de L_U con un 1, obteniéndose entonces:

$$1000_2 \leftarrow 000_2 \leftarrow 0_2 = 0000_2$$

$$1111_2 \leftarrow 111_2 \leftarrow 1_2 = 1111_2$$

Finalmente, codificamos el último símbolo, **a**:

$$\mathrm{L}_{I3} = 0 + \lfloor \frac{(15 - 0 + 1) \cdot 0}{3} \rfloor = 0 = 0000_2$$

$$\mathrm{L}_{U3} = 0 + \lfloor \frac{(15 - 0 + 1) \cdot 1}{3} \rfloor - 1 = 15 = 0100_2$$

Este resultado cumple con la condición **E1**, por lo que se recalculan los límites:

$$0000_2 \leftarrow 000_2 \leftarrow 0_2 = 0000_2$$

$$0100_2 \leftarrow 100_2 \leftarrow 1_2 = 1001_2$$

Por tanto, tenemos dos bits guardados en la etiqueta, 10. Para finalizar, se almacena el límite inferior, completando la etiqueta que queda así: 100000.

Algoritmo decodificador con aritmética entera

La decodificación comienza tomando los m bits más significativos de la etiqueta e inicializando el intervalo con los mismos valores con los que fue iniciada la codificación

$$t = 1000_2 = 8$$

$$\mathrm{L}_I = 0000_2 = 0$$

$$\mathrm{L}_U = 1111_2 = 15$$

Donde t contiene los 4 bits más significativos de la etiqueta.

El proceso comienza determinando en qué intervalo de la tabla de frecuencias acumuladas $3{,}9$ se encuentra t.

$$\lfloor \frac{(t - \mathrm{L}_I + 1) \cdot Total\ \ Count - 1}{\mathrm{L}_U - \mathrm{L}_I + 1} \rfloor = \lfloor \frac{(8 - 0 + 1) \cdot 3 - 1}{15 - 0 + 1} \rfloor = 1$$

1 pertenece al intervalo $1 \leq 1 < 3$, que corresponde al símbolo **b**.

Se recalculan los límites según la probabilidad del símbolo decodificado, es decir **b**, mediante las siguientes ecuaciones:

$$\mathrm{L}_{I1} = 0 + \lfloor \frac{(15 - 0 + 1) \cdot 1}{3} \rfloor = 5 = 0101_2$$

$$\mathrm{L}_{U1} = 0 + \lfloor \frac{(15 - 0 + 1) \cdot 3}{3} \rfloor - 1 = 15 = 1111_2$$

Se actualizan los valores

$$t = 1000_2 = 8$$

$$\mathrm{L}_I = 0101_2 = 0$$

$$\mathrm{L}_U = 1111_2 = 15$$

Se decodifica el siguiente

$$\lfloor \frac{(t - \mathrm{L}_I + 1) \cdot Total\ \ Count - 1}{\mathrm{L}_U - \mathrm{L}_I + 1} \rfloor = \lfloor \frac{(8 - 5 + 1) \cdot 3 - 1}{15 - 5 + 1} \rfloor = 1$$

El cual se encuentra nuevamente dentro de $1 \leq 1 < 3$ que corresponde al símbolo **b**.
El proceso se repite, hasta obtener toda la secuencia original.

$$\mathrm{L}_{I2} = 5 + \lfloor \frac{(15 - 5 + 1) \cdot 1}{3} \rfloor = 8 = 1000_2$$

$$\mathrm{L}_{U2} = 5 + \lfloor \frac{(15 - 5 + 1) \cdot 3}{3} \rfloor - 1 = 1111_2$$

Con los límites dentro de la condición **E2** no sólo se realiza el desplazamiento antes descrito, sino que ahora se tiene que actualizar t, desplazándolo a la izquierda y agregándole de derecha a izquierda el siguiente bit de la etiqueta:

$$\mathrm{L}_I = 1000_2 \leftarrow 000_2 \leftarrow 0_2 = 0000_2$$

$$\mathrm{L}_U = 1111_2 \leftarrow 111_2 \leftarrow 1_2 = 1111_2$$

$$t = 1000_2 \leftarrow 000_2 \leftarrow 1_2 = 0000_2$$

$$\lfloor \frac{(t - \mathrm{L}_I + 1) \cdot Total\ \ Count - 1}{\mathrm{L}_U - \mathrm{L}_I + 1} \rfloor = \lfloor \frac{(0 - 0 + 1) \cdot 3 - 1}{15 - 0 + 1} \rfloor = 0$$

0 está dentro del intervalo $0 \leq 0 < 1$, que corresponde al símbolo **a**.
Como de antemano se sabe que es el último símbolo de la cadena aquí se termina el proceso de decodificación. En la práctica se suele agregar un símbolo que indica el fin de codificación.

Los detalles del algoritmo de decodificación de secuencias se puede consultar en el apéndice A.

3.5. Algoritmo Dispersor de Información

En el almacenamiento distribuido de la información se tiene un archivo que debe guardarse en un disco localizado en un componente conectado a una red de comunicaciones. Si el equipo llegara a experimentar una falla de paro, entonces sería imposible la recuperación inmediata de su información.

Con el fin de incorporar la capacidad para tolerar fallas se utiliza un enfoque de almacenamiento basado en el algoritmo de dispersión de información (IDA, Information Dispersal Algorithm) [6]

Consideremos un archivo **F** formado por una sucesión de vectores $\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_N$, sobre un campo finito arbitrario. Cada vector consta de m componentes i.e. tiene m dimensiones.

La idea central del algoritmo es aplicar una transformación lineal sobre la información contenida en el archivo fuente, de tal manera que se le agregue redundancia. Esto se logra al multiplicar cada vector $\vec{b}_i$, por una matriz **A** de $n \times m$. Así, se obtiene un vector $\vec{c}_i$ de n dimensiones.

$$\begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1m} \\ a_{21} & a_{22} & \ldots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nm} \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} \tag{3.27}$$

Evidentemente, **A** es una matriz de n renglones por m columnas. Se requiere que cualesquiera m de sus renglones formen un conjunto de vectores linealmente independientes.

Cada componente del vector $\vec{c}_i$ se transmite por el sistema distribuido y es almacenado en n terminales. Supóngase ahora que, del vector $\vec{c}_i$ se pierden k de sus coordenadas, o bien, que se seleccionan m de sus componentes, en posiciones arbitrarias pero conocidas, para formar el vector $\vec{d}_i$. Esto es equivalente a suponer que $\vec{d}_i$ se obtiene de $\vec{b}_i$ a través de la transformación lineal

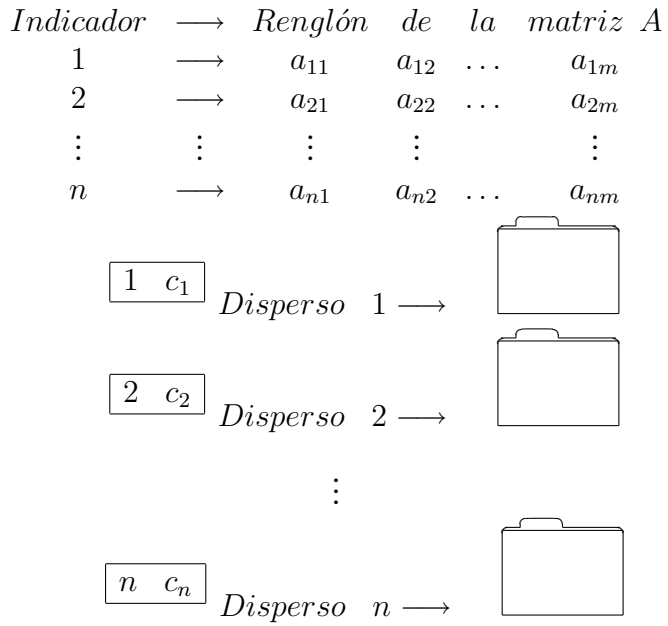
$$\mathbf{B}\vec{b}_i = \vec{d}_i, \tag{3.28}$$

donde, a su vez, **B** se construye seleccionando los m renglones de **A** en las mismas posiciones de las componentes de $\vec{c}_i$ que participan en $\vec{d}_i$. Por el requisito de construcción, los m renglones de **B** son linealmente independientes y, por tanto, es invertible. En consecuencia,

$\vec{b}_i$ puede reconstruirse a partir de $\vec{d}_i$ y **B**:

$$\vec{b}_i = \mathbf{B}^{-1}\vec{d}_i. \tag{3.29}$$

Como se mencionó, la $j - ésima$ coordenada del vector $\vec{c}_i$ se envía a una misma terminal donde forma parte de un archivo denominado "disperso". Por otro lado, el receptor encargado de recuperar **F** necesita conocer **B** para generar su inversa. Para solventar esta dificultad se acordó que cada disperso fuera almacenado junto con un indicador que represente el renglón de **A** que lo generó.



Para generar **B** el receptor deberá disponer de al menos m de los n dispersos para construir una matriz invertible con la que pueda recuperar al vector $\vec{\mathrm{b}_i}$.

El parámetro m (número de columnas de la matriz **A**) determina el mínimo de dispersos necesarios para la reconstrucción del vector $\vec{\mathrm{b}_i}$. El parámetro n (número de renglones de la matriz **A**) determina el número total de dispersos generados. Si $n > m$, el algoritmo es redundante, ya que se generan más dispersos de los que se necesitan para la reconstrucción. El parámetro k, definido como $k = n - m$, determinará cuántos fallos o pérdidas se pueden tolerar en la transmisión. La elección de estos parámetros dependerá de si queremos tener sólo una redistribución de datos ($n = m$; $k = 0$) o un sistema resistente a fallos ($n > m$; $k = n - m$).

3.5.1. Dispersión

En el artículo publicado por M. O. Rabin [6] se señala a modo de ejemplo que, para lograr un respaldo con un buen margen de seguridad, sería necesario generar cinco dispersos de cada archivo. Esta sugerencia, aunada a la mayor facilidad de generar y operar sobre matrices de dimensiones pequeñas, fueron las razones por las que en este trabajo se eligieron $n = 5$, $m = 3$ y por tanto $k = 2$.

Existen diversos métodos para generar matrices linealmente independientes. Lyuu [20] propuso utilizar el método de Vandermonde para construir matrices linealmente independientes de grandes dimensiones. Inicialmente, Rabin sugirió un método recursivo que consiste en proponer unos valores para la primera fila de la matriz, repetirlos en la siguiente fila e irlos probando y modificando hasta que cumplan la condición de independencia lineal entre renglones. Estos pasos se repiten hasta crear una matriz de las dimensiones deseadas. Este método resulta impráctico para generar matrices de dimensiones grandes

pero, debido a que el esquema de dispersión propuesto en la presente tesis requiere una matriz pequeña (5×3), fue empleado para generar $\mathbf{A}$. Por ejemplo:

$$\mathbf{A} = \begin{pmatrix} \alpha^5+\alpha^4+1 & \alpha^5+\alpha^4+\alpha+1 & \alpha^5+\alpha^4+\alpha \\ \alpha^5+\alpha^4+1 & \alpha^5+\alpha^4+1 & \alpha^5+\alpha^4+1 \\ \alpha^5+\alpha^4+\alpha & \alpha^5+\alpha^4+\alpha+1 & \alpha^5+\alpha^4+1 \\ \alpha^5+\alpha^4+\alpha & \alpha^5+\alpha^4+\alpha & \alpha^5+\alpha^4+\alpha+1 \\ \alpha^5+\alpha^4+\alpha & \alpha^5+\alpha^4+\alpha+1 & \alpha^5+\alpha^4+\alpha+1 \end{pmatrix} \tag{3.30}$$

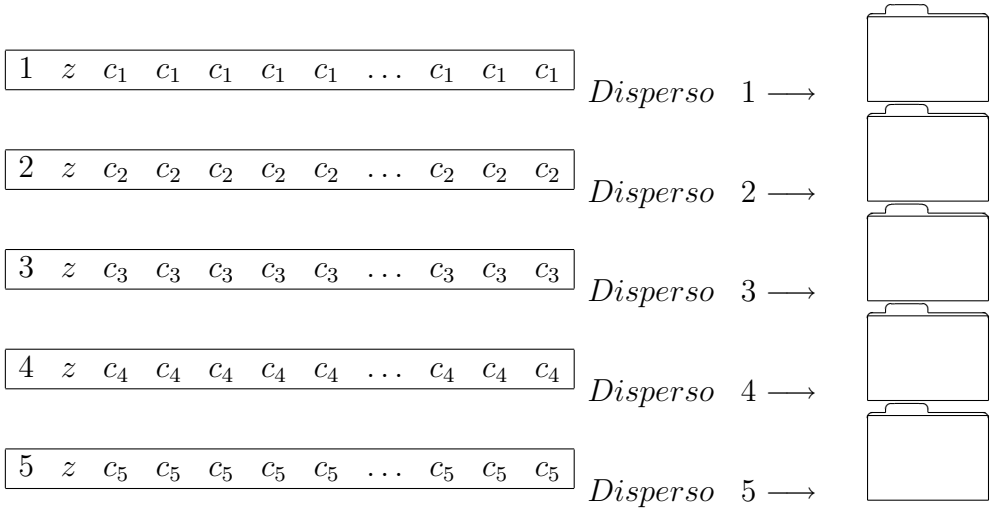
Una vez establecidos los parámetros m, n, k y la matriz $\mathbf{A}$, se secciona el archivo ($\mathbf{F}$) en ternas de vectores (puesto que $m = 3$). Si la longitud del archivo no es múltiplo de tres, se agregan al final del archivo cuantos ceros sean necesarios para que lo sea. El número de ceros agregados debe ser comunicado al receptor para que no los tome en cuenta en la reconstrucción. Esta información también se agrega al disperso.

$$F = \; b_1, \;\; b_2, \;\; b_3, \;\; b_4, \;\; b_5, \;\; b_6, \;\; \ldots \;\; b_{i-1}, \;\; b_i, \;\; 0 \tag{3.31}$$

En el siguiente paso del algoritmo, se multiplica cada terna de vectores por la matriz $\mathbf{A}$, generando un vector $\tilde{\mathbf{c}}$ con cinco elementos por cada producto.

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \\ a_{51} & a_{52} & a_{53} \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{pmatrix} \tag{3.32}$$

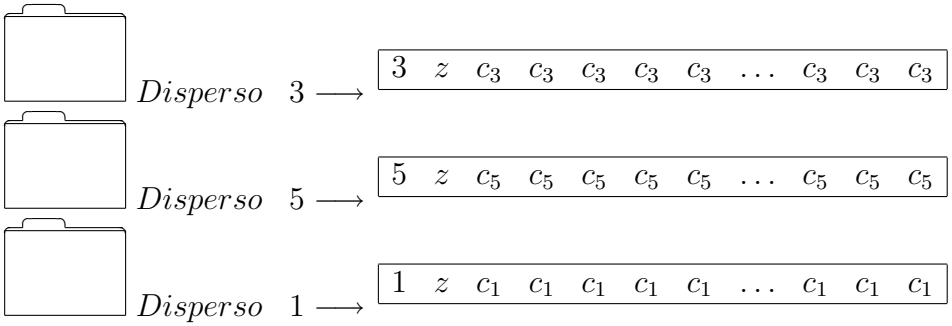
Esta operación se repetirá $\mathbf{N} = |\mathbf{F}|/\mathbf{3}$ veces, donde $|\mathbf{F}|$ es el número de caracteres de $\mathbf{F}$. Por tanto, cada disperso es de longitud $(|\mathbf{F}|/\mathbf{3}) + \mathbf{2}$ **bytes**. Los dos bytes agregados corresponden al byte necesario para representar el renglón $\mathbf{i}$ de la matriz $\mathbf{A}$ que dió origen a ese disperso y el byte $\mathbf{z}$ que indica el número de ceros agregados al final de la imagen para completar las ternas.



Cada disperso será almacenado en una terminal distinta.

3.5.2. Reconstrucción

Para la recuperación del archivo, el receptor necesita cualesquiera tres de los cinco dispersos. El programa de control o el propio usuario se encargará de buscar los dispersos en las terminales de almacenamiento predestinadas. En caso de recuperar los cinco dispersos, se desecharán dos aleatoriamente seleccionados.



$$
\begin{array}{ccccc}
Indicador\ recuperado & \longrightarrow & Renglón & de\ la & matriz\ A \\
3 & \longrightarrow & a_{31} & a_{32} & a_{33} \\
5 & \longrightarrow & a_{51} & a_{52} & a_{53} \\
1 & \longrightarrow & a_{11} & a_{12} & a_{13}
\end{array}
$$

Una vez que se tienen los tres dispersos distintos entre sí, procedemos a la reconstrucción de la imagen. Para ello necesitamos generar la inversa de la matriz $\mathbf{B}$ $(\mathbf{3x3})$ compuesta por los tres renglones de la matriz $\mathbf{A}$ que dieron origen a estos tres dispersos.

Dado que se utilizan tres de cinco dispersos y que no importa el orden de recuperación de éstos para generar la matriz $\mathbf{B}$, podemos tener hasta $\binom{\mathbf{5}}{\mathbf{3}}$ matrices distintas, por ejemplo:

$$
\begin{pmatrix} a_{31} & a_{32} & a_{33} \\ a_{51} & a_{52} & a_{53} \\ a_{11} & a_{12} & a_{13} \end{pmatrix}, \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{31} & a_{32} & a_{33} \\ a_{51} & a_{52} & a_{53} \end{pmatrix}, \cdots, \begin{pmatrix} a_{51} & a_{52} & a_{53} \\ a_{31} & a_{32} & a_{33} \\ a_{11} & a_{12} & a_{13} \end{pmatrix}, \begin{pmatrix} a_{51} & a_{52} & a_{53} \\ a_{11} & a_{12} & a_{13} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}
$$

Una vez que tenemos **D**, la matriz inversa de **B**, la multiplicamos por cada terna $\vec{\mathrm{d}_i}$ para reconstruir $\vec{\mathrm{b}_i}$

$$
\begin{pmatrix} d_{10} & d_{11} & d_{12} \\ d_{20} & d_{21} & d_{22} \\ d_{30} & d_{31} & d_{32} \end{pmatrix} \cdot \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \tag{3.33}
$$

Este proceso continúa hasta terminar con el contenido de vectores $\tilde{\mathbf{c}}$. Antes de generar la imagen final se deben eliminar los $\mathbf{z}$ datos finales de relleno que indique el disperso.

$$
F = \ b_1, \ \ b_2, \ \ b_3, \ \ b_4, \ \ b_5, \ \ b_6, \ \ \ldots \ \ b_{|F|} \tag{3.34}
$$

Capítulo 4
Manipulador de Imágenes Médicas

El Manipulador de Imágenes Médicas (figura 4.1) es un dispositivo que está formado por una interfaz USB y un FPGA SPARTAN equivalente a $500,000$ compuertas en el que están configurados, los algoritmos wavelet análisis, wavelet síntesis, codificador, dispersión y reconstrucción, descritos en el capítulo anterior.

El proceso de construcción del manipulador consistió en las siguientes etapas:

- Programación de los algoritmos seleccionados en C++ para generar la estrategia a seguir para su posterior descripción en VHDL. Los listados de algunos de estos programas se encuentran en el Anexo A.

- Diseño, descripción y prueba por separado de los componentes requeridos por cada algoritmo. Por la naturaleza del lenguaje de descripción utilizado, fue posible describir componentes independientes y de diferentes niveles de complejidad, los cuales pueden interconectarse con otros componentes modulares para generar componentes más complejos. Los niveles de complejidad se asignaron de la siguiente manera:

  - Nivel uno: el componente está formado solamente por circuitos de compuertas.
  - Nivel dos: el componente contiene componentes de nivel uno interconectados entre sí o con compuertas.
  - Nivel tres: el componente involucra componentes de nivel dos interconectados entre sí, con componentes de nivel uno o con compuertas.
  - Nivel cuatro: el componente comprende componentes de nivel tres conectados con componentes del mismo nivel o de niveles inferiores.

- Integración y prueba de los componentes de bajo nivel en un componente de nivel cuatro (o superior), que ejecuta uno de los algoritmos requeridos.

Es importante aclarar que cada componente fue evaluado de manera independiente y que el desempeño global del manipulador se probó utilizando como entrada a cada componente la salida del componente anterior. Se siguió esta metodología debido a que para la integración final de todos los componentes era necesaria la incorporación de memorias

externas y posiblemente la utilización de un Pipeline, lo que escapa de los objetivos de la presente tesis.

En las siguientes secciones se ampliará el proceso de implementación del manipulador, de acuerdo a la siguiente estructura. En primer lugar, el funcionamiento de cada componente será descrito de manera global (en las secciones 4.1;4.2;4.3,4.4 y 4.5), se presentará su diagrama general, con sus entradas y salidas. Posteriormente, se mostrará el diagrama interno donde se pueden observar los diferentes componentes de que está compuesto y cómo se interconectan. El siguiente paso será la descripción detallada de cada bloque, su diagrama interno, tabla de verdad (cuando la haya) y simulación.

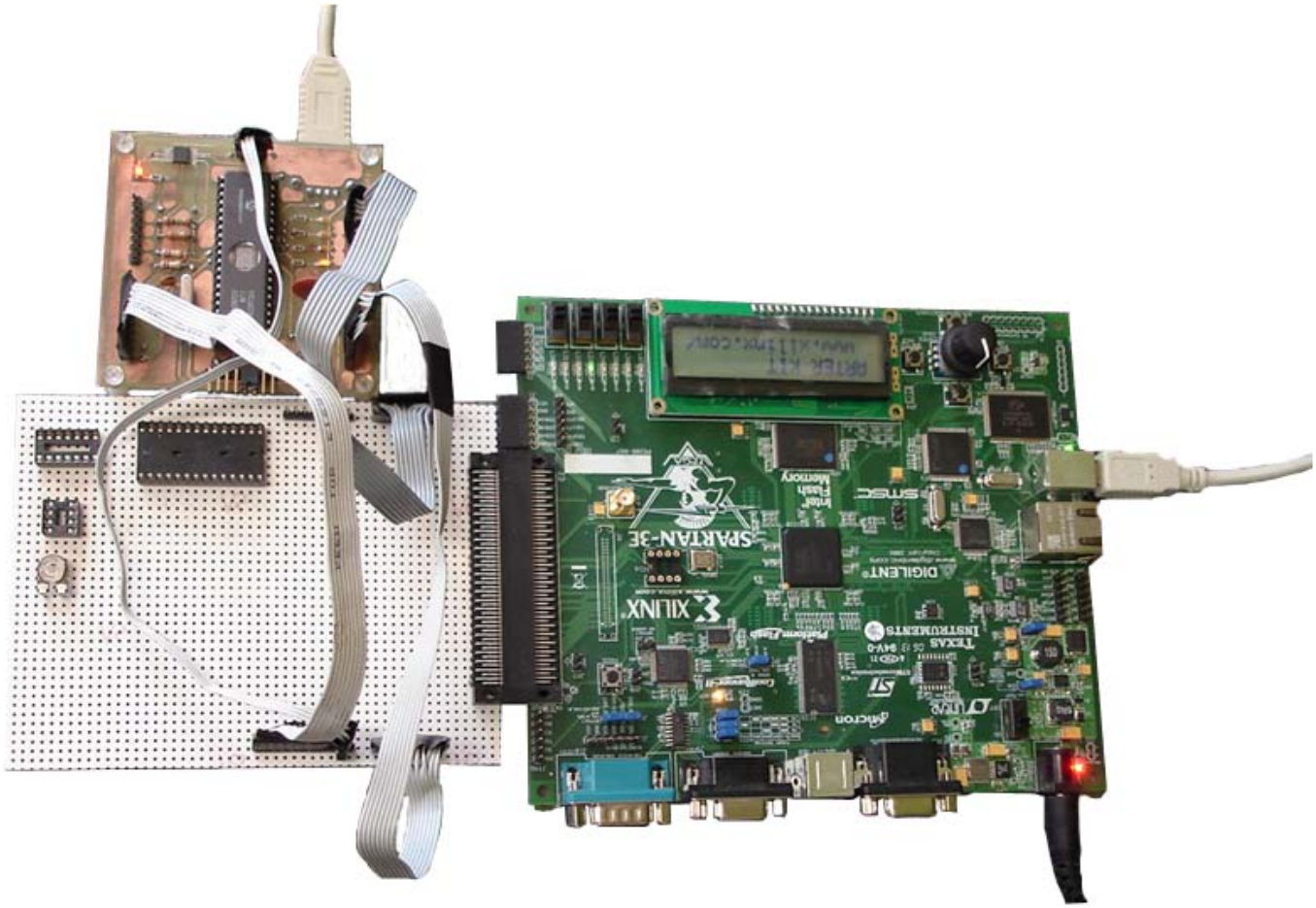


Figura 4.1: *Manipulador de Imágenes. Consta de una interfaz USB 1.0 (tarjeta superior izquierda), un conector FX2-100S-1 (tarjeta inferior izquierda) y un FPGA SPARTAN 3E equivalente a 500,000 compuertas (tarjeta central). La interfaz USB 1.0 que esta construida con base en un PIC16C765 ingresa los datos provenientes de la PC al FPGA mediante el conector FX2-100S-1.*

4.1. Unidad Aritmética Lógica para elementos del campo finito F_{2^8}

Se describió en VHDL un componente llamado **ALU** (Arithmetic Logic Unit, Unidad Aritmética Lógica) capaz de procesar, con datos de ocho bits y aritmética de campos finitos, las operaciones: multiplicación, división, suma y resta (figura 4.2). La unidad es totalmente combinacional, por lo tanto, el tiempo que le llevará realizar una operación dependerá unicamente de los tiempos de propagación de las señales de entrada a la salida.

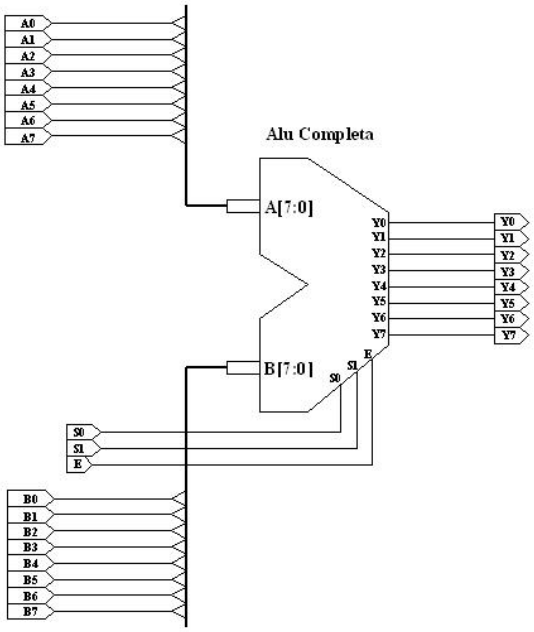


Figura 4.2: *Componente de nivel tres llamado **Unidad Aritmética Lógica ALU**. Efectúa de manera combinacional las operaciones de suma-resta, multiplicación y división para elementos del campo finito* F_{2^8}*. Consta de: dos entradas de datos **A[7:0]** y **B[7:0]**, de ocho bits cada una, en las cuales se reciben los operandos; tres entradas de control de un bit cada una,* **S0**, **S1** *y* **E**; *y una salida de ocho bits **Y[7:0]**, por donde se obtiene el resultado de la operación seleccionada por los bits de control* **S0** *y* **S1**.

La **ALU** realiza las operaciones de multiplicación, división y suma (que es equivalente a la resta) de manera simultánea sobre los datos **A[7:0]** y **B[7:0]**, de tal forma que el resultado de cada operación se encuentra disponible en las entradas de un **Multiplexor** el cual, a partir de las entradas **S0** y **S1**, selecciona el resultado de la operación deseada, que será mostrado en la salida **Y[7:0]**. La entrada **E** deshabilita este componente, llevando todas sus salidas a un estado cero cuando sea necesario, como en el caso de intentar una multiplicación o una división con un operador igual a cero.

En la figura 4.3 se muestra el diagrama interno del componente de nivel tres **ALU**, el cual está compuesto por cuatro componentes de menor nivel: **Multiplicador**, **Divisor**, **Sumador−Restador** y un **Multiplexor de tambor** el cual emplea las señales de control para seleccionar el resultado de una de estas operaciones para mostrarlo en la salida **Y[7:0]**.

Cada componente de menor nivel fue construido y probado independientemente, para asegurar que el diseño fuera confiable. A continuación serán descritos los componentes individuales que integran al componente **ALU**.

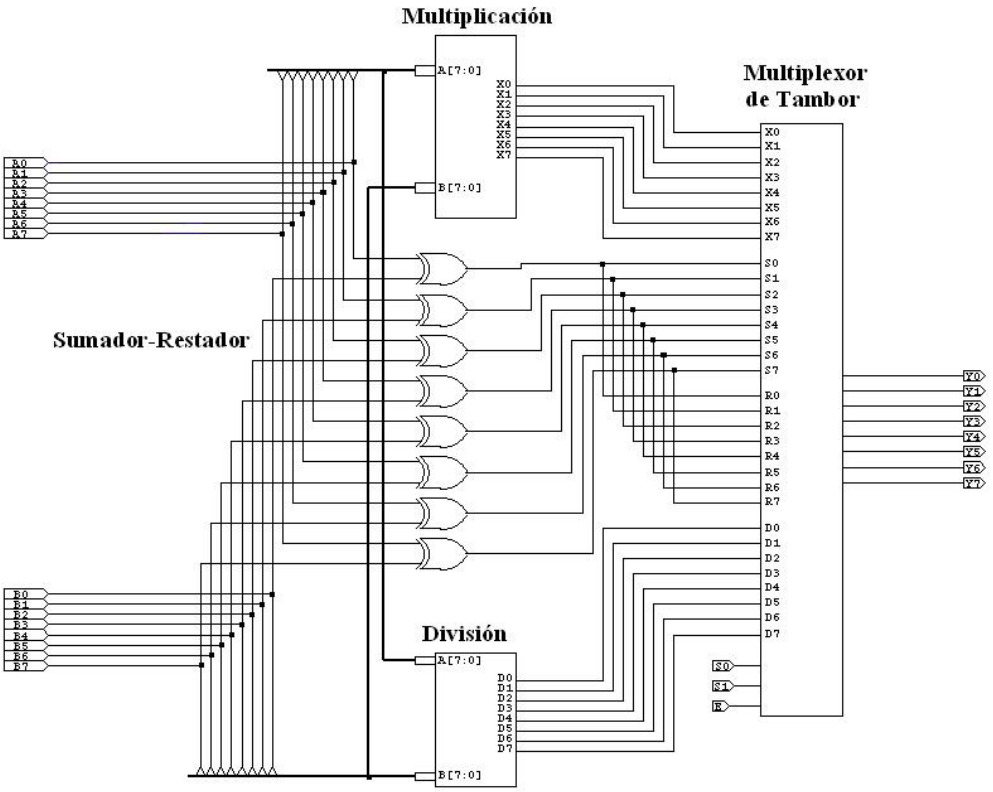


Figura 4.3: *Diagrama interno de la **Unidad Aritmética Lógica ALU**. Se muestran las interconexiones de los elementos que lo integran: **Multiplicador**, **Divisor**, **Sumador-Restador** y un **Multiplexor de tambor**. El componente realiza simultáneamente las cuatro operaciones sobre los datos leidos en las entradas de ocho bits **A** y **B**. Con las entradas de control **S0** y **S1**, de un bit cada una, seleccionamos qué resultado será mostrado en la salida **Y[7:0]**, de ocho bits: si la combinación* [**S1** : **S0**] *vale cero, en la salida se mostrará el producto; si vale uno, la suma; si es dos, la resta y si es tres, la división. La entrada de control **E** deshabilita el funcionamiento de la **Unidad Aritmética Lógica ALU** llevando la salida **Y[7:0]** a cero cuando es necesario, por ejemplo, cuando se intenta multiplicar o dividir por un dato que valga cero.*

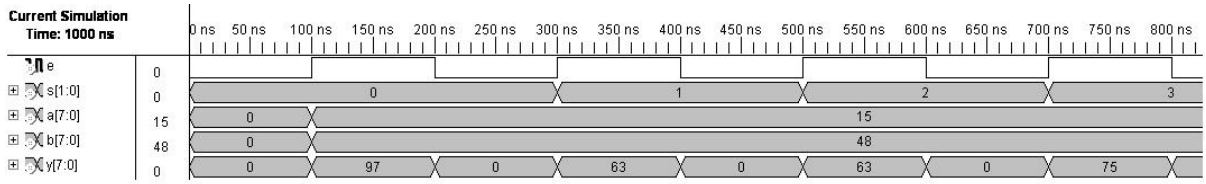


Figura 4.4: *Simulación del componente de nivel tres* ***Unidad Aritmética Lógica ALU****. La combinación formada por los bits de las entradas* ***S0*** *y* ***S1*** *seleccionan el resultado de una de las cuatro operaciones realizadas por el componente (0=multiplicación, 1=suma, 2=resta y 3=división). Si la entrada E es puesta a cero, la salida será forzada a cero.*

4.1.1. Componente Sumador-Restador

La suma y la resta en aritmética de campos finitos F_{2^8} consiste en efectuar la operación XOR bit a bit entre los operandos. Por lo tanto, generamos un componente de nivel uno llamado **Sumador-Restador** (figura 4.5) el cual, para sumar (o restar) los datos en sus entradas de ocho bits **A[7:0]** y **B[7:0]** realiza la operación XOR bit a bit entre ellos, generando una salida de ocho bits denominada **S[7:0]**. Consta de ocho compuertas XOR de dos entradas.

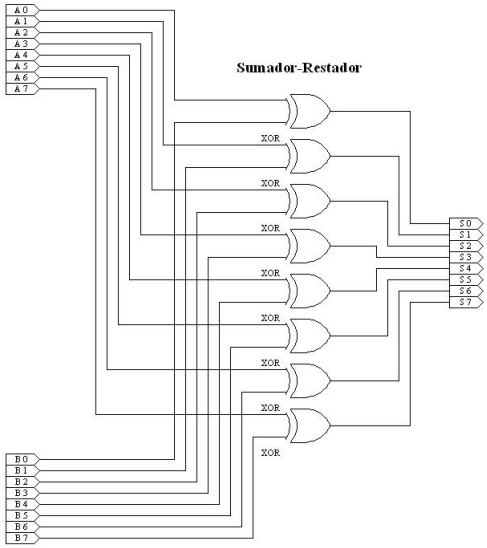


Figura 4.5: *Diagrama del componente de nivel uno que efectúa las operaciones de suma y resta, que en el campo finito* F_{2^8} *son iguales. Este componente consiste en ocho compuertas XOR de dos entradas. Recibe en sus entradas* ***A[7:0]*** *y* ***B[7:0]*** *los datos en su representación polinomial, sobre los cuales efectúa la operación* XOR *bit a bit, generando un dato de ocho bits en la salida* ***S[7:0]****.*

En la figura 4.6 se puede apreciar la simulación del funcionamiento de este componente.

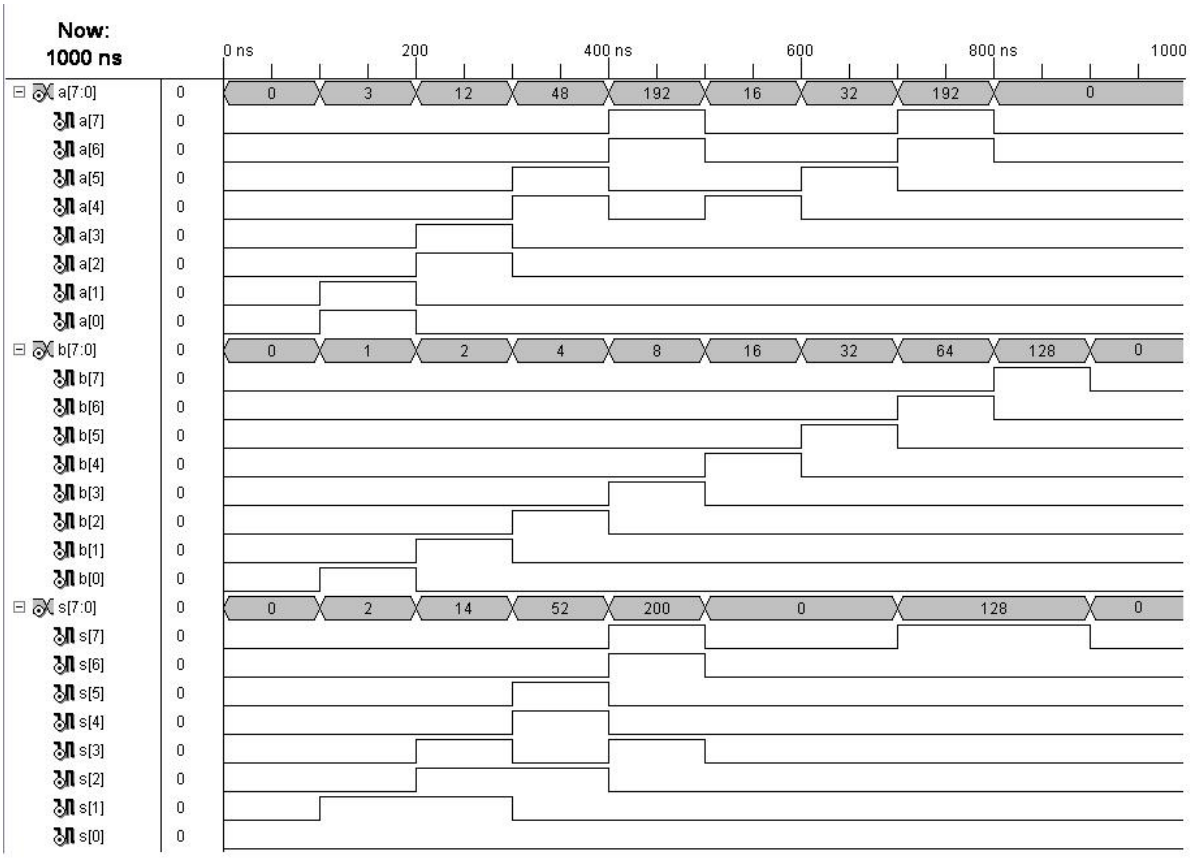


Figura 4.6: *Simulación de la operación Suma-Resta. Realizando las operaciones manualmente es posible verificar que los resultados obtenidos son correctos.*

4.1.2. Componente Multiplicador

Este componente de nivel dos genera el producto ($\mathbf{X}[\mathbf{7}:\mathbf{0}]$) de dos datos, $\mathbf{A}[\mathbf{7}:\mathbf{0}]$ y $\mathbf{B}[\mathbf{7}:\mathbf{0}]$, sobre el campo finito $F_{\mathbf{2^8}}$. El algoritmo en el cual se basó la descripción en VHDL (sección 3.1.3) fue programado en C++ , y puede ser consultado en el apéndice A. El **Multiplicador** (figura 4.7) está compuesto por varios componentes de menor nivel, cada uno de los cuales será descrito en detalle más adelante.

Su funcionamiento consiste en los siguientes pasos:

1. Verificar que las entradas ($\mathbf{A}[\mathbf{7}:\mathbf{0}]$,$\mathbf{B}[\mathbf{7}:\mathbf{0}]$) sean distintas de cero. En caso contrario, la salida del **Multiplicador** ($\mathbf{X}[\mathbf{7}:\mathbf{0}]$) es forzada a cero. Esta comprobación se realiza con ayuda de dos componentes **Cero** y uno **Compuerta**.

2. Si los datos de entrada son distintos de cero se obtiene su representación logarítmica, función que es realizada por los componentes **Logaritmo**.

3. Los logaritmos de $\mathbf{A}[\mathbf{7}:\mathbf{0}]$ y $\mathbf{B}[\mathbf{7}:\mathbf{0}]$ deben ser entonces sumados módulo 255 para aseguarar que el resultado de la multiplicación sea un elemento del campo. Esta operación es efectuada mediante la combinación de dos componentes **Sumador** y un **Comparador**.

4. Finalmente, se obtiene el antilogaritmo del total mediante el componente **Antilogaritmo**, que es el resultado final de la multiplicación.

Es importante mencionar que, cuando hablamos de utilizar u obtener la representación logarítmica o exponencial, los valores manejados por el FPGA son los exponentes a los cuales se eleva la base α. Estos exponentes son los que se emplean para realizar las sumas, restas y obtención de antilogaritmos.

Ejemplo

$$\textbf{Multiplicación} \quad \mathbf{A}\cdot\mathbf{B}$$

$$\mathbf{A} = 01000000 = \alpha^6$$

$$\mathbf{B} = 01000000 = \alpha^6$$

$$\log_\alpha(\mathbf{A}(\alpha)) = \log_\alpha(01000000) = 6$$

$$\log_\alpha(\mathbf{B}(\alpha)) = \log_\alpha(01000000) = 6$$

$$\mathbf{A}(\alpha)\cdot\mathbf{B}(\alpha) = \alpha^{\mathbf{6+6}} = \alpha^{\mathbf{12}}$$

$$\alpha^{12 \ \% \ 255 \ = \ 12} = \alpha^{12}$$

$$\mathbf{X} = \text{Antilog}_\alpha\ (12) = \alpha^5 + \alpha^4 + \alpha^2 + \alpha = 00110110$$

Para resolver el problema que representa el hecho de que la transformación logarítmica de cero no está definida, por conveniencia se optó por asignarle un cero como valor por defecto. Puesto que el logaritmo de uno también es cero, se genera una ambigüedad en el significado de obtener un cero a la salida de los componentes **Logaritmo**. Esta decisión conlleva incorporar elementos de control para distinguir cuándo un cero representa un valor por defecto o el logaritmo de uno.

En la figura 4.8 se muestra la simulación de este componente. Es importante resaltar que cuando se realizan operaciones de multiplicación cuyo producto no sobrepasa el valor de 255 -el máximo valor del campo finito-, el resultado es muy similar a la operación de multiplicación de números enteros; sin embargo, si el resultado es mayor que 255, el producto repetirá alguno de los valores del campo.

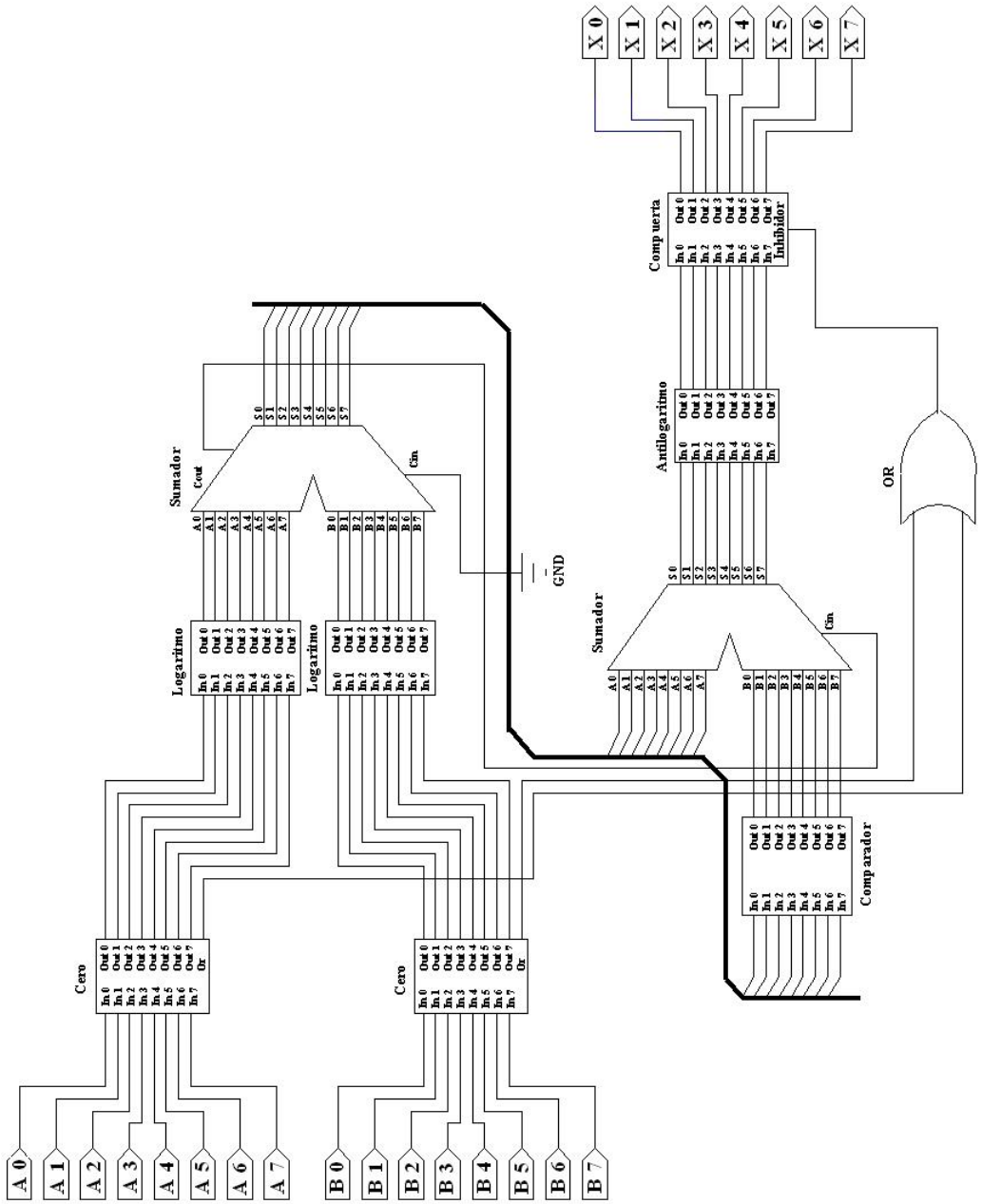


Figura 4.7: *Diagrama interno del componente nivel dos* ***Multiplicador****, que efectúa el producto entre los datos de ocho bits colocados en sus entradas* $\mathbf{A}[\mathbf{7}:\mathbf{0}]$ *y* $\mathbf{B}[\mathbf{7}:\mathbf{0}]$ *y lo entrega por su salida* $(\mathbf{X}[\mathbf{7}:\mathbf{0}])$*. Está integrado por diversos componentes de nivel uno interconectados: dos componentes* ***Cero****, que verifican que los datos a multiplicar sean distintos de cero; dos componentes* ***Logaritmo****, que transforman los datos de una representación vectorial a una exponencial; dos componentes* ***Sumador****, uno realizará la suma de exponenciales y el otro la operación de módulo; un componente* ***Comparador****, que verifica si el resultado del primer sumador es* 255*; un* ***Antilogaritmo*** *para regresar de la representación exponencial a la vectorial; y un componente* ***Compuerta****, que inhibe al multiplicador en caso de que la salida de alguno de los componentes* ***Cero*** *se ponga en nivel alto, indicando que fue detectado un cero a la entrada de la* ***ALU****.*

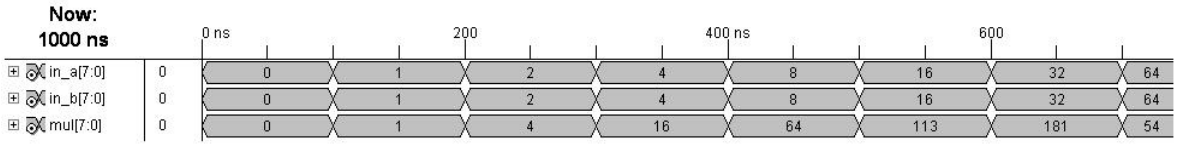


Figura 4.8: *Simulación del componente* ***Multiplicador****. Las operaciones realizadas son la obtención del cuadrado de las potencias de 2. A partir de la operación* 16×16*, se observa la cerradura del campo sobre la multiplicación.*

A continuación se describen cada uno de los componentes de menor nivel que forman el **Multiplicador** y presentan las simulaciones de su operación.

Componente Cero

El componente **Cero** (figura 4.9) es un componente de nivel uno implementado en forma de buffer que deja pasar la entrada **In[7:0]** a salida **Out[7:0]**. Si la entrada es igual a cero, la salida **Nor** se activa en el estado alto; en caso contrario, permanece en el estado bajo.

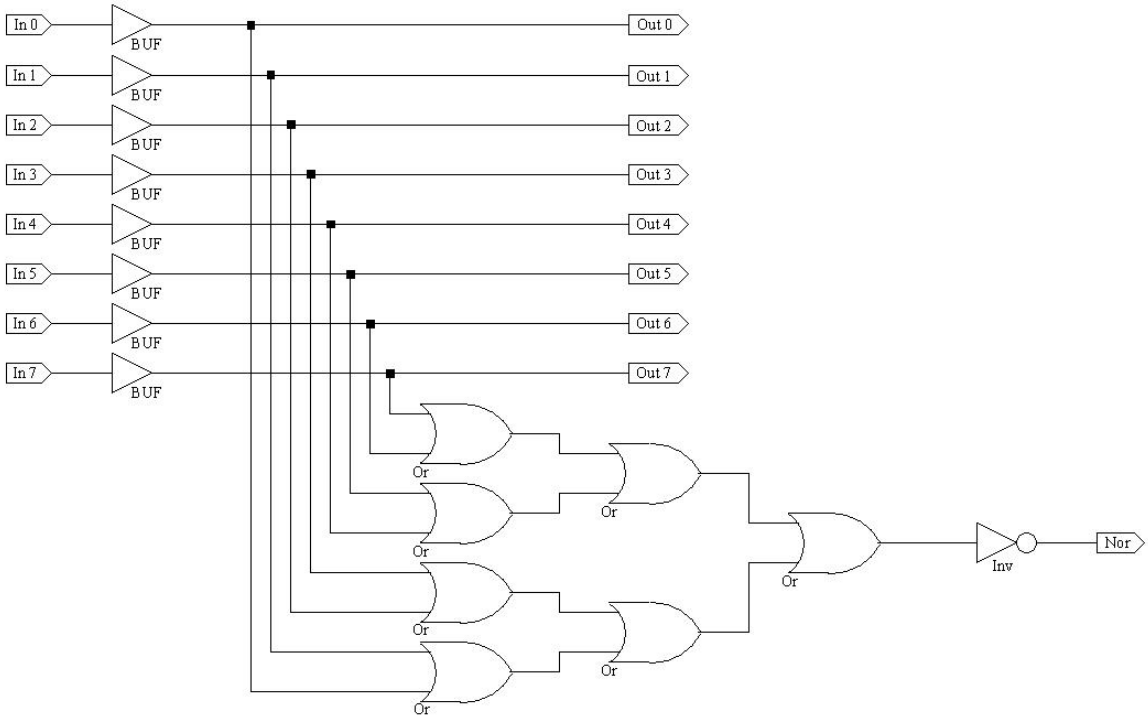


Figura 4.9: *Diagrama interno del componente de nivel uno* ***Cero****. Si la entrada es igual a cero, la salida* **Nor** *valdrá uno. En caso contrario, valdrá cero.*

En la simulación (figura 4.10) se aprecia que la salida **Nor** realiza la función de una compuerta **NOR**.

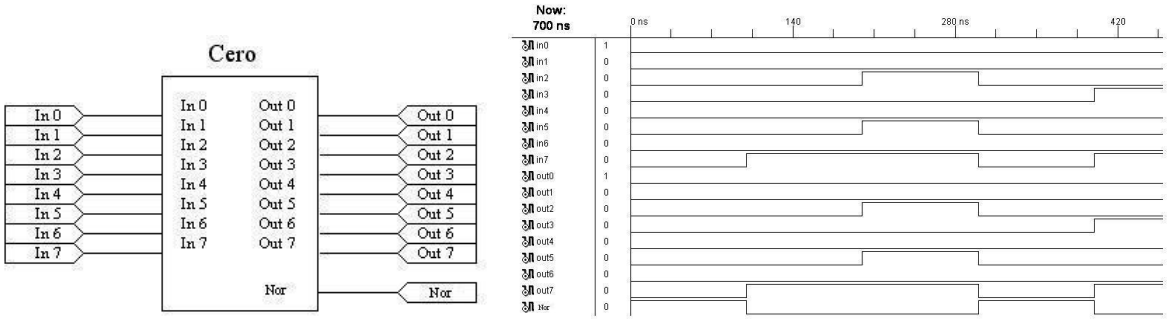


Figura 4.10: *Diagrama a bloques y simulación del componente* ***Cero****. En esta última es posible verificar que el componente funciona como se esperaba.*

Componente Logaritmo

Logaritmo es un componente de nivel uno que se encarga de generar el logaritmo base α de la entrada **In[7:0]** y lo muestra en la salida **Out[7:0]**, es decir, mapea cada polinomio de su representación vectorial a su correspondiente representación exponencial.

En la figura 4.11 se puede observar el diagrama a bloques y la simulación del componente **Logaritmo**.

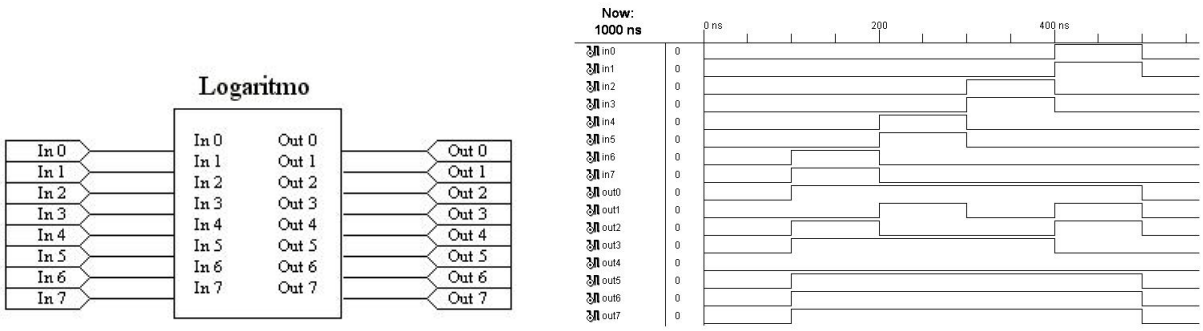


Figura 4.11: *Diagrama a bloques y simulación del componente* ***Logaritmo****, de nivel uno.*

Para la elaboración de este componente se generó una tabla de verdad de ocho bits de entrada y de salida, donde la combinación de salida equivale al logaritmo de la entrada. Por razones prácticas, de aquí en adelante se omitirán las comas en la representación vectorial.

Obtuvimos las funciones booleanas de las ocho salidas que representan a la tabla de verdad. Cada salida fue simplificada y representada por una función de suma de productos, descrita en VHDL.

Representación Vectorial del Polinomio								Logaritmo base α del polinomio							
$In7$	$In6$	$In5$	$In4$	$In3$	$In2$	$In1$	$In0$	$Out7$	$Out6$	$Out5$	$Out4$	$Out3$	$Out2$	$Out1$	$Out0$
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	1	1	1	0	0	1	1	1
0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	1	1	1	0	0	1	1	1	1
				⋮								⋮			
1	1	1	1	0	0	0	0	1	0	1	1	1	0	1	1
1	1	1	1	0	0	0	1	1	1	1	0	1	1	1	0
1	1	1	1	0	0	1	0	0	1	1	1	1	1	1	0
1	1	1	1	0	0	1	1	0	0	0	1	0	0	0	0
1	1	1	1	0	1	0	0	1	1	0	0	0	0	0	1
1	1	1	1	0	1	0	1	1	0	1	0	0	1	0	1
1	1	1	1	0	1	1	0	1	1	0	0	1	0	0	1
1	1	1	1	0	1	1	1	0	0	1	0	1	1	1	1
1	1	1	1	1	0	0	0	1	0	0	1	0	1	0	1
1	1	1	1	1	0	0	1	1	1	0	1	1	0	0	0
1	1	1	1	1	0	1	0	1	1	0	0	0	1	1	0
1	1	1	1	1	0	1	1	1	0	1	0	1	1	0	0
1	1	1	1	1	1	0	0	0	1	1	0	0	0	0	0
1	1	1	1	1	1	0	1	0	1	1	0	0	1	0	0
1	1	1	1	1	1	1	0	1	0	1	0	1	1	1	1
1	1	1	1	1	1	1	1	0	1	0	1	0	1	1	1

Cuadro 4.1: *Tabla de verdad del componente **Logaritmo**. Se aprecia la ambigüedad del significado de un cero a la salida, causada por la asignación de un cero como valor por defecto para una entrada igual a cero.*

Componente Sumador

Este componente de nivel uno tiene dos entradas de datos de ocho bits $\mathbf{A}[\mathbf{7}:\mathbf{0}]$ y $\mathbf{B}[\mathbf{7}:\mathbf{0}]$; una entrada de acarreo de un bit **Cin**, una salida de ocho bits $\mathbf{S}[\mathbf{7}:\mathbf{0}]$ y una salida de acarreo, llamada **Cout** (figura 4.12). Este componente efectúa la suma entera de dos datos de ocho bits **A** y **B**. Si el resultado es menor a 255, la salida $\mathbf{S}[\mathbf{7}:\mathbf{0}]$ mostrará el valor de la suma y **Cout** valdrá 0; pero si la suma es mayor a 255, en los ocho bits de la salida $\mathbf{S}[\mathbf{7}:\mathbf{0}]$ se encontrará el valor de la suma módulo 256 y **Cout** valdrá 1. Cuando el bit de entrada **Cin** vale 1, se adiciona este valor a la suma.

En la figura 4.13 se encuentra la simulación de la operación de este componente. En ella se aprecia que, mientras el valor de la suma sea menor a 255, la salida $\mathbf{S}[\mathbf{7}:\mathbf{0}]$ reflejará directamente este resultado y **Cout** valdrá cero, pero en cuanto la suma supere este

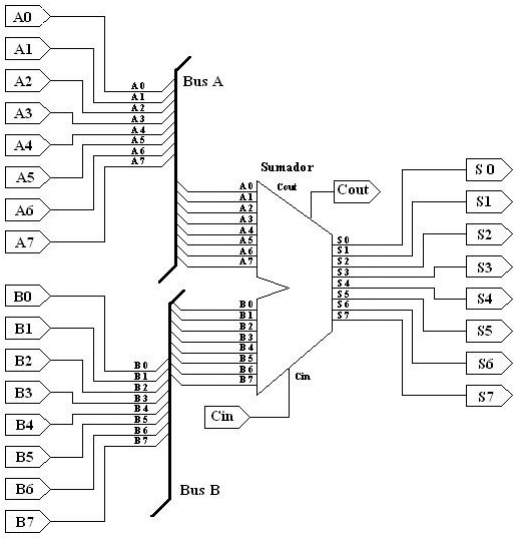


Figura 4.12: *Componente de nivel uno llamado* ***Sumador****. Es un sumador de ocho bits con entrada y salida de acarreo.*

límite, en $\mathbf{S}[\mathbf{7} : \mathbf{0}]$ se obtendrá un valor menor y **Cout** valdrá uno.

Componente Comparador

Comparador es un componente de nivel uno que se encarga de verificar si la entrada **In[7:0]** equivale a 255, valor que está descrito como constante dentro de este componente. Si la comparación es positiva, es decir, la entrada vale 255, la salida **Out[7:0]** genera el valor 1. En caso de que la comparación sea negativa, la salida valdrá 0. Este componente está constituido por quince compuertas AND de dos entradas (fig. 4.14). El valor 255 se construye al contectar una de las entradas de las primeras ocho compuertas a Vcc.

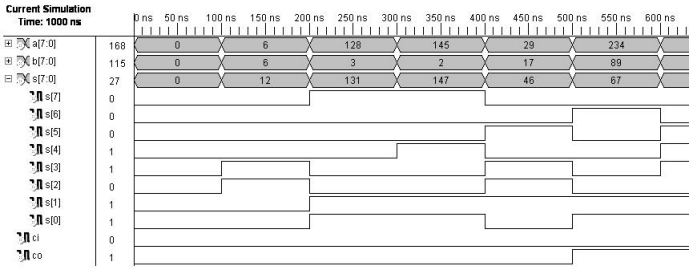


Figura 4.13: *Simulación del componente* ***Sumador****, de nivel uno. La salida* $\mathbf{S}[\mathbf{7} : \mathbf{0}]$ *contiene el resultado de sumar los valores colocados en las entradas* $\mathbf{A}$ *y* $\mathbf{B}$*. Si esta suma sobrepasa* 255*, máximo valor posible de representar con ocho bits, en* $\mathbf{S}[\mathbf{7} : \mathbf{0}]$ *se encontrará el resultado módulo* 256 *y la salida* ***Cout*** *cambiará a uno.*

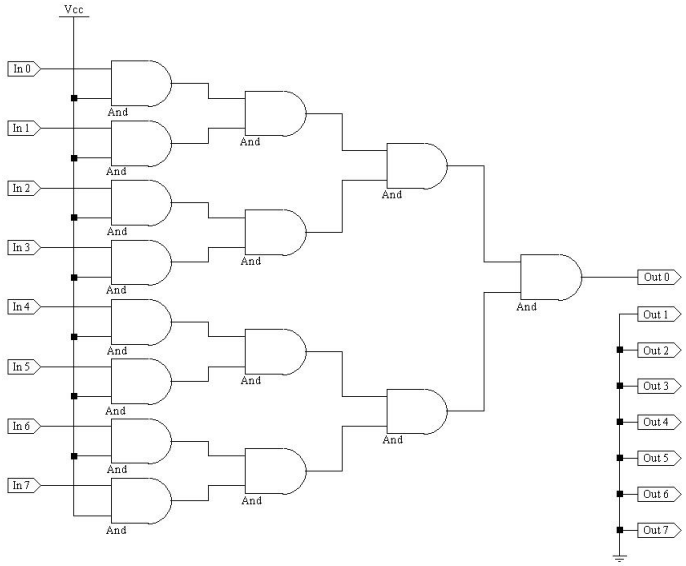


Figura 4.14: *Diagrama Interno del componente* ***Comparador****, de nivel uno. Es visible su construcción netamente combinacional.*

En la figura 4.15 se encuentra el diagrama general y la simulación del funcionamiento del **Comparador**.

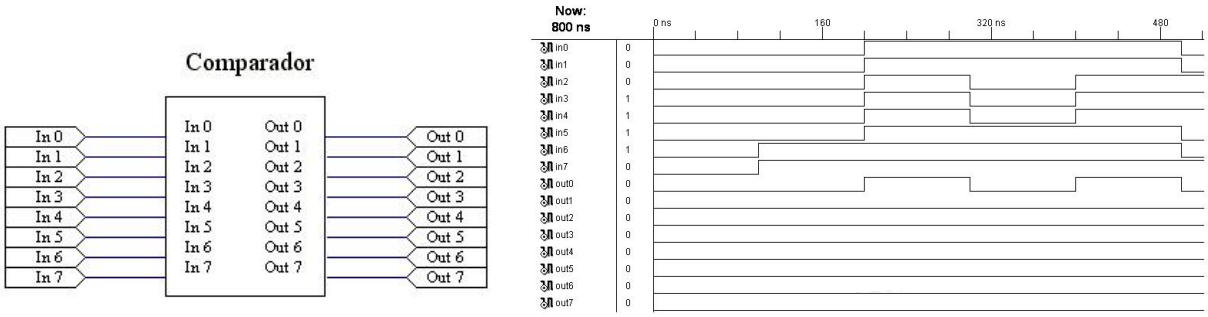


Figura 4.15: *Diagrama a bloques y simulación del componente* ***Comparador****. Verifica si la entrada* ***In[7:0]*** *vale 255, en cuyo caso obtenemos un uno en la salida* ***Out[7:0]****. En caso contrario,* ***Out[7:0]*** *vale cero.*

Operación Suma Módulo 255

Al efectuar la suma de los exponentes durante la multiplicación sobre el campo finito, se pueden presentar tres casos:

1. El resultado es menor a 255. En este caso, el valor entregado por la salida **S[7:0]** será un elemento válido del campo F_{2^8}, por lo que no será necesario hacerle ninguna operación adicional.

2. El resultado es mayor a 255. En la descripción del componente **Sumador** se mencionó que los ocho bits de la salida **S[7:0]** representan la suma módulo 256. Sin embargo, para que el campo sea cerrado en la multiplicación, requiere que la multiplicación sea módulo 255. Gracias a que el resultado de la operación módulo 255 es igual a módulo $256 + 1$, es posible resolver fácilmente este problema: basta con conectar la entrada **A** de un segundo **Sumador** ($A2$) a la salida **S** del primero ($S1$), colocando un cero en la otra entrada **B2**. Al conectar la salida **Cout** del primer **Sumador** (**Cout1**) con la entrada **Cin** del segundo (**Cin**), se asegura que cada vez que la suma de los exponentes sea mayor que 256, a la salida del primer sumador se le adicionará un uno. De esta manera, en la salida **S[7:0]** del segundo **Sumador**, $S2$ se obtendrá la suma módulo 255.

3. Pero si el resultado del primer **Sumador** es igual a 255, **Cout** no se activará, puesto que este valor es el más grande que se puede representar con los ocho bits de la salida $S1$. Sin embargo, es necesario sumarle un uno, puesto que el módulo 255 de 255 es 0. Para manejar este caso fue necesario intercalar un **Comparador** entre la salida $S1$ del primer **Sumador** y la entrada $B2$ del segundo. De esta manera, si $S1$ vale 255, el segundo **Sumador** recibirá un uno en su entrada $B2$ y lo sumará al valor de $S1$, presente en la entrada $A2$, generando así el módulo 255 en la salida $S2$. Pero si $S1$ es diferente a 255, la entrada $B2$ recibirá un cero. La configuración final resultantes se muestra en la figura 4.16.

Esta configuración realiza entonces dos operaciones:

Ejemplo

11111111	11111111
+00000001	+00000111
00000000	00000001
100000000	100000111

Cuadro 4.2: *Ejemplo de operaciones que se realizan para la operación de módulo* 255

S2 es el resultado de sumar **S1**+**B2**+**Cin2**. Si **S1** vale 255, **B2** recibirá un uno, mientras **Cin2** estará en nivel bajo. Si **S1** es mayor que 255, entonces **B2** valdrá cero y **Cin2** valdrá uno.

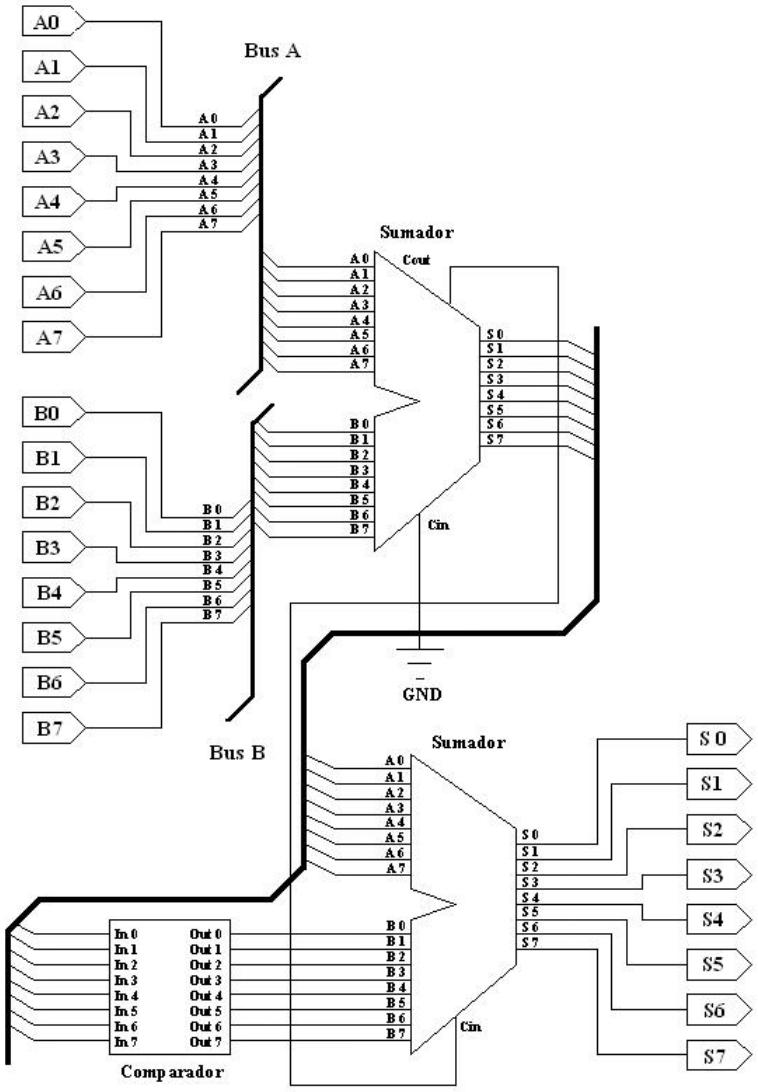


Figura 4.16: *Configuración de componentes que interconectados entre sí realizan la suma módulo 255, operación utilizada en la multiplicación y división sobre el campo finito* F_{2^8}.

Componente Antilogaritmo

El Componente **Antilogaritmo** es un componente de nivel uno (figura 4.17) que genera el antilogaritmo de la entrada **In[7:0]** y lo coloca en la salida **Out[7:0]**.

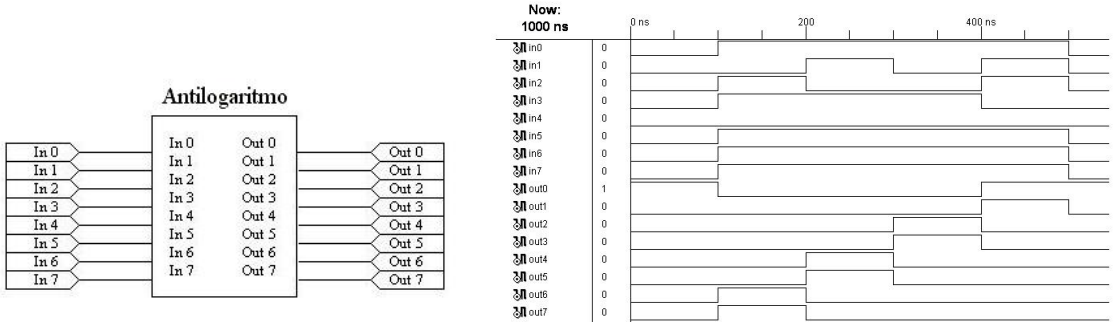


Figura 4.17: *Diagrama a bloques y simulación del componente **Antilogaritmo**. Las entradas de este componente reciben los valores obtenidos a la salida del componente **Logaritmo**, recuperándose en las salidas los valores antes de ser transformados a logaritmo.*

En la simulación podemos observar que el componente **Antilogaritmo** realiza la operación inversa a la del componente **Logaritmo**.

Logaritmo base α del polinomio								Representación Vectorial del Polinomio							
$Out7$	$Out6$	$Out5$	$Out4$	$Out3$	$Out2$	$Out1$	$Out0$	$In7$	$In6$	$In5$	$In4$	$In3$	$In2$	$In1$	$In0$
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0
1	1	1	0	0	1	1	1	0	0	0	0	0	0	1	1
				⋮								⋮			
1	0	1	0	1	1	0	0	1	1	1	1	1	0	1	1
0	1	1	0	0	0	0	0	1	1	1	1	1	1	0	0
0	1	1	0	0	1	0	0	1	1	1	1	1	1	0	1
1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	0
0	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1

Cuadro 4.3: *Tabla de verdad del componente **Antilogaritmo**. Se puede apreciar que esta tabla es la inversa de la utilizada para la transformación logarítmica.*

El procedimiento seguido para su elaboración fue muy similar al utilizado en la descripción del componente **Logaritmo**: elaborar la tabla de verdad (Cuadro 4.3), obtener las funciones booleanas que representan a las salidas de esta tabla y simplificarlas en sumas de productos, las cuales se describen en VHDL.

Componente Compuerta

Compuerta es un componente de nivel uno cuya función es dejar pasar los valores de la entrada **In** a la salida **Out** o forzar esta última a cero si recibe un nivel alto en su entrada **Inhibir**. Este bloque está descrito como ocho compuertas AND (fig. 4.18).

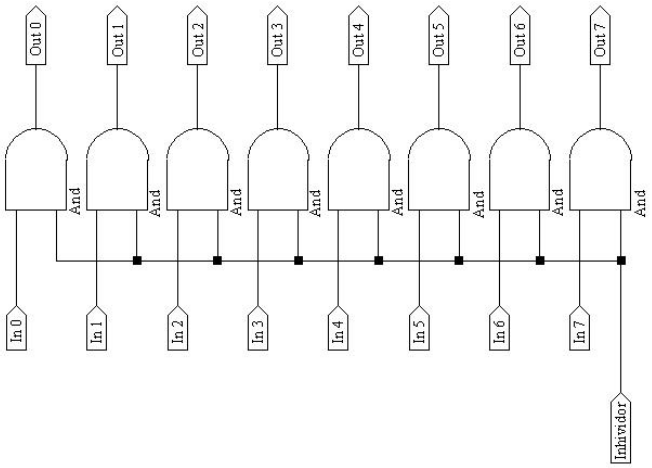


Figura 4.18: *Diagrama Interno del componente* ***Compuerta****, de nivel uno. Está formado por ocho compuertas AND. Cuenta con una entrada llamada* ***Inhibir*** *y una salida* ***Out****, de ocho bits cada una.*

En la figura 4.19 se encuentran, del lado izquierdo, la estructura externa del componente **Compuerta** y del lado derecho, una simulación de su funcionamiento, donde se observa que, en cuanto la entrada **Inhibir** cambia al estado alto, la salida **Out** es forzada a cero.

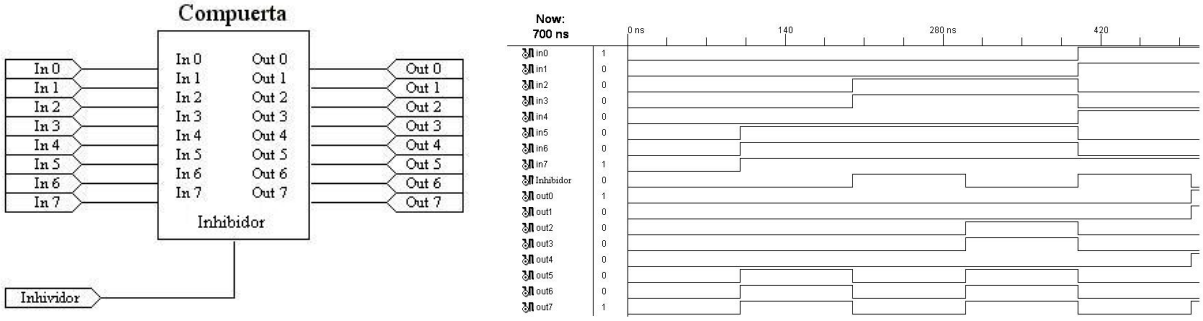


Figura 4.19: *Diagrama a bloques y simulación del componente* ***Compuerta****, donde se observa que si la entrada* **Inhibir** *contiene un uno, la salida* **Out** *vale cero. De lo contrario,* **Out** *es igual a* **In**.

4.1.3. Componente Divisor

Este componente de nivel dos calcula combinacionalmente la división (**X**) de dos datos (**A** y **B**) en campo finito F_{2^8}. El algoritmo en el cual se basó la descripción en VHDL también fue programado en C++ y se puede consultar en el apéndice A.

Así como la multiplicación de dos datos se obtiene sumando sus logaritmos, la división es la resta de los mismos. Para aprovechar la estructura empleada en el **Multiplicador**, podemos realizar la división como una suma de exponentes, siempre y cuando uno de los exponentes sea previamente convertido a su inverso multiplicativo, es decir, un α^m que sumado al α^n original dé como resultado α^{255}. Esta operación fue descrita en el capítulo de fundamentos (sección 3.1.3).

Ejemplo

$$\textbf{División} \quad \mathbf{A} \div \mathbf{B}$$

$$\mathbf{A} = 00110110$$

$$\mathbf{B} = 01000000$$

$$\log_\alpha(\mathbf{A}(\alpha)) = \log_\alpha(00110110) = \alpha^{12}$$

$$\log_\alpha(\mathbf{B}(\alpha)) = \log_\alpha(01000000) = \alpha^{6}$$

$$\mathbf{B}^{-1} = \alpha^{255-6} = \alpha^{249}$$

$$\mathbf{A}(\alpha) \div \mathbf{B}(\alpha) = \alpha^{\mathbf{12+249}} = \alpha^{\mathbf{261}}$$

$$\alpha^{261 \ \% \ 255 \ = \ 6} = \alpha^{6}$$

$$\mathbf{X} = \mathrm{Antilog}_\alpha \ (6) = \alpha^6 = 01000000$$

Gracias a esta propiedad de la aritmética de campo finito, la estructura de la operación división está basada en la estructura propuesta para la multiplicación, descrita en la sección anterior. En la figura 4.21 se puede observar el diagrama interno del componente de nivel dos **Divisor**. Este componente tiene dos entradas de ocho bits llamadas **A** y **B**, por donde ingresan el dividendo y el divisor, y una salida de ocho bits denominada **D**, que entrega el cociente. Cabe aclarar que, por ser una división sobre el campo finito, no se generan residuos.

El **Divisor** está compuesto por diversos componentes de nivel uno interconectados entre sí, algunos de los cuales ya fueron descritos en la sección del multiplicador. Como

está basado en la misma estructura que el componente **Multiplicador**, los componentes de nivel uno que integran al **Divisor** son prácticamente los mismos. La única diferencia es la adición del componente **Restador**, el cual calcula el inverso multiplicativo del dato en B.

En la simulación podemos observar el funcionamiento de este componente.

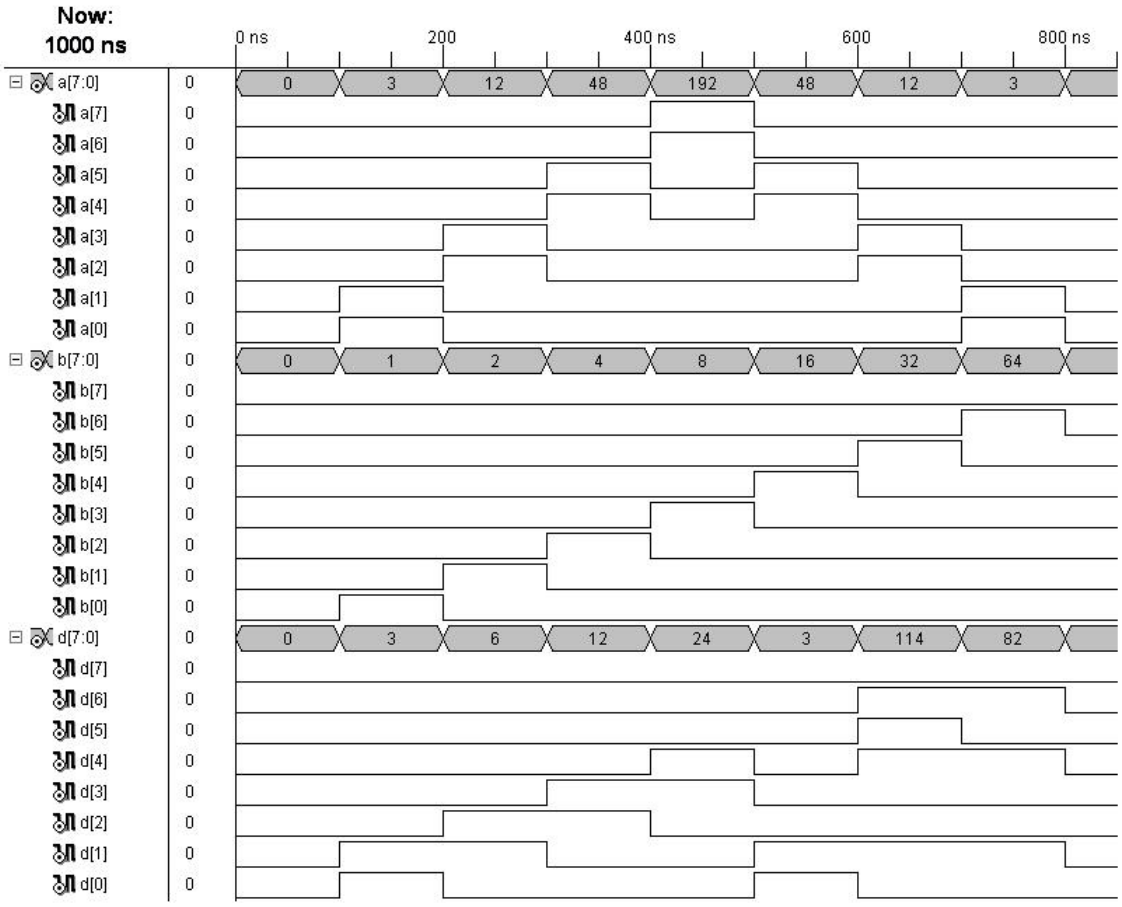


Figura 4.20: *Simulación del componente **División**. El dato en la entrada* **A** *es el resultado de la multiplicación efectuada en la sección anterior, mientras que el dato en la entrada* **B** *es uno de los factores multiplicados anteriormente. El resultado obtenido en* **D** *es el otro factor involucrado en la multiplicación. De esta manera, se verifica que el **Divisor** efectúa la operación inversa al **Multiplicador**.*

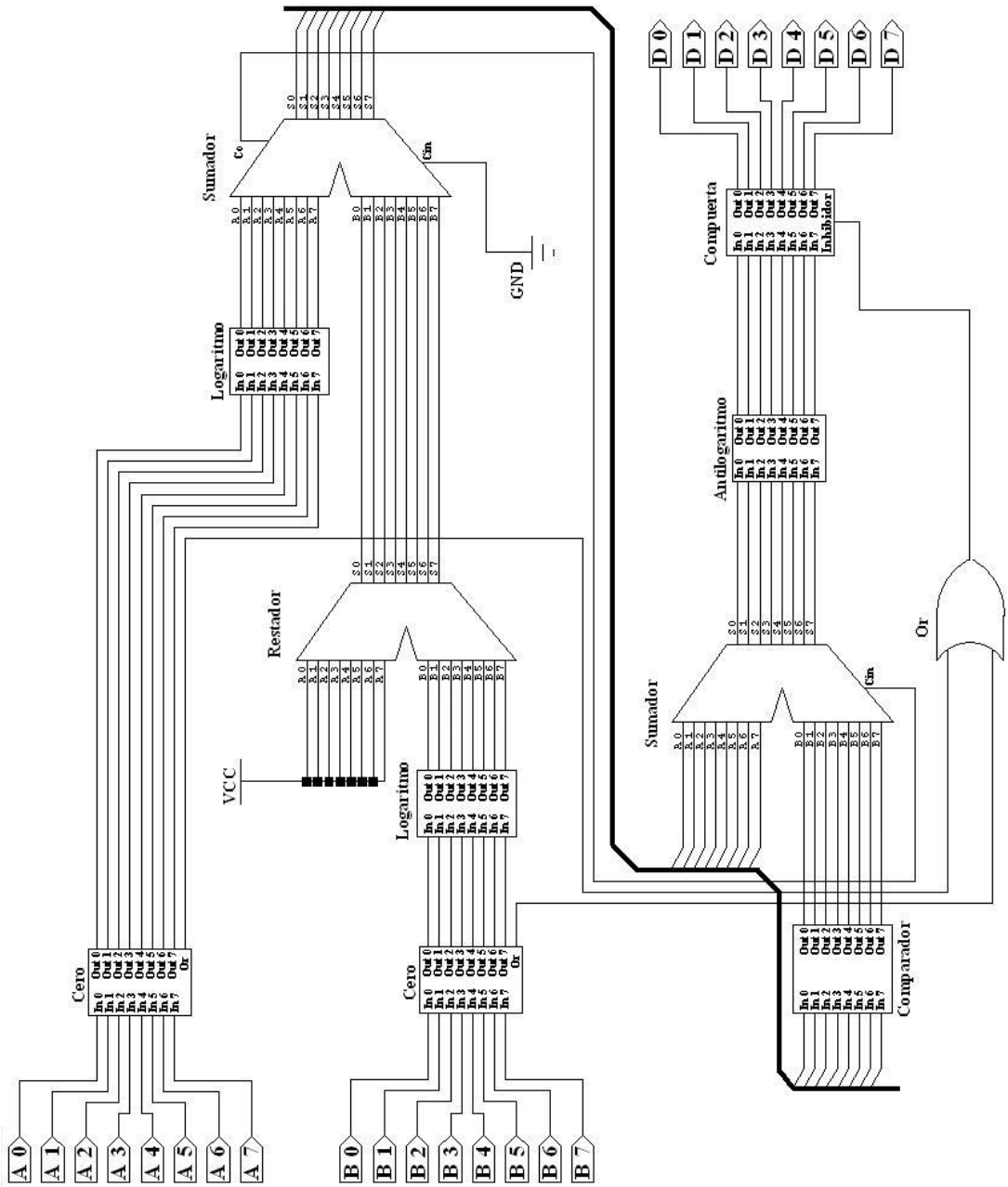


Figura 4.21: *Diagrama interno del componente* ***División****. Está compuesto por los componentes de nivel uno: dos componentes* ***Cero****, que detectan si alguna de las entradas es cero; dos componentes* ***Logaritmo****, que transforman las entradas de la representación vectorial a la representación exponencial; un* ***Restador****, que obtiene el inverso multiplicativo de la entrada* **B***; un* ***Sumador****, que se encarga de efectuar la suma del logaritmo de* **A** *y el inverso multiplicativo del logaritmo de* **B***; un* ***Comparador****, verifica si el resultado de* ***Suma*** *es igual a* 255*, en cuyo caso le enviará un 0000001 al siguiente* ***Sumador*** *para que realice la operación de módulo; un* ***Antilogaritmo****, que cambia el resultado de la representación exponencial a la vectorial; y un* ***Compuerta****, que fuerza la salida a cero si alguno de los datos de entrada vale cero.*

Componente Restador

El componente **Restador** (figura 4.22) realiza la resta de dos datos de ocho bits, presentes en sus entradas $\mathbf{A}[\mathbf{7}:\mathbf{0}]$ y $\mathbf{B}[\mathbf{7}:\mathbf{0}]$, y entrega el resultado en su salida $\mathbf{S}[\mathbf{7}:\mathbf{0}]$.

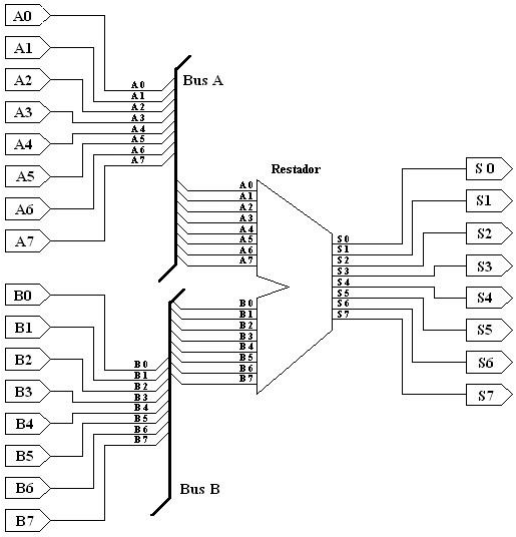


Figura 4.22: *Componente de nivel uno llamado **Restador**, que cuenta con dos entradas (*$\mathbf{A}[\mathbf{7}:\mathbf{0}]$ *y* $\mathbf{B}[\mathbf{7}:\mathbf{0}]$*) y una salida (*$\mathbf{S}[\mathbf{7}:\mathbf{0}]$*), de ocho bits todas ellas. Efectúa la operación* $\mathbf{S} = \mathbf{A} - \mathbf{B}$.

Esta resta es una sencilla operación de aritmética entera, simplemente se resta $\mathbf{A} - \mathbf{B}$. Puesto que este componente se emplea en el cálculo del inverso multiplicativo requerido por la división debe conectarse de una manera específica, explicada en la siguiente sección, lo cual limita el resultado de la resta, por lo que no son necesarias señales de control adicionales.

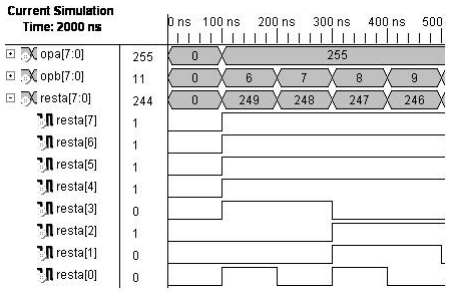


Figura 4.23: *Simulación del componente **Restador**. El operando* $\mathbf{A}$ *tiene asignado como constante el valor* 255*; el operando* $\mathbf{B}$ *es el único que puede variar, con valores que caen en el intervalo de 0 a 255. Por esta razón el componente nunca obtendrá un valor negativo como resultado de la resta.*

Cálculo del Inverso Multiplicativo

Para calcular el inverso multiplicativo de un dato en aritmética de campo finito se requiere encontrar un número tal que, sumado al valor original, se obtenga 255 (ver sección 3.1.3). Por lo tanto, el inverso multiplicativo de un número X puede obtenerse al calcular la resta 255-X. Así, en la entrada A se deben introducir ocho unos (255) y en la entrada B, el logaritmo del dato a convertir, para que de esta manera en S se obtenga el logaritmo del inverso de B. La conexión final se muestra en la figura 4.24.

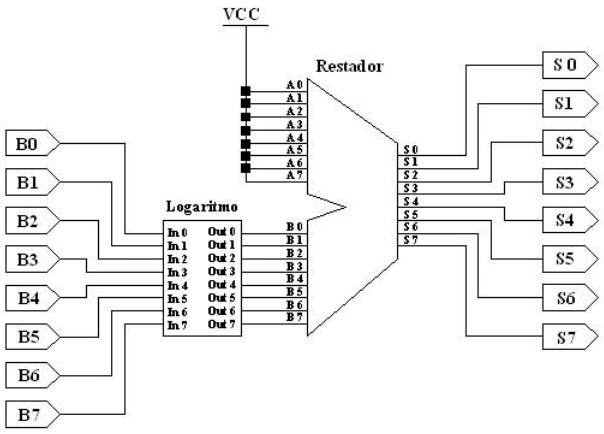


Figura 4.24: *Configuración de componentes que interconectados entre sí realizan la conversión al inverso multiplicativo.*

4.2. Descomposición Wavelet

Las transformadas z de las ecuaciones de diferencias del par de filtros seleccionados para efectuar la transformación wavelet sobre las imágenes se presentan en la fig. 4.25, junto con sus respuestas de magnitud y fase en el dominio de la frecuencia. En la respuesta en magnitud es apreciable que el filtro $\tilde{a}$ es un filtro pasa-altas mientras que el filtro $\tilde{b}$ es un pasa-todo, y ambos tienen fase lineal.

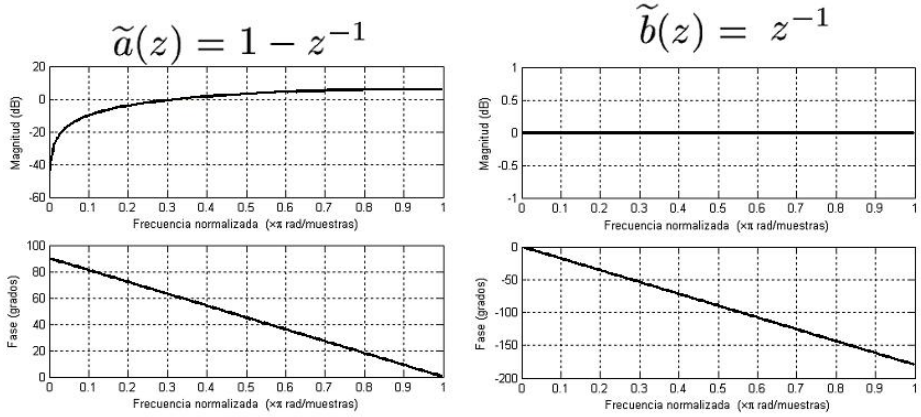


Figura 4.25: *Par de filtros seleccionados para la transformación wavelet. La representación en el plano z del filtro pasa altas ($\tilde{a}$), así como su magnitud y fase en frecuencia, se encuentran en la columna de la izquierda. En la columna derecha se presenta la información del filtro pasa todo $\tilde{b}$. Ambos filtros presentan fase lineal.*

En la figura 4.26 se muestra la representación gráfica de los filtros utilizados en la transformación wavelet.

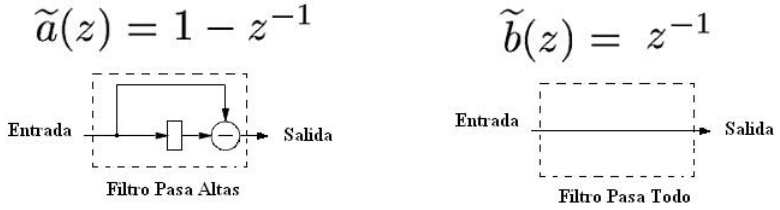


Figura 4.26: *Realización de los filtros utilizados en la transformación wavelet.*

La forma tradicional de describir filtros sobre campos finitos es utilizando máquinas de estados [29]. Sin embargo, si se dispone de la imagen completa almacenada en un arreglo de registros, es posible describir estos filtros de manera combinacional. Por ejemplo, la ecuación del filtro ($\tilde{a}$) efectúa la resta del dato actual menos el anterior, lo cual en la

aritmética del campo finito $F(2^8)$ consiste en una operación or exclusiva entre los operandos [véase secc. 3.1.3]. Así, la descripción combinacional de este filtro se reduce a construir un arreglo de compuertas or exclusiva (fig. 4.27). Es importante resaltar que en el arreglo presentado en la figura ya fue incorporada la decimación, de tal manera que a la salida se obtiene un arreglo de tamaño $\frac{m}{2} + 1$, donde m es el tamaño del arreglo de entrada.

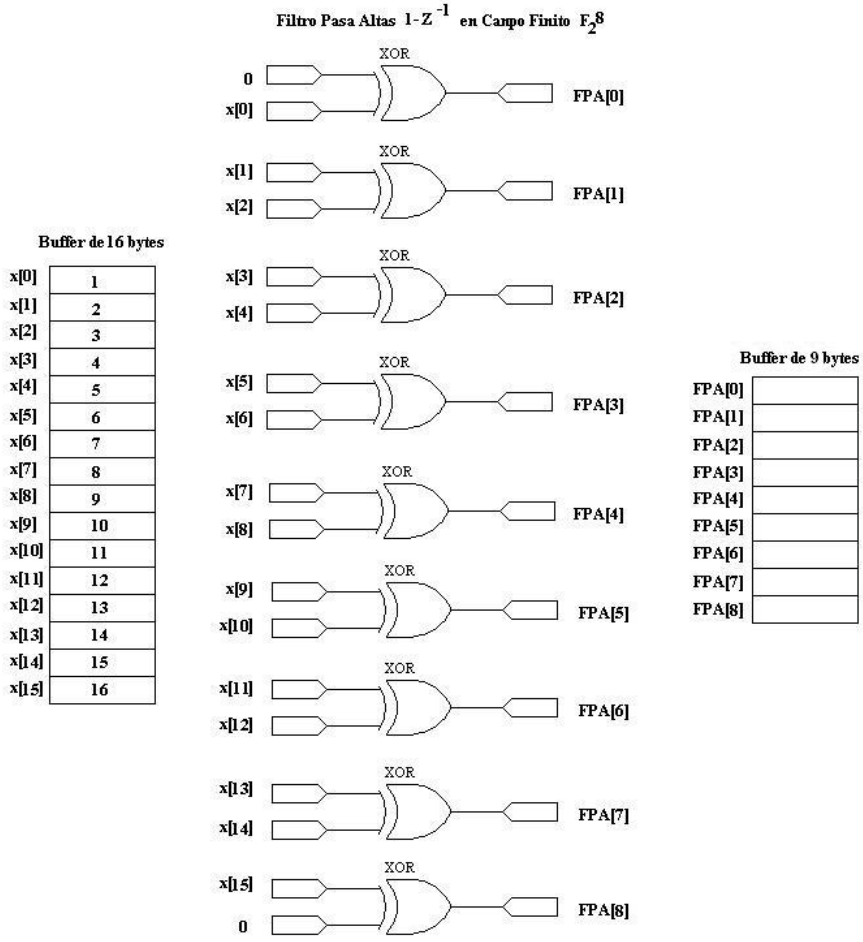


Figura 4.27: *Configuración de componentes nivel uno que interconectados entre sí realizan combinacionalmente el filtrado $\tilde{a} = 1 - z^{-1}$ de cualquier secuencia contenida en el buffer de entrada. Debido a que se utilizó la aritmética del campo finito $F(2^8)$ la resta se logra mediante operaciones or exclusiva entre datos contiguos. La configuración mostrada ya incluye la decimación.*

Dado que el filtro de descomposición $\tilde{b} = z^{-1}$ es un pasa todo, los datos de entrada son tranferidos a la salida sin ninguna alteración, es decir, no se realiza ninguna operación en el campo finito $F(2^8)$. La descripción de este filtro ya con la operacion de decimación se muestra figura 4.28.

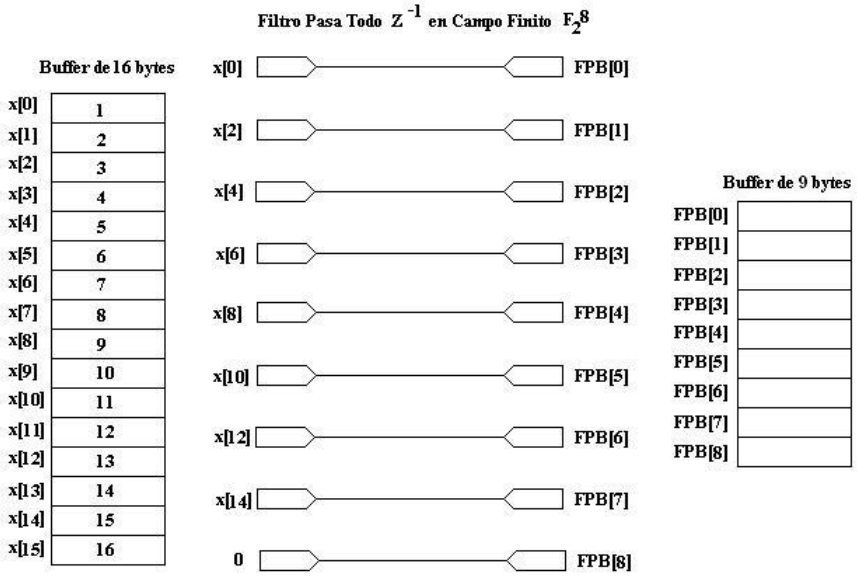


Figura 4.28: *Interconexión de componentes nivel uno que realizan combinacionalmente el filtrado z^{-1} de cualquier secuencia contenida en el buffer de entrada. Dado que éste es un filtro pasa todo no se realizan operaciones con aritmética del campo finito $F(2^8)$.*

4.2.1. Análisis wavelet

La descomposición wavelet más adecuada a nuestro proceso de compresión es la que consiste en filtrar la imagen en cuatro subbandas de tamaño $N/4$ (figura 4.29), siendo N el tamaño original de la señal, ya que de esta forma tendremos que describir un solo componente que llamaremos **Compresión**, el cual se encargará de comprimir cada subbanda.

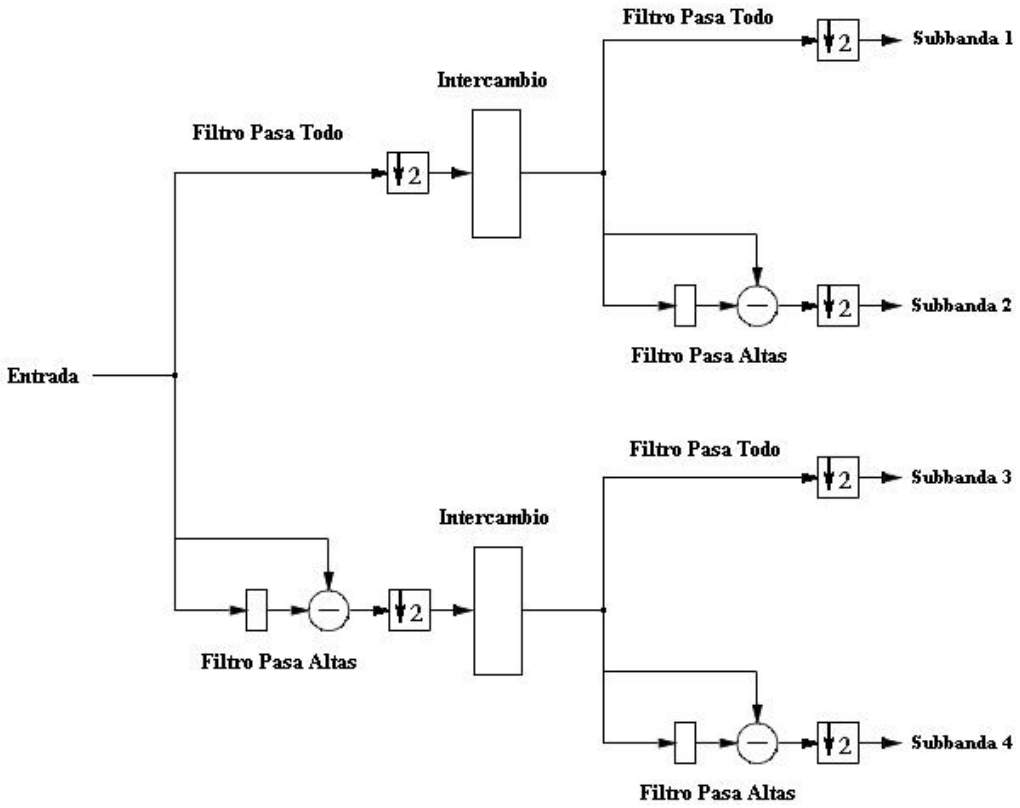


Figura 4.29: *Árbol de descomposición wavelet empleado para el filtrado de la imagen. Cada una de las subbandas obtenidas a la salida tiene el mismo tamaño. Se basa en la concatenación del par de filtros $\tilde{a}$ y $\tilde{b}$. El bloque Intercambio reorganiza los datos de tal forma que la información de los renglones quede acomodada consecutivamente en el arreglo de entrada a la segunda etapa.*

Para ejemplificar el funcionamiento de esta implementación se realizará el filtrado en una hipotética imagen de 16 pixeles. Los datos de la imagen son almacenados en un arreglo de registros, de tal manera que los datos del primer renglón (Figura 4.30) ocupan los primeros cuatro registros, los del segundo renglón ocupan del registro 5 al 8 y así sucesivamente.

El análisis wavelet se realiza por partes. En primer lugar se deben filtrar los datos contenidos por las cuatro columnas. Por la manera como fueron almacenados los datos en el arreglo de registros la información de las columnas se encuentra ordenado en las

posiciones consecutivas, así que sólo hace falta tomar directamente los datos del arreglo y procesarlos mediante los filtros $\tilde{a}$ y $\tilde{b}$ implementados. Los filtros fueron diseñados para que el filtrado y la decimación se efectúen simultáneamente. En la figura 4.30. se muestra la configuración empleada para describir los dos filtros necesarios. Es importante resaltar que por cada filtrado se genera un dato extra, proveniente de la convolución de la señal de entrada con el filtro, lo que se refleja en las salidas de ambos procesos. Estos datos serán eliminados en el proceso de síntesis wavelet.

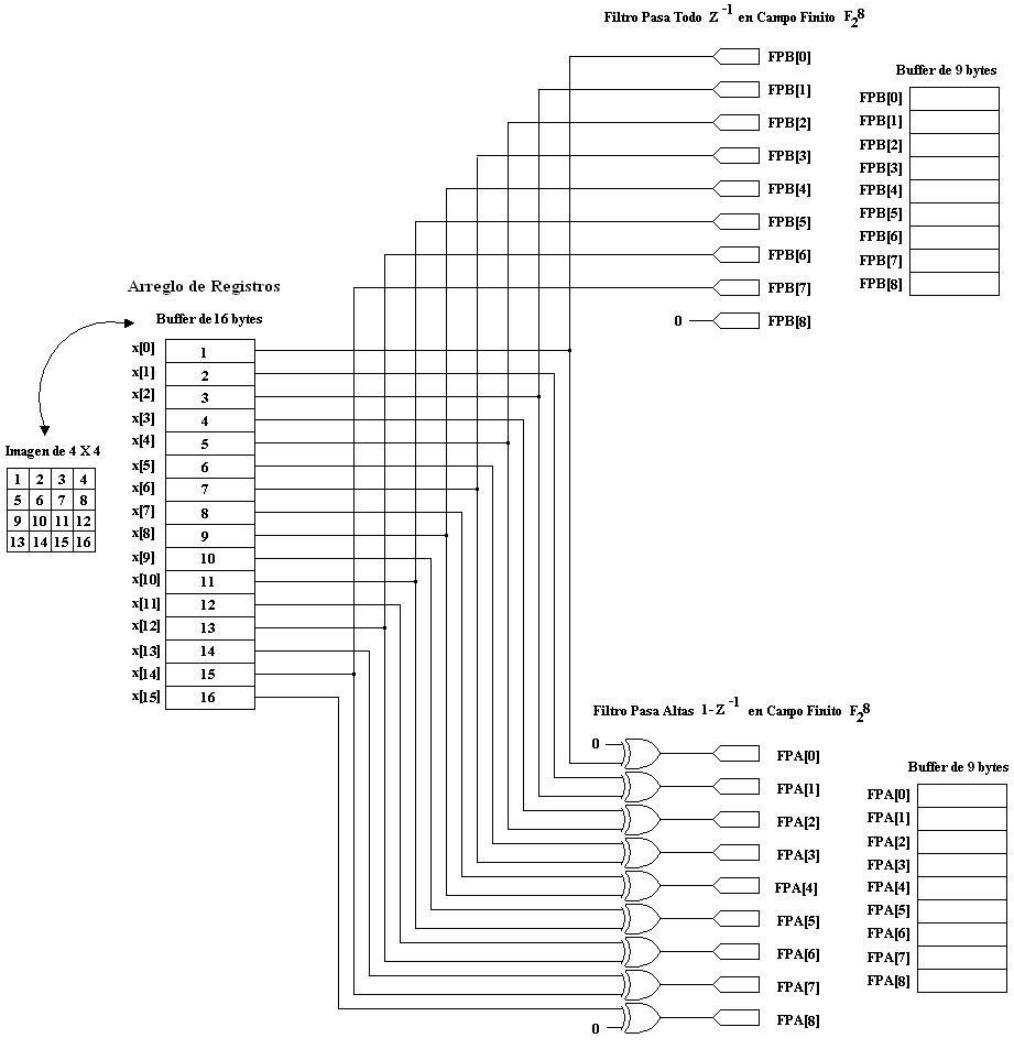


Figura 4.30: *Arreglo de componentes nivel uno que filtra y decima de manera combinacional la señal de entrada. Recibe 16 bytes de entrada y genera dos arreglos de 9 bytes cada uno.*

El funcionamiento de este arreglo de componentes nivel uno se muestra en la simulación (fig. 4.31). Por razones prácticas sólo se muestran los últimos cuatro datos de entrada ($x\_12, ..., x\_15$). Las salidas llamadas **fpb** contienen el resultado de la decimación y filtrado pasa-todo $\tilde{b}$, mientras que la salida del filtrado pasa altas $\tilde{a}$ y su decimación son mostrados en las salidas **fba**.

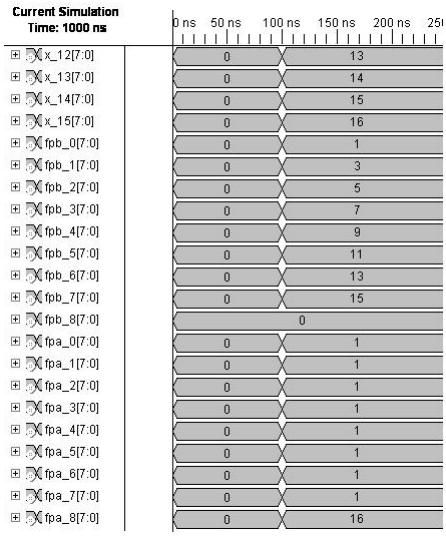


Figura 4.31: *Simulación del arreglo de componentes nivel uno descritos en la figura 4.30. Por razones de espacio en la imagen se omiten los primeros 12 datos de los 16 bytes de entrada llamados x. Las salidas **fpb** representan el resultado del filtrado y decimación $\tilde{b}$ de las columnas y las salidas **fpa** del filtrado y decimado $\tilde{a}$.*

Una vez filtradas las columnas, el siguiente paso debe ser filtrar los renglones de la imagen. Para realizar este filtrado es necesario intercambiar las columnas por renglones; esto se logra reordenando los datos contenidos en los arreglos de salida mediante el componente **Intercambio** (fig. 4.32).

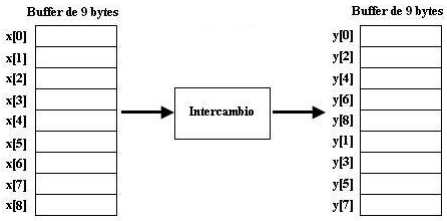


Figura 4.32: *Esquema funcional del componente Intercambio. Reorganiza los datos almacenados en los arreglos **fpb** y **fpa** para que ahora sea el contenido de los renglones el que se encuentra en las posiciones consecutivas.*

Las simulaciones de la operación del componente Intercambio sobre los resultados obtenidos del primer filtrado se muestran en las figuras 4.33 y 4.34.

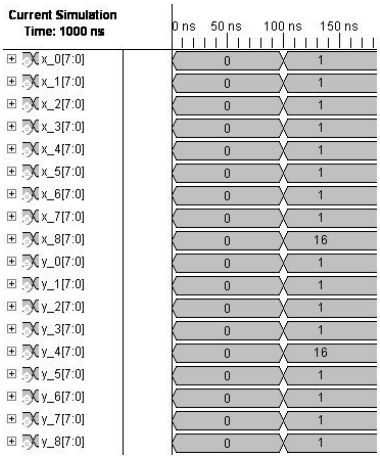


Figura 4.33: *Primera etapa de la simulación del componente* ***intercambio****, en donde la entrada es la subbanda generada por el filtro pasa altas.*

Terminada la fase de intercambio se repite el proceso de filtrado y decimación, para el cual se utiliza la misma estructura de componentes que la etapa de filtrado anterior sobre cada uno de los arreglos reorganizados. En esta ocasión, cada una de las 2 entradas son de 9 bytes y las cuatro salidas son de 5 bytes.

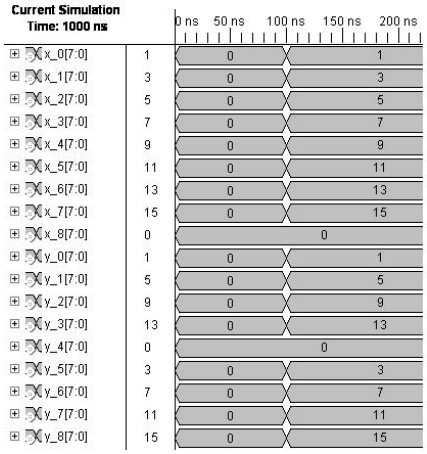


Figura 4.34: *Segunda etapa de la simulación del componente* ***intercambio****, en donde la entrada es la subbanda generada por el filtro pasa todo.*

El resultado de la última etapa de la descomposición wavelet se muestra en la simulación de este componente, la cual se presenta en dos figuras: la primera contiene los

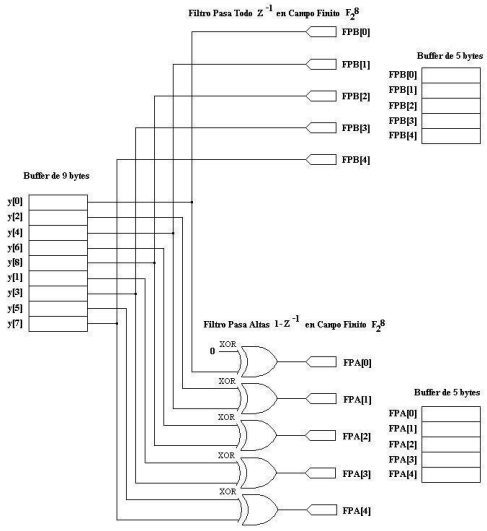


Figura 4.35: *Diagrama de conexión de la etapa dos, en la cual se realiza el filtrado final de la imagen de tamaño 4×4.*

resultados del filtrado cuya entrada (x[8:0]) corresponde a la salida reorganizada del filtro pasa altas (fpa[8:0]) y en la segunda, el banco de filtros recibe como entrada la salida intercambiada del filtro pasa todo (fpb[8:0]).

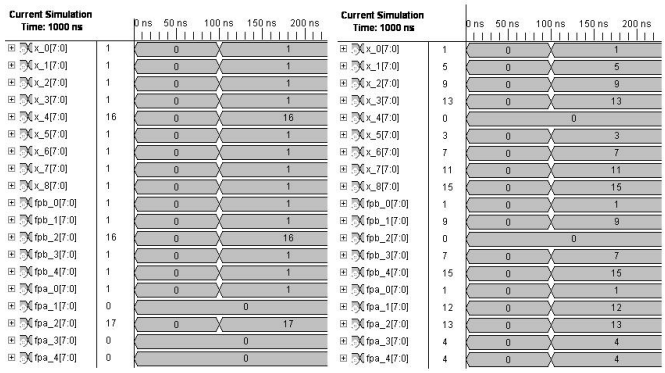


Figura 4.36: *Resultados de la última etapa de filtrado para la descomposición wavelet. Las entradas al banco de filtros corresponden a los datos intercambiados de los arreglos fpa (izquierda) y fpb (derecha).*

En la figura 4.37 se muestra un resumen de la operación de descomposición wavelet sobre los datos utilizados como ejemplo. Al extremo izquierdo se encuentra el buffer de entrada, que contiene los datos de la imagen 4x4 acomodados en una sola columna. Los arreglos de salida contienen los datos reales obtenidos de la aplicación de las dos etapas de filtrado-decimación, incluyendo el intercambio de las columnas por los renglones entre

el primer y segundo filtrado. Cada salida equivale a una subbanda. Sus nombres indican el camino recorrido: las tres primeras letras indican de cuál de los filtros de la primera etapa provienen (FPB equivale a la salida del primer filtro pasa bajas y FPA del pasa altas) y las últimas tres corresponden al filtro de la segunda etapa por el que fueron procesados. Así, la subbanda FPBFPB contiene la salida del segundo filtro pasa bajas que recibió como entrada la salida del primer filtro pasa bajas, FPBFPA es la salida del segundo filtro pasa altas que recibió como entrada la salida del primer filtro pasa bajas y así sucesivamente.

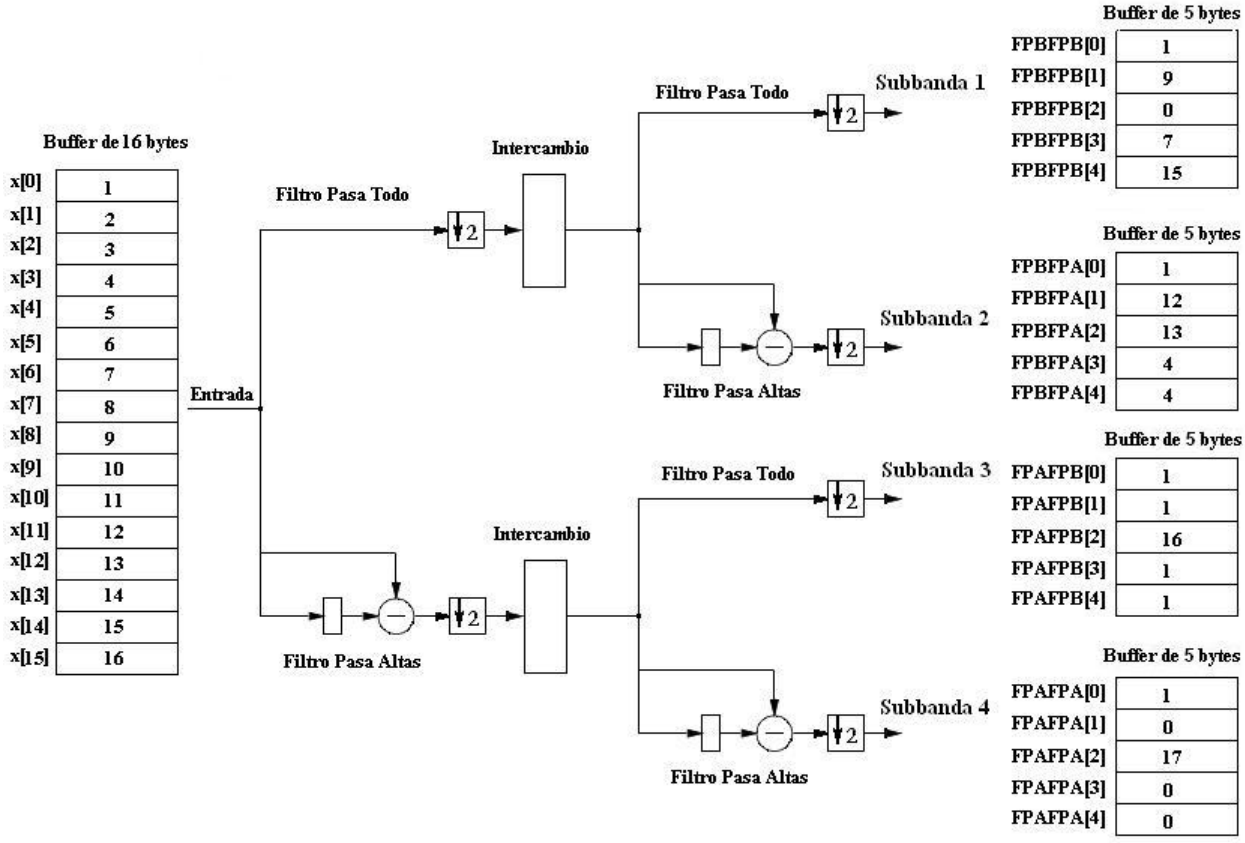


Figura 4.37: *Resultado numérico del análisis wavelet. Los datos del arreglo x[15:0] corresponden a los valores contenidos en la imagen 4x4 tomada como ejemplo. Los vectores de salida (FPBFPB, FPBFPA, FPAFPB y FPAFPA) contienen los datos reales de cada subbanda, que se obtienen al procesar estos valores de entrada con la estructura de filtros, decimadores e intercambiadores implementada.*

4.2.2. Síntesis wavelet

Para recuperar la imagen original a partir de las subbandas obtenidas de su transformada wavelet es necesario efectuar el proceso inverso a la descomposición. La estructura implementada para la reconstrucción wavelet combinacional (fig. 4.38) está formada por casi los mismos componentes que la descomposición y podría parecer como su imagen especular, pero con algunas diferencias: en lugar de decimadores posteriores al filtrado se emplean remuestreadores previos a los filtros los cuales, aunque iguales a los de la descomposición, han sido intercambiados, de manera que el sitio que ocupaba el filtro pasa todo ahora lo ocupa el pasa altas y viceversa. Las salidas de la primera etapa de filtrado se suman entre ellas, son remuestreadas nuevamente, se intercambian sus renglones por columnas y son sometidas a una segunda etapa de filtrado, cuyas salidas sumadas corresponden a la imagen original, recuperada íntegramente, sin pérdida de información.

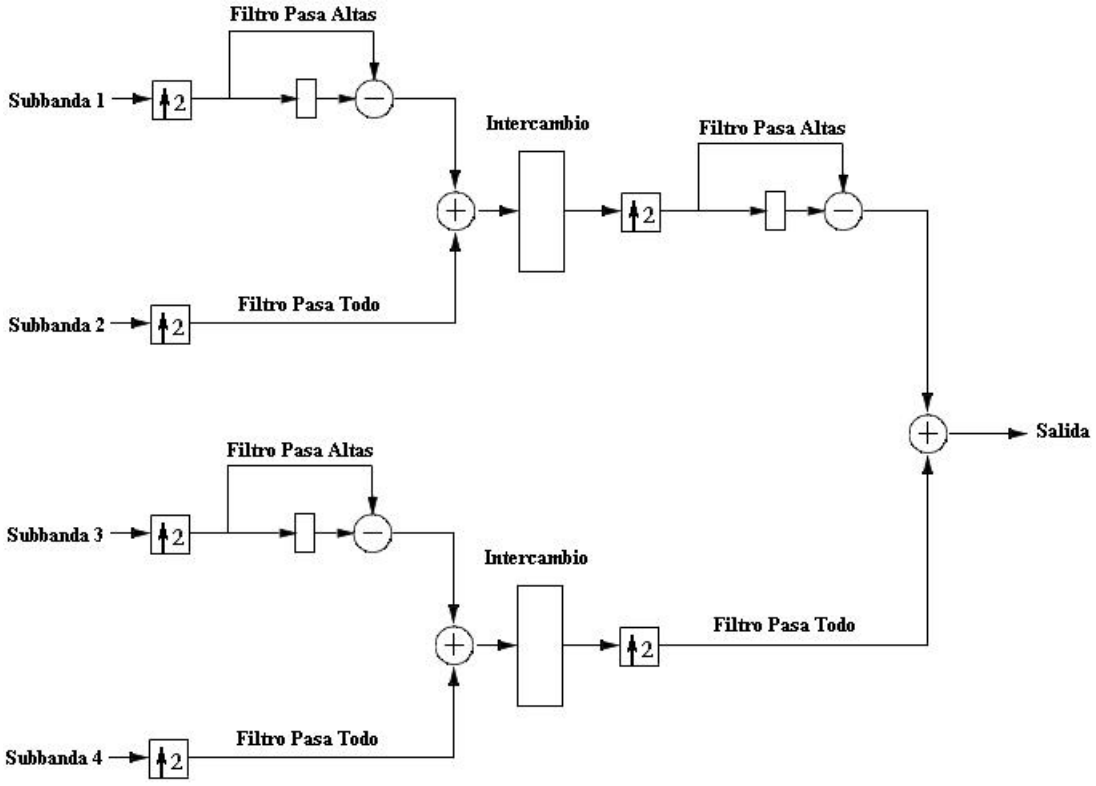


Figura 4.38: *Diagrama a bloques de la Síntesis Wavelet. La estructura consta de casi los mismos elementos que la descomposición, a excepción de que ahora se emplean remuestreadores antes de los filtros en lugar de decimadores. Los filtros son los mismos, pero ocupan sitios intercambiados.*

En la primera etapa de síntesis se reciben como entradas las cuatro subbandas. En primer lugar son remuestreadas, es decir, se intercala un cero entre cada dato, dando como resultado arreglos del doble de longitud de las subbandas originales (fig. 4.39).

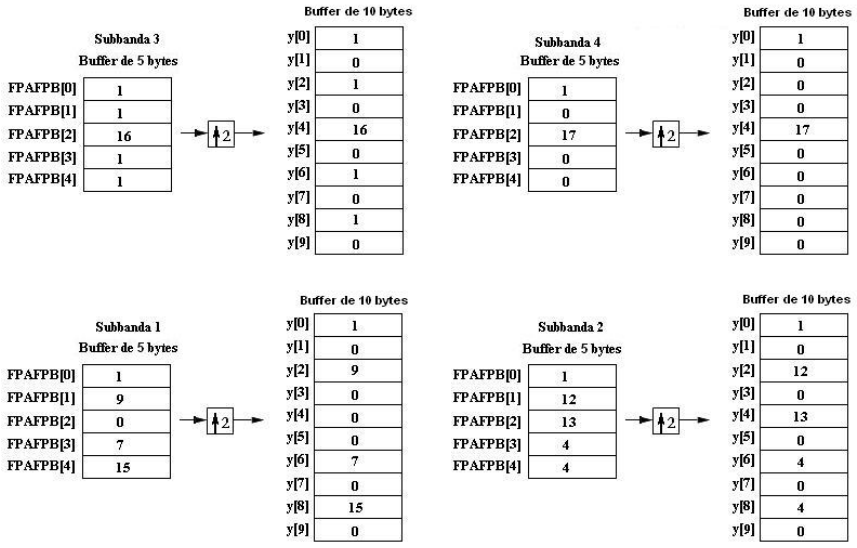


Figura 4.39: *Remuestreo de las cuatro subbandas. Las entradas reciben los valores de una de las subbandas obtenidas del Análisis Wavelet. Como se esperaba de esta operación, los diez bytes de salida contienen los valores de las subbandas con ceros intercalados.*

Para generar el remuestreo se describió un componente con cinco entradas y diez salidas. Está compuesto por diez buffers (figura 4.40), de los cuales cinco están conectados a las entradas y cinco a tierra. La conexión está hecha de tal manera que los dos tipos de buffers se intercalen.

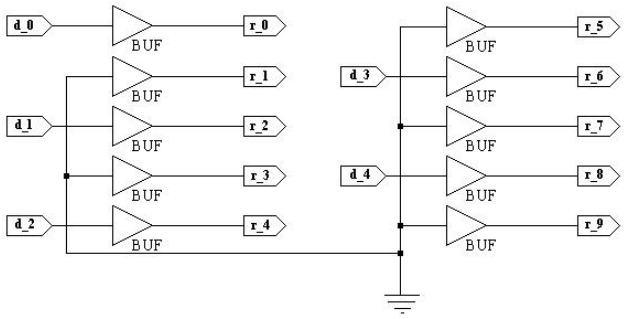


Figura 4.40: *Conexiones internas de la descripción del componente Remuestreo. Los cinco buffers conectados a las entradas se intercalan con los cinco conectados a tierra.*

La simulación de esta descripción se muestra en la figura 4.41

Una vez remuestreadas, las subbandas ingresan a la primera etapa de filtrado de recomposición. Dado que las ubicaciones de los filtros están intercambiadas, los datos de cada subbanda serán procesados por los filtros complementarios a los que los generaron. Así, la subbanda FPBFPB, proveniente de los dos filtros pasa bajas de la descomposición

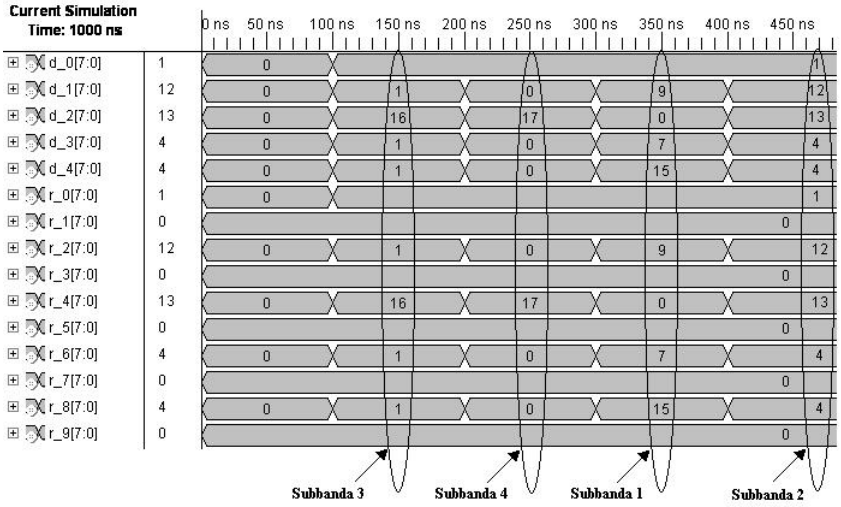


Figura 4.41: *Simulación del remuestreo de las cuatro subbandas. Las entradas reciben los valores de una de las subbandas obtenidas del Análisis Wavelet. Como se esperaba de esta operación, los diez bytes de salida contienen los valores de las subbandas con ceros intercalados.*

será filtrada por los dos filtros pasa altas de la síntesis, mientras que la subbanda FPBFPA pasará primero por el filtro pasa bajas y luego por el pasa altas. Después de atravesar la primera etapa de síntesis, las salidas de los filtros se suman (fig 4.42).

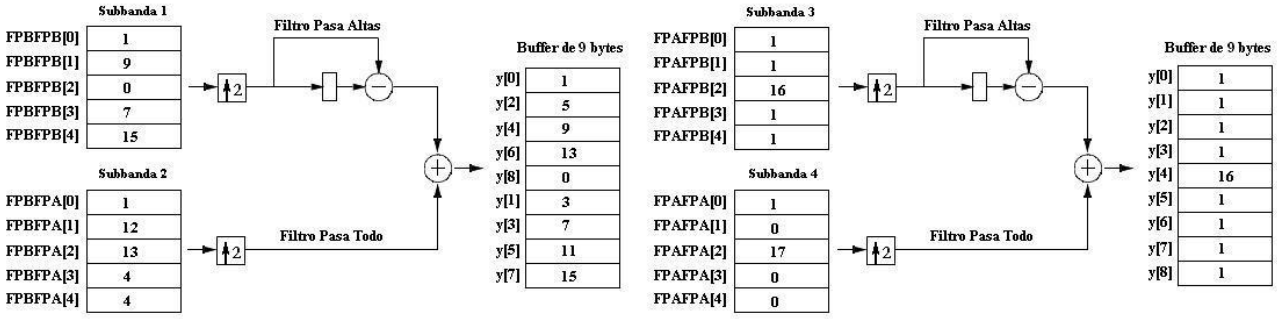


Figura 4.42: *Resultado de la primera etapa de síntesis. En las entradas se reciben las cuatro subbandas de 5 bytes cada una. A la salida se obtienen dos arreglos de 9 bytes, que a su vez servirán de entrada para la siguiente etapa de síntesis.*

La simulación presentada en la figura 4.43 contiene los valores numéricos resultantes de esta primera etapa de reconstrucción. Las dos entradas **ra** y **rb** reciben, en diferentes tiempos, los vectores de las cuatro subbandas remuestreadas. En la salida **y** se encuentra la suma de los resultados de cada filtrado.

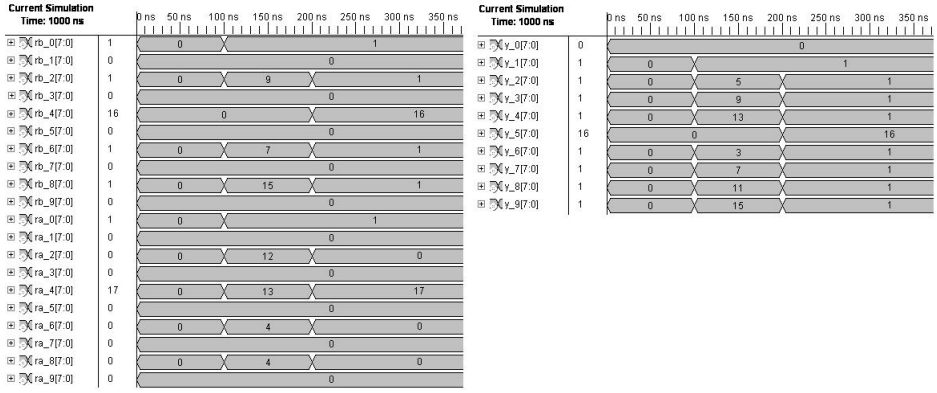


Figura 4.43: *Simulación del resultado de la primera etapa de síntesis. Las entradas **ra** y **rb** reciben las subbandas remuestreadas; la salida **y** entrega la suma de las salidas de los filtros.*

El arreglo resultante de la suma pasa por el componente Intercambio, el cual reorganiza su contenido (figura 4.44).

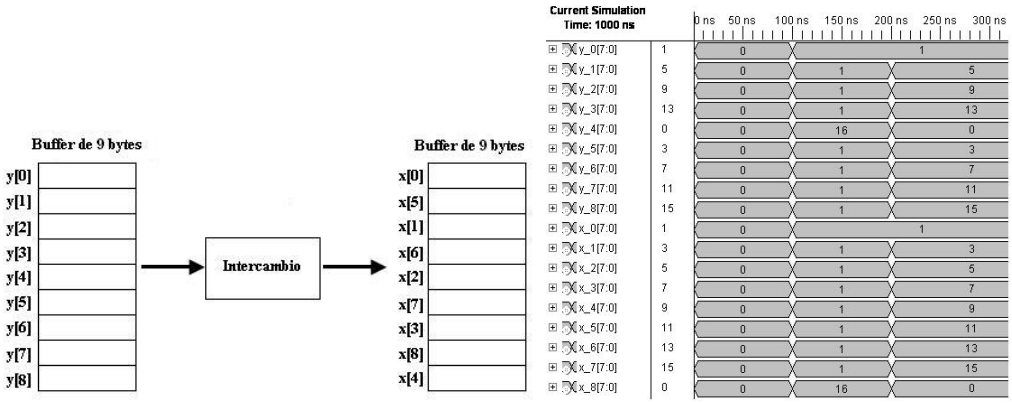


Figura 4.44: *Simulación del funcionamiento del componente Intercambio. La información recibida a la entrada es reorganizada apropiadamente.*

A continuación se efectúa un nuevo remuestreo, con lo cual la longitud de los arreglos se vuelve a duplicar. El siguiente paso es la segunda etapa de filtrado y suma, a la salida de la cual se obtene un arreglo con el tamaño y la información correspondiente a la imagen original.

En la figura 4.45 se encuentra resumido gráficamente todo el proceso de reconstrucción wavelet. Las entradas corresponden a las cuatro subbandas obtenidas de la etapa de descomposición. El arreglo obtenido a la salida es idéntico al arreglo inicial empleado como ejemplo en la descomposición (fig. 4.30).

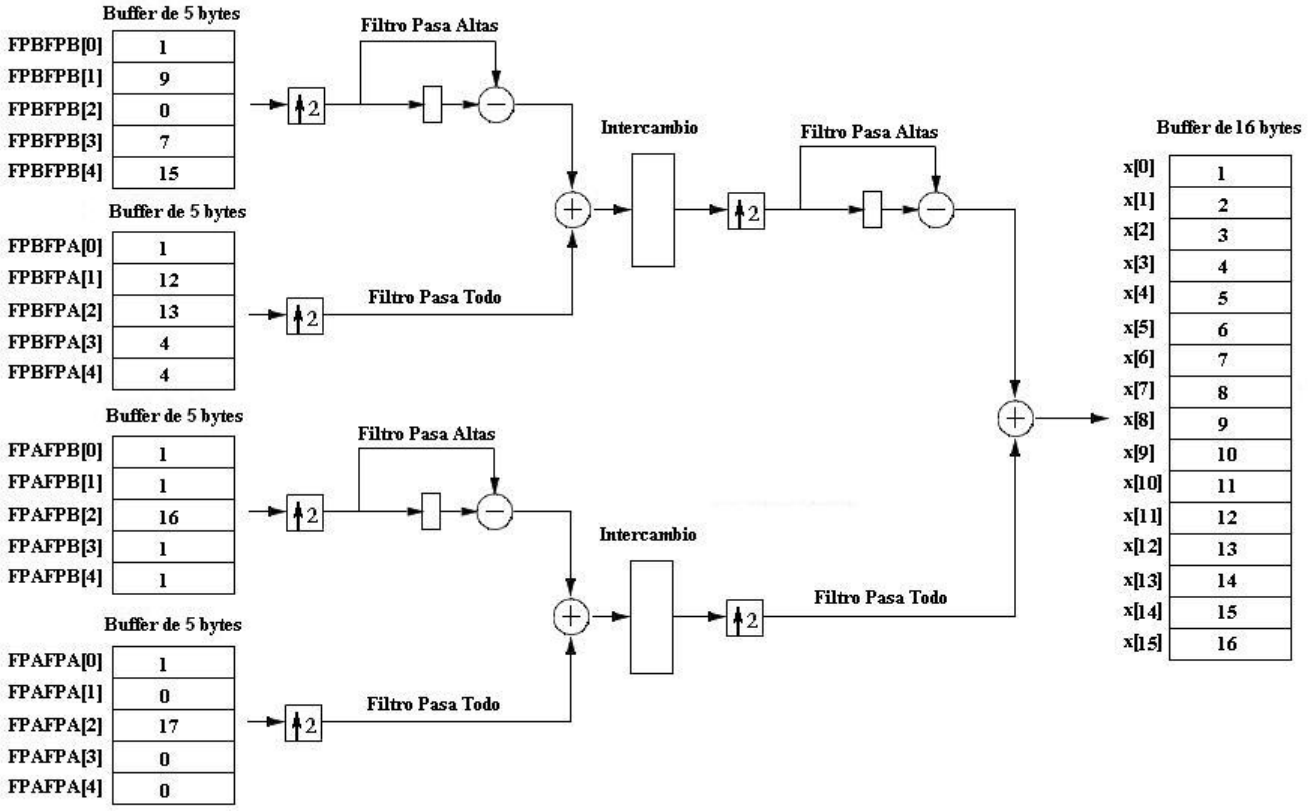


Figura 4.45: *Resultado númerico de la síntesis wavelet completa. Los datos del arreglo de salida corresponden en orden y valor a los datos de la imagen empleada como ejemplo y colocada a la entrada de la descomposición.*

En la figura 4.46 se encuentran las simulaciones del funcionamiento de la segunda etapa de síntesis. A la izquierda encontramos los arreglos de entrada, de 18 bytes cada uno. A la derecha, el arreglo de salida, también de 18 bytes, que contiene los valores de la imagen original, junto con los ceros agregados en la primera etapa de descomposición en los extremos del arreglo.

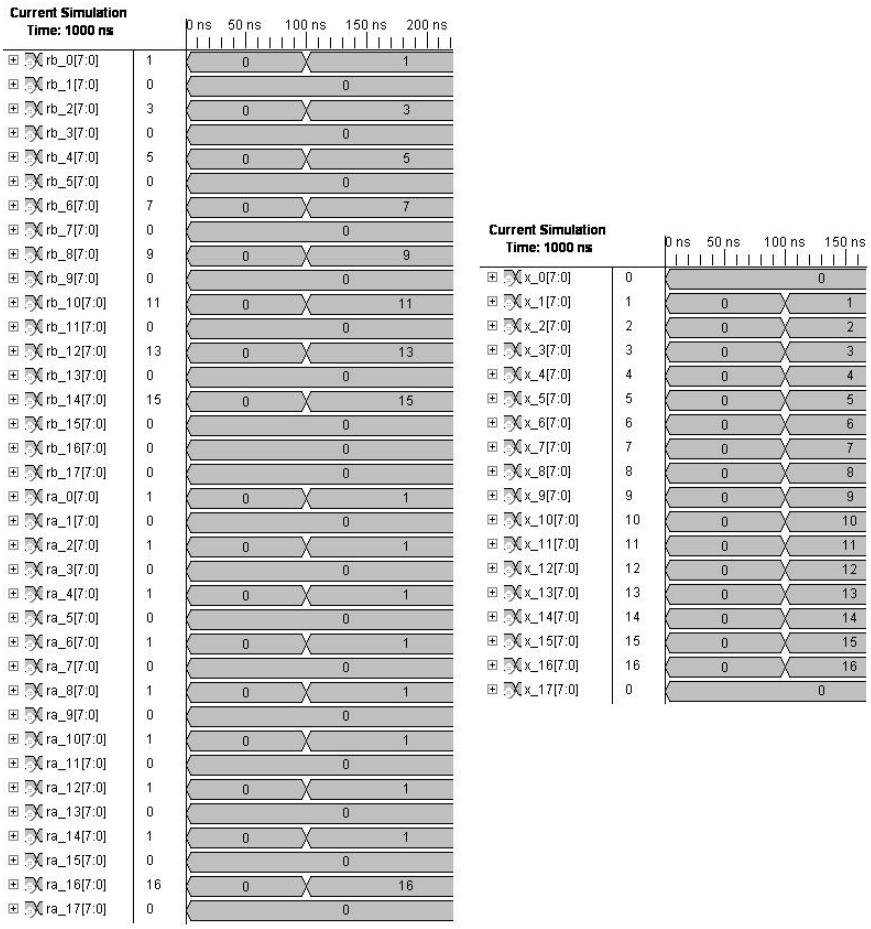


Figura 4.46: *Simulación de la segunda etapa de síntesis. En la imagen de la izquierda se encuentran las entradas, cuyos valores provienen de la etapa de síntesis anterior. A la derecha se muestra la salida obtenida. De inmediato se aprecia que los valores obtenidos corresponden a la imagen original, byte a byte.*

De esta manera corroboramos que tanto la descomposición como la reconstrucción sin pérdida pudieron ser implementadas combinacionalmente con éxito, siempre y cuando se cuente con toda la información de la imagen almacenada en un arreglo desde el principio y se emplee aritmética de campo finito para representar las ecuaciones de los filtros.

Sin embargo, tener toda la información de la imagen almacenada en un arreglo de registros implicaría una gran pérdida de espacio en hardware. Para resolver este problema se optó por seccionar la imagen en arreglos de registros de 16 bytes, lo que no afecta el proceso de filtrado. Desafortunadamente, al hacer el procesamiento de esta forma se pierde gran parte de la velocidad de un proceso combinacional, puesto que no se pueden procesar todos los datos en paralelo.

4.3. Compresión por Codificación Aritmética para números enteros

La naturaleza de este algoritmo es intrínsecamente secuencial. La propuesta para describirlo en VHDL y mejorar considerablemente su tiempo de ejecución consiste en seccionarlo en dos partes, una combinacional, la cual realizará el cálculo del intervalo correspondiente al símbolo en cuestión, y otra parte secuencial, que generará la etiqueta de este símbolo(fig. 4.47). Esta propuesta mejora significativamente el tiempo de procesamiento a costa de un consumo masivo de recursos de hardware.

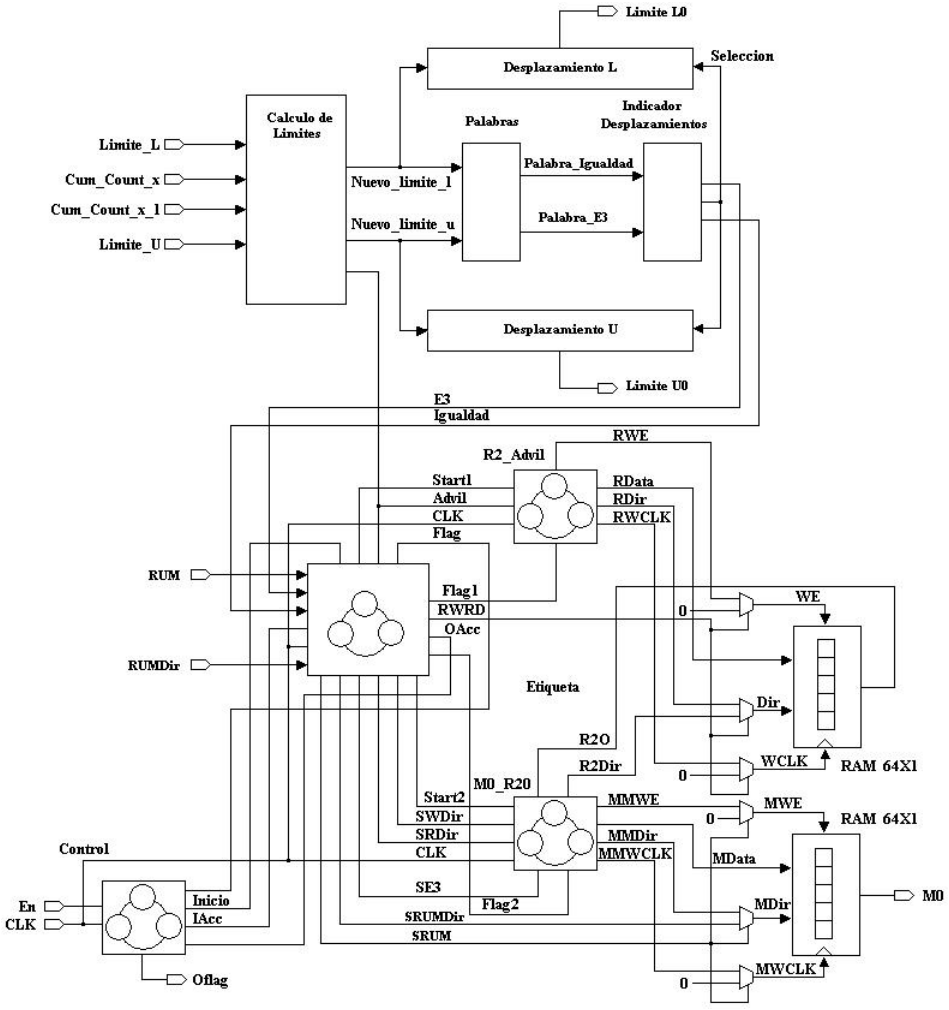


Figura 4.47: *Diagrama de Bloques del Compresor descrito. En la parte superior se encuentra la sección combinacional, encargada de calcular el intervalo de cada símbolo recibido y cuyas salidas se interconectan con la parte secuencial, la cual genera la etiqueta que representa a la secuencia de símbolos completa.*

Para demostrar el correcto funcionamiento del compresor completo, es decir, de la unión de las secciones combinacional y secuencial, se presentan a continuación tres ejemplos de diferentes tipos de convergencia.

Los tres casos que se presentan corresponden a la codificación de un mismo elemento (con frecuencia acumulada de 660). Sin embargo, la longitud de la cadena generada en cada ocasión cambia de manera importante. Esta variabilidad se debe a que el elemento se presenta en diferentes condiciones dentro del proceso de compresión. En el primer caso, es el primer elemento de la cadena, es decir, el intervalo no ha sido acotado aún por ningún otro símbolo. En el segundo caso, el intervalo ya ha sido acotado y los límites previos están muy cercanos uno del otro y casi en el extremo de la convergencia hacia el segmento superior (intervalo **E2**). En el tercer caso también encontramos un intervalo ya acotado, pero en esta ocasión los límites se encuentran en posiciones intermedias (intervalo **E3**).

Estos ejemplos demuestran que la etiqueta generada por cada elemento a codificar es diferente dependiendo de los elementos pasados. En un conjunto de símbolos con alta entropía, las etiquetas generadas serán muy largas. Los valores teóricos para los nuevos límites en el primer caso están dados por:

$$\mathrm{L}_{I1} = 0 + \lfloor \frac{(4095 - 0 + 1) \cdot 549}{1024} \rfloor = 2196 = 100010010100_2$$

$$\mathrm{L}_{U1} = 0 + \lfloor \frac{(4095 - 0 + 1) \cdot 660}{1024} \rfloor - 1 = 2639 = 101001001111_2$$

Al evaluar la representación binaria de estos elementos, observamos que los dos bits más significativos son iguales, cumpliendo primero la intervalo E2 y luego la condición E1. La etiqueta deberá ser entonces 1,0. En la simulación (fig. 4.48) se corrobora que la etiqueta generada es la esperada.

En el caso 2, los nuevos límites calculados son:

$$\mathrm{L}_{I1} = 4000 + \lfloor \frac{(4095 - 4000 + 1) \cdot 549}{1024} \rfloor = 4051 = 111111010011_2$$

$$\mathrm{L}_{U1} = 4000 + \lfloor \frac{(4095 - 4000 + 1) \cdot 660}{1024} \rfloor - 1 = 4060 = 111111011100_2$$

Observamos que los primeros 8 bits de ambos límites son iguales. En consecuencia, la etiqueta esperada será 1,1,1,1,1,1,1,0,1. La etiqueta obtenida en la simulación (fig. 4.49) es correcta.

En el tercer caso los nuevos límites son:

$$\mathrm{L}_{I1} = 1375 + \lfloor \frac{(4095 - 1375 + 1) \cdot 549}{1024} \rfloor = 2833 = 101100010001_2$$

$$\mathrm{L}_{U1} = 1375 + \lfloor \frac{(4095 - 1375 + 1) \cdot 660}{1024} \rfloor - 1 = 3127 = 110000110111_2$$

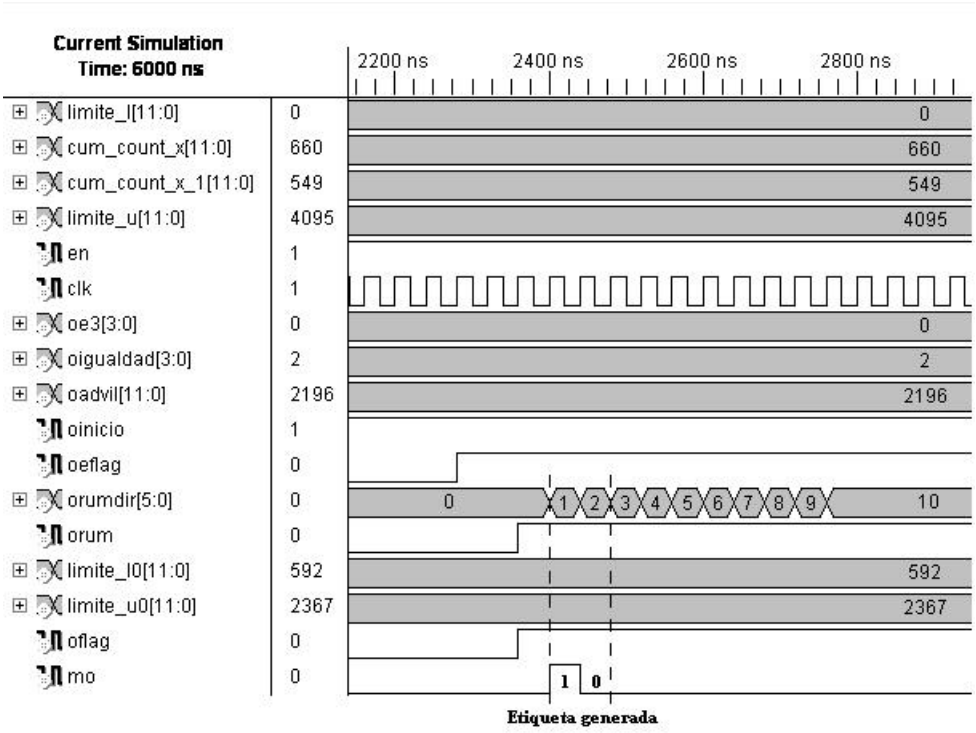


Figura 4.48: *Simulación del primer caso, intervalo no acotado. La etiqueta generada corresponde a la esperada por el cálculo teórico de los nuevos límites.*

Al analizar la representación binaria de ambos límites encontramos que solamente coinciden en el bit más significativo (cumple la intervalo E2) y los dos bits siguientes son diferentes (dos intervalo E3). En consecuencia, la etiqueta generada será de solo un bit (1), pero se almacenará un dos en el contador de intervalo E3, cuyo efecto no es apreciable en la etiqueta correspondiente al elemento actual, pero debe ser considerado en la generación de la etiqueta del elemento siguiente. En la figura 4.50 se puede observar la simulación del comportamiento del compresor ante este caso. La etiqueta resultante es la esperada y se almacenan las dos intervalo E3.

En las siguientes secciones se abordará de manera más explícita el diseño y funcionamiento de cada parte el compresor.

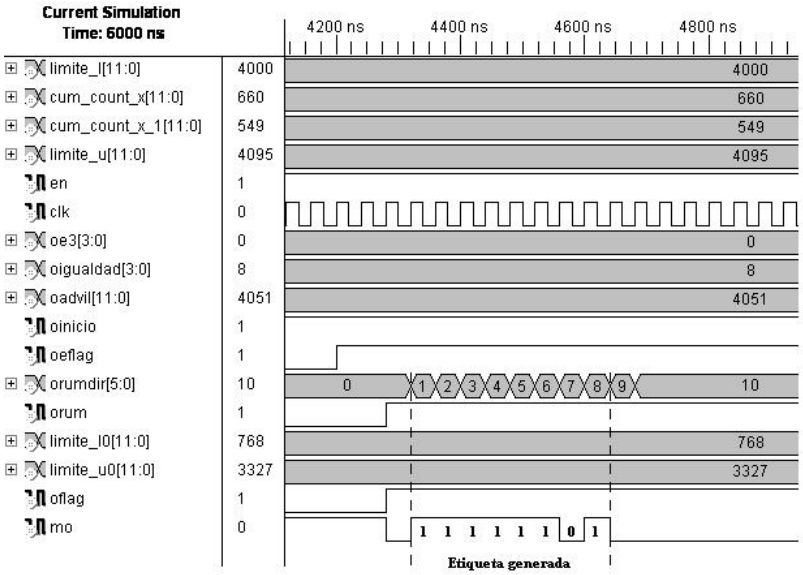


Figura 4.49: *Simulación del segundo caso, límites muy cercanos a un extremo y entre ellos. La etiqueta resultante es larga, ya que ambos límites se parecen mucho entre ellos. Coincide con la etiqueta esperada.*

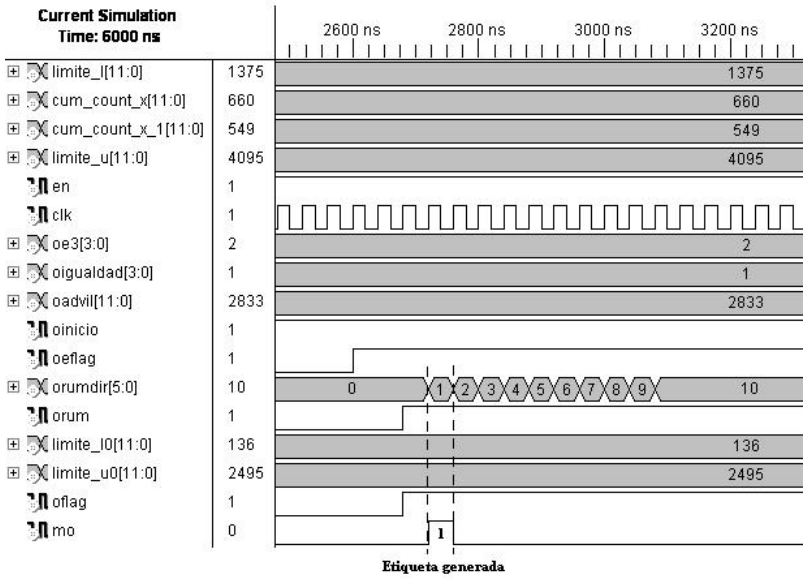


Figura 4.50: *Simulacion del compresor para el tercer caso, intervalo ya acotado pero que abarca tanto la porción inferior como la superior de la recta. La etiqueta generada es la correcta y se almacena el número correcto de condiciones E3 para ser consideradas en el procesamiento del siguiente elemento.*

Componente Cálculo del Intervalo

Este componente tiene cuatro vectores de entrada: **Limite_L, Cum_Count_x, Cum_Count_x_1 y Limite_U**; y cinco vectores de salida:**Limite_L0, Limite_U0, E3, Igualdad y Advil**. Está integrado por cinco componentes combinacionales que en conjunto determinan los límites del intervalo correspondiente al símbolo a codificar, además de generar las señales de control que requiere la parte combinacional para generar la etiqueta correcta.

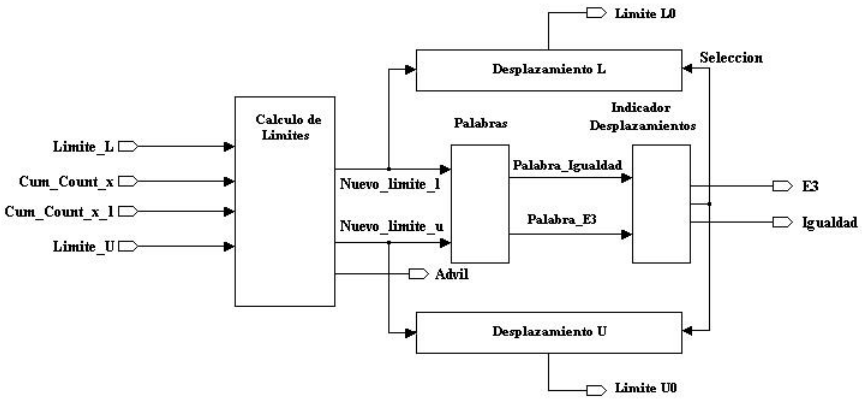


Figura 4.51: *Diagrama interno del componente Cálculo del Intervalo, el cual calcula el intervalo correspondiente al símbolo recibido a la entrada y genera las señales de control necesarias para que la parte secuencial genere la etiqueta apropiada. Al ser descito de forma combinacional, el tiempo de ejecución de este proceso sólo depende del tiempo de propagación del hardware.*

Con el fin de explicar con mayor detalle su funcionamiento, en la figura 4.51 se muestra la interconexión de los bloques que conforman este componente. Al ser descritos combinacionalmente, el tiempo empleado por este proceso está delimitado únicamente por el tiempo de propagación de la señal de entrada.

En la figura 4.52 se muestra la simulación de este componente. En ella se aprecia la evolución de los límites del intervalo y las señales de control para la parte secuencial.

A continuación se describirá la estructura y función de los diversos componentes que integran la parte combinacional Cálculo de Intervalo.

Componente Cálculo de Límite

El componente de nivel dos **Cálculo de Límite** se encarga de recalcular los límites superior e inferior del intervalo de acuerdo a la probabilidad del símbolo a codificar. Esto es efectuado por medio de componentes de niveles inferiores que realizan las operaciones de suma, resta y multiplicación sobre números enteros. La figura 4.53 muestra la secuencia e interconexiones de los componentes que efectúan las operaciones utilizadas.

Las ecuaciones descritas fueron las siguientes:

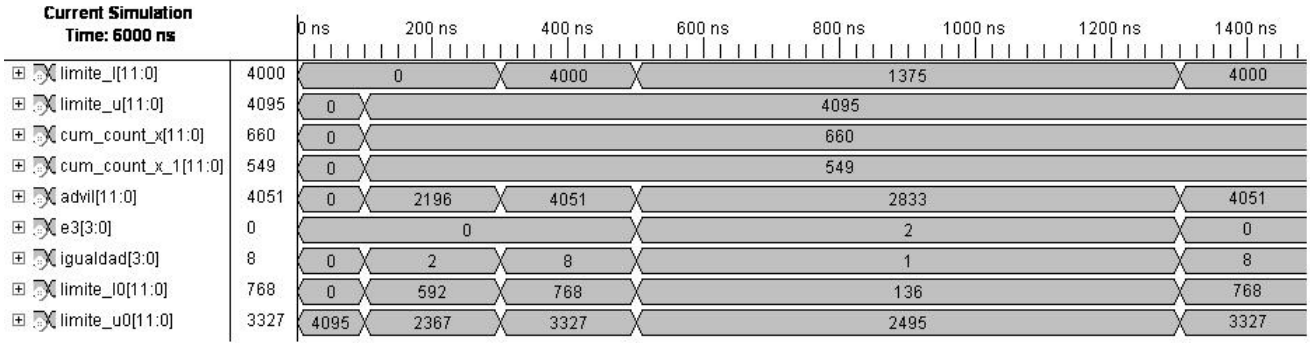


Figura 4.52: *Simulación del funcionamiento del componente Cálculo de Intervalo. Es posible seguir la evolución de las señales de salida, que contienen tanto el valor de los nuevos límites del intervalo como las señales de control para la parte secuencial.*

$$\mathrm{L}_{I(n+1)} = \mathrm{L}_{In} + \lfloor \frac{(\mathrm{L}_{Un} - \mathrm{L}_{In} + 1)Cum\_Count(i-1)}{Total\ \ Count} \rfloor$$

$$\mathrm{L}_{U(n+1)} = \mathrm{L}_{In} + \lfloor \frac{(\mathrm{L}_{Un} - \mathrm{L}_{In} + 1)Cum\_Count(i)}{Total\ \ Count} \rfloor - 1$$

$TotalCount$ es un valor fijo, ya que sólo se procesarán imágenes de $1024\ \ bytes$.

Para la multiplicación se utilizaron los multiplicadores propios del FPGA.

El correcto funcionamiento de este componente se puede verificar en la figura 4.54, la cual presenta la simulación del funcionamiento de este componente. Los valores obtenidos en esta simulación son los esperados, obtenidos al calcular manualmente los límites con las ecuaciones descritas.

Componente Palabras

Una vez calculados los límites del nuevo intervalo se debe verificar si ocurre alguna de las condiciones de convergencia (**E1**, **E2** ó **E3**), labor que realiza el componente **Palabras** de nivel uno[véase sección 3.4]. Este componente está integrado por dos configuraciones combinacionales: la primera, denominada Palabra Igualdad, determina en cuáles bits coinciden los límites superior e inferior mediante compuertas or exclusiva. La segunda, llamada PalabraE3 y constituida de un arreglo de compuertas AND, determina cuántas condiciones **E3** se presentan.

En la figura 4.55 se puede observar la primera de las dos descripciones que conforman a este componente. Esta sección fue descrita de tal forma que resolviera la siguiente operación:

$$PI \ = \ \overline{l \ \oplus \ u} \tag{4.1}$$

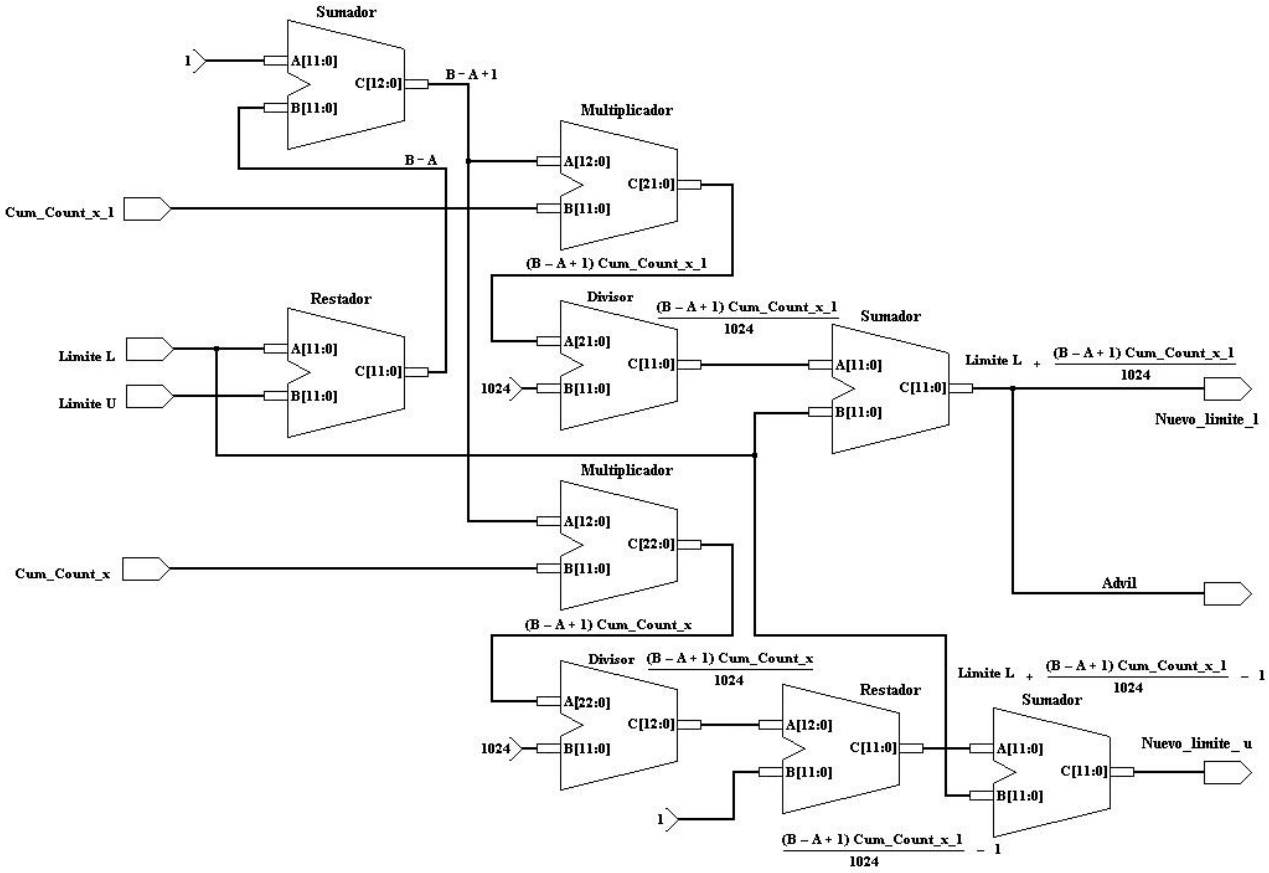


Figura 4.53: *Componente Cálculo de Límite. Consta de tres componentes Sumador, dos componentes Restador, dos Multiplicador y dos Divisor. Estos componentes efectúan de manera combinacional las operaciones aritméticas básicas con operadores enteros. Al interconectarse como se muestra calculan os nuevos límites del intervalo, correspondientes al símbolo a codificar.*

El vector **PI** contiene las coincidencias de los límites. Esta operación indica si los límites convergen, pero no distingue si la intervalo que se presenta es **E1** ó **E2**.

Para determinar si se presenta alguna intervalo **E3** es necesario describir la ecuación:

$$PE = l \cdot \overline{u} \tag{4.2}$$

El resultado **PE** contiene el número de veces que se presenta esta condición. El diagrama de la descripción de esta ecuación se muestra en la figura 4.56.

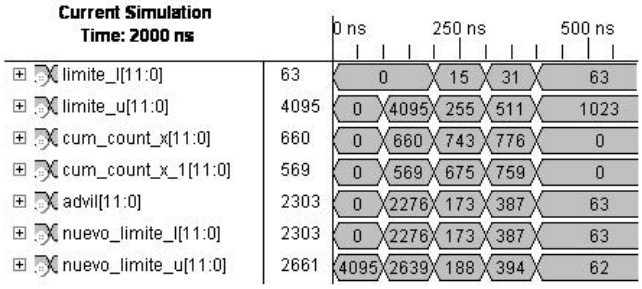


Figura 4.54: *Simulación del componente Cálculo Límite. Los nuevos límites obtenidos a la salida del componente son los esperados por el cálculo manual, indicando así que este componente funciona correctamente.*

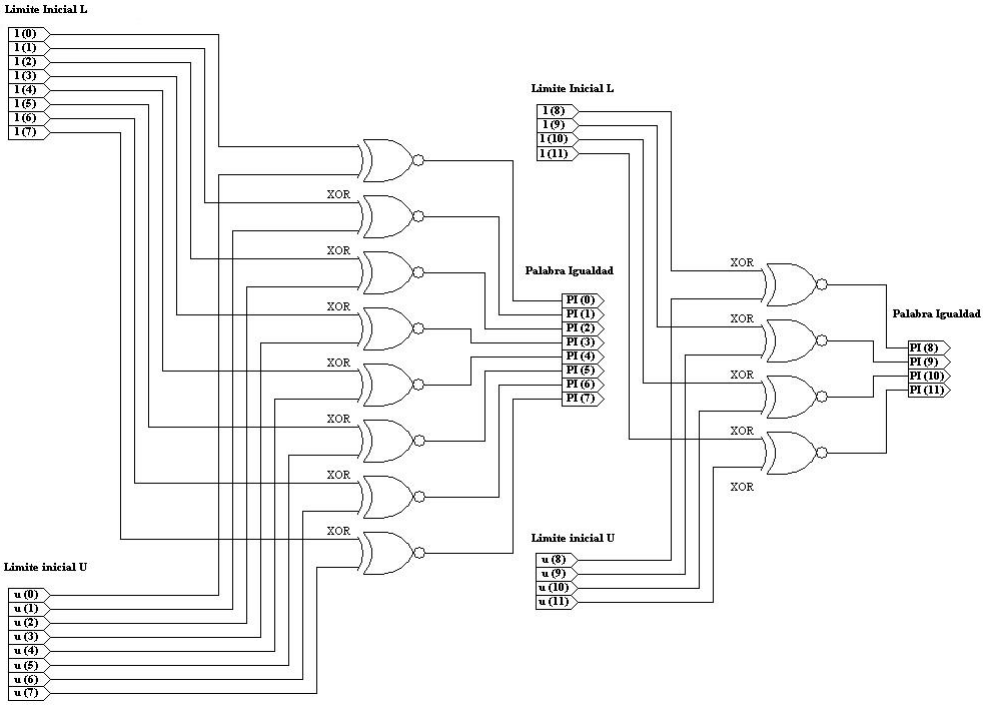


Figura 4.55: *Configuración Palabra Igualdad, formado por un arreglo de compuertas or exclusiva. Recibe los límites del intervalo y determina cuáles de sus bits coinciden.*

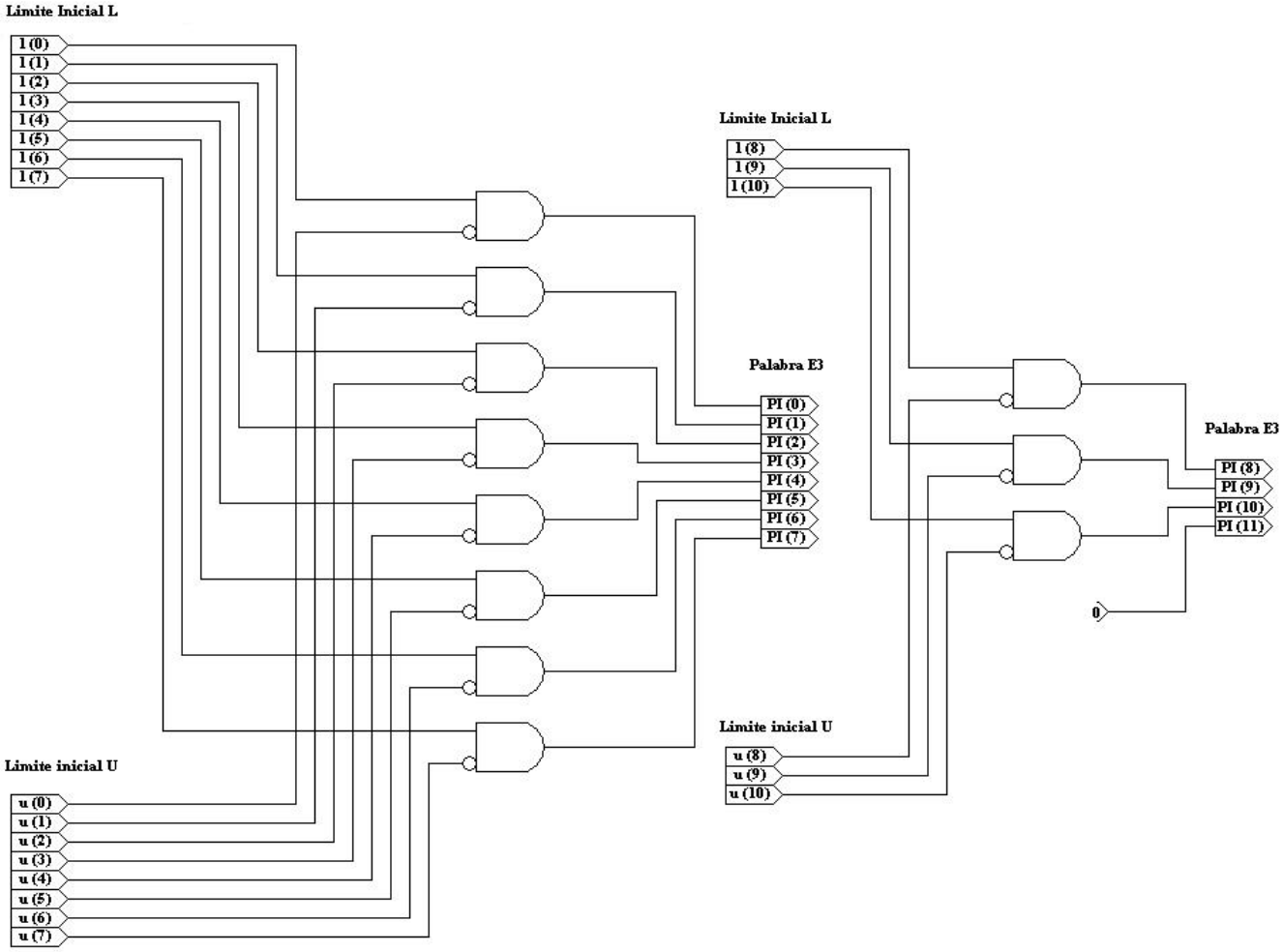


Figura 4.56: *Cofiguración PalabraE3. Recibe los límites del intervalo como entradas y a la salida entrega un vector que contiene el número de ocasiones en que se presentará la condición* ***E3****.*

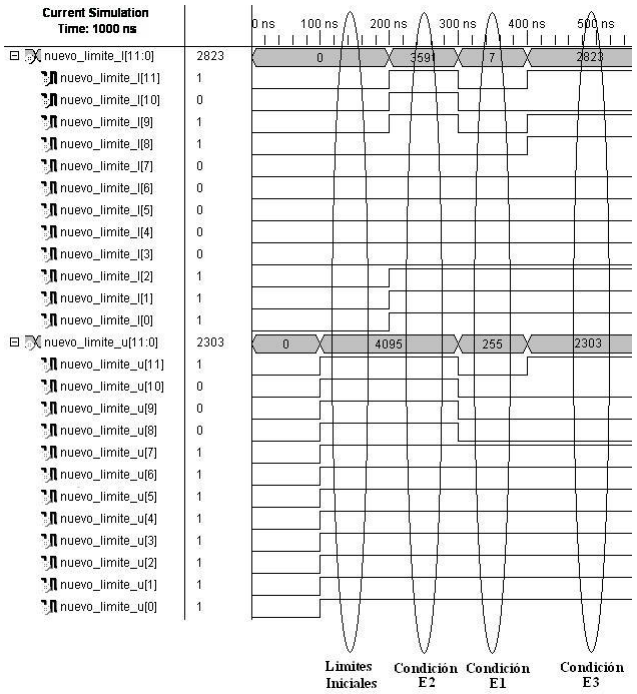


Figura 4.57: *Simulación de entradas del bloque Palabras. Los valores empleados como entradas fueron seleccionados de tal manera que cumplieran las tres condiciones.*

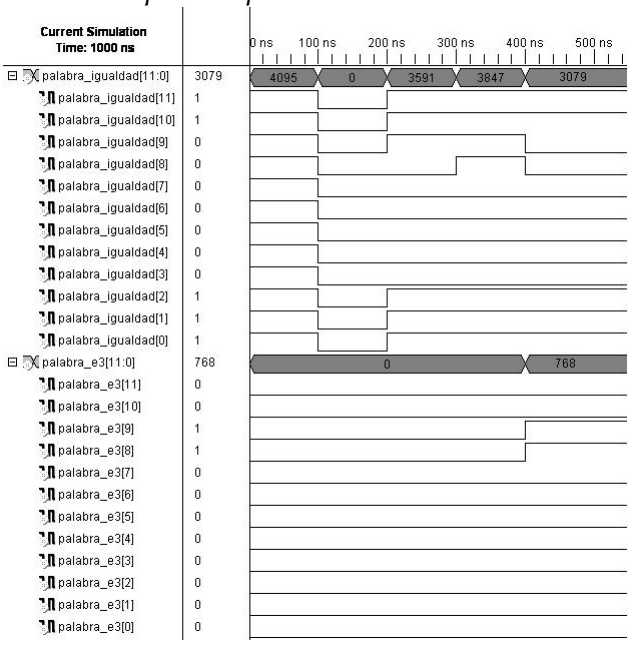


Figura 4.58: *Salida de la simulación del componente **Palabras**. Este componente es capaz de identificar eficazmente cuántas condiciones **E1**,**E2** y **E3** se pueden presentar entre los valores de los límites.*

La simulación que avala el correcto funcionamiento de este componete se muestra en las figuras 4.57 y 4.58 .En la parte superior (fig. 4.57) se emplean como entradas valores de los límites que se sabe que generarán condiciones **E1**,**E2** y **E3**. En la parte baja (fig 4.58) se observa el resultado del procesamiento.

Componente Indicador de Desplazamiento

Este componente es una parte fundamental del algoritmo de codificación aritmética, ya que determina de una forma combinacional el tipo y número de desplazamientos que deben de realizarse sobre los nuevos límites.

Figura 4.59: *Componente **Indicador de Desplazamiento**, es un componente descrito combinacionalmente, basado en una tabla de verdad. Con la ayuda de dos vectores de entrada llamados: Palabra_ Igualdad y Palabra_ E3, genera tres salidas llamadas: E3, Igualdad y Selección, de las cuales las dos primeras pasaran a la segunda parte secuencial. La salida Selección indica cuantos desplazamientos a la izquierda de los limites se deben realizar.*

Es un componente con dos entradas provenientes del componente Palabra, llamadas Palabra_ Igualdad y Palabra_ E3, en las que se basa el cálculo de los desplazamientos; y tres salidas, Seleccion, E3 e Igualdad, de las cuales E3 e Igualdad pasarán como entradas a la etapa secuencial, llamada Generador de Cadena.

La salida Seleccion es un vector que contiene el número de desplazamientos que deben efectuarse en los límites. Para describir esta operación se determinaron todos los posibles casos que generan un desplazamiento. En la figura 4.60, se muestra la simulación de este componente.

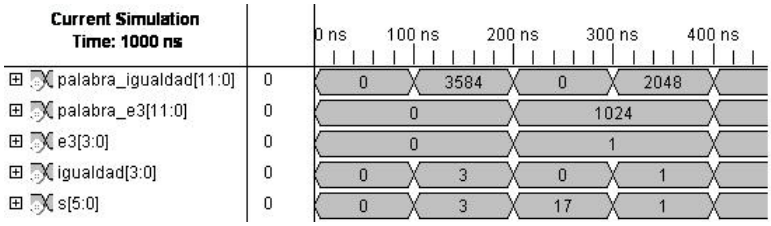


Figura 4.60: *Simulación del componente **Indicador de desplazamiento**. La salida S indica la cantidad de desplazamientos que les deben ser realizados a los limites superior e inferior.*

Componente DesplazamientoL

Este componente es de nivel uno y realiza la expansión del límite inferior en caso de presentarse alguna condición de convergencia hacia el extremo inferior de la recta, indicada por el componente **Indicador de Desplazamiento**.

Está compuesto por dos entradas : Nuevo_ limite_ l y Seleccion; y una salida, Limite_ l0 (fig. 4.61).

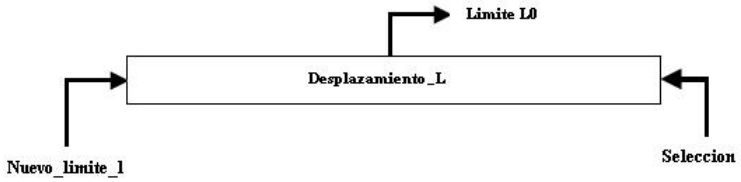


Figura 4.61: *Diagrama de bloque del componente* ***DesplazamientoL*** *de nivel uno.*

El vector Nuevo_ limite_ l será desplazado hacia la derecha el número de veces indicado por Seleccion. Los espacios liberados por los desplazamientos son llenados con ceros. El resultado de este deplazamiento será almacenado en el vector Limite_ l0.

La descripción de esta operación se basó en una tabla de verdad, de tal forma que con solo dos vectores de entrada (I, valor del límite antes del desplazamiento y S, señal de control que indica cuántos desplazamientos deberán efectuarse y si hay alguna condición E3) se pueda obtener el vector de salida (O, valor del límite ya desplazado y, en su caso, con el bit más significativo invertido). En caso de que Seleccion (S) indique la condición **E3** (parte baja de la tabla de verdad Cuadro 4.4), el límite será desplazado y el bit más significativo será invertido.

La simulación de este componente se encuentra en la figura 4.62, en donde se puede verificar su correcto funcionamiento, ya que el vector de salida O contiene los valores del vector de entrada I desplazado el tipo y número de veces indicado por el vector S.

Componente DesplazamientoU

Tiene un funcionamiento parecido al bloque anterior y es un componente de nivel uno (fig. 4.63) , está basado en una tabla de verdad (Cuadro 4.5)de manera que, con solo dos vectores de entrada, Limite Inicial l[11:0] y selección[5:0], podemos obtener en la salida LimiteU0 el vector de entrada Limite Inicial U[11:0] desplazado hacia la izquierda el número de veces requerido y, cuando sea necesario, desplazarlo e invertir el bit más significativo. En esta ocasión los espacios liberados son llenados con unos.

La figura 4.64 muestra el resultado del funcionamiento del componente DesplazamientoU.

Entradas																		Salidas											
$I11$	$I10$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$		$S5$	$S4$	$S3$	$S2$	$S1$	$S0$	$O11$	$O10$	$O9$	$O8$	$O7$	$O6$	$O5$	$O4$	$O3$	$O2$	$O1$	$O0$
$I11$	$I10$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$		0	0	0	0	0	0	$I11$	$I10$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$
$I11$	$I10$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$		0	0	0	0	0	1	$I10$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$	0
$I11$	$I10$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$		0	0	0	0	1	0	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$	0	0
$I11$	$I10$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$		0	0	0	0	1	1	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$	0	0	0
$I11$	$I10$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$		0	0	0	1	0	0	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$	0	0	0	0
$I11$	$I10$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$		0	0	0	1	0	1	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$	0	0	0	0	0
			$\vdots$												$\vdots$							$\vdots$								
$I11$	$I10$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$		1	0	0	0	0	1	$\overline{I10}$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$	0
$I11$	$I10$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$		1	0	0	0	1	0	$\overline{I9}$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$	0	0
$I11$	$I10$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$		1	0	0	0	1	1	$\overline{I8}$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$	0	0	0
$I11$	$I10$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$		1	0	0	1	0	0	$\overline{I7}$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$	0	0	0	0
$I11$	$I10$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$		1	0	0	1	0	1	$\overline{I6}$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$	0	0	0	0	0

Cuadro 4.4: *Tabla de verdad implementada en la descripción del componente DesplazamientoL. El vector I contiene el valor inicial del límite, S indica el número de desplazamientos necesarios y si ha ocurrido una condición E3, y O contiene el valor del límite ya desplazado.*

Entradas																		Salidas											
$I11$	$I10$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$		$S5$	$S4$	$S3$	$S2$	$S1$	$S0$	$O11$	$O10$	$O9$	$O8$	$O7$	$O6$	$O5$	$O4$	$O3$	$O2$	$O1$	$O0$
$I11$	$I10$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$		0	0	0	0	0	0	$I11$	$I10$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$
$I11$	$I10$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$		0	0	0	0	0	1	$I10$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$	1
$I11$	$I10$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$		0	0	0	0	1	0	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$	1	1
$I11$	$I10$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$		0	0	0	0	1	1	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$	1	1	1
$I11$	$I10$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$		0	0	0	1	0	0	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$	1	1	1	1
$I11$	$I10$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$		0	0	0	1	0	1	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$	1	1	1	1	1
			$\vdots$												$\vdots$							$\vdots$								
$I11$	$I10$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$		1	0	0	0	0	1	$\overline{I10}$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$	1
$I11$	$I10$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$		1	0	0	0	1	0	$\overline{I9}$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$	1	1
$I11$	$I10$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$		1	0	0	0	1	1	$\overline{I8}$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$	1	1	1
$I11$	$I10$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$		1	0	0	1	0	0	$\overline{I7}$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$	1	1	1	1
$I11$	$I10$	$I9$	$I8$	$I7$	$I6$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$		1	0	0	1	0	1	$\overline{I6}$	$I5$	$I4$	$I3$	$I2$	$I1$	$I0$	1	1	1	1	1

Cuadro 4.5: *Tabla de verdad del componente DesplazamientoU*

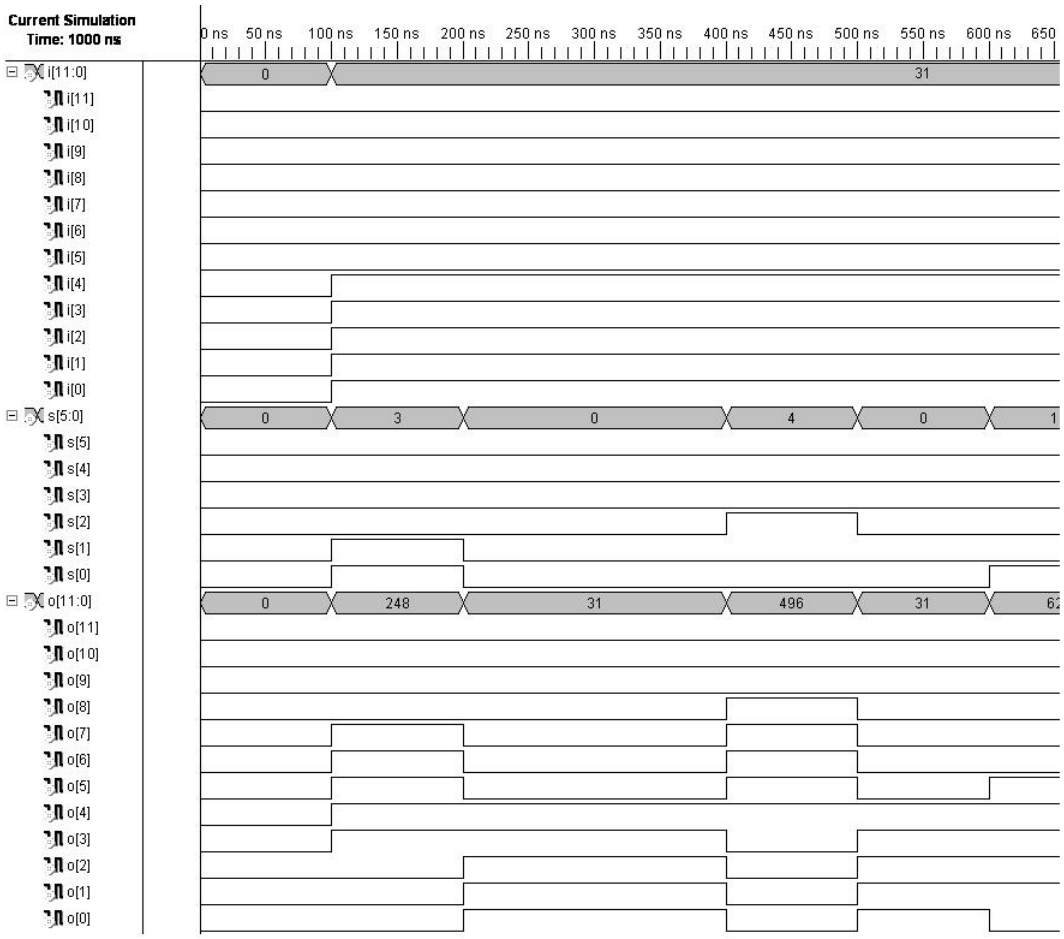


Figura 4.62: *Simulación del funcionamiento del componente* ***DesplazamientoL****. Es sencillo verificar que el vector de salida* ***o*** *contiene los valores del vector de entrada* ***i*** *desplazados el número de veces que indica el vector de control* ***s****.*

Figura 4.63: *Diagrama a bloque del componente* ***DesplazamientoU*** *de nivel uno.*

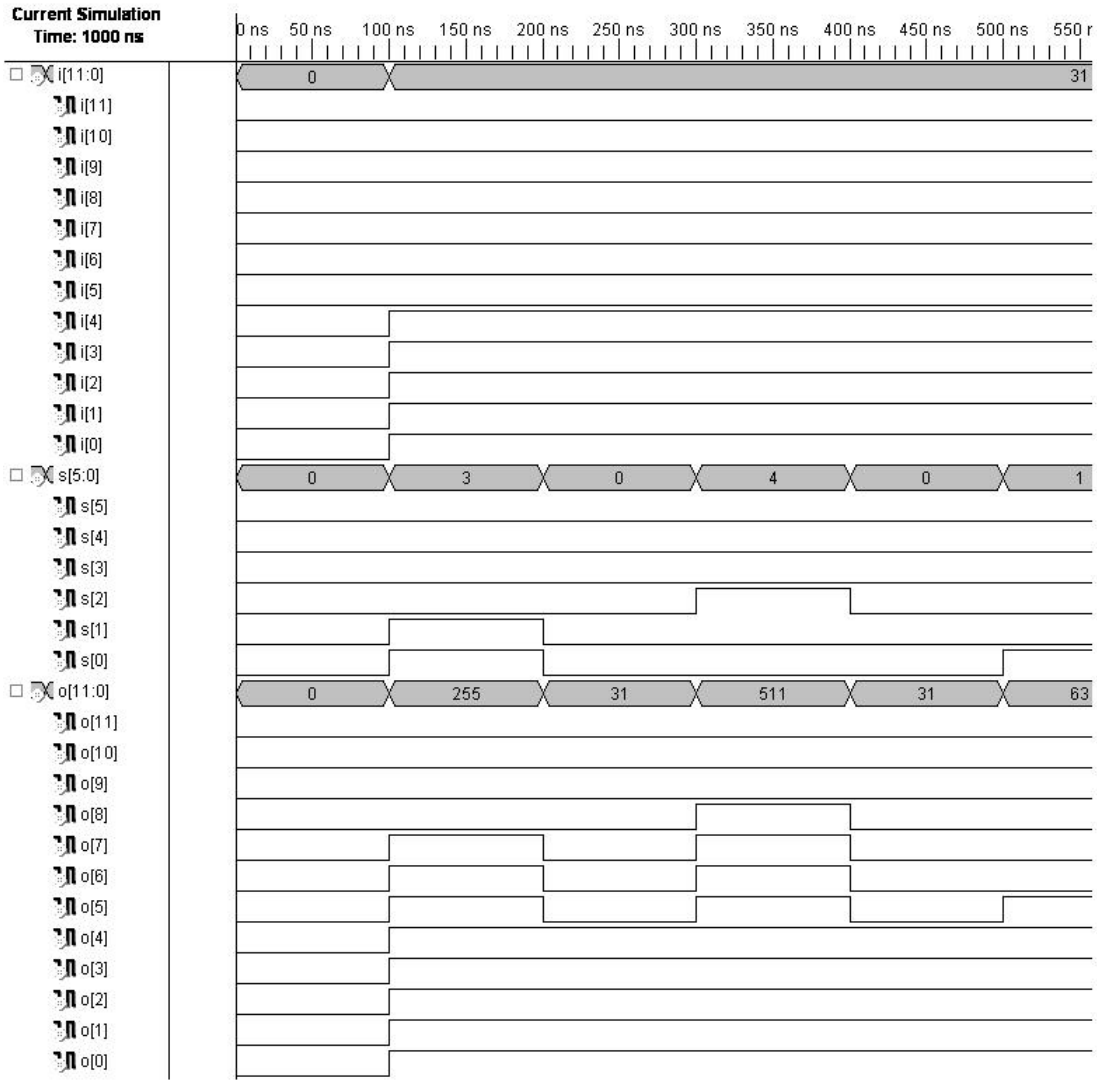


Figura 4.64: *Funcionamiento del componente* ***DesplazamientoU****. El vector de salida o contiene los valores del vector de entrada i desplazados el número de veces que indica el vector de control s. Es de resaltar que en cada desplazamiento se agregan unos en la parte baja del vector de salida, a diferencia del componente* ***DesplazamientoL*** *en donde se agregan ceros.*

Parte Secuencial generadora de la Etiqueta

La parte secuencial es la encargada de generar la etiqueta resultante del proceso de compresión. Recibe tres vectores de entrada, provenientes de la parte combinacional, llamados Advil, E3 e Igual. También posee tres entradas de control (figura 4.65). La etiqueta generada es almacenada en una memoria RAM 64X1.

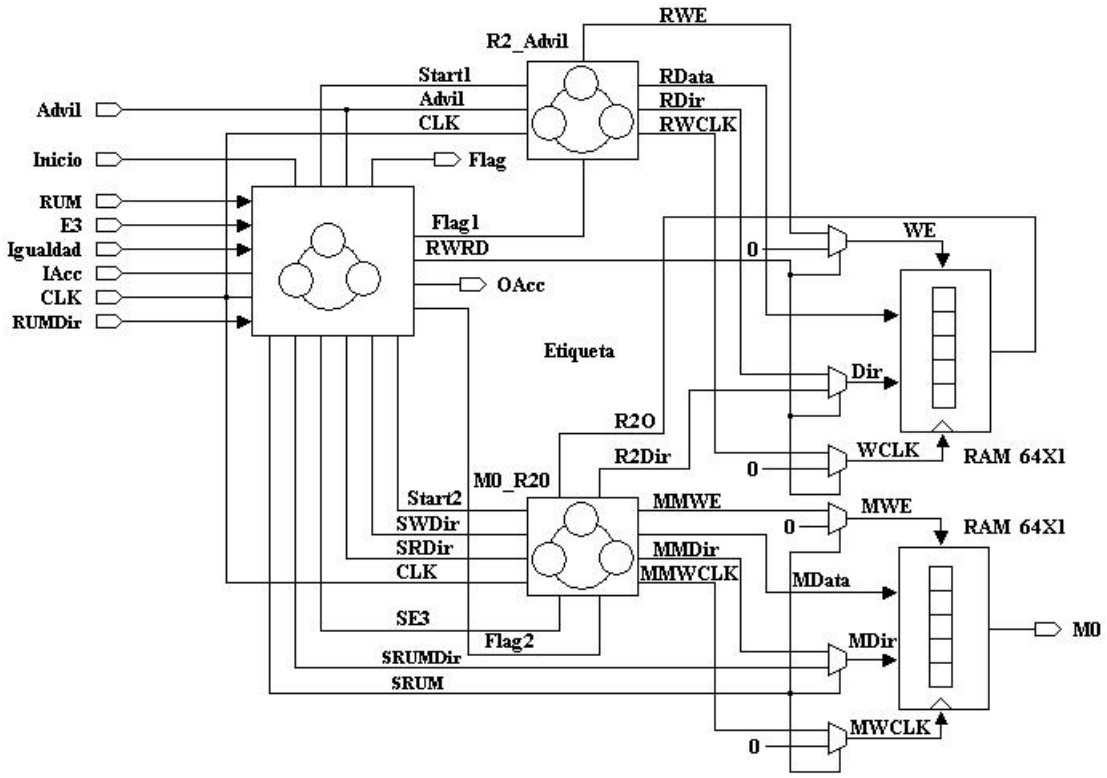


Figura 4.65: *Diagrama interno de la parte secuencial, que genera la etiqueta resultante de la compresión. Contiene tres máquinas de estados: R2_ Advil, M0_ R20 y Etiqueta. El inicio del proceso es determinado por la máquina de control.*

Contiene tres máquinas de estados: R2_ Advil, M0_ R20 y Etiqueta. Etiqueta es la máquina de estados central, ya que recibe las señales de entrada y se encarga de gestionar las señales de control para R2_ Advil y M0_ R20 .

La forma como se estructuró la máquina de estados Etiqueta es mostrada en un diagrama ASM (figura 4.66). La idea principal es copiar en una memoria RAM el vector Advil, para después generar, con ayuda de los vectores Igual y E3, la etiqueta correspondiente. Para efectuar esta operación, primero gestiona, por medio de la señal Start1, el inicio de la maquina R2_ Advil, la cual le proporciona los datos a manipular. Mediante la señal de activación Start2, controla el inicio de la máquina M0_ R20, la cual es la encargada de almacenar los datos de la etiqueta generada en la RAM 64X1.

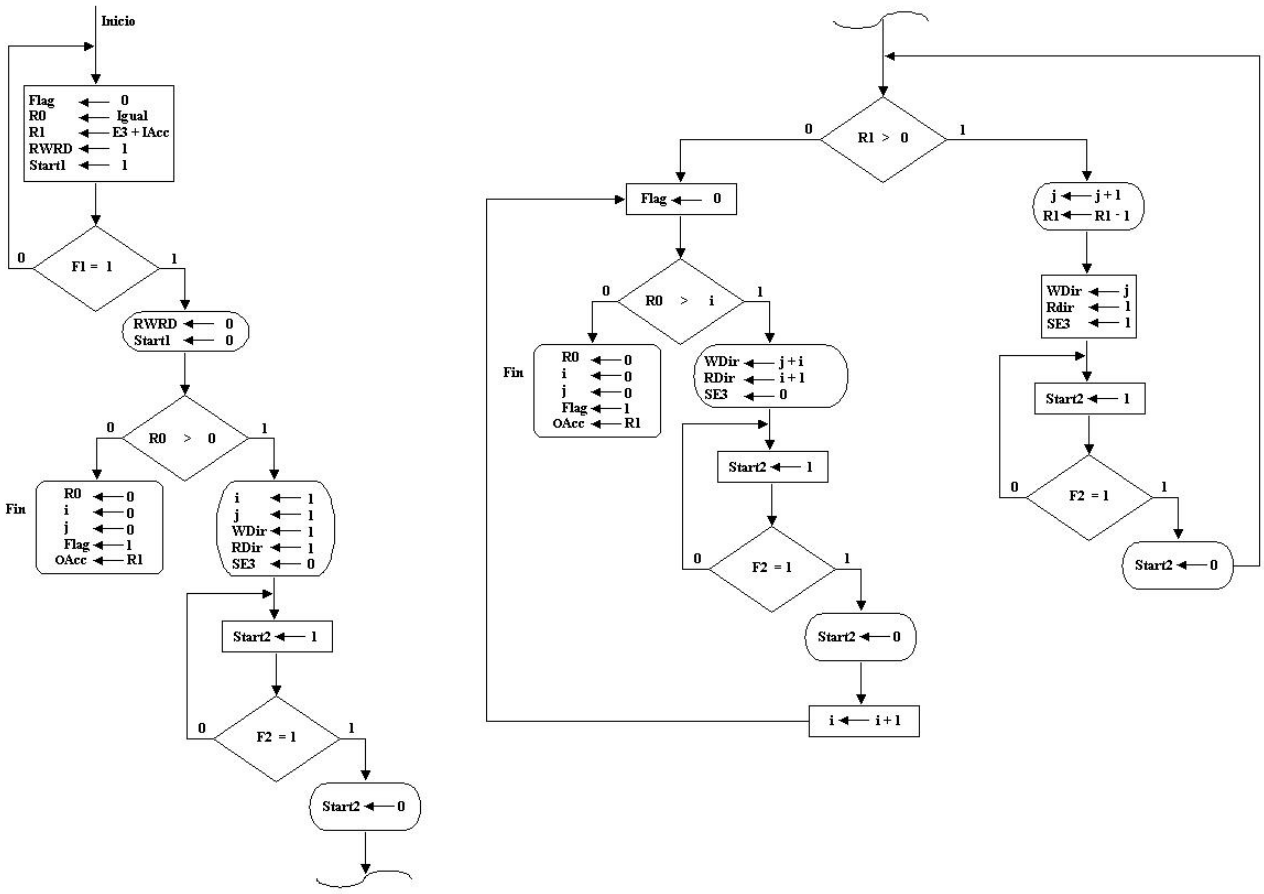


Figura 4.66: *Diagrama ASM del funcionamiento interno de la máquina llamada Etiqueta.*

La simulación del funcionamiento de este componente se puede observar en la figura 4.67.

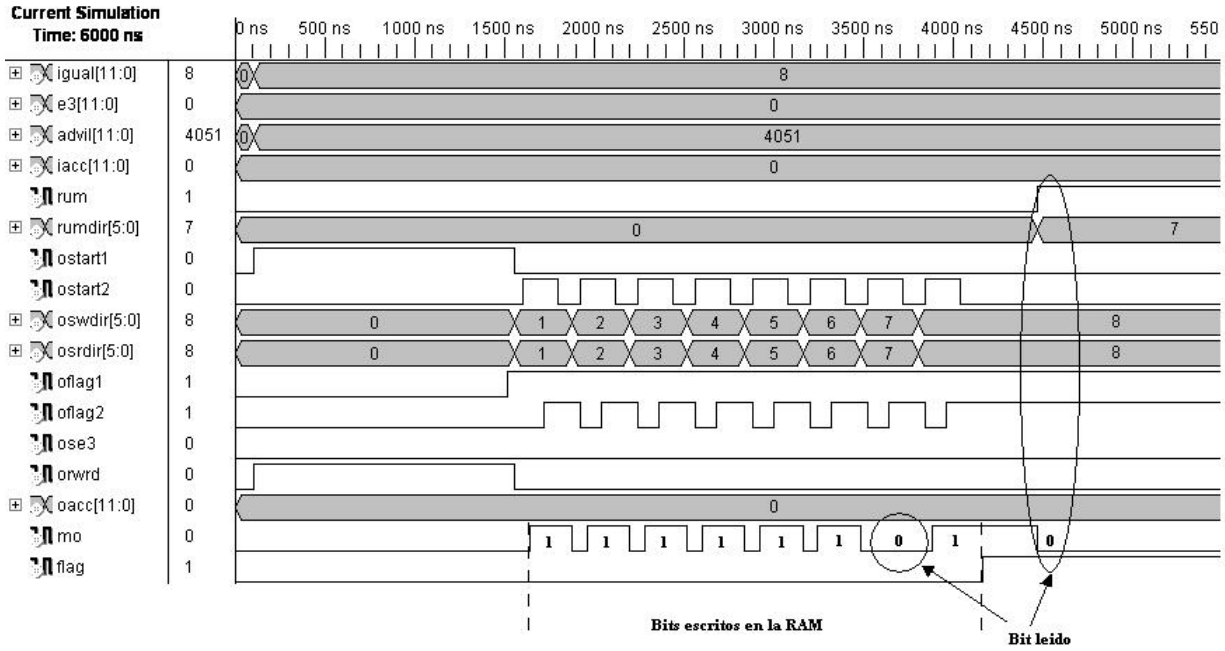


Figura 4.67: *Bloque M0_ R20, es una máquina de estados que realiza la copia.*

Maquina R2_ Advil

La función de este componente es realizar la copia del vector Advil en la memoria RAM 64X1. El cronograma que debe generar para la correcta escritura de la RAM se muestra en la figura 4.68.

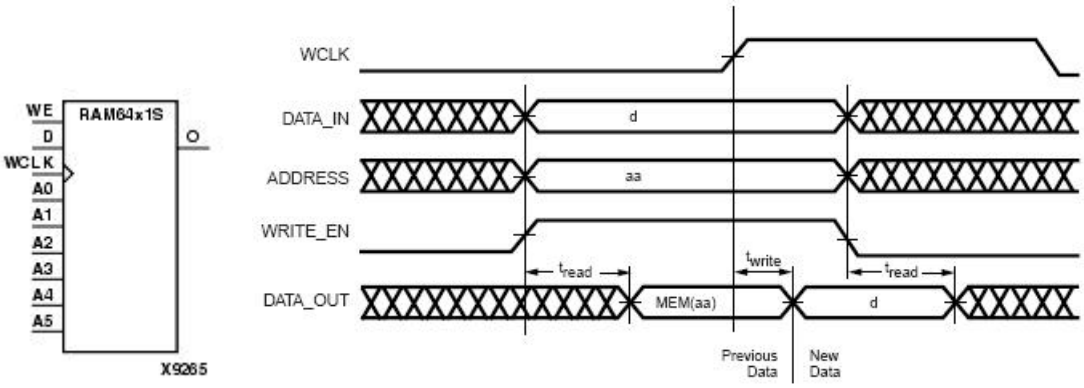


Figura 4.68: *Diagrama bloque y cronograma de la RAM estática de 64X1, datos proporcionados por Xilinx.*

Esta máquina tiene como entradas una señal que inicia el proceso de copiado (Start1), una (salida Flag1) que indica el final del proceso, un vector llamado Advil que contiene el dato a copiar, así como la entrada de reloj CLK y cuatro señales de control para la memoria RAM (figura 4.69).

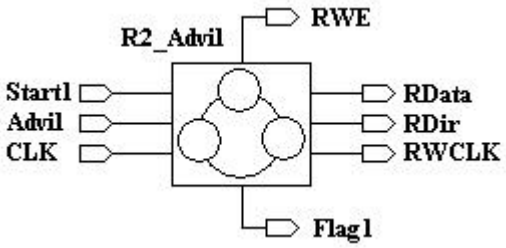


Figura 4.69: *Bloque R2_ Advil, es una máquina de estados que realiza la copia del registro Advil a la memoria R2.*

El diagrama de estados utilizado se muestra a continuación (fig. 4.70)

En la figura 4.71 se muestra la simulacion del funcionamiento de este componente. La parte resaltada corresponde al momento en el que se escribe en la memoria.

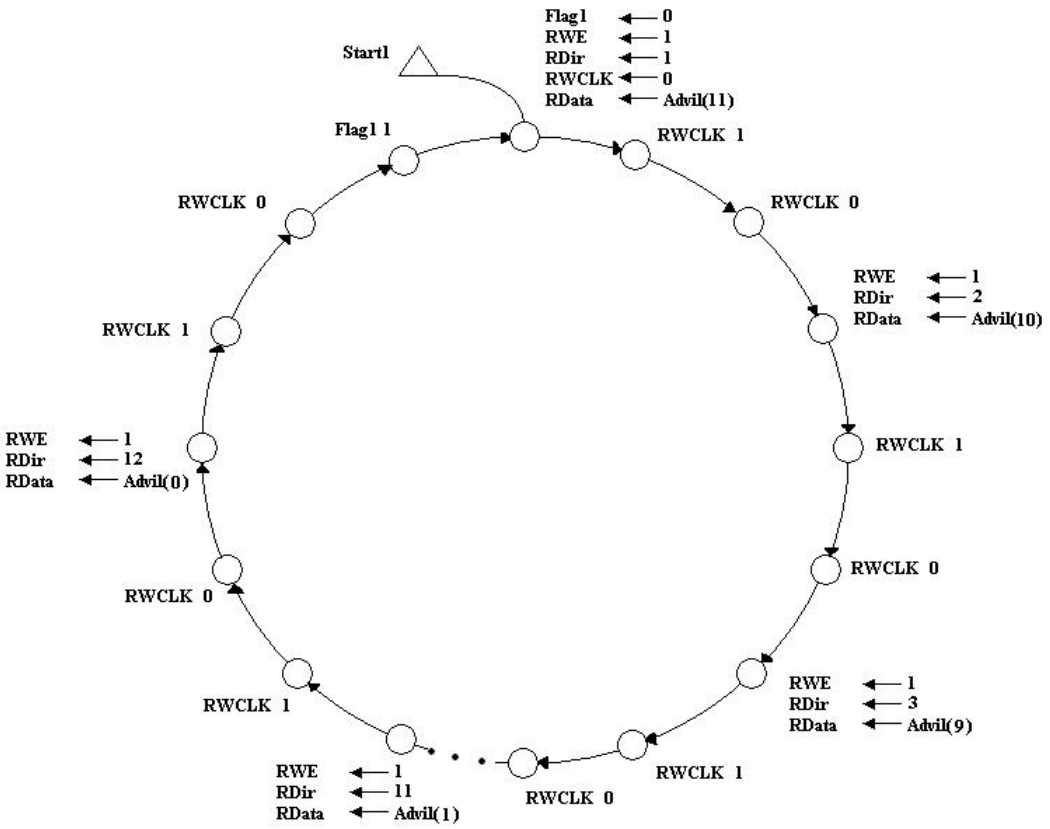


Figura 4.70: *Diagrama de estados del componente R2_ Advil.*

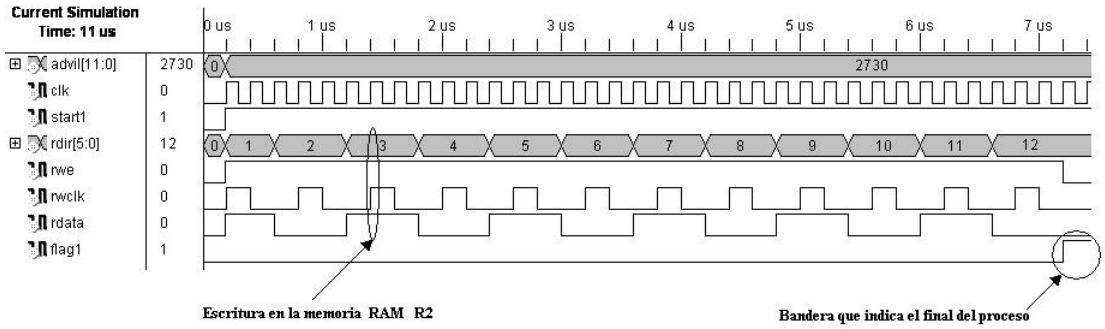


Figura 4.71: *Simulación de la operación de la máquina R2_ Advil.Se puede observar la correcta temporización de las señales de control, con lo que se asegura la correcta programación de la RAM.*

Máquina M0_ R20

La función de esta máquina es copiar el bit indicado RDir de la memoria R2 a la memoria UM, la dirección a donde es copiado y el sentido de RO2 es determinado por WDir y SE3.

La máquina M0_ R20 tiene como señales de entrada Start2, que da inicio al proceso, Flag2, que indica el final del mismo, R20, que contiene el bit a copiar, dos entradas WDir que contienen la posición a escribir en la memoria, UM y RDir que indican la dirección a leer de la memoria R2: si se activa la entrada SE3, se copia el bit R2O invertido.

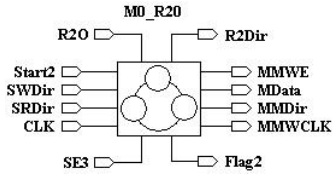


Figura 4.72: *Bloque M0_ R20, es una máquina de estados que realiza la copia.*

El diagrama de flujo mostrado en la figura 4.73 describe el funcionamiento de esta máquina.

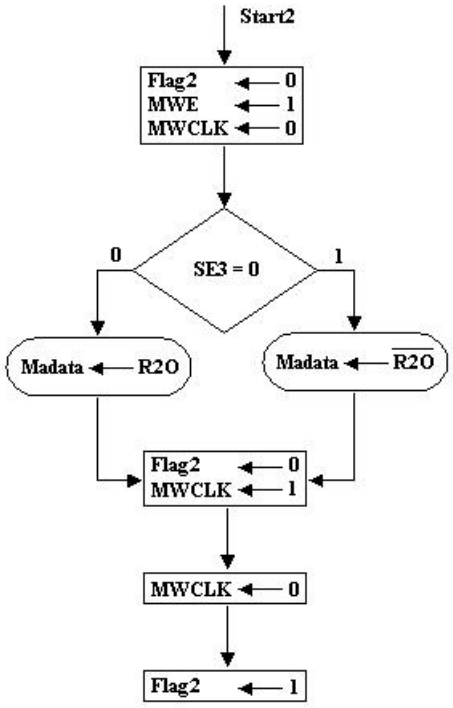


Figura 4.73: *Diagrama de flujo de la maquina M0_ R20.*

El correcto funcionamiento de la máquina fue evaluado por medio de la simulación mostrada en la figura 4.74.

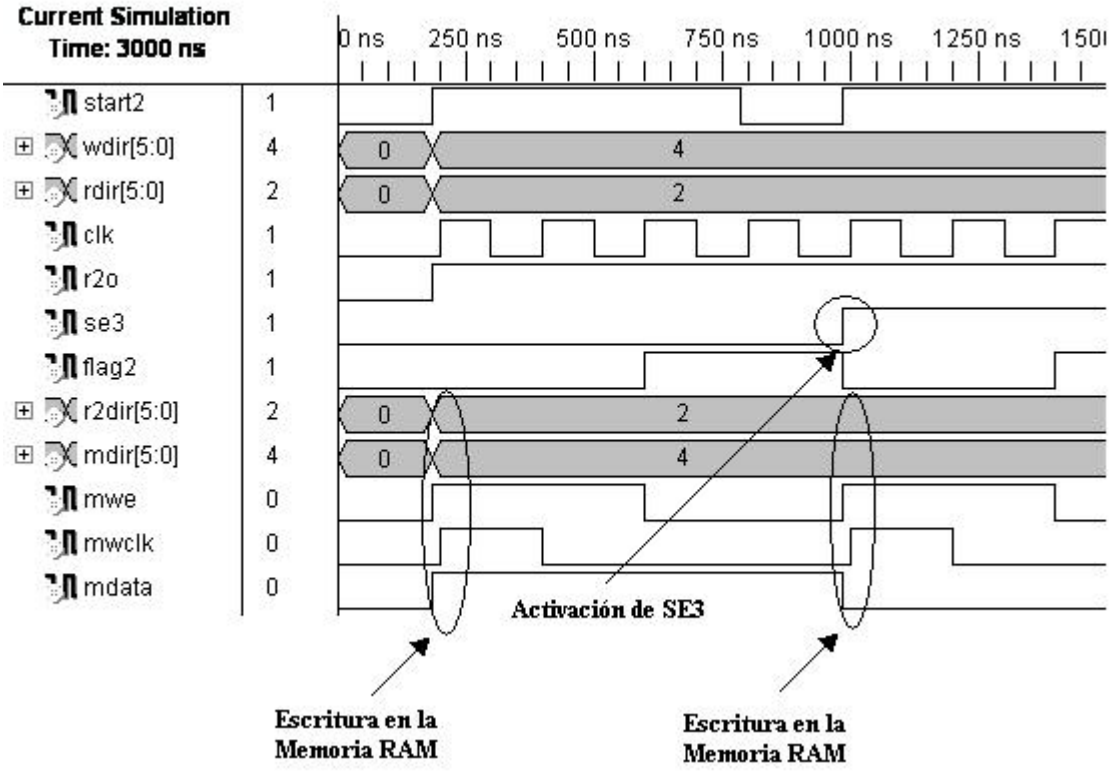


Figura 4.74: *Bloque M0_ R20, es una máquina de estados que realiza la copia.*

Máquina Control

Es un componente de nivel dos que realiza la sincronización de las máquinas encargadas de generar la etiqueta (parte secuencial) con la parte que realiza el cálculo de los límites (parte combinacional).

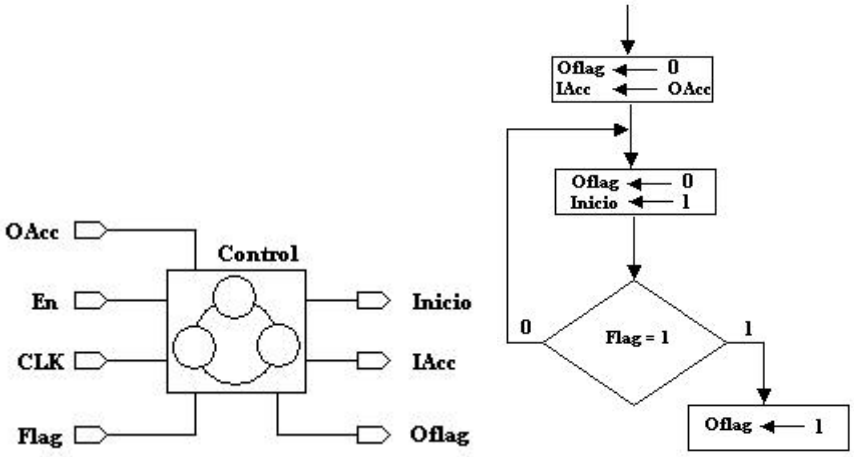


Figura 4.75: *Componente de nivel dos que gestiona el adecuado funcionamiento entre las etapas secuencial y combinacional.*

El funcionamiento de este componente es descrito mediante un diagrama ASM en que se plantean todas las variables de control (fig. 4.75).

Con el fin de verificar su correcto funcionamiento se simuló de manera independiente de todos los componentes que controla. El resultado de la simulación se puede verificar en la figura 4.76.

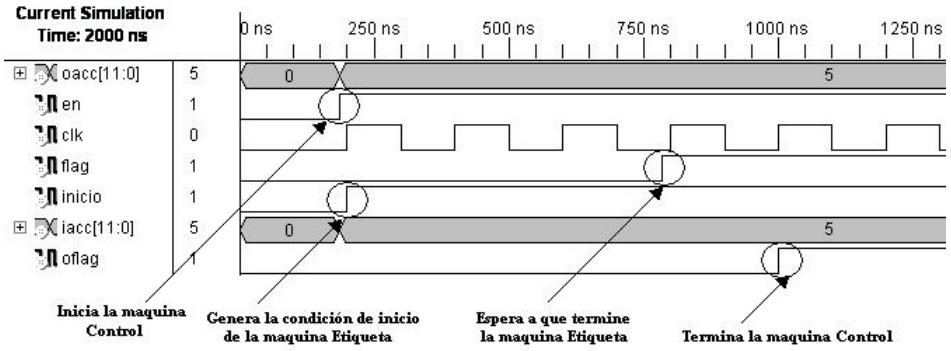


Figura 4.76: *Simulación de la máquina Control.*

4.4. Dispersor

Al describir en VHDL el **Dispersor**, generamos un componente de nivel tres con tres entradas (**b0**; **b1**; **b2**) y cinco salidas (**C0**; **C1**; **C2**; **C3**; **C4**), mostrado en la figura 4.77.

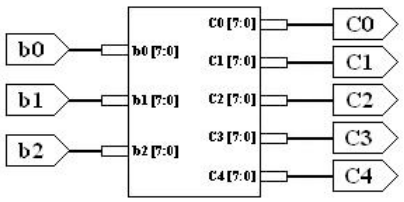


Figura 4.77: *Componente **Dispersor** de nivel tres, efectúa la dispersión de manera completamente combinacional. Consta de tres entradas de ocho bits cada una(**b0**; **b1**; **b2**), por donde recibe los datos a dispersar, y cinco salidas (**C0**; **C1**; **C2**; **C3**; **C4**) de ocho bits cada una a través de las cuales entrega los datos ya dispersos.*

Este componente está formado por tres componentes **Logaritmo** de nivel uno, ocho componentes **SemiMultiplicador** (Mx), de nivel dos, y cinco componentes **SumaModular** de nivel uno, interconectados como se muestra en la figura 4.78. Los componentes de nivel inferior serán descritos con más detalle posteriormente.

El **Dispersor** recibe en las entradas un vector $\tilde{\mathbf{b}}$ de tres elementos. Estos datos son multiplicados por cada uno de los valores de la matriz $\mathbf{A}$ (ec. 3.30) y los productos son sumados para formar los cinco componentes del vector $\tilde{\mathbf{c}}$, que formarán parte de los dispersos. Los elementos del vector $\tilde{\mathbf{c}}$ son mostrados en las terminales de salida. La operación completa realizada por el componente **Dispersor** es:

$$\begin{array}{rcl}
\mathrm{c}_0 &=& \mathrm{b}_0\cdot(\alpha^5+\alpha^4+1) \oplus \mathrm{b}_1\cdot(\alpha^5+\alpha^4+\alpha+1) \oplus \mathrm{b}_2\cdot(\alpha^5+\alpha^4+\alpha)\\
\mathrm{c}_1 &=& \mathrm{b}_0\cdot(\alpha^5+\alpha^4+1) \oplus \mathrm{b}_1\cdot(\alpha^5+\alpha^4+1) \oplus \mathrm{b}_2\cdot(\alpha^5+\alpha^4+1)\\
\mathrm{c}_2 &=& \mathrm{b}_0\cdot(\alpha^5+\alpha^4+\alpha) \oplus \mathrm{b}_1\cdot(\alpha^5+\alpha^4+\alpha+1) \oplus \mathrm{b}_2\cdot(\alpha^5+\alpha^4+1)\\
\mathrm{c}_3 &=& \mathrm{b}_0\cdot(\alpha^5+\alpha^4+\alpha) \oplus \mathrm{b}_1\cdot(\alpha^5+\alpha^4+\alpha) \oplus \mathrm{b}_2\cdot(\alpha^5+\alpha^4+\alpha+1)\\
\mathrm{c}_4 &=& \mathrm{b}_0\cdot(\alpha^5+\alpha^4+\alpha) \oplus \mathrm{b}_1\cdot(\alpha^5+\alpha^4+\alpha+1) \oplus \mathrm{b}_2\cdot(\alpha^5+\alpha^4+\alpha+1)
\end{array} \tag{4.3}$$

El siguiente ejemplo ilustra las operaciones realizadas para obtener el valor de c_0. Los datos a dispersar son: $\mathrm{b}_0 = 00000001$, $\mathrm{b}_1 = 00000010$ y $\mathrm{b}_2 = 00000011$. Las operaciones son efectuadas mediante la aritmética de campo finito, por lo que los valores deben ser representados como elementos del campo (sección 3.1.3). El cálculo de los restantes componentes del vector $\tilde{\mathbf{c}}$ es muy similar, por lo que no se muestra explícitamente.

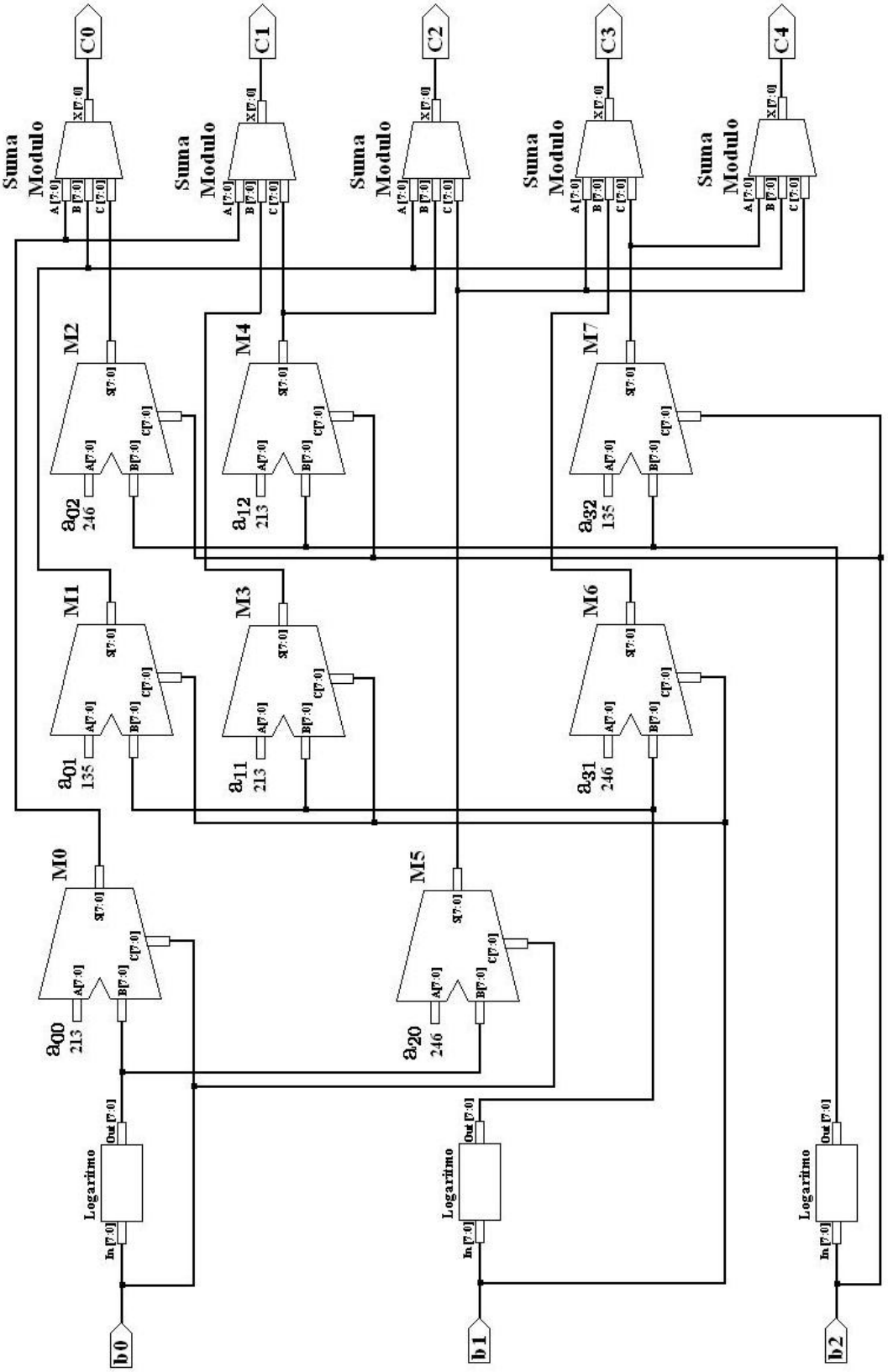


Figura 4.78: *Estructura interna del componente* ***Dispersor****, de nivel tres. Se muestran las interconexiones entre los diferentes componentes de niveles inferiores que lo forman: tres componentes* ***Logaritmo****, de nivel uno, ocho componentes* ***SemiMultiplicador(Mx)****, de nivel dos y cinco componentes* ***SumaModular****, de nivel uno.*

Ejemplo 4.2

Multiplicación $\mathrm{b_0 \cdot a_{00}}$

$$\mathrm{b_0}(\alpha) = 1$$

$$\mathrm{a_{00}}(\alpha) = \alpha^5 + \alpha^4 + 1$$

$$\log_\alpha(\mathrm{b_0}(\alpha)) = \log_\alpha(1) = \alpha^0$$

$$\log_\alpha(\mathrm{a_{00}}(\alpha)) = \log_\alpha(\alpha^5 + \alpha^4 + 1) = \alpha^{213}$$

$$\mathrm{b_0}(\alpha) \cdot \mathrm{a_{00}}(\alpha) = \alpha^0 + \alpha^{213} = \alpha^{213}$$

$$\alpha^{213 \ \% \ 255 \ = \ 213} = \alpha^{213}$$

$$\mathrm{Antilog}_\alpha \ (213) = \alpha^5 + \alpha^4 + 1$$

Multiplicación $\mathrm{b_1 \cdot a_{01}}$

$$\mathrm{b_1}(\alpha) = \alpha$$

$$\mathrm{a_{01}}(\alpha) = \alpha^5 + \alpha^4 + \alpha + 1$$

$$\log_\alpha(\mathrm{b_1}(\alpha)) = \log_\alpha(\alpha) = \alpha^1$$

$$\log_\alpha(\mathrm{a_{01}}(\alpha)) = \log_\alpha(\alpha^5 + \alpha^4 + \alpha + 1) = \alpha^{135}$$

$$\mathrm{b_1}(\alpha) \cdot \mathrm{a_{01}}(\alpha) = \alpha^1 + \alpha^{135} = \alpha^{136}$$

$$\alpha^{136 \ \% \ 255 \ = \ 136} = \alpha^{136}$$

$$\mathrm{Antilog}_\alpha \ (136) = \alpha^6 + \alpha^5 + \alpha^2 + \alpha^1$$

Multiplicación $\mathrm{b_2 \cdot a_{02}}$

$$\mathrm{b_2}(\alpha) = \alpha^1 + \alpha^0$$

$$\mathrm{a_{02}}(\alpha) = \alpha^5 + \alpha^4 + \alpha$$

$$\log_\alpha(\mathrm{b_2}(\alpha)) = \log_\alpha(\alpha^1 + \alpha^0) = \alpha^{231}$$

$$\log_\alpha(\mathrm{a_{02}}(\alpha)) = \log_\alpha(\alpha^5 + \alpha^4 + \alpha) = \alpha^{246}$$

$$\mathrm{b_2}(\alpha) \cdot \mathrm{a_{02}}(\alpha) = \alpha^{231} + \alpha^{246} = \alpha^{477}$$

$$\alpha^{477 \ \% \ 255 \ = \ 222} = \alpha^{222}$$

$$\mathrm{Antilog}_\alpha \ (222) = \alpha^6 + \alpha^4 + \alpha^2 + \alpha^1$$

$$\mathbf{C_0} = \alpha^5 + \alpha^4 + 1 \bigoplus \alpha^6 + \alpha^5 + \alpha^2 + \alpha^1 \bigoplus \alpha^6 + \alpha^4 + \alpha^2 + \alpha^1 = \mathbf{1}$$

En la figura 4.79 se muestra la simulación de la ejecución de las operaciones 4.3 con los datos del ejemplo 4.2 por el componente **Dispersor**.

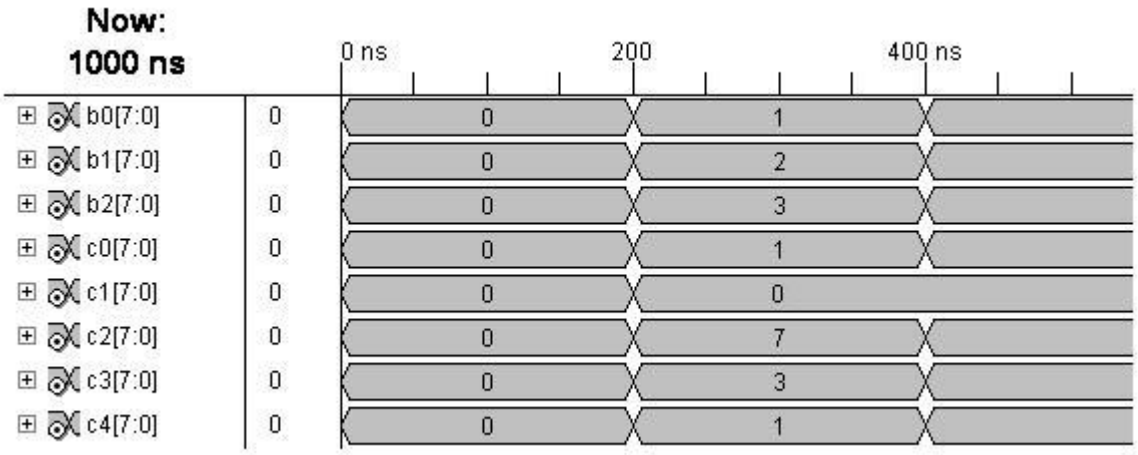


Figura 4.79: *Simulación del componente* ***Dispersor****. Los datos a dispersar son 00000001 (1), 00000010 (2)y 00000011 (3), contenidos en las entradas* $\mathbf{b0}, \mathbf{b1}$ *y* $\mathbf{b2}$*. El resultado de la dispersión se obtiene en las salidas* $\mathbf{C0}, \mathbf{C1}, \mathbf{C2}, \mathbf{C3}$ *y* $\mathbf{C4}$*. Puede verificarse que los resultados son correctos.*

Se puede verificar, realizando manualmente las operaciones arriba descritas, que los cinco vectores obtenidos por el **Dispersor** son correctos.

Componente SemiMultiplicador

De las ecuaciones 4.3 destaca que serían necesarios quince **Multiplicadores** como los descritos anteriormente (sección 4.1.2) para efectuar la dispersión. Un análisis detallado de estas ecuaciones pone en evidencia que el empleo de **Multiplicadores** normales resultaría altamente ineficiente, ya que este componente considera que sus operadores ($\mathbf{A}[\mathbf{7}:\mathbf{0}]$ y $\mathbf{B}[\mathbf{7}:\mathbf{0}]$) pueden tomar cualrquier valor entre 0 y 255, por lo que debe transformar cada operando a su representación logarítmica. Esto conllevaría la implementación de treinta componentes **Logaritmo**. Sin embargo, en las operaciones requeridas por la dispersión, sólo uno de los datos de entrada es desconocido, ya que el otro valor proviene de la matriz $\mathbf{A}$, la cual conocemos y es constante, por lo que podemos calcular sus logaritmos por anticipado, formando una nueva matriz $\mathbf{LA}$ (ec. 4.4). Además, el operador variable sería utilizado como entrada por otros cuatro **Multiplicadores**, lo que llevaría a repetir cinco veces la transformación logarítmica del mismo dato. Por estas razones, se optó por rediseñar el componente **Multiplicador** de tal forma que los componentes de nivel uno empleados para la conversión a logaritmo de las entradas fueran externos. El componente

de nivel dos resultante se nombró **SemiMultiplicador**.

$$\mathbf{LA} = \begin{pmatrix} \alpha^{213} & \alpha^{135} & \alpha^{231} \\ \alpha^{213} & \alpha^{213} & \alpha^{213} \\ \alpha^{231} & \alpha^{135} & \alpha^{213} \\ \alpha^{231} & \alpha^{231} & \alpha^{135} \\ \alpha^{231} & \alpha^{135} & \alpha^{135} \end{pmatrix} \tag{4.4}$$

El componente **SemiMultiplicador** (fig. 4.80) consta de tres entradas ($\mathbf{A}$,$\mathbf{B}$ y $\mathbf{C}$) y una salida ($\mathbf{X}$), de ocho bits cada una. Su operación consiste en sumar dos números en representación logarítmica ubicados en las entradas ($\mathbf{A}$,$\mathbf{B}$) y generar el antilogaritmo del total, mostrándolo en la salida $\mathbf{X}$. La entrada $\mathbf{A}$ recibirá siempre algún valor de la matriz $\mathbf{LA}$. La entrada $\mathbf{B}$ recibe el logaritmo de alguno de los vectores $\mathbf{b}$. Como consecuencia de realizar externamente la transformación logarítmica, es necesario asegurar que el valor en la entrada $\mathbf{B}$ no sea el logaritmo de cero, que matemáticamente no existe. Por razones prácticas, el componente **Logaritmo** genera una ambigüedad entre el logaritmo de uno y de cero. Por lo tanto, se requiere la entrada $\mathbf{C}$, la cual se encarga de discernir si un cero en $\mathbf{B}$ corresponde a una entrada $\mathbf{b} = \mathbf{1}$ o a $\mathbf{b} = \mathbf{0}$. Si es el último caso, forzará la salida $(\mathbf{X})$ a cero.

La operación realizada por este componente es:

$$X \;=\; \text{Antilog}_{\alpha}\ ((A+B)\ \mathit{modulo}\ 255) \tag{4.5}$$

En la figura 4.80 se simula la operación del componente **SemiMultiplicador** con los valores de entrada $\mathbf{0}$, $\mathbf{1}$, $\mathbf{2}$ y $\mathbf{3}$. El cero se agregó para ejemplificar el funcionamiento de la entrada $\mathbf{C}$.

Los valores en la variable $\mathbf{C}$ representan los datos a dispersar, ($\mathbf{0}$, $\mathbf{1}$, $\mathbf{2}$ y $\mathbf{3}$). La entrada $\mathbf{B}$ recibe estos mismos datos pero ya transformados por el bloque **Logaritmo**. En la entrada $\mathbf{A}$ se tienen los valores de la matriz $\mathbf{LA}$. Podemos observar cómo cuando la entrada $\mathbf{C}$ es puesta a cero, la salida también es cero. Cuando $\mathbf{C}$ es distinto de cero, la salida es el antilogaritmo de la suma de las entradas $\mathbf{A}$ y $\mathbf{B}$.

Cabe aclarar que, a pesar del aparente desfase en tiempos entre los dos Sumadores debido a la inserción del **Comparador** (fig. 4.80), el resultado obtenido es el esperado. Esta aparente contradicción se puede explicar gracias al mecanismo de mapeo en **LUTs** (Look-Up Tables) empleado durante la síntesis del diseño.

Los componentes **Sumador**, **Antilogaritmo**, **Comparador** y **Compuerta** son iguales a los que integran el **Multiplicador**, y fueron descritos en la sección 4.1.1. Los componentes **Logaritmo** externos también son iguales a los descritos en esa sección.

Gracias a la utilización de componentes **SemiMultiplicador** en la implementación del **Dispersor** se redujo drásticamente el número de componentes **Logaritmo** en comparación con los que se emplearían con los componentes **Multiplicador**: tres en lugar de treinta, una reducción del $90\,\%$.

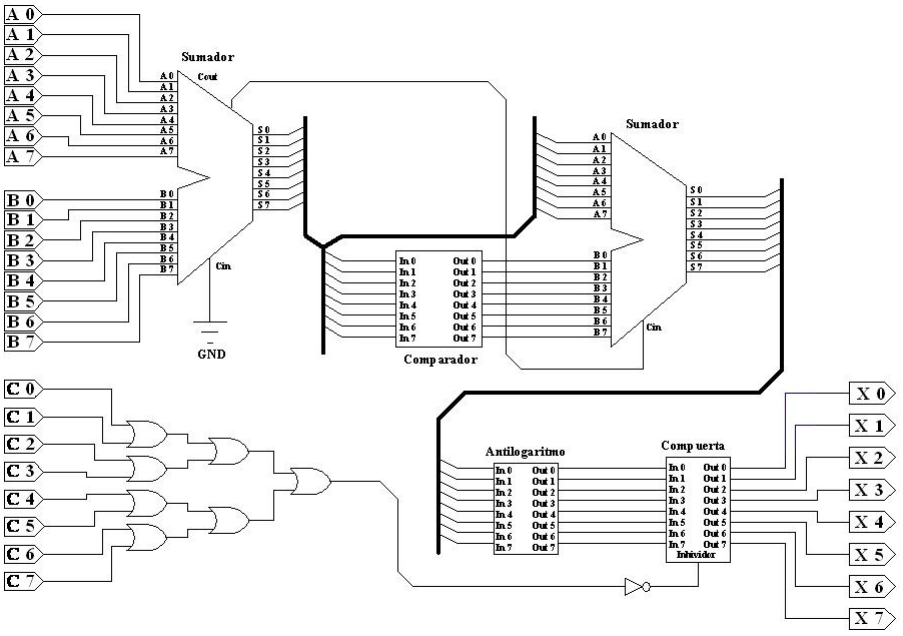


Figura 4.80: *Diagrama interno del componente de nivel dos* ***SemiMultiplicador****, compuesto por los siguientes componentes de niveles inferiores: dos componentes* ***Sumador****, un componente* ***Comparador****, un componente* ***Antilogaritmo*** *y uno* ***Compuerta****, todos de nivel uno. Este componente recibe en sus entradas* **A** *y* **B** *la representación exponencial de un valor de la matriz* **A** *y de los datos a dispersar respectivamente, mientras que en la entrada* **C** *recibe la representación vectorial de la entrada* **B***, a fin de detectar cuando un cero en* **B** *representa un valor por defecto y no la representación exponencial del logaritmo de uno.*

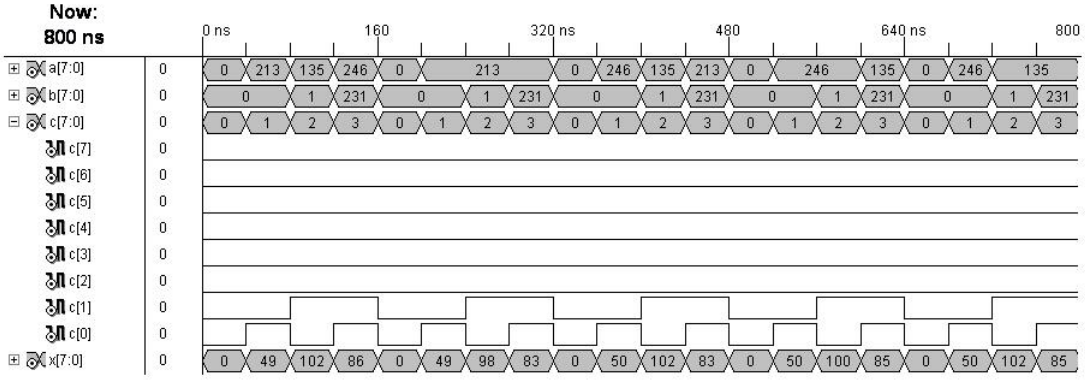


Figura 4.81: *Simulación de la operación del componente* ***SemiMultiplicador****. Los datos en las entradas corresponden a la representación exponencial de los valores empleados en el ejemplo 4.2, con un cero adicional para mostrar el funcionamiento de la entrada* **C***. Podemos corroborar que los valores obtenidos en las salidas corresponden a los resultados obtenidos en el ejemplo.*

Una optimización adicional consistió en reducir el número de componentes **SemiMultiplicador** de los quince previstos a ocho. Esto fue posible gracias a que, como algunos de los valores de la matriz **LA** se repiten, los resultados de las operaciones que los involucran son iguales.

Componente SumaModular

El componente de nivel uno **SumaModular** (figura 4.82) está formado por ocho compuertas XOR, por lo tanto es de nivel uno. Realiza la operación $\mathbf{A} \oplus \mathbf{B} \oplus \mathbf{C} = \mathbf{S}$.

El siguiente ejemplo realiza la operación de suma modular del ejemplo 4.2, para obtener el valor de c_0.

Ejemplo 4.3

	Representación vectorial	Representación polinomial
$\mathbf{A}[\mathbf{7:0}]$	(0,0,1,1,0,0,0,1)	$\alpha^6 + \alpha^5 + \alpha^0$
$\mathbf{B}[\mathbf{7:0}]$	$\bigoplus$(0,1,1,0,0,1,1,0)	$\bigoplus$ $\alpha^6 + \alpha^5 + \alpha^2 + \alpha^1$
$\mathbf{C}[\mathbf{7:0}]$	(0,1,0,1,0,1,1,0)	$\alpha^6 + \alpha^4 + \alpha^2 + \alpha^1$
$\mathbf{S}[\mathbf{7:0}]$	$(0,0,0,0,0,0,0,1)$	α^1

Cuadro 4.6: *Suma vectorial y polinomial en el campo* F_{2^8}

En la figura 4.83 se muestra la simulación de este bloque, que corresponde a la suma módulo dos de **A**, **B** y **C**. Los valores corresponden a los obtenidos en el ejemplo 4.2.

Construcción del archivo disperso

Una vez que ha concluido la construcción del vector $\tilde{\mathbf{c}}$, éste es enviado, junto con el número de renglón de la matriz que lo generó, vía USB a la computadora donde está instalado el programa de control. Éste se encarga de concatenar los datos enviados por el manipulador en un sólo archivo, el cual es almacenado en una terminal preseleccionada. Es importante resaltar que, debido a la forma en que fue diseñado el **Reconstructor**, ya no es necesario agregar al inicio del archivo todos los valores del renglón de la matriz **A** que generó el disperso (que representarían un total de 24 bits extra) sino que basta con indicar el número del renglón generador, lo que significa incorporar sólo ocho bits adicionales.

Matriz A				**Renglón**
a_{11}	a_{12}	a_{13}	$\longrightarrow$	$= 1$
a_{21}	a_{22}	a_{23}	$\longrightarrow$	$= 2$
a_{31}	a_{32}	a_{33}	$\longrightarrow$	$= 3$
a_{41}	a_{42}	a_{43}	$\longrightarrow$	$= 4$
a_{51}	a_{52}	a_{53}	$\longrightarrow$	$= 5$

(4.6)

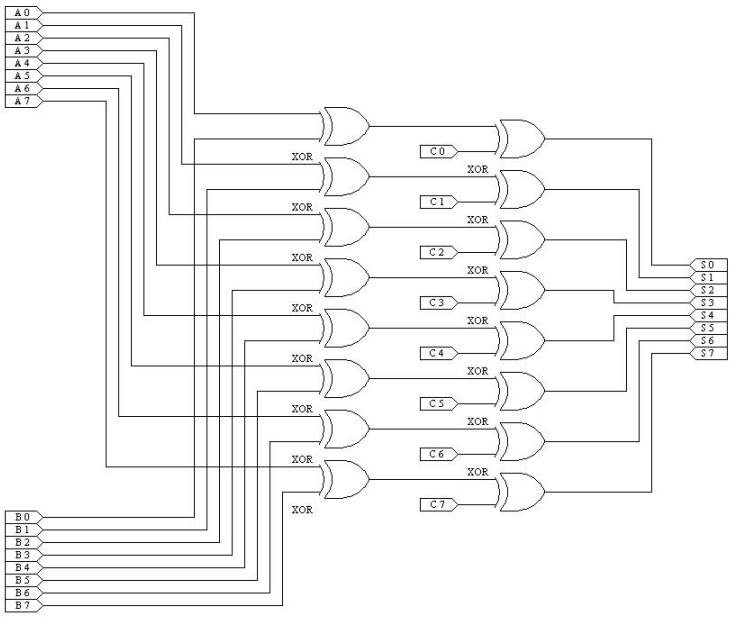


Figura 4.82: *Diagrama interno de componente de nivel uno* ***SumaModular****. Este componente efectúa una operación Or Exclusiva bit a bit entre tres datos de ocho bits cada uno, ingresados por las entradas* **A**, **B** *y* **C**. *El resultado, también de ocho bits, es entregado por la salida* **S**.

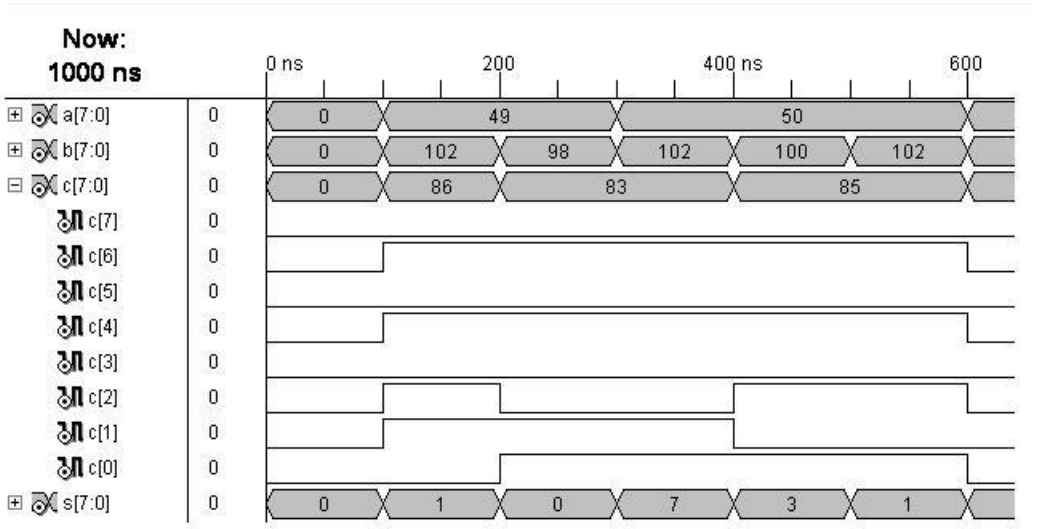


Figura 4.83: *Simulación del funcionamiento del componente* ***SumaModular****. Los datos corresponden a los empleados en el ejemplo 4.2*

4.5. Reconstrucción

Para la recuperación de la imagen, el receptor necesita cualesquiera tres de los cinco dispersos. El programa de control o el propio usuario se encargará de buscar los dispersos en las terminales de almacenamiento predestinadas. En caso de recuperar los cinco dispersos, se desecharán dos aleatoriamente seleccionados.

El resultado de la implementación en VHDL del algoritmo de reconstrucción fue un componente de nivel cuatro llamado **Reconstructor**, cuyo diagrama a bloques se muestra en la figura 4.84. Cuenta con tres entradas de ocho bits, **c0**; **c1**; **c2**, por donde se introducen los datos provenientes de los dispersos; tres entradas de cuatro bits, **r0**; **r1**; **r2**, en donde se indica, en orden consecutivo de menor a mayor, el número de renglón que generó cada disperso; y tres salidas de ocho bits, **b0**; **b1**; **b2** que contienen los datos obtenidos de la reconstrucción.

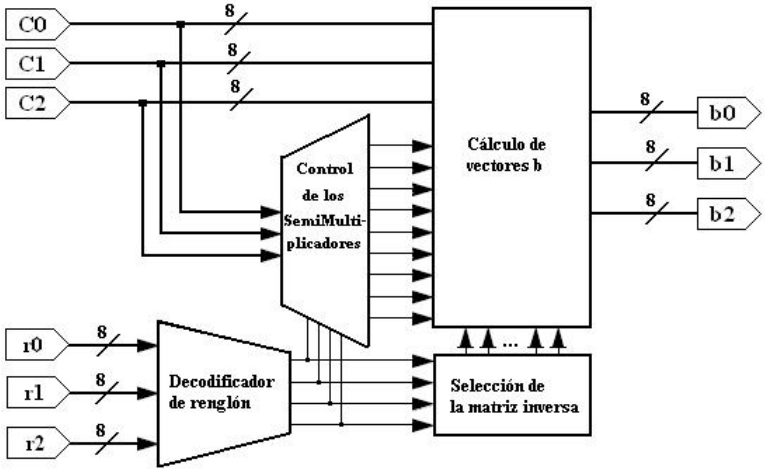


Figura 4.84: *Diagrama interno del componente de nivel cuatro* ***Reconstructor****, compuesto por los siguientes componentes de niveles inferiores:****Decodificador de Renglón****,* ***Selección de matriz inversa*** *,****Control de los semimultiplicadores*** *y* ***Cálculo de vectores b****. Este componente recibe en sus entradas* **c0**, **c1**, **c2** *los datos dispersados, mientras que las entradas* **r0**, **r1** *y* **r2** *reciben los numerales que identifican a la matriz* ***B*** *correspondiente a los dispersos ingresados. Tiene tres salidas* **b0**, **b1**, **b2** *por donde se obtienen los datos reconstruidos.*

En el diagrama se aprecia que este componente está integrado por cuatro componentes de menor nivel. El componente central, llamado **Cálculo de vectores b**, de nivel tres, es quien propiamente realiza la reconstrucción. Sin embargo, su correcta operación depende de la información proporcionada por los otros componentes, **Decodificador de Renglón**, **Selección de matriz inversa** y **Control de los semimultiplicadores**, cuyo funcionamiento será descrito en detalle más adelante.

Los valores de **c0**, **c1** y **c2** corresponden a los generados por los renglones 3, 4, y 5 de la matriz **A** al dispersar los datos 00000001,00000010,00000011(fig. 4.85). Las entradas

r0, **r1** y **r2** contienen el numeral de cada renglón generador. En las salidas **b0**, **b1** y **b2** encontramos los datos recuperados, que coinciden con los datos originales antes de la dispersión.

El siguiente ejemplo desarrolla paso a paso las operaciones requeridas para la recuperación del primer vector de los datos dispersados a partir de los archivos dispersos.

Ejemplo 4.4

En primer lugar, se recuperan los numerales indicadores de los renglones generadores.

$$\begin{array}{lcccc} r0 = 3 & \longrightarrow & a_{31} & a_{32} & a_{33} \\ r1 = 4 & \longrightarrow & a_{41} & a_{42} & a_{43} \\ r2 = 5 & \longrightarrow & a_{51} & a_{52} & a_{53} \end{array} \tag{4.7}$$

Estos valores nos indican qué renglones de la matriz $\mathbf{A}$ debemos tomar para construir la matriz $\mathbf{B}$. La representación polinomial resultante es:

$$\mathbf{B} = \begin{pmatrix} \alpha^5+\alpha^4+\alpha & \alpha^5+\alpha^4+\alpha+1 & \alpha^5+\alpha^4+1 \\ \alpha^5+\alpha^4+\alpha & \alpha^5+\alpha^4+\alpha & \alpha^5+\alpha^4+\alpha+1 \\ \alpha^5+\alpha^4+\alpha & \alpha^5+\alpha^4+\alpha+1 & \alpha^5+\alpha^4+\alpha+1 \end{pmatrix} \tag{4.8}$$

El siguiente paso es calcular la matriz inversa de $\mathbf{B}$, llamada $\mathbf{D}$.

$$\mathbf{D} = \begin{pmatrix} \alpha^7+\alpha^6+\alpha^3+1 & \alpha^7+\alpha^6+\alpha^5+\alpha+1 & \alpha^7+\alpha^6+\alpha^3 \\ 0 & 1 & 1 \\ \alpha^7+\alpha^5+\alpha^4+\alpha^3 & 0 & \alpha^7+\alpha^5+\alpha^4+\alpha^3 \end{pmatrix} \tag{4.9}$$

Es notable que en esta matriz existen valores iguales a cero, para los cuales no existe el logaritmo. Sin embargo, para efectuar las operaciones requeridas es necesario obtener la representación logarítmica de $\mathbf{D}$, llamada $\mathbf{LD}$, lo que conduce a la necesidad de asignar algún valor a los elementos indefinidos. Por razones prácticas, el valor por defecto escogido fue cero.

$$\mathbf{LD} = \begin{pmatrix} \alpha^{143} & \alpha^{144} & \alpha^{248} \\ 0 & \alpha^0 & \alpha^0 \\ \alpha^{254} & 0 & \alpha^{254} \end{pmatrix} \tag{4.10}$$

A continuación se muestran las operaciones efectuadas mediante las representaciones polinomial y exponencial con la finalidad de ilustrar la relación que existe entre ellas.

Multiplicación $c_0 \cdot d_{00}$

$$c_0(\alpha) = \alpha^2 + \alpha + 1$$

$$d_{00}(\alpha) = \alpha^7 + \alpha^6 + \alpha^3 + 1$$

$$\log_\alpha(c_0(\alpha)) = \log_\alpha(\alpha^2 + \alpha + 1) = \alpha^{59}$$

$$\log_\alpha(d_{00}(\alpha)) = \log_\alpha(\alpha^7 + \alpha^6 + \alpha^3 + 1) = \alpha^{143}$$

$$c_0(\alpha) \cdot d_{00}(\alpha) = \alpha^{59} + \alpha^{143} = \alpha^{202}$$

$$\alpha^{202 \ \% \ 255 \ = \ 202} = \alpha^{202}$$

$$\text{Antilog}_\alpha \ (202) = \alpha^7 + \alpha^4 + \alpha^3 + \alpha^2 + 1$$

Multiplicación $c_1 \cdot d_{01}$

$$c_1(\alpha) = \alpha + 1$$

$$d_{01}(\alpha) = \alpha^5 + \alpha^4 + \alpha + 1$$

$$\log_\alpha(c_1(\alpha)) = \log_\alpha(\alpha + 1) = \alpha^{231}$$

$$\log_\alpha(d_{01}(\alpha)) = \log_\alpha(\alpha^5 + \alpha^4 + \alpha + 1) = \alpha^{144}$$

$$b_1(\alpha) \cdot d_{01}(\alpha) = \alpha^{231} + \alpha^{144} = \alpha^{375}$$

$$\alpha^{375 \ \% \ 255 \ = \ 120} = \alpha^{120}$$

$$\text{Antilog}_\alpha \ (120) = \alpha^6 + \alpha^4 + \alpha^2$$

Multiplicación $c_2 \cdot d_{02}$

$$c_2(\alpha) = 1$$

$$d_{02}(\alpha) = \alpha^7 + \alpha^6 + \alpha^3$$

$$\log_\alpha(c_2(\alpha)) = \log_\alpha(1) = \alpha^0$$

$$\log_\alpha(d_{02}(\alpha)) = \log_\alpha(\alpha^7 + \alpha^6 + \alpha^3) = \alpha^{248}$$

$$c_2(\alpha) \cdot d_{02}(\alpha) = \alpha^0 + \alpha^{248} = \alpha^{248}$$

$$\alpha^{248 \ \% \ 255 \ = \ 248} = \alpha^{248}$$

$$\text{Antilog}_\alpha \ (248) = \alpha^7 + \alpha^6 + \alpha^3$$

$$\mathbf{b_0} = \alpha^7 + \alpha^4 + \alpha^3 + \alpha^2 + 1 \bigoplus \alpha^6 + \alpha^4 + \alpha^2 \bigoplus \alpha^7 + \alpha^6 + \alpha^3 = \mathbf{1}$$

A continuación se muestra la simulación de la operación global del componente **Reconstructor**. Los valores de las entradas **c0**, **c1** y **c2** corresponden a los datos provenientes de los archivos dispersos, mientras que las entradas **r0**, **r1** y **r2** contienen los números indicadores de los renglones generadores, ordenados de manera ascendente. En las salidas **b0**, **b1** y **b2** encontramos los valores originalmente dispersados, tomados del ejemplo 4.2.

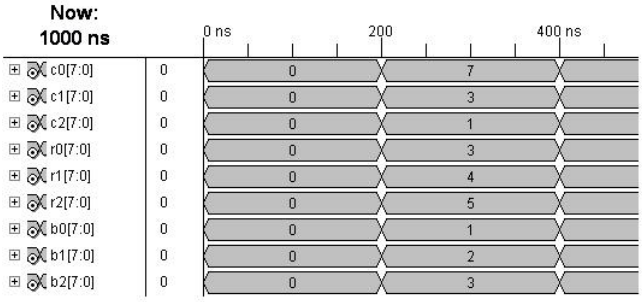


Figura 4.85: *Simulación del componente* ***Reconstructor*** *de nivel cuatro. Los datos en las entradas* **c0**, **c1** *y* **c2** *son los valores provenientes de los dispersos generados por los renglones 3, 4 y 5 de la matriz* **A**, *identificados en las entradas* **r0**, **r1** *y* **r2**. *Los datos reconstruidos están en las salidas* **b0**, **b1** *y* **b2**.

A continuación se describirán los componentes de menor nivel que integran el componente **Reconstructor**.

Componente Cálculo de $\tilde{\mathrm{b}}$

El componente de nivel tres llamado **Cálculo de** $\tilde{\mathrm{b}}$ realiza la multiplicación de cada elemento de la matriz inversa **D** con cada elemento del vector $\tilde{\mathbf{c}}$ y la suma modular de los productos resultantes para obtener los vectores $\tilde{\mathbf{b}}$, es decir, los datos reconstruidos.

$$\begin{aligned} b_0 &= c_0 \cdot d_{00} \oplus c_1 \cdot d_{01} \oplus c_2 \cdot d_{02} \\ b_1 &= c_0 \cdot d_{10} \oplus c_1 \cdot d_{11} \oplus c_2 \cdot d_{12} \\ b_2 &= c_0 \cdot d_{20} \oplus c_1 \cdot d_{21} \oplus c_2 \cdot d_{22} \end{aligned} \tag{4.11}$$

Este componente de nivel tres contiene nueve componentes **SemiMultiplicador**, tres componentes **Logaritmo** y tres componentes **SumaModular**, interconectados como se ilustra en la figura 4.86. La estructura y el funcionamiento de estos componentes fueron descritos en las secciones anteriores.

Este componente tiene en total 21 entradas de 8 bits:

- 3 por donde ingresan los valores de $\tilde{\mathbf{c}}$, llamadas c_0, c_1 y c_2,
- 9 nombradas a_{00},a_{01},...,a_{22}, en donde ingresan los valores de la matriz **D**, provenientes del componente **Selección de la matriz inversa**
- 9 denominadas $Inhibidor0, Inhibidor1, ..., Inhibidor8$, provenientes del componente **Control de los Semimultiplicadores**, que controlan el correcto funcionamiento de los semimultiplicadores

El vector resultante se entrega en las salidas de 8 bits b_0, b_1 y b_2.

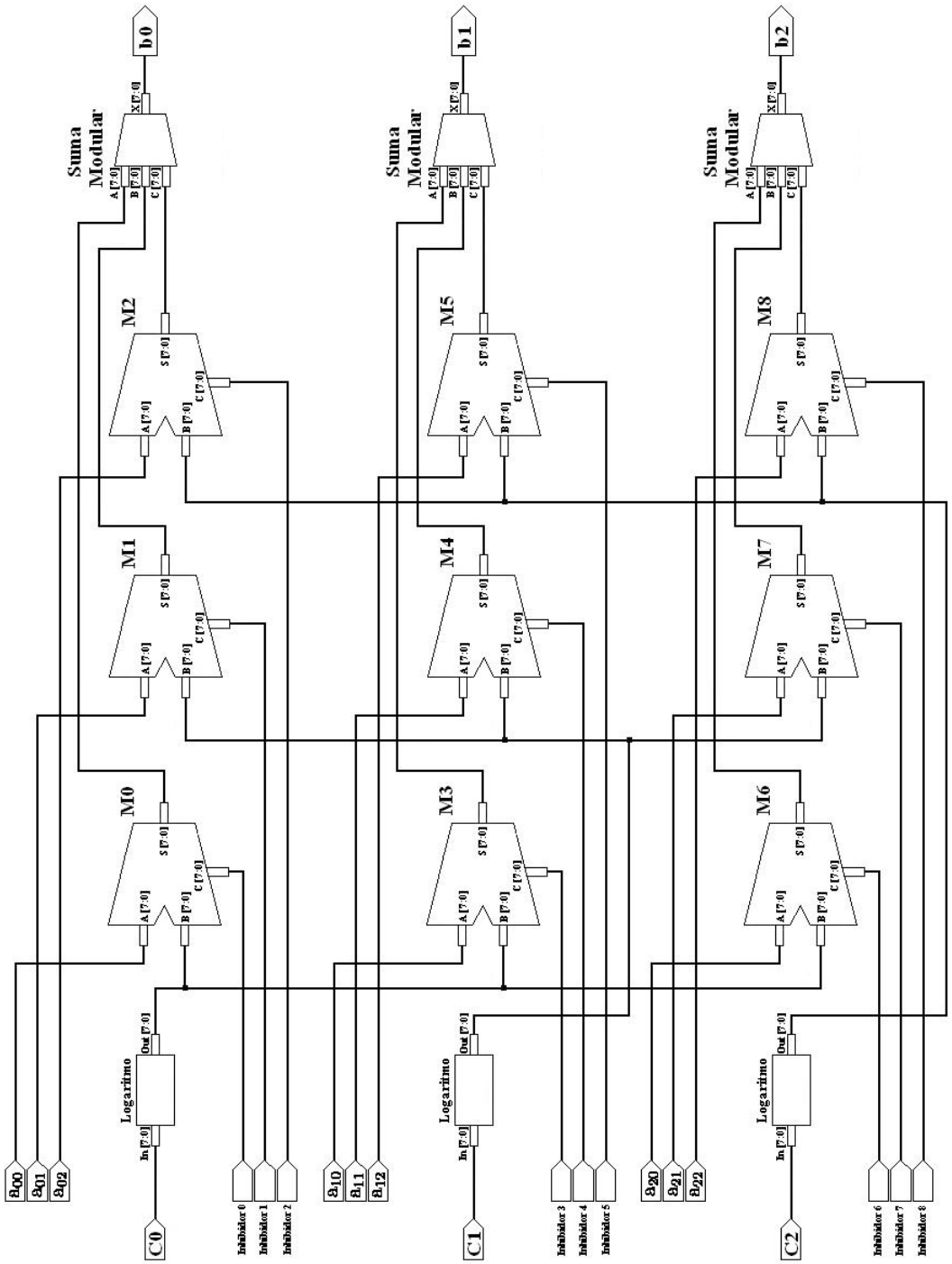


Figura 4.86: *Diagrama interno del componente de nivel tres* ***Cálculo de*** b̃. *Consta de nueve* ***SemiMultiplicadores****, tres* ***Logaritmos*** *y tres* ***SumaModular****. Realiza la multiplicación y suma modular de los datos a reconstruir por la matriz inversa apropiada. Recibe como entradas los datos de los dispersos y los valores de la matriz inversa por los cuales deben multiplicarse, así como señales de control para inhibir los* ***SemiMultiplicadores*** *en cuyas entradas se encuentren ceros asignados por defecto en la conversión a logaritmo de datos de entrada con valor cero.*

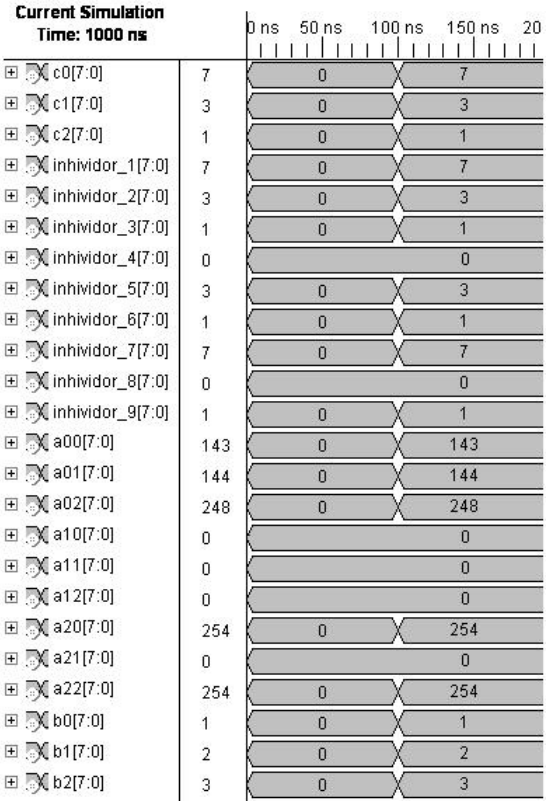


Figura 4.87: *Simulación del componente de nivel tres* ***Cálculo de*** $\tilde{\mathrm{b}}$. *Las entradas* **c0**, **c1** *y* **c2** *contienen los datos provenientes de los dispersos. Los valores en las entradas* $Inhibidor0, Inhibidor1, ..., Inhibidor8$ *indican qué valores, tanto de los datos de entrada como de la matriz inversa, corresponden a valores por defecto asignados durante la conversión a logaritmo a datos cero. Los valores en las entradas* a_{00},a_{01},...,a_{22} *introducen los valores de la matriz inversa apropiada para reconstruir los datos originales, recuperados en las salidas* b_0, b_1 *y* b_2.

La simulación del funcionamiento de este componente se muestra en la figura 4.87. Los datos empleados corresponden al ejemplo que se ha ido desarrollando.

Componente Selección de matriz inversa

Como se mencionó en la sección 3.5.2, es necesario generar la inversa de la matriz $\mathbf{B}$, de 3×3, compuesta por los renglones de la matriz $\mathbf{A}$ que dieron origen a los dispersos recuperados. Sin embargo, como puede construirse a partir de cualesquiera tres dispersos y sin importar el orden, existen $\binom{5}{3}$ posibles matrices $\mathbf{B}$, lo cual implicaría construir dentro del FPGA un algoritmo secuencial que fuera capaz de calcular la inversa de la matriz correspondiente a los dispersos seleccionados.

Como estrategia para simplificar el diseño y conservar su carácter combinacional, se decidió que los dispersos fueran ordenados automaticamente de menor a mayor antes de

formar la matriz $\mathbf{B}$. De esta manera, las posibles combinaciones de renglones se limitan a 10. Dado que el número de matrices es pequeño, podemos calcular previamente tanto sus inversas como su expresión logarítmica, para luego describir estas últimas como constantes en el FPGA.

El componente de nivel dos **Selección de matriz inversa** entrega al componente **Componente Cálculo de** $\tilde{\mathbf{b}}$ la matriz inversa apropiada. Consta de nueve multiplexores de $\mathbf{10}$ a $\mathbf{1}$ en una configuración de tambor, interconectados como se muestra en la figura 4.89. Cada multiplexor entrega a la salida uno de los nueve miembros (a00, a01,...,a22) que forman a la matriz inversa. Las entradas de selección llegan a los nueve multiplexores al mismo tiempo, haciendo que todos seleccionen la misma entrada, construyendo así la matriz solicitada por el componente **Decodificador de renglón**.

Los valores de las diez posibles matrices inversas fueron previamente calculados con el método de cofactores (descrito en el Apéndice A), convertidos a su representación logarítmica y descritos como constantes en este componente, por lo que sólo es necesario, dadas las entradas de un bit $\mathbf{s0}$; $\mathbf{s1}$; $\mathbf{s2}$; $\mathbf{s3}$ provenientes del componente **Decodificador de renglón**, seleccionar una de las diez matrices posibles. Los valores exponenciales de la matriz $\mathbf{LD}$ serán entregados por las terminales de salida de ocho bits $\mathbf{a00}, \mathbf{a11}, ..., \mathbf{a22}$ al componente **Cálculo de** $\tilde{\mathbf{b}}$, listos para ser utilizados por los **SemiMultiplicadores**.

Es importante resaltar que, debido a que las matrices ya están descritas en hardware, no se pierde tiempo de procesamiento en su cálculo.

La siguiente simulación muestra cómo seleccionar cada matriz. Cabe resaltar que la combinación $\mathbf{r0 = 3}$, $\mathbf{r1 = 4}$ y $\mathbf{r2 = 5}$ selecciona la matriz inversa número 10, cuyos valores corresponden a la matriz $\mathbf{LD}$ utilizada en el ejemplo 4.4.

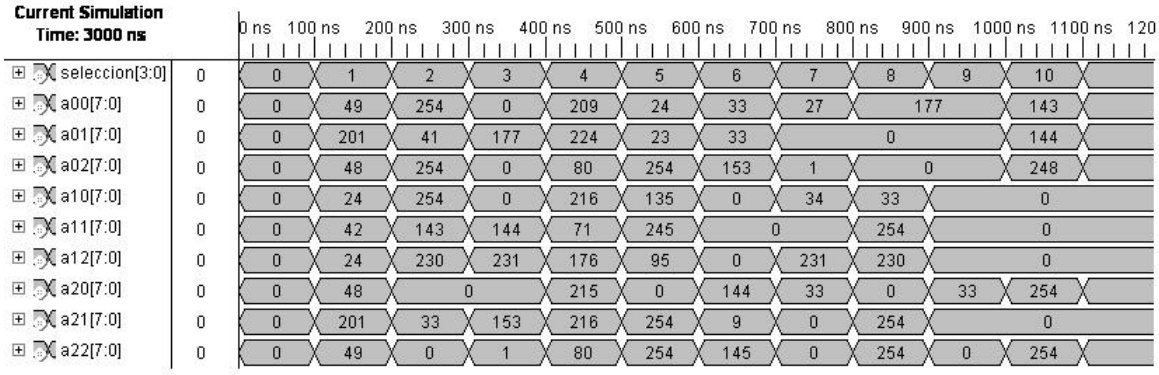


Figura 4.88: *Simulación de la operación del componente de nivel dos* ***Selección de la matriz inversa****. En ella se muestra cómo seleccionar cualquiera de las diez posibles matrices inversas. Los valores entregados en las salidas son la representación exponencial de la transformación logarítmica de estas matrices. La combinación de los renglones correspondientes a los valores del ejemplo 4.3 forman la matriz número diez, cuyos valores fueron empleados como entradas en la simulación del componente* ***Cálculo de*** $\tilde{\mathbf{b}}$*.*

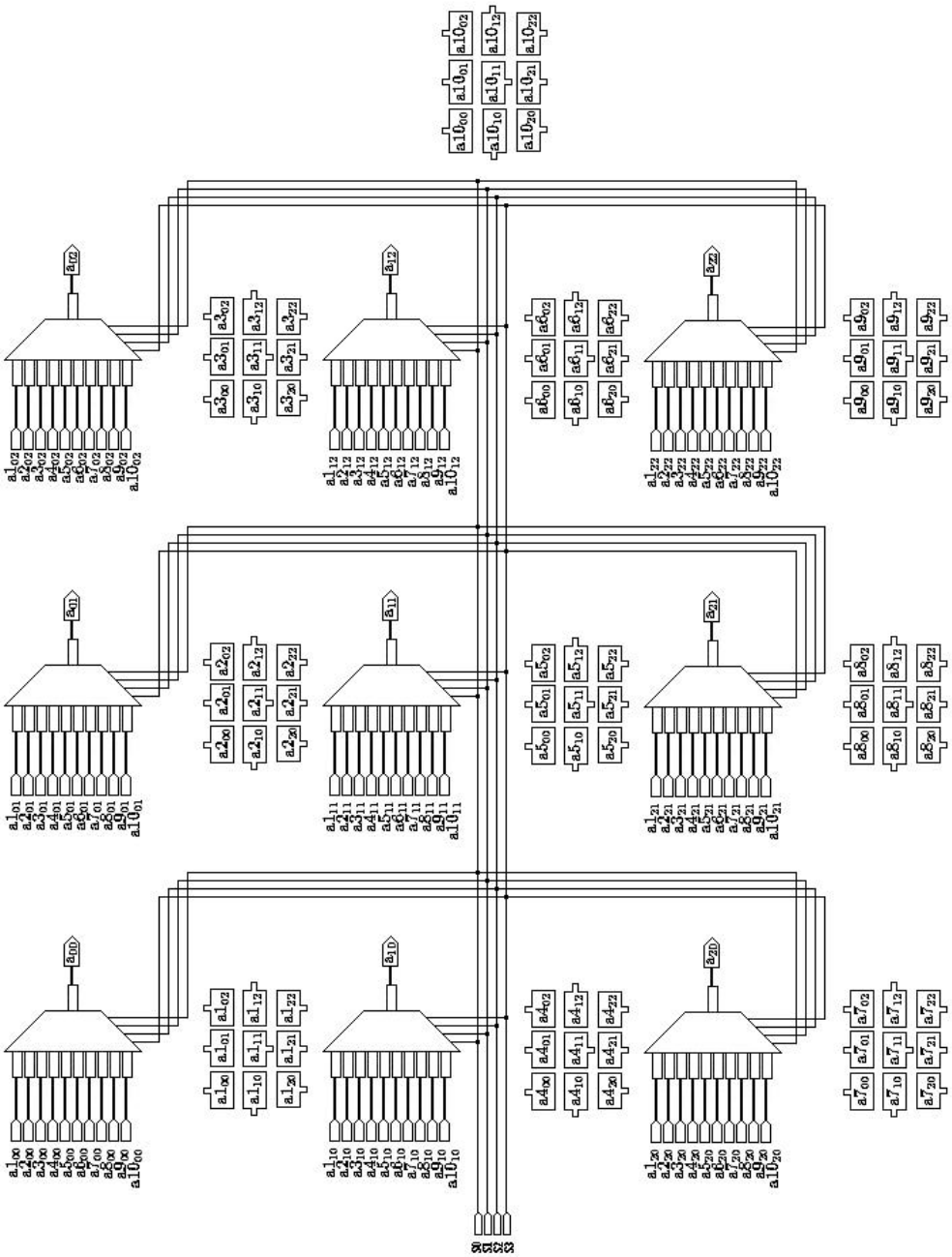


Figura 4.89: *Diagrama interno del componente de nivel dos* ***Selección de matriz inversa****. Cada uno de los nueve multiplexores contiene los diez posibles datos para un elemento de la matriz* **B***. Comparten los datos de selección, para que simultáneamente entreguen a la salida la matriz apropiada.*

Componente Decodificador de renglón

Este componente se implementó para que el FPGA seleccione la matriz inversa correspondiente al conjunto de los dispersos recuperados ordenados de menor a mayor.

El componente nivel uno **Decodificador de renglón** recibe tres entradas de 8 bits, en cada una de las cuales se indica qué número de renglón de la matriz **A** generó uno de los dispersos.

$$\begin{array}{cccc} a_{11} & a_{12} & a_{13} & \longrightarrow \quad r0 = 1 \\ a_{31} & a_{32} & a_{33} & \longrightarrow \quad r1 = 3 \\ a_{51} & a_{52} & a_{53} & \longrightarrow \quad r2 = 5 \end{array} \tag{4.12}$$

Una vez que este componente detecta los valores de **r0**; **r1**; **r2** genera una combinación de cuatro bits que es mostrada en las salidas **s0**; **s1**; **s2**; **s3** y que corresponde al número de la matriz inversa formada por los renglones indicados por los dispersos. Esta combinación es utilizada como selector por el componente **Selección de matriz inversa**. El diagrama general del **Decodificador de renglón** se ilustra en la figura 4.90.

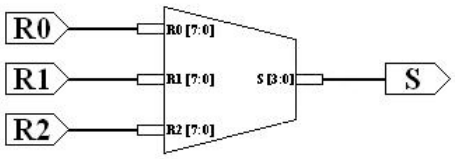


Figura 4.90: *Diagrama del componente de nivel dos* ***Decodificador de renglón****. Recibe en las entradas* **r0**; **r1** *y* **r2** *los índices de los renglones generadores de los dispersos recuperados. Con estos datos genera en sus salidas* **s0**; **s1**; **s2** *y* **s3** *la combinación de bits necesaria para seleccionar la matriz inversa adecuada.*

La simulación de la operación de este componente se muestra en la figura 4.91. En ella se aprecia cómo al cambiar las entradas se modifica el código de selección.

Componente Control de los SemiMultiplicadores

El siguiente componente de nivel dos es necesario porque el SemiMultiplicador no contiene bloques de conversión a logaritmos, sino que asume que los datos de entrada ya están convertidos a logaritmos. Esto ocasiona un problema, debido a que algunas de las matrices inversas contienen valores cero que no deben tomarse en cuenta. Sin embargo, si el **SemiMultiplicador** recibe un dato cero como entrada sin una señal de control que le indique lo contrario, lo toma como si fueran el logaritmo de uno, provocando una salida errónea.

Como las matrices inversas son conocidas es fácil ubicar qué datos deben ser tomados en cuenta y cuáles no. El componente **Control de los Semimultiplicadores** consta de cuatro entradas de un bit **s0**; **s1**; **s2**; **s3**, provenientes del **Decodificador de renglón**, que le hacen saber qué matriz inversa se utilizará, y nueve salidas de 8 bits **inhibidor1**, ..., **inhibidor9**,

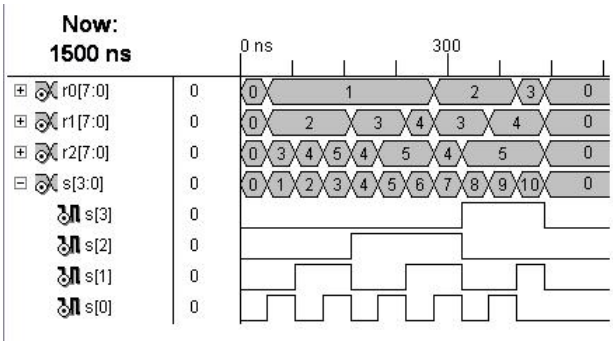


Figura 4.91: *Simulación del componente de nivel dos* ***Decodificador de renglón****. En las entradas se encuentran las diez posibles combinaciones resultantes de ordenar el número de renglón generador de menor a mayor. A la salida se obtiene una combinación de bits que identifica a una de estas combinaciones.*

que indican al componente **Cálculo de** $\tilde{\mathbf{b}}$ qué **SemiMultiplicador** debe bloquearse para que genere un cero a la salida. Las entradas de ocho bits $\mathbf{c0}, \mathbf{c1}, \mathbf{c2}$ monitorean si algún valor de los datos dispersados es cero, en cuyo caso también inhibirá al **SemiMultiplicador** que debería emplear este dato como factor (fig. 4.92).

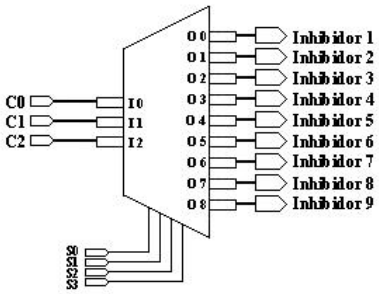


Figura 4.92: *Diagrama del componente de nivel dos* ***Control de los SemiMultiplicadores****. Este componente de control recibe a la entrada el número de matriz seleccionada y los datos a reconstruir. Con esta información, identifica si alguno de los* ***SemiMultiplicadores*** *se debe inhibir.*

En la figura 4.93 se muestra la simulación de la operación de este componente.

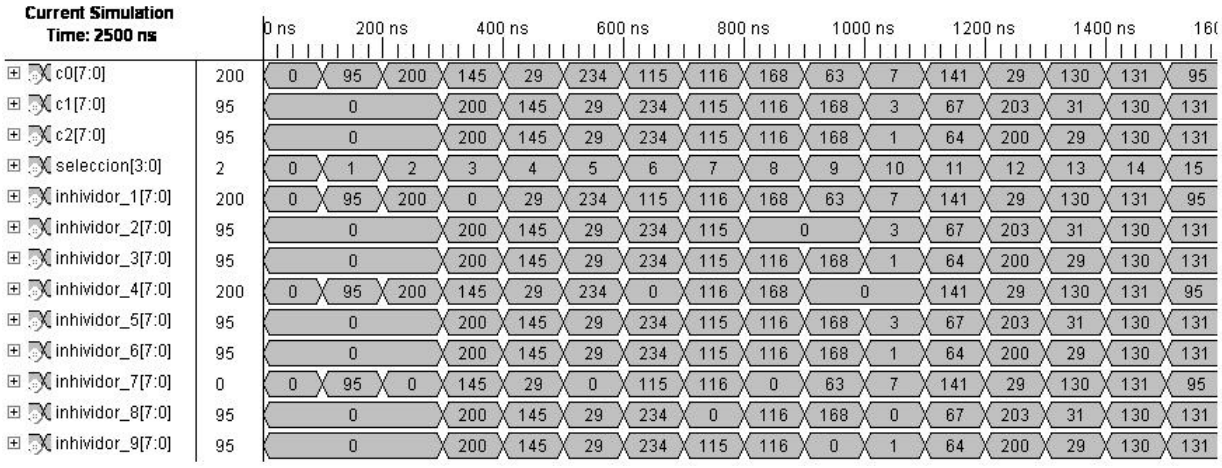


Figura 4.93: *Simulación del componente de nivel dos* ***Control de los SemiMultiplicadores****. Los datos en las entradas provienen del ejemplo 4.3. Las salidas correspondientes a los* ***SemiMultiplicadores*** *que deben inhibirse contienen un valor igual a cero.*

Capítulo 5
Desempeño del Manipulador de Imágenes Médicas

5.1. Metodología de evaluación del desempeño

La evaluación del desempeño del manipulador desarrollado consta de tres partes:

- La determinación de sus principales características: puesto que el dispositivo construido tiene como principales objetivos comprimir, dispersar y recuperar sin pérdida de información imágenes médicas, las variables de interés para la caracterización de su desempeño son el espacio de almacenamiento final requerido para los archivos dispersos en comparación con el espacio original, así como la integridad de la información recuperada.

- En segundo lugar, es necesario comparar el desempeño del esquema propuesto con esquemas alternativos que ofrezcan propiedades similares en cuanto al espacio de almacenamiento requerido (redundancia agregada) y la tolerancia a fallas.

- En la parte final se evalúa la rapidez con la que se ejecutan los algoritmos involucrados en la manipulación.

5.1.1. Imágenes de prueba

Para la determinación de las características de funcionamiento del manipulador empleamos 10 imágenes de resonancia magnética (MRI, magnetic resonance images), con pixeles de 8 bits y 4 kb de tamaño, llamadas mri8bpp.i, $1 \leq i \leq 10$. Estas imágenes fueron obtenidas de la transformación lineal de imágenes MRI de 16 bpp (bits por pixel), efectuada mediante un programa escrito en Matlab que se puede consultar en el apéndice A.

5.1.2. Caracterización del desempeño del manipulador

Eficiencia de la compresión

La eficiencia de un compresor está determinada por la relación existente entre el tamaño original y el tamaño resultante después del proceso. Existen varias formas de expresar esta relación, una de ellas es la razón de compresión (véase sección 3.2.3). Sin embargo, es más común representar la eficiencia de compresión mediante un porcentaje o tasa de compresión, que se define como el inverso de la razón de compresión multiplicado por cien.

En el caso del manipulador desarrollado, el tamaño final del archivo depende del desempeño de las etapas de que consta el esquema de almacenamiento (figura 1.1).

La primera etapa es el grado de reducción de la entropía conseguida mediante la transformación Wavelet : entre menos entropía posea un conjunto de datos, permitirá una codificación aritmética más eficiente y, en consecuencia, una mayor tasa de compresión. El resultado esperado de la transformada wavelet es la descomposición de la entrada en cuatro subbandas de un cuarto del tamaño de la imagen original y con menor entropía (sección 3.3). Por lo tanto, de esta etapa nos interesa verificar que efectivamente se obtengan las cuatro subbandas con el tamaño esperado y que su entropia sea menor a la de la imagen original.

En la siguiente etapa, cada subbanda es comprimida por el método de codificación aritmética (sección 3.4), que arroja como resultado un archivo comprimido por cada una. La variable medida en esta etapa fue la tasa de compresión de cada archivo.

La tercera etapa consiste en la concatenación de las subbandas comprimidas en un solo archivo, tarea realizada por el empaquetador. Este archivo será posteriormente dispersado por el algoritmo de Rabin. Para poder recuperar sin pérdida la información original, es necesario incorporar al archivo concatenado palabras de control que le permitan al desempaquetador distinguir dónde termina cada subbanda, de manera que sea posible separar el archivo reconstruido en los cuatro archivos a descomprimir. El tamaño final del archivo concatenado corresponde entonces a la suma del tamaño de cada subbanda comprimida más ocho bytes de control, dos por cada subbanda.

La cuarta y última etapa es la dispersión. Como resultado de esta transformación se generan cinco archivos dispersos por cada imagen, que serán distribuidos a diferentes computadoras de la red para su almacenamiento. El tamaño de éstos depende del número de archivos necesarios para la reconstrucción elegido (n). En este caso, n=3. Por lo tanto cada disperso deberá ser de tamaño $F/3$, donde F equivale al tamaño del archivo a dispersar, que en nuestro caso es el tamaño del archivo que contiene la concatenación de las subbandas comprimidas. El espacio de almacenamiento final que ocupará cada imagen dispersada será entonces $5 \times (F/3)$. Para esta etapa se evaluará que se generen correctamente los cinco archivos dispersos y que su tamaño sea el esperado.

Integridad de la información

Debido a que los archivos dispersos resultantes contienen sólo parte de la información original y además ésta se encuentra encriptada, la única manera de comprobar que la manipulación sin pérdida fue exitosa es recorriendo el camino en dirección contraria, es decir, efectuar sobre los archivos dispersos las operaciones inversas de cada etapa, de manera que al terminar se recupere un archivo del mismo tamaño que la imagen original. Si la información contenida en esta imagen recuperada coincide totalmente con la imagen original, pixel a pixel, sabremos que tanto el camino de ida como el de vuelta fueron exitosos y que la compresión efectivamente se realizó sin pérdida de información.

5.1.3. Comparación con otros esquemas

Una vez caracterizado el desempeño del esquema de manipulación propuesto en la presente tesis, es posible plantear las propiedades que deberían tener esquemas alternativos para que fueran comparables con éste. Debido a la naturaleza innovadora de la propuesta de manipulación y almacenamiento de información propuesta en la presente tesis no es fácil plantear esquemas con características comparables.

5.1.4. Resultados

Imágenes procesadas

En la figura 5.1 se muestra un ejemplo del tipo de imágenes empleadas para la evaluación del desempeño.

En el histograma podemos apreciar que los valores más frecuentes de los pixeles presentes en la imagen se encuentran en las regiones cercanas a 0 (correspondiente al negro) y a 70 (gris claro), no llegan a 255 (blanco). La imagen tiene poca resolución y detalles debido a la transformación aplicada sobre ella, sin embargo cumple con el tamaño adecuado (4K) para los propósitos de caracterización y evaluación del desempeño del manipulador.

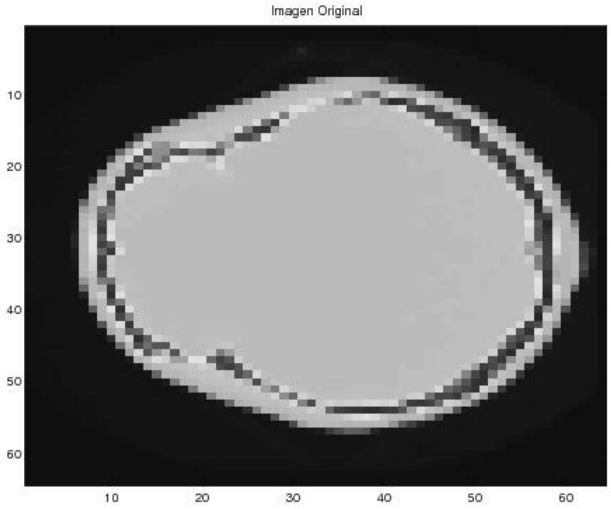


Figura 5.1: *La imagen es un ejemplo del tipo de imágenes utilizadas utilizadas para la evaluación del desempeño del manipulador*

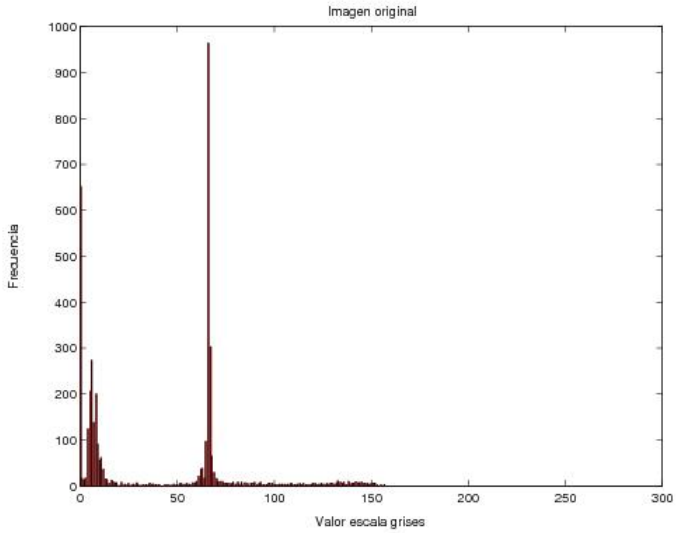


Figura 5.2: *Histograma de la imagen mostrada en la figura 5.1*

Eficiencia de compresión

La tabla 1 contiene los valores de las entropías originales de cada imagen evaluada.

Imagen	Entropía
mri8bpp.1	1.89692
mri8bpp.2	1.91647
mri8bpp.3	1.93795
mri8bpp.4	1.97536
mri8bpp.5	1.99417
mri8bpp.6	2.03673
mri8bpp.7	2.04838
mri8bpp.8	2.0315
mri8bpp.9	2.05459
mri8bpp.10	2.05751

Cuadro 5.1: *Entropías de las imágenes antes de la transformada Wavelet*

La siguiente tabla muestra los valores de las entropías de cada subbanda. El nombre de cada subbanda corresponde a las iniciales de la trayectoria de filtros seguida para su formación; así, la subbanda llamada FPAFPA indica que es el resultado del filtrado pasa altas de la primera y segunda descomposición wavelet.

	SubBandas			
Imagenes	FPBFPB	FPBFPA	FPAFPA	FPAFPB
mri8bpp.1	1.8834	1.1073	1.2981	1.1462
mri8bpp.2	1.8882	1.1346	1.2832	1.1158
mri8bpp.3	1.8959	1.1190	1.3301	1.1637
mri8bpp.4	1.9360	1.1777	1.3503	1.2395
mri8bpp.5	1.9742	1.1608	1.3825	1.2652
mri8bpp.6	2.0172	1.1420	1.3551	1.3089
mri8bpp.7	2.0388	1.1879	1.4621	1.3324
mri8bpp.8	2.0050	1.1612	1.3909	1.3125
mri8bpp.9	2.0362	1.1254	1.4090	1.2820
mri8bpp.10	2.0818	1.2958	1.5191	1.3526

Cuadro 5.2: *Entropías de las subbandas de las imagenes despues de la transformada Wavelet*

Como podemos observar, las entropías de la misma subbanda para cada imagen son similares. Esto se debe a que las imágenes empleadas se parecen entre sí. Sin embargo, es notorio que las subbandas FPBFPB tienen mayor entropía que las otras subbandas. La razón es que esta subbanda es la que conserva más información de la imagen original, al

ser resultado de únicamente dos decimaciones consecutivas, ya que el filtro de descomposición FPB es un filtro pasa todo.

La siguiente figura contiene las representaciones gráficas de las subbandas de salida de la transformación wavelet de la imagen de la figura 5.1. La imagen superior izquierda corresponde a la subbanda FPBFPB, a su derecha se encuentra la banda FPAFPA y en la fila inferior a la izquierda está la subbanda FPBFPA y a la derecha la FPAFPB. Es apreciable que la imagen FPBFPB contiene más información que el resto.

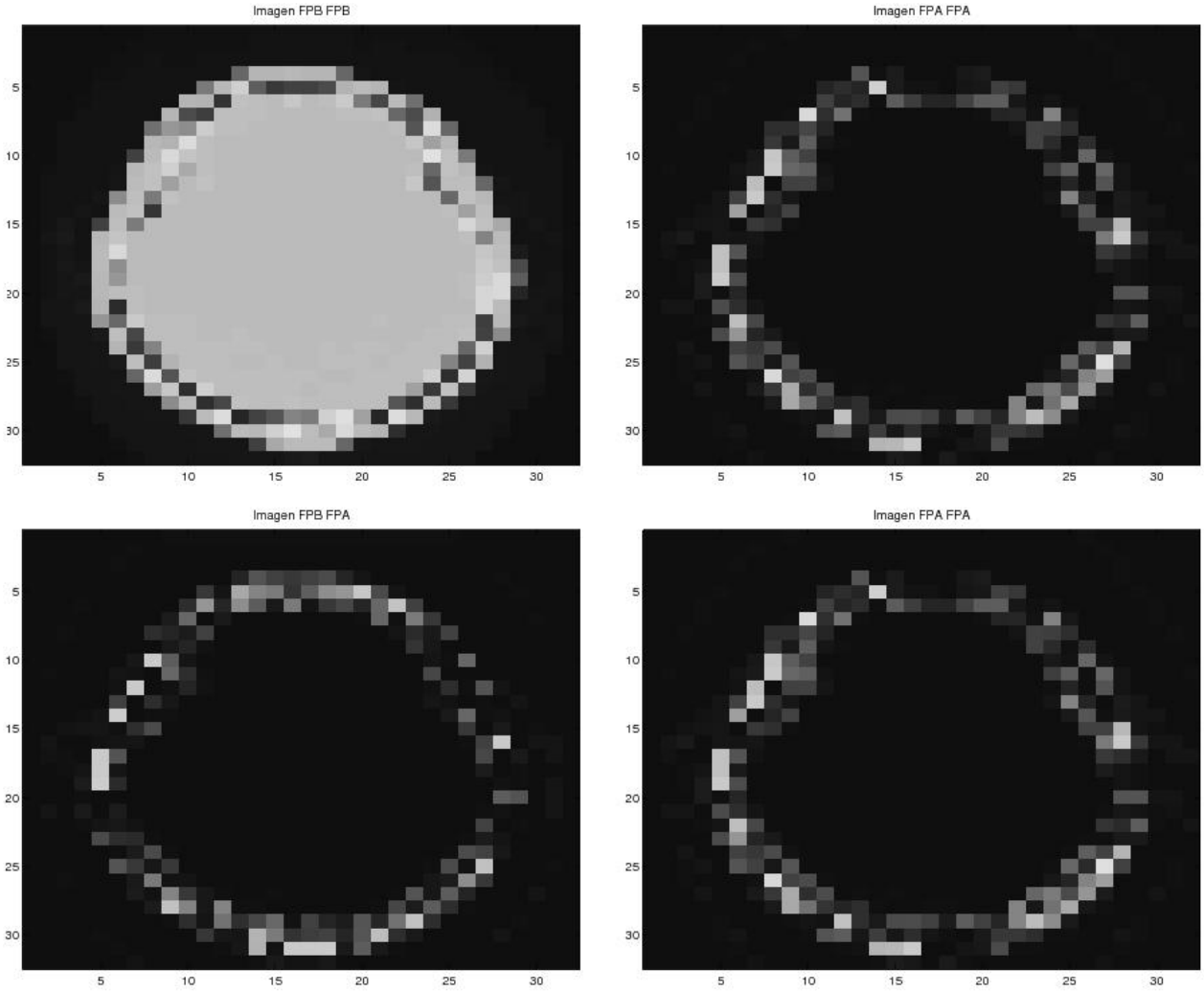


Figura 5.3: *Imágenes resultantes del filtrado*

El efecto de la transformación wavelet sobre la imagen original es muy evidente en la modificación de la morfología del histograma. La figura 5.3 contiene los histogramas de las subbandas. El objetivo de la transformación wavelet es reorganizar los pixeles de

tal forma que los que tengan valores parecidos se encuentren cercanos entre sí. Esto se traduce en un histograma menos disperso.

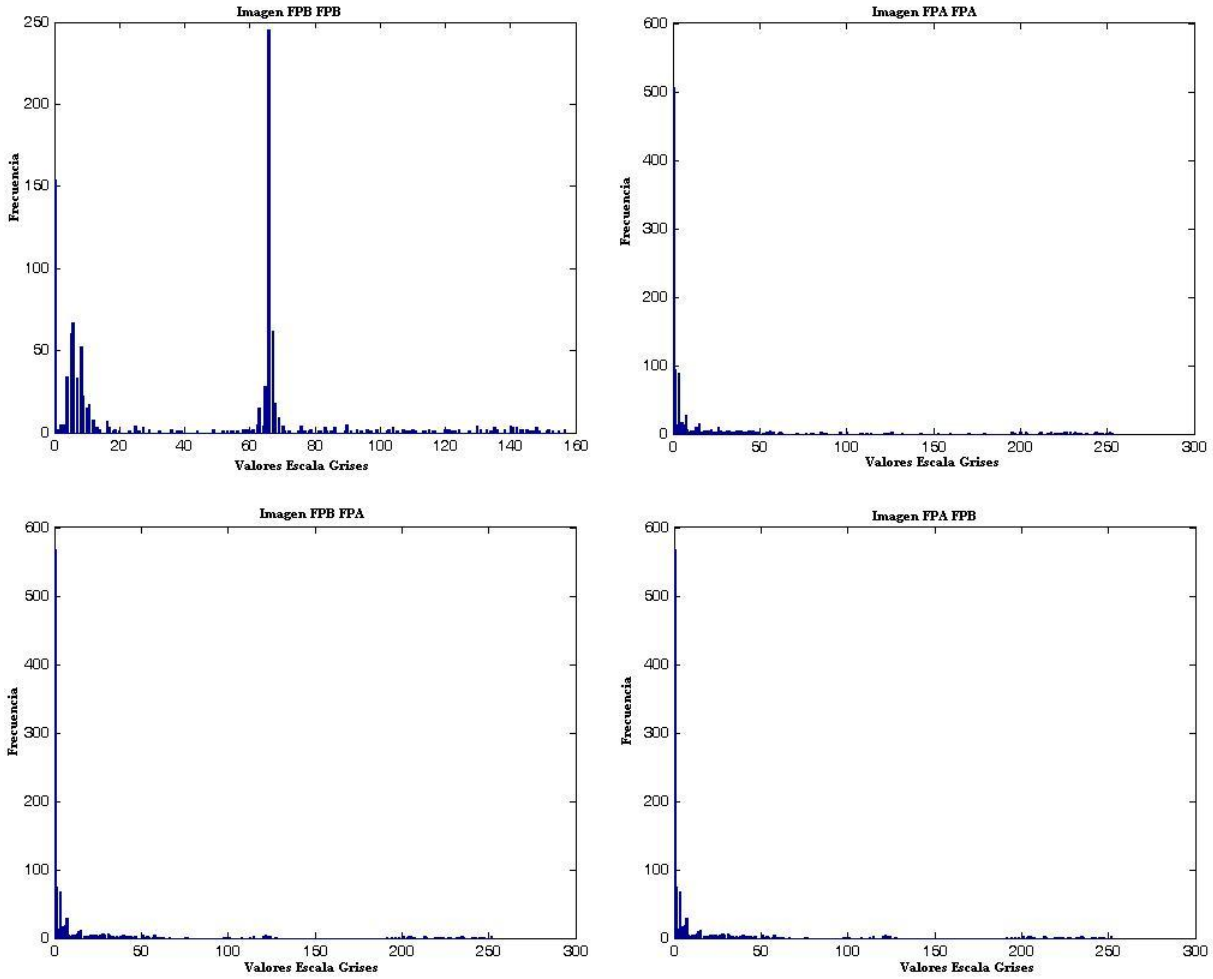


Figura 5.4: *Histograma de las imágenes resultantes de la transformación wavelet.*

Podemos apreciar que los histogramas de todas las subbandas están más concentrados que el de la imagen original. Aunque el histograma de la subbanda FPBFPB tiene la misma forma que el de la imagen original (fig. 5.1), la frecuencia de aparición de los pixeles contenidos es diferente, por efecto de la decimación. Como resultado de la compresión, en la siguiente tabla encontramos la tasa de compresión de cada una de las subbandas de las imágenes manipuladas:

Imagenes	SubBandas			
	FPBFPB	**FPBFPA**	**FPAFPA**	**FPAFPB**
mri8bpp.1	47.0	53.7	51.3	53.2
mri8bpp.2	46.9	53.3	51.5	53.6
mri8bpp.3	46.8	53.5	50.8	53
mri8bpp.4	46.3	52.8	50.6	52
mri8bpp.5	45.8	53.0	50.2	51.7
mri8bpp.6	45.3	53.3	50.5	51.1
mri8bpp.7	45.0	52.7	49.2	50.8
mri8bpp.8	45.4	53.0	50.1	51.1
mri8bpp.9	45.0	51.8	49.8	51.5
mri8bpp.10	44.4	51.3	48.4	50.6

Cuadro 5.3: *Tasas de compresión de las subbandas de las imágenes.*

Las tasas de compresión de las subbandas se encuentran en un rango del 40 al 50 %, aceptable para la compresión de imágenes. Este nivel de compresión pudo lograrse gracias a la reducción de las entropías efectuada por la transformación wavelet. Al concatenar las subbandas comprimidas en un solo archivo, la tasa de compresión global puede considererase como el promedio de las tasas individuales. En la tabla siguiente se muestra la tasa de compresión global de cada imagen procesada.

Imagen	**Compresión %**
mri8bpp.1	52.4
mri8bpp.2	52.4
mri8bpp.3	52.1
mri8bpp.4	51.5
mri8bpp.5	51.3
mri8bpp.6	51.2
mri8bpp.7	50.6
mri8bpp.8	51.0
mri8bpp.9	50.7
mri8bpp.10	49.8

Cuadro 5.4: *Tasa de compresión de los archivos formados por la concatenación de las subbandas comprimidas.*

Como resultado final de la vía de ida, la siguiente tabla contiene el tamaño de los dispersos de cada imagen. Se puede verificar manualmente viendo las propiedades del archivo y corroborando que el tamaño de los dispersos es el esperado, indicando así que la dispersión fue exitosa.

Imagen	Espacio requerido para almacenar la imagen (kb)
mri8bpp.1	3.18
mri8bpp.2	3.18
mri8bpp.3	3.20
mri8bpp.4	3.24
mri8bpp.5	3.25
mri8bpp.6	3.26
mri8bpp.7	3.30
mri8bpp.8	3.27
mri8bpp.9	3.29
mri8bpp.10	3.35

Cuadro 5.5: *Tamaño de los archivos dispersos de cada imagen procesada.*

Comparación con otros esquemas

Las características conseguidas para nuestro manipulador son : una tasa de compresión de alrededor del 50 %, generación de cinco archivos dispersos de tamaño $F/(3 \times 2)$, que serán almacenados en cinco computadoras conectadas a la red del hospital, que le otorgan al sistema una tolerancia a dos fallas, lo que se traduce en que sólo se requieran tres de los cinco archivos para recuperar el archivo original, no importando cuales de los cinco archivos sean.

Diseño de esquemas para comparación

Puesto que el esquema de manipulación y almacenamiento propuesto incorpora numerosas características novedosas, es difícil compararlo con un solo esquema "tradicional"de respaldo de imágenes médicas.

El primer esquema a comparar consiste en una compresión con una tasa del 50 %, generación de 5 copias que se almacenarán en diferentes computadoras. Este esquema se compara con el nuestro en la tasa de compresión y la cantidad de computadoras empleadas para el almacenamiento, pero sería tolerante a cuatro fallas en vez de sólo dos y se agregarían $5 * F$ bytes de redundancia (fig. 5.4). El espacio total requerido por este esquema es de $5 * F/2$ bytes.

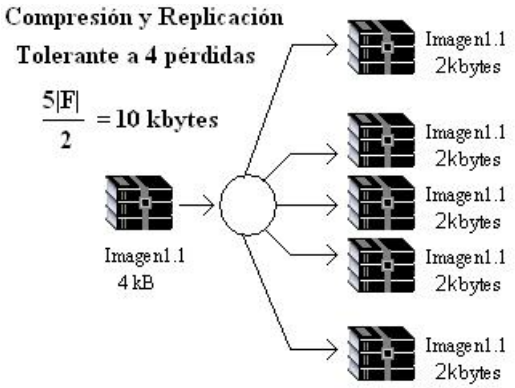


Figura 5.5: *El diagrama y desempeño de un esquema de replicación simple tolerante a cuatro fallas, que consiste en generar 5 copias de cada imagen y almacenarlas en diferentes equipos. El espacio de almacenamiento requerido por una imagen de 4kb bajo este esquema sería de* $5 * F = 20kb$. *A la derecha se encuentran el diagrama y desempeño de un esquema de compresión y replicación tolerante a cuatro fallas. Consiste en comprimir en un 50 % la imagen y generar cinco copias que se almacenarán en una computadora diferente cada una. El espacio de almacenamiento total requerido es de* $5 * F/2$ *bytes.*

El segundo esquema consiste en comprimir las imágenes en un 50 %, generar tres copias y almacenarlas en tres computadoras diferentes. Este esquema tiene una tolerancia a dos fallas, la redundancia agregada sería de $3 * F/2$ (fig. 5.5).

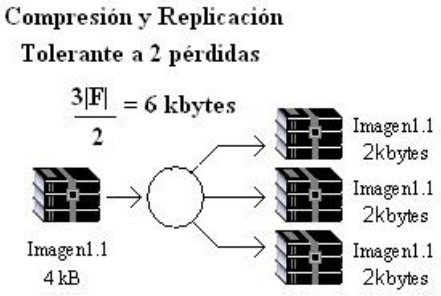


Figura 5.6: *El esquema de compresión y replicación tolerante a dos fallas requiere un espacio de almacenamiento de 3*F/2.*

Bajo cualquiera de estos dos esquemas, cualquier copia que sea recuperada y descomprimida contendrá la misma información de la imagen original. Sin embargo, ninguno de estos esquemas incluye la transformación lineal de los datos que tiene un efecto de encriptado, por lo que cualquier persona que abra los archivos podría tener acceso a la información contenida en ellos.

En la figura 5.6 se puede apreciar graficamente el esquema propuesto.

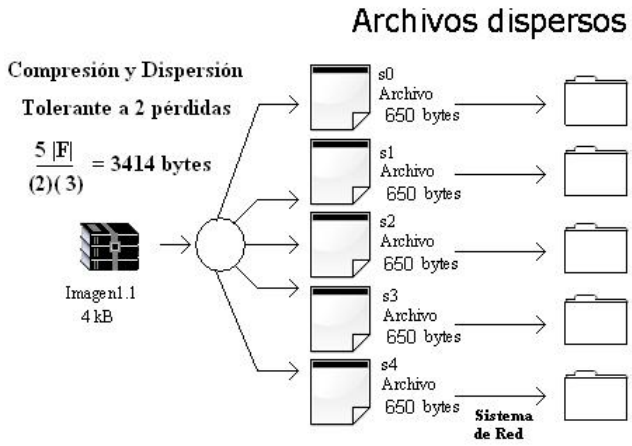


Figura 5.7: *Espacio de almacenamiento requerido en disco para 10 imágenes mri(imágenes de resonancia magnética) monocromáticas de 4k bytes*

Los espacios de almacenamiento requeridos por cada esquema se comparan tanto en el cuadro 5.6 como en la gráfica de la figura 5.7.

Tanto en espacio de almacenamiento requerido, en redundancia agregada así como en seguridad de la información, la propuesta de esta tesis es mejor.

Imagen	**Esquema** $(5 * F/2)$	**Esquema** $(3 * F/2)$	**Esquema propuesto** $(5 * F/2 * 3)$
mri8bpp.1	9.51kb	5.70kb	3.17kb
mri8bpp.2	9.50kb	5.70kb	3.16kb
mri8bpp.3	9.56kb	5.73kb	3.18kb
mri8bpp.4	9.68kb	5.80kb	3.22kb
mri8bpp.5	9.73kb	5.83kb	3.24kb
mri8bpp.6	9.75kb	5.85kb	3.25kb
mri8bpp.7	9.87kb	5.92kb	3.29kb
mri8bpp.8	9.78kb	5.87kb	3.26kb
mri8bpp.9	9.85kb	5.91kb	3.29kb
mri8bpp.10	10.02kb	6.01kb	3.35kb

Cuadro 5.6: *Comparación de los espacios de almacenamiento requeridos entre los diferentes esquemas.*

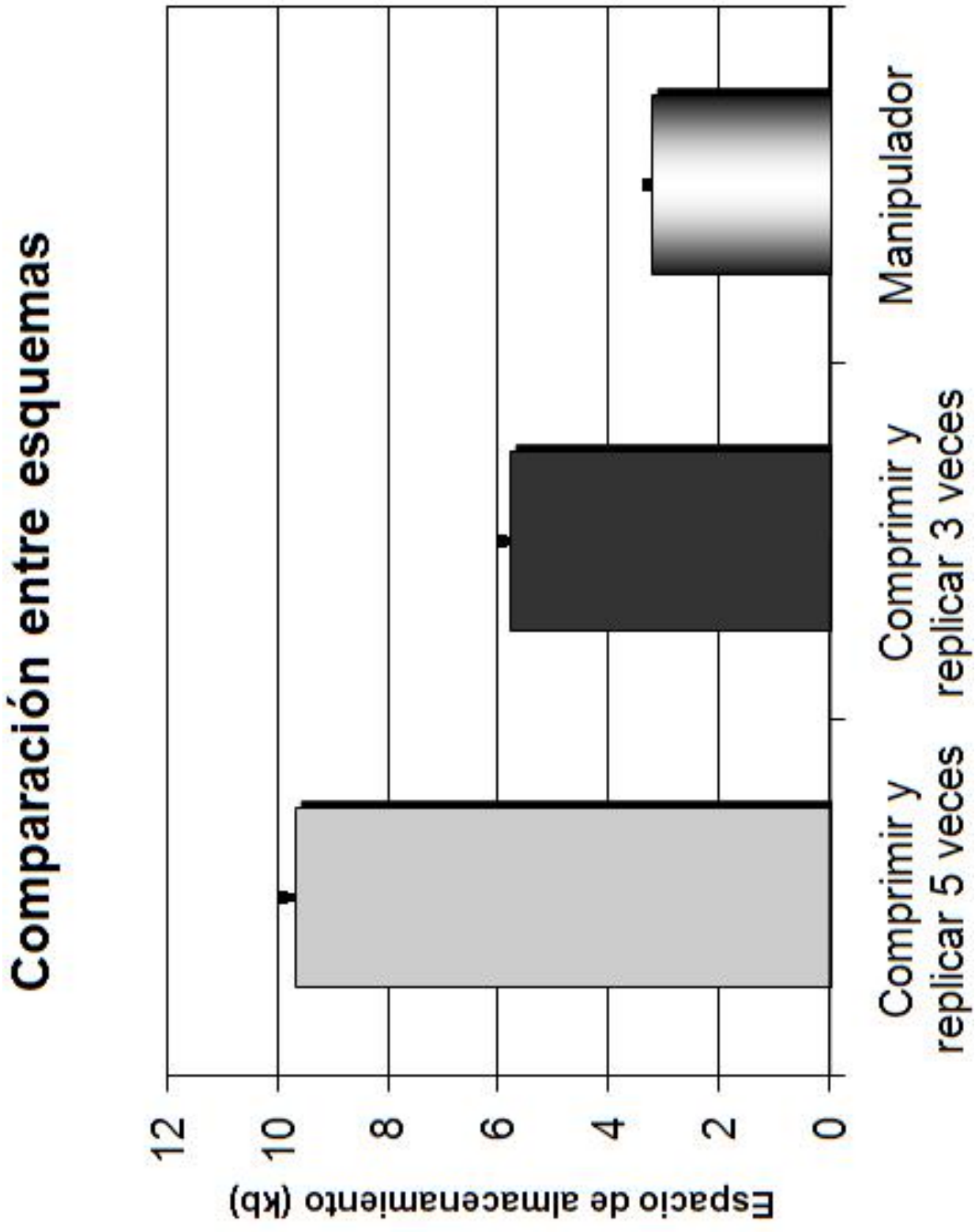


Figura 5.8: *Espacio de almacenamiento requerido en disco para 10 imágenes mri(imágenes de resonancia magnética) monocromáticas de 4k bytes*

5.1.5. Evaluación del Hardware

Para evaluar esta parte se utilizó el entorno de desarrollo ISE 8.0, el cual genera un reporte del tiempo de propagación del componente sintetizado durante el proceso de síntesis .

En la tabla 5.7 se muestra el tiempo de propagación de los componentes de más alta jerarquía descritos en hardware.

Componente	Tiempo de propagación
Unidad aritmética lógica	48.683ns
Síntesis wavelet	6.280ns
Análisis wavelet	6.376ns
Dispersor	9.533ns
Constructor	46.754ns

Cuadro 5.7: *Tiempo de ejecución de los componentes descritos en el FPGA. Estos fueron determinados con la herramienta ISE 8.*

Como los componentes fueron descritos combinacionalmente el tiempo de propagación es el mismo que el tiempo de ejecución.

Dado que el esquema de almacenamiento propuesto puede ser programado en un lenguaje de alto nivel que puede ser ejecutado sobre una computadora de escritorio, se evaluó el tiempo que le toma a una Work Station con procesador XEON dispersar 3 bytes; para así poder realizar la comparación estadística con el componente dispersor descrito en la sección 4.4.

El programa que se evaluó fue elaborado en lenguaje C++. Para determinar el tiempo de ejecución se utilizaron instrucciones de la librería time.h propias de este lenguaje. El programa se pueder consultar en el apéndice B.

Sin embargo, debido a la poca precisión de las instrucciones utilizadas para medir el tiempo, se tuvo que determinar indirectamente el tiempo tomado para dispersar 3 bytes:

$$Tiempo\ tomado\ para\ dispersar\ 3\ bytes\ = \frac{Tiempo\ tomado\ para\ dispersar\ 300000\ bytes}{100000}$$

Debido a que la Work Station multiplexa los procesos que se están ejecutando se decidió correr 100 veces el programa y generar un promedio de tiempo. Los tiempos de ejecución se pueden observar en la figura 5.9.

En promedio le toma 93130.0 ns a la Work Station dispersar 3 bytes.

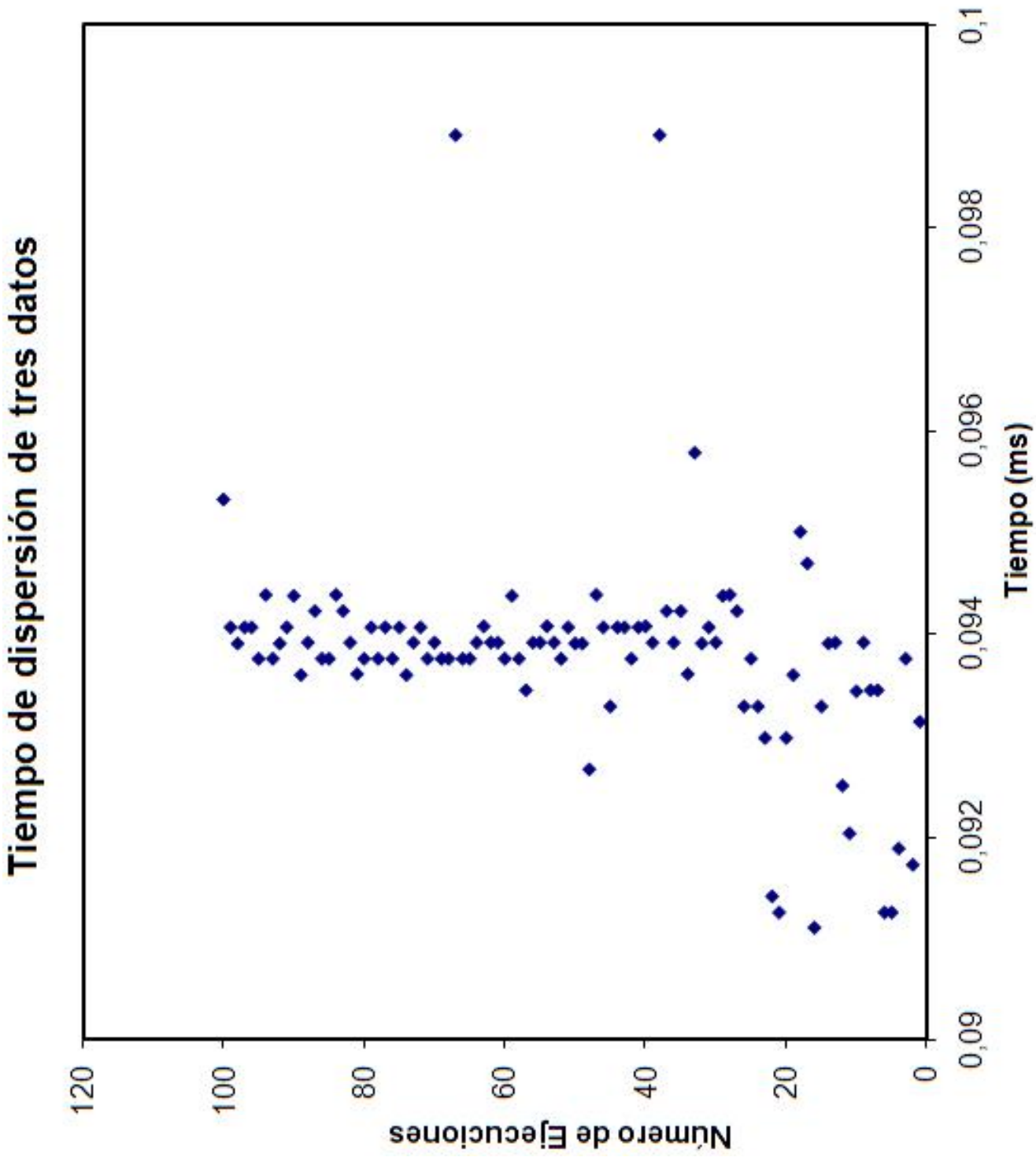


Figura 5.9: *La gráfica muestra el tiempo de ejecución que tarda la computadora en dispersar tres bytes(datos). Cada punto representa el número de ejecución y el tiempo que tardó cada una.*

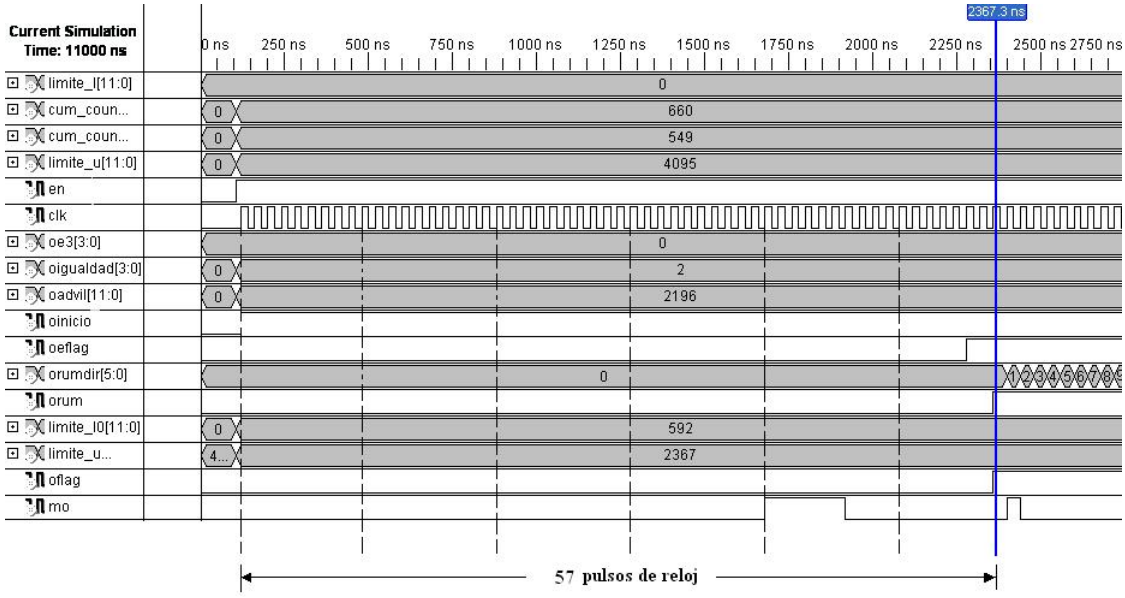


Figura 5.10: *La figura muestra el número de pulsos de reloj que tarda el componente Compresor en procesar los datos de entrada tomados de la figura 4.48.*

Dado que el componente Compresor está formado por una parte combinacional que realiza los cálculos y otra secuencial que se encarga del control y generación de la etiqueta (ver figura 4.47), no es posible determinar el tiempo de ejecución con la misma metodología, por lo que se determinó estadísticamente.

Se variaron aleatoriamente los cuatro parámetros de entrada y se determinó el número de pulsos que tardó el componente en terminar de procesar cada combinación de los datos de entrada. El análisis se realizó de esta forma para que el tiempo de ejecución fuera responsabilidad del reloj maestro que gobierna el componente Compresor. En las figuras 5.10, 5.11 y 5.12 se puede observar el número de pulsos necesarios para el procesamiento de los datos tomados de la figuras 4.48, 4.49 y 4.50 respectivamente.

Debido a que los cálculos realizados por la parte combinacional de este componente se ejecutan paralelamente con la máquina de control, la cual es secuencial, el tiempo de ejecución de la parte combinacional se puede despreciar, ya que el tiempo de ejecución de este componente es dominado por la parte secuencial.

Finalmente, para poder determinar el tiempo de ejecución medio es necesario ejecutar este componente **n** veces, cada vez con diferentes combinaciones de datos de entrada. En la figura 5.13 se representa el comportamiento del componente con **n=22**. En promedio, el componente secuencial tarda 73 ± 28 pulsos de reloj. Este tiempo puede cambiar si **n** aumenta. Sin embargo, dada la arquitectura del componente Compresor, se sabe que en el peor escenario, es decir, cuando los límites son iguales, el número máximo de pulsos de reloj necesarios para procesarlos es de 128.

Matemáticamente se demostró (sección 6.1.2) que el proceso de compresión y su pro-

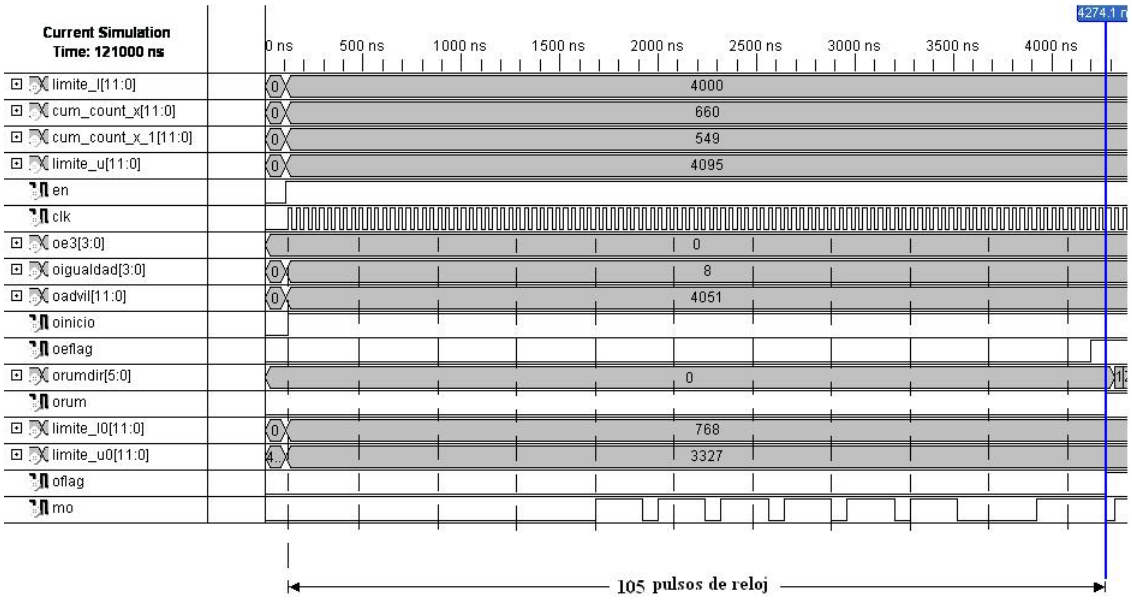


Figura 5.11: *La figura muestra el número de ciclos de reloj que tarda el componente Compresor en procesar los datos de entrada tomados de la figura 4.49.*

ceso inverso no son algoritmos posibles de implementar en el campo finito F_2^8; por lo cual el proceso inverso a la compresión no se realizo en hardware.

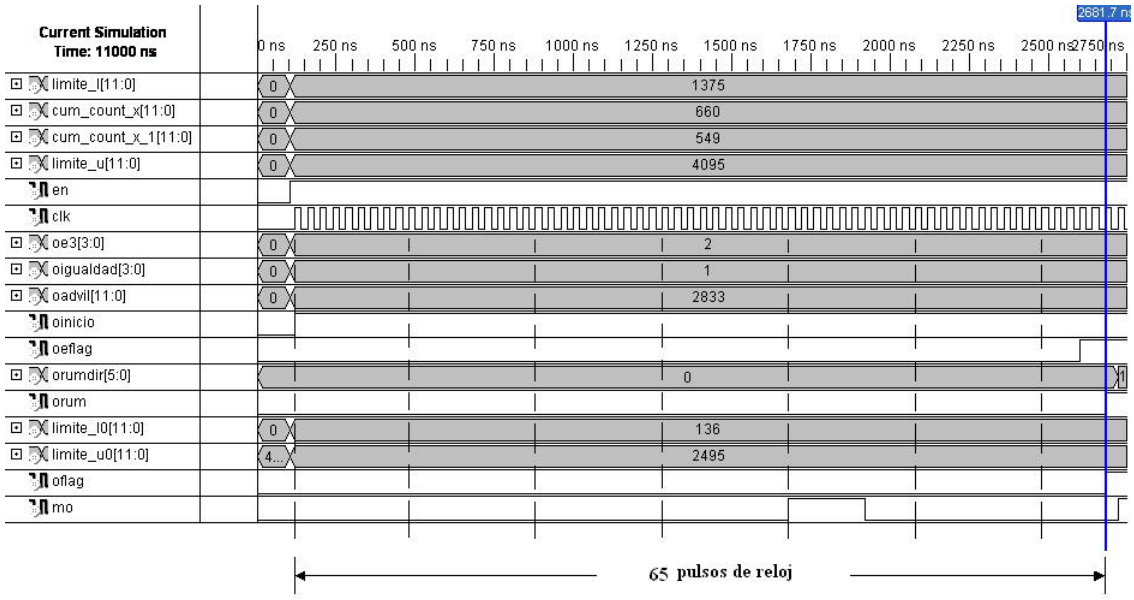


Figura 5.12: *La figura muestra el número de ciclos de reloj que tarda el componente Compresor en procesar los datos de entrada tomados de la figura 4.50.*

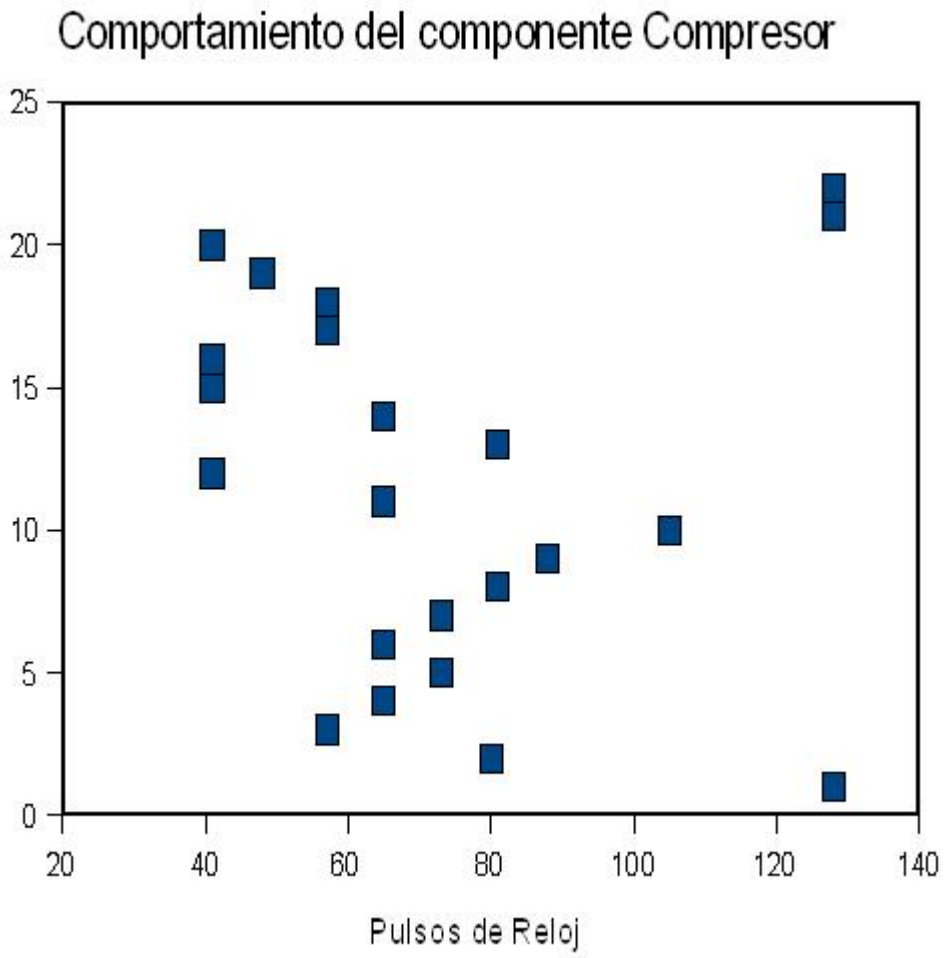


Figura 5.13: *La figura muestra el numero de ciclos de reloj que tarda el componente compresor en procesar dos palabras de control.*

Capítulo 6
Conclusiones

El empleo del campo finito F_{2^8} para la manipulación de imágenes médicas no solo permite procesarlas sin pérdida debido al redondeo o truncamiento de los datos, sino que asegura que cualquier operación sobre él será invertible.

La propuesta de almacenamiento de imágenes basada en operaciones sobre este campo mejora el esquema tradicional de almacenamiento seguido hasta ahora en los hospitales, en donde la imagen se comprime y respalda en una o varias máquinas dependiendo de la importancia del estudio, generando redundancia en los datos.

El Manipulador de Imágenes disminuye la redundancia de los datos al transformarlos, comprimirlos y dispersarlos en diferentes sistemas de almacenamiento (computadoras, cluster, CDs, etc...) existentes en el hospital, logrando un almacenamiento distribuido de los datos y un mejor aprovechamiento de los recursos disponibles. También garantiza la integridad de la información, ya que el esquema desarrollado tolera dos fallas y los datos son encriptados mediante una transformación lineal sobre el campo finito.

Entre las ventajas del Manipulador de Imágenes se pueden subrayar:

- Rapidez de procesamiento. El dispositivo fue realizado en hardware reconfigurable, en el cual se describió una estructura mayoritariamente combinacional, adecuada a los algoritmos seleccionados para este procesamiento.

- Alta Conectividad. Al contar con una interfaz plug and play, el dispositivo puede ser instalado en cualquier máquina, convirtiendo a la misma en servidor y a las restantes en esclavos. Durante el proceso de diseño se utilizaron diferentes interfaces, entre las cuales están los protocolos USB 1.0 con el microcontrolador PIC16C765 (Microchip), USB Full Speed con el microcontrolador PIC18F4550 (Microchip), y USB 2.0 con el microcontrolador CY7C68013A de Cypress, siendo éste último el más adecuado para la comunicación PC-Manipulador.

- Procesamiento sin pérdida. El empleo de aritmética sobre el campo finito impide la generación de pérdidas por redondeo, lo que permite recuperar la información íntegramente.

- Recuperación veloz. El esquema de almacenamiento distribuido hace que la capacidad para recuperar la información sea más rápida que el esquema tradicional, ya que

la información está depositada en diferentes lugares, eliminado el cuello de botella generado al tener los datos almacenados en un solo sitio.

Sin duda, manipular imágenes sobre el campo F_{2^8} resuelve el problema de swelling (ensanchamiento de los datos) provocado por la manipulación de imágenes en aritmética sobre anillos de carcterística cero.

6.1. Conclusiones particulares

6.1.1. Wavelets

Los filtros de descomposición wavelet propuestos se pueden emplear en campo finito F_{2^8}, gracias a que este campo genera un homomorfismo aditivo con el anillo diádico R módulo $Z/256Z$, sobre el cual fueron diseñados los filtros seleccionados.

$$\phi(a+b) = \phi(a) + \phi(b), \qquad para\ \ a, b\ \epsilon R\ \ cualquiera$$

donde ϕ es una aplicación del anillo $Z/256Z$, es decir, el $+$ que aparece en el primer miembro de la ecuación pertenece a la operación sobre el campo, mientras que el $+$ que aparece en el segundo miembro de la ecuación pertenece a la operación sobre el anillo.

6.1.2. Compresión

No es posible implementar el algoritmo de codificación aritmética empleando aritmética sobre el campo finito porque el algoritmo de codificación aritmética está fuertemente basado en la densidad de orden existente entre los números racionales. Esta propiedad establece que, si $\alpha < \beta$, siempre existirá otro número racional γ tal que $\alpha < \gamma < \beta$. En otras palabras, si

$$\alpha = \frac{a}{b}\ \ y\ \ \beta = \frac{c}{d},\ \ con\ \ b\ \ y\ \ d\ \ positivos,\ \ basta\ \ con\ \ tomar\ \ \gamma = \frac{a+c}{b+d}$$

por tanto, entre dos números racionales existen infinitos racionales distintos, lo que nos permite subdividir un intervalo original repetidamente y obtener intervalos que serán más pequeños que el inicial.

En el campo finito F_{2^8} no existe tal densidad de orden, puesto que el campo contiene un número finito y conocido de elementos, entre los cuales no existen otros elementos

$$\mathbb{Z}_p = \mathbb{Z}/p\mathbb{Z} = \{\overline{0}, \overline{1}, \overline{2}, \cdots, \overline{p-1}\}$$

Si planteamos que:

$$\overline{0} < \overline{1} <, ..., < \overline{p-1}$$

y suponemos que:

$$\overline{p-2} < \overline{p-1}$$

si sumamos un elemento a la igualdad, ésta se debe conservar

$$\overline{p-2}+1 < \overline{p-1}+1$$

que es igual a escribir

$$\overline{p-1} < \overline{0}$$

porque en el campo finito $p = 0$.

Como resultado se obtiene una contradicción, lo que demuestra que no es posible implementar el codificador aritmético empleando la aritmética de este campo finito, al menos no bajo el mismo principio de funcionamiento.

6.1.3. Algoritmo Dispersor de Información (IDA)

El empleo de este algoritmo potencializó tanto la reducción del espacio de almacenamiento requerido como la integridad en el respaldo de información valiosa para el hospital. El almacenamiento distribuido reduce la redundancia y es robusto por ser tolerante a dos fallas.

6.1.4. Diseño del Manipulador

La estructura planteada de forma combinacional de la Unidad Aritmética para campo F_{2^8} logró un mejor desempeño en tiempo de procesamiento con respecto a su similar descrita con máquinas de estado. Esto fue posible gracias a la descripción de las tablas de logaritmos y antilogaritmos como funciones canónicas, lo que mejoró notablemente el tiempo de procesamiento de operaciones como la multiplicación y la división, que son dos de las operaciones que consumen más tiempo de procesamiento en los esquemas basados en máquinas de estados.

El éxito en la implementación de la transformada wavelet de una forma combinacional llevó a la conclusión de que, al ser posible describir uno de los filtros de descomposición $(1 - z^{-1})$ como un arreglo de compuertas or-exclusiva, existe una relación entre los filtros digitales y el álgebra de Reed-Muller que sería interesante explorar a fondo.

El compresor fue planteado como una combinación de estructuras de diferente naturaleza: una combinacional, encargada de realizar los cálculos de acotamiento, y la otra secuencial, responsable de generar la etiqueta de compresión. La estructura final resultante es estable y mejora los tiempos de codificación de los esquemas tradicionales, en los cuales todas las operaciones son realizadas por máquinas de estados.

La descripción del algoritmo IDA como un sistema combinacional mejoró el tiempo de dispersion y reconstrucción de la imagen, de tal forma que la duración de dichas operaciones sólo depende del tiempo de propagación de los datos.

6.2. Limitaciones

La unidad aritmética para F_{2^8} pudo ser descrita de una forma combinacional gracias a que el número de elementos del campo seleccionado son 256 y se conoce el polinomio primitivo. Sin embargo, esta estructura limita a 256 el rango de posibles valores de los pixeles de las imágenes, lo cual es muy inferior a la calidad de imágenes generadas por los equipos de imagenología médica actuales, algunos de los cuales manejan imágenes a color. Es posible mapear estas imágenes sobre campos finitos de mayor orden, pero sería muy complicado describir una estructura combinacional para las operaciones aritméticas sobre estos campos, ya que el número de elementos sería demasiado grande como para que programas computacionales generaran las ecuaciones canónicas correspondientes. Además, se debe tomar en cuenta que, aunque se sabe que para cada campo finito existe un polinomio primitivo, no en todos es conocido.

Aunque la descripción de la transformada wavelet como un sistema combinacional resulta en excelentes tiempos de procesamiento, es necesario que todos los datos a procesar se encuentren almacenados en un arreglo disponible en el FPGA, lo que resulta en un consumo masivo del área del chip. La limitación principal asociada a tener una estructura combinacional en este proyecto es que sólo se pueden descomponer imágenes de 4kb.

Esta limitación se comparte con el compresor, ya que, al realizar combinacionalmente algunas de las operaciones de compresión, el tamaño de las imágenes es fijo, sólo se pueden comprimir imagenes de 4kb. Si se desea comprimir imágenes con un tamaño diferente, es necesario cambiar las estructuras.

6.3. Perspectivas

El presente proyecto concluyó con la generación de un manipulador de imágenes funcional, que puede ser perfeccionado.

La elección del campo finito F_{2^8} fue acertada como una primera aproximación a la manipulacion de imágenes. Es claro que, una vez explorada la estabilidad de los campos finitos, el siguiente paso es migrar a campos finitos más grandes, en los que se puedan manipular imágenes de mayor número de bits por pixel o a color. Para ello se propone replantear la estructura de la unidad aritmética de campo finito utilizando una máquina de estados, que permita la manipulación de imágenes con más de 256 posibles valores en sus pixeles.

Se plantea conseguir la flexibilización de tanto la transformada wavelet como el algoritmo de compresión mediante la descripción de estructuras secuenciales para cada uno de ellos, de tal forma que se puedan procesar imágenes de diferentes tamaños.

Otro planteamiento a futuro es implementar un sistema encargado de distribuir y recuperar los archivos dispersos en la red del hospital. Experimentos realizados sobre esta comunicación sugieren una comunicación FTP (File Transfer Protocol).

Una de las propuestas a futuro es cambiar la interfaz USB 2.0 por una interfaz interna

como el bus PCI, con lo cual se puede alcanzar mayor velocidad en la comunicación PC-Manipulador.

Capítulo 7
Apéndice A

Imagen Digital

Una imagen digital es una función bidimensional de intensidad de luz $f(x, y)$ que se ha discretizado tanto en las coordenadas espaciales como en el brillo [23].

Matemáticamente hablando se puede describir una imagen como una matriz **N** $\times$ **M**, donde cada elemento es una cantidad discreta:

$$\mathbf{f}(\mathbf{x}, \mathbf{y}) \approx \begin{pmatrix} f(0,0) & f(0,1) & \dots & f(0,M-1) \\ f(1,0) & f(1,1) & \dots & f(1,M-1) \\ \vdots & \vdots & \ddots & \vdots \\ f(N-1,0) & f(N-1,1) & \dots & f(N-1,M-1) \end{pmatrix}$$

donde x y y representan las coordenadas espaciales y el valor de i en un punto cualquiera (x, y) es proporcional al brillo (o nivel de gris) de la imagen en ese punto.

Una imagen digital puede considerarse como una matriz cuyos índices de fila y columna identifican un punto de la imagen y el valor del correspondiente elemento de la matriz indica el nivel de gris en ese punto al cual se le suele llamar *pixel* (abreviatura de picture element o elemento de imagen).

Algoritmo que fue utilizado para calcular la inversa de la matriz **D** recuperada de los dispersos. Las operaciones fueron realizadas sobre el campo finito f_{2^8}, es por eso que utilizamos una función llamada ALU.

```
void InvierteM()
{
 int i,j;
 unsigned int d=0;

 for (j=0; j<3; j++)
   d=Alu('+',d, Alu('*',a[0][j],
   Alu('+',
   Alu('*',a[1][(j+1)%3],a[2][(j+2)%3]),
   Alu('*',a[1][(j+2)%3],a[2][(j+1)%3]))));
```

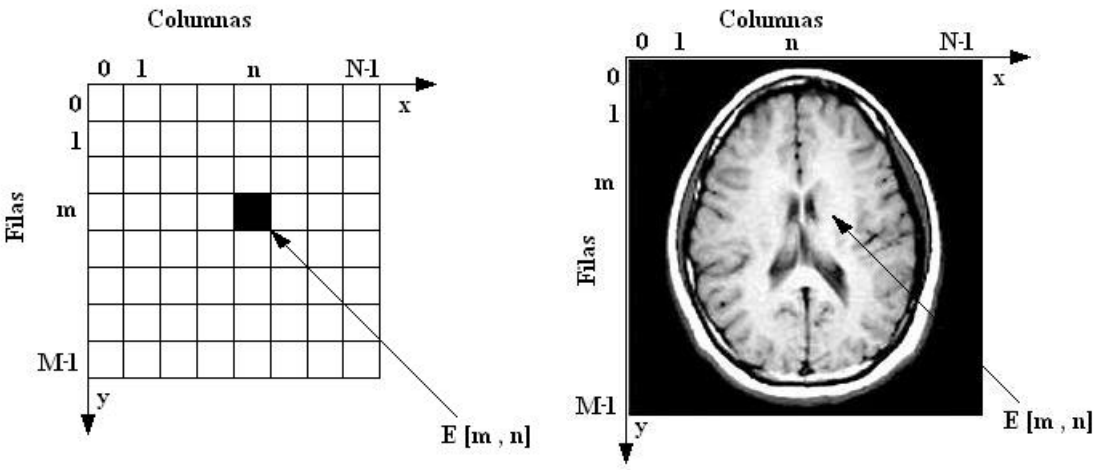


Figura 7.1: Imagen 2-D

```
   for (i=0; i<3; i++)
      for (j=0; j<3; j++)
       g[j][i]=Alu('/', Alu('+',
        Alu('*',a[(i+1)%3][(j+1)%3],a[(i+2)%3][(j+2)%3]),
          Alu('*',a[(i+1)%3][(j+2)%3],a[(i+2)%3][(j+1)%3])
             z ),
               d
          );
}
```

El algoritmo propuesto en el cual se basó la descripción en VHDL de la multiplicación fue programada en C++

```
Int Logaritmo_A;
Int Logaritmo_B;

Logaritmo_A = Log(Op_A);
Logaritmo_B = Log(Op_B);

  if(((Op_A) == 0)||((Op_B) == 0))
  {
       Multiplicacion = 0;
       return 0;
  }
  else
  {
       Multiplicacion  =  (Logaritmo_A)+(Logaritmo_B);
```

```
        if(Multiplicacion > 255)
        {
             Multiplicacion  = (Multiplicacion - 255);
             return  AntLog(Multiplicacion);
        }

        if(Multiplicacion  ==  255)
        {
             Multiplicacion  =  0;
             return AntLog(Multiplicacion);
        }

        if(Multiplicacion < 255)
        {
             return AntLog(Multiplicacion);
        }
  }
```

El algoritmo propuesto en el cual se basó la descripción en VHDL de la división fue programada en C++

```
Int Logaritmo_A;
Int Logaritmo_B;

Logaritmo_A = Log(Op_A);
Logaritmo_B = Log(Op_B);

  if(((Op_A) == 0)||((Op_B) == 0))
  {
        Division=0;
        return 0;
  }
  else
  {
        Logaritmo_B  =       255  -       Logaritmo_B;
        Division=(Logaritmo_A)+(Logaritmo_B);

        if(Division > 255)
        {
             Division  = (Division - 255);
             return  AntLog(Division);
        }

        if(Division  ==  255)
        {
             Division  =  0;
```

```
            return AntLog(Division);
        }

        if(Division < 255)
        {
            return AntLog(Division);
        }
    }
```

Algoritmo utilizado para la compresión. Es llamado codificador aritmetico para numeros entero Inicializa l y u
obten el simbolo

$$l \leftrightarrow l + \lfloor \frac{(u - l + 1) \times Cum_Count(x - 1)}{TotalCount} \rfloor$$

$$u \leftrightarrow l + \lfloor \frac{(u - l + 1) \times Cum_Count(x)}{TotalCount} \rfloor - 1$$

While(El MSB de u y l sean iguales a b o se cumpla la condición E_3)
if(El MSB de u y l son iguales a b)
{
Envia b
Desplaza el limite l una posición ala izquierda y agrega un 0 en el LSB
Desplaza el limite u una posición ala izquierda y agrega un 1 en el LSB
While(Scale3 > 0)
{
envia el complemento de b
decrementa Scale3
}
}

if(E_3 se cumple)
{
Desplaza el limite l una posición ala izquierda y agrega un 0 en el LSB
Desplaza el limite u una posición ala izquierda y agrega un 1 en el LSB
complementa el MSB de l y u
incrementa Scale3
}
Algoritmo utilizado para la descompresión. Es llamado decodificador aritmetico para numeros entero:
Inicializa l y u
Lee los primeros m bits de la cadena recibida y guardala en t
$k = 0$

$$while(\lfloor \frac{(t - l + 1) \times TotalCount - 1}{u - l + 1} \rfloor \ \geq \ Cum\_Count(k))$$

$k \leftrightarrow k+1$
decodifica el simbolo x

$$l \leftrightarrow l + \lfloor \frac{(u-l+1) \times Cum\_Count(x-1)}{TotalCount} \rfloor$$

$$u \leftrightarrow l + \lfloor \frac{(u-l+1) \times Cum\_Count(x)}{TotalCount} \rfloor - 1$$

While(El MSB de u y l sean iguales a b o se cumpla la condición E_3)
if(El MSB de u y l son iguales a b)
{
Envia b
Desplaza el limite l una posición ala izquierda y agrega un 0 en el LSB
Desplaza el limite u una posición ala izquierda y agrega un 1 en el LSB
Desplaza t una posicioón ala izquierda y lee el proximo bit de la cadena recibida y agregalo como LSB
}

if(E_3 se cumple)
{
Desplaza el limite l una posición ala izquierda y agrega un 0 en el LSB
Desplaza el limite u una posición ala izquierda y agrega un 1 en el LSB
Desplaza t una posición ala izquierda y lee el proximo bit de la cadena recibida y agregalo como LSB
complementa el MSB de l,u y t
incrementa Scale3
}

Algoritmo realizado en Matlab para transformar las imagenes MRI de 16 bpp a 8 bpp.

```
for i = 1:93
  % lee el archivo original de 16 bpp
  eval(sprintf('id = fopen(''quarter.%u'',''rb'')',i));
  x = fread(id,[64,64],'uint16');
  fclose(id);
  % convierte linealmente a 8 bpp y despliega
  x = uint8(fix(255*x/65535));
  imagesc(x); pause;
  % escribe el archivo convertido a 8 bbb
  eval(sprintf('id = fopen(''mri8bpp.%u'',''wb'')',i));
  fwrite(id,x);
  fclose(id);
end
```

Capítulo 8
Apéndice B

Campo Finito
Elemento Logaritmo

```
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Logaritmo is
    Port ( A  : in  STD_LOGIC_VECTOR(7 downto 0);
           LOGA  : out STD_LOGIC_VECTOR(7 downto 0));
end Logaritmo;

architecture Behavioral of Logaritmo is
signal X0,X1,X2,X3,X4,X5,X6,X7  : STD_LOGIC;
signal Z0,Z1,Z2,Z3,Z4,Z5,Z6,Z7  : STD_LOGIC;

begin

X0 <= A(0);
X1 <= A(1);
X2 <= A(2);
X3 <= A(3);
X4 <= A(4);
X5 <= A(5);
X6 <= A(6);
X7 <= A(7);

LOGA(0) <= Z0;
LOGA(1) <= Z1;
LOGA(2) <= Z2;
LOGA(3) <= Z3;
LOGA(4) <= Z4;
LOGA(5) <= Z5;
LOGA(6) <= Z6;
LOGA(7) <= Z7;
```

Z7 <= **((not X6) and (not X5) and (not X4) and (not X3) and (X2) and (not X1)**

```
and (X0 ) ) or ( ( not X6) and (not X5) and (not X3) and (X2) and (X1) and (not
X0 ) ) or ( ( not X7) and (not X6) and (not X5) and (not X4) and (X3) and (not X2)
and (X1 ) ) or ( ( not X7) and (not X6) and (not X4) and (X3) and (X2) and (not
X1) and (not X0 ) ) or ( ( not X7) and (not X6) and (not X5) and (X4) and (not X3)
and (X2) and (not X0 ) ) or ( (not X7) and (X5) and (not X3) and (not X2) and (X1)
and (not X0 ) ) or ( ( not X7) and (not X6) and (X5) and (not X4) and (X3) and (not
X1 ) ) or ( ( X6) and (not X4) and (not X3) and (X2) and (not X1) and (not X0 )
) or (( not X7) and (X6) and (not X5) and (not X3) and (X1) and (X0 ) ) or ( ( not
X7) and (X6) and (not X5) and (X4) and (not X2) and (not X0 ) ) or ( ( not X7) and
(X5) and (X3) and (X2) and (not X1) and (X0 ) ) or ((X7) and (not X5) and (not X4)
and (X3) and (not X2) and (not X1) and (not X0 ) ) or ( ( X7) and (not X6) and (not
X4) and (X2) and (X1) and (not X0 ) ) or ( ( X7) and (not X6) and (not X5) and (X4)
and (not X2) and (X0 ) ) or ( ( X7) and (X6) and (not X5) and (not X4) and (not
X3) and (not X2) and (not X0 )) or (( X6) and (X4) and (X3) and (not X2) and (X1)
and (not X0 ) ) or ((X7) and (X6) and (X5) and (X4) and (not X3) and (not X1 ) )
or ( ( not X7) and (not X6) and (not X5) and (X3) and (X2) and (X1) and (X0 ) )
or (( not X7) and (not X6) and (X4) and (not X2) and (not X1) and (X0 ) ) or ( (
not X7) and (not X6) and (not X5) and (X4) and (X3) and (not X2) and (not X1 ) )
or ((not X7) and (not X4) and (X3) and (not X2) and (X1) and (X0 )) or ((not X7)
and (X6) and (X5) and (X4) and (X1) and (X0 )) or ((X7) and (not X6) and (not X5)
and (not X4) and (not X3) and (X0 ) ) or ( ( X7) and (not X6) and (X3) and (X2)
and (not X1) and (X0 ) ) or ( ( X7) and (not X5) and (not X4) and (not X3) and (X2)
and (not X1) and (X0 ) ) or ( ( X7) and (X6) and (X5) and (X2) and (X1) and (not
X0 ) ) or ( ( not X7) and (not X5) and (not X4) and (not X2) and (X1) and (X0 )
) or ( ( not X7) and (X6) and (X4) and (not X2) and (X1 )) or ((not X7) and (X6)
and (X4) and (X3) and (not X1) and (not X0 ) ) or ((not X6) and (X5) and (not X4)
and (X3) and (X2) and (not X0 ) ) or ( ( X7) and (not X6) and (X5) and (X4) and
(X2) and (X1) and (X0 ) ) or ( ( X7) and (X6) and (not X5) and (not X4) and (X3)
and (not X2) and (X0 ) ) or ( ( X7) and (X6) and (X5) and (not X4) and (not X3)
and (X1) and (X0 ) ) or ( (X7) and (X6) and (X5) and (X4) and (X3) and (not X2 )
) or ( ( not X7) and (not X6) and (X5) and (X3) and (X2) and (not X0 ) ) or ( (
X6) and (X5) and (not X4) and (X2) and (X1) and (not X0 ) ) or ( ( X6) and (X5)
and (not X3) and (X2) and (not X1) and (not X0 )) or ( ( X7) and (X6) and (not X4)
and (X3) and (not X2) and (not X1 )) or ( ( not X7) and (not X6) and (not X5) and
(X4) and (X2) and (X1 )) or ((X5) and (X4) and (X3) and (not X2) and (X1) and (not
X0 ) ) or ( ( not X6) and (X5) and (X4) and (not X3) and (X1) and (X0 ) ) or ( (
not X5) and (X4) and (not X3) and (X2) and (X1) and (not X0 ) ) or ( ( X7) and (X5)
and (not X4) and (not X3) and (not X2) and (X0 ) ) or ( ( X7) and (X5) and (not
X3) and (X2) and (not X0)) or ((not X7) and (X6) and (X5) and (not X4) and (not
X3) and (not X2) and (not X1 )) or ((X7) and (not X6) and (X5) and (not X3) and
(not X1) and (not X0 )) or (( not X7) and (not X6) and (X5) and (X4) and (not X3)
and (not X2 )) or (( X7) and (X5) and (X4) and (X3) and (not X2) and (not X1 ))
or (( X6) and (not X5) and (X4) and (X3) and (X2) and (not X1)) or ((not X6) and
(X4) and (X3) and (not X1) and (X0)) or (( X7) and (X6) and (not X5) and (not X4)
and (not X1) and (not X0));

   --Input Cost <= 367 Gate Cost <= 52

   Z6 <= ((not X6) and (not X5) and (X4) and (X3) and (not X2) and (not X1 ) ) or
( ( not X6) and (X5) and (not X4) and (X3) and (not X2 ) ) or ( ( not X7) and (not
X6) and (X5) and (X4) and (X3) and (X1) and (X0 ) ) or ( ( not X7) and (X6) and
(not X5) and (not X4) and (not X2) and (X0 ) ) or ( ( not X7) and (X6) and (not
X5) and (X4) and (not X3) and (not X0 ) ) or ( ( X6) and (X5) and (not X4) and (not
```

```
X3) and (not X1) and (not X0)) or ((not X7) and (X6) and (X5) and (not X4) and (X2)
and (X1) and (X0 ) ) or ( ( not X7) and (X6) and (X5) and (X4) and (X1) and (not
X0 ) ) or ( ( X7) and (not X6) and (not X5) and (not X4) and (X3) and (X1) and (X0
) ) or ( ( not X6) and (X5) and (not X4) and (not X3) and (X2) and (X0 ) ) or (
( X7) and (X6) and (not X5) and (not X4) and (not X3) and (X2) and (X1 ) ) or (
(X7) and (X6) and (not X5) and (not X4) and (not X2) and (not X1) and (not X0))
or (( X7) and (X6) and (not X4) and (X3) and (X2) and (not X1) and (X0 )) or ((
not X5) and (X4) and (not X3) and (X2) and (not X1) and (X0 ) ) or ( ( X7) and (X6)
and (not X5) and (X4) and (X3) and (not X2) and (X1) and (X0 ) ) or ( ( X7) and
(X6) and (not X5) and (X3) and (X2) and (X1) and (not X0 ) ) or ( (X7) and (X5)
and (not X4) and (X3) and (not X2) and (X1 ) ) or ( ( X7) and (X6) and (X5) and
(X4) and (X3) and (X2) and (X0 ) ) or ( ( not X6) and (not X5) and (not X3) and
(X2) and (not X1) and (X0 ) ) or ( ( not X7) and (not X5) and (not X4) and (X3)
and (not X2) and (X1) and (not X0 ) ) or ( ( not X6) and (not X5) and (X4) and (not
X3) and (X2) and (not X1 ) ) or ( ( not X7) and (X6) and (not X5) and (not X2) and
(X1) and (not X0 ) ) or ( (not X7) and (X6) and (not X4) and (not X3) and (not X2)
and (X1 ) ) or ( ( not X7) and (X6) and (X4) and (not X3) and (X2) and (X0 ) ) or
( ( not X7) and (X5) and (X4) and (X3) and (not X2) and (X0 ) ) or ( ( X7) and (not
X6) and (X3) and (not X2) and (not X1) and (X0 ) ) or ( ( X7) and (not X6) and (not
X5) and (X4) and (X2) and (not X1 ) ) or ( ( X7) and (not X6) and (X5) and (not
X4) and (not X1) and (not X0 ) ) or ( (X7) and (X4) and (not X3) and (not X2) and
(X1) and (not X0 ) ) or ( ( X7) and (X6) and (X5) and (not X3) and (not X2) and
(not X1) and (X0 ) ) or ( ( X7) and (X5) and (not X4) and (X3) and (X2) and (not
X0 ) ) or ( ( X7) and (X5) and (X4) and (not X2) and (X1) and (not X0 ) ) or ( (
X7) and (X6) and (X5) and (X2) and (not X1) and (not X0 ) ) or ( ( not X7) and (X6)
and (X3) and (not X2) and (not X1) and (X0 ) ) or ( (X7) and (not X6) and (not X5)
and (not X3) and (X1) and (not X0 ) ) or ( ( not X6) and (X5) and (not X4) and (X2)
and (not X1) and (X0 ) ) or ( ( X7) and (not X6) and (not X3) and (X2) and (not
X1) and (not X0 ) ) or ( ( X7) and (not X6) and (X4) and (X3) and (not X2) and (not
X1 ) ) or ( ( X7) and (not X5) and (not X4) and (not X3) and (X2) and (not X0 )
) or ( ( X6) and (X5) and (X4) and (not X3) and (X1) and (not X0 ) ) or ( (not X6)
and (not X5) and (not X3) and (not X2) and (X1) and (X0 ) ) or ( ( not X7) and (not
X5) and (X4) and (not X3) and (X2) and (X0 ) ) or ( ( not X7) and (not X6) and (X5)
and (not X4) and (not X2) and (X0 ) ) or ( ( X7) and (X5) and (X4) and (not X2)
and (not X1) and (X0 ) ) or ( ( not X7) and (X6) and (X4) and (X3) and (not X1)
and (X0 ) ) or ( ( X7) and (not X6) and (X4) and (not X3) and (not X2) and (X1 ))
or ((not X7) and (X6) and (not X5) and (not X4) and (X1) and (not X0 ) ) or ( (
not X7) and (X6) and (not X5) and (X3) and (X2) and (not X1) and (not X0 ) ) or
( ( not X6) and (not X5) and (not X4) and (X3) and (X2) and (not X1) and (not X0
) ) or ( ( not X7) and (not X6) and (X5) and (X4) and (not X3) and (not X2) and
(not X0 ) ) or ( ( not X6) and (X5) and (X4) and (not X2) and (not X1) and (X0 )
) or ((not X7) and (not X6) and (X5) and (not X4) and (X2) and (X1) and (not X0
) ) or ( ( not X5) and (not X4) and (not X3) and (X2) and (X1) and (not X0));

   --Input Cost <= 386 Gate Cost <= 54

   Z5 <= ((not X7) and (not X6) and (not X5) and (not X4) and (not X3) and (X1)
and (X0 ) ) or ( ( not X7) and (not X6) and (not X5) and (not X4) and (X2) and (X1
) ) or ( ( not X7) and (not X6) and (not X5) and (X3) and (X2) and (not X0 ) ) or
( ( not X7) and (X6) and (X5) and (X4) and (X3) and (X2) and (X1) and (X0 ) ) or
( ( X7) and (not X6) and (not X3) and (X2) and (not X1) and (not X0 ) ) or ( (not
X6) and (not X5) and (not X4) and (not X3) and (X2) and (X1) and (X0 ) ) or ( (
X7) and (not X6) and (not X5) and (not X4) and (X3) and (not X1) and (X0 ) ) or
```

```
( ( X7) and (not X6) and (X4) and (X3) and (X1) and (X0 )) or ( ( X7) and (X6) and
(not X5) and (not X4) and (not X2) and (not X0 ) ) or ( ( X7) and (X6) and (not
X5) and (not X4) and (not X2) and (X1 ) ) or ( ( X6) and (not X4) and (not X3) and
(X2) and (not X1) and (X0 )) or ((X7) and (X6) and (X5) and (not X3) and (X2) and
(X0 ) ) or ( ( X7) and (X5) and (X4) and (X3) and (not X2) and (X1) and (X0 ) )
or ( ( X7) and (X6) and (X5) and (X4) and (X3) and (X2) and (not X0 ) ) or ( ( not
X7) and (X6) and (not X5) and (not X2) and (X1) and (not X0 ) ) or ( ( X7) and (not
X6) and (X5) and (not X4) and (X1) and (not X0 ) ) or ( ( X7) and (not X6) and (not
X4) and (X3) and (not X2) and (X1 ) ) or ((X7) and (not X6) and (X4) and (not X3)
and (not X2) and (X1 ) ) or ( ( not X6) and (X5) and (X4) and (X3) and (not X2)
and (not X1 ) ) or ( ( X5) and (not X4) and (not X3) and (X2) and (not X1) and (not
X0 ) ) or ( ( not X7) and (X4) and (not X3) and (not X2) and (X1) and (not X0 )
) or ( ( not X7) and (not X6) and (X4) and (X3) and (not X2) and (not X1 ) ) or
( ( not X6) and (X5) and (not X4) and (not X2) and (X1) and (not X0 ) ) or ((not
X7) and (X5) and (X4) and (not X2) and (not X1) and (not X0 ) ) or ( ( not X7) and
(X6) and (not X5) and (not X3) and (X2) and (not X1 ) ) or ( ( not X7) and (X6)
and (not X5) and (X4) and (not X3) and (X0 ) ) or (( not X7) and (X6) and (X5) and
(not X4) and (not X3) and (not X1 ) ) or ( ( X7) and (not X6) and (not X4) and (X3)
and (not X2) and (not X0 ) ) or ( ( X7) and (not X6) and (not X5) and (X4) and (not
X3) and (not X1 )) or ((not X6) and (X4) and (X3) and (X2) and (not X1) and (not
X0 ) ) or ( ( X7) and (not X5) and (X3) and (not X2) and (X1) and (not X0 ) ) or
( ( X6) and (X5) and (X4) and (not X3) and (not X2) and (not X0 ) ) or ( ( not X7)
and (not X6) and (X5) and (not X4) and (not X1) and (X0 ) ) or ( ( not X7) and (not
X5) and (X4) and (X3) and (X2) and (not X0 ) ) or ( ( not X7) and (X6) and (X5)
and (not X4) and (X3) and (not X2) and (X1 ) ) or ( (not X7) and (X4) and (X3) and
(not X2) and (not X1) and (X0 ) ) or ( ( X7) and (not X6) and (not X5) and (X3)
and (X1) and (not X0 ) ) or ( ( X7) and (X6) and (not X5) and (X4) and (not X3)
and (X2) and (not X0 ) ) or ( ( X7) and (X6) and (X5) and (not X3) and (not X1)
and (X0 ) ) or ( ( not X7) and (not X5) and (X3) and (not X2) and (not X1) and (X0
) ) or ( ( not X7) and (not X6) and (not X4) and (X3) and (X2) and (X1 ) ) or (
(not X7) and (not X6) and (X5) and (X2) and (not X1) and (X0 ) ) or ( ( not X7)
and (X6) and (X5) and (not X4) and (not X3) and (X0 ) ) or ( ( X7) and (X6) and
(not X5) and (not X4) and (X1) and (not X0 ) ) or ( ( X7) and (X6) and (not X5)
and (X4) and (X3) and (not X1) and (X0 ) ) or ( ( not X7) and (not X5) and (X4)
and (not X3) and (X2) and (X0 ) ) or ( ( not X7) and (not X4) and (not X3) and (X2)
and (X1) and (X0 ) ) or ( (X6) and (not X5) and (not X4) and (X3) and (not X2) and
(not X0 ) ) or ( ( not X7) and (not X5) and (not X4) and (X3) and (X2) and (X1 )
) or ( ( X7) and (X6) and (X5) and (X2) and (not X1) and (X0 ) ) or ( ( not X7)
and (X6) and (not X4) and (X2) and (not X1) and (X0));

   --Input Cost <= 367 Gate Cost <= 52

   Z4 <= ((not X7) and (not X6) and (not X5) and (X2) and (X1) and (X0 ) ) or (
( not X6) and (X5) and (X4) and (not X3) and (not X2) and (X1) and (not X0 ) ) or
( ( not X7) and (X6) and (not X4) and (not X3) and (not X2) and (X1) and (not X0
) ) or ( ( not X7) and (X6) and (X5) and (not X4) and (not X3) and (X2) and (not
X1 ) ) or ( ( not X7) and (X6) and (X4) and (not X3) and (X1) and (X0 ) ) or ( (
not X7) and (X5) and (X4) and (X3) and (X2) and (not X0 ) ) or ( ( X7) and (not
X6) and (not X5) and (not X4) and (not X3) and (not X2) and (X0 ) ) or ( ( X7) and
(not X6) and (X5) and (not X2) and (not X1) and (not X0 ) ) or ( ( X7) and (X5)
and (not X4) and (X2) and (X1 ) ) or ( ( X7) and (not X6) and (X5) and (X4) and
(X1) and (X0 ) ) or ( ( X7) and (not X5) and (not X4) and (not X3) and (X2) and
(not X1) and (not X0 ) ) or ( ( X7) and (X6) and (not X5) and (not X4) and (X3)
```

```
and (not X2) and (not X0 ) ) or ( ( X7) and (X6) and (X5) and (not X4) and (not
X3) and (X0 ) ) or ( ( X6) and (X5) and (X4) and (X3) and (not X2) and (not X1 )
) or ( ( X7) and (X3) and (X2) and (X1) and (X0 ) ) or ( ( not X7) and (X6) and
(not X5) and (not X4) and (X2) and (not X1) and (X0 ) ) or ( ( not X7) and (X6)
and (not X5) and (not X3) and (X2) and (X1) and (not X0 ) ) or ( ( X7) and (not
X6) and (not X4) and (X3) and (X2) and (not X1 ) ) or ( ( not X6) and (X4) and (not
X3) and (X2) and (not X1) and (not X0 ) ) or ( ( X7) and (X6) and (not X5) and (X2)
and (X1) and (X0 ) ) or ( ( X7) and (X6) and (X4) and (not X3) and (not X2) and
(X1 ) ) or ( ( X7) and (not X5) and (X4) and (not X3) and (not X1) and (X0 ) ) or
( ( X7) and (X6) and (not X5) and (X4) and (X2) and (X1 ) ) or ( ( X7) and (X6)
and (not X5) and (X3) and (X1) and (X0 ) ) or ( ( X6) and (X4) and (not X3) and
(not X2) and (not X1) and (not X0 ) ) or ( ( X7) and (X6) and (not X5) and (X3)
and (X2) and (X0 ) ) or ( ( X6) and (X5) and (not X4) and (X3) and (X1) and (X0
) ) or ( ( not X7) and (not X6) and (not X4) and (X3) and (not X2) and (X1 ) ) or
( ( not X7) and (X6) and (X5) and (X3) and (not X2) and (not X1 ) ) or ( ( not X7)
and (not X6) and (X3) and (X2) and (X1) and (not X0 ) ) or ( ( not X7) and (not
X6) and (X5) and (X3) and (not X1) and (not X0 ) ) or ( ( not X6) and (X4) and (X3)
and (X2) and (X1 ) ) or ( ( not X6) and (not X5) and (X3) and (not X2) and (X1)
and (not X0 ) ) or ( ( not X6) and (X5) and (X3) and (X2) and (not X1) and (X0 )
) or ( ( not X5) and (not X4) and (X3) and (X2) and (X0 ) ) or ( ( not X7) and (not
X6) and (X5) and (not X4) and (not X3) and (not X1) and (X0 ) ) or ( ( not X7) and
(not X5) and (X3) and (not X2) and (X1) and (not X0 ) ) or ( ( not X6) and (X5)
and (not X4) and (X3) and (not X0 ) ) or ( ( not X7) and (not X6) and (X4) and (not
X3) and (not X2) and (not X1) and (X0 ) ) or ( ( not X5) and (X4) and (X2) and (X1)
and (X0 ) ) or ( ( not X7) and (not X6) and (not X5) and (X4) and (not X3) and (X2
) ) or ( ( not X7) and (not X5) and (X4) and (X3) and (not X2) and (not X1) and
(X0 ) ) or ( ( not X7) and (not X5) and (X4) and (X2) and (not X1) and (not X0 )
) or ( ( not X7) and (X6) and (not X5) and (X4) and (not X1) and (not X0));

   --Input Cost <= 313 Gate Cost <= 45

   Z3 <= ((not X7) and (not X6) and (not X5) and (not X4) and (not X3) and (X2)
and (X0 ) ) or ( ( not X5) and (X4) and (not X3) and (not X2) and (not X1) and (X0
) ) or ( ( not X7) and (not X6) and (X3) and (not X2) and (X1) and (X0 ) ) or (
( not X7) and (not X6) and (X5) and (not X4) and (not X3) and (not X2) and (X0 )
) or ( ( not X7) and (X6) and (not X5) and (not X4) and (not X3) and (not X2) and
(X1) and (not X0 ) ) or ( ( not X7) and (not X4) and (X3) and (X2) and (X1) and
(not X0 ) ) or ( ( X6) and (not X5) and (X4) and (not X3) and (not X2) and (X0 )
) or ( ( not X7) and (X6) and (not X5) and (X4) and (X2) and (not X1 ) ) or ( (
not X7) and (X6) and (X5) and (not X3) and (not X2) and (not X1) and (not X0 ) )
or ( ( X6) and (X5) and (X2) and (X1) and (not X0 ) ) or ( ( X6) and (X5) and (not
X4) and (X3) and (X1) and (not X0 ) ) or ( ( X6) and (X5) and (not X4) and (X3)
and (X2 ) ) or ( ( X7) and (not X6) and (not X5) and (not X4) and (not X3) and (X2)
and (not X1) and (not X0 ) ) or ( ( X7) and (X6) and (not X5) and (not X2) and (not
X1 ) ) or ( ( X7) and (X6) and (not X4) and (not X1) and (X0 ) ) or ( ( not X7)
and (X6) and (not X5) and (not X4) and (X3) and (not X2) and (X0 ) ) or ( ( not
X7) and (X4) and (not X3) and (X2) and (X1) and (not X0 ) ) or ( ( X7) and (not
X6) and (not X5) and (not X4) and (X3) and (X2) and (X1 ) ) or ( ( X7) and (not
X6) and (X4) and (X2) and (not X1) and (X0 ) ) or ( ( X7) and (not X5) and (not
X3) and (not X2) and (X1) and (X0 ) ) or ( ( X7) and (X6) and (not X5) and (X3)
and (not X2) and (not X0 ) ) or ( ( X7) and (X6) and (not X5) and (X3) and (not
X1) and (not X0 ) ) or ( ( X6) and (not X5) and (X4) and (X2) and (not X1) and (not
X0 ) ) or ( ( not X7) and (X4) and (X3) and (not X2) and (not X1) and (not X0 )
```

```
) or ( ( X7) and (not X6) and (not X5) and (X4) and (not X3) and (X1) and (not X0
) ) or ( ( not X5) and (X4) and (X3) and (X2) and (not X1) and (not X0 ) ) or (
( X7) and (X5) and (not X4) and (not X3) and (X1) and (not X0 ) ) or ( ( X7) and
(not X6) and (X5) and (not X4) and (X2) and (X0 ) ) or ( ( X7) and (not X6) and
(X5) and (not X2) and (X1) and (X0 ) ) or ( ( not X6) and (not X4) and (not X3)
and (X2) and (X1) and (X0 ) ) or ( ( X7) and (not X6) and (X5) and (X4) and (X3)
and (not X2) and (not X0 ) ) or ( ( not X7) and (not X6) and (not X5) and (not X3)
and (X2) and (X1 ) ) or ( ( X7) and (not X6) and (X5) and (not X4) and (X3) and
(not X1 ) ) or ( ( X5) and (X4) and (not X3) and (not X2) and (not X1) and (not
X0 ) ) or ( ( not X7) and (not X6) and (not X5) and (X3) and (X2) and (not X0 )
) or ( ( not X7) and (not X6) and (X3) and (X2) and (not X1) and (not X0 ) ) or
( ( X7) and (X6) and (X5) and (not X3) and (X1) and (not X0 ) ) or ( ( not X7) and
(not X6) and (X5) and (X4) and (X2) and (X0 ) ) or ( ( X6) and (X5) and (X4) and
(X3) and (not X2) and (X0 ) ) or ( ( not X7) and (X5) and (X4) and (X2) and (X1)
and (X0 ) ) or ( ( X7) and (X5) and (not X4) and (X3) and (X0 ) ) or ( ( not X7)
and (X6) and (X4) and (not X3) and (not X1) and (X0 ) ) or ( ( X7) and (X6) and
(not X2) and (not X1) and (X0 ) ) or ( ( X7) and (X6) and (not X5) and (X4) and
(X3) and (X2) and (X1 ) ) or ( ( X7) and (not X4) and (X3) and (X2) and (X1) and
(X0 ) ) or ( ( X7) and (X6) and (X4) and (not X3) and (X2) and (X1) and (X0));
```

```
--Input Cost <= 329 Gate Cost <= 47
```

```
Z2 <= ((not X7) and (not X6) and (not X4) and (not X3) and (not X2) and (X1)
and (X0 ) ) or ( ( not X6) and (not X5) and (X4) and (not X2) and (X1) and (not
X0 ) ) or ( ( not X5) and (X4) and (X3) and (X2) and (not X1) and (not X0 ) ) or
( ( not X6) and (X5) and (X4) and (X3) and (not X2) and (not X1) and (not X0 ) )
or ( ( not X7) and (X5) and (X4) and (X3) and (X2) and (X0 ) ) or ( ( not X7) and
(X6) and (X4) and (not X3) and (not X2) and (X1) and (X0 ) ) or ( ( not X7) and
(X6) and (X4) and (not X3) and (X2) and (X1) and (not X0 ) ) or ( ( not X7) and
(X6) and (X5) and (not X3) and (not X2) and (not X0 ) ) or ( ( not X7) and (X6)
and (X4) and (X3) and (not X1) and (X0 ) ) or ( ( X7) and (not X6) and (not X5)
and (not X3) and (X1) and (not X0 ) ) or ( ( X7) and (not X6) and (not X4) and (not
X3) and (X2) and (X1 ) ) or ( ( X7) and (X6) and (not X5) and (not X4) and (X3)
and (X2) and (X1 ) ) or ( ( X7) and (X6) and (X3) and (X1) and (X0 ) ) or ( ( X7)
and (X6) and (X5) and (X4) and (not X3) and (not X1) and (X0 ) ) or ( ( not X6)
and (not X5) and (X4) and (not X3) and (not X2) and (not X1 ) ) or ( ( not X7) and
(not X6) and (not X5) and (X4) and (not X2) and (not X1) and (X0 ) ) or ( ( not
X7) and (not X6) and (X5) and (not X4) and (X3) and (not X2) and (X1 ) ) or ( (
not X7) and (not X6) and (X5) and (X4) and (not X3) and (X1 ) ) or ( ( X7) and (not
X6) and (not X5) and (X3) and (not X2) and (X1 ) ) or ( ( X7) and (not X6) and (X5)
and (X3) and (not X2) and (not X1) and (X0 ) ) or ( ( X7) and (X5) and (X3) and
(X2) and (X1) and (X0 ) ) or ( ( X7) and (X6) and (X5) and (X4) and (X2) and (X0
) ) or ( ( not X7) and (not X6) and (X4) and (not X3) and (not X2) and (not X1)
and (X0 ) ) or ( ( not X7) and (X6) and (not X4) and (not X3) and (X2) and (X0 )
) or ( ( X7) and (not X6) and (not X5) and (X4) and (not X3) and (not X2 ) ) or
( ( X7) and (not X6) and (not X4) and (not X3) and (not X2) and (not X1 ) ) or (
( X7) and (not X6) and (X5) and (not X4) and (X3) and (X2) and (not X1 ) ) or (
( X7) and (X6) and (not X5) and (not X4) and (not X3) and (not X2 ) ) or ( ( X6)
and (not X4) and (not X3) and (X2) and (X1) and (X0 ) ) or ( ( X7) and (X5) and
(X4) and (not X2) and (X1) and (not X0 ) ) or ( ( not X7) and (X5) and (not X4)
and (not X3) and (not X1) and (not X0 ) ) or ( ( X6) and (X5) and (X4) and (X3)
and (X2) and (X1 ) ) or ( ( X7) and (X4) and (X3) and (not X2) and (not X1) and
(not X0 ) ) or ( ( X6) and (X5) and (not X4) and (X3) and (not X1) and (not X0 )
```

```
) or ( ( not X7) and (X5) and (not X4) and (X3) and (X2) and (not X0 ) ) or ( (
not X7) and (X6) and (not X5) and (not X4) and (not X2) and (not X1 ) ) or ( ( not
X7) and (not X6) and (not X5) and (not X4) and (X3) and (X2) and (X1 ) ) or ( (
X6) and (not X5) and (X3) and (not X2) and (not X1) and (X0 ) ) or ( ( not X7) and
(not X6) and (not X5) and (not X4) and (X2) and (not X1) and (X0 ) ) or ( ( not
X7) and (X6) and (not X5) and (X3) and (not X1) and (not X0 ) ) or ( ( not X5) and
(X4) and (not X3) and (X2) and (not X1) and (X0 ) ) or ( ( not X7) and (X6) and
(X5) and (X4) and (X2) and (not X0 ) ) or ( ( X7) and (not X6) and (not X5) and
(not X4) and (not X3) and (not X1 ) ) or ( ( X7) and (X6) and (not X5) and (not
X3) and (X2) and (not X1) and (not X0 ) ) or ( ( X6) and (X5) and (not X4) and (X2)
and (not X1) and (not X0 ) ) or ( ( X7) and (not X6) and (X5) and (X4) and (not
X3) and (not X1) and (not X0 ) ) or ( ( not X7) and (X5) and (not X3) and (X2) and
(not X1) and (not X0));

   --Input Cost <= 343 Gate Cost <= 48

   Z1 <= ((not X7) and (not X6) and (not X5) and (not X4) and (not X3) and (X2)
and (not X1 ) ) or ( ( not X7) and (not X6) and (not X5) and (X1) and (X0 ) ) or
( ( not X7) and (not X6) and (not X5) and (X4) and (not X3) and (X0 ) ) or ( ( not
X7) and (not X6) and (X3) and (not X2) and (not X1) and (not X0 ) ) or ( ( not X7)
and (not X6) and (X5) and (X3) and (X2) and (not X1) and (X0 ) ) or ( ( X6) and
(not X5) and (not X4) and (X3) and (X2) and (not X1) and (X0 ) ) or ( ( not X7)
and (not X5) and (X4) and (not X3) and (X2) and (not X1) and (X0 ) ) or ( ( not
X7) and (X4) and (X3) and (X2) and (X1) and (X0 ) ) or ( ( not X7) and (X6) and
(X5) and (not X4) and (not X3) and (X2) and (not X1 ) ) or ( ( X5) and (X4) and
(X3) and (X2) and (X1 ) ) or ( ( X7) and (not X6) and (not X5) and (not X4) and
(not X2) and (not X0 ) ) or ( ( X7) and (X4) and (X3) and (X2) and (X1) and (not
X0 ) ) or ( ( X7) and (X5) and (X3) and (X2) and (X1) and (X0 ) ) or ( ( X7) and
(X6) and (not X5) and (not X4) and (not X3) and (X2 ) ) or ( ( X7) and (X6) and
(not X5) and (X4) and (X3) and (not X0 ) ) or ( ( X7) and (X6) and (X5) and (X4)
and (not X3) and (not X2) and (not X1 ) ) or ( ( not X7) and (not X6) and (not X5)
and (not X4) and (X3) and (not X2) and (not X1 ) ) or ( ( not X7) and (not X5) and
(X4) and (not X3) and (X2) and (X1) and (not X0 ) ) or ( ( not X6) and (X5) and
(X4) and (X3) and (not X2 ) ) or ( ( not X7) and (X6) and (not X5) and (not X4)
and (not X2) and (X1 ) ) or ( ( not X7) and (X6) and (not X5) and (X3) and (not
X2) and (X1) and (not X0 ) ) or ( ( X7) and (not X5) and (X4) and (X3) and (X2)
and (not X1) and (X0 ) ) or ( ( X7) and (X6) and (X3) and (X2) and (X1) and (not
X0 ) ) or ( ( X7) and (X6) and (X4) and (not X2) and (X1) and (not X0 ) ) or ( (
not X7) and (not X6) and (X4) and (X2) and (not X1) and (X0 ) ) or ( ( not X7) and
(X5) and (not X4) and (X3) and (not X2) and (not X0 ) ) or ( ( not X7) and (not
X6) and (X5) and (X4) and (not X2) and (not X0 ) ) or ( ( not X7) and (X6) and (not
X5) and (not X4) and (X3) and (X1) and (not X0 ) ) or ( ( not X7) and (X6) and (not
X4) and (not X2) and (X1) and (not X0 ) ) or ( ( not X7) and (X6) and (X5) and (not
X3) and (X1) and (X0 ) ) or ( ( X7) and (not X6) and (not X5) and (X4) and (not
X3) and (not X1) and (not X0 ) ) or ( ( X7) and (not X6) and (X4) and (X3) and (not
X2) and (X0 ) ) or ( ( X7) and (X6) and (not X5) and (X4) and (not X3) and (X1 )
) or ( ( not X7) and (not X6) and (X5) and (X4) and (not X3) and (not X1) and (not
X0 ) ) or ( ( not X7) and (X6) and (not X5) and (not X4) and (not X2) and (not X0
) ) or ( ( not X7) and (X5) and (X4) and (not X2) and (not X1) and (not X0 ) ) or
( ( X7) and (not X5) and (X4) and (not X3) and (not X2) and (X1 ) ) or ( ( X5) and
(X4) and (not X3) and (not X2) and (not X1) and (not X0 ) ) or ( ( X7) and (not
X6) and (not X3) and (X2) and (not X1) and (X0 ) ) or ( ( X6) and (X5) and (X4)
and (X2) and (X1) and (X0 ) ) or ( ( not X7) and (X6) and (not X5) and (not X3)
```

```
and (not X2) and (not X1) and (not X0 ) ) or ( ( X6) and (not X5) and (not X4) and
(not X3) and (not X2) and (X0 ) ) or ( ( not X7) and (not X4) and (not X3) and (X1)
and (X0 ) ) or ( ( X7) and (not X6) and (not X4) and (not X3) and (not X1) and (X0
) ) or ( ( not X6) and (X5) and (not X4) and (not X3) and (X2) and (X1) and (not
X0 ) ) or ( ( X7) and (X5) and (not X4) and (X3) and (not X2) and (not X1) and (X0
) ) or ( ( not X7) and (X6) and (X5) and (not X4) and (not X2) and (not X1) and
(X0 ) ) or ( ( X7) and (not X6) and (X5) and (not X4) and (X2) and (not X1) and
(not X0 ) ) or ( ( X7) and (X6) and (not X5) and (not X4) and (not X1) and (X0 )
) or ( ( not X6) and (X5) and (not X4) and (not X3) and (not X2) and (X0 ) ) or
( ( X7) and (not X6) and (X5) and (X4) and (X3) and (not X0 ) ) or ( ( not X6) and
(X4) and (not X2) and (X1) and (X0));

   --Input Cost <= 377 Gate Cost <= 53

   Z0 <=((not X7) and (not X5) and (not X4) and (X2) and (X1) and (X0 ) ) or ( (
not X7) and (not X6) and (X4) and (not X2) and (not X1) and (X0 ) ) or ( ( not X7)
and (not X6) and (X5) and (not X4) and (not X3) and (X2) and (not X0 ) ) or ( (
X6) and (not X4) and (not X3) and (X2) and (not X1 ) ) or ( ( not X7) and (X6) and
(not X5) and (X2) and (not X1) and (X0 ) ) or ( ( X6) and (X5) and (not X4) and
(not X3) and (not X1) and (X0 ) ) or ( ( X6) and (X5) and (not X4) and (X3) and
(not X0 ) ) or ( ( not X7) and (X5) and (X4) and (X3) and (not X1) and (X0 ) ) or
( ( X7) and (not X6) and (not X5) and (not X3) and (X1) and (X0 ) ) or ( ( X7) and
(not X6) and (not X5) and (X4) and (not X2) and (not X1) and (not X0 ) ) or ( (
X7) and (X5) and (not X4) and (X3) and (not X1) and (not X0 ) ) or ( ( not X6) and
(X5) and (X4) and (not X3) and (not X2) and (not X1) and (X0 ) ) or ( ( not X6)
and (X5) and (X4) and (X3) and (X2) and (not X1) and (X0 ) ) or ( ( X7) and (X6)
and (not X5) and (not X4) and (X2) and (not X0 ) ) or ( ( X7) and (X6) and (not
X5) and (not X2) and (not X1) and (X0 ) ) or ( ( X6) and (not X5) and (X4) and (not
X3) and (not X2) and (X1) and (not X0 ) ) or ( ( X7) and (X4) and (not X3) and (X2)
and (X1) and (X0 ) ) or ( ( X7) and (X6) and (not X5) and (X4) and (X3) and (not
X2) and (X0 ) ) or ( ( X7) and (X6) and (X5) and (X4) and (X2) and (X1 ) ) or (
( not X7) and (not X6) and (X4) and (X3) and (not X2) and (X1 ) ) or ( ( not X7)
and (not X6) and (X5) and (not X3) and (not X2) and (not X1) and (not X0 ) ) or
( ( not X7) and (not X4) and (not X3) and (X2) and (not X1) and (X0 ) ) or ( ( not
X7) and (X6) and (not X4) and (X3) and (X2) and (X0 ) ) or ( ( X6) and (not X5)
and (X3) and (X2) and (not X1) and (X0 ) ) or ( ( X7) and (X5) and (not X4) and
(not X3) and (not X2) and (X1) and (not X0 ) ) or ( ( X7) and (X6) and (X5) and
(X4) and (not X2) and (not X1) and (not X0 ) ) or ( ( not X7) and (X5) and (not
X4) and (X3) and (not X2) and (X1 ) ) or ( ( not X7) and (X6) and (X4) and (not
X3) and (not X2) and (not X1) and (not X0 ) ) or ( ( X7) and (not X6) and (not X5)
and (not X3) and (X2) and (X0 ) ) or ( ( X7) and (not X6) and (X5) and (X4) and
(not X3) and (X1) and (not X0 ) ) or ( ( X6) and (not X5) and (not X4) and (X3)
and (X2) and (X1 ) ) or ( ( X7) and (X6) and (X5) and (not X3) and (X2) and (not
X1 ) ) or ( ( not X7) and (X6) and (not X5) and (not X4) and (X3) and (X0 ) ) or
( ( not X7) and (X6) and (not X5) and (X4) and (X3) and (not X0 ) ) or ( ( not X6)
and (not X5) and (X4) and (not X3) and (X2) and (not X1) and (not X0 ) ) or ( (
X7) and (X5) and (not X3) and (X2) and (X1) and (X0 ) ) or ( ( not X7) and (not
X6) and (not X5) and (X4) and (not X3) and (X2) and (X1 ) ) or ( ( X5) and (X4)
and (X3) and (X2) and (X1) and (not X0 ) ) or ( ( not X7) and (not X6) and (X4)
and (X3) and (X2) and (not X1) and (not X0 ) ) or ( ( X7) and (not X6) and (X5)
and (not X4) and (X1) and (X0 ) ) or ( ( not X7) and (not X6) and (X5) and (not
X2) and (X1) and (X0 ) ) or ( ( X7) and (X5) and (X3) and (X2) and (X1) and (not
X0 ) ) or ( ( not X7) and (X6) and (X4) and (not X3) and (not X2) and (X1) and (X0
```

```
) ) or ( ( not X7) and (not X5) and (not X4) and (not X3) and (not X2) and (X1 )
) or ( ( not X5) and (not X4) and (X3) and (not X2) and (not X1) and (X0 ) ) or
( ( not X7) and (not X6) and (not X5) and (not X4) and (X3) and (not X1) and (not
X0 ) ) or ( ( X7) and (not X6) and (not X4) and (not X2) and (X1) and (X0 ) ) or
( ( X7) and (not X4) and (X3) and (X2) and (not X1) and (not X0 ) ) or ( ( X7) and
(not X5) and (not X4) and (not X3) and (not X2) and (not X1) and (not X0 ) ) or
( ( not X6) and (not X5) and (not X4) and (not X3) and (not X2) and (X1));
   --Input Cost <= 365 Gate Cost <= 51
   ---***Total Input Cost <= 2847***
   ---***Total Gate Cost <= 402***
   end Behavioral;
```

Elemento cero

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Cero is
    Port (      In0 : in  STD_LOGIC;
In1 : in  STD_LOGIC;
In2 : in  STD_LOGIC;
In3 : in  STD_LOGIC;
In4 : in  STD_LOGIC;
In5 : in  STD_LOGIC;
In6 : in  STD_LOGIC;
In7 : in  STD_LOGIC;

Out0 : out  STD_LOGIC;
Out1 : out  STD_LOGIC;
Out2 : out  STD_LOGIC;
Out3 : out  STD_LOGIC;
Out4 : out  STD_LOGIC;
Out5 : out  STD_LOGIC;
Out6 : out  STD_LOGIC;
Out7 : out  STD_LOGIC;

Sal  : out STD_LOGIC
           );
end Cero;

architecture Behavioral of Cero is
begin

Out0 <= In0;
Out1 <= In1;
Out2 <= In2;
Out3 <= In3;
Out4 <= In4;
Out5 <= In5;
Out6 <= In6;
```

```
Out7 <= In7;

Sal <=  not(In0 or In1 or In2 or In3 or In4 or In5 or In6 or In7);

end Behavioral;
```

Elemento Restador

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Restador is
    Port (  OpA  :  in  STD_LOGIC_VECTOR(7 downto  0);
OpB  :  in  STD_LOGIC_VECTOR(7 downto  0);
Resta : out STD_LOGIC_VECTOR(7 downto  0));
end Restador;

architecture Behavioral of Restador is
--********************************************************************************
component Sumador is
port ( A : in STD_LOGIC_VECTOR(7 downto 0);
B : in STD_LOGIC_VECTOR(7 downto 0);
S : out STD_LOGIC_VECTOR(7 downto 0);
Ci : in STD_LOGIC;
Co : out STD_LOGIC);
end component;
--********************************************************************************
signal B : STD_LOGIC_VECTOR(7 downto 0);
signal Co : STD_LOGIC;
--********************************************************************************
begin
--********************************************************************************
B(0) <= not(OpB(0));
B(1) <= not(OpB(1));
B(2) <= not(OpB(2));
B(3) <= not(OpB(3));
B(4) <= not(OpB(4));
B(5) <= not(OpB(5));
B(6) <= not(OpB(6));
B(7) <= not(OpB(7));
--********************************************************************************
U0 : Sumador port map (OpA,B,Resta,'1',Co);
--********************************************************************************
end Behavioral;
```

Elemento Sumador

```
----------------------------------------------------------------------------------
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Sumador is
port( A : in STD_LOGIC_VECTOR(7 downto 0);
B : in STD_LOGIC_VECTOR(7 downto 0);
S : out STD_LOGIC_VECTOR(7 downto 0);
Ci : in STD_LOGIC;
Co : out STD_LOGIC);
end Sumador;

architecture Behavioral of Sumador is

signal C : STD_LOGIC_VECTOR(8 downto 0);
begin

C(0) <= Ci;

S(0) <= (A(0) xor B(0)) xor C(0);
C(1) <= (A(0) and B(0)) or (C(0) and (A(0) xor B(0)));

S(1) <= (A(1) xor B(1)) xor C(1);
C(2) <= (A(1) and B(1)) or (C(1) and (A(1) xor B(1)));

S(2) <= (A(2) xor B(2)) xor C(2);
C(3) <= (A(2) and B(2)) or (C(2) and (A(2) xor B(2)));

S(3) <= (A(3) xor B(3)) xor C(3);
C(4) <= (A(3) and B(3)) or (C(3) and (A(3) xor B(3)));

S(4) <= (A(4) xor B(4)) xor C(4);
C(5) <= (A(4) and B(4)) or (C(4) and (A(4) xor B(4)));

S(5) <= (A(5) xor B(5)) xor C(5);
C(6) <= (A(5) and B(5)) or (C(5) and (A(5) xor B(5)));

S(6) <= (A(6) xor B(6)) xor C(6);
C(7) <= (A(6) and B(6)) or (C(6) and (A(6) xor B(6)));

S(7) <= (A(7) xor B(7)) xor C(7);
C(8) <= (A(7) and B(7)) or (C(7) and (A(7) xor B(7)));

Co <= C(8);

end Behavioral;
```

Elemento Comparador

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Comparador is
    Port (  In0  :  in   STD_LOGIC;
In1  :  in   STD_LOGIC;
In2  :  in   STD_LOGIC;
In3  :  in   STD_LOGIC;
In4  :  in   STD_LOGIC;
In5  :  in   STD_LOGIC;
In6  :  in   STD_LOGIC;
In7  :  in   STD_LOGIC;

Out0 : out STD_LOGIC;
Out1 : out STD_LOGIC;
Out2 : out STD_LOGIC;
Out3 : out STD_LOGIC;
Out4 : out STD_LOGIC;
Out5 : out STD_LOGIC;
Out6 : out STD_LOGIC;
Out7 : out STD_LOGIC
);

end Comparador;

architecture Behavioral of Comparador is
------------------------------------------------------------
signal Com_A  :     STD_LOGIC_VECTOR(7 downto 0);
signal Com_B  :     STD_LOGIC_VECTOR(7 downto 0);
signal AigualB : STD_LOGIC_VECTOR(7 downto 0);
------------------------------------------------------------
begin
------------------------------------------------------------
Com_A(0) <= In0;
Com_A(1) <= In1;
Com_A(2) <= In2;
Com_A(3) <= In3;
Com_A(4) <= In4;
Com_A(5) <= In5;
Com_A(6) <= In6;
Com_A(7) <= In7;

Com_B(0) <= '1';
Com_B(1) <= '1';
Com_B(2) <= '1';
Com_B(3) <= '1';
Com_B(4) <= '1';
Com_B(5) <= '1';
Com_B(6) <= '1';
Com_B(7) <= '1';
```

```
Out0 <= AigualB(0);
Out1 <= AigualB(1);
Out2 <= AigualB(2);
Out3 <= AigualB(3);
Out4 <= AigualB(4);
Out5 <= AigualB(5);
Out6 <= AigualB(6);
Out7 <= AigualB(7);

-------------------------------------------------------------
AigualB <= "00000001" when Com_A=Com_B else "00000000";

end Behavioral;
```

Elemento Antilogaritmo

```
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity  AntiLogaritmo is
    Port ( B   : in   STD_LOGIC_VECTOR(7 downto 0);
           AntiLogB  : out  STD_LOGIC_VECTOR(7 downto 0));
end  AntiLogaritmo;

architecture Behavioral of AntiLogaritmo is
signal E0,E1,E2,E3,E4,E5,E6,E7 :  STD_LOGIC;
signal   S0,S1,S2,S3,S4,S5,S6,S7 :  STD_LOGIC;

begin

E0 <= B(0);
E1 <= B(1);
E2 <= B(2);
E3 <= B(3);
E4 <= B(4);
E5 <= B(5);
E6 <= B(6);
E7 <= B(7);

AntiLogB(0) <= S0;
AntiLogB(1) <= S1;
AntiLogB(2) <= S2;
AntiLogB(3) <= S3;
AntiLogB(4) <= S4;
AntiLogB(5) <= S5;
AntiLogB(6) <= S6;
AntiLogB(7) <= S7;
```

```
S7 <=(( not E6) and (E5) and (not E3) and (not E2) and (E1) and (not E0 ) ) or (
( not E6) and (E5) and (E4) and (not E3) and (not E0 ) ) or ( ( not E7) and (E6)
and (not E4) and (not E3) and (not E2) and (not E1 ) ) or ( ( not E7) and (E6) and
(not E5) and (E4) and (not E3) and (E2 ) ) or ( ( not E7) and (E6) and (not E3)
and (E2) and (not E1) and (E0 ) ) or ( ( not E7) and (E6) and (E5) and (E4) and
(E3) and (E2) and (E1 ) ) or ( ( E7) and (E6) and (E5) and (E4) and (not E2) and
(not E1) and (E0 ) ) or ( ( not E7) and (not E6) and (E4) and (E3) and (not E2)
and (not E0 ) ) or ( ( not E7) and (not E5) and (E4) and (not E3) and (not E2) and
(E0 ) ) or ( ( not E7) and (E6) and (E5) and (not E4) and (not E1) and (not E0 )
) or ( ( not E7) and (E5) and (E4) and (not E3) and (not E2) and (not E1) and (not
E0 ) ) or ( ( not E7) and (E5) and (E3) and (not E2) and (not E1) and (E0 ) ) or
( ( not E6) and (not E5) and (E4) and (not E2) and (not E1 ) ) or ( ( E7) and (not
E6) and (E5) and (E4) and (not E2) and (E1) and (E0 ) ) or ( ( E7) and (E5) and
(not E4) and (E2) and (not E1) and (E0 ) ) or ( ( E7) and (E6) and (E4) and (not
E3) and (E2) and (not E1) and (not E0 ) ) or ( ( not E5) and (not E4) and (E3) and
(not E2) and (not E1) and (E0 ) ) or ( ( not E7) and (E6) and (E5) and (not E3)
and (not E2) and (E1) and (E0 ) ) or ( ( E7) and (E6) and (not E5) and (E2) and
(not E1) and (not E0 ) ) or ( ( E6) and (not E5) and (not E3) and (E2) and (E1)
and (E0 ) ) or ( ( not E7) and (E6) and (not E5) and (E2) and (not E1) and (E0 )
) or ( ( not E7) and (not E4) and (not E3) and (E2) and (E1) and (E0 ) ) or ( (
E7) and (not E5) and (not E4) and (not E2) and (not E1) and (E0 ) ) or ( ( E7) and
(E6) and (not E5) and (E4) and (E3) and (E1) and (E0 ) ) or ( ( not E7) and (not
E5) and (E4) and (E3) and (not E2) and (E1 ) ) or ( ( E7) and (not E6) and (not
E3) and (E2) and (not E1) and (E0 ) ) or ( ( E7) and (E6) and (E4) and (E3) and
(not E2) and (not E1 ) ) or ( ( not E5) and (not E4) and (E3) and (E1) and (not
E0 ) ) or ( ( E6) and (not E5) and (E4) and (E3) and (not E1) and (not E0 ) ) or
( ( E7) and (E6) and (E5) and (E4) and (E3) and (E1) and (not E0 ) ) or ( ( not
E7) and (not E6) and (not E5) and (E4) and (E3 ) ) or ( ( not E7) and (E6) and (E5)
and (E3) and (E2) and (not E0 ) ) or ( ( E7) and (E6) and (not E4) and (not E3)
and (E1) and (not E0 ) ) or ( ( E7) and (not E6) and (E5) and (E2) and (not E1)
and (not E0 ) ) or ( ( not E7) and (not E6) and (E5) and (not E4) and (E3) and (E2
) ) or ( ( E7) and (not E6) and (not E3) and (E2) and (E1) and (not E0 ) ) or (
( E7) and (not E6) and (E5) and (not E4) and (E3) and (not E2) and (not E0 ) ) or
( ( E7) and (not E6) and (E4) and (E2) and (E1) and (not E0 ) ) or ( ( not E6) and
(not E4) and (E3) and (E2) and (E1) and (E0 ) ) or ( ( E7) and (not E6) and (not
E4) and (E3) and (E1) and (E0 ) ) or ( ( E7) and (E6) and (E5) and (not E4) and
(E3) and (E2) and (E1));

   --Input Cost <= 293 Gate Cost <= 42

   S6 <=(( not E6) and (E5) and (not E3) and (not E2) and (not E1) and (E0 ) ) or
( ( not E7) and (E6) and (not E5) and (E4) and (not E3) and (E2 ) ) or ( ( not E7)
and (E6) and (not E5) and (E4) and (E1) and (E0 ) ) or ( ( not E7) and (E6) and
(not E3) and (E2) and (not E1) and (not E0 ) ) or ( ( E6) and (E5) and (E3) and
(E2) and (not E1) and (E0 ) ) or ( ( not E7) and (E6) and (E5) and (E4) and (E3)
and (E2) and (E1) and (not E0 ) ) or ( ( E7) and (not E6) and (E5) and (E4) and
(not E2) and (E1 ) ) or ( ( E7) and (not E6) and (not E3) and (E2) and (not E1 )
) or ( ( E6) and (not E5) and (not E3) and (E2) and (E1) and (not E0 ) ) or ( (
E7) and (E5) and (not E4) and (E3) and (E2) and (not E0 ) ) or ( ( E7) and (E6)
and (E5) and (E4) and (not E2) and (not E1) and (not E0 ) ) or ( ( E7) and (E6)
and (E4) and (not E3) and (E1) and (E0 ) ) or ( ( not E7) and (not E6) and (E5)
and (E4) and (not E3) and (E0 ) ) or ( ( not E7) and (not E6) and (E4) and (E3)
and (not E2) and (not E1 ) ) or ( ( not E7) and (E5) and (E3) and (not E2) and (not
```

```
E1) and (not E0 ) ) or ( ( not E7) and (E6) and (E5) and (E3) and (not E2) and (E1)
and (E0 ) ) or ( ( not E5) and (E4) and (not E3) and (E2) and (E1) and (E0 ) ) or
( ( E7) and (not E6) and (E5) and (not E4) and (E1) and (E0 ) ) or ( ( E7) and (not
E6) and (E4) and (E2) and (not E1) and (E0 ) ) or ( ( E7) and (E6) and (not E5)
and (E4) and (E3) and (E1) and (not E0 ) ) or ( ( E7) and (E6) and (not E4) and
(not E3) and (not E1) and (E0 ) ) or ( ( E7) and (E5) and (not E4) and (not E3)
and (E2) and (not E1 ) ) or ( ( E7) and (E6) and (E5) and (E4) and (E3) and (not
E2) and (not E1 ) ) or ( ( not E5) and (not E4) and (E3) and (not E2) and (not E1
) ) or ( ( not E7) and (E5) and (not E4) and (E3) and (E1) and (E0 ) ) or ( ( not
E7) and (not E6) and (E5) and (not E4) and (E3) and (E2 ) ) or ( ( not E7) and (not
E6) and (E5) and (E3) and (E2) and (E1) and (E0 ) ) or ( ( E7) and (not E6) and
(not E4) and (E3) and (E1) and (not E0 ) ) or ( ( not E5) and (not E4) and (E3)
and (not E1) and (E0 ) ) or ( ( E7) and (not E5) and (E3) and (not E2) and (not
E1) and (not E0 ) ) or ( ( not E7) and (not E6) and (not E5) and (E4) and (E3) and
(not E1 ) ) or ( ( not E7) and (not E5) and (E4) and (E3) and (not E2) and (E0 )
) or ( ( not E7) and (E6) and (E5) and (not E4) and (not E3) and (E1 ) ) or ( (
not E6) and (not E4) and (E3) and (E2) and (E1 ) ) or ( ( E7) and (not E6) and (E5)
and (not E4) and (not E2) and (E0 ) ) or ( ( E7) and (E6) and (not E5) and (not
E2) and (E1) and (E0 ) ) or ( ( not E7) and (not E6) and (not E4) and (E2) and (E1)
and (not E0 ) ) or ( ( not E7) and (not E6) and (not E5) and (E4) and (E3) and (not
E0 ) ) or ( ( not E7) and (not E5) and (E4) and (not E2) and (E1) and (not E0 )
) or ( ( not E7) and (E6) and (not E5) and (E2) and (not E1) and (not E0 ) ) or
( ( not E7) and (E6) and (not E4) and (not E2) and (not E1) and (not E0 ) ) or (
( not E7) and (E6) and (E4) and (not E3) and (not E2) and (E1) and (not E0 ) ) or
( ( not E7) and (not E5) and (E4) and (not E3) and (not E2) and (not E0 ) ) or (
( E7) and (not E5) and (not E4) and (not E2) and (not E1) and (not E0 ) ) or ( (
not E6) and (not E5) and (E4) and (not E2) and (not E1) and (not E0));

    --Input Cost <= 319 Gate Cost <= 46

   S5 <= (( not E6) and (not E5) and (E4) and (not E3) and (not E2) and (not E1)
and (not E0 ) ) or ( ( not E6) and (not E5) and (E4) and (not E3) and (E1) and (E0
) ) or ( ( not E7) and (not E6) and (not E5) and (E4) and (E2) and (E1 ) ) or (
( not E6) and (not E4) and (not E3) and (E2) and (not E1) and (E0 ) ) or ( ( not
E7) and (E6) and (not E5) and (not E4) and (not E2) and (E0 ) ) or ( ( E6) and (not
E5) and (E4) and (not E3) and (not E2) and (E1) and (not E0 ) ) or ( ( E6) and (E5)
and (not E4) and (not E3) and (not E0 ) ) or ( ( E5) and (not E4) and (not E3) and
(not E2) and (not E1) and (not E0 ) ) or ( ( not E6) and (E5) and (E3) and (not
E2) and (not E1) and (E0 ) ) or ( ( E7) and (E5) and (E4) and (E2) and (E1) and
(not E0 ) ) or ( ( E6) and (not E5) and (not E4) and (not E2) and (not E1 ) ) or
( ( E7) and (E6) and (E5) and (not E4) and (not E2) and (E1) and (E0 ) ) or ( (
not E7) and (E5) and (not E4) and (not E2) and (E1) and (not E0 ) ) or ( ( not E7)
and (E6) and (E5) and (E4) and (not E3) and (not E2) and (not E1 ) ) or ( ( E7)
and (not E6) and (not E4) and (not E3) and (not E2) and (E1) and (E0 ) ) or ( (
E7) and (not E6) and (E4) and (not E3) and (not E1) and (E0 ) ) or ( ( E7) and (not
E6) and (E5) and (E4) and (E3) and (E1 ) ) or ( ( E7) and (not E4) and (not E3)
and (E2) and (not E1) and (E0 ) ) or ( ( E6) and (E5) and (E4) and (not E3) and
(E2) and (E1) and (E0 ) ) or ( ( not E7) and (not E6) and (E5) and (not E4) and
(E3) and (E1) and (E0 ) ) or ( ( E5) and (E4) and (E3) and (E1) and (not E0 ) )
or ( ( not E7) and (E6) and (E5) and (E4) and (not E1) and (E0 ) ) or ( ( E7) and
(not E6) and (not E5) and (not E3) and (E0 ) ) or ( ( E7) and (not E6) and (not
E5) and (not E4) and (E3) and (not E2) and (E1 ) ) or ( ( E7) and (not E6) and (not
E5) and (E4) and (not E2) and (not E1 ) ) or ( ( E7) and (E5) and (not E4) and (E3)
```

```
and (E2) and (not E0 ) ) or ( ( E7) and (E6) and (not E4) and (E2) and (E1) and
(not E0 ) ) or ( ( E6) and (E5) and (E4) and (not E2) and (not E1) and (E0 ) ) or
( ( not E7) and (E6) and (E3) and (not E2) and (not E1) and (E0 ) ) or ( ( not E7)
and (E6) and (not E5) and (E3) and (E2 ) ) or ( ( E7) and (not E6) and (not E5)
and (not E4) and (not E3) and (E2 ) ) or ( ( not E6) and (E5) and (E3) and (E2)
and (E1) and (E0 ) ) or ( ( E7) and (E6) and (E4) and (not E3) and (E1) and (not
E0 ) ) or ( ( not E7) and (not E5) and (not E4) and (E3) and (not E1 ) ) or ( (
not E7) and (E4) and (not E3) and (E2) and (E1) and (E0 ) ) or ( ( not E7) and (E6)
and (not E4) and (E3) and (not E2) and (not E0 ) ) or ( ( not E6) and (not E5) and
(not E4) and (E3) and (E2) and (not E1 ) ) or ( ( not E6) and (not E5) and (not
E4) and (E3) and (not E2) and (not E0 ) ) or ( ( E7) and (E6) and (not E5) and (E4)
and (E2) and (not E1 ) ) or ( ( E7) and (not E5) and (E4) and (E2) and (not E1)
and (not E0 ) ) or ( ( E7) and (E6) and (E5) and (E2) and (not E0 ) ) or ( ( not
E6) and (E4) and (E3) and (E2) and (E1) and (not E0 ) ) or ( ( E7) and (E6) and
(not E5) and (E4) and (E3) and (E1) and (E0 ) ) or ( ( E7) and (E6) and (not E5)
and (E4) and (E3) and (not E2) and (not E0));

    --Input Cost <= 311 Gate Cost <= 45

   S4 <= (( not E7) and (not E6) and (not E5) and (E4) and (not E3) and (E1) and
(not E0 ) ) or ( ( not E7) and (not E6) and (not E5) and (E3) and (not E1) and (not
E0 ) ) or ( ( not E7) and (not E6) and (E5) and (not E4) and (not E3) and (not E2)
and (not E1) and (E0 ) ) or ( ( not E7) and (not E6) and (E5) and (not E2) and (E1)
and (not E0 ) ) or ( ( not E7) and (not E6) and (E5) and (not E4) and (E2) and (E1)
and (E0 ) ) or ( ( E6) and (not E5) and (not E4) and (not E3) and (not E1) and (E0
) ) or ( ( E6) and (not E5) and (not E4) and (not E3) and (not E2) and (E1) and
(not E0 ) ) or ( ( not E7) and (not E5) and (not E4) and (E3) and (not E2) and (E1)
and (E0 ) ) or ( ( not E7) and (E6) and (E4) and (not E3) and (E2) and (not E1 )
) or ( ( not E7) and (E6) and (E5) and (not E4) and (not E3) and (not E1) and (not
E0 ) ) or ( ( E6) and (E4) and (not E3) and (not E2) and (E1) and (E0 ) ) or ( (
not E7) and (E6) and (E5) and (E4) and (E3) and (not E2) and (not E1) and (not E0
) ) or ( ( E7) and (not E6) and (not E5) and (not E3) and (E1) and (E0 ) ) or (
( E7) and (not E6) and (E5) and (not E4) and (not E3) and (E2) and (not E1) and
(E0 ) ) or ( ( E7) and (E6) and (not E5) and (E3) and (not E2) and (not E0 ) ) or
( ( E7) and (E6) and (E5) and (E4) and (not E3) and (not E1) and (not E0 ) ) or
( ( not E7) and (E5) and (E4) and (E3) and (E2) and (E1 ) ) or ( ( E6) and (not
E5) and (not E4) and (E3) and (not E1) and (not E0 ) ) or ( ( not E7) and (E6) and
(not E5) and (E4) and (E2) and (E1) and (E0 ) ) or ( ( E6) and (E5) and (E4) and
(E2) and (E1) and (not E0 ) ) or ( ( E7) and (E6) and (not E4) and (E2) and (E1)
and (not E0 ) ) or ( ( E7) and (E6) and (not E5) and (not E4) and (E3) and (not
E1 ) ) or ( ( E7) and (E6) and (E5) and (not E4) and (not E3) and (not E2) and (E0
) ) or ( ( E7) and (E6) and (E5) and (not E3) and (E2) and (not E0 ) ) or ( ( not
E5) and (E4) and (not E3) and (E2) and (not E1) and (E0 ) ) or ( ( not E7) and (not
E6) and (not E5) and (E4) and (not E2 ) ) or ( ( E6) and (not E5) and (E4) and (E3)
and (E1) and (not E0 ) ) or ( ( E7) and (not E6) and (not E5) and (not E3) and (E2)
and (not E1) and (not E0 ) ) or ( ( E7) and (not E6) and (E4) and (E3) and (E0 )
) or ( ( E7) and (not E5) and (E3) and (E2) and (E1) and (not E0 ) ) or ( ( E7)
and (not E5) and (not E3) and (not E2) and (not E1) and (E0 ) ) or ( ( E7) and (E6)
and (E5) and (E3) and (not E2) and (E1 ) ) or ( ( E6) and (E5) and (E4) and (E2)
and (not E1) and (E0 ) ) or ( ( not E7) and (not E6) and (E5) and (E3) and (E2)
and (E0 ) ) or ( ( E7) and (not E5) and (not E4) and (not E3) and (not E2) and (not
E0 ) ) or ( ( E7) and (not E6) and (E4) and (not E3) and (not E2) and (E1 ) ) or
( ( E7) and (E5) and (not E3) and (E2) and (E1) and (not E0 ) ) or ( ( E7) and (not
```

```
E6) and (not E4) and (E3) and (E2) and (not E0 ) ) or ( ( not E7) and (not E6) and
(not E4) and (E2) and (not E1) and (not E0 ) ) or ( ( not E7) and (not E6) and (not
E5) and (not E4) and (E3) and (not E0 ) ) or ( ( E7) and (E5) and (E3) and (not
E2) and (E1) and (not E0 ) ) or ( ( E5) and (not E4) and (E3) and (E2) and (E1)
and (E0 ) ) or ( ( E7) and (not E6) and (E5) and (E4) and (E3) and (not E1 ) ) or
( ( not E7) and (not E6) and (not E4) and (E3) and (not E2) and (not E0 ) ) or (
( not E7) and (E6) and (E5) and (not E4) and (E3) and (E2) and (not E0 ) ) or (
( not E7) and (not E6) and (E5) and (E4) and (not E3) and (not E1) and (not E0));
   --Input Cost <= 337 Gate Cost <= 47
   S3 <=(( not E7) and (E6) and (not E4) and (not E3) and (E2) and (not E1 ) ) or
( ( not E7) and (E6) and (E5) and (not E4) and (not E1 ) ) or ( ( E7) and (not E5)
and (not E4) and (not E3) and (not E2) and (not E0 ) ) or ( ( E7) and (not E6) and
(E5) and (not E4) and (E3) and (not E0 ) ) or ( ( E7) and (E6) and (E5) and (not
E3) and (not E2) and (E1) and (E0 ) ) or ( ( E7) and (not E4) and (E3) and (not
E2) and (E1) and (not E0 ) ) or ( ( E7) and (E6) and (E5) and (E4) and (E3) and
(E2) and (not E0 ) ) or ( ( not E7) and (not E6) and (E5) and (not E3) and (not
E2 ) ) or ( ( not E7) and (E6) and (E5) and (not E3) and (E2) and (E1) and (E0 )
) or ( ( not E7) and (E5) and (E4) and (not E3) and (not E1) and (not E0 ) ) or
( ( not E6) and (E5) and (not E4) and (not E3) and (E1) and (not E0 ) ) or ( ( E7)
and (E6) and (not E5) and (not E2) and (not E1) and (not E0 ) ) or ( ( E6) and (not
E5) and (E3) and (not E2) and (E1) and (E0 ) ) or ( ( E6) and (E5) and (not E4)
and (not E3) and (not E2) and (not E1) and (not E0 ) ) or ( ( E7) and (E5) and (not
E4) and (E3) and (not E2) and (not E1) and (E0 ) ) or ( ( E6) and (E4) and (not
E3) and (not E2) and (E1) and (not E0 ) ) or ( ( not E6) and (not E5) and (E4) and
(not E3) and (not E2) and (E1 ) ) or ( ( E7) and (not E6) and (E3) and (not E2)
and (not E1) and (E0 ) ) or ( ( E7) and (not E6) and (not E4) and (E3) and (E2)
and (E1 ) ) or ( ( E7) and (E6) and (not E4) and (not E3) and (E2) and (E1) and
(not E0 ) ) or ( ( not E6) and (not E5) and (not E3) and (not E2) and (E1) and (E0
) ) or ( ( E7) and (not E6) and (E3) and (not E2) and (E1) and (not E0 ) ) or (
( E7) and (not E6) and (E5) and (E4) and (E2) and (E1) and (E0 ) ) or ( ( not E5)
and (not E4) and (E3) and (E2) and (E1) and (not E0 ) ) or ( ( not E7) and (not
E4) and (E3) and (not E2) and (E1) and (E0 ) ) or ( ( E7) and (E5) and (E4) and
(not E3) and (not E2) and (E0 ) ) or ( ( not E6) and (E5) and (E4) and (not E3)
and (E2) and (not E0 ) ) or ( ( not E6) and (not E5) and (E4) and (E2) and (not
E1 ) ) or ( ( E7) and (not E6) and (E5) and (E4) and (not E2) and (not E1 ) ) or
( ( E7) and (E6) and (E5) and (E4) and (E2) and (not E1) and (E0 ) ) or ( ( not
E7) and (not E6) and (E5) and (E4) and (not E0 ) ) or ( ( E7) and (not E5) and (not
E4) and (E2) and (not E1) and (E0 ) ) or ( ( E7) and (E6) and (E4) and (E3) and
(not E1) and (not E0 ) ) or ( ( not E7) and (E6) and (E3) and (not E2) and (not
E1) and (E0 ) ) or ( ( not E7) and (not E4) and (E3) and (E2) and (not E1) and (E0
) ) or ( ( not E7) and (E6) and (not E5) and (E4) and (not E3) and (not E1) and
(E0 ) ) or ( ( not E7) and (E6) and (not E5) and (E4) and (E3) and (not E0 ) ) or
( ( not E7) and (E5) and (E3) and (E2) and (not E1) and (E0 ) ) or ( ( not E7) and
(not E5) and (E4) and (E3) and (E2) and (E1 ) ) or ( ( not E7) and (not E5) and
(E4) and (E2) and (E1) and (E0 ) ) or ( ( E7) and (E6) and (not E5) and (E4) and
(E3) and (E2) and (E0));
   --Input Cost <= 293 Gate Cost <= 42
   S2 <=(( not E7) and (E6) and (not E4) and (not E3) and (not E2) and (E1) and
(E0 ) ) or ( ( not E6) and (not E5) and (not E3) and (not E2) and (E1) and (not
E0 ) ) or ( ( E7) and (not E6) and (E5) and (not E4) and (E3) and (E0 ) ) or ( (
E6) and (not E5) and (E3) and (not E2) and (E1) and (not E0 ) ) or ( ( E7) and (E6)
```

```
and (E5) and (not E3) and (not E2) and (E1) and (not E0 ) ) or ( ( E7) and (E6)
and (E4) and (E3) and (not E2) and (E1) and (E0 ) ) or ( ( E7) and (E6) and (E5)
and (E4) and (E3) and (E2) and (not E1 ) ) or ( ( not E7) and (E5) and (not E4)
and (E3) and (E2) and (E1) and (E0 ) ) or ( ( not E7) and (E5) and (E4) and (not
E3) and (not E2) and (E0 ) ) or ( ( E7) and (E6) and (E5) and (E4) and (not E3)
and (not E1) and (not E0 ) ) or ( ( not E7) and (E6) and (E4) and (E3) and (E2)
and (E1) and (E0 ) ) or ( ( E7) and (not E6) and (not E4) and (E3) and (E2) and
(E1) and (not E0 ) ) or ( ( E7) and (not E6) and (E3) and (not E2) and (not E1 )
) or ( ( E7) and (not E6) and (E5) and (not E3) and (E2) and (E0 ) ) or ( ( E7)
and (E5) and (not E4) and (E3) and (not E2) and (not E1 ) ) or ( ( E6) and (E4)
and (not E3) and (not E2) and (not E1) and (E0 ) ) or ( ( E7) and (E6) and (E4)
and (not E3) and (E2) and (E1) and (E0 ) ) or ( ( not E6) and (not E5) and (E4)
and (not E3) and (not E2) and (E0 ) ) or ( ( not E6) and (E5) and (E4) and (not
E3) and (not E2) and (not E0 ) ) or ( ( E7) and (not E6) and (E5) and (E4) and (E2)
and (E1 ) ) or ( ( E7) and (E6) and (not E4) and (not E3) and (E2) and (not E1)
and (E0 ) ) or ( ( not E7) and (not E6) and (E5) and (E4) and (not E2) and (E0 )
) or ( ( not E5) and (not E4) and (E3) and (E2) and (not E1 ) ) or ( ( not E5) and
(E4) and (E3) and (not E2) and (E1) and (E0 ) ) or ( ( E7) and (not E5) and (not
E4) and (not E2) and (not E1) and (E0 ) ) or ( ( E7) and (E6) and (not E5) and (E2)
and (E1) and (E0 ) ) or ( ( not E7) and (E6) and (not E5) and (E4) and (E2) and
(E1 ) ) or ( ( not E6) and (E5) and (not E4) and (not E3) and (not E1) and (E0 )
) or ( ( not E7) and (E6) and (E5) and (not E3) and (E2) and (E1) and (not E0 )
) or ( ( not E7) and (E5) and (E3) and (E2) and (not E1) and (not E0 ) ) or ( (
not E7) and (not E6) and (E5) and (E4) and (not E1) and (E0 ) ) or ( ( not E7) and
(E6) and (E5) and (not E4) and (E1) and (E0 ) ) or ( ( E7) and (not E6) and (not
E5) and (E2) and (not E1) and (not E0 ) ) or ( ( not E6) and (E5) and (E4) and (not
E3) and (E1) and (E0 ) ) or ( ( not E7) and (E6) and (E3) and (not E2) and (not
E1) and (not E0 ) ) or ( ( E6) and (not E5) and (not E4) and (E2) and (not E1) and
(not E0 ) ) or ( ( not E7) and (E6) and (E5) and (not E4) and (not E1) and (not
E0 ) ) or ( ( not E7) and (not E4) and (E3) and (not E2) and (E1) and (not E0 )
) or ( ( not E7) and (not E6) and (E5) and (not E3) and (not E2) and (not E0 ) )
or ( ( not E7) and (E6) and (not E5) and (E4) and (E3) and (E0 ) ) or ( ( not E7)
and (not E6) and (not E5) and (E4) and (E3) and (E2 ) ) or ( ( not E7) and (not
E5) and (E4) and (not E3) and (E2) and (not E0 ) ) or ( ( not E7) and (E6) and (not
E5) and (E4) and (not E2) and (not E1 ) ) or ( ( E7) and (E6) and (not E5) and (E4)
and (E3) and (E2) and (not E0));

   --Input Cost <= 318 Gate Cost <= 45

   S1 <=(( not E7) and (E6) and (not E4) and (not E3) and (not E2) and (E1 ) ) or
( ( not E7) and (E6) and (E5) and (not E3) and (E2) and (E0 ) ) or ( ( not E7) and
(not E4) and (E3) and (not E2) and (E0 ) ) or ( ( E7) and (not E6) and (not E4)
and (E3) and (E2) and (not E1 ) ) or ( ( E7) and (not E6) and (E5) and (not E4)
and (E3) and (not E0 ) ) or ( ( E7) and (not E6) and (E5) and (E2) and (E1) and
(not E0 ) ) or ( ( E6) and (not E5) and (E4) and (E3) and (not E1) and (E0 ) ) or
( ( E7) and (E6) and (E5) and (not E3) and (not E2) and (not E1) and (E0 ) ) or
( ( E7) and (not E4) and (E3) and (not E2) and (not E1) and (not E0 ) ) or ( ( E7)
and (E5) and (not E4) and (E2) and (E1) and (E0 ) ) or ( ( E7) and (E6) and (E4)
and (E3) and (not E2) and (E1 ) ) or ( ( E7) and (E6) and (E5) and (E4) and (E3)
and (E2) and (not E1) and (not E0 ) ) or ( ( not E6) and (E5) and (not E4) and (E3)
and (E2) and (E1 ) ) or ( ( not E7) and (E6) and (not E5) and (E2) and (E1) and
(E0 ) ) or ( ( not E7) and (E6) and (E5) and (not E4) and (E1) and (not E0 ) ) or
( ( not E7) and (E5) and (E4) and (not E3) and (not E2) and (not E0 ) ) or ( ( not
```

```
E7) and (E5) and (E3) and (not E2) and (E1) and (E0 ) ) or ( ( E7) and (not E6)
and (E5) and (E4) and (E2) and (not E1) and (E0 ) ) or ( ( E7) and (not E5) and
(not E4) and (not E2) and (E1) and (E0 ) ) or ( ( E6) and (not E5) and (not E4)
and (E3) and (not E2) and (E0 ) ) or ( ( E6) and (E4) and (not E3) and (not E2)
and (not E1) and (not E0 ) ) or ( ( E7) and (E6) and (E5) and (E4) and (not E3)
and (not E2) and (E0 ) ) or ( ( E7) and (E6) and (E4) and (not E3) and (E2) and
(E1) and (not E0 ) ) or ( ( not E6) and (not E5) and (E4) and (not E3) and (not
E2 ) ) or ( ( not E7) and (not E5) and (E4) and (E2) and (not E1) and (E0 ) ) or
( ( E6) and (not E5) and (E4) and (E2) and (E1) and (not E0 ) ) or ( ( not E7) and
(E6) and (not E5) and (E4) and (E3) and (not E0 ) ) or ( ( not E7) and (E6) and
(E4) and (E3) and (E2) and (E1) and (not E0 ) ) or ( ( not E6) and (not E5) and
(E4) and (not E2) and (E1 ) ) or ( ( not E6) and (E4) and (not E3) and (not E2)
and (not E1) and (E0 ) ) or ( ( not E7) and (not E5) and (E4) and (E3) and (E2 )
) or ( ( not E7) and (not E6) and (E5) and (E4) and (not E2) and (not E0 ) ) or
( ( not E7) and (E6) and (not E5) and (not E3) and (E1) and (E0 ) ) or ( ( E7) and
(not E6) and (not E3) and (E2) and (E1) and (E0 ) ) or ( ( not E6) and (E5) and
(E4) and (not E3) and (E1) and (not E0 ) ) or ( ( E7) and (E6) and (not E5) and
(not E4) and (E2) and (not E0 ) ) or ( ( not E7) and (not E5) and (E3) and (E2)
and (not E1) and (not E0 ) ) or ( ( not E7) and (not E6) and (E5) and (E4) and (not
E1) and (not E0 ) ) or ( ( not E6) and (E5) and (not E3) and (E2) and (not E1) and
(not E0 ) ) or ( ( E7) and (not E6) and (E3) and (not E2) and (not E1) and (not
E0 ) ) or ( ( E7) and (not E5) and (not E3) and (not E2) and (not E1) and (not E0
) ) or ( ( E7) and (E5) and (not E4) and (not E3) and (E2) and (not E1) and (not
E0 ) ) or ( ( not E6) and (not E5) and (not E3) and (not E2) and (not E1) and (E0
) ) or ( ( not E6) and (E5) and (not E4) and (not E3) and (not E1) and (not E0 )
) or ( ( not E7) and (not E6) and (not E3) and (not E2) and (not E1) and (E0));

   --Input Cost <= 319 Gate Cost <= 46

   S0 <=(( not E6) and (E5) and (not E3) and (not E2) and (E1) and (E0 ) ) or (
( not E6) and (E5) and (E4) and (not E3) and (E0 ) ) or ( ( not E7) and (E6) and
(not E4) and (not E3) and (not E2) and (not E1) and (E0 ) ) or ( ( not E7) and (E6)
and (not E5) and (E4) and (not E3) and (E2 ) ) or ( ( not E7) and (E6) and (E5)
and (E3) and (E2) and (E0 ) ) or ( ( E7) and (not E6) and (not E4) and (E3) and
(E2) and (not E1) and (not E0 ) ) or ( ( E6) and (not E5) and (E4) and (E3) and
(not E1 ) ) or ( ( E7) and (E6) and (E5) and (not E3) and (not E2) and (not E1)
and (not E0 ) ) or ( ( not E7) and (E5) and (E4) and (not E3) and (not E2) and (not
E1) and (E0 ) ) or ( ( not E7) and (E6) and (E5) and (not E3) and (E2) and (not
E0 ) ) or ( ( not E6) and (not E5) and (E4) and (not E2) and (not E1) and (E0 )
) or ( ( E7) and (E6) and (E5) and (E4) and (not E2) and (E1) and (not E0 ) ) or
( ( E7) and (E6) and (E4) and (not E3) and (E2) and (not E1) and (E0 ) ) or ( (
E7) and (E6) and (E5) and (E4) and (E3) and (not E2) and (E0 ) ) or ( ( not E7)
and (not E5) and (E4) and (E2) and (not E1) and (not E0 ) ) or ( ( not E7) and (not
E6) and (not E5) and (E4) and (E3) and (E2 ) ) or ( ( not E7) and (not E6) and (E4)
and (E3) and (not E2) and (E0 ) ) or ( ( not E7) and (not E5) and (E3) and (not
E2) and (E1) and (E0 ) ) or ( ( not E7) and (E6) and (E5) and (not E4) and (not
E1) and (E0 ) ) or ( ( not E7) and (E5) and (E3) and (not E2) and (E1) and (not
E0 ) ) or ( ( E7) and (not E6) and (not E5) and (not E3) and (not E2) and (not E0
) ) or ( ( E7) and (not E6) and (E4) and (E2) and (E1) and (E0 ) ) or ( ( E7) and
(E6) and (not E5) and (not E4) and (E2) and (E0 ) ) or ( ( not E7) and (E6) and
(not E5) and (E2) and (E1) and (not E0 ) ) or ( ( not E7) and (E6) and (not E4)
and (not E3) and (E1) and (not E0 ) ) or ( ( not E7) and (not E4) and (E3) and (not
E2) and (not E0 ) ) or ( ( E7) and (not E6) and (not E3) and (E2) and (E1 ) ) or
```

```
( ( not E7) and (not E6) and (not E3) and (not E2) and (not E1) and (not E0 ) )
or ( ( not E5) and (not E4) and (E3) and (E1) and (E0 ) ) or ( ( E7) and (not E6)
and (E5) and (E2) and (not E1) and (E0 ) ) or ( ( E7) and (E6) and (not E5) and
(E3) and (not E2) and (not E0 ) ) or ( ( E7) and (E6) and (not E4) and (not E3)
and (E1) and (E0 ) ) or ( ( E7) and (E6) and (E5) and (not E4) and (E2) and (E1
) ) or ( ( not E7) and (E5) and (not E4) and (E3) and (E2) and (E0 ) ) or ( ( E7)
and (not E6) and (E5) and (not E4) and (E3) and (not E2) and (E0 ) ) or ( ( E7)
and (not E6) and (E5) and (E4) and (E2) and (not E1 ) ) or ( ( E7) and (not E5)
and (not E4) and (not E2) and (E1) and (not E0 ) ) or ( ( not E6) and (not E5) and
(E4) and (not E2) and (E1) and (not E0 ) ) or ( ( not E6) and (E5) and (not E4)
and (E3) and (E2) and (E1) and (not E0 ) ) or ( ( not E6) and (E4) and (not E3)
and (not E2) and (not E1 ) ) or ( ( not E7) and (E6) and (not E5) and (not E3) and
(E1) and (not E0));
   --Input Cost <= 290 Gate Cost <= 42
   ---***Total Input Cost <= 2480***
   ---***Total Gate Cost <= 355***
   end Behavioral;
```

Elemento Compuerta

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Compuerta is
port (
In0  :  in   STD_LOGIC;
In1  :  in   STD_LOGIC;
In2  :  in   STD_LOGIC;
In3  :  in   STD_LOGIC;
In4  :  in   STD_LOGIC;
In5  :  in   STD_LOGIC;
In6  :  in   STD_LOGIC;
In7  :  in   STD_LOGIC;

Inhividor: in STD_LOGIC;

Out0 : out STD_LOGIC;
Out1 : out STD_LOGIC;
Out2 : out STD_LOGIC;
Out3 : out STD_LOGIC;
Out4 : out STD_LOGIC;
Out5 : out STD_LOGIC;
Out6 : out STD_LOGIC;
Out7 : out STD_LOGIC
);

end Compuerta;

architecture Behavioral of Compuerta is
----------------------------------------
```

```
signal A : STD_LOGIC_VECTOR(7 downto 0);
signal B : STD_LOGIC_VECTOR(7 downto 0);
signal X : STD_LOGIC_VECTOR(7 downto 0);
----------------------------------------
begin
----------------------------------------
A(0) <= In0;
A(1) <= In1;
A(2) <= In2;
A(3) <= In3;
A(4) <= In4;
A(5) <= In5;
A(6) <= In6;
A(7) <= In7;

B(0) <= not(Inhividor);
B(1) <= not(Inhividor);
B(2) <= not(Inhividor);
B(3) <= not(Inhividor);
B(4) <= not(Inhividor);
B(5) <= not(Inhividor);
B(6) <= not(Inhividor);
B(7) <= not(Inhividor);

Out0 <= X(0);
Out1 <= X(1);
Out2 <= X(2);
Out3 <= X(3);
Out4 <= X(4);
Out5 <= X(5);
Out6 <= X(6);
Out7 <= X(7);
----------------------------------------
X(0) <= A(0) and B(0);
X(1) <= A(1) and B(1);
X(2) <= A(2) and B(2);
X(3) <= A(3) and B(3);
X(4) <= A(4) and B(4);
X(5) <= A(5) and B(5);
X(6) <= A(6) and B(6);
X(7) <= A(7) and B(7);

end Behavioral;
```

Elemento Divisor

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity Divisor is
    Port (  A  : in   STD_LOGIC_VECTOR(7 downto 0);
B  : in   STD_LOGIC_VECTOR(7 downto 0);
D : out STD_LOGIC_VECTOR (7 downto 0));
end Divisor;

architecture Behavioral of Divisor is
--*****************************************************************************
component Cero is
    Port (  In0  : in  STD_LOGIC;
In1  : in  STD_LOGIC;
In2  : in  STD_LOGIC;
In3  : in  STD_LOGIC;
In4  : in  STD_LOGIC;
In5  : in  STD_LOGIC;
In6  : in  STD_LOGIC;
In7  : in  STD_LOGIC;

Out0  : out  STD_LOGIC;
Out1  : out  STD_LOGIC;
Out2  : out  STD_LOGIC;
Out3 : out  STD_LOGIC;
Out4  : out  STD_LOGIC;
Out5  : out  STD_LOGIC;
Out6  : out  STD_LOGIC;
Out7  : out  STD_LOGIC;

Sal : out STD_LOGIC);
end component;

component Logaritmo is
    Port (  In0   : in   STD_LOGIC;
In1 : in   STD_LOGIC;
In2 : in   STD_LOGIC;
In3 : in   STD_LOGIC;
In4 : in   STD_LOGIC;
In5 : in   STD_LOGIC;
In6 : in   STD_LOGIC;
In7 : in   STD_LOGIC;

Out0 : out STD_LOGIC;
Out1 : out STD_LOGIC;
Out2 : out STD_LOGIC;
Out3 : out STD_LOGIC;
Out4 : out STD_LOGIC;
Out5 : out STD_LOGIC;
Out6 : out STD_LOGIC;
Out7 : out STD_LOGIC);
end component;

component Restador is
```

```
    Port (  OpA  :  in  STD_LOGIC_VECTOR(7 downto  0);
OpB  :  in  STD_LOGIC_VECTOR(7 downto  0);
Resta : out STD_LOGIC_VECTOR(7 downto  0));
end component;

component Sumador is
port  ( A : in STD_LOGIC_VECTOR(7 downto 0);
B : in STD_LOGIC_VECTOR(7 downto 0);
S : out STD_LOGIC_VECTOR(7 downto 0);
Ci : in STD_LOGIC;
Co : out STD_LOGIC);
end component;

component Comparador is
    Port (  In0 :  in  STD_LOGIC;
In1  :  in  STD_LOGIC;
In2  :  in  STD_LOGIC;
In3  :  in  STD_LOGIC;
In4  :  in  STD_LOGIC;
In5  :  in  STD_LOGIC;
In6  :  in  STD_LOGIC;
In7  :  in  STD_LOGIC;

Out0 : out STD_LOGIC;
Out1 : out STD_LOGIC;
Out2 : out STD_LOGIC;
Out3 : out STD_LOGIC;
Out4 : out STD_LOGIC;
Out5 : out STD_LOGIC;
Out6 : out STD_LOGIC;
Out7 : out STD_LOGIC);
end component;

component  AntiLogaritmo is
    Port (  In0   :  in  STD_LOGIC;
In1 :  in  STD_LOGIC;
In2 :  in  STD_LOGIC;
In3 :  in  STD_LOGIC;
In4 :  in  STD_LOGIC;
In5 :  in  STD_LOGIC;
In6 :  in  STD_LOGIC;
In7 :  in  STD_LOGIC;

Out0 : out STD_LOGIC;
Out1 : out STD_LOGIC;
Out2 : out STD_LOGIC;
Out3 : out STD_LOGIC;
Out4 : out STD_LOGIC;
Out5 : out STD_LOGIC;
Out6 : out STD_LOGIC;
Out7 : out STD_LOGIC);
```

```
end component;

component Compuerta is
port ( In0  :  in  STD_LOGIC;
In1  :  in  STD_LOGIC;
In2  :  in  STD_LOGIC;
In3  :  in  STD_LOGIC;
In4  :  in  STD_LOGIC;
In5  :  in  STD_LOGIC;
In6  :  in  STD_LOGIC;
In7  :  in  STD_LOGIC;

Inhividor : in STD_LOGIC;

Out0 : out STD_LOGIC;
Out1 : out STD_LOGIC;
Out2 : out STD_LOGIC;
Out3 : out STD_LOGIC;
Out4 : out STD_LOGIC;
Out5 : out STD_LOGIC;
Out6 : out STD_LOGIC;
Out7 : out STD_LOGIC);
end component;
--**********************************************************************************
constant Dos_cinco_cinco : STD_LOGIC_VECTOR(7 downto 0) := "11111111";
constant Cin : STD_LOGIC := '0';
--**********************************************************************************
signal Cero_A : STD_LOGIC_VECTOR(7 downto 0);
signal Sal_A : STD_LOGIC;
signal Suma_A : STD_LOGIC_VECTOR(7 downto 0);

signal Cero_B : STD_LOGIC_VECTOR(7 downto 0);
signal Sal_B : STD_LOGIC;
signal Suma_B : STD_LOGIC_VECTOR(7 downto 0);

signal Resta_B : STD_LOGIC_VECTOR(7 downto 0);

signal Cout : STD_LOGIC;
signal S_A : STD_LOGIC_VECTOR(7 downto 0);

signal Op_B : STD_LOGIC_VECTOR(7 downto 0);
signal Cout_B : STD_LOGIC;
signal S_B : STD_LOGIC_VECTOR(7 downto 0);

signal And_A : STD_LOGIC_VECTOR(7 downto 0);

signal Inhividor : STD_LOGIC;
--**********************************************************************************
begin
--**********************************************************************************
Inhividor <= (Sal_A) or (Sal_B);
```

```
--*************************************************************************
U0 : Cero port map(A(0),A(1),A(2),A(3),A(4),A(5),A(6),A(7)
,Cero_A(0),Cero_A(1),Cero_A(2),Cero_A(3)
,Cero_A(4),Cero_A(5),Cero_A(6),Cero_A(7),Sal_A);

U1 : Cero port map(B(0),B(1),B(2),B(3),B(4),B(5),B(6),B(7)
,Cero_B(0),Cero_B(1),Cero_B(2),Cero_B(3)
,Cero_B(4),Cero_B(5),Cero_B(6),Cero_B(7),Sal_B);

U2 : Logaritmo port map(Cero_A(0),Cero_A(1),Cero_A(2),Cero_A(3)
,Cero_A(4),Cero_A(5),Cero_A(6),Cero_A(7)
,Suma_A(0),Suma_A(1),Suma_A(2),Suma_A(3)
,Suma_A(4),Suma_A(5),Suma_A(6),Suma_A(7));

U3 : Logaritmo port map(Cero_B(0),Cero_B(1),Cero_B(2),Cero_B(3)
,Cero_B(4),Cero_B(5),Cero_B(6),Cero_B(7)
,Suma_B(0),Suma_B(1),Suma_B(2),Suma_B(3)
,Suma_B(4),Suma_B(5),Suma_B(6),Suma_B(7));

U4 : Restador port map(Dos_cinco_cinco,Suma_B,Resta_B);

U5 : Sumador port map(Suma_A,Resta_B,S_A,Cin,Cout);

U6 : Comparador port map(S_A(0),S_A(1),S_A(2),S_A(3),S_A(4),S_A(5)
,S_A(6),S_A(7),Op_B(0),Op_B(1),Op_B(2),Op_B(3)
,Op_B(4),Op_B(5),Op_B(6),Op_B(7));

U7 : Sumador port map(S_A,Op_B,S_B,Cout,Cout_B);

U8 : AntiLogaritmo port  map(S_B(0),S_B(1),S_B(2),S_B(3),S_B(4),S_B(5)
,S_B(6),S_B(7),And_A(0),And_A(1),And_A(2)
,And_A(3),And_A(4),And_A(5),And_A(6),And_A(7));

U9 : Compuerta port map(And_A(0),And_A(1),And_A(2),And_A(3),And_A(4),
And_A(5),And_A(6),And_A(7),Inhividor,D(0),D(1),D(2)
,D(3),D(4),D(5),D(6),D(7));
--*************************************************************************
end Behavioral;
```

Elemento Multiplicador

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Multiplicador is
port ( In_A : in STD_LOGIC_VECTOR(7 downto 0);
In_B : in STD_LOGIC_VECTOR(7 downto 0);
Mul : out STD_LOGIC_VECTOR(7 downto 0));
end Multiplicador;
```

```
architecture Behavioral of Multiplicador is
--*********************************************************************************
component Cero is
    Port (  In0 : in  STD_LOGIC;
In1 : in  STD_LOGIC;
In2 : in  STD_LOGIC;
In3 : in  STD_LOGIC;
In4 : in  STD_LOGIC;
In5 : in  STD_LOGIC;
In6 : in  STD_LOGIC;
In7 : in  STD_LOGIC;

Out0 : out  STD_LOGIC;
Out1 : out  STD_LOGIC;
Out2 : out  STD_LOGIC;
Out3 : out  STD_LOGIC;
Out4 : out  STD_LOGIC;
Out5 : out  STD_LOGIC;
Out6 : out  STD_LOGIC;
Out7 : out  STD_LOGIC;

Sal : out STD_LOGIC);
end component;

component Logaritmo is
    Port (  In0   : in   STD_LOGIC;
In1   : in   STD_LOGIC;
In2   : in   STD_LOGIC;
In3   : in   STD_LOGIC;
In4   : in   STD_LOGIC;
In5   : in   STD_LOGIC;
In6   : in   STD_LOGIC;
In7   : in   STD_LOGIC;

Out0 : out STD_LOGIC;
Out1 : out STD_LOGIC;
Out2 : out STD_LOGIC;
Out3 : out STD_LOGIC;
Out4 : out STD_LOGIC;
Out5 : out STD_LOGIC;
Out6 : out STD_LOGIC;
Out7 : out STD_LOGIC);
end component;

component Sumador is
port ( A : in STD_LOGIC_VECTOR(7 downto 0);
B : in STD_LOGIC_VECTOR(7 downto 0);
S : out STD_LOGIC_VECTOR(7 downto 0);
Ci : in STD_LOGIC;
```

```
Co : out STD_LOGIC);
end component;

component Comparador is
    Port (  In0  :  in   STD_LOGIC;
In1  :  in   STD_LOGIC;
In2  :  in   STD_LOGIC;
In3  :  in   STD_LOGIC;
In4  :  in   STD_LOGIC;
In5  :  in   STD_LOGIC;
In6  :  in   STD_LOGIC;
In7  :  in   STD_LOGIC;

Out0 : out STD_LOGIC;
Out1 : out STD_LOGIC;
Out2 : out STD_LOGIC;
Out3 : out STD_LOGIC;
Out4 : out STD_LOGIC;
Out5 : out STD_LOGIC;
Out6 : out STD_LOGIC;
Out7 : out STD_LOGIC);
end component;

component  AntiLogaritmo is
    Port (  In0   : in   STD_LOGIC;
In1   : in   STD_LOGIC;
In2   : in   STD_LOGIC;
In3   : in   STD_LOGIC;
In4   : in   STD_LOGIC;
In5   : in   STD_LOGIC;
In6   : in   STD_LOGIC;
In7   : in   STD_LOGIC;

Out0 : out STD_LOGIC;
Out1 : out STD_LOGIC;
Out2 : out STD_LOGIC;
Out3 : out STD_LOGIC;
Out4 : out STD_LOGIC;
Out5 : out STD_LOGIC;
Out6 : out STD_LOGIC;
Out7 : out STD_LOGIC);
end component;


component Compuerta is
port (
In0  :  in   STD_LOGIC;
In1  :  in   STD_LOGIC;
In2  :  in   STD_LOGIC;
In3  :  in   STD_LOGIC;
In4  :  in   STD_LOGIC;
```

```
In5  :  in   STD_LOGIC;
In6  :  in   STD_LOGIC;
In7  :  in   STD_LOGIC;

Inhividor: in STD_LOGIC;

Out0 : out STD_LOGIC;
Out1 : out STD_LOGIC;
Out2 : out STD_LOGIC;
Out3 : out STD_LOGIC;
Out4 : out STD_LOGIC;
Out5 : out STD_LOGIC;
Out6 : out STD_LOGIC;
Out7 : out STD_LOGIC);
end component;

--*********************************************************************************
constant Dos_cinco_cinco : STD_LOGIC_VECTOR(7 downto 0) := "11111111";
constant Cin : STD_LOGIC := '0';
--*********************************************************************************
signal Cero_A : STD_LOGIC_VECTOR(7 downto 0);
signal Sal_A : STD_LOGIC;
signal Suma_A : STD_LOGIC_VECTOR(7 downto 0);

signal Cero_B : STD_LOGIC_VECTOR(7 downto 0);
signal Sal_B : STD_LOGIC;
signal Suma_B : STD_LOGIC_VECTOR(7 downto 0);


signal Cout : STD_LOGIC;
signal S_A : STD_LOGIC_VECTOR(7 downto 0);

signal Op_B : STD_LOGIC_VECTOR(7 downto 0);
signal Cout_B : STD_LOGIC;
signal S_B : STD_LOGIC_VECTOR(7 downto 0);

signal And_A : STD_LOGIC_VECTOR(7 downto 0);

signal Inhividor : STD_LOGIC;
--*********************************************************************************
begin
--*********************************************************************************
Inhividor <= Sal_A or Sal_B;
--*********************************************************************************
U0 : Cero port map(In_A(0),In_A(1),In_A(2),In_A(3),In_A(4),In_A(5)
,In_A(6),In_A(7),Cero_A(0),Cero_A(1),Cero_A(2),Cero_A(3)
,Cero_A(4),Cero_A(5),Cero_A(6),Cero_A(7),Sal_A);

U1 : Cero port map(In_B(0),In_B(1),In_B(2),In_B(3),In_B(4),In_B(5)
,In_B(6),In_B(7),Cero_B(0),Cero_B(1),Cero_B(2),Cero_B(3)
,Cero_B(4),Cero_B(5),Cero_B(6),Cero_B(7),Sal_B);
```

```
U2 : Logaritmo port map(Cero_A(0),Cero_A(1),Cero_A(2),Cero_A(3),Cero_A(4)
,Cero_A(5),Cero_A(6),Cero_A(7),Suma_A(0),Suma_A(1)
,Suma_A(2),Suma_A(3),Suma_A(4),Suma_A(5),Suma_A(6)
,Suma_A(7));

U3 : Logaritmo port map(Cero_B(0),Cero_B(1),Cero_B(2),Cero_B(3),Cero_B(4)
,Cero_B(5),Cero_B(6),Cero_B(7),Suma_B(0),Suma_B(1)
,Suma_B(2),Suma_B(3),Suma_B(4),Suma_B(5),Suma_B(6)
,Suma_B(7));

U4 : Sumador port map(Suma_A,Suma_B,S_A,Cin,Cout);

U5 : Comparador port map(S_A(0),S_A(1),S_A(2),S_A(3),S_A(4),S_A(5),S_A(6)
,S_A(7),Op_B(0),Op_B(1),Op_B(2),Op_B(3),Op_B(4),Op_B(5)
,Op_B(6),Op_B(7));

U6 : Sumador port map(S_A,Op_B,S_B,Cout,Cout_B);

U7 : AntiLogaritmo port  map(S_B(0),S_B(1),S_B(2),S_B(3),S_B(4),S_B(5),S_B(6)
,S_B(7),And_A(0),And_A(1),And_A(2),And_A(3),And_A(4)
,And_A(5),And_A(6),And_A(7));

U8 : Compuerta port map(And_A(0),And_A(1),And_A(2),And_A(3),And_A(4),And_A(5)
,And_A(6),And_A(7),Inhividor,Mul(0),Mul(1),Mul(2),Mul(3)
,Mul(4),Mul(5),Mul(6),Mul(7));

--*********************************************************************************
--*********************************************************************************
end Behavioral;
```

Elemento Sumador-Restador

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Sumador_Restador is
    Port (  A  :  in  STD_LOGIC_VECTOR(7 downto 0);
B  :  in  STD_LOGIC_VECTOR(7 downto 0);
S  :  out STD_LOGIC_VECTOR(7 downto 0));
end Sumador_Restador;

architecture Behavioral of Sumador_Restador is
begin
--*********************************************************************************
S(0) <= A(0) xor B(0);
S(1) <= A(1) xor B(1);
S(2) <= A(2) xor B(2);
S(3) <= A(3) xor B(3);
```

```
S(4) <= A(4) xor B(4);
S(5) <= A(5) xor B(5);
S(6) <= A(6) xor B(6);
S(7) <= A(7) xor B(7);
--**********************************************************************************
end Behavioral;
```

Wavelet
wavelet analisis

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Wavelets_Analisis is
    Port (  Registro_1  :  in    STD_LOGIC_VECTOR(7  downto  0);
Registro_2  :  in    STD_LOGIC_VECTOR(7  downto  0);
Registro_3  :  in    STD_LOGIC_VECTOR(7  downto  0);
Registro_4 :  in    STD_LOGIC_VECTOR(7  downto  0);
Registro_5  :  in    STD_LOGIC_VECTOR(7  downto  0);
Registro_6  :  in    STD_LOGIC_VECTOR(7  downto  0);
Registro_7  :  in    STD_LOGIC_VECTOR(7  downto  0);
Registro_8  :  in    STD_LOGIC_VECTOR(7  downto  0);
Registro_9  :  in    STD_LOGIC_VECTOR(7  downto  0);
Registro_10   :  in    STD_LOGIC_VECTOR(7  downto  0);
Registro_11  :  in    STD_LOGIC_VECTOR(7  downto  0);
Registro_12  :  in    STD_LOGIC_VECTOR(7  downto  0);
Registro_13   :  in    STD_LOGIC_VECTOR(7  downto  0);
Registro_14   :  in    STD_LOGIC_VECTOR(7  downto  0);
Registro_15  :  in    STD_LOGIC_VECTOR(7  downto  0);
Registro_16  :  in    STD_LOGIC_VECTOR(7  downto  0);

FPBFPA_1 : out STD_LOGIC_VECTOR(7  downto  0);
FPBFPA_2 : out STD_LOGIC_VECTOR(7  downto  0);
FPBFPA_3 : out STD_LOGIC_VECTOR(7  downto  0);
FPBFPA_4 : out STD_LOGIC_VECTOR(7  downto  0);
FPBFPA_5 : out STD_LOGIC_VECTOR(7  downto  0);

FPBFPB_1 : out STD_LOGIC_VECTOR(7  downto  0);
FPBFPB_2 : out STD_LOGIC_VECTOR(7  downto  0);
FPBFPB_3 : out STD_LOGIC_VECTOR(7  downto  0);
FPBFPB_4 : out STD_LOGIC_VECTOR(7  downto  0);
FPBFPB_5 : out STD_LOGIC_VECTOR(7  downto  0);

FPAFPA_1 : out STD_LOGIC_VECTOR(7  downto  0);
FPAFPA_2 : out STD_LOGIC_VECTOR(7  downto  0);
FPAFPA_3 : out STD_LOGIC_VECTOR(7  downto  0);
FPAFPA_4 : out STD_LOGIC_VECTOR(7  downto  0);
FPAFPA_5 : out STD_LOGIC_VECTOR(7  downto  0);

FPAFPB_1 : out STD_LOGIC_VECTOR(7  downto  0);
FPAFPB_2 : out STD_LOGIC_VECTOR(7  downto  0);
FPAFPB_3 : out STD_LOGIC_VECTOR(7  downto  0);
FPAFPB_4 : out STD_LOGIC_VECTOR(7  downto  0);
FPAFPB_5 : out STD_LOGIC_VECTOR(7  downto  0));
end Wavelets_Analisis;
--**********************************************************************************
```

```
architecture Behavioral of Wavelets_Analisis is
--*********************************************************************************
signal Cero : STD_LOGIC_VECTOR(7 downto 0):="00000000";

signal FPA_1 : STD_LOGIC_VECTOR(7 downto 0):="00000000";
signal FPA_2 : STD_LOGIC_VECTOR(7 downto 0):="00000000";
signal FPA_3 : STD_LOGIC_VECTOR(7 downto 0):="00000000";
signal FPA_4 : STD_LOGIC_VECTOR(7 downto 0):="00000000";
signal FPA_5 : STD_LOGIC_VECTOR(7 downto 0):="00000000";
signal FPA_6 : STD_LOGIC_VECTOR(7 downto 0):="00000000";
signal FPA_7 : STD_LOGIC_VECTOR(7 downto 0):="00000000";
signal FPA_8 : STD_LOGIC_VECTOR(7 downto 0):="00000000";
signal FPA_9 : STD_LOGIC_VECTOR(7 downto 0):="00000000";
--*********************************************************************************
begin
--*********************************************************************************
--*********************************************************************************
--*********************************************************************************
--*********************************************************************************
FPBFPB_1 <= Registro_1;
FPBFPB_2 <= Registro_9;
FPBFPB_3 <= Cero;
FPBFPB_4 <= Registro_7;
FPBFPB_5 <= Registro_15;
--*********************************************************************************
--*********************************************************************************
FPBFPA_1 <= Registro_1;
FPBFPA_2 <= Registro_5 xor Registro_9;
FPBFPA_3 <= Registro_13;
FPBFPA_4 <= Registro_3 xor Registro_7;
FPBFPA_5 <= Registro_11 xor Registro_15;
--*********************************************************************************
--*********************************************************************************
FPA_1 <= Registro_1;
FPA_2 <= Registro_2 xor Registro_3;
FPA_3 <= Registro_4 xor Registro_5;
FPA_4 <= Registro_6 xor Registro_7;
FPA_5 <= Registro_8 xor Registro_9;
FPA_6 <= Registro_10 xor Registro_11;
FPA_7 <= Registro_12 xor Registro_13;
FPA_8 <= Registro_14 xor Registro_15;
FPA_9 <= Registro_16;
--*********************************************************************************
--*********************************************************************************
FPAFPB_1 <= FPA_1;
FPAFPB_2 <= FPA_5;
FPAFPB_3 <= FPA_9;
FPAFPB_4 <= FPA_4;
FPAFPB_5 <= FPA_8;
--*********************************************************************************
--*********************************************************************************
```

```
FPAFPA_1 <= FPA_1;
FPAFPA_2 <= FPA_3  xor FPA_5;
FPAFPA_3 <= FPA_7 xor FPA_9;
FPAFPA_4 <= FPA_2 xor FPA_4;
FPAFPA_5 <= FPA_6 xor FPA_8;
--*************************************************************************
--*************************************************************************
--*************************************************************************
--*************************************************************************
end Behavioral;
```

Wavelet sintesis

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Wavelets_Sintesis is
    Port (  Registro_1  :  out    STD_LOGIC_VECTOR(7  downto  0);
Registro_2  :  out    STD_LOGIC_VECTOR(7  downto  0);
Registro_3  :  out    STD_LOGIC_VECTOR(7  downto  0);
Registro_4 :  out    STD_LOGIC_VECTOR(7  downto  0);
Registro_5  :  out    STD_LOGIC_VECTOR(7  downto  0);
Registro_6  :  out    STD_LOGIC_VECTOR(7  downto  0);
Registro_7  :  out    STD_LOGIC_VECTOR(7  downto  0);
Registro_8  :  out    STD_LOGIC_VECTOR(7  downto  0);
Registro_9  :  out    STD_LOGIC_VECTOR(7  downto  0);
Registro_10  :  out    STD_LOGIC_VECTOR(7  downto  0);
Registro_11  :  out    STD_LOGIC_VECTOR(7  downto  0);
Registro_12  :  out    STD_LOGIC_VECTOR(7  downto  0);
Registro_13   :  out    STD_LOGIC_VECTOR(7  downto  0);
Registro_14  :  out    STD_LOGIC_VECTOR(7  downto  0);
Registro_15  :  out    STD_LOGIC_VECTOR(7  downto  0);
Registro_16  :  out    STD_LOGIC_VECTOR(7  downto  0);

FPBFPA_1 : in STD_LOGIC_VECTOR(7  downto  0);
FPBFPA_2 : in STD_LOGIC_VECTOR(7  downto  0);
FPBFPA_3 : in STD_LOGIC_VECTOR(7  downto  0);
FPBFPA_4 : in STD_LOGIC_VECTOR(7  downto  0);
FPBFPA_5 : in STD_LOGIC_VECTOR(7  downto  0);

FPBFPB_1 : in STD_LOGIC_VECTOR(7  downto  0);
FPBFPB_2 : in STD_LOGIC_VECTOR(7  downto  0);
FPBFPB_3 : in STD_LOGIC_VECTOR(7  downto  0);
FPBFPB_4 : in STD_LOGIC_VECTOR(7  downto  0);
```

```
FPBFPB_5 : in STD_LOGIC_VECTOR(7  downto  0);

FPAFPA_1 : in STD_LOGIC_VECTOR(7  downto  0);
FPAFPA_2 : in STD_LOGIC_VECTOR(7  downto  0);
FPAFPA_3 : in STD_LOGIC_VECTOR(7  downto  0);
FPAFPA_4 : in STD_LOGIC_VECTOR(7  downto  0);
FPAFPA_5 : in STD_LOGIC_VECTOR(7  downto  0);

FPAFPB_1 : in STD_LOGIC_VECTOR(7  downto  0);
FPAFPB_2 : in STD_LOGIC_VECTOR(7  downto  0);
FPAFPB_3 : in STD_LOGIC_VECTOR(7  downto  0);
FPAFPB_4 : in STD_LOGIC_VECTOR(7  downto  0);
FPAFPB_5 : in STD_LOGIC_VECTOR(7  downto  0));
end Wavelets_Sintesis;

architecture Behavioral of Wavelets_Sintesis is
--*****************************************************************************
signal FPA_1 : STD_LOGIC_VECTOR(7  downto  0):="00000000";
signal FPA_2 : STD_LOGIC_VECTOR(7  downto  0):="00000000";
signal FPA_3 : STD_LOGIC_VECTOR(7  downto  0):="00000000";
signal FPA_4 : STD_LOGIC_VECTOR(7  downto  0):="00000000";
signal FPA_5 : STD_LOGIC_VECTOR(7  downto  0):="00000000";
signal FPA_6 : STD_LOGIC_VECTOR(7  downto  0):="00000000";
signal FPA_7 : STD_LOGIC_VECTOR(7  downto  0):="00000000";
signal FPA_8 : STD_LOGIC_VECTOR(7  downto  0):="00000000";
signal FPA_9 : STD_LOGIC_VECTOR(7  downto  0):="00000000";
signal FPA_10 : STD_LOGIC_VECTOR(7  downto  0):="00000000";


signal FPB_1 : STD_LOGIC_VECTOR(7  downto  0):="00000000";
signal FPB_2 : STD_LOGIC_VECTOR(7  downto  0):="00000000";
signal FPB_3 : STD_LOGIC_VECTOR(7  downto  0):="00000000";
signal FPB_4 : STD_LOGIC_VECTOR(7  downto  0):="00000000";
signal FPB_5 : STD_LOGIC_VECTOR(7  downto  0):="00000000";
signal FPB_6 : STD_LOGIC_VECTOR(7  downto  0):="00000000";
signal FPB_7 : STD_LOGIC_VECTOR(7  downto  0):="00000000";
signal FPB_8 : STD_LOGIC_VECTOR(7  downto  0):="00000000";
signal FPB_9 : STD_LOGIC_VECTOR(7  downto  0):="00000000";
signal FPB_10 : STD_LOGIC_VECTOR(7  downto  0):="00000000";
--*****************************************************************************
begin
--*****************************************************************************
FPA_1 <= FPBFPB_1 xor FPBFPA_1;
FPA_2 <= FPBFPB_1;
FPA_3 <= FPBFPB_2 xor FPBFPA_2;
FPA_4 <= FPBFPB_2;
FPA_5 <= FPBFPB_3 xor FPBFPA_3;
FPA_6 <= FPBFPB_3;
FPA_7 <= FPBFPB_4 xor FPBFPA_4;
FPA_8 <= FPBFPB_4;
FPA_9 <= FPBFPB_5 xor FPBFPA_5;
```

```
FPA_10 <= FPBFPB_5;
--*****************************************************************************
FPB_1 <= FPAFPB_1 xor FPAFPA_1;
FPB_2 <= FPAFPB_1;
FPB_3 <= FPAFPB_2 xor FPAFPA_2;
FPB_4 <= FPAFPB_2;
FPB_5 <= FPAFPB_3 xor FPAFPA_3;
FPB_6 <= FPAFPB_3;
FPB_7 <= FPAFPB_4 xor FPAFPA_4;
FPB_8 <= FPAFPB_4;
FPB_9 <= FPAFPB_5 xor FPAFPA_5;
FPB_10 <= FPAFPB_5;
--*****************************************************************************
Registro_1 <= FPA_2;
Registro_2 <= FPA_7 xor FPB_7;
Registro_3 <= FPA_7;
Registro_4 <= FPA_3 xor FPB_3;
Registro_5 <= FPA_3;
Registro_6 <= FPA_8 xor FPB_8;
Registro_7 <= FPA_8;
Registro_8 <= FPA_4 xor FPB_4;
Registro_9 <= FPA_4;
Registro_10 <= FPA_9 xor FPB_9;
Registro_11 <= FPA_9;
Registro_12 <= FPA_5 xor FPB_5;
Registro_13 <= FPA_5;
Registro_14 <= FPA_10 xor FPB_10;
Registro_15 <= FPA_10;
Registro_16 <= FPB_6;
--*****************************************************************************
end Behavioral;
```

Lectura de la Compresion

```
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Lectura_de_la_compresion is
    Port (  Limite_L  :  in   STD_LOGIC_VECTOR(11 downto 0);
Cum_Count_x  :  in   STD_LOGIC_VECTOR(11 downto 0);
Cum_Count_x_1  :  in   STD_LOGIC_VECTOR(11 downto 0);
Limite_U  :  in   STD_LOGIC_VECTOR(11 downto 0);
En  :  in   STD_LOGIC;
CLK  :  in   STD_LOGIC;
  --*******************************
  OE3 : out STD_LOGIC_VECTOR(3 downto 0);
  OIgualdad : out STD_LOGIC_VECTOR(3 downto 0);
  OAdvil :  out STD_LOGIC_VECTOR(11 downto 0);
  OInicio : out STD_LOGIC;
  OEFlag : out STD_LOGIC;
  ORUMDir : out STD_LOGIC_VECTOR(5 downto 0):="000000";
  ORUM : out STD_LOGIC;
  --*******************************
  Limite_L0 :  out   STD_LOGIC_VECTOR(11 downto 0);
  Limite_U0 :  out   STD_LOGIC_VECTOR(11 downto 0);
  Oflag : out STD_LOGIC;
  MO : out STD_LOGIC);
end Lectura_de_la_compresion;

architecture Behavioral of Lectura_de_la_compresion is
--*************************************************
component Compresor is
    Port ( Limite_L  :  in   STD_LOGIC_VECTOR(11 downto 0);
           Cum_Count_x  :  in   STD_LOGIC_VECTOR(11 downto 0);
           Cum_Count_x_1  :  in   STD_LOGIC_VECTOR(11 downto 0);
           Limite_U  :  in   STD_LOGIC_VECTOR(11 downto 0);
           En  :  in   STD_LOGIC;
           CLK  :  in   STD_LOGIC;
  RUM : in STD_LOGIC:='0';
  RUMDir : in STD_LOGIC_VECTOR (5 downto 0):="000000";
  --*******************************
  OE3 : out STD_LOGIC_VECTOR(3 downto 0);
  OIgualdad : out STD_LOGIC_VECTOR(3 downto 0);
  OAdvil :  out STD_LOGIC_VECTOR(11 downto 0);
  OInicio : out STD_LOGIC;
  OEFalg : out STD_LOGIC;
  --*******************************
  Limite_L0 :  out   STD_LOGIC_VECTOR(11 downto 0);
  Limite_U0 :  out   STD_LOGIC_VECTOR(11 downto 0);
  Oflag : out STD_LOGIC;
  MO : out STD_LOGIC);
```

```
end component;
--*************************************************
signal RUM : STD_LOGIC:='0';
signal RUMDir : STD_LOGIC_VECTOR(5 downto 0):="000000";
signal ROflag : STD_LOGIC;
signal ROEFlag : STD_LOGIC;
--*************************************************
type Estados is(d0,d1,d2,d3,d4,d5,d6,
d7,d8,d9,d10);
signal Edo_Presente, Edo_Futuro : Estados;
--***********************************
begin
--***********************************
Oflag <= ROflag;
ORUMDir <= RUMDir;
OEFlag <= ROEFlag;
ORUM <= RUM;
--*************************************************
UCompresor: Compresor port map ( Limite_L,Cum_Count_x,Cum_Count_x_1,Limite_U,En,
CLK,RUM,RUMDir,OE3,OIgualdad,
OAdvil,OInicio,ROEFlag,Limite_L0,Limite_U0,
ROflag,MO);
--*************************************************
Lectura : process(ROFlag,Edo_Presente)
begin
--*********************************
if(ROflag='1')then
--*********************************
case Edo_Presente is
when d0 =>RUMDir <= "000000";
RUM <= '1';
Edo_Futuro <= d1;

when d1 =>RUMDir <= "000001";
RUM <= '1';
Edo_Futuro <= d2;

when d2 =>RUMDir <= "000010";
RUM <= '1';
Edo_Futuro <= d3;

when d3 =>RUMDir <= "000011";
RUM <= '1';
Edo_Futuro <= d4;

when d4 =>RUMDir <= "000100";
RUM <= '1';
Edo_Futuro <= d5;

when d5 =>RUMDir <= "000101";
RUM <= '1';
```

```
Edo_Futuro <= d6;

when d6 =>RUMDir <= "000110";
RUM <= '1';
Edo_Futuro <= d7;

when d7 =>RUMDir <= "000111";
RUM <= '1';
Edo_Futuro <= d8;

when d8 =>RUMDir <= "001000";
RUM <= '1';
Edo_Futuro <= d9;

when d9 =>RUMDir <= "001001";
RUM <= '1';
Edo_Futuro <= d10;

when d10 =>RUMDir <= "001010";
RUM <= '1';

end case;
--**********************************
else
RUM <= '0';

end if;
end process;
--*****************************************************
--**********************************
Control: process(CLK)
begin
--**********************************
if(CLK'event and CLK='1')then
Edo_Presente <= Edo_Futuro;
end if;
--**********************************
end process Control;
--**********************************
end Behavioral;
```

Compresor

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Compresor is
    Port ( Limite_L  :  in   STD_LOGIC_VECTOR(11 downto 0);
           Cum_Count_x  :  in   STD_LOGIC_VECTOR(11 downto 0);
           Cum_Count_x_1  :  in   STD_LOGIC_VECTOR(11 downto 0);
           Limite_U  :  in   STD_LOGIC_VECTOR(11 downto 0);
           En  :  in   STD_LOGIC;
           CLK  :  in   STD_LOGIC;
  RUM : in STD_LOGIC:='0';
  RUMDir : in STD_LOGIC_VECTOR (5 downto 0):="000000";
  --*********************************
  OE3 : out STD_LOGIC_VECTOR(3 downto 0);
  OIgualdad : out STD_LOGIC_VECTOR(3 downto 0);
  OAdvil :  out STD_LOGIC_VECTOR(11 downto 0);
  OInicio : out STD_LOGIC;
  OEFalg : out STD_LOGIC;
  --*********************************
  Limite_L0 :  out   STD_LOGIC_VECTOR(11 downto 0);
  Limite_U0 :  out   STD_LOGIC_VECTOR(11 downto 0);
  Oflag : out STD_LOGIC;
  MO : out STD_LOGIC);
end Compresor;

architecture Behavioral of Compresor is
--************************************
component Calculo_del_Intervalo is
Port (    Limite_L  :  in   STD_LOGIC_VECTOR(11 downto 0);
           Limite_U  :  in   STD_LOGIC_VECTOR(11  downto 0);
  Cum_Count_x : in   STD_LOGIC_VECTOR(11  downto 0);
  Cum_Count_x_1 : in   STD_LOGIC_VECTOR(11  downto 0);
  Advil : out STD_LOGIC_VECTOR(11  downto 0);
  E3 :  out   STD_LOGIC_VECTOR(3  downto 0);
  Igualdad :  out   STD_LOGIC_VECTOR(3  downto 0);
  Limite_l0 : out   STD_LOGIC_VECTOR(11  downto 0);
  Limite_u0 : out   STD_LOGIC_VECTOR(11  downto 0));
end component;

component Control is
Port (  OAcc  : in  STD_LOGIC_VECTOR (11 downto 0);
           En  : in  STD_LOGIC;
           CLK  : in  STD_LOGIC;
           Flag  : in  STD_LOGIC;
  --**********************
           Inicio : out  STD_LOGIC:='0';
           IAcc  : out  STD_LOGIC_VECTOR (11 downto 0):="000000000000";
```

```
           Oflag  : out  STD_LOGIC:='0');
end component;

component Etiqueta is
Port (
Inicio : in STD_LOGIC;
CLK : in STD_LOGIC;
Igual  :  in   STD_LOGIC_VECTOR (11 downto 0);
E3  :  in   STD_LOGIC_VECTOR (11 downto 0);
Advil  :  in   STD_LOGIC_VECTOR (11 downto 0);
IAcc : in STD_LOGIC_VECTOR (11 downto 0);
RUM : in STD_LOGIC;
RUMDir : in STD_LOGIC_VECTOR (5 downto 0);
--*********************************************
OAcc : out STD_LOGIC_VECTOR (11 downto 0):="000000000000";
MO : out STD_LOGIC:='0';
Flag : out STD_LOGIC:='0');
end component;
--***********************************
signal Advil : STD_LOGIC_VECTOR(11  downto 0);
signal E3 :    STD_LOGIC_VECTOR(3  downto 0);
signal Igualdad :    STD_LOGIC_VECTOR(3  downto 0);
--*********************
signal Inicio : STD_LOGIC;
signal Igual   :    STD_LOGIC_VECTOR (11 downto 0);
signal IE3 :    STD_LOGIC_VECTOR(11  downto 0);
signal IAcc : STD_LOGIC_VECTOR (11 downto 0);
signal OAcc : STD_LOGIC_VECTOR (11 downto 0):="000000000000";
signal Flag : STD_LOGIC:='0';
--***********************************
begin
--***********************************
OE3(0) <= IE3(0);
OE3(1) <= IE3(1);
OE3(2) <= IE3(2);
OE3(3) <= IE3(3);

OIgualdad(0) <= Igual(0);
OIgualdad(1) <= Igual(1);
OIgualdad(2) <= Igual(2);
OIgualdad(3) <= Igual(3);

OAdvil <= Advil;
OInicio <= Inicio;
OEFalg <= Flag;
--***********************************
Igual(0) <= Igualdad(0);
Igual(1) <= Igualdad(1);
Igual(2) <= Igualdad(2);
Igual(3) <= Igualdad(3);
Igual(4) <= '0';
```

```
Igual(5) <= '0';
Igual(6) <= '0';
Igual(7) <= '0';
Igual(8) <= '0';
Igual(9) <= '0';
Igual(10) <= '0';
Igual(11) <= '0';

IE3(0) <= E3(0);
IE3(1) <= E3(1);
IE3(2) <= E3(2);
IE3(3) <= E3(3);
IE3(4) <= '0';
IE3(5) <= '0';
IE3(6) <= '0';
IE3(7) <= '0';
IE3(8) <= '0';
IE3(9) <= '0';
IE3(10) <= '0';
IE3(11) <= '0';
--************************************
UCalculo_del_Intervalo : Calculo_del_Intervalo
port map(Limite_L,Limite_U,Cum_Count_x,Cum_Count_x_1,
Advil,E3,Igualdad,Limite_l0,Limite_u0);
UControl: Control port map(OAcc,En,CLK,Flag,Inicio,
IAcc,Oflag);
UEtiqueta : Etiqueta port map(Inicio,CLK,Igual,
IE3,Advil,IAcc,RUM,RUMDir,OAcc,MO,Flag);
--************************************
end Behavioral;
```

Calculo del Intervalo

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Calculo_del_Intervalo is
Port (    Limite_L  :  in   STD_LOGIC_VECTOR(11 downto 0);
           Limite_U  :  in   STD_LOGIC_VECTOR(11  downto 0);
  Cum_Count_x : in   STD_LOGIC_VECTOR(11  downto 0);
  Cum_Count_x_1 : in   STD_LOGIC_VECTOR(11  downto 0);
  Advil : out STD_LOGIC_VECTOR(11  downto 0);
  E3 :  out   STD_LOGIC_VECTOR(3  downto 0);
  Igualdad :  out   STD_LOGIC_VECTOR(3  downto 0);
  Limite_l0 : out   STD_LOGIC_VECTOR(11  downto 0);
  Limite_u0 : out   STD_LOGIC_VECTOR(11  downto 0));
end Calculo_del_Intervalo;

architecture Behavioral of Calculo_del_Intervalo is
--*********************************************************************************
component Palabras is
Port (  Nuevo_limite_l : in  STD_LOGIC_VECTOR  (11 downto 0);
Nuevo_limite_u : in  STD_LOGIC_VECTOR  (11 downto 0);
Palabra_Igualdad  : out  STD_LOGIC_VECTOR (11 downto 0);
Palabra_E3  : out  STD_LOGIC_VECTOR (11 downto 0));
end component;


component DesplazamientoL is
    Port (  I  :  in   STD_LOGIC_VECTOR(11 downto  0);
S  :  in   STD_LOGIC_VECTOR(5  downto  0);
O : out STD_LOGIC_VECTOR(11 downto  0));
end component;


component DesplazamientoU is
    Port (  I  :  in   STD_LOGIC_VECTOR(11 downto  0);
S  :  in   STD_LOGIC_VECTOR(5  downto  0);
O : out STD_LOGIC_VECTOR(11 downto  0));
end component;


component Indicador_Desplazamientos is
    Port (  Palabra_Igualdad  : in   STD_LOGIC_VECTOR(11  downto 0);
Palabra_E3   : in   STD_LOGIC_VECTOR(11  downto  0);
E3 : out   STD_LOGIC_VECTOR(3  downto  0);
Igualdad : out   STD_LOGIC_VECTOR(3  downto  0);
S : out   STD_LOGIC_VECTOR(5  downto  0));
end component;
```

```
component Calculo_Limite is
    Port (  Limite_L  :  in   STD_LOGIC_VECTOR(11 downto 0);
Limite_U :  in   STD_LOGIC_VECTOR(11 downto 0);
Cum_Count_x : in STD_LOGIC_VECTOR(11 downto 0);
Cum_Count_x_1 : in STD_LOGIC_VECTOR(11 downto 0);
Advil : out STD_LOGIC_VECTOR(11 downto 0);
Nuevo_limite_l : out STD_LOGIC_VECTOR(11 downto 0);
Nuevo_limite_u : out STD_LOGIC_VECTOR(11 downto 0));
end component;

--*************************************************************************
signal Nuevo_limite_l : STD_LOGIC_VECTOR(11  downto  0);
signal Nuevo_limite_u : STD_LOGIC_VECTOR(11  downto  0);

signal Palabra_Igualdad : STD_LOGIC_VECTOR(11  downto  0);
signal Palabra_E3 : STD_LOGIC_VECTOR(11  downto  0);
signal Seleccion : STD_LOGIC_VECTOR(5 downto  0);
--*************************************************************************
begin
--*************************************************************************
U0 :  Calculo_Limite port map(Limite_l,Limite_u,Cum_Count_x,Cum_Count_x_1,
Advil,Nuevo_limite_l,Nuevo_limite_u);
U1 : Palabras port map (Nuevo_limite_l,Nuevo_limite_u,Palabra_Igualdad,
Palabra_E3);
U2 : Indicador_Desplazamientos port map (Palabra_Igualdad,Palabra_E3,E3,
Igualdad,Seleccion);
U3 : DesplazamientoL port  map (Nuevo_limite_l,Seleccion,Limite_l0);
U4 : DesplazamientoU port  map (Nuevo_limite_u,Seleccion,Limite_u0);
--*************************************************************************

--*************************************************************************
end Behavioral;
```

Calculo_ Limite

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity Calculo_Limite is
    Port (  Limite_L  :  in   STD_LOGIC_VECTOR(11 downto 0);
Limite_U :  in   STD_LOGIC_VECTOR(11 downto 0);
Cum_Count_x : in STD_LOGIC_VECTOR(11 downto 0);
Cum_Count_x_1 : in STD_LOGIC_VECTOR(11 downto 0);
--*********************************************************
Advil : out STD_LOGIC_VECTOR(11 downto 0);
--*********************************************************
Nuevo_limite_l : out STD_LOGIC_VECTOR(11 downto 0);
Nuevo_limite_u : out STD_LOGIC_VECTOR(11 downto 0));
end Calculo_Limite;

architecture Behavioral of Calculo_Limite is
--*************************************************************************
signal lim_I_L : UNSIGNED(11 downto 0):="000000000000";
signal lim_I_U : UNSIGNED(11 downto 0):="000000000000";

signal C_C_X : UNSIGNED(11 downto 0):="000000000000";
signal C_C_X_1 : UNSIGNED(11 downto 0):="000000000000";

signal Auxiliarlim_I_L : UNSIGNED(15 downto 0):="0000000000000000";
signal Auxiliarlim_I_U : UNSIGNED(15 downto 0):="0000000000000000";
signal Auxiliar_x : UNSIGNED(15 downto 0):="0000000000000000";
signal Auxiliar_x_1 : UNSIGNED(15 downto 0):="0000000000000000";

signal Mul_L : UNSIGNED(31 downto 0):="00000000000000000000000000000000";
signal Mul_U : UNSIGNED(31 downto 0):="00000000000000000000000000000000";

signal Div_L : UNSIGNED(11 downto 0):="000000000000";
signal Div_U : UNSIGNED(11 downto 0):="000000000000";

signal lim_F_L : UNSIGNED(11 downto 0):="000000000000";
signal lim_F_U : UNSIGNED(11 downto 0):="000000000000";

signal uno : UNSIGNED(11 downto 0):="000000000001";

--*************************************************************************
begin
--*************************************************************************
lim_I_L <= UNSIGNED(Limite_L);
lim_I_U <= UNSIGNED(Limite_U);
C_C_X   <= UNSIGNED(Cum_Count_X);
```

```
C_C_X_1  <= UNSIGNED(Cum_Count_X_1);
--*****************************************************************************

Auxiliar_x(0) <= C_C_X(0);
Auxiliar_x(1) <= C_C_X(1);
Auxiliar_x(2) <= C_C_X(2);
Auxiliar_x(3) <= C_C_X(3);
Auxiliar_x(4) <= C_C_X(4);
Auxiliar_x(5) <= C_C_X(5);
Auxiliar_x(6) <= C_C_X(6);
Auxiliar_x(7) <= C_C_X(7);
Auxiliar_x(8) <= C_C_X(8);
Auxiliar_x(9) <= C_C_X(9);
Auxiliar_x(10) <= C_C_X(10);
Auxiliar_x(11) <= C_C_X(11);
Auxiliar_x(12) <= '0';
Auxiliar_x(13) <= '0';
Auxiliar_x(14) <= '0';
Auxiliar_x(15) <= '0';

Auxiliar_x_1(0) <= C_C_X_1(0);
Auxiliar_x_1(1) <= C_C_X_1(1);
Auxiliar_x_1(2) <= C_C_X_1(2);
Auxiliar_x_1(3) <= C_C_X_1(3);
Auxiliar_x_1(4) <= C_C_X_1(4);
Auxiliar_x_1(5) <= C_C_X_1(5);
Auxiliar_x_1(6) <= C_C_X_1(6);
Auxiliar_x_1(7) <= C_C_X_1(7);
Auxiliar_x_1(8) <= C_C_X_1(8);
Auxiliar_x_1(9) <= C_C_X_1(9);
Auxiliar_x_1(10) <= C_C_X_1(10);
Auxiliar_x_1(11) <= C_C_X_1(11);
Auxiliar_x_1(12) <= '0';
Auxiliar_x_1(13) <= '0';
Auxiliar_x_1(14) <= '0';
Auxiliar_x_1(15) <= '0';

Auxiliarlim_I_L(0) <= lim_I_L(0);
Auxiliarlim_I_L(1) <= lim_I_L(1);
Auxiliarlim_I_L(2) <= lim_I_L(2);
Auxiliarlim_I_L(3) <= lim_I_L(3);
Auxiliarlim_I_L(4) <= lim_I_L(4);
Auxiliarlim_I_L(5) <= lim_I_L(5);
Auxiliarlim_I_L(6) <= lim_I_L(6);
Auxiliarlim_I_L(7) <= lim_I_L(7);
Auxiliarlim_I_L(8) <= lim_I_L(8);
Auxiliarlim_I_L(9) <= lim_I_L(9);
Auxiliarlim_I_L(10) <= lim_I_L(10);
Auxiliarlim_I_L(11) <= lim_I_L(11);
Auxiliarlim_I_L(12) <= '0';
Auxiliarlim_I_L(13) <= '0';
```

```
Auxiliarlim_I_L(14) <= '0';
Auxiliarlim_I_L(15) <= '0';

Auxiliarlim_I_U(0) <= lim_I_U(0);
Auxiliarlim_I_U(1) <= lim_I_U(1);
Auxiliarlim_I_U(2) <= lim_I_U(2);
Auxiliarlim_I_U(3) <= lim_I_U(3);
Auxiliarlim_I_U(4) <= lim_I_U(4);
Auxiliarlim_I_U(5) <= lim_I_U(5);
Auxiliarlim_I_U(6) <= lim_I_U(6);
Auxiliarlim_I_U(7) <= lim_I_U(7);
Auxiliarlim_I_U(8) <= lim_I_U(8);
Auxiliarlim_I_U(9) <= lim_I_U(9);
Auxiliarlim_I_U(10) <= lim_I_U(10);
Auxiliarlim_I_U(11) <= lim_I_U(11);
Auxiliarlim_I_U(12) <= '0';
Auxiliarlim_I_U(13) <= '0';
Auxiliarlim_I_U(14) <= '0';
Auxiliarlim_I_U(15) <= '0';
--************************************************************************
Mul_L <= (Auxiliar_x_1) * ((Auxiliarlim_I_U - Auxiliarlim_I_L) + "0000000000000001" );

Mul_U <= (Auxiliar_x) * ((Auxiliarlim_I_U - Auxiliarlim_I_L) + "0000000000000001" );
--************************************************************************
Div_L(0) <= Mul_L(10);
Div_L(1) <= Mul_L(11);
Div_L(2) <= Mul_L(12);
Div_L(3) <= Mul_L(13);
Div_L(4) <= Mul_L(14);
Div_L(5) <= Mul_L(15);
Div_L(6) <= Mul_L(16);
Div_L(7) <= Mul_L(17);
Div_L(8) <= Mul_L(18);
Div_L(9) <= Mul_L(19);
Div_L(10) <= Mul_L(20);
Div_L(11) <= Mul_L(21);
--************************************************************************
Div_U(0) <= Mul_U(10);
Div_U(1) <= Mul_U(11);
Div_U(2) <= Mul_U(12);
Div_U(3) <= Mul_U(13);
Div_U(4) <= Mul_U(14);
Div_U(5) <= Mul_U(15);
Div_U(6) <= Mul_U(16);
Div_U(7) <= Mul_U(17);
Div_U(8) <= Mul_U(18);
Div_U(9) <= Mul_U(19);
Div_U(10) <= Mul_U(20);
Div_U(11) <= Mul_U(21);
--************************************************************************
lim_F_L <= Div_L + lim_I_L;
```

```
lim_F_U <= (Div_U + lim_I_L)- uno;
--*************************************************************************
-- 0000 0000 0010 1111 1111 1101 --  12285
-- 1 2 4 8      16 32 64 128       256 512 1024 2048   4096  8192 16384
--32768 65536 131072 262144 524288 1048576 2097152 4194304 8388608
-- 0000 0000 0000 0000 0000 1011 --Desplazado
--
--*************************************************************************
Nuevo_limite_l <= STD_LOGIC_VECTOR(lim_F_L);

Nuevo_limite_u <= STD_LOGIC_VECTOR(lim_F_U);
--*************************************************************************
Advil <= STD_LOGIC_VECTOR(lim_F_L);
--*************************************************************************
end Behavioral;
```

Componente Palabras

```
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Palabras is
Port (  Nuevo_limite_l : in  STD_LOGIC_VECTOR  (11 downto 0);
Nuevo_limite_u : in  STD_LOGIC_VECTOR  (11 downto 0);
Palabra_Igualdad  : out  STD_LOGIC_VECTOR (11 downto 0);
Palabra_E3  : out  STD_LOGIC_VECTOR (11 downto 0));
end Palabras;

architecture Behavioral of Palabras is

begin
-- La palabra E3 indica con un 1 logico, la posicion en la que el limite L  es 1
-- y el limite U es 0.
Palabra_E3(11) <= '0';
Palabra_E3(10) <= ( Nuevo_limite_l(10) and  ( not Nuevo_limite_u(10) ) );
Palabra_E3(9) <= ( Nuevo_limite_l(9) and  ( not Nuevo_limite_u(9)  ) );
Palabra_E3(8) <= ( Nuevo_limite_l(8) and  ( not Nuevo_limite_u(8)  ) );
Palabra_E3(7) <= ( Nuevo_limite_l(7) and  ( not Nuevo_limite_u(7)  ) );
Palabra_E3(6) <= ( Nuevo_limite_l(6) and  ( not Nuevo_limite_u(6)  ) );
Palabra_E3(5) <= ( Nuevo_limite_l(5) and  ( not Nuevo_limite_u(5)  ) );
Palabra_E3(4) <= ( Nuevo_limite_l(4) and  ( not Nuevo_limite_u(4)  ) );
Palabra_E3(3) <= ( Nuevo_limite_l(3) and  ( not Nuevo_limite_u(3)  ) );
Palabra_E3(2) <= ( Nuevo_limite_l(2) and  ( not Nuevo_limite_u(2)  ) );
Palabra_E3(1) <= ( Nuevo_limite_l(1) and  ( not Nuevo_limite_u(1)  ) );
Palabra_E3(0) <= ( Nuevo_limite_l(0) and  ( not Nuevo_limite_u(0)  ) );
-- la palabra de Igualdad indica con un 1 logico ,
--la posicion en la ambos limites son iguales
Palabra_Igualdad(11) <= not( Nuevo_limite_l(11) xor  Nuevo_limite_u(11) );
Palabra_Igualdad(10) <= not( Nuevo_limite_l(10) xor  Nuevo_limite_u(10) );
Palabra_Igualdad(9) <= not( Nuevo_limite_l(9) xor  Nuevo_limite_u(9) );
Palabra_Igualdad(8) <= not( Nuevo_limite_l(8) xor  Nuevo_limite_u(8) );
Palabra_Igualdad(7) <= not( Nuevo_limite_l(7) xor  Nuevo_limite_u(7) );
Palabra_Igualdad(6) <= not( Nuevo_limite_l(6) xor  Nuevo_limite_u(6) );
Palabra_Igualdad(5) <= not( Nuevo_limite_l(5) xor  Nuevo_limite_u(5) );
Palabra_Igualdad(4) <= not( Nuevo_limite_l(4) xor  Nuevo_limite_u(4) );
Palabra_Igualdad(3) <= not( Nuevo_limite_l(3) xor  Nuevo_limite_u(3) );
Palabra_Igualdad(2) <= not( Nuevo_limite_l(2) xor  Nuevo_limite_u(2) );
Palabra_Igualdad(1) <= not( Nuevo_limite_l(1) xor  Nuevo_limite_u(1) );
Palabra_Igualdad(0) <= not( Nuevo_limite_l(0) xor  Nuevo_limite_u(0) );

end Behavioral;
```

Indicador de Desplazamientos

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Indicador_Desplazamientos is
Port (  Palabra_Igualdad  : in   STD_LOGIC_VECTOR(11 downto 0);
Palabra_E3   : in   STD_LOGIC_VECTOR(11 downto 0);
E3   : out   STD_LOGIC_VECTOR(3 downto 0);
Igualdad : out   STD_LOGIC_VECTOR(3 downto 0);
S   : out   STD_LOGIC_VECTOR(5 downto 0));
end Indicador_Desplazamientos;

architecture Behavioral of Indicador_Desplazamientos is
--*********************************************************************
--Mascaras para la condicion de igualdad
constant Mascara_1 : STD_LOGIC_VECTOR(11 downto 0) :="10----------";
constant Mascara_2 : STD_LOGIC_VECTOR(11 downto 0) :="110---------";
constant Mascara_3 : STD_LOGIC_VECTOR(11 downto 0) :="1110--------";
constant Mascara_4 : STD_LOGIC_VECTOR(11 downto 0) :="11110-------";
constant Mascara_5 : STD_LOGIC_VECTOR(11 downto 0) :="111110------";
constant Mascara_6 : STD_LOGIC_VECTOR(11 downto 0) :="1111110-----";
constant Mascara_7 : STD_LOGIC_VECTOR(11 downto 0) :="11111110----";
constant Mascara_8 : STD_LOGIC_VECTOR(11 downto 0) :="111111110---";
constant Mascara_9 : STD_LOGIC_VECTOR(11 downto 0) :="1111111110--";
constant Mascara_10 : STD_LOGIC_VECTOR(11 downto 0) :="11111111110-";
constant Mascara_11 : STD_LOGIC_VECTOR(11 downto 0) :="111111111110";
constant Mascara_12 : STD_LOGIC_VECTOR(11 downto 0) :="111111111111";
--*********************************************************************

--*********************************************************************
--Mascaras para la condicion E3
constant Mascara_15 : STD_LOGIC_VECTOR(11 downto 0) :="-10---------";
constant Mascara_16 : STD_LOGIC_VECTOR(11 downto 0) :="-110--------";
constant Mascara_17 : STD_LOGIC_VECTOR(11 downto 0) :="-1110-------";
constant Mascara_18 : STD_LOGIC_VECTOR(11 downto 0) :="-11110------";
constant Mascara_19 : STD_LOGIC_VECTOR(11 downto 0) :="-111110-----";
constant Mascara_20 : STD_LOGIC_VECTOR(11 downto 0) :="-1111110----";
constant Mascara_21 : STD_LOGIC_VECTOR(11 downto 0) :="-11111110---";
constant Mascara_22 : STD_LOGIC_VECTOR(11 downto 0) :="-111111110--";
constant Mascara_23 : STD_LOGIC_VECTOR(11 downto 0) :="-1111111110-";
constant Mascara_24 : STD_LOGIC_VECTOR(11 downto 0) :="-11111111110";
constant Mascara_25 : STD_LOGIC_VECTOR(11 downto 0) :="-11111111111";

constant Mascara_27 : STD_LOGIC_VECTOR(11 downto 0) :="--10--------";
constant Mascara_28 : STD_LOGIC_VECTOR(11 downto 0) :="--110-------";
constant Mascara_29 : STD_LOGIC_VECTOR(11 downto 0) :="--1110------";
constant Mascara_30 : STD_LOGIC_VECTOR(11 downto 0) :="--11110-----";
```

```
constant Mascara_31 : STD_LOGIC_VECTOR(11 downto 0) :="--111110----";
constant Mascara_32 : STD_LOGIC_VECTOR(11 downto 0) :="--1111110---";
constant Mascara_33 : STD_LOGIC_VECTOR(11 downto 0) :="--11111110--";
constant Mascara_34 : STD_LOGIC_VECTOR(11 downto 0) :="--111111110-";
constant Mascara_35 : STD_LOGIC_VECTOR(11 downto 0) :="--1111111110";
constant Mascara_36 : STD_LOGIC_VECTOR(11 downto 0) :="--1111111111";

constant Mascara_37 : STD_LOGIC_VECTOR(11 downto 0) :="---10-------";
constant Mascara_38 : STD_LOGIC_VECTOR(11 downto 0) :="---110------";
constant Mascara_39 : STD_LOGIC_VECTOR(11 downto 0) :="---1110-----";
constant Mascara_40 : STD_LOGIC_VECTOR(11 downto 0) :="---11110----";
constant Mascara_41 : STD_LOGIC_VECTOR(11 downto 0) :="---111110---";
constant Mascara_42 : STD_LOGIC_VECTOR(11 downto 0) :="---1111110--";
constant Mascara_43 : STD_LOGIC_VECTOR(11 downto 0) :="---11111110-";
constant Mascara_44 : STD_LOGIC_VECTOR(11 downto 0) :="---111111110";
constant Mascara_45 : STD_LOGIC_VECTOR(11 downto 0) :="---111111111";

constant Mascara_46 : STD_LOGIC_VECTOR(11 downto 0) :="----10------";
constant Mascara_47 : STD_LOGIC_VECTOR(11 downto 0) :="----110-----";
constant Mascara_48 : STD_LOGIC_VECTOR(11 downto 0) :="----1110----";
constant Mascara_49 : STD_LOGIC_VECTOR(11 downto 0) :="----11110---";
constant Mascara_50 : STD_LOGIC_VECTOR(11 downto 0) :="----111110--";
constant Mascara_51 : STD_LOGIC_VECTOR(11 downto 0) :="----1111110-";
constant Mascara_52 : STD_LOGIC_VECTOR(11 downto 0) :="----11111110";
constant Mascara_53 : STD_LOGIC_VECTOR(11 downto 0) :="----11111111";

constant Mascara_54 : STD_LOGIC_VECTOR(11 downto 0) :="-----10-----";
constant Mascara_55 : STD_LOGIC_VECTOR(11 downto 0) :="-----110----";
constant Mascara_56 : STD_LOGIC_VECTOR(11 downto 0) :="-----1110---";
constant Mascara_57 : STD_LOGIC_VECTOR(11 downto 0) :="-----11110--";
constant Mascara_58 : STD_LOGIC_VECTOR(11 downto 0) :="-----111110-";
constant Mascara_59 : STD_LOGIC_VECTOR(11 downto 0) :="-----1111110";
constant Mascara_60 : STD_LOGIC_VECTOR(11 downto 0) :="-----1111111";


constant Mascara_61 : STD_LOGIC_VECTOR(11 downto 0) :="------10----";
constant Mascara_62 : STD_LOGIC_VECTOR(11 downto 0) :="------110---";
constant Mascara_63 : STD_LOGIC_VECTOR(11 downto 0) :="------1110--";
constant Mascara_64 : STD_LOGIC_VECTOR(11 downto 0) :="------11110-";
constant Mascara_65 : STD_LOGIC_VECTOR(11 downto 0) :="------111110";
constant Mascara_66 : STD_LOGIC_VECTOR(11 downto 0) :="------111111";

constant Mascara_67 : STD_LOGIC_VECTOR(11 downto 0) :="-------10---";
constant Mascara_68 : STD_LOGIC_VECTOR(11 downto 0) :="-------110--";
constant Mascara_69 : STD_LOGIC_VECTOR(11 downto 0) :="-------1110-";
constant Mascara_70 : STD_LOGIC_VECTOR(11 downto 0) :="-------11110";
constant Mascara_71 : STD_LOGIC_VECTOR(11 downto 0) :="-------11111";

constant Mascara_72 : STD_LOGIC_VECTOR(11 downto 0) :="--------10--";
constant Mascara_73 : STD_LOGIC_VECTOR(11 downto 0) :="--------110-";
constant Mascara_74 : STD_LOGIC_VECTOR(11 downto 0) :="--------1110";
```

```
constant Mascara_75 : STD_LOGIC_VECTOR(11 downto 0) :="--------1111";

constant Mascara_76 : STD_LOGIC_VECTOR(11 downto 0) :="---------10-";
constant Mascara_77 : STD_LOGIC_VECTOR(11 downto 0) :="---------110";
constant Mascara_78 : STD_LOGIC_VECTOR(11 downto 0) :="---------111";

constant Mascara_79 : STD_LOGIC_VECTOR(11 downto 0) :="----------10";
constant Mascara_80 : STD_LOGIC_VECTOR(11 downto 0) :="----------11";

constant Mascara_81 : STD_LOGIC_VECTOR(11 downto 0) :="-----------1";
--*************************************************************************
constant Mascara_82 : STD_LOGIC_VECTOR(11 downto 0) :="-0----------";
constant Mascara_83 : STD_LOGIC_VECTOR(11 downto 0) :="--0---------";
constant Mascara_84 : STD_LOGIC_VECTOR(11 downto 0) :="---0--------";
constant Mascara_85 : STD_LOGIC_VECTOR(11 downto 0) :="----0-------";
constant Mascara_86 : STD_LOGIC_VECTOR(11 downto 0) :="-----0------";
constant Mascara_87 : STD_LOGIC_VECTOR(11 downto 0) :="------0-----";
constant Mascara_88 : STD_LOGIC_VECTOR(11 downto 0) :="-------0----";
constant Mascara_89 : STD_LOGIC_VECTOR(11 downto 0) :="--------0---";
constant Mascara_90 : STD_LOGIC_VECTOR(11 downto 0) :="---------0--";
constant Mascara_91 : STD_LOGIC_VECTOR(11 downto 0) :="----------0-";
constant Mascara_92 : STD_LOGIC_VECTOR(11 downto 0) :="-----------0";

constant Mascara_14 : STD_LOGIC_VECTOR(11 downto 0) :="0-----------";
constant Mascara_26 : STD_LOGIC_VECTOR(11 downto 0) :="1-----------";
--*************************************************************************
begin
-- Esta seccion es para cuando se cumple la condicion de igualdad entre los
--limites, pero ninguna condicion E3
S <= "000000" when( Palabra_Igualdad="000000000000") and
( std_match(Mascara_82,Palabra_E3)) else
"000001" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_83,Palabra_E3)) else
"000010" when( std_match(Mascara_2,Palabra_Igualdad)) and
( std_match(Mascara_84,Palabra_E3)) else
"000011" when( std_match(Mascara_3,Palabra_Igualdad)) and
( std_match(Mascara_85,Palabra_E3)) else
"000100" when( std_match(Mascara_4,Palabra_Igualdad)) and
( std_match(Mascara_86,Palabra_E3)) else
"000101" when( std_match(Mascara_5,Palabra_Igualdad)) and
( std_match(Mascara_87,Palabra_E3)) else
"000110" when( std_match(Mascara_6,Palabra_Igualdad)) and
( std_match(Mascara_88,Palabra_E3)) else
"000111" when( std_match(Mascara_7,Palabra_Igualdad)) and
( std_match(Mascara_89,Palabra_E3)) else
"001000" when( std_match(Mascara_8,Palabra_Igualdad)) and
( std_match(Mascara_90,Palabra_E3)) else
"001001" when( std_match(Mascara_9,Palabra_Igualdad)) and
( std_match(Mascara_91,Palabra_E3)) else
"001010" when( std_match(Mascara_10,Palabra_Igualdad)) and
( std_match(Mascara_92,Palabra_E3)) else
```

```
"001011"when(std_match(Mascara_11,Palabra_Igualdad))else
"001100" when( std_match(Mascara_12,Palabra_Igualdad)) else
--**************************************************************************
-- Esta parte es para cuando se presenta la condicion E3,
--pero ninguna condicion de igualdad entre limites
"010001" when( std_match(Mascara_14,Palabra_Igualdad)) and
( std_match(Mascara_15,Palabra_E3)) else
"010010" when( std_match(Mascara_14,Palabra_Igualdad)) and
( std_match(Mascara_16,Palabra_E3)) else
"010011" when( std_match(Mascara_14,Palabra_Igualdad)) and
( std_match(Mascara_17,Palabra_E3)) else
"010100" when( std_match(Mascara_14,Palabra_Igualdad)) and
( std_match(Mascara_18,Palabra_E3)) else
"010101" when( std_match(Mascara_14,Palabra_Igualdad)) and
( std_match(Mascara_19,Palabra_E3)) else
"010110" when( std_match(Mascara_14,Palabra_Igualdad)) and
( std_match(Mascara_20,Palabra_E3)) else
"010111" when( std_match(Mascara_14,Palabra_Igualdad)) and
( std_match(Mascara_21,Palabra_E3)) else
"011000" when( std_match(Mascara_14,Palabra_Igualdad)) and
( std_match(Mascara_22,Palabra_E3)) else
"011001" when( std_match(Mascara_14,Palabra_Igualdad)) and
( std_match(Mascara_23,Palabra_E3)) else
"011010" when( std_match(Mascara_14,Palabra_Igualdad)) and
( std_match(Mascara_24,Palabra_E3)) else
"011011" when( std_match(Mascara_14,Palabra_Igualdad)) and
( std_match(Mascara_25,Palabra_E3)) else
--**************************************************************************
-- Aqui se declara cuando tenemos una condicion de igualdad y alguna condicion E3
"110001" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_15,Palabra_E3)) else
"110010" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_16,Palabra_E3)) else
"110011" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_17,Palabra_E3)) else
"110100" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_18,Palabra_E3)) else
"110101" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_19,Palabra_E3)) else
"110110" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_20,Palabra_E3)) else
"110111" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_21,Palabra_E3)) else
"111000" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_22,Palabra_E3)) else
"111001" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_23,Palabra_E3)) else
"111010" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_24,Palabra_E3)) else
"111011" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_25,Palabra_E3)) else
```

```
--*****************************************************************************
--*****************************************************************************
--*****************************************************************************
--*****************************************************************************
-- Toda esta seccion es para Cuando se presenta la condicion de igualdad
-- y despues de el desplazamiento
--  se genera la condicion E3.
"110010" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_27,Palabra_E3)) else
"110011" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_28,Palabra_E3)) else
"110100" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_29,Palabra_E3)) else
"110101" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_30,Palabra_E3)) else
"110110" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_31,Palabra_E3)) else
"110111" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_32,Palabra_E3)) else
"111000" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_33,Palabra_E3)) else
"111001" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_34,Palabra_E3)) else
"111010" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_35,Palabra_E3)) else
"111011" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_36,Palabra_E3)) else
--*****************************************************************************
"110011" when( std_match(Mascara_2,Palabra_Igualdad)) and
( std_match(Mascara_37,Palabra_E3)) else
"110100" when( std_match(Mascara_2,Palabra_Igualdad)) and
( std_match(Mascara_38,Palabra_E3)) else
"110101" when( std_match(Mascara_2,Palabra_Igualdad)) and
( std_match(Mascara_39,Palabra_E3)) else
"110110" when( std_match(Mascara_2,Palabra_Igualdad)) and
( std_match(Mascara_40,Palabra_E3)) else
"110111" when( std_match(Mascara_2,Palabra_Igualdad)) and
( std_match(Mascara_41,Palabra_E3)) else
"111000" when( std_match(Mascara_2,Palabra_Igualdad)) and
( std_match(Mascara_42,Palabra_E3)) else
"111001" when( std_match(Mascara_2,Palabra_Igualdad)) and
( std_match(Mascara_43,Palabra_E3)) else
"111010" when( std_match(Mascara_2,Palabra_Igualdad)) and
( std_match(Mascara_44,Palabra_E3)) else
"111011" when( std_match(Mascara_2,Palabra_Igualdad)) and
( std_match(Mascara_45,Palabra_E3)) else
--*****************************************************************************
"110100" when( std_match(Mascara_3,Palabra_Igualdad)) and
( std_match(Mascara_46,Palabra_E3)) else
"110101" when( std_match(Mascara_3,Palabra_Igualdad)) and
( std_match(Mascara_47,Palabra_E3)) else
```

```
"110110" when( std_match(Mascara_3,Palabra_Igualdad)) and
( std_match(Mascara_48,Palabra_E3)) else
"110111" when( std_match(Mascara_3,Palabra_Igualdad)) and
( std_match(Mascara_49,Palabra_E3)) else
"111000" when( std_match(Mascara_3,Palabra_Igualdad)) and
( std_match(Mascara_50,Palabra_E3)) else
"111001" when( std_match(Mascara_3,Palabra_Igualdad)) and
( std_match(Mascara_51,Palabra_E3)) else
"111010" when( std_match(Mascara_3,Palabra_Igualdad)) and
( std_match(Mascara_52,Palabra_E3)) else
"111011" when( std_match(Mascara_3,Palabra_Igualdad)) and
( std_match(Mascara_53,Palabra_E3)) else
--*************************************************************************
"110101" when( std_match(Mascara_4,Palabra_Igualdad)) and
( std_match(Mascara_54,Palabra_E3)) else
"110110" when( std_match(Mascara_4,Palabra_Igualdad)) and
( std_match(Mascara_55,Palabra_E3)) else
"110111" when( std_match(Mascara_4,Palabra_Igualdad)) and
( std_match(Mascara_56,Palabra_E3)) else
"111000" when( std_match(Mascara_4,Palabra_Igualdad)) and
( std_match(Mascara_57,Palabra_E3)) else
"111001" when( std_match(Mascara_4,Palabra_Igualdad)) and
( std_match(Mascara_58,Palabra_E3)) else
"111010" when( std_match(Mascara_4,Palabra_Igualdad)) and
( std_match(Mascara_59,Palabra_E3)) else
"111011" when( std_match(Mascara_4,Palabra_Igualdad)) and
( std_match(Mascara_60,Palabra_E3)) else
--*************************************************************************
"110110" when( std_match(Mascara_5,Palabra_Igualdad)) and
( std_match(Mascara_61,Palabra_E3)) else
"110111" when( std_match(Mascara_5,Palabra_Igualdad)) and
( std_match(Mascara_62,Palabra_E3)) else
"111000" when( std_match(Mascara_5,Palabra_Igualdad)) and
( std_match(Mascara_63,Palabra_E3)) else
"111001" when( std_match(Mascara_5,Palabra_Igualdad)) and
( std_match(Mascara_64,Palabra_E3)) else
"111010" when( std_match(Mascara_5,Palabra_Igualdad)) and
( std_match(Mascara_65,Palabra_E3)) else
"111011" when( std_match(Mascara_5,Palabra_Igualdad)) and
( std_match(Mascara_66,Palabra_E3)) else
--*************************************************************************
"110111" when( std_match(Mascara_6,Palabra_Igualdad)) and
( std_match(Mascara_67,Palabra_E3)) else
"111000" when( std_match(Mascara_6,Palabra_Igualdad)) and
( std_match(Mascara_68,Palabra_E3)) else
"111001" when( std_match(Mascara_6,Palabra_Igualdad)) and
( std_match(Mascara_69,Palabra_E3)) else
"111010" when( std_match(Mascara_6,Palabra_Igualdad)) and
( std_match(Mascara_70,Palabra_E3)) else
"111011" when( std_match(Mascara_6,Palabra_Igualdad)) and
( std_match(Mascara_71,Palabra_E3)) else
```

```
--*************************************************************************
"111000" when( std_match(Mascara_7,Palabra_Igualdad)) and
( std_match(Mascara_72,Palabra_E3)) else
"111001" when( std_match(Mascara_7,Palabra_Igualdad)) and
( std_match(Mascara_73,Palabra_E3)) else
"111010" when( std_match(Mascara_7,Palabra_Igualdad)) and
( std_match(Mascara_74,Palabra_E3)) else
"111011" when( std_match(Mascara_7,Palabra_Igualdad)) and
( std_match(Mascara_75,Palabra_E3)) else
--*************************************************************************
"111001" when( std_match(Mascara_8,Palabra_Igualdad)) and
( std_match(Mascara_76,Palabra_E3)) else
"111010" when( std_match(Mascara_8,Palabra_Igualdad)) and
( std_match(Mascara_77,Palabra_E3)) else
"111011" when( std_match(Mascara_8,Palabra_Igualdad)) and
( std_match(Mascara_78,Palabra_E3)) else
--*************************************************************************
"111010" when( std_match(Mascara_9,Palabra_Igualdad)) and
( std_match(Mascara_79,Palabra_E3)) else
"111011" when( std_match(Mascara_9,Palabra_Igualdad)) and
( std_match(Mascara_80,Palabra_E3)) else
--*************************************************************************
"111011" when( std_match(Mascara_10,Palabra_Igualdad)) and
( std_match(Mascara_81,Palabra_E3)) else
--*************************************************************************
--*************************************************************************
--*************************************************************************
"000000";
--*************************************************************************
--*************************************************************************
--*************************************************************************
--*************************************************************************

E3 <=
--*************************************************************************
-- Esta parte es para cuando se presenta la condicion E3, pero ninguna condicion
--de igualdad entre limites
"0001" when( std_match(Mascara_14,Palabra_Igualdad)) and
( std_match(Mascara_15,Palabra_E3)) else
"0010" when( std_match(Mascara_14,Palabra_Igualdad)) and
( std_match(Mascara_16,Palabra_E3)) else
"0011" when( std_match(Mascara_14,Palabra_Igualdad)) and
( std_match(Mascara_17,Palabra_E3)) else
"0100" when( std_match(Mascara_14,Palabra_Igualdad)) and
( std_match(Mascara_18,Palabra_E3)) else
"0101" when( std_match(Mascara_14,Palabra_Igualdad)) and
( std_match(Mascara_19,Palabra_E3)) else
"0110" when( std_match(Mascara_14,Palabra_Igualdad)) and
( std_match(Mascara_20,Palabra_E3)) else
"0111" when( std_match(Mascara_14,Palabra_Igualdad)) and
( std_match(Mascara_21,Palabra_E3)) else
```

```
"1000" when( std_match(Mascara_14,Palabra_Igualdad)) and
( std_match(Mascara_22,Palabra_E3)) else
"1001" when( std_match(Mascara_14,Palabra_Igualdad)) and
( std_match(Mascara_23,Palabra_E3)) else
"1010" when( std_match(Mascara_14,Palabra_Igualdad)) and
( std_match(Mascara_24,Palabra_E3)) else
"1011" when( std_match(Mascara_14,Palabra_Igualdad)) and
( std_match(Mascara_25,Palabra_E3)) else
--*************************************************************************
-- Aqui se declara cuando tenemos una condicion de igualdad y alguna condicion E3
"0001" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_15,Palabra_E3)) else
"0010" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_16,Palabra_E3)) else
"0011" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_17,Palabra_E3)) else
"0100" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_18,Palabra_E3)) else
"0101" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_19,Palabra_E3)) else
"0110" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_20,Palabra_E3)) else
"0111" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_21,Palabra_E3)) else
"1000" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_22,Palabra_E3)) else
"1001" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_23,Palabra_E3)) else
"1010" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_24,Palabra_E3)) else
"1011" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_25,Palabra_E3)) else
--*************************************************************************
--*************************************************************************
--*************************************************************************
--*************************************************************************
-- Toda esta seccion es para Cuando se presenta la condicion de igualdad y
-- despues de el desplazamiento
--  se genera la condicion E3.
"0001" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_27,Palabra_E3)) else
"0010" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_28,Palabra_E3)) else
"0011" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_29,Palabra_E3)) else
"0100" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_30,Palabra_E3)) else
"0101" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_31,Palabra_E3)) else
"0110" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_32,Palabra_E3)) else
```

```
"0111" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_33,Palabra_E3)) else
"1000" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_34,Palabra_E3)) else
"1001" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_35,Palabra_E3)) else
"1010" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_36,Palabra_E3)) else
--**************************************************************************
"0001" when( std_match(Mascara_2,Palabra_Igualdad)) and
( std_match(Mascara_37,Palabra_E3)) else
"0010" when( std_match(Mascara_2,Palabra_Igualdad)) and
( std_match(Mascara_38,Palabra_E3)) else
"0011" when( std_match(Mascara_2,Palabra_Igualdad)) and
( std_match(Mascara_39,Palabra_E3)) else
"0100" when( std_match(Mascara_2,Palabra_Igualdad)) and
( std_match(Mascara_40,Palabra_E3)) else
"0101" when( std_match(Mascara_2,Palabra_Igualdad)) and
( std_match(Mascara_41,Palabra_E3)) else
"0110" when( std_match(Mascara_2,Palabra_Igualdad)) and
( std_match(Mascara_42,Palabra_E3)) else
"0111" when( std_match(Mascara_2,Palabra_Igualdad)) and
( std_match(Mascara_43,Palabra_E3)) else
"1000" when( std_match(Mascara_2,Palabra_Igualdad)) and
( std_match(Mascara_44,Palabra_E3)) else
"1001" when( std_match(Mascara_2,Palabra_Igualdad)) and
( std_match(Mascara_45,Palabra_E3)) else
--**************************************************************************
"0001" when( std_match(Mascara_3,Palabra_Igualdad)) and
( std_match(Mascara_46,Palabra_E3)) else
"0010" when( std_match(Mascara_3,Palabra_Igualdad)) and
( std_match(Mascara_47,Palabra_E3)) else
"0011" when( std_match(Mascara_3,Palabra_Igualdad)) and
( std_match(Mascara_48,Palabra_E3)) else
"0100" when( std_match(Mascara_3,Palabra_Igualdad)) and
( std_match(Mascara_49,Palabra_E3)) else
"0101" when( std_match(Mascara_3,Palabra_Igualdad)) and
( std_match(Mascara_50,Palabra_E3)) else
"0110" when( std_match(Mascara_3,Palabra_Igualdad)) and
( std_match(Mascara_51,Palabra_E3)) else
"0111" when( std_match(Mascara_3,Palabra_Igualdad)) and
( std_match(Mascara_52,Palabra_E3)) else
"1000" when( std_match(Mascara_3,Palabra_Igualdad)) and
( std_match(Mascara_53,Palabra_E3)) else
--**************************************************************************
"0001" when( std_match(Mascara_4,Palabra_Igualdad)) and
( std_match(Mascara_54,Palabra_E3)) else
"0010" when( std_match(Mascara_4,Palabra_Igualdad)) and
( std_match(Mascara_55,Palabra_E3)) else
"0011" when( std_match(Mascara_4,Palabra_Igualdad)) and
( std_match(Mascara_56,Palabra_E3)) else
```

```
"0100" when( std_match(Mascara_4,Palabra_Igualdad)) and
( std_match(Mascara_57,Palabra_E3)) else
"0101" when( std_match(Mascara_4,Palabra_Igualdad)) and
( std_match(Mascara_58,Palabra_E3)) else
"0110" when( std_match(Mascara_4,Palabra_Igualdad)) and
( std_match(Mascara_59,Palabra_E3)) else
"0111" when( std_match(Mascara_4,Palabra_Igualdad)) and
( std_match(Mascara_60,Palabra_E3)) else
--*************************************************************************
"0001" when( std_match(Mascara_5,Palabra_Igualdad)) and
( std_match(Mascara_61,Palabra_E3)) else
"0010" when( std_match(Mascara_5,Palabra_Igualdad)) and
( std_match(Mascara_62,Palabra_E3)) else
"0011" when( std_match(Mascara_5,Palabra_Igualdad)) and
( std_match(Mascara_63,Palabra_E3)) else
"0100" when( std_match(Mascara_5,Palabra_Igualdad)) and
( std_match(Mascara_64,Palabra_E3)) else
"0101" when( std_match(Mascara_5,Palabra_Igualdad)) and
( std_match(Mascara_65,Palabra_E3)) else
"0110" when( std_match(Mascara_5,Palabra_Igualdad)) and
( std_match(Mascara_66,Palabra_E3)) else
--*************************************************************************
"0001" when( std_match(Mascara_6,Palabra_Igualdad)) and
( std_match(Mascara_67,Palabra_E3)) else
"0010" when( std_match(Mascara_6,Palabra_Igualdad)) and
( std_match(Mascara_68,Palabra_E3)) else
"0011" when( std_match(Mascara_6,Palabra_Igualdad)) and
( std_match(Mascara_69,Palabra_E3)) else
"0100" when( std_match(Mascara_6,Palabra_Igualdad)) and
( std_match(Mascara_70,Palabra_E3)) else
"0101" when( std_match(Mascara_6,Palabra_Igualdad)) and
( std_match(Mascara_71,Palabra_E3)) else
--*************************************************************************
"0001" when( std_match(Mascara_7,Palabra_Igualdad)) and
( std_match(Mascara_72,Palabra_E3)) else
"0010" when( std_match(Mascara_7,Palabra_Igualdad)) and
( std_match(Mascara_73,Palabra_E3)) else
"0011" when( std_match(Mascara_7,Palabra_Igualdad)) and
( std_match(Mascara_74,Palabra_E3)) else
"0100" when( std_match(Mascara_7,Palabra_Igualdad)) and
( std_match(Mascara_75,Palabra_E3)) else
--*************************************************************************
"0001" when( std_match(Mascara_8,Palabra_Igualdad)) and
( std_match(Mascara_76,Palabra_E3)) else
"0010" when( std_match(Mascara_8,Palabra_Igualdad)) and
( std_match(Mascara_77,Palabra_E3)) else
"0011" when( std_match(Mascara_8,Palabra_Igualdad)) and
( std_match(Mascara_78,Palabra_E3)) else
--*************************************************************************
"0001" when( std_match(Mascara_9,Palabra_Igualdad)) and
( std_match(Mascara_79,Palabra_E3)) else
```

```
"0010" when( std_match(Mascara_9,Palabra_Igualdad)) and
( std_match(Mascara_80,Palabra_E3)) else
--*****************************************************************************
"0001" when ( std_match(Mascara_10,Palabra_Igualdad)) and
( std_match(Mascara_81,Palabra_E3)) else
--*****************************************************************************
--*****************************************************************************
--*****************************************************************************
"0000";
--*****************************************************************************
--*****************************************************************************
--*****************************************************************************
--*****************************************************************************

-- Esta seccion es para cuando se cumple la condicion de igualdad entre los
-- limites, pero ninguna condicion E3
Igualdad <= "0000" when( Palabra_Igualdad="000000000000") and
( std_match(Mascara_82,Palabra_E3)) else
"0001" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_83,Palabra_E3)) else
"0010" when( std_match(Mascara_2,Palabra_Igualdad)) and
( std_match(Mascara_84,Palabra_E3)) else
"0011" when( std_match(Mascara_3,Palabra_Igualdad)) and
( std_match(Mascara_85,Palabra_E3)) else
"0100" when( std_match(Mascara_4,Palabra_Igualdad)) and
( std_match(Mascara_86,Palabra_E3)) else
"0101" when( std_match(Mascara_5,Palabra_Igualdad)) and
( std_match(Mascara_87,Palabra_E3)) else
"0110" when( std_match(Mascara_6,Palabra_Igualdad)) and
( std_match(Mascara_88,Palabra_E3)) else
"0111" when( std_match(Mascara_7,Palabra_Igualdad)) and
( std_match(Mascara_89,Palabra_E3)) else
"1000" when( std_match(Mascara_8,Palabra_Igualdad)) and
( std_match(Mascara_90,Palabra_E3)) else
"1001" when( std_match(Mascara_9,Palabra_Igualdad)) and
( std_match(Mascara_91,Palabra_E3)) else
"1010" when( std_match(Mascara_10,Palabra_Igualdad)) and
( std_match(Mascara_92,Palabra_E3))else
"1011" when(std_match(Mascara_11,Palabra_Igualdad)) else
"1100" when(std_match(Mascara_12,Palabra_Igualdad)) else
--*****************************************************************************
--*****************************************************************************
-- Aqui se declara cuando tenemos una condicion de igualdad y alguna condicion E3
"0001" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_15,Palabra_E3)) else
"0001" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_16,Palabra_E3)) else
"0001" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_17,Palabra_E3)) else
"0001" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_18,Palabra_E3)) else
```

```
"0001" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_19,Palabra_E3)) else
"0001" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_20,Palabra_E3)) else
"0001" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_21,Palabra_E3)) else
"0001" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_22,Palabra_E3)) else
"0001" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_23,Palabra_E3)) else
"0001" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_24,Palabra_E3)) else
"0001" when( std_match(Mascara_26,Palabra_Igualdad)) and
( std_match(Mascara_25,Palabra_E3)) else
--*************************************************************************
--*************************************************************************
--*************************************************************************
--*************************************************************************
-- Toda esta seccion es para Cuando se presenta la condicion de igualdad y
-- despues de el desplazamiento
--  se genera la condicion E3.
"0001" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_27,Palabra_E3)) else
"0001" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_28,Palabra_E3)) else
"0001" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_29,Palabra_E3)) else
"0001" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_30,Palabra_E3)) else
"0001" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_31,Palabra_E3)) else
"0001" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_32,Palabra_E3)) else
"0001" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_33,Palabra_E3)) else
"0001" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_34,Palabra_E3)) else
"0001" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_35,Palabra_E3)) else
"0001" when( std_match(Mascara_1,Palabra_Igualdad)) and
( std_match(Mascara_36,Palabra_E3)) else
--*************************************************************************
"0010" when( std_match(Mascara_2,Palabra_Igualdad)) and
( std_match(Mascara_37,Palabra_E3)) else
"0010" when( std_match(Mascara_2,Palabra_Igualdad)) and
( std_match(Mascara_38,Palabra_E3)) else
"0010" when( std_match(Mascara_2,Palabra_Igualdad)) and
( std_match(Mascara_39,Palabra_E3)) else
"0010" when( std_match(Mascara_2,Palabra_Igualdad)) and
( std_match(Mascara_40,Palabra_E3)) else
"0010" when( std_match(Mascara_2,Palabra_Igualdad)) and
```

```
( std_match(Mascara_41,Palabra_E3)) else
"0010" when( std_match(Mascara_2,Palabra_Igualdad)) and
( std_match(Mascara_42,Palabra_E3)) else
"0010" when( std_match(Mascara_2,Palabra_Igualdad)) and
( std_match(Mascara_43,Palabra_E3)) else
"0010" when( std_match(Mascara_2,Palabra_Igualdad)) and
( std_match(Mascara_44,Palabra_E3)) else
"0010" when( std_match(Mascara_2,Palabra_Igualdad)) and
( std_match(Mascara_45,Palabra_E3)) else
--*************************************************************************
"0011" when( std_match(Mascara_3,Palabra_Igualdad)) and
( std_match(Mascara_46,Palabra_E3)) else
"0011" when( std_match(Mascara_3,Palabra_Igualdad)) and
( std_match(Mascara_47,Palabra_E3)) else
"0011" when( std_match(Mascara_3,Palabra_Igualdad)) and
( std_match(Mascara_48,Palabra_E3)) else
"0011" when( std_match(Mascara_3,Palabra_Igualdad)) and
( std_match(Mascara_49,Palabra_E3)) else
"0011" when( std_match(Mascara_3,Palabra_Igualdad)) and
( std_match(Mascara_50,Palabra_E3)) else
"0011" when( std_match(Mascara_3,Palabra_Igualdad)) and
( std_match(Mascara_51,Palabra_E3)) else
"0011" when( std_match(Mascara_3,Palabra_Igualdad)) and
( std_match(Mascara_52,Palabra_E3)) else
"0011" when( std_match(Mascara_3,Palabra_Igualdad)) and
( std_match(Mascara_53,Palabra_E3)) else
--*************************************************************************
"0100" when( std_match(Mascara_4,Palabra_Igualdad)) and
( std_match(Mascara_54,Palabra_E3)) else
"0100" when( std_match(Mascara_4,Palabra_Igualdad)) and
( std_match(Mascara_55,Palabra_E3)) else
"0100" when( std_match(Mascara_4,Palabra_Igualdad)) and
( std_match(Mascara_56,Palabra_E3)) else
"0100" when( std_match(Mascara_4,Palabra_Igualdad)) and
( std_match(Mascara_57,Palabra_E3)) else
"0100" when( std_match(Mascara_4,Palabra_Igualdad)) and
( std_match(Mascara_58,Palabra_E3)) else
"0100" when( std_match(Mascara_4,Palabra_Igualdad)) and
( std_match(Mascara_59,Palabra_E3)) else
"0100" when( std_match(Mascara_4,Palabra_Igualdad)) and
( std_match(Mascara_60,Palabra_E3)) else
--*************************************************************************
"0101" when( std_match(Mascara_5,Palabra_Igualdad)) and
( std_match(Mascara_61,Palabra_E3)) else
"0101" when( std_match(Mascara_5,Palabra_Igualdad)) and
( std_match(Mascara_62,Palabra_E3)) else
"0101" when( std_match(Mascara_5,Palabra_Igualdad)) and
( std_match(Mascara_63,Palabra_E3)) else
"0101" when( std_match(Mascara_5,Palabra_Igualdad)) and
( std_match(Mascara_64,Palabra_E3)) else
"0101" when( std_match(Mascara_5,Palabra_Igualdad)) and
```

```
( std_match(Mascara_65,Palabra_E3)) else
"0101" when( std_match(Mascara_5,Palabra_Igualdad)) and
( std_match(Mascara_66,Palabra_E3)) else
--*****************************************************************************
"0110" when( std_match(Mascara_6,Palabra_Igualdad)) and
( std_match(Mascara_67,Palabra_E3)) else
"0110" when( std_match(Mascara_6,Palabra_Igualdad)) and
( std_match(Mascara_68,Palabra_E3)) else
"0110" when( std_match(Mascara_6,Palabra_Igualdad)) and
( std_match(Mascara_69,Palabra_E3)) else
"0110" when( std_match(Mascara_6,Palabra_Igualdad)) and
( std_match(Mascara_70,Palabra_E3)) else
"0110" when( std_match(Mascara_6,Palabra_Igualdad)) and
( std_match(Mascara_71,Palabra_E3)) else
--*****************************************************************************
"0111" when( std_match(Mascara_7,Palabra_Igualdad)) and
( std_match(Mascara_72,Palabra_E3)) else
"0111" when( std_match(Mascara_7,Palabra_Igualdad)) and
( std_match(Mascara_73,Palabra_E3)) else
"0111" when( std_match(Mascara_7,Palabra_Igualdad)) and
( std_match(Mascara_74,Palabra_E3)) else
"0111" when( std_match(Mascara_7,Palabra_Igualdad)) and
( std_match(Mascara_75,Palabra_E3)) else
--*****************************************************************************
"1000" when( std_match(Mascara_8,Palabra_Igualdad)) and
( std_match(Mascara_76,Palabra_E3)) else
"1000" when( std_match(Mascara_8,Palabra_Igualdad)) and
( std_match(Mascara_77,Palabra_E3)) else
"1000" when( std_match(Mascara_8,Palabra_Igualdad)) and
( std_match(Mascara_78,Palabra_E3)) else
--*****************************************************************************
"1001" when( std_match(Mascara_9,Palabra_Igualdad)) and
( std_match(Mascara_79,Palabra_E3)) else
"1001" when( std_match(Mascara_9,Palabra_Igualdad)) and
( std_match(Mascara_80,Palabra_E3)) else
--*****************************************************************************
"1010" when( std_match(Mascara_10,Palabra_Igualdad)) and
( std_match(Mascara_81,Palabra_E3)) else
--*****************************************************************************
--*****************************************************************************
--*****************************************************************************
"0000";
end Behavioral;
```

desplazamiento L

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity DesplazamientoL is
Port (  I  :  in   STD_LOGIC_VECTOR(11 downto 0);
S  :  in   STD_LOGIC_VECTOR(5  downto 0);
O : out STD_LOGIC_VECTOR(11 downto 0));
end DesplazamientoL;

architecture Behavioral of DesplazamientoL is
--*************************************************************************
signal L0 : STD_LOGIC_VECTOR(11 downto 0);
signal L1 : STD_LOGIC_VECTOR(11 downto 0);
signal L2 : STD_LOGIC_VECTOR(11 downto 0);
signal L3 : STD_LOGIC_VECTOR(11 downto 0);
signal L4 : STD_LOGIC_VECTOR(11 downto 0);
signal L5 : STD_LOGIC_VECTOR(11 downto 0);
signal L6 : STD_LOGIC_VECTOR(11 downto 0);
signal L7 : STD_LOGIC_VECTOR(11 downto 0);
signal L8 : STD_LOGIC_VECTOR(11 downto 0);
signal L9 : STD_LOGIC_VECTOR(11 downto 0);
signal L10 : STD_LOGIC_VECTOR(11 downto 0);
signal L11 : STD_LOGIC_VECTOR(11 downto 0);
signal L12 : STD_LOGIC_VECTOR(11 downto 0);


signal IL0 : STD_LOGIC_VECTOR(11 downto 0);
signal IL1 : STD_LOGIC_VECTOR(11 downto 0);
signal IL2 : STD_LOGIC_VECTOR(11 downto 0);
signal IL3 : STD_LOGIC_VECTOR(11 downto 0);
signal IL4 : STD_LOGIC_VECTOR(11 downto 0);
signal IL5 : STD_LOGIC_VECTOR(11 downto 0);
signal IL6 : STD_LOGIC_VECTOR(11 downto 0);
signal IL7 : STD_LOGIC_VECTOR(11 downto 0);
signal IL8 : STD_LOGIC_VECTOR(11 downto 0);
signal IL9 : STD_LOGIC_VECTOR(11 downto 0);
signal IL10 : STD_LOGIC_VECTOR(11 downto 0);
signal IL11 : STD_LOGIC_VECTOR(11 downto 0);
--*************************************************************************
begin
--*************************************************************************
---------------------------- Desplaza I :1 -------------------------------------
L11(11)<=I(10);
L11(10)<=I(9);
L11(9)<=I(8);
L11(8)<=I(7);
L11(7)<=I(6);
L11(6)<=I(5);
```

```
L11(5)<=I(4);
L11(4)<=I(3);
L11(3)<=I(2);
L11(2)<=I(1);
L11(1)<=I(0);
L11(0)<='0';
---------------------------- Desplaza I : 2 -------------------------------------
L10(11)<=I(9);
L10(10)<=I(8);
L10(9)<=I(7);
L10(8)<=I(6);
L10(7)<=I(5);
L10(6)<=I(4);
L10(5)<=I(3);
L10(4)<=I(2);
L10(3)<=I(1);
L10(2)<=I(0);
L10(1)<='0';
L10(0)<='0';
--------------------------------------------------------------------------------

---------------------------- Desplaza I :3 -------------------------------------
L9(11)<=I(8);
L9(10)<=I(7);
L9(9)<=I(6);
L9(8)<=I(5);
L9(7)<=I(4);
L9(6)<=I(3);
L9(5)<=I(2);
L9(4)<=I(1);
L9(3)<=I(0);
L9(2)<='0';
L9(1)<='0';
L9(0)<='0';
--------------------------------------------------------------------------------

---------------------------- Desplaza I :4 -------------------------------------
L8(11)<=I(7);
L8(10)<=I(6);
L8(9)<=I(5);
L8(8)<=I(4);
L8(7)<=I(3);
L8(6)<=I(2);
L8(5)<=I(1);
L8(4)<=I(0);
L8(3)<='0';
L8(2)<='0';
L8(1)<='0';
L8(0)<='0';
--------------------------------------------------------------------------------
```

```
---------------------------- Desplaza I :5 -------------------------------------
L7(11)<=I(6);
L7(10)<=I(5);
L7(9)<=I(4);
L7(8)<=I(3);
L7(7)<=I(2);
L7(6)<=I(1);
L7(5)<=I(0);
L7(4)<='0';
L7(3)<='0';
L7(2)<='0';
L7(1)<='0';
L7(0)<='0';
--------------------------------------------------------------------------------

---------------------------- Desplaza I :6 -------------------------------------
L6(11)<=I(5);
L6(10)<=I(4);
L6(9)<=I(3);
L6(8)<=I(2);
L6(7)<=I(1);
L6(6)<=I(0);
L6(5)<='0';
L6(4)<='0';
L6(3)<='0';
L6(2)<='0';
L6(1)<='0';
L6(0)<='0';
--------------------------------------------------------------------------------

---------------------------- Desplaza I :7 -------------------------------------
L5(11)<=I(4);
L5(10)<=I(3);
L5(9)<=I(2);
L5(8)<=I(1);
L5(7)<=I(0);
L5(6)<='0';
L5(5)<='0';
L5(4)<='0';
L5(3)<='0';
L5(2)<='0';
L5(1)<='0';
L5(0)<='0';
--------------------------------------------------------------------------------

---------------------------- Desplaza I :8 -------------------------------------
L4(11)<=I(3);
L4(10)<=I(2);
L4(9)<=I(1);
L4(8)<=I(0);
L4(7)<='0';
```

```
L4(6)<='0';
L4(5)<='0';
L4(4)<='0';
L4(3)<='0';
L4(2)<='0';
L4(1)<='0';
L4(0)<='0';
-------------------------------------------------------------------------------

---------------------------- Desplaza I :9 ------------------------------------
L3(11)<=I(2);
L3(10)<=I(1);
L3(9)<=I(0);
L3(8)<='0';
L3(7)<='0';
L3(6)<='0';
L3(5)<='0';
L3(4)<='0';
L3(3)<='0';
L3(2)<='0';
L3(1)<='0';
L3(0)<='0';
-------------------------------------------------------------------------------

---------------------------- Desplaza I :10 -----------------------------------
L2(11)<=I(1);
L2(10)<=I(0);
L2(9)<='0';
L2(8)<='0';
L2(7)<='0';
L2(6)<='0';
L2(5)<='0';
L2(4)<='0';
L2(3)<='0';
L2(2)<='0';
L2(1)<='0';
L2(0)<='0';
-------------------------------------------------------------------------------

---------------------------- Desplaza I :11 -----------------------------------
L1(11)<=I(0);
L1(10)<='0';
L1(9)<='0';
L1(8)<='0';
L1(7)<='0';
L1(6)<='0';
L1(5)<='0';
L1(4)<='0';
L1(3)<='0';
L1(2)<='0';
L1(1)<='0';
```

```
L1(0)<='0';
-------------------------------------------------------------------------------
---------------------------- Sin desplazamiento ---------------------------------
L0 <= I;
---------------------------- Desplazamiento Total -------------------------------
L12 <= "000000000000";
-------------------------------------------------------------------------------
--*****************************************************************************



--*****************************************************************************
IL11(11) <= not  I(10);
IL11(10) <= I(9);
IL11(9) <= I(8);
IL11(8) <= I(7);
IL11(7) <= I(6);
IL11(6) <= I(5);
IL11(5) <= I(4);
IL11(4) <= I(3);
IL11(3) <= I(2);
IL11(2) <= I(1);
IL11(1) <= I(0);
IL11(0) <= '0';
---------------------------- DespILaza II :1 ------------------------------------
IL10(11) <= not  I(9);
IL10(10) <=  I(8);
IL10(9) <= I(7);
IL10(8) <= I(6);
IL10(7) <= I(5);
IL10(6) <= I(4);
IL10(5) <= I(3);
IL10(4) <= I(2);
IL10(3) <= I(1);
IL10(2) <= I(0);
IL10(1) <= '0';
IL10(0) <= '0';
-------------------------------------------------------------------------------

---------------------------- DespILaza II :2 ------------------------------------
IL9(11) <= not  I(8);
IL9(10) <=  I(7);
IL9(9) <= I(6);
IL9(8) <= I(5);
IL9(7) <= I(4);
IL9(6) <= I(3);
IL9(5) <= I(2);
IL9(4) <= I(1);
IL9(3) <= I(0);
IL9(2) <= '0';
IL9(1) <= '0';
```

```
IL9(0) <= '0';
-------------------------------------------------------------------------------

---------------------------- DespILaza II :3 -----------------------------------
IL8(11) <= not  I(7);
IL8(10) <=   I(6);
IL8(9) <= I(5);
IL8(8) <= I(4);
IL8(7) <= I(3);
IL8(6) <= I(2);
IL8(5) <= I(1);
IL8(4) <= I(0);
IL8(3) <= '0';
IL8(2) <= '0';
IL8(1) <= '0';
IL8(0) <= '0';
-------------------------------------------------------------------------------

---------------------------- DespILaza II :4 -----------------------------------
IL7(11) <= not  I(6);
IL7(10) <=  I(5);
IL7(9) <= I(4);
IL7(8) <= I(3);
IL7(7) <= I(2);
IL7(6) <= I(1);
IL7(5) <= I(0);
IL7(4) <= '0';
IL7(3) <= '0';
IL7(2) <= '0';
IL7(1) <= '0';
IL7(0) <= '0';
-------------------------------------------------------------------------------

---------------------------- DespILaza II :5 -----------------------------------
IL6(11) <= not  I(5);
IL6(10) <=  I(4);
IL6(9) <= I(3);
IL6(8) <= I(2);
IL6(7) <= I(1);
IL6(6) <= I(0);
IL6(5) <= '0';
IL6(4) <= '0';
IL6(3) <= '0';
IL6(2) <= '0';
IL6(1) <= '0';
IL6(0) <= '0';
-------------------------------------------------------------------------------

---------------------------- DespILaza II :6 -----------------------------------
IL5(11) <= not I(4);
IL5(10) <=  I(3);
```

```
IL5(9) <= I(2);
IL5(8) <= I(1);
IL5(7) <= I(0);
IL5(6) <= '0';
IL5(5) <= '0';
IL5(4) <= '0';
IL5(3) <= '0';
IL5(2) <= '0';
IL5(1) <= '0';
IL5(0) <= '0';
-------------------------------------------------------------------------------

---------------------------- DespILaza II :7 -----------------------------------
IL4(11) <= not  I(3);
IL4(10) <=  I(2);
IL4(9) <= I(1);
IL4(8) <= I(0);
IL4(7) <= '0';
IL4(6) <= '0';
IL4(5) <= '0';
IL4(4) <= '0';
IL4(3) <= '0';
IL4(2) <= '0';
IL4(1) <= '0';
IL4(0) <= '0';
-------------------------------------------------------------------------------

---------------------------- DespILaza II :8 -----------------------------------
IL3(11) <= not  I(2);
IL3(10) <=  I(1);
IL3(9) <= I(0);
IL3(8) <= '0';
IL3(7) <= '0';
IL3(6) <= '0';
IL3(5) <= '0';
IL3(4) <= '0';
IL3(3) <= '0';
IL3(2) <= '0';
IL3(1) <= '0';
IL3(0) <= '0';
-------------------------------------------------------------------------------

---------------------------- DespILaza II :9 -----------------------------------
IL2(11) <= not  I(1);
IL2(10) <=  I(0);
IL2(9) <= '0';
IL2(8) <= '0';
IL2(7) <= '0';
IL2(6) <= '0';
IL2(5) <= '0';
IL2(4) <= '0';
```

```
IL2(3) <= '0';
IL2(2) <= '0';
IL2(1) <= '0';
IL2(0) <= '0';
-------------------------------------------------------------------------------

---------------------------- DespILaza II :10 -------------------------------------
IL1(11) <= not  I(0);
IL1(10) <=  '0';
IL1(9) <= '0';
IL1(8) <= '0';
IL1(7) <= '0';
IL1(6) <= '0';
IL1(5) <= '0';
IL1(4) <= '0';
IL1(3) <= '0';
IL1(2) <= '0';
IL1(1) <= '0';
IL1(0) <= '0';
-------------------------------------------------------------------------------
---------------------------- Sin despILazamiento ----------------------------------
IL0 <= I;
-------------------------------------------------------------------------------

--*******************************************************************************



O <=  L0 when S="000000" else
L11 when S="000001" else
L10 when S="000010" else
L9 when S="000011" else
L8 when S="000100" else
L7 when S="000101" else
L6 when S="000110" else
L5 when S="000111" else
L4 when S="001000" else
L3 when S="001001" else
L2 when S="001010" else
L1 when S="001011" else
L12 when S="001100" else
--**********************************
IL0 when S="010000" else
IL11 when S="010001" else
IL10 when S="010010" else
IL9 when S="010011" else
IL8 when S="010100" else
IL7 when S="010101" else
IL6 when S="010110" else
IL5 when S="010111" else
```

```
IL4 when S="011000" else
IL3 when S="011001" else
IL2 when S="011010" else
IL1 when S="011011" else
--************************************
IL0 when S="110000" else
IL11 when S="110001" else
IL10 when S="110010" else
IL9 when S="110011" else
IL8 when S="110100" else
IL7 when S="110101" else
IL6 when S="110110" else
IL5 when S="110111" else
IL4 when S="111000" else
IL3 when S="111001" else
IL2 when S="111010" else
IL1 when S="111011" else
--************************************
"000000000001";

end Behavioral;
```

Desplazamiento U

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity DesplazamientoU is
Port (  I  :  in   STD_LOGIC_VECTOR(11 downto 0);
S  :  in   STD_LOGIC_VECTOR(5  downto 0);
O : out STD_LOGIC_VECTOR(11 downto 0));
end DesplazamientoU;

architecture Behavioral of DesplazamientoU is
--*********************************************************************************
signal U0 : STD_LOGIC_VECTOR(11 downto 0);
signal U1 : STD_LOGIC_VECTOR(11 downto 0);
signal U2 : STD_LOGIC_VECTOR(11 downto 0);
signal U3 : STD_LOGIC_VECTOR(11 downto 0);
signal U4 : STD_LOGIC_VECTOR(11 downto 0);
signal U5 : STD_LOGIC_VECTOR(11 downto 0);
signal U6 : STD_LOGIC_VECTOR(11 downto 0);
signal U7 : STD_LOGIC_VECTOR(11 downto 0);
signal U8 : STD_LOGIC_VECTOR(11 downto 0);
signal U9 : STD_LOGIC_VECTOR(11 downto 0);
signal U10 : STD_LOGIC_VECTOR(11 downto 0);
signal U11 : STD_LOGIC_VECTOR(11 downto 0);
signal U12 : STD_LOGIC_VECTOR(11 downto 0);


signal IU0 : STD_LOGIC_VECTOR(11 downto 0);
signal IU1 : STD_LOGIC_VECTOR(11 downto 0);
signal IU2 : STD_LOGIC_VECTOR(11 downto 0);
signal IU3 : STD_LOGIC_VECTOR(11 downto 0);
signal IU4 : STD_LOGIC_VECTOR(11 downto 0);
signal IU5 : STD_LOGIC_VECTOR(11 downto 0);
signal IU6 : STD_LOGIC_VECTOR(11 downto 0);
signal IU7 : STD_LOGIC_VECTOR(11 downto 0);
signal IU8 : STD_LOGIC_VECTOR(11 downto 0);
signal IU9 : STD_LOGIC_VECTOR(11 downto 0);
signal IU10 : STD_LOGIC_VECTOR(11 downto 0);
signal IU11 : STD_LOGIC_VECTOR(11 downto 0);
--*********************************************************************************
begin
--*********************************************************************************
---------------------------- Desplaza I :1 ------------------------------------
U11(11) <= I(10);
U11(10) <= I(9);
U11(9) <= I(8);
U11(8) <= I(7);
U11(7) <= I(6);
```

```
U11(6) <= I(5);
U11(5) <= I(4);
U11(4) <= I(3);
U11(3) <= I(2);
U11(2) <= I(1);
U11(1) <= I(0);
U11(0) <= '1';
---------------------------- Desplaza I : 2 -------------------------------------
U10(11) <= I(9);
U10(10) <= I(8);
U10(9) <= I(7);
U10(8) <= I(6);
U10(7) <= I(5);
U10(6) <= I(4);
U10(5) <= I(3);
U10(4) <= I(2);
U10(3) <= I(1);
U10(2) <= I(0);
U10(1) <= '1';
U10(0) <= '1';
--------------------------------------------------------------------------------

---------------------------- Desplaza I :3 -------------------------------------
U9(11) <= I(8);
U9(10) <= I(7);
U9(9) <= I(6);
U9(8) <= I(5);
U9(7) <= I(4);
U9(6) <= I(3);
U9(5) <= I(2);
U9(4) <= I(1);
U9(3) <= I(0);
U9(2) <= '1';
U9(1) <= '1';
U9(0) <= '1';
--------------------------------------------------------------------------------

---------------------------- Desplaza I :4 -------------------------------------
U8(11) <= I(7);
U8(10) <= I(6);
U8(9) <= I(5);
U8(8) <= I(4);
U8(7) <= I(3);
U8(6) <= I(2);
U8(5) <= I(1);
U8(4) <= I(0);
U8(3) <= '1';
U8(2) <= '1';
U8(1) <= '1';
U8(0) <= '1';
--------------------------------------------------------------------------------
```

```
---------------------------- Desplaza I :5 ---------------------------------------
U7(11) <= I(6);
U7(10) <= I(5);
U7(9) <= I(4);
U7(8) <= I(3);
U7(7) <= I(2);
U7(6) <= I(1);
U7(5) <= I(0);
U7(4) <= '1';
U7(3) <= '1';
U7(2) <= '1';
U7(1) <= '1';
U7(0) <= '1';
----------------------------------------------------------------------------------

---------------------------- Desplaza I :6 ---------------------------------------
U6(11) <= I(5);
U6(10) <= I(4);
U6(9) <= I(3);
U6(8) <= I(2);
U6(7) <= I(1);
U6(6) <= I(0);
U6(5) <= '1';
U6(4) <= '1';
U6(3) <= '1';
U6(2) <= '1';
U6(1) <= '1';
U6(0) <= '1';
----------------------------------------------------------------------------------

---------------------------- Desplaza I :7 ---------------------------------------
U5(11) <= I(4);
U5(10) <= I(3);
U5(9) <= I(2);
U5(8) <= I(1);
U5(7) <= I(0);
U5(6) <= '1';
U5(5) <= '1';
U5(4) <= '1';
U5(3) <= '1';
U5(2) <= '1';
U5(1) <= '1';
U5(0) <= '1';
----------------------------------------------------------------------------------

---------------------------- Desplaza I :8 ---------------------------------------
U4(11) <= I(3);
U4(10) <= I(2);
U4(9) <= I(1);
U4(8) <= I(0);
```

```
U4(7) <= '1';
U4(6) <= '1';
U4(5) <= '1';
U4(4) <= '1';
U4(3) <= '1';
U4(2) <= '1';
U4(1) <= '1';
U4(0) <= '1';
-------------------------------------------------------------------------------

---------------------------- Desplaza I :9 ------------------------------------
U3(11) <= I(2);
U3(10) <= I(1);
U3(9) <= I(0);
U3(8) <= '1';
U3(7) <= '1';
U3(6) <= '1';
U3(5) <= '1';
U3(4) <= '1';
U3(3) <= '1';
U3(2) <= '1';
U3(1) <= '1';
U3(0) <= '1';
-------------------------------------------------------------------------------

---------------------------- Desplaza I :10 -----------------------------------
U2(11) <= I(1);
U2(10) <= I(0);
U2(9) <= '1';
U2(8) <= '1';
U2(7) <= '1';
U2(6) <= '1';
U2(5) <= '1';
U2(4) <= '1';
U2(3) <= '1';
U2(2) <= '1';
U2(1) <= '1';
U2(0) <= '1';
-------------------------------------------------------------------------------

---------------------------- Desplaza I :11 -----------------------------------
U1(11) <= I(0);
U1(10) <= '1';
U1(9) <= '1';
U1(8) <= '1';
U1(7) <= '1';
U1(6) <= '1';
U1(5) <= '1';
U1(4) <= '1';
U1(3) <= '1';
U1(2) <= '1';
```

```
U1(1) <= '1';
U1(0) <= '1';
-------------------------------------------------------------------------------
---------------------------- Sin desplazamiento ---------------------------------
U0 <= I;
---------------------------- Desplazamiento Total -------------------------------
U12 <= "000000000000";
-------------------------------------------------------------------------------
--*****************************************************************************



--*****************************************************************************
IU11(11) <= not  I(10);
IU11(10) <= I(9);
IU11(9) <= I(8);
IU11(8) <= I(7);
IU11(7) <= I(6);
IU11(6) <= I(5);
IU11(5) <= I(4);
IU11(4) <= I(3);
IU11(3) <= I(2);
IU11(2) <= I(1);
IU11(1) <= I(0);
IU11(0) <= '1';
---------------------------- Desplaza II :1 -------------------------------------
IU10(11) <= not  I(9);
IU10(10) <=  I(8);
IU10(9) <= I(7);
IU10(8) <= I(6);
IU10(7) <= I(5);
IU10(6) <= I(4);
IU10(5) <= I(3);
IU10(4) <= I(2);
IU10(3) <= I(1);
IU10(2) <= I(0);
IU10(1) <= '1';
IU10(0) <= '1';
-------------------------------------------------------------------------------

---------------------------- Desplaza II :2 -------------------------------------
IU9(11) <= not  I(8);
IU9(10) <=  I(7);
IU9(9) <= I(6);
IU9(8) <= I(5);
IU9(7) <= I(4);
IU9(6) <= I(3);
IU9(5) <= I(2);
IU9(4) <= I(1);
IU9(3) <= I(0);
IU9(2) <= '1';
```

```
IU9(1) <= '1';
IU9(0) <= '1';
-------------------------------------------------------------------------------

---------------------------- Desplaza II :3 -----------------------------------
IU8(11) <= not  I(7);
IU8(10) <=   I(6);
IU8(9) <= I(5);
IU8(8) <= I(4);
IU8(7) <= I(3);
IU8(6) <= I(2);
IU8(5) <= I(1);
IU8(4) <= I(0);
IU8(3) <= '1';
IU8(2) <= '1';
IU8(1) <= '1';
IU8(0) <= '1';
-------------------------------------------------------------------------------

---------------------------- Desplaza II :4 -----------------------------------
IU7(11) <= not  I(6);
IU7(10) <=  I(5);
IU7(9) <= I(4);
IU7(8) <= I(3);
IU7(7) <= I(2);
IU7(6) <= I(1);
IU7(5) <= I(0);
IU7(4) <= '1';
IU7(3) <= '1';
IU7(2) <= '1';
IU7(1) <= '1';
IU7(0) <= '1';
-------------------------------------------------------------------------------

---------------------------- Desplaza II :5 -----------------------------------
IU6(11) <= not  I(5);
IU6(10) <=  I(4);
IU6(9) <= I(3);
IU6(8) <= I(2);
IU6(7) <= I(1);
IU6(6) <= I(0);
IU6(5) <= '1';
IU6(4) <= '1';
IU6(3) <= '1';
IU6(2) <= '1';
IU6(1) <= '1';
IU6(0) <= '1';
-------------------------------------------------------------------------------

---------------------------- Desplaza II :6 -----------------------------------
IU5(11) <= not I(4);
```

```
IU5(10) <=  I(3);
IU5(9) <= I(2);
IU5(8) <= I(1);
IU5(7) <= I(0);
IU5(6) <= '1';
IU5(5) <= '1';
IU5(4) <= '1';
IU5(3) <= '1';
IU5(2) <= '1';
IU5(1) <= '1';
IU5(0) <= '1';
-------------------------------------------------------------------------------

---------------------------- Desplaza II :7 -------------------------------------
IU4(11) <= not  I(3);
IU4(10) <=  I(2);
IU4(9) <= I(1);
IU4(8) <= I(0);
IU4(7) <= '1';
IU4(6) <= '1';
IU4(5) <= '1';
IU4(4) <= '1';
IU4(3) <= '1';
IU4(2) <= '1';
IU4(1) <= '1';
IU4(0) <= '1';
-------------------------------------------------------------------------------

---------------------------- Desplaza II :8 -------------------------------------
IU3(11) <= not  I(2);
IU3(10) <=  I(1);
IU3(9) <= I(0);
IU3(8) <= '1';
IU3(7) <= '1';
IU3(6) <= '1';
IU3(5) <= '1';
IU3(4) <= '1';
IU3(3) <= '1';
IU3(2) <= '1';
IU3(1) <= '1';
IU3(0) <= '1';
-------------------------------------------------------------------------------

---------------------------- Desplaza II :9 -------------------------------------
IU2(11) <= not  I(1);
IU2(10) <=  I(0);
IU2(9) <= '1';
IU2(8) <= '1';
IU2(7) <= '1';
IU2(6) <= '1';
IU2(5) <= '1';
```

```
IU2(4) <= '1';
IU2(3) <= '1';
IU2(2) <= '1';
IU2(1) <= '1';
IU2(0) <= '1';
-------------------------------------------------------------------------------

---------------------------- Desplaza II :10 ------------------------------------
IU1(11) <= not  I(0);
IU1(10) <=  '1';
IU1(9) <= '1';
IU1(8) <= '1';
IU1(7) <= '1';
IU1(6) <= '1';
IU1(5) <= '1';
IU1(4) <= '1';
IU1(3) <= '1';
IU1(2) <= '1';
IU1(1) <= '1';
IU1(0) <= '1';
-------------------------------------------------------------------------------
---------------------------- Sin desplazamiento ---------------------------------
IU0 <= I;
-------------------------------------------------------------------------------

--*****************************************************************************



O <=  U0 when S="000000" else
U11 when S="000001" else
U10 when S="000010" else
U9 when S="000011" else
U8 when S="000100" else
U7 when S="000101" else
U6 when S="000110" else
U5 when S="000111" else
U4 when S="001000" else
U3 when S="001001" else
U2 when S="001010" else
U1 when S="001011" else
U12 when S="001100" else
--**********************************
IU0 when S="010000" else
IU11 when S="010001" else
IU10 when S="010010" else
IU9 when S="010011" else
IU8 when S="010100" else
IU7 when S="010101" else
IU6 when S="010110" else
```

```
IU5 when S="010111" else
IU4 when S="011000" else
IU3 when S="011001" else
IU2 when S="011010" else
IU1 when S="011011" else
--************************************
IU0 when S="110000" else
IU11 when S="110001" else
IU10 when S="110010" else
IU9 when S="110011" else
IU8 when S="110100" else
IU7 when S="110101" else
IU6 when S="110110" else
IU5 when S="110111" else
IU4 when S="111000" else
IU3 when S="111001" else
IU2 when S="111010" else
IU1 when S="111011" else
--************************************
"000000000000";
end Behavioral;
```

Etiqueta

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library UNISIM;
use UNISIM.VComponents.all;

entity Etiqueta is
Port (
Inicio : in STD_LOGIC;
CLK : in STD_LOGIC;
Igual  :  in   STD_LOGIC_VECTOR (11 downto 0);
E3  :  in   STD_LOGIC_VECTOR (11 downto 0);
Advil  :  in   STD_LOGIC_VECTOR (11 downto 0);
IAcc : in STD_LOGIC_VECTOR (11 downto 0):="000000000000";
RUM : in STD_LOGIC:='0';
RUMDir : in STD_LOGIC_VECTOR (5 downto 0):="000000";
--*********************************************
OStart1 : out STD_LOGIC;
OStart2 : out STD_LOGIC;
OSWDir : out STD_LOGIC_VECTOR(5 downto 0);
OSRDir : out STD_LOGIC_VECTOR(5 downto 0);
OFlag1 : out STD_LOGIC;
OFlag2 : out STD_LOGIC;
OSE3 : out STD_LOGIC;
ORWRD : out STD_LOGIC;
--*********************************************
OAcc : out STD_LOGIC_VECTOR (11 downto 0):="000000000000";
MO : out STD_LOGIC:='0';
Flag : out STD_LOGIC:='0');
end Etiqueta;

architecture Behavioral of Etiqueta is
--****** Componentes **************
component R2_Advil is
    Port (  Advil : in STD_LOGIC_VECTOR (11 downto 0);
CLK  :  in   STD_LOGIC;
Start1 : in STD_LOGIC;
--*************************
RDir : out STD_LOGIC_VECTOR (5 downto 0);
RWE  :  out   STD_LOGIC;
RWCLK  :  out   STD_LOGIC;
RData  :  out   STD_LOGIC;
Flag1 : out STD_LOGIC);
end component;

component M0_R20 is
    Port (  Start2 : in STD_LOGIC;
```

```
SWDir : in STD_LOGIC_VECTOR (5 downto 0);
SRDir : in STD_LOGIC_VECTOR (5 downto 0);
CLK  :  in   STD_LOGIC;
R2O : in STD_LOGIC;
SE3 : in STD_LOGIC;
--************************
Flag2 : out STD_LOGIC;
R2Dir : out STD_LOGIC_VECTOR (5 downto 0);
--************************
MDir : out STD_LOGIC_VECTOR (5 downto 0);
MWE  :  out   STD_LOGIC;
MWCLK  :  out   STD_LOGIC;
MData  :  out   STD_LOGIC);
--************************
end component;
--************  Señales ************
signal MDir : STD_LOGIC_VECTOR (5 downto 0):="000000";
signal MData : STD_LOGIC:='0';
signal MWE : STD_LOGIC:='0';
signal MWCLK : STD_LOGIC:='0';

signal MMDir : STD_LOGIC_VECTOR (5 downto 0):="000000";
signal MMWE : STD_LOGIC:='0';
signal MMWCLK : STD_LOGIC:='0';

signal R2Dir : STD_LOGIC_VECTOR (5 downto 0):="000000";
signal RData : STD_LOGIC:='0';
signal RWE : STD_LOGIC:='0';
signal RWCLK : STD_LOGIC:='0';
signal R2O : STD_LOGIC:='0';

signal Dir : STD_LOGIC_VECTOR (5 downto 0):="000000";
signal Data : STD_LOGIC:='0';
signal WE : STD_LOGIC:='0';
signal WCLK : STD_LOGIC:='0';

signal RWRD : STD_LOGIC:='0';

signal R0  :     STD_LOGIC_VECTOR (11 downto 0):="000000000000";
signal R1  :     STD_LOGIC_VECTOR (11 downto 0):="000000000000";
signal i : STD_LOGIC_VECTOR (5 downto 0):="000000";
signal j : STD_LOGIC_VECTOR (5 downto 0):="000000";

signal Start1 : STD_LOGIC:='0';
signal Start2 : STD_LOGIC:='0';

signal Flag2 : STD_LOGIC:='0';
signal Flag1 : STD_LOGIC:='0';


signal SWDir : STD_LOGIC_VECTOR (5 downto 0):="000000";
```

```
signal SRDir : STD_LOGIC_VECTOR (5 downto 0):="000000";
signal RDir : STD_LOGIC_VECTOR (5 downto 0):="000000";
signal SE3 : STD_LOGIC:='0';
--**********************************
signal SIAcc : STD_LOGIC_VECTOR (5 downto 0):="000000";
--**********************************
type Estados is(d0,d1,d2,d3,dI1,dI2,dI3,
dI4,dE1,dE2,dE3,dE4,dend);
signal Edo_Presente, Edo_Futuro : Estados;
--**********************************
begin
--**********************************
OStart1 <= Start1;
OStart2 <= Start2;
OSWDir <= SWDir;
OSRDir <= SRDir;
OFlag1 <= Flag1;
OFlag2 <= Flag2;
OSE3 <= SE3;
ORWRD <= RWRD;
--**********************************
Dir <= RDir when RWRD = '1' else
R2Dir;
WE <= RWE when RWRD = '1' else
'0';
WCLK <= RWCLK when RWRD = '1' else
'0';
--*****************
MDir <= MMDir when RUM = '0' else
RUMDir;
MWE <= MMWE when RUM = '0' else
'0';
MWCLK <= MMWCLK when RUM = '0' else
'0';
--**********************************
R2   : RAM64X1S port map
(R2O,Dir(0),Dir(1),Dir(2),Dir(3),Dir(4),Dir(5),RData,WCLK,WE);
UM : RAM64X1S port map
(MO,MDir(0),MDir(1),MDir(2),MDir(3),MDir(4),MDir(5),MData,MWCLK,MWE);
--**********************************
U2 : R2_Advil port map
(Advil,CLK,Start1,RDir,RWE,RWCLK,RData,Flag1);
--**********************************
U3 : M0_R20 port map
(Start2,SWDir,SRDir,CLK,R2O,SE3,Flag2,R2Dir,MMDir,MMWE,MMWCLK,MData);
--**********************************
```

```
--*********************************
Marca: process(Inicio,Edo_Presente)
begin
--*********************************
if(Inicio='1')then
--*********************************
case Edo_Presente is
when d0 =>Flag <= '0';
R0 <= Igual;
R1 <= E3+IAcc;
RWRD <= '1';
Start1 <= '1';
Edo_Futuro <= d1;

--**** Pulso de Escritura ****
when d1 =>
--***********************
if (Flag1='1')then
RWRD <= '0';
Start1 <= '0';
--***********************
if(R0>"000000000000")then
i <= "000001";
j <= "000001";
SWDir <= "000001";
SRDir <= "000001";
SE3 <= '0';
Edo_Futuro <= d2;
else
Edo_Futuro <= dend;
end if;
--***********************
else
Edo_Futuro <= d0;
end if;
--***********************
when d2 =>Start2 <= '1';
Edo_Futuro <= d3;

when d3 =>
--***********************
if (Flag2='1')then
Start2 <= '0';
Edo_Futuro <= dE1;
else
Edo_Futuro <= d2;
end if;
--***********************
```

```
--****************************
when dI1 =>Flag <= '0';
if(R0 >i)then
SWDir <= j + i ;
SRDir <= i+"000001";
SE3 <= '0';
Edo_Futuro <= dI2;
else
Edo_Futuro <= dend;
end if;
--****************************
when dI2 =>Start2 <= '1';
Edo_Futuro <= dI3;
--****************************
when dI3 =>
--**********************
if (Flag2='1')then
Start2 <= '0';
Edo_Futuro <= dI4;
else
Edo_Futuro <= dI2;
end if;
--**********************
when dI4 =>i <= i + "000001";
Edo_Futuro <= dI1;
--****************************
```

```
--****************************
when dE1 =>
--**********************
if(R1 >"000000000000")then
j <= j  + "000001";
R1 <= R1 - "000000000001";
Edo_Futuro <= dE2;
else
Edo_Futuro <= dI1;
end  if;
--****************************
when dE2 =>SWDir <= j;
SRDir <= "000001";
SE3 <= '1';
Edo_Futuro <= dE3;
--****************************
when dE3 =>Start2 <= '1';
Edo_Futuro <= dE4;
--****************************
when dE4 =>
```

```
--**********************
if (Flag2='1')then
Start2 <= '0';
Edo_Futuro <= dE1;
else
Edo_Futuro <= dE3;
end if;
--**********************
--***************************
```

```
--***************************
when dend =>R0 <= "000000000000";
i <= "000000";
j <= "000000";
OAcc <= R1;
Flag <= '1';
end case;
--*********************************
else
Edo_Futuro <= d0;
end if;
--*********************************
end process;
--*********************************
--*********************************
Control: process(CLK)
begin
--*********************************
if(CLK'event and CLK='1')then
Edo_Presente <= Edo_Futuro;
end if;
--*********************************
end process Control;
--*********************************
end Behavioral;
```

M0_R20

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity M0_R20 is
    Port (  Start2 : in STD_LOGIC;
SWDir : in STD_LOGIC_VECTOR (5 downto 0);
SRDir : in STD_LOGIC_VECTOR (5 downto 0);
CLK  :  in   STD_LOGIC;
R2O : in STD_LOGIC:='0';
SE3 : in STD_LOGIC:='0';
--************************
Flag2 : out STD_LOGIC:='0';
R2Dir : out STD_LOGIC_VECTOR (5 downto 0);
--************************
MDir : out STD_LOGIC_VECTOR (5 downto 0);
MWE  :  out   STD_LOGIC:='0';
MWCLK  :  out   STD_LOGIC:='0';
MData  :  out   STD_LOGIC:='0');
--************************
end M0_R20;

architecture Behavioral of M0_R20 is
--***********************************
type EstadosW is(d0,d1,d2,d3);
signal Edo_PresenteW, Edo_FuturoW : EstadosW;
--***********************************
begin
--***********************************
-- Se propone las condiciones para la lectura
-- de R2.
R2Dir <= SRDir;

MDir <= SWDir;
--***********************************
Mem: process(Start2,Edo_PresenteW)
begin

if(Start2='1')then

case Edo_PresenteW is
--when d0 =>Flag2 <= '0';
--Edo_FuturoW <= d1;
--***************************
when d0 =>Flag2 <= '0';
MWE <= '1';
MWCLK <= '0';
if(SE3='0')then
```

```
Mdata <= R2O;
else
Mdata <= not(R2O);
end  if;
Edo_FuturoW <= d1;

--**** Pulso de Escritura ****
when d1 =>Flag2 <= '0';
MWCLK <= '1';
Edo_FuturoW <= d2;

when d2 =>MWCLK <= '0';
Edo_FuturoW <= d3;
--****************************

when d3 =>Flag2 <= '1';
MWE <= '0';
--****************************
end case;
else
Edo_FuturoW <= d0;
end if;
end process;
--*********************************
Control: process(CLK)
begin
--*********************************
if(CLK'event and CLK='1')then
Edo_PresenteW <= Edo_FuturoW;
end if;
--*********************************
end process Control;
--*********************************
end Behavioral;
```

R2_ Advil

```
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity R2_Advil is
    Port (  Advil : in STD_LOGIC_VECTOR (11 downto 0);
CLK  :  in   STD_LOGIC;
Start1 : in STD_LOGIC;
--*************************
RDir : out STD_LOGIC_VECTOR (5 downto 0):="000000";
RWE  :  out   STD_LOGIC:='0';
RWCLK  :  out   STD_LOGIC:='0';
RData  :  out   STD_LOGIC:='0';
Flag1 : out STD_LOGIC:='0');
end R2_Advil;

architecture Behavioral of R2_Advil is
--***********************************
type EstadosW is(d0,d1,d2,d3,d4,d5,d6,d7,d8,d9,d10,d11,d12,d13,
d14,d15,d16,d17,d18,d19,d20,d21,d22,d23,d24,
d25,d26,d27,d28,d29,d30,d31,d32,d33,d34,d35,d36);
signal Edo_PresenteW, Edo_FuturoW : EstadosW;
--***********************************
begin
--***********************************
Mem: process(Start1,Edo_PresenteW)
begin
if(Start1='1')then

case Edo_PresenteW is
when d0 =>Flag1 <= '0';
RWE <= '1';
RDir <= "000001";
RData <= Advil(11);
RWCLK <= '0';
Edo_FuturoW <= d1;

--**** Pulso de Escritura ****
when d1 =>RWCLK <= '1';
Edo_FuturoW <= d2;

when d2 =>RWCLK <= '0';
Edo_FuturoW <= d3;
--***************************

when d3 =>RWE <= '1';
RDir <= "000010";
RData <= Advil(10);
```

```
Edo_FuturoW <= d4;

--**** Pulso de Escritura ****
when d4 =>RWCLK <= '1';
Edo_FuturoW <= d5;

when d5 =>RWCLK <= '0';
Edo_FuturoW <= d6;
--****************************

when d6 =>RWE <= '1';
RDir <= "000011";
RData <= Advil(9);
Edo_FuturoW <= d7;

--**** Pulso de Escritura ****
when d7 =>RWCLK <= '1';
Edo_FuturoW <= d8;

when d8 =>RWCLK <= '0';
Edo_FuturoW <= d9;
--****************************

when d9 =>RWE <= '1';
RDir <= "000100";
RData <= Advil(8);
Edo_FuturoW <= d10;

--**** Pulso de Escritura ****
when d10 =>RWCLK <= '1';
Edo_FuturoW <= d11;

when d11 =>RWCLK <= '0';
Edo_FuturoW <= d12;
--****************************

when d12 =>RWE <= '1';
RDir <= "000101";
RData <= Advil(7);
Edo_FuturoW <= d13;

--**** Pulso de Escritura ****
when d13 =>RWCLK <= '1';
Edo_FuturoW <= d14;

when d14 =>RWCLK <= '0';
Edo_FuturoW <= d15;
--****************************

when d15 =>RWE <= '1';
RDir <= "000110";
```

```
RData <= Advil(6);
Edo_FuturoW <= d16;

--**** Pulso de Escritura ****
when d16 =>RWCLK <= '1';
Edo_FuturoW <= d17;

when d17 =>RWCLK <= '0';
Edo_FuturoW <= d18;
--**************************

when d18 =>RWE <= '1';
RDir <= "000111";
RData <= Advil(5);
Edo_FuturoW <= d19;

--**** Pulso de Escritura ****
when d19 =>RWCLK <= '1';
Edo_FuturoW <= d20;

when d20 =>RWCLK <= '0';
Edo_FuturoW <= d21;
--**************************

when d21 =>RWE <= '1';
RDir <= "001000";
RData <= Advil(4);
Edo_FuturoW <= d22;

--**** Pulso de Escritura ****
when d22 =>RWCLK <= '1';
Edo_FuturoW <= d23;

when d23 =>RWCLK <= '0';
Edo_FuturoW <= d24;
--**************************

when d24 =>RWE <= '1';
RDir <= "001001";
RData <= Advil(3);
Edo_FuturoW <= d25;

--**** Pulso de Escritura ****
when d25 =>RWCLK <= '1';
Edo_FuturoW <= d26;

when d26 =>RWCLK <= '0';
Edo_FuturoW <= d27;
--**************************

when d27 =>RWE <= '1';
```

```
RDir <= "001010";
RData <= Advil(2);
Edo_FuturoW <= d28;

--**** Pulso de Escritura ****
when d28 =>RWCLK <= '1';
Edo_FuturoW <= d29;

when d29 =>RWCLK <= '0';
Edo_FuturoW <= d30;
--*****************************

when d30 =>RWE <= '1';
RDir <= "001011";
RData <= Advil(1);
Edo_FuturoW <= d31;

--**** Pulso de Escritura ****
when d31 =>RWCLK <= '1';
Edo_FuturoW <= d32;

when d32 =>RWCLK <= '0';
Edo_FuturoW <= d33;
--*****************************

when d33 =>RWE <= '1';
RDir <= "001100";
RData <= Advil(0);
Edo_FuturoW <= d34;

--**** Pulso de Escritura ****
when d34 =>RWCLK <= '1';
Edo_FuturoW <= d35;

when d35 =>RWCLK <= '0';
Edo_FuturoW <= d36;
--*****************************

when d36 =>Flag1 <= '1';
RWE <= '0';
end case;
else
Edo_FuturoW <= d0;
end if;
end process;


--*********************************
Control: process(CLK)
begin
--*********************************
```

```
if(CLK'event and CLK='1')then
Edo_PresenteW <= Edo_FuturoW;
end if;
--**********************************
end process Control;
--**********************************
end Behavioral;
```

Control

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Control is
Port ( OAcc  : in  STD_LOGIC_VECTOR (11 downto 0);
           En  : in  STD_LOGIC;
           CLK  : in  STD_LOGIC;
           Flag  : in  STD_LOGIC;
  --**********************
           Inicio : out  STD_LOGIC:='0';
           IAcc  : out  STD_LOGIC_VECTOR (11 downto 0):="000000000000";
           Oflag  : out  STD_LOGIC:='0');
end Control;

architecture Behavioral of Control is
--***********************************
type Estados is(d0,d1,d2,d3);
signal Edo_Presente, Edo_Futuro : Estados;
--***********************************
begin
--***********************************
Control: process(En,Edo_Presente)
begin
--***********************************
if(En='1')then
--***********************************
case Edo_Presente is
when d0 =>Oflag <= '0';
IAcc <= OAcc;
Edo_Futuro <= d1;

when d1 => Oflag <= '0';
Inicio <= '1';
Edo_Futuro <= d2;

when d2 =>Oflag <= '0';
if(Flag = '1')then
Edo_Futuro <= d3;
else
Edo_Futuro <= d1;
end if;
when d3 =>Oflag <= '1';

end case;
--***********************************
else
Edo_Futuro <= d0;
```

```
end if;
--**********************************

end process Control;
--**********************************
Reloj : process(CLK)
begin
--**********************************
if(CLK'event and CLK='1')then
Edo_Presente <= Edo_Futuro;
end if;
--**********************************
end process Reloj;
--**********************************
end Behavioral;
```

Dispersor

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Dispersor is
Port (  b0 : in   STD_LOGIC_VECTOR(7 downto 0);
b1 : in   STD_LOGIC_VECTOR(7 downto 0);
b2 : in   STD_LOGIC_VECTOR(7 downto 0);
C0 : out   STD_LOGIC_VECTOR(7 downto 0);
C1 : out   STD_LOGIC_VECTOR(7 downto 0);
C2 : out   STD_LOGIC_VECTOR(7 downto 0);
C3 : out   STD_LOGIC_VECTOR(7 downto 0);
C4 : out   STD_LOGIC_VECTOR(7 downto 0));
end Dispersor;

architecture Behavioral of Dispersor is
--**********************************************************************************
component SemiMultiplicador is
port ( In_A : in STD_LOGIC_VECTOR(7 downto 0);
In_B : in STD_LOGIC_VECTOR(7 downto 0);
In_Or : in STD_LOGIC_VECTOR(7 downto 0);
Mul : out STD_LOGIC_VECTOR(7 downto 0));
end component;


component Logaritmo is
Port  (  A  : in  STD_LOGIC_VECTOR(7 downto 0);
LOGA  : out STD_LOGIC_VECTOR(7 downto 0));
end component;
--**********************************************************************************
signal Log_b0  :    STD_LOGIC_VECTOR(7 downto 0);
signal Log_b1  :    STD_LOGIC_VECTOR(7 downto 0);
signal Log_b2  :    STD_LOGIC_VECTOR(7 downto 0);

signal M0   :    STD_LOGIC_VECTOR(7 downto 0);
signal M1   :    STD_LOGIC_VECTOR(7 downto 0);
signal M2   :    STD_LOGIC_VECTOR(7 downto 0);
signal M3   :    STD_LOGIC_VECTOR(7 downto 0);
signal M4   :    STD_LOGIC_VECTOR(7 downto 0);
signal M5   :    STD_LOGIC_VECTOR(7 downto 0);
signal M6   :    STD_LOGIC_VECTOR(7 downto 0);
signal M7   :    STD_LOGIC_VECTOR(7 downto 0);

signal a00   :    STD_LOGIC_VECTOR(7 downto 0):=B"11010101";-- Caracter ASCII 49
signal a01   :    STD_LOGIC_VECTOR(7 downto 0):=B"10000111";-- Caracter ASCII 51
signal a02   :    STD_LOGIC_VECTOR(7 downto 0):=B"11110110";-- Caracter ASCII 50
signal a11   :    STD_LOGIC_VECTOR(7 downto 0):=B"11010101";-- Caracter ASCII 49
signal a12   :    STD_LOGIC_VECTOR(7 downto 0):=B"11010101";-- Caracter ASCII 49
```

```
signal a20   :    STD_LOGIC_VECTOR(7 downto 0):=B"11110110";-- Caracter ASCII 50
signal a31   :    STD_LOGIC_VECTOR(7 downto 0):=B"11110110";-- Caracter ASCII 50
signal a32   :    STD_LOGIC_VECTOR(7 downto 0):=B"10000111";-- Caracter ASCII 51
--*************************************************************************
begin
--*************************************************************************
U0 : Logaritmo port map(b0,Log_b0);
U1 : Logaritmo port map(b1,Log_b1);
U2 : Logaritmo port map(b2,Log_b2);
--*************************************************************************
U3 : SemiMultiplicador port map(Log_b0,a00,b0,M0);
U4 : SemiMultiplicador port map(Log_b1,a01,b1,M1);
U5 : SemiMultiplicador port map(Log_b2,a02,b2,M2);
C0 <= M0 xor M1 xor M2;
--*************************************************************************
U6 : SemiMultiplicador port map(Log_b1,a11,b1,M3);
U7 : SemiMultiplicador port map(Log_b2,a12,b2,M4);
C1 <= M0 xor M3 xor M4;
--*************************************************************************
U8 : SemiMultiplicador port map(Log_b0,a20,b0,M5);
C2 <= M5 xor M1 xor M4;
--*************************************************************************
U9 : SemiMultiplicador port map(Log_b1,a31,b1,M6);
U10 : SemiMultiplicador port map(Log_b2,a32,b2,M7);
C3 <= M5 xor M6 xor M7;
--*************************************************************************
C4 <= M5 xor M1 xor M7;
--*************************************************************************
end Behavioral;
```

SemiMultiplicador

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity SemiMultiplicador is
    Port (  A  :  in   STD_LOGIC_VECTOR(7 downto 0);
B  :  in   STD_LOGIC_VECTOR(7 downto 0);
C  :  in   STD_LOGIC_VECTOR(7 downto 0);
X  :  out   STD_LOGIC_VECTOR(7 downto 0));
end SemiMultiplicador;

architecture Behavioral of SemiMultiplicador is
--*************************************************************************
component Sumador is
port ( A  : in STD_LOGIC_VECTOR(7 downto 0);
B  : in STD_LOGIC_VECTOR(7 downto 0);
S  : out STD_LOGIC_VECTOR(7 downto 0);
Ci  : in STD_LOGIC;
```

```
Co  : out STD_LOGIC);
end component;

component Comparador is
Port ( In0 : in STD_LOGIC;
In1  : in STD_LOGIC;
In2  : in STD_LOGIC;
In3  : in STD_LOGIC;
In4  : in STD_LOGIC;
In5  : in STD_LOGIC;
In6  : in STD_LOGIC;
In7  : in STD_LOGIC;
Out0  : out STD_LOGIC;
Out1  : out STD_LOGIC;
Out2  : out STD_LOGIC;
Out3  : out STD_LOGIC;
Out4  : out STD_LOGIC;
Out5  : out STD_LOGIC;
Out6  : out STD_LOGIC;
Out7  : out STD_LOGIC);
end component;

component AntiLogaritmo is
Port ( In0  : in STD_LOGIC;
In1  : in STD_LOGIC;
In2 : in STD_LOGIC;
In3 : in STD_LOGIC;
In4  : in STD_LOGIC;
In5  : in STD_LOGIC;
In6  : in STD_LOGIC;
In7  : in STD_LOGIC;
Out0  : out STD_LOGIC;
Out1  : out STD_LOGIC;
Out2  : out STD_LOGIC;
Out3  : out STD_LOGIC;
Out4  : out STD_LOGIC;
Out5  : out STD_LOGIC;
Out6  : out STD_LOGIC;
Out7  : out STD_LOGIC);
end component;

component Compuerta is
port (In0  : in STD_LOGIC;
In1  : in STD_LOGIC;
In2  : in STD_LOGIC;
In3  : in STD_LOGIC;
In4  : in STD_LOGIC;
In5  : in STD_LOGIC;
In6  : in STD_LOGIC;
In7  : in STD_LOGIC;
Inhividor: in STD_LOGIC;
```

```
Out0  : out STD_LOGIC;
Out1 : out STD_LOGIC;
Out2  : out STD_LOGIC;
Out3  : out STD_LOGIC;
Out4  : out STD_LOGIC;
Out5 : out STD_LOGIC;
Out6  : out STD_LOGIC;
Out7  : out STD_LOGIC);
end component;
--*********************************************************************************
constant Dos_cinco_cinco  : STD_LOGIC_VECTOR(7 downto 0)  := "11111111";
constant Cin  : STD_LOGIC  := '0';
--*********************************************************************************
signal Cero_A  : STD_LOGIC_VECTOR(7 downto 0);
signal Sal_A  : STD_LOGIC;
signal Suma_A  : STD_LOGIC_VECTOR(7 downto 0);
signal Cero_B  : STD_LOGIC_VECTOR(7 downto 0);
signal Sal_B  : STD_LOGIC;
signal Suma_B  : STD_LOGIC_VECTOR(7 downto 0);
signal Cout  : STD_LOGIC;
signal S_A  : STD_LOGIC_VECTOR(7 downto 0);
signal Op_B  : STD_LOGIC_VECTOR(7 downto 0);
signal Cout_B  : STD_LOGIC;
signal S_B  : STD_LOGIC_VECTOR(7 downto 0);
signal And_A  : STD_LOGIC_VECTOR(7 downto 0);
signal Inhividor  : STD_LOGIC;
--*********************************************************************************
begin
--*********************************************************************************
Suma_A(0) <= A(0);
Suma_A(1) <= A(1);
Suma_A(2) <= A(2);
Suma_A(3) <= A(3);
Suma_A(4) <= A(4);
Suma_A(5) <= A(5);
Suma_A(6) <= A(6);
Suma_A(7) <= A(7);

Suma_B(0) <= B(0);
Suma_B(1) <= B(1);
Suma_B(2) <= B(2);
Suma_B(3) <= B(3);
Suma_B(4) <= B(4);
Suma_B(5) <= B(5);
Suma_B(6) <= B(6);
Suma_B(7) <= B(7);


Inhividor <= not( (((C(0) or C(1)) or (C(2) or C(3)))
or ((C(4) or C(5)) or (C(6) or C(7))) );
--*********************************************************************************
```

```
U0 : Sumador  port map(Suma_A,Suma_B,S_A,Cin,Cout);

U1 : Comparador  port map(S_A(0),S_A(1),S_A(2),S_A(3),S_A(4),S_A(5),S_A(6)
,S_A(7),Op_B(0),Op_B(1),Op_B(2),Op_B(3),Op_B(4),Op_B(5)
,Op_B(6),Op_B(7));

U2 : Sumador  port map(S_A,Op_B,S_B,Cout,Cout_B);

U3 : AntiLogaritmo  port map(S_B(0),S_B(1),S_B(2),S_B(3),S_B(4),S_B(5),S_B(6)
,S_B(7),And_A(0),And_A(1),And_A(2),And_A(3),And_A(4)
,And_A(5),And_A(6),And_A(7));

U4 : Compuerta  port map(And_A(0),And_A(1),And_A(2),And_A(3),And_A(4),And_A(5)
,And_A(6),And_A(7),Inhividor,Mul(0),Mul(1),Mul(2),Mul(3)
,Mul(4),Mul(5),Mul(6),Mul(7));
--*********************************************************************************

--*********************************************************************************
end Behavioral;
```

Constructor

```
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Constructor is
    Port (  b0 : in   STD_LOGIC_VECTOR(7 downto 0);
b1 : in   STD_LOGIC_VECTOR(7 downto 0);
b2 : in   STD_LOGIC_VECTOR(7 downto 0);
--************************************
R0 : in   STD_LOGIC_VECTOR(7 downto 0);
R1 : in   STD_LOGIC_VECTOR(7 downto 0);
R2 : in   STD_LOGIC_VECTOR(7 downto 0);
--************************************
C0 : out   STD_LOGIC_VECTOR(7 downto 0);
C1 : out   STD_LOGIC_VECTOR(7 downto 0);
C2 : out   STD_LOGIC_VECTOR(7 downto 0));
end Constructor;

architecture Behavioral of Constructor is
--*********************************************************************************
component SemiMultiplicador is
port ( In_A : in STD_LOGIC_VECTOR(7 downto 0);
In_B : in STD_LOGIC_VECTOR(7 downto 0);
In_Or : in STD_LOGIC_VECTOR(7 downto 0);
Mul : out STD_LOGIC_VECTOR(7 downto 0));
end component;
```

```
component Logaritmo is
Port  (  A  : in  STD_LOGIC_VECTOR(7 downto 0);
LOGA  : out STD_LOGIC_VECTOR(7 downto 0));
end component;
--*****************************************************************************
signal Log_b0  :    STD_LOGIC_VECTOR(7 downto 0);
signal Log_b1  :    STD_LOGIC_VECTOR(7 downto 0);
signal Log_b2  :    STD_LOGIC_VECTOR(7 downto 0);

signal M0   :    STD_LOGIC_VECTOR(7 downto 0);
signal M1   :    STD_LOGIC_VECTOR(7 downto 0);
signal M2   :    STD_LOGIC_VECTOR(7 downto 0);
signal M3   :    STD_LOGIC_VECTOR(7 downto 0);
signal M4   :    STD_LOGIC_VECTOR(7 downto 0);
signal M5   :    STD_LOGIC_VECTOR(7 downto 0);
signal M6   :    STD_LOGIC_VECTOR(7 downto 0);
signal M7   :    STD_LOGIC_VECTOR(7 downto 0);
signal M8   :    STD_LOGIC_VECTOR(7 downto 0);

signal a_1_00   :    STD_LOGIC_VECTOR(7 downto 0):=B"00110001";-- Caracter ASCII 49
signal a_1_01   :    STD_LOGIC_VECTOR(7 downto 0):=B"11001001";-- Caracter ASCII 201
signal a_1_02   :    STD_LOGIC_VECTOR(7 downto 0):=B"00110000";-- Caracter ASCII 48
signal a_1_10   :    STD_LOGIC_VECTOR(7 downto 0):=B"00011000";-- Caracter ASCII 24
signal a_1_11   :    STD_LOGIC_VECTOR(7 downto 0):=B"00101010";-- Caracter ASCII 42
signal a_1_12   :    STD_LOGIC_VECTOR(7 downto 0):=B"00011000";-- Caracter ASCII 24
signal a_1_20   :    STD_LOGIC_VECTOR(7 downto 0):=B"00110000";-- Caracter ASCII 48
signal a_1_21   :    STD_LOGIC_VECTOR(7 downto 0):=B"11001001";-- Caracter ASCII 201
signal a_1_22   :    STD_LOGIC_VECTOR(7 downto 0):=B"00110001";-- Caracter ASCII 49

signal a_2_00   :    STD_LOGIC_VECTOR(7 downto 0):=B"11111110";-- Caracter ASCII 254
signal a_2_01   :    STD_LOGIC_VECTOR(7 downto 0):=B"00101001";-- Caracter ASCII 41
signal a_2_02   :    STD_LOGIC_VECTOR(7 downto 0):=B"11111110";-- Caracter ASCII 254
signal a_2_10   :    STD_LOGIC_VECTOR(7 downto 0):=B"11111110";-- Caracter ASCII 254
signal a_2_11   :    STD_LOGIC_VECTOR(7 downto 0):=B"10001111";-- Caracter ASCII 143
signal a_2_12   :    STD_LOGIC_VECTOR(7 downto 0):=B"11100110";-- Caracter ASCII 230
signal a_2_20   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000000";-- Caracter ASCII 0
signal a_2_21   :    STD_LOGIC_VECTOR(7 downto 0):=B"00100001";-- Caracter ASCII 33
signal a_2_22   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000000";-- Caracter ASCII 0

signal a_3_00   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000000";-- Caracter ASCII 0
signal a_3_01   :    STD_LOGIC_VECTOR(7 downto 0):=B"10110001";-- Caracter ASCII 177
signal a_3_02   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000000";-- Caracter ASCII 0
signal a_3_10   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000000";-- Caracter ASCII 0
signal a_3_11   :    STD_LOGIC_VECTOR(7 downto 0):=B"10010000";-- Caracter ASCII 144
signal a_3_12   :    STD_LOGIC_VECTOR(7 downto 0):=B"11100111";-- Caracter ASCII 231
signal a_3_20   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000000";-- Caracter ASCII 0
signal a_3_21   :    STD_LOGIC_VECTOR(7 downto 0):=B"10011001";-- Caracter ASCII 153
signal a_3_22   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000001";-- Caracter ASCII 1

signal a_4_00   :    STD_LOGIC_VECTOR(7 downto 0):=B"11010001";-- Caracter ASCII 209
signal a_4_01   :    STD_LOGIC_VECTOR(7 downto 0):=B"11100000";-- Caracter ASCII 224
```

```
signal a_4_02   :    STD_LOGIC_VECTOR(7 downto 0):=B"01010000";-- Caracter ASCII 80
signal a_4_10   :    STD_LOGIC_VECTOR(7 downto 0):=B"11011000";-- Caracter ASCII 216
signal a_4_11   :    STD_LOGIC_VECTOR(7 downto 0):=B"01000111";-- Caracter ASCII 71
signal a_4_12   :    STD_LOGIC_VECTOR(7 downto 0):=B"10110000";-- Caracter ASCII 176
signal a_4_20   :    STD_LOGIC_VECTOR(7 downto 0):=B"11010111";-- Caracter ASCII 215
signal a_4_21   :    STD_LOGIC_VECTOR(7 downto 0):=B"11011000";-- Caracter ASCII 216
signal a_4_22   :    STD_LOGIC_VECTOR(7 downto 0):=B"01010000";-- Caracter ASCII 80

signal a_5_00   :    STD_LOGIC_VECTOR(7 downto 0):=B"00011000";-- Caracter ASCII 24
signal a_5_01   :    STD_LOGIC_VECTOR(7 downto 0):=B"00010111";-- Caracter ASCII 23
signal a_5_02   :    STD_LOGIC_VECTOR(7 downto 0):=B"11111110";-- Caracter ASCII 254
signal a_5_10   :    STD_LOGIC_VECTOR(7 downto 0):=B"10000111";-- Caracter ASCII 135
signal a_5_11   :    STD_LOGIC_VECTOR(7 downto 0):=B"11110101";-- Caracter ASCII 245
signal a_5_12   :    STD_LOGIC_VECTOR(7 downto 0):=B"01011111";-- Caracter ASCII 95
signal a_5_20   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000000";-- Caracter ASCII 0
signal a_5_21   :    STD_LOGIC_VECTOR(7 downto 0):=B"11111110";-- Caracter ASCII 254
signal a_5_22   :    STD_LOGIC_VECTOR(7 downto 0):=B"11111110";-- Caracter ASCII 254

signal a_6_00   :    STD_LOGIC_VECTOR(7 downto 0):=B"00100001";-- Caracter ASCII 33
signal a_6_01   :    STD_LOGIC_VECTOR(7 downto 0):=B"00100001";-- Caracter ASCII 33
signal a_6_02   :    STD_LOGIC_VECTOR(7 downto 0):=B"10011001";-- Caracter ASCII 153
signal a_6_10   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000000";-- Caracter ASCII 0
signal a_6_11   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000000";-- Caracter ASCII 0
signal a_6_12   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000000";-- Caracter ASCII 0
signal a_6_20   :    STD_LOGIC_VECTOR(7 downto 0):=B"10010000";-- Caracter ASCII 144
signal a_6_21   :    STD_LOGIC_VECTOR(7 downto 0):=B"00001001";-- Caracter ASCII 9
signal a_6_22   :    STD_LOGIC_VECTOR(7 downto 0):=B"10010001";-- Caracter ASCII 145

signal a_7_00   :    STD_LOGIC_VECTOR(7 downto 0):=B"00011011";-- Caracter ASCII 27
signal a_7_01   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000000";-- Caracter ASCII 0
signal a_7_02   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000001";-- Caracter ASCII 1
signal a_7_10   :    STD_LOGIC_VECTOR(7 downto 0):=B"00100010";-- Caracter ASCII 34
signal a_7_11   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000000";-- Caracter ASCII 0
signal a_7_12   :    STD_LOGIC_VECTOR(7 downto 0):=B"11100111";-- Caracter ASCII 231
signal a_7_20   :    STD_LOGIC_VECTOR(7 downto 0):=B"00100001";-- Caracter ASCII 33
signal a_7_21   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000000";-- Caracter ASCII 0
signal a_7_22   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000000";-- Caracter ASCII 0



signal a_8_00   :    STD_LOGIC_VECTOR(7 downto 0):=B"10110001";-- Caracter ASCII 177
signal a_8_01   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000000";-- Caracter ASCII 0
signal a_8_02   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000000";-- Caracter ASCII 0
signal a_8_10   :    STD_LOGIC_VECTOR(7 downto 0):=B"00100001";-- Caracter ASCII 33
signal a_8_11   :    STD_LOGIC_VECTOR(7 downto 0):=B"11111110";-- Caracter ASCII 254
signal a_8_12   :    STD_LOGIC_VECTOR(7 downto 0):=B"11100110";-- Caracter ASCII 230
signal a_8_20   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000000";-- Caracter ASCII 0
signal a_8_21   :    STD_LOGIC_VECTOR(7 downto 0):=B"11111110";-- Caracter ASCII 254
signal a_8_22   :    STD_LOGIC_VECTOR(7 downto 0):=B"11111110";-- Caracter ASCII 254
```

```
signal a_9_00   :    STD_LOGIC_VECTOR(7 downto 0):=B"10110001";-- Caracter ASCII 177
signal a_9_01   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000000";-- Caracter ASCII 0
signal a_9_02   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000000";-- Caracter ASCII 0
signal a_9_10   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000000";-- Caracter ASCII 0
signal a_9_11   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000000";-- Caracter ASCII 0
signal a_9_12   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000000";-- Caracter ASCII 0
signal a_9_20   :    STD_LOGIC_VECTOR(7 downto 0):=B"00100001";-- Caracter ASCII 33
signal a_9_21   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000000";-- Caracter ASCII 0
signal a_9_22   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000000";-- Caracter ASCII 0


signal a_10_00   :    STD_LOGIC_VECTOR(7 downto 0):=B"10001111";-- Caracter ASCII 143
signal a_10_01   :    STD_LOGIC_VECTOR(7 downto 0):=B"10010000";-- Caracter ASCII 144
signal a_10_02   :    STD_LOGIC_VECTOR(7 downto 0):=B"11111000";-- Caracter ASCII 248
signal a_10_10   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000000";-- Caracter ASCII 0
signal a_10_11   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000000";-- Caracter ASCII 0
signal a_10_12   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000000";-- Caracter ASCII 0
signal a_10_20   :    STD_LOGIC_VECTOR(7 downto 0):=B"11111110";-- Caracter ASCII 254
signal a_10_21   :    STD_LOGIC_VECTOR(7 downto 0):=B"00000000";-- Caracter ASCII 0
signal a_10_22   :    STD_LOGIC_VECTOR(7 downto 0):=B"11111110";-- Caracter ASCII 254



signal a00 : STD_LOGIC_VECTOR(7 downto 0);
signal a01 : STD_LOGIC_VECTOR(7 downto 0);
signal a02 : STD_LOGIC_VECTOR(7 downto 0);

signal a10 : STD_LOGIC_VECTOR(7 downto 0);
signal a11 : STD_LOGIC_VECTOR(7 downto 0);
signal a12 : STD_LOGIC_VECTOR(7 downto 0);

signal a20 : STD_LOGIC_VECTOR(7 downto 0);
signal a21 : STD_LOGIC_VECTOR(7 downto 0);
signal a22 : STD_LOGIC_VECTOR(7 downto 0);

signal Seleccion : STD_LOGIC_VECTOR(3 downto 0);



signal Inhividor_1 : STD_LOGIC_VECTOR(7 downto 0);
signal Inhividor_2 : STD_LOGIC_VECTOR(7 downto 0);
signal Inhividor_3 : STD_LOGIC_VECTOR(7 downto 0);

signal Inhividor_4 : STD_LOGIC_VECTOR(7 downto 0);
signal Inhividor_5 : STD_LOGIC_VECTOR(7 downto 0);
signal Inhividor_6 : STD_LOGIC_VECTOR(7 downto 0);

signal Inhividor_7 : STD_LOGIC_VECTOR(7 downto 0);
signal Inhividor_8 : STD_LOGIC_VECTOR(7 downto 0);
signal Inhividor_9 : STD_LOGIC_VECTOR(7 downto 0);
--**********************************************************************************
```

```
begin
--*************************************************************************
--Esta seccion esta por  que la inverza de la matriz genera datos cero,
--y esto confunde al multiplicador,ya que este fue diseñado a haceptar
--datos ya transformados en logaritmos. El problema es que si no tomamos
--medidas, el multpilicador pienza que el cero ya es el logaritmo, generando
--un posible antilogaritmo

Inhividor_1 <="00000000" when Seleccion="0011" else
b0;

Inhividor_2 <="00000000" when (Seleccion="1000")or(Seleccion="1001")
else b1;
Inhividor_3 <=b2;

Inhividor_4 <="00000000" when (Seleccion="0110")or(Seleccion="1001")or(Seleccion="1010")
else b0;
Inhividor_5 <=b1;
Inhividor_6 <=b2;

Inhividor_7 <="00000000" when (Seleccion="0010")or(Seleccion="0101")or(Seleccion="1000")
else b0;
Inhividor_8 <="00000000" when (Seleccion="0111")or(Seleccion="1010")
else b1;
Inhividor_9 <="00000000" when (Seleccion="1001")
else b2;
--*************************************************************************
--Esta seccion nos ayuda a interpretar que dispersos fueron perdidos, la
--combinacion de los dispersos recuperados la dan R0,R1,R2.
--Cado uno indica el renglon recuperado.
Seleccion <= "0001" when (R0="00000001" and R1="00000010" and R2="00000011") else
"0010" when (R0="00000001" and R1="00000010" and R2="00000100") else
"0011" when (R0="00000001" and R1="00000010" and R2="00000101") else
"0100" when (R0="00000001" and R1="00000011" and R2="00000100") else
"0101" when (R0="00000001" and R1="00000011" and R2="00000101") else
"0110" when (R0="00000001" and R1="00000100" and R2="00000101") else
"0111" when (R0="00000010" and R1="00000011" and R2="00000100") else
"1000" when (R0="00000010" and R1="00000011" and R2="00000101") else
"1001" when (R0="00000010" and R1="00000100" and R2="00000101") else
"1010" when (R0="00000011" and R1="00000100" and R2="00000101") else
"0000";
--*************************************************************************
--Esto se hace para que se puedan direccionar todas las inversas de las matrices
a00 <= a_1_00 when Seleccion="0001" else
a_2_00 when Seleccion="0010" else
a_3_00 when Seleccion="0011" else
a_4_00 when Seleccion="0100" else
a_5_00 when Seleccion="0101" else
a_6_00 when Seleccion="0110" else
a_7_00 when Seleccion="0111" else
a_8_00 when Seleccion="1000" else
```

```
a_9_00 when Seleccion="1001" else
a_10_00 when Seleccion="1010" else
"00000000";

a01 <= a_1_01 when Seleccion="0001" else
a_2_01 when Seleccion="0010" else
a_3_01 when Seleccion="0011" else
a_4_01 when Seleccion="0100" else
a_5_01 when Seleccion="0101" else
a_6_01 when Seleccion="0110" else
a_7_01 when Seleccion="0111" else
a_8_01 when Seleccion="1000" else
a_9_01 when Seleccion="1001" else
a_10_01 when Seleccion="1010" else
"00000000";

a02 <= a_1_02 when Seleccion="0001" else
a_2_02 when Seleccion="0010" else
a_3_02 when Seleccion="0011" else
a_4_02 when Seleccion="0100" else
a_5_02 when Seleccion="0101" else
a_6_02 when Seleccion="0110" else
a_7_02 when Seleccion="0111" else
a_8_02 when Seleccion="1000" else
a_9_02 when Seleccion="1001" else
a_10_02 when Seleccion="1010" else
"00000000";

a10 <= a_1_10 when Seleccion="0001" else
a_2_10 when Seleccion="0010" else
a_3_10 when Seleccion="0011" else
a_4_10 when Seleccion="0100" else
a_5_10 when Seleccion="0101" else
a_6_10 when Seleccion="0110" else
a_7_10 when Seleccion="0111" else
a_8_10 when Seleccion="1000" else
a_9_10 when Seleccion="1001" else
a_10_10 when Seleccion="1010" else
"00000000";

a11 <= a_1_11 when Seleccion="0001" else
a_2_11 when Seleccion="0010" else
a_3_11 when Seleccion="0011" else
a_4_11 when Seleccion="0100" else
a_5_11 when Seleccion="0101" else
a_6_11 when Seleccion="0110" else
a_7_11 when Seleccion="0111" else
a_8_11 when Seleccion="1000" else
a_9_11 when Seleccion="1001" else
a_10_11 when Seleccion="1010" else
"00000000";
```

```
a12 <= a_1_12 when Seleccion="0001" else
a_2_12 when Seleccion="0010" else
a_3_12 when Seleccion="0011" else
a_4_12 when Seleccion="0100" else
a_5_12 when Seleccion="0101" else
a_6_12 when Seleccion="0110" else
a_7_12 when Seleccion="0111" else
a_8_12 when Seleccion="1000" else
a_9_12 when Seleccion="1001" else
a_10_12 when Seleccion="1010" else
"00000000";

a20 <= a_1_20 when Seleccion="0001" else
a_2_20 when Seleccion="0010" else
a_3_20 when Seleccion="0011" else
a_4_20 when Seleccion="0100" else
a_5_20 when Seleccion="0101" else
a_6_20 when Seleccion="0110" else
a_7_20 when Seleccion="0111" else
a_8_20 when Seleccion="1000" else
a_9_20 when Seleccion="1001" else
a_10_20 when Seleccion="1010" else
"00000000";

a21 <= a_1_21 when Seleccion="0001" else
a_2_21 when Seleccion="0010" else
a_3_21 when Seleccion="0011" else
a_4_21 when Seleccion="0100" else
a_5_21 when Seleccion="0101" else
a_6_21 when Seleccion="0110" else
a_7_21 when Seleccion="0111" else
a_8_21 when Seleccion="1000" else
a_9_21 when Seleccion="1001" else
a_10_21 when Seleccion="1010" else
"00000000";

a22 <= a_1_22 when Seleccion="0001" else
a_2_22 when Seleccion="0010" else
a_3_22 when Seleccion="0011" else
a_4_22 when Seleccion="0100" else
a_5_22 when Seleccion="0101" else
a_6_22 when Seleccion="0110" else
a_7_22 when Seleccion="0111" else
a_8_22 when Seleccion="1000" else
a_9_22 when Seleccion="1001" else
a_10_22 when Seleccion="1010" else
"00000000";

--*****************************************************************************
U0 : Logaritmo port map(b0,Log_b0);
```

```
U1 : Logaritmo port map(b1,Log_b1);
U2 : Logaritmo port map(b2,Log_b2);
--*************************************************************************
U3 : SemiMultiplicador port map(Log_b0,a00,Inhividor_1,M0);
U4 : SemiMultiplicador port map(Log_b1,a01,Inhividor_2,M1);
U5 : SemiMultiplicador port map(Log_b2,a02,Inhividor_3,M2);
C0 <= M0 xor M1 xor M2;

U6 : SemiMultiplicador port map(Log_b0,a10,Inhividor_4,M3);
U7 : SemiMultiplicador port map(Log_b1,a11,Inhividor_5,M4);
U8 : SemiMultiplicador port map(Log_b2,a12,Inhividor_6,M5);
C1 <= M3 xor M4 xor M5;

U10 : SemiMultiplicador port map(Log_b0,a20,Inhividor_7,M6);
U11 : SemiMultiplicador port map(Log_b1,a21,Inhividor_8,M7);
U12 : SemiMultiplicador port map(Log_b2,a22,Inhividor_9,M8);
C2 <= M6 xor M7 xor M8;

end Behavioral;
```

Código fuente del algoritmo dispersor de información utilizado para comparar el desempeño de este programa con el descrito en VHDL. Fue elaborado en c++.

```
#include <iostream.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <fstream.h>
#include <time.h>
#include <dos.h>


char  Cadena[15];
FILE *in, *out0, *out1, *out2, *out3, *out4, *out5, *out6, *out7;



   unsigned int                  orden;
   unsigned int                exp[256];
   unsigned int                log[256];
   unsigned int a[5][3]={'1', '3', '2' ,
                         '1', '1', '1' ,
                         '2', '3', '1' ,
                         '2', '2', '3' ,
                         '2', '3', '3'};

   unsigned int                   b[3];
   const int PRIMITIVO            =369;

//*************************************************************************
```

```
//*****************************************************************************
int Log(int Pol)
{
int X0=1,X1=2,X2=4,X3=8,X4=16,X5=32,X6=64,X7=128;
   int Z0=0,Z1=0,Z2=0,Z3=0,Z4=0,Z5=0,Z6=0,Z7=0;

   int Y=0;

   X0=(Pol)&(X0);

   X1=(Pol)&(X1);
   if(X1==2)
   X1=1;
   else
   X1=0;


   X2=(Pol)&(X2);
   if(X2==4)
   X2=1;
   else
   X2=0;


   X3=(Pol)&(X3);
   if(X3==8)
   X3=1;
   else
   X3=0;



   X4=(Pol)&(X4);
   if(X4==16)
   X4=1;
   else
   X4=0;



   X5=(Pol)&(X5);
   if(X5==32)
   X5=1;
   else
   X5=0;



   X6=(Pol)&(X6);
   if(X6==64)
   X6=1;
```

```
   else
   X6=0;



   X7=(Pol)&(X7);
   if(X7==128)
   X7=1;
   else
   X7=0;
```

```
Z7 = ((!X6) & (!X5) & (!X4) & (!X3) & (X2) & (!X1) & (X0 ) ) || ( ( !X6) & (!X5)
& (!X3) & (X2) & (X1) & (!X0 ) ) || ( ( !X7) & (!X6) & (!X5) & (!X4) & (X3) & (!X2)
& (X1 ) ) || ( ( !X7) & (!X6) & (!X4) & (X3) & (X2) & (!X1) & (!X0 ) ) || ( ( !X7)
& (!X6) & (!X5) & (X4) & (!X3) & (X2) & (!X0 ) ) || ( ( !X7) & (X5) & (!X3) & (!X2)
& (X1) & (!X0 ) ) || ( ( !X7) & (!X6) & (X5) & (!X4) & (X3) & (!X1 ) ) || ( ( X6)
& (!X4) & (!X3) & (X2) & (!X1) & (!X0 ) ) || ( ( !X7) & (X6) & (!X5) & (!X3) & (X1)
& (X0 ) ) || ( ( !X7) & (X6) & (!X5) & (X4) & (!X2) & (!X0 ) ) || ( ( !X7) & (X5)
& (X3) & (X2) & (!X1) & (X0 ) ) || ( ( X7) & (!X5) & (!X4) & (X3) & (!X2) & (!X1)
& (!X0 ) ) || ( ( X7) & (!X6) & (!X4) & (X2) & (X1) & (!X0 ) ) || ( ( X7) & (!X6)
& (!X5) & (X4) & (!X2) & (X0 ) ) || ( ( X7) & (X6) & (!X5) & (!X4) & (!X3) & (!X2)
& (!X0 ) ) || ( ( X6) & (X4) & (X3) & (!X2) & (X1) & (!X0 ) ) || ( ( X7) & (X6)
& (X5) & (X4) & (!X3) & (!X1 ) ) || ( ( !X7) & (!X6) & (!X5) & (X3) & (X2) & (X1)
& (X0 ) ) || ( ( !X7) & (!X6) & (X4) & (!X2) & (!X1) & (X0 ) ) || ( ( !X7) & (!X6)
& (!X5) & (X4) & (X3) & (!X2) & (!X1 ) ) || ( ( !X7) & (!X4) & (X3) & (!X2) & (X1)
& (X0 ) ) || ( ( !X7) & (X6) & (X5) & (X4) & (X1) & (X0 ) ) || ( ( X7) & (!X6) &
(!X5) & (!X4) & (!X3) & (X0 ) ) || ( ( X7) & (!X6) & (X3) & (X2) & (!X1) & (X0 )
) || ( ( X7) & (!X5) & (!X4) & (!X3) & (X2) & (!X1) & (X0 ) ) || ( ( X7) & (X6)
& (X5) & (X2) & (X1) & (!X0 ) ) || ( ( !X7) & (!X5) & (!X4) & (!X2) & (X1) & (X0
) ) || ( ( !X7) & (X6) & (X4) & (!X2) & (X1 ) ) || ( ( !X7) & (X6) & (X4) & (X3)
& (!X1) & (!X0 ) ) || ( ( !X6) & (X5) & (!X4) & (X3) & (X2) & (!X0 ) ) || ( ( X7)
& (!X6) & (X5) & (X4) & (X2) & (X1) & (X0 ) ) || ( ( X7) & (X6) & (!X5) & (!X4)
& (X3) & (!X2) & (X0 ) ) || ( ( X7) & (X6) & (X5) & (!X4) & (!X3) & (X1) & (X0 )
) || ( ( X7) & (X6) & (X5) & (X4) & (X3) & (!X2 ) ) || ( ( !X7) & (!X6) & (X5) &
(X3) & (X2) & (!X0 ) ) || ( ( X6) & (X5) & (!X4) & (X2) & (X1) & (!X0 ) ) || ( (
X6) & (X5) & (!X3) & (X2) & (!X1) & (!X0 ) ) || ( ( X7) & (X6) & (!X4) & (X3) &
(!X2) & (!X1 ) ) || ( ( !X7) & (!X6) & (!X5) & (X4) & (X2) & (X1 ) ) || ( ( X5)
& (X4) & (X3) & (!X2) & (X1) & (!X0 ) ) || ( ( !X6) & (X5) & (X4) & (!X3) & (X1)
& (X0 ) ) || ( ( !X5) & (X4) & (!X3) & (X2) & (X1) & (!X0 ) ) || ( ( X7) & (X5)
& (!X4) & (!X3) & (!X2) & (X0 ) ) || ( ( X7) & (X5) & (!X3) & (X2) & (!X0 ) ) ||
( ( !X7) & (X6) & (X5) & (!X4) & (!X3) & (!X2) & (!X1 ) ) || ( ( X7) & (!X6) & (X5)
& (!X3) & (!X1) & (!X0 ) ) || ( ( !X7) & (!X6) & (X5) & (X4) & (!X3) & (!X2 ) )
|| ( ( X7) & (X5) & (X4) & (X3) & (!X2) & (!X1 ) ) || ( ( X6) & (!X5) & (X4) & (X3)
& (X2) & (!X1 ) ) || ( ( !X6) & (X4) & (X3) & (!X1) & (X0 ) ) || ( ( X7) & (X6)
& (!X5) & (!X4) & (!X1) & (!X0));
   //Input Cost = 367 Gate Cost = 52
   Z6 = ((!X6) & (!X5) & (X4) & (X3) & (!X2) & (!X1 ) ) || ( ( !X6) & (X5) & (!X4)
& (X3) & (!X2 ) ) || ( ( !X7) & (!X6) & (X5) & (X4) & (X3) & (X1) & (X0 ) ) || (
( !X7) & (X6) & (!X5) & (!X4) & (!X2) & (X0 ) ) || ( ( !X7) & (X6) & (!X5) & (X4)
& (!X3) & (!X0 ) ) || ( ( X6) & (X5) & (!X4) & (!X3) & (!X1) & (!X0 ) ) || ( ( !X7)
& (X6) & (X5) & (!X4) & (X2) & (X1) & (X0 ) ) || ( ( !X7) & (X6) & (X5) & (X4) &
```

```
(X1) & (!X0 ) ) || ( ( X7) & (!X6) & (!X5) & (!X4) & (X3) & (X1) & (X0 ) ) || (
( !X6) & (X5) & (!X4) & (!X3) & (X2) & (X0 ) ) || ( ( X7) & (X6) & (!X5) & (!X4)
& (!X3) & (X2) & (X1 ) ) || ( ( X7) & (X6) & (!X5) & (!X4) & (!X2) & (!X1) & (!X0
) ) || ( ( X7) & (X6) & (!X4) & (X3) & (X2) & (!X1) & (X0 ) ) || ( ( !X5) & (X4)
& (!X3) & (X2) & (!X1) & (X0 ) ) || ( ( X7) & (X6) & (!X5) & (X4) & (X3) & (!X2)
& (X1) & (X0 ) ) || ( ( X7) & (X6) & (!X5) & (X3) & (X2) & (X1) & (!X0 ) ) || (
( X7) & (X5) & (!X4) & (X3) & (!X2) & (X1 ) ) || ( ( X7) & (X6) & (X5) & (X4) &
(X3) & (X2) & (X0 ) ) || ( ( !X6) & (!X5) & (!X3) & (X2) & (!X1) & (X0 ) ) || (
( !X7) & (!X5) & (!X4) & (X3) & (!X2) & (X1) & (!X0 ) ) || ( ( !X6) & (!X5) & (X4)
& (!X3) & (X2) & (!X1 ) ) || ( ( !X7) & (X6) & (!X5) & (!X2) & (X1) & (!X0 ) ) ||
( ( !X7) & (X6) & (!X4) & (!X3) & (!X2) & (X1 ) ) || ( ( !X7) & (X6) & (X4) & (!X3)
& (X2) & (X0 ) ) || ( ( !X7) & (X5) & (X4) & (X3) & (!X2) & (X0 ) ) || ( ( X7) &
(!X6) & (X3) & (!X2) & (!X1) & (X0 ) ) || ( ( X7) & (!X6) & (!X5) & (X4) & (X2)
& (!X1 ) ) || ( ( X7) & (!X6) & (X5) & (!X4) & (!X1) & (!X0 ) ) || ( ( X7) & (X4)
& (!X3) & (!X2) & (X1) & (!X0 ) ) || ( ( X7) & (X6) & (X5) & (!X3) & (!X2) & (!X1)
& (X0 ) ) || ( ( X7) & (X5) & (!X4) & (X3) & (X2) & (!X0 ) ) || ( ( X7) & (X5) &
(X4) & (!X2) & (X1) & (!X0 ) ) || ( ( X7) & (X6) & (X5) & (X2) & (!X1) & (!X0 )
) || ( ( !X7) & (X6) & (X3) & (!X2) & (!X1) & (X0 ) ) || ( ( X7) & (!X6) & (!X5)
& (!X3) & (X1) & (!X0 ) ) || ( ( !X6) & (X5) & (!X4) & (X2) & (!X1) & (X0 ) ) ||
( ( X7) & (!X6) & (!X3) & (X2) & (!X1) & (!X0 ) ) || ( ( X7) & (!X6) & (X4) & (X3)
& (!X2) & (!X1 ) ) || ( ( X7) & (!X5) & (!X4) & (!X3) & (X2) & (!X0 ) ) || ( ( X6)
& (X5) & (X4) & (!X3) & (X1) & (!X0 ) ) || ( ( !X6) & (!X5) & (!X3) & (!X2) & (X1)
& (X0 ) ) || ( ( !X7) & (!X5) & (X4) & (!X3) & (X2) & (X0 ) ) || ( ( !X7) & (!X6)
& (X5) & (!X4) & (!X2) & (X0 ) ) || ( ( X7) & (X5) & (X4) & (!X2) & (!X1) & (X0
) ) || ( ( !X7) & (X6) & (X4) & (X3) & (!X1) & (X0 ) ) || ( ( X7) & (!X6) & (X4)
& (!X3) & (!X2) & (X1 ) ) || ( ( !X7) & (X6) & (!X5) & (!X4) & (X1) & (!X0 ) ) ||
( ( !X7) & (X6) & (!X5) & (X3) & (X2) & (!X1) & (!X0 ) ) || ( ( !X6) & (!X5) & (!X4)
& (X3) & (X2) & (!X1) & (!X0 ) ) || ( ( !X7) & (!X6) & (X5) & (X4) & (!X3) & (!X2)
& (!X0 ) ) || ( ( !X6) & (X5) & (X4) & (!X2) & (!X1) & (X0 ) ) || ( ( !X7) & (!X6)
& (X5) & (!X4) & (X2) & (X1) & (!X0 ) ) || ( ( !X5) & (!X4) & (!X3) & (X2) & (X1)
& (!X0));
```

```
   //Input Cost = 386 Gate Cost = 54
```

```
   Z5 = ((!X7) & (!X6) & (!X5) & (!X4) & (!X3) & (X1) & (X0 ) ) || ( ( !X7) & (!X6)
& (!X5) & (!X4) & (X2) & (X1 ) ) || ( ( !X7) & (!X6) & (!X5) & (X3) & (X2) & (!X0
) ) || ( ( !X7) & (X6) & (X5) & (X4) & (X3) & (X2) & (X1) & (X0 ) ) || ( ( X7) &
(!X6) & (!X3) & (X2) & (!X1) & (!X0 ) ) || ( ( !X6) & (!X5) & (!X4) & (!X3) & (X2)
& (X1) & (X0 ) ) || ( ( X7) & (!X6) & (!X5) & (!X4) & (X3) & (!X1) & (X0 ) ) ||
( ( X7) & (!X6) & (X4) & (X3) & (X1) & (X0 ) ) || ( ( X7) & (X6) & (!X5) & (!X4)
& (!X2) & (!X0 ) ) || ( ( X7) & (X6) & (!X5) & (!X4) & (!X2) & (X1 ) ) || ( ( X6)
& (!X4) & (!X3) & (X2) & (!X1) & (X0 ) ) || ( ( X7) & (X6) & (X5) & (!X3) & (X2)
& (X0 ) ) || ( ( X7) & (X5) & (X4) & (X3) & (!X2) & (X1) & (X0 ) ) || ( ( X7) &
(X6) & (X5) & (X4) & (X3) & (X2) & (!X0 ) ) || ( ( !X7) & (X6) & (!X5) & (!X2) &
(X1) & (!X0 ) ) || ( ( X7) & (!X6) & (X5) & (!X4) & (X1) & (!X0 ) ) || ( ( X7) &
(!X6) & (!X4) & (X3) & (!X2) & (X1 ) ) || ( ( X7) & (!X6) & (X4) & (!X3) & (!X2)
& (X1 ) ) || ( ( !X6) & (X5) & (X4) & (X3) & (!X2) & (!X1 ) ) || ( ( X5) & (!X4)
& (!X3) & (X2) & (!X1) & (!X0 ) ) || ( ( !X7) & (X4) & (!X3) & (!X2) & (X1) & (!X0
) ) || ( ( !X7) & (!X6) & (X4) & (X3) & (!X2) & (!X1 ) ) || ( ( !X6) & (X5) & (!X4)
& (!X2) & (X1) & (!X0 ) ) || ( ( !X7) & (X5) & (X4) & (!X2) & (!X1) & (!X0 ) ) ||
( ( !X7) & (X6) & (!X5) & (!X3) & (X2) & (!X1 ) ) || ( ( !X7) & (X6) & (!X5) & (X4)
& (!X3) & (X0 ) ) || ( ( !X7) & (X6) & (X5) & (!X4) & (!X3) & (!X1 ) ) || ( ( X7)
& (!X6) & (!X4) & (X3) & (!X2) & (!X0 ) ) || ( ( X7) & (!X6) & (!X5) & (X4) & (!X3)
```

```
& (!X1 ) ) || ( ( !X6) & (X4) & (X3) & (X2) & (!X1) & (!X0 ) ) || ( ( X7) & (!X5)
& (X3) & (!X2) & (X1) & (!X0 ) ) || ( ( X6) & (X5) & (X4) & (!X3) & (!X2) & (!X0
) ) || ( ( !X7) & (!X6) & (X5) & (!X4) & (!X1) & (X0 ) ) || ( ( !X7) & (!X5) & (X4)
& (X3) & (X2) & (!X0 ) ) || ( ( !X7) & (X6) & (X5) & (!X4) & (X3) & (!X2) & (X1
) ) || ( ( !X7) & (X4) & (X3) & (!X2) & (!X1) & (X0 ) ) || ( ( X7) & (!X6) & (!X5)
& (X3) & (X1) & (!X0 ) ) || ( ( X7) & (X6) & (!X5) & (X4) & (!X3) & (X2) & (!X0
) ) || ( ( X7) & (X6) & (X5) & (!X3) & (!X1) & (X0 ) ) || ( ( !X7) & (!X5) & (X3)
& (!X2) & (!X1) & (X0 ) ) || ( ( !X7) & (!X6) & (!X4) & (X3) & (X2) & (X1 ) ) ||
( ( !X7) & (!X6) & (X5) & (X2) & (!X1) & (X0 ) ) || ( ( !X7) & (X6) & (X5) & (!X4)
& (!X3) & (X0 ) ) || ( ( X7) & (X6) & (!X5) & (!X4) & (X1) & (!X0 ) ) || ( ( X7)
& (X6) & (!X5) & (X4) & (X3) & (!X1) & (X0 ) ) || ( ( !X7) & (!X5) & (X4) & (!X3)
& (X2) & (X0 ) ) || ( ( !X7) & (!X4) & (!X3) & (X2) & (X1) & (X0 ) ) || ( ( X6)
& (!X5) & (!X4) & (X3) & (!X2) & (!X0 ) ) || ( ( !X7) & (!X5) & (!X4) & (X3) & (X2)
& (X1 ) ) || ( ( X7) & (X6) & (X5) & (X2) & (!X1) & (X0 ) ) || ( ( !X7) & (X6) &
(!X4) & (X2) & (!X1) & (X0));
   //Input Cost = 367 Gate Cost = 52
   Z4 = ((!X7) & (!X6) & (!X5) & (X2) & (X1) & (X0 ) ) || ( ( !X6) & (X5) & (X4)
& (!X3) & (!X2) & (X1) & (!X0 ) ) || ( ( !X7) & (X6) & (!X4) & (!X3) & (!X2) & (X1)
& (!X0 ) ) || ( ( !X7) & (X6) & (X5) & (!X4) & (!X3) & (X2) & (!X1 ) ) || ( ( !X7)
& (X6) & (X4) & (!X3) & (X1) & (X0 ) ) || ( ( !X7) & (X5) & (X4) & (X3) & (X2) &
(!X0 ) ) || ( ( X7) & (!X6) & (!X5) & (!X4) & (!X3) & (!X2) & (X0 ) ) || ( ( X7)
& (!X6) & (X5) & (!X2) & (!X1) & (!X0 ) ) || ( ( X7) & (X5) & (!X4) & (X2) & (X1
) ) || ( ( X7) & (!X6) & (X5) & (X4) & (X1) & (X0 ) ) || ( ( X7) & (!X5) & (!X4)
& (!X3) & (X2) & (!X1) & (!X0 ) ) || ( ( X7) & (X6) & (!X5) & (!X4) & (X3) & (!X2)
& (!X0 ) ) || ( ( X7) & (X6) & (X5) & (!X4) & (!X3) & (X0 ) ) || ( ( X6) & (X5)
& (X4) & (X3) & (!X2) & (!X1 ) ) || ( ( X7) & (X3) & (X2) & (X1) & (X0 ) ) || (
( !X7) & (X6) & (!X5) & (!X4) & (X2) & (!X1) & (X0 ) ) || ( ( !X7) & (X6) & (!X5)
& (!X3) & (X2) & (X1) & (!X0 ) ) || ( ( X7) & (!X6) & (!X4) & (X3) & (X2) & (!X1
) ) || ( ( !X6) & (X4) & (!X3) & (X2) & (!X1) & (!X0 ) ) || ( ( X7) & (X6) & (!X5)
& (X2) & (X1) & (X0 ) ) || ( ( X7) & (X6) & (X4) & (!X3) & (!X2) & (X1 ) ) || (
( X7) & (!X5) & (X4) & (!X3) & (!X1) & (X0 ) ) || ( ( X7) & (X6) & (!X5) & (X4)
& (X2) & (X1 ) ) || ( ( X7) & (X6) & (!X5) & (X3) & (X1) & (X0 ) ) || ( ( X6) &
(X4) & (!X3) & (!X2) & (!X1) & (!X0 ) ) || ( ( X7) & (X6) & (!X5) & (X3) & (X2)
& (X0 ) ) || ( ( X6) & (X5) & (!X4) & (X3) & (X1) & (X0 ) ) || ( ( !X7) & (!X6)
& (!X4) & (X3) & (!X2) & (X1 ) ) || ( ( !X7) & (X6) & (X5) & (X3) & (!X2) & (!X1
) ) || ( ( !X7) & (!X6) & (X3) & (X2) & (X1) & (!X0 ) ) || ( ( !X7) & (!X6) & (X5)
& (X3) & (!X1) & (!X0 ) ) || ( ( !X6) & (X4) & (X3) & (X2) & (X1 ) ) || ( ( !X6)
& (!X5) & (X3) & (!X2) & (X1) & (!X0 ) ) || ( ( !X6) & (X5) & (X3) & (X2) & (!X1)
& (X0 ) ) || ( ( !X5) & (!X4) & (X3) & (X2) & (X0 ) ) || ( ( !X7) & (!X6) & (X5)
& (!X4) & (!X3) & (!X1) & (X0 ) ) || ( ( !X7) & (!X5) & (X3) & (!X2) & (X1) & (!X0
) ) || ( ( !X6) & (X5) & (!X4) & (X3) & (!X0 ) ) || ( ( !X7) & (!X6) & (X4) & (!X3)
& (!X2) & (!X1) & (X0 ) ) || ( ( !X5) & (X4) & (X2) & (X1) & (X0 ) ) || ( ( !X7)
& (!X6) & (!X5) & (X4) & (!X3) & (X2 ) ) || ( ( !X7) & (!X5) & (X4) & (X3) & (!X2)
& (!X1) & (X0 ) ) || ( ( !X7) & (!X5) & (X4) & (X2) & (!X1) & (!X0 ) ) || ( ( !X7)
& (X6) & (!X5) & (X4) & (!X1) & (!X0));
   //Input Cost = 313 Gate Cost = 45
   Z3 = ((!X7) & (!X6) & (!X5) & (!X4) & (!X3) & (X2) & (X0 ) ) || ( ( !X5) & (X4)
& (!X3) & (!X2) & (!X1) & (X0 ) ) || ( ( !X7) & (!X6) & (X3) & (!X2) & (X1) & (X0
) ) || ( ( !X7) & (!X6) & (X5) & (!X4) & (!X3) & (!X2) & (X0 ) ) || ( ( !X7) & (X6)
& (!X5) & (!X4) & (!X3) & (!X2) & (X1) & (!X0 ) ) || ( ( !X7) & (!X4) & (X3) & (X2)
& (X1) & (!X0 ) ) || ( ( X6) & (!X5) & (X4) & (!X3) & (!X2) & (X0 ) ) || ( ( !X7)
```

```
& (X6) & (!X5) & (X4) & (X2) & (!X1 ) ) || ( ( !X7) & (X6) & (X5) & (!X3) & (!X2)
& (!X1) & (!X0 ) ) || ( ( X6) & (X5) & (X2) & (X1) & (!X0 ) ) || ( ( X6) & (X5)
& (!X4) & (X3) & (X1) & (!X0 ) ) || ( ( X6) & (X5) & (!X4) & (X3) & (X2 ) ) || (
( X7) & (!X6) & (!X5) & (!X4) & (!X3) & (X2) & (!X1) & (!X0 ) ) || ( ( X7) & (X6)
& (!X5) & (!X2) & (!X1 ) ) || ( ( X7) & (X6) & (!X4) & (!X1) & (X0 ) ) || ( ( !X7)
& (X6) & (!X5) & (!X4) & (X3) & (!X2) & (X0 ) ) || ( ( !X7) & (X4) & (!X3) & (X2)
& (X1) & (!X0 ) ) || ( ( X7) & (!X6) & (!X5) & (!X4) & (X3) & (X2) & (X1 ) ) ||
( ( X7) & (!X6) & (X4) & (X2) & (!X1) & (X0 ) ) || ( ( X7) & (!X5) & (!X3) & (!X2)
& (X1) & (X0 ) ) || ( ( X7) & (X6) & (!X5) & (X3) & (!X2) & (!X0 ) ) || ( ( X7)
& (X6) & (!X5) & (X3) & (!X1) & (!X0 ) ) || ( ( X6) & (!X5) & (X4) & (X2) & (!X1)
& (!X0 ) ) || ( ( !X7) & (X4) & (X3) & (!X2) & (!X1) & (!X0 ) ) || ( ( X7) & (!X6)
& (!X5) & (X4) & (!X3) & (X1) & (!X0 ) ) || ( ( !X5) & (X4) & (X3) & (X2) & (!X1)
& (!X0 ) ) || ( ( X7) & (X5) & (!X4) & (!X3) & (X1) & (!X0 ) ) || ( ( X7) & (!X6)
& (X5) & (!X4) & (X2) & (X0 ) ) || ( ( X7) & (!X6) & (X5) & (!X2) & (X1) & (X0 )
) || ( ( !X6) & (!X4) & (!X3) & (X2) & (X1) & (X0 ) ) || ( ( X7) & (!X6) & (X5)
& (X4) & (X3) & (!X2) & (!X0 ) ) || ( ( !X7) & (!X6) & (!X5) & (!X3) & (X2) & (X1
) ) || ( ( X7) & (!X6) & (X5) & (!X4) & (X3) & (!X1 ) ) || ( ( X5) & (X4) & (!X3)
& (!X2) & (!X1) & (!X0 ) ) || ( ( !X7) & (!X6) & (!X5) & (X3) & (X2) & (!X0 ) )
|| ( ( !X7) & (!X6) & (X3) & (X2) & (!X1) & (!X0 ) ) || ( ( X7) & (X6) & (X5) &
(!X3) & (X1) & (!X0 ) ) || ( ( !X7) & (!X6) & (X5) & (X4) & (X2) & (X0 ) ) || (
( X6) & (X5) & (X4) & (X3) & (!X2) & (X0 ) ) || ( ( !X7) & (X5) & (X4) & (X2) &
(X1) & (X0 ) ) || ( ( X7) & (X5) & (!X4) & (X3) & (X0 ) ) || ( ( !X7) & (X6) & (X4)
& (!X3) & (!X1) & (X0 ) ) || ( ( X7) & (X6) & (!X2) & (!X1) & (X0 ) ) || ( ( X7)
& (X6) & (!X5) & (X4) & (X3) & (X2) & (X1 ) ) || ( ( X7) & (!X4) & (X3) & (X2) &
(X1) & (X0 ) ) || ( ( X7) & (X6) & (X4) & (!X3) & (X2) & (X1) & (X0));
   //Input Cost = 329 Gate Cost = 47
   Z2 = ((!X7) & (!X6) & (!X4) & (!X3) & (!X2) & (X1) & (X0 ) ) || ( ( !X6) & (!X5)
& (X4) & (!X2) & (X1) & (!X0 ) ) || ( ( !X5) & (X4) & (X3) & (X2) & (!X1) & (!X0
) ) || ( ( !X6) & (X5) & (X4) & (X3) & (!X2) & (!X1) & (!X0 ) ) || ( ( !X7) & (X5)
& (X4) & (X3) & (X2) & (X0 ) ) || ( ( !X7) & (X6) & (X4) & (!X3) & (!X2) & (X1)
& (X0 ) ) || ( ( !X7) & (X6) & (X4) & (!X3) & (X2) & (X1) & (!X0 ) ) || ( ( !X7)
& (X6) & (X5) & (!X3) & (!X2) & (!X0 ) ) || ( ( !X7) & (X6) & (X4) & (X3) & (!X1)
& (X0 ) ) || ( ( X7) & (!X6) & (!X5) & (!X3) & (X1) & (!X0 ) ) || ( ( X7) & (!X6)
& (!X4) & (!X3) & (X2) & (X1 ) ) || ( ( X7) & (X6) & (!X5) & (!X4) & (X3) & (X2)
& (X1 ) ) || ( ( X7) & (X6) & (X3) & (X1) & (X0 ) ) || ( ( X7) & (X6) & (X5) & (X4)
& (!X3) & (!X1) & (X0 ) ) || ( ( !X6) & (!X5) & (X4) & (!X3) & (!X2) & (!X1 ) )
|| ( ( !X7) & (!X6) & (!X5) & (X4) & (!X2) & (!X1) & (X0 ) ) || ( ( !X7) & (!X6)
& (X5) & (!X4) & (X3) & (!X2) & (X1 ) ) || ( ( !X7) & (!X6) & (X5) & (X4) & (!X3)
& (X1 ) ) || ( ( X7) & (!X6) & (!X5) & (X3) & (!X2) & (X1 ) ) || ( ( X7) & (!X6)
& (X5) & (X3) & (!X2) & (!X1) & (X0 ) ) || ( ( X7) & (X5) & (X3) & (X2) & (X1) &
(X0 ) ) || ( ( X7) & (X6) & (X5) & (X4) & (X2) & (X0 ) ) || ( ( !X7) & (!X6) & (X4)
& (!X3) & (!X2) & (!X1) & (X0 ) ) || ( ( !X7) & (X6) & (!X4) & (!X3) & (X2) & (X0
) ) || ( ( X7) & (!X6) & (!X5) & (X4) & (!X3) & (!X2 ) ) || ( ( X7) & (!X6) & (!X4)
& (!X3) & (!X2) & (!X1 ) ) || ( ( X7) & (!X6) & (X5) & (!X4) & (X3) & (X2) & (!X1
) ) || ( ( X7) & (X6) & (!X5) & (!X4) & (!X3) & (!X2 ) ) || ( ( X6) & (!X4) & (!X3)
& (X2) & (X1) & (X0 ) ) || ( ( X7) & (X5) & (X4) & (!X2) & (X1) & (!X0 ) ) || (
( !X7) & (X5) & (!X4) & (!X3) & (!X1) & (!X0 ) ) || ( ( X6) & (X5) & (X4) & (X3)
& (X2) & (X1 ) ) || ( ( X7) & (X4) & (X3) & (!X2) & (!X1) & (!X0 ) ) || ( ( X6)
& (X5) & (!X4) & (X3) & (!X1) & (!X0 ) ) || ( ( !X7) & (X5) & (!X4) & (X3) & (X2)
& (!X0 ) ) || ( ( !X7) & (X6) & (!X5) & (!X4) & (!X2) & (!X1 ) ) || ( ( !X7) & (!X6)
& (!X5) & (!X4) & (X3) & (X2) & (X1 ) ) || ( ( X6) & (!X5) & (X3) & (!X2) & (!X1)
```

```
& (X0 ) ) || ( ( !X7) & (!X6) & (!X5) & (!X4) & (X2) & (!X1) & (X0 ) ) || ( ( !X7)
& (X6) & (!X5) & (X3) & (!X1) & (!X0 ) ) || ( ( !X5) & (X4) & (!X3) & (X2) & (!X1)
& (X0 ) ) || ( ( !X7) & (X6) & (X5) & (X4) & (X2) & (!X0 ) ) || ( ( X7) & (!X6)
& (!X5) & (!X4) & (!X3) & (!X1 ) ) || ( ( X7) & (X6) & (!X5) & (!X3) & (X2) & (!X1)
& (!X0 ) ) || ( ( X6) & (X5) & (!X4) & (X2) & (!X1) & (!X0 ) ) || ( ( X7) & (!X6)
& (X5) & (X4) & (!X3) & (!X1) & (!X0 ) ) || ( ( !X7) & (X5) & (!X3) & (X2) & (!X1)
& (!X0));
   //Input Cost = 343 Gate Cost = 48
   Z1 = ((!X7) & (!X6) & (!X5) & (!X4) & (!X3) & (X2) & (!X1 ) ) || ( ( !X7) & (!X6)
& (!X5) & (X1) & (X0 ) ) || ( ( !X7) & (!X6) & (!X5) & (X4) & (!X3) & (X0 ) ) ||
( ( !X7) & (!X6) & (X3) & (!X2) & (!X1) & (!X0 ) ) || ( ( !X7) & (!X6) & (X5) &
(X3) & (X2) & (!X1) & (X0 ) ) || ( ( X6) & (!X5) & (!X4) & (X3) & (X2) & (!X1) &
(X0 ) ) || ( ( !X7) & (!X5) & (X4) & (!X3) & (X2) & (!X1) & (X0 ) ) || ( ( !X7)
& (X4) & (X3) & (X2) & (X1) & (X0 ) ) || ( ( !X7) & (X6) & (X5) & (!X4) & (!X3)
& (X2) & (!X1 ) ) || ( ( X5) & (X4) & (X3) & (X2) & (X1 ) ) || ( ( X7) & (!X6) &
(!X5) & (!X4) & (!X2) & (!X0 ) ) || ( ( X7) & (X4) & (X3) & (X2) & (X1) & (!X0 )
) || ( ( X7) & (X5) & (X3) & (X2) & (X1) & (X0 ) ) || ( ( X7) & (X6) & (!X5) & (!X4)
& (!X3) & (X2 ) ) || ( ( X7) & (X6) & (!X5) & (X4) & (X3) & (!X0 ) ) || ( ( X7)
& (X6) & (X5) & (X4) & (!X3) & (!X2) & (!X1 ) ) || ( ( !X7) & (!X6) & (!X5) & (!X4)
& (X3) & (!X2) & (!X1 ) ) || ( ( !X7) & (!X5) & (X4) & (!X3) & (X2) & (X1) & (!X0
) ) || ( ( !X6) & (X5) & (X4) & (X3) & (!X2 ) ) || ( ( !X7) & (X6) & (!X5) & (!X4)
& (!X2) & (X1 ) ) || ( ( !X7) & (X6) & (!X5) & (X3) & (!X2) & (X1) & (!X0 ) ) ||
( ( X7) & (!X5) & (X4) & (X3) & (X2) & (!X1) & (X0 ) ) || ( ( X7) & (X6) & (X3)
& (X2) & (X1) & (!X0 ) ) || ( ( X7) & (X6) & (X4) & (!X2) & (X1) & (!X0 ) ) || (
( !X7) & (!X6) & (X4) & (X2) & (!X1) & (X0 ) ) || ( ( !X7) & (X5) & (!X4) & (X3)
& (!X2) & (!X0 ) ) || ( ( !X7) & (!X6) & (X5) & (X4) & (!X2) & (!X0 ) ) || ( ( !X7)
& (X6) & (!X5) & (!X4) & (X3) & (X1) & (!X0 ) ) || ( ( !X7) & (X6) & (!X4) & (!X2)
& (X1) & (!X0 ) ) || ( ( !X7) & (X6) & (X5) & (!X3) & (X1) & (X0 ) ) || ( ( X7)
& (!X6) & (!X5) & (X4) & (!X3) & (!X1) & (!X0 ) ) || ( ( X7) & (!X6) & (X4) & (X3)
& (!X2) & (X0 ) ) || ( ( X7) & (X6) & (!X5) & (X4) & (!X3) & (X1 ) ) || ( ( !X7)
& (!X6) & (X5) & (X4) & (!X3) & (!X1) & (!X0 ) ) || ( ( !X7) & (X6) & (!X5) & (!X4)
& (!X2) & (!X0 ) ) || ( ( !X7) & (X5) & (X4) & (!X2) & (!X1) & (!X0 ) ) || ( ( X7)
& (!X5) & (X4) & (!X3) & (!X2) & (X1 ) ) || ( ( X5) & (X4) & (!X3) & (!X2) & (!X1)
& (!X0 ) ) || ( ( X7) & (!X6) & (!X3) & (X2) & (!X1) & (X0 ) ) || ( ( X6) & (X5)
& (X4) & (X2) & (X1) & (X0 ) ) || ( ( !X7) & (X6) & (!X5) & (!X3) & (!X2) & (!X1)
& (!X0 ) ) || ( ( X6) & (!X5) & (!X4) & (!X3) & (!X2) & (X0 ) ) || ( ( !X7) & (!X4)
& (!X3) & (X1) & (X0 ) ) || ( ( X7) & (!X6) & (!X4) & (!X3) & (!X1) & (X0 ) ) ||
( ( !X6) & (X5) & (!X4) & (!X3) & (X2) & (X1) & (!X0 ) ) || ( ( X7) & (X5) & (!X4)
& (X3) & (!X2) & (!X1) & (X0 ) ) || ( ( !X7) & (X6) & (X5) & (!X4) & (!X2) & (!X1)
& (X0 ) ) || ( ( X7) & (!X6) & (X5) & (!X4) & (X2) & (!X1) & (!X0 ) ) || ( ( X7)
& (X6) & (!X5) & (!X4) & (!X1) & (X0 ) ) || ( ( !X6) & (X5) & (!X4) & (!X3) & (!X2)
& (X0 ) ) || ( ( X7) & (!X6) & (X5) & (X4) & (X3) & (!X0 ) ) || ( ( !X6) & (X4)
& (!X2) & (X1) & (X0));
   //Input Cost = 377 Gate Cost = 53
   Z0 = ((!X7) & (!X5) & (!X4) & (X2) & (X1) & (X0 ) ) || ( ( !X7) & (!X6) & (X4)
& (!X2) & (!X1) & (X0 ) ) || ( ( !X7) & (!X6) & (X5) & (!X4) & (!X3) & (X2) & (!X0
) ) || ( ( X6) & (!X4) & (!X3) & (X2) & (!X1 ) ) || ( ( !X7) & (X6) & (!X5) & (X2)
& (!X1) & (X0 ) ) || ( ( X6) & (X5) & (!X4) & (!X3) & (!X1) & (X0 ) ) || ( ( X6)
& (X5) & (!X4) & (X3) & (!X0 ) ) || ( ( !X7) & (X5) & (X4) & (X3) & (!X1) & (X0
) ) || ( ( X7) & (!X6) & (!X5) & (!X3) & (X1) & (X0 ) ) || ( ( X7) & (!X6) & (!X5)
& (X4) & (!X2) & (!X1) & (!X0 ) ) || ( ( X7) & (X5) & (!X4) & (X3) & (!X1) & (!X0
```

```
) ) || ( ( !X6) & (X5) & (X4) & (!X3) & (!X2) & (!X1) & (X0 ) ) || ( ( !X6) & (X5)
& (X4) & (X3) & (X2) & (!X1) & (X0 ) ) || ( ( X7) & (X6) & (!X5) & (!X4) & (X2)
& (!X0 ) ) || ( ( X7) & (X6) & (!X5) & (!X2) & (!X1) & (X0 ) ) || ( ( X6) & (!X5)
& (X4) & (!X3) & (!X2) & (X1) & (!X0 ) ) || ( ( X7) & (X4) & (!X3) & (X2) & (X1)
& (X0 ) ) || ( ( X7) & (X6) & (!X5) & (X4) & (X3) & (!X2) & (X0 ) ) || ( ( X7) &
(X6) & (X5) & (X4) & (X2) & (X1 ) ) || ( ( !X7) & (!X6) & (X4) & (X3) & (!X2) &
(X1 ) ) || ( ( !X7) & (!X6) & (X5) & (!X3) & (!X2) & (!X1) & (!X0 ) ) || ( ( !X7)
& (!X4) & (!X3) & (X2) & (!X1) & (X0 ) ) || ( ( !X7) & (X6) & (!X4) & (X3) & (X2)
& (X0 ) ) || ( ( X6) & (!X5) & (X3) & (X2) & (!X1) & (X0 ) ) || ( ( X7) & (X5) &
(!X4) & (!X3) & (!X2) & (X1) & (!X0 ) ) || ( ( X7) & (X6) & (X5) & (X4) & (!X2)
& (!X1) & (!X0 ) ) || ( ( !X7) & (X5) & (!X4) & (X3) & (!X2) & (X1 ) ) || ( ( !X7)
& (X6) & (X4) & (!X3) & (!X2) & (!X1) & (!X0 ) ) || ( ( X7) & (!X6) & (!X5) & (!X3)
& (X2) & (X0 ) ) || ( ( X7) & (!X6) & (X5) & (X4) & (!X3) & (X1) & (!X0 ) ) || (
( X6) & (!X5) & (!X4) & (X3) & (X2) & (X1 ) ) || ( ( X7) & (X6) & (X5) & (!X3) &
(X2) & (!X1 ) ) || ( ( !X7) & (X6) & (!X5) & (!X4) & (X3) & (X0 ) ) || ( ( !X7)
& (X6) & (!X5) & (X4) & (X3) & (!X0 ) ) || ( ( !X6) & (!X5) & (X4) & (!X3) & (X2)
& (!X1) & (!X0 ) ) || ( ( X7) & (X5) & (!X3) & (X2) & (X1) & (X0 ) ) || ( ( !X7)
& (!X6) & (!X5) & (X4) & (!X3) & (X2) & (X1 ) ) || ( ( X5) & (X4) & (X3) & (X2)
& (X1) & (!X0 ) ) || ( ( !X7) & (!X6) & (X4) & (X3) & (X2) & (!X1) & (!X0 ) ) ||
( ( X7) & (!X6) & (X5) & (!X4) & (X1) & (X0 ) ) || ( ( !X7) & (!X6) & (X5) & (!X2)
& (X1) & (X0 ) ) || ( ( X7) & (X5) & (X3) & (X2) & (X1) & (!X0 ) ) || ( ( !X7) &
(X6) & (X4) & (!X3) & (!X2) & (X1) & (X0 ) ) || ( ( !X7) & (!X5) & (!X4) & (!X3)
& (!X2) & (X1 ) ) || ( ( !X5) & (!X4) & (X3) & (!X2) & (!X1) & (X0 ) ) || ( ( !X7)
& (!X6) & (!X5) & (!X4) & (X3) & (!X1) & (!X0 ) ) || ( ( X7) & (!X6) & (!X4) & (!X2)
& (X1) & (X0 ) ) || ( ( X7) & (!X4) & (X3) & (X2) & (!X1) & (!X0 ) ) || ( ( X7)
& (!X5) & (!X4) & (!X3) & (!X2) & (!X1) & (!X0 ) ) || ( ( !X6) & (!X5) & (!X4) &
(!X3) & (!X2) & (X1));
   //Input Cost = 365 Gate Cost = 51
   // ***Total Input Cost = 2847*** // ***Total Gate Cost = 402***

   Z0=Z0*1;

   Z1=Z1*2;

   Z2=Z2*4;

   Z3=Z3*8;

   Z4=Z4*16;

   Z5=Z5*32;

   Z6=Z6*64;

   Z7=Z7*128;

   Y=Z0+Z1+Z2+Z3+Z4+Z5+Z6+Z7;

   return Y;
}
//*********************************************************************************
```

```
//*****************************************************************************
int AntLog(int Pot)
{
int X0=1,X1=2,X2=4,X3=8,X4=16,X5=32,X6=64,X7=128;
   int Z0=0,Z1=0,Z2=0,Z3=0,Z4=0,Z5=0,Z6=0,Z7=0;

   int Y=0;

   X0=(Pot)&(X0);

   X1=(Pot)&(X1);
   if(X1==2)
   X1=1;
   else
   X1=0;


   X2=(Pot)&(X2);
   if(X2==4)
   X2=1;
   else
   X2=0;


   X3=(Pot)&(X3);
   if(X3==8)
   X3=1;
   else
   X3=0;



   X4=(Pot)&(X4);
   if(X4==16)
   X4=1;
   else
   X4=0;



   X5=(Pot)&(X5);
   if(X5==32)
   X5=1;
   else
   X5=0;



   X6=(Pot)&(X6);
   if(X6==64)
   X6=1;
```

```
   else
   X6=0;



   X7=(Pot)&(X7);
   if(X7==128)
   X7=1;
   else
   X7=0;
```

```
Z7 =(( !X6) & (X5) & (!X3) & (!X2) & (X1) & (!X0 ) ) || ( ( !X6) & (X5) & (X4) &
(!X3) & (!X0 ) ) || ( ( !X7) & (X6) & (!X4) & (!X3) & (!X2) & (!X1 ) ) || ( ( !X7)
& (X6) & (!X5) & (X4) & (!X3) & (X2 ) ) || ( ( !X7) & (X6) & (!X3) & (X2) & (!X1)
& (X0 ) ) || ( ( !X7) & (X6) & (X5) & (X4) & (X3) & (X2) & (X1 ) ) || ( ( X7) &
(X6) & (X5) & (X4) & (!X2) & (!X1) & (X0 ) ) || ( ( !X7) & (!X6) & (X4) & (X3) &
(!X2) & (!X0 ) ) || ( ( !X7) & (!X5) & (X4) & (!X3) & (!X2) & (X0 ) ) || ( ( !X7)
& (X6) & (X5) & (!X4) & (!X1) & (!X0 ) ) || ( ( !X7) & (X5) & (X4) & (!X3) & (!X2)
& (!X1) & (!X0 ) ) || ( ( !X7) & (X5) & (X3) & (!X2) & (!X1) & (X0 ) ) || ( ( !X6)
& (!X5) & (X4) & (!X2) & (!X1 ) ) || ( ( X7) & (!X6) & (X5) & (X4) & (!X2) & (X1)
& (X0 ) ) || ( ( X7) & (X5) & (!X4) & (X2) & (!X1) & (X0 ) ) || ( ( X7) & (X6) &
(X4) & (!X3) & (X2) & (!X1) & (!X0 ) ) || ( ( !X5) & (!X4) & (X3) & (!X2) & (!X1)
& (X0 ) ) || ( ( !X7) & (X6) & (X5) & (!X3) & (!X2) & (X1) & (X0 ) ) || ( ( X7)
& (X6) & (!X5) & (X2) & (!X1) & (!X0 ) ) || ( ( X6) & (!X5) & (!X3) & (X2) & (X1)
& (X0 ) ) || ( ( !X7) & (X6) & (!X5) & (X2) & (!X1) & (X0 ) ) || ( ( !X7) & (!X4)
& (!X3) & (X2) & (X1) & (X0 ) ) || ( ( X7) & (!X5) & (!X4) & (!X2) & (!X1) & (X0
) ) || ( ( X7) & (X6) & (!X5) & (X4) & (X3) & (X1) & (X0 ) ) || ( ( !X7) & (!X5)
& (X4) & (X3) & (!X2) & (X1 ) ) || ( ( X7) & (!X6) & (!X3) & (X2) & (!X1) & (X0
) ) || ( ( X7) & (X6) & (X4) & (X3) & (!X2) & (!X1 ) ) || ( ( !X5) & (!X4) & (X3)
& (X1) & (!X0 ) ) || ( ( X6) & (!X5) & (X4) & (X3) & (!X1) & (!X0 ) ) || ( ( X7)
& (X6) & (X5) & (X4) & (X3) & (X1) & (!X0 ) ) || ( ( !X7) & (!X6) & (!X5) & (X4)
& (X3 ) ) || ( ( !X7) & (X6) & (X5) & (X3) & (X2) & (!X0 ) ) || ( ( X7) & (X6) &
(!X4) & (!X3) & (X1) & (!X0 ) ) || ( ( X7) & (!X6) & (X5) & (X2) & (!X1) & (!X0
) ) || ( ( !X7) & (!X6) & (X5) & (!X4) & (X3) & (X2 ) ) || ( ( X7) & (!X6) & (!X3)
& (X2) & (X1) & (!X0 ) ) || ( ( X7) & (!X6) & (X5) & (!X4) & (X3) & (!X2) & (!X0
) ) || ( ( X7) & (!X6) & (X4) & (X2) & (X1) & (!X0 ) ) || ( ( !X6) & (!X4) & (X3)
& (X2) & (X1) & (X0 ) ) || ( ( X7) & (!X6) & (!X4) & (X3) & (X1) & (X0 ) ) || (
( X7) & (X6) & (X5) & (!X4) & (X3) & (X2) & (X1));
   // Input Cost = 293 Gate Cost = 42
   Z6 =(( !X6) & (X5) & (!X3) & (!X2) & (!X1) & (X0 ) ) || ( ( !X7) & (X6) & (!X5)
& (X4) & (!X3) & (X2 ) ) || ( ( !X7) & (X6) & (!X5) & (X4) & (X1) & (X0 ) ) || (
( !X7) & (X6) & (!X3) & (X2) & (!X1) & (!X0 ) ) || ( ( X6) & (X5) & (X3) & (X2)
& (!X1) & (X0 ) ) || ( ( !X7) & (X6) & (X5) & (X4) & (X3) & (X2) & (X1) & (!X0 )
) || ( ( X7) & (!X6) & (X5) & (X4) & (!X2) & (X1 ) ) || ( ( X7) & (!X6) & (!X3)
& (X2) & (!X1 ) ) || ( ( X6) & (!X5) & (!X3) & (X2) & (X1) & (!X0 ) ) || ( ( X7)
& (X5) & (!X4) & (X3) & (X2) & (!X0 ) ) || ( ( X7) & (X6) & (X5) & (X4) & (!X2)
& (!X1) & (!X0 ) ) || ( ( X7) & (X6) & (X4) & (!X3) & (X1) & (X0 ) ) || ( ( !X7)
& (!X6) & (X5) & (X4) & (!X3) & (X0 ) ) || ( ( !X7) & (!X6) & (X4) & (X3) & (!X2)
& (!X1 ) ) || ( ( !X7) & (X5) & (X3) & (!X2) & (!X1) & (!X0 ) ) || ( ( !X7) & (X6)
& (X5) & (X3) & (!X2) & (X1) & (X0 ) ) || ( ( !X5) & (X4) & (!X3) & (X2) & (X1)
& (X0 ) ) || ( ( X7) & (!X6) & (X5) & (!X4) & (X1) & (X0 ) ) || ( ( X7) & (!X6)
```

```
& (X4) & (X2) & (!X1) & (X0 ) ) || ( ( X7) & (X6) & (!X5) & (X4) & (X3) & (X1) &
(!X0 ) ) || ( ( X7) & (X6) & (!X4) & (!X3) & (!X1) & (X0 ) ) || ( ( X7) & (X5) &
(!X4) & (!X3) & (X2) & (!X1 ) ) || ( ( X7) & (X6) & (X5) & (X4) & (X3) & (!X2) &
(!X1 ) ) || ( ( !X5) & (!X4) & (X3) & (!X2) & (!X1 ) ) || ( ( !X7) & (X5) & (!X4)
& (X3) & (X1) & (X0 ) ) || ( ( !X7) & (!X6) & (X5) & (!X4) & (X3) & (X2 ) ) || (
( !X7) & (!X6) & (X5) & (X3) & (X2) & (X1) & (X0 ) ) || ( ( X7) & (!X6) & (!X4)
& (X3) & (X1) & (!X0 ) ) || ( ( !X5) & (!X4) & (X3) & (!X1) & (X0 ) ) || ( ( X7)
& (!X5) & (X3) & (!X2) & (!X1) & (!X0 ) ) || ( ( !X7) & (!X6) & (!X5) & (X4) & (X3)
& (!X1 ) ) || ( ( !X7) & (!X5) & (X4) & (X3) & (!X2) & (X0 ) ) || ( ( !X7) & (X6)
& (X5) & (!X4) & (!X3) & (X1 ) ) || ( ( !X6) & (!X4) & (X3) & (X2) & (X1 ) ) ||
( ( X7) & (!X6) & (X5) & (!X4) & (!X2) & (X0 ) ) || ( ( X7) & (X6) & (!X5) & (!X2)
& (X1) & (X0 ) ) || ( ( !X7) & (!X6) & (!X4) & (X2) & (X1) & (!X0 ) ) || ( ( !X7)
& (!X6) & (!X5) & (X4) & (X3) & (!X0 ) ) || ( ( !X7) & (!X5) & (X4) & (!X2) & (X1)
& (!X0 ) ) || ( ( !X7) & (X6) & (!X5) & (X2) & (!X1) & (!X0 ) ) || ( ( !X7) & (X6)
& (!X4) & (!X2) & (!X1) & (!X0 ) ) || ( ( !X7) & (X6) & (X4) & (!X3) & (!X2) & (X1)
& (!X0 ) ) || ( ( !X7) & (!X5) & (X4) & (!X3) & (!X2) & (!X0 ) ) || ( ( X7) & (!X5)
& (!X4) & (!X2) & (!X1) & (!X0 ) ) || ( ( !X6) & (!X5) & (X4) & (!X2) & (!X1) &
(!X0));
   //Input Cost = 319 Gate Cost = 46
   Z5 = (( !X6) & (!X5) & (X4) & (!X3) & (!X2) & (!X1) & (!X0 ) ) || ( ( !X6) &
(!X5) & (X4) & (!X3) & (X1) & (X0 ) ) || ( ( !X7) & (!X6) & (!X5) & (X4) & (X2)
& (X1 ) ) || ( ( !X6) & (!X4) & (!X3) & (X2) & (!X1) & (X0 ) ) || ( ( !X7) & (X6)
& (!X5) & (!X4) & (!X2) & (X0 ) ) || ( ( X6) & (!X5) & (X4) & (!X3) & (!X2) & (X1)
& (!X0 ) ) || ( ( X6) & (X5) & (!X4) & (!X3) & (!X0 ) ) || ( ( X5) & (!X4) & (!X3)
& (!X2) & (!X1) & (!X0 ) ) || ( ( !X6) & (X5) & (X3) & (!X2) & (!X1) & (X0 ) ) ||
( ( X7) & (X5) & (X4) & (X2) & (X1) & (!X0 ) ) || ( ( X6) & (!X5) & (!X4) & (!X2)
& (!X1 ) ) || ( ( X7) & (X6) & (X5) & (!X4) & (!X2) & (X1) & (X0 ) ) || ( ( !X7)
& (X5) & (!X4) & (!X2) & (X1) & (!X0 ) ) || ( ( !X7) & (X6) & (X5) & (X4) & (!X3)
& (!X2) & (!X1 ) ) || ( ( X7) & (!X6) & (!X4) & (!X3) & (!X2) & (X1) & (X0 ) ) ||
( ( X7) & (!X6) & (X4) & (!X3) & (!X1) & (X0 ) ) || ( ( X7) & (!X6) & (X5) & (X4)
& (X3) & (X1 ) ) || ( ( X7) & (!X4) & (!X3) & (X2) & (!X1) & (X0 ) ) || ( ( X6)
& (X5) & (X4) & (!X3) & (X2) & (X1) & (X0 ) ) || ( ( !X7) & (!X6) & (X5) & (!X4)
& (X3) & (X1) & (X0 ) ) || ( ( X5) & (X4) & (X3) & (X1) & (!X0 ) ) || ( ( !X7) &
(X6) & (X5) & (X4) & (!X1) & (X0 ) ) || ( ( X7) & (!X6) & (!X5) & (!X3) & (X0 )
) || ( ( X7) & (!X6) & (!X5) & (!X4) & (X3) & (!X2) & (X1 ) ) || ( ( X7) & (!X6)
& (!X5) & (X4) & (!X2) & (!X1 ) ) || ( ( X7) & (X5) & (!X4) & (X3) & (X2) & (!X0
) ) || ( ( X7) & (X6) & (!X4) & (X2) & (X1) & (!X0 ) ) || ( ( X6) & (X5) & (X4)
& (!X2) & (!X1) & (X0 ) ) || ( ( !X7) & (X6) & (X3) & (!X2) & (!X1) & (X0 ) ) ||
( ( !X7) & (X6) & (!X5) & (X3) & (X2 ) ) || ( ( X7) & (!X6) & (!X5) & (!X4) & (!X3)
& (X2 ) ) || ( ( !X6) & (X5) & (X3) & (X2) & (X1) & (X0 ) ) || ( ( X7) & (X6) &
(X4) & (!X3) & (X1) & (!X0 ) ) || ( ( !X7) & (!X5) & (!X4) & (X3) & (!X1 ) ) ||
( ( !X7) & (X4) & (!X3) & (X2) & (X1) & (X0 ) ) || ( ( !X7) & (X6) & (!X4) & (X3)
& (!X2) & (!X0 ) ) || ( ( !X6) & (!X5) & (!X4) & (X3) & (X2) & (!X1 ) ) || ( ( !X6)
& (!X5) & (!X4) & (X3) & (!X2) & (!X0 ) ) || ( ( X7) & (X6) & (!X5) & (X4) & (X2)
& (!X1 ) ) || ( ( X7) & (!X5) & (X4) & (X2) & (!X1) & (!X0 ) ) || ( ( X7) & (X6)
& (X5) & (X2) & (!X0 ) ) || ( ( !X6) & (X4) & (X3) & (X2) & (X1) & (!X0 ) ) || (
( X7) & (X6) & (!X5) & (X4) & (X3) & (X1) & (X0 ) ) || ( ( X7) & (X6) & (!X5) &
(X4) & (X3) & (!X2) & (!X0));
   //Input Cost = 311 Gate Cost = 45
   Z4 = (( !X7) & (!X6) & (!X5) & (X4) & (!X3) & (X1) & (!X0 ) ) || ( ( !X7) & (!X6)
& (!X5) & (X3) & (!X1) & (!X0 ) ) || ( ( !X7) & (!X6) & (X5) & (!X4) & (!X3) & (!X2)
```

```
& (!X1) & (X0 ) ) || ( ( !X7) & (!X6) & (X5) & (!X2) & (X1) & (!X0 ) ) || ( ( !X7)
& (!X6) & (X5) & (!X4) & (X2) & (X1) & (X0 ) ) || ( ( X6) & (!X5) & (!X4) & (!X3)
& (!X1) & (X0 ) ) || ( ( X6) & (!X5) & (!X4) & (!X3) & (!X2) & (X1) & (!X0 ) ) ||
( ( !X7) & (!X5) & (!X4) & (X3) & (!X2) & (X1) & (X0 ) ) || ( ( !X7) & (X6) & (X4)
& (!X3) & (X2) & (!X1 ) ) || ( ( !X7) & (X6) & (X5) & (!X4) & (!X3) & (!X1) & (!X0
) ) || ( ( X6) & (X4) & (!X3) & (!X2) & (X1) & (X0 ) ) || ( ( !X7) & (X6) & (X5)
& (X4) & (X3) & (!X2) & (!X1) & (!X0 ) ) || ( ( X7) & (!X6) & (!X5) & (!X3) & (X1)
& (X0 ) ) || ( ( X7) & (!X6) & (X5) & (!X4) & (!X3) & (X2) & (!X1) & (X0 ) ) ||
( ( X7) & (X6) & (!X5) & (X3) & (!X2) & (!X0 ) ) || ( ( X7) & (X6) & (X5) & (X4)
& (!X3) & (!X1) & (!X0 ) ) || ( ( !X7) & (X5) & (X4) & (X3) & (X2) & (X1 ) ) ||
( ( X6) & (!X5) & (!X4) & (X3) & (!X1) & (!X0 ) ) || ( ( !X7) & (X6) & (!X5) & (X4)
& (X2) & (X1) & (X0 ) ) || ( ( X6) & (X5) & (X4) & (X2) & (X1) & (!X0 ) ) || ( (
X7) & (X6) & (!X4) & (X2) & (X1) & (!X0 ) ) || ( ( X7) & (X6) & (!X5) & (!X4) &
(X3) & (!X1 ) ) || ( ( X7) & (X6) & (X5) & (!X4) & (!X3) & (!X2) & (X0 ) ) || (
( X7) & (X6) & (X5) & (!X3) & (X2) & (!X0 ) ) || ( ( !X5) & (X4) & (!X3) & (X2)
& (!X1) & (X0 ) ) || ( ( !X7) & (!X6) & (!X5) & (X4) & (!X2 ) ) || ( ( X6) & (!X5)
& (X4) & (X3) & (X1) & (!X0 ) ) || ( ( X7) & (!X6) & (!X5) & (!X3) & (X2) & (!X1)
& (!X0 ) ) || ( ( X7) & (!X6) & (X4) & (X3) & (X0 ) ) || ( ( X7) & (!X5) & (X3)
& (X2) & (X1) & (!X0 ) ) || ( ( X7) & (!X5) & (!X3) & (!X2) & (!X1) & (X0 ) ) ||
( ( X7) & (X6) & (X5) & (X3) & (!X2) & (X1 ) ) || ( ( X6) & (X5) & (X4) & (X2) &
(!X1) & (X0 ) ) || ( ( !X7) & (!X6) & (X5) & (X3) & (X2) & (X0 ) ) || ( ( X7) &
(!X5) & (!X4) & (!X3) & (!X2) & (!X0 ) ) || ( ( X7) & (!X6) & (X4) & (!X3) & (!X2)
& (X1 ) ) || ( ( X7) & (X5) & (!X3) & (X2) & (X1) & (!X0 ) ) || ( ( X7) & (!X6)
& (!X4) & (X3) & (X2) & (!X0 ) ) || ( ( !X7) & (!X6) & (!X4) & (X2) & (!X1) & (!X0
) ) || ( ( !X7) & (!X6) & (!X5) & (!X4) & (X3) & (!X0 ) ) || ( ( X7) & (X5) & (X3)
& (!X2) & (X1) & (!X0 ) ) || ( ( X5) & (!X4) & (X3) & (X2) & (X1) & (X0 ) ) || (
( X7) & (!X6) & (X5) & (X4) & (X3) & (!X1 ) ) || ( ( !X7) & (!X6) & (!X4) & (X3)
& (!X2) & (!X0 ) ) || ( ( !X7) & (X6) & (X5) & (!X4) & (X3) & (X2) & (!X0 ) ) ||
( ( !X7) & (!X6) & (X5) & (X4) & (!X3) & (!X1) & (!X0));

   // Input Cost = 337 Gate Cost = 47

   Z3 =( ( !X7) & (X6) & (!X4) & (!X3) & (X2) & (!X1 ) ) || ( ( !X7) & (X6) & (X5)
& (!X4) & (!X1 ) ) || ( ( X7) & (!X5) & (!X4) & (!X3) & (!X2) & (!X0 ) ) || ( (
X7) & (!X6) & (X5) & (!X4) & (X3) & (!X0 ) ) || ( ( X7) & (X6) & (X5) & (!X3) &
(!X2) & (X1) & (X0 ) ) || ( ( X7) & (!X4) & (X3) & (!X2) & (X1) & (!X0 ) ) || (
( X7) & (X6) & (X5) & (X4) & (X3) & (X2) & (!X0 ) ) || ( ( !X7) & (!X6) & (X5) &
(!X3) & (!X2 ) ) || ( ( !X7) & (X6) & (X5) & (!X3) & (X2) & (X1) & (X0 ) ) || (
( !X7) & (X5) & (X4) & (!X3) & (!X1) & (!X0 ) ) || ( ( !X6) & (X5) & (!X4) & (!X3)
& (X1) & (!X0 ) ) || ( ( X7) & (X6) & (!X5) & (!X2) & (!X1) & (!X0 ) ) || ( ( X6)
& (!X5) & (X3) & (!X2) & (X1) & (X0 ) ) || ( ( X6) & (X5) & (!X4) & (!X3) & (!X2)
& (!X1) & (!X0 ) ) || ( ( X7) & (X5) & (!X4) & (X3) & (!X2) & (!X1) & (X0 ) ) ||
( ( X6) & (X4) & (!X3) & (!X2) & (X1) & (!X0 ) ) || ( ( !X6) & (!X5) & (X4) & (!X3)
& (!X2) & (X1 ) ) || ( ( X7) & (!X6) & (X3) & (!X2) & (!X1) & (X0 ) ) || ( ( X7)
& (!X6) & (!X4) & (X3) & (X2) & (X1 ) ) || ( ( X7) & (X6) & (!X4) & (!X3) & (X2)
& (X1) & (!X0 ) ) || ( ( !X6) & (!X5) & (!X3) & (!X2) & (X1) & (X0 ) ) || ( ( X7)
& (!X6) & (X3) & (!X2) & (X1) & (!X0 ) ) || ( ( X7) & (!X6) & (X5) & (X4) & (X2)
& (X1) & (X0 ) ) || ( ( !X5) & (!X4) & (X3) & (X2) & (X1) & (!X0 ) ) || ( ( !X7)
& (!X4) & (X3) & (!X2) & (X1) & (X0 ) ) || ( ( X7) & (X5) & (X4) & (!X3) & (!X2)
& (X0 ) ) || ( ( !X6) & (X5) & (X4) & (!X3) & (X2) & (!X0 ) ) || ( ( !X6) & (!X5)
& (X4) & (X2) & (!X1 ) ) || ( ( X7) & (!X6) & (X5) & (X4) & (!X2) & (!X1 ) ) ||
( ( X7) & (X6) & (X5) & (X4) & (X2) & (!X1) & (X0 ) ) || ( ( !X7) & (!X6) & (X5)
& (X4) & (!X0 ) ) || ( ( X7) & (!X5) & (!X4) & (X2) & (!X1) & (X0 ) ) || ( ( X7)
```

```
& (X6) & (X4) & (X3) & (!X1) & (!X0 ) ) || ( ( !X7) & (X6) & (X3) & (!X2) & (!X1)
& (X0 ) ) || ( ( !X7) & (!X4) & (X3) & (X2) & (!X1) & (X0 ) ) || ( ( !X7) & (X6)
& (!X5) & (X4) & (!X3) & (!X1) & (X0 ) ) || ( ( !X7) & (X6) & (!X5) & (X4) & (X3)
& (!X0 ) ) || ( ( !X7) & (X5) & (X3) & (X2) & (!X1) & (X0 ) ) || ( ( !X7) & (!X5)
& (X4) & (X3) & (X2) & (X1 ) ) || ( ( !X7) & (!X5) & (X4) & (X2) & (X1) & (X0 )
) || ( ( X7) & (X6) & (!X5) & (X4) & (X3) & (X2) & (X0));
   //Input Cost = 293 Gate Cost = 42
   Z2 =(( !X7) & (X6) & (!X4) & (!X3) & (!X2) & (X1) & (X0 ) ) || ( ( !X6) & (!X5)
& (!X3) & (!X2) & (X1) & (!X0 ) ) || ( ( X7) & (!X6) & (X5) & (!X4) & (X3) & (X0
) ) || ( ( X6) & (!X5) & (X3) & (!X2) & (X1) & (!X0 ) ) || ( ( X7) & (X6) & (X5)
& (!X3) & (!X2) & (X1) & (!X0 ) ) || ( ( X7) & (X6) & (X4) & (X3) & (!X2) & (X1)
& (X0 ) ) || ( ( X7) & (X6) & (X5) & (X4) & (X3) & (X2) & (!X1 ) ) || ( ( !X7) &
(X5) & (!X4) & (X3) & (X2) & (X1) & (X0 ) ) || ( ( !X7) & (X5) & (X4) & (!X3) &
(!X2) & (X0 ) ) || ( ( X7) & (X6) & (X5) & (X4) & (!X3) & (!X1) & (!X0 ) ) || (
( !X7) & (X6) & (X4) & (X3) & (X2) & (X1) & (X0 ) ) || ( ( X7) & (!X6) & (!X4) &
(X3) & (X2) & (X1) & (!X0 ) ) || ( ( X7) & (!X6) & (X3) & (!X2) & (!X1 ) ) || (
( X7) & (!X6) & (X5) & (!X3) & (X2) & (X0 ) ) || ( ( X7) & (X5) & (!X4) & (X3) &
(!X2) & (!X1 ) ) || ( ( X6) & (X4) & (!X3) & (!X2) & (!X1) & (X0 ) ) || ( ( X7)
& (X6) & (X4) & (!X3) & (X2) & (X1) & (X0 ) ) || ( ( !X6) & (!X5) & (X4) & (!X3)
& (!X2) & (X0 ) ) || ( ( !X6) & (X5) & (X4) & (!X3) & (!X2) & (!X0 ) ) || ( ( X7)
& (!X6) & (X5) & (X4) & (X2) & (X1 ) ) || ( ( X7) & (X6) & (!X4) & (!X3) & (X2)
& (!X1) & (X0 ) ) || ( ( !X7) & (!X6) & (X5) & (X4) & (!X2) & (X0 ) ) || ( ( !X5)
& (!X4) & (X3) & (X2) & (!X1 ) ) || ( ( !X5) & (X4) & (X3) & (!X2) & (X1) & (X0
) ) || ( ( X7) & (!X5) & (!X4) & (!X2) & (!X1) & (X0 ) ) || ( ( X7) & (X6) & (!X5)
& (X2) & (X1) & (X0 ) ) || ( ( !X7) & (X6) & (!X5) & (X4) & (X2) & (X1 ) ) || (
( !X6) & (X5) & (!X4) & (!X3) & (!X1) & (X0 ) ) || ( ( !X7) & (X6) & (X5) & (!X3)
& (X2) & (X1) & (!X0 ) ) || ( ( !X7) & (X5) & (X3) & (X2) & (!X1) & (!X0 ) ) ||
( ( !X7) & (!X6) & (X5) & (X4) & (!X1) & (X0 ) ) || ( ( !X7) & (X6) & (X5) & (!X4)
& (X1) & (X0 ) ) || ( ( X7) & (!X6) & (!X5) & (X2) & (!X1) & (!X0 ) ) || ( ( !X6)
& (X5) & (X4) & (!X3) & (X1) & (X0 ) ) || ( ( !X7) & (X6) & (X3) & (!X2) & (!X1)
& (!X0 ) ) || ( ( X6) & (!X5) & (!X4) & (X2) & (!X1) & (!X0 ) ) || ( ( !X7) & (X6)
& (X5) & (!X4) & (!X1) & (!X0 ) ) || ( ( !X7) & (!X4) & (X3) & (!X2) & (X1) & (!X0
) ) || ( ( !X7) & (!X6) & (X5) & (!X3) & (!X2) & (!X0 ) ) || ( ( !X7) & (X6) & (!X5)
& (X4) & (X3) & (X0 ) ) || ( ( !X7) & (!X6) & (!X5) & (X4) & (X3) & (X2 ) ) || (
( !X7) & (!X5) & (X4) & (!X3) & (X2) & (!X0 ) ) || ( ( !X7) & (X6) & (!X5) & (X4)
& (!X2) & (!X1 ) ) || ( ( X7) & (X6) & (!X5) & (X4) & (X3) & (X2) & (!X0));
   //Input Cost = 318 Gate Cost = 45
   Z1 =(( !X7) & (X6) & (!X4) & (!X3) & (!X2) & (X1 ) ) || ( ( !X7) & (X6) & (X5)
& (!X3) & (X2) & (X0 ) ) || ( ( !X7) & (!X4) & (X3) & (!X2) & (X0 ) ) || ( ( X7)
& (!X6) & (!X4) & (X3) & (X2) & (!X1 ) ) || ( ( X7) & (!X6) & (X5) & (!X4) & (X3)
& (!X0 ) ) || ( ( X7) & (!X6) & (X5) & (X2) & (X1) & (!X0 ) ) || ( ( X6) & (!X5)
& (X4) & (X3) & (!X1) & (X0 ) ) || ( ( X7) & (X6) & (X5) & (!X3) & (!X2) & (!X1)
& (X0 ) ) || ( ( X7) & (!X4) & (X3) & (!X2) & (!X1) & (!X0 ) ) || ( ( X7) & (X5)
& (!X4) & (X2) & (X1) & (X0 ) ) || ( ( X7) & (X6) & (X4) & (X3) & (!X2) & (X1 )
) || ( ( X7) & (X6) & (X5) & (X4) & (X3) & (X2) & (!X1) & (!X0 ) ) || ( ( !X6) &
(X5) & (!X4) & (X3) & (X2) & (X1 ) ) || ( ( !X7) & (X6) & (!X5) & (X2) & (X1) &
(X0 ) ) || ( ( !X7) & (X6) & (X5) & (!X4) & (X1) & (!X0 ) ) || ( ( !X7) & (X5) &
(X4) & (!X3) & (!X2) & (!X0 ) ) || ( ( !X7) & (X5) & (X3) & (!X2) & (X1) & (X0 )
) || ( ( X7) & (!X6) & (X5) & (X4) & (X2) & (!X1) & (X0 ) ) || ( ( X7) & (!X5) &
(!X4) & (!X2) & (X1) & (X0 ) ) || ( ( X6) & (!X5) & (!X4) & (X3) & (!X2) & (X0 )
) || ( ( X6) & (X4) & (!X3) & (!X2) & (!X1) & (!X0 ) ) || ( ( X7) & (X6) & (X5)
```

```
& (X4) & (!X3) & (!X2) & (X0 ) ) || ( ( X7) & (X6) & (X4) & (!X3) & (X2) & (X1)
& (!X0 ) ) || ( ( !X6) & (!X5) & (X4) & (!X3) & (!X2 ) ) || ( ( !X7) & (!X5) & (X4)
& (X2) & (!X1) & (X0 ) ) || ( ( X6) & (!X5) & (X4) & (X2) & (X1) & (!X0 ) ) || (
( !X7) & (X6) & (!X5) & (X4) & (X3) & (!X0 ) ) || ( ( !X7) & (X6) & (X4) & (X3)
& (X2) & (X1) & (!X0 ) ) || ( ( !X6) & (!X5) & (X4) & (!X2) & (X1 ) ) || ( ( !X6)
& (X4) & (!X3) & (!X2) & (!X1) & (X0 ) ) || ( ( !X7) & (!X5) & (X4) & (X3) & (X2
) ) || ( ( !X7) & (!X6) & (X5) & (X4) & (!X2) & (!X0 ) ) || ( ( !X7) & (X6) & (!X5)
& (!X3) & (X1) & (X0 ) ) || ( ( X7) & (!X6) & (!X3) & (X2) & (X1) & (X0 ) ) || (
( !X6) & (X5) & (X4) & (!X3) & (X1) & (!X0 ) ) || ( ( X7) & (X6) & (!X5) & (!X4)
& (X2) & (!X0 ) ) || ( ( !X7) & (!X5) & (X3) & (X2) & (!X1) & (!X0 ) ) || ( ( !X7)
& (!X6) & (X5) & (X4) & (!X1) & (!X0 ) ) || ( ( !X6) & (X5) & (!X3) & (X2) & (!X1)
& (!X0 ) ) || ( ( X7) & (!X6) & (X3) & (!X2) & (!X1) & (!X0 ) ) || ( ( X7) & (!X5)
& (!X3) & (!X2) & (!X1) & (!X0 ) ) || ( ( X7) & (X5) & (!X4) & (!X3) & (X2) & (!X1)
& (!X0 ) ) || ( ( !X6) & (!X5) & (!X3) & (!X2) & (!X1) & (X0 ) ) || ( ( !X6) & (X5)
& (!X4) & (!X3) & (!X1) & (!X0 ) ) || ( ( !X7) & (!X6) & (!X3) & (!X2) & (!X1) &
(X0));
   //Input Cost = 319 Gate Cost = 46
   Z0 =(( !X6) & (X5) & (!X3) & (!X2) & (X1) & (X0 ) ) || ( ( !X6) & (X5) & (X4)
& (!X3) & (X0 ) ) || ( ( !X7) & (X6) & (!X4) & (!X3) & (!X2) & (!X1) & (X0 ) ) ||
( ( !X7) & (X6) & (!X5) & (X4) & (!X3) & (X2 ) ) || ( ( !X7) & (X6) & (X5) & (X3)
& (X2) & (X0 ) ) || ( ( X7) & (!X6) & (!X4) & (X3) & (X2) & (!X1) & (!X0 ) ) ||
( ( X6) & (!X5) & (X4) & (X3) & (!X1 ) ) || ( ( X7) & (X6) & (X5) & (!X3) & (!X2)
& (!X1) & (!X0 ) ) || ( ( !X7) & (X5) & (X4) & (!X3) & (!X2) & (!X1) & (X0 ) ) ||
( ( !X7) & (X6) & (X5) & (!X3) & (X2) & (!X0 ) ) || ( ( !X6) & (!X5) & (X4) & (!X2)
& (!X1) & (X0 ) ) || ( ( X7) & (X6) & (X5) & (X4) & (!X2) & (X1) & (!X0 ) ) || (
( X7) & (X6) & (X4) & (!X3) & (X2) & (!X1) & (X0 ) ) || ( ( X7) & (X6) & (X5) &
(X4) & (X3) & (!X2) & (X0 ) ) || ( ( !X7) & (!X5) & (X4) & (X2) & (!X1) & (!X0 )
) || ( ( !X7) & (!X6) & (!X5) & (X4) & (X3) & (X2 ) ) || ( ( !X7) & (!X6) & (X4)
& (X3) & (!X2) & (X0 ) ) || ( ( !X7) & (!X5) & (X3) & (!X2) & (X1) & (X0 ) ) ||
( ( !X7) & (X6) & (X5) & (!X4) & (!X1) & (X0 ) ) || ( ( !X7) & (X5) & (X3) & (!X2)
& (X1) & (!X0 ) ) || ( ( X7) & (!X6) & (!X5) & (!X3) & (!X2) & (!X0 ) ) || ( ( X7)
& (!X6) & (X4) & (X2) & (X1) & (X0 ) ) || ( ( X7) & (X6) & (!X5) & (!X4) & (X2)
& (X0 ) ) || ( ( !X7) & (X6) & (!X5) & (X2) & (X1) & (!X0 ) ) || ( ( !X7) & (X6)
& (!X4) & (!X3) & (X1) & (!X0 ) ) || ( ( !X7) & (!X4) & (X3) & (!X2) & (!X0 ) )
|| ( ( X7) & (!X6) & (!X3) & (X2) & (X1 ) ) || ( ( !X7) & (!X6) & (!X3) & (!X2)
& (!X1) & (!X0 ) ) || ( ( !X5) & (!X4) & (X3) & (X1) & (X0 ) ) || ( ( X7) & (!X6)
& (X5) & (X2) & (!X1) & (X0 ) ) || ( ( X7) & (X6) & (!X5) & (X3) & (!X2) & (!X0
) ) || ( ( X7) & (X6) & (!X4) & (!X3) & (X1) & (X0 ) ) || ( ( X7) & (X6) & (X5)
& (!X4) & (X2) & (X1 ) ) || ( ( !X7) & (X5) & (!X4) & (X3) & (X2) & (X0 ) ) || (
( X7) & (!X6) & (X5) & (!X4) & (X3) & (!X2) & (X0 ) ) || ( ( X7) & (!X6) & (X5)
& (X4) & (X2) & (!X1 ) ) || ( ( X7) & (!X5) & (!X4) & (!X2) & (X1) & (!X0 ) ) ||
( ( !X6) & (!X5) & (X4) & (!X2) & (X1) & (!X0 ) ) || ( ( !X6) & (X5) & (!X4) & (X3)
& (X2) & (X1) & (!X0 ) ) || ( ( !X6) & (X4) & (!X3) & (!X2) & (!X1 ) ) || ( ( !X7)
& (X6) & (!X5) & (!X3) & (X1) & (!X0));
   //Input Cost = 290 Gate Cost = 42
   //***Total Input Cost = 2480*** //***Total Gate Cost = 355***

   Z0=Z0*1;

   Z1=Z1*2;
```

```
   Z2=Z2*4;

   Z3=Z3*8;

   Z4=Z4*16;

   Z5=Z5*32;

   Z6=Z6*64;

   Z7=Z7*128;

   Y=Z0+Z1+Z2+Z3+Z4+Z5+Z6+Z7;

   return Y;

}

//*****************************************************************************
//*****************************************************************************
int Alu(char C ,int Op_A, int Op_B)
{
   int Multiplicacion=0,Suma=0,Resta=0,Division=0;


            if(C=='+')
             {
                      Suma=((Op_A)^(Op_B));
                           return Suma;
                     }

            if(C=='-')
             {
                           Resta=(Op_A)^(Op_B);
                           return Resta;
                     }



            if(C=='*')
             {
                                      if(((Op_A)==0)||((Op_B)==0))
       {
          Multiplicacion=0;
                                         return 0;
                                      }
       else
                                      {
                                       Multiplicacion=(Log(Op_A))+(Log(Op_B));
                                         if(Multiplicacion>255)
          {
```

```
          Multiplicacion=(Multiplicacion-255);
                                                return AntLog(Multiplicacion);
                                        }
                                        if(Multiplicacion==255)
          {
          Multiplicacion=0;
                                        }
                                    }
                             return AntLog(Multiplicacion);
                        }

    if(C=='/')
              {
                               if(((Op_A)==0)||((Op_B)==0))
       {
          Division=0;
                                        return 0;
                                    }
       else
                                    {
       Division=(Log(Op_A))-(Log(Op_B));
         if(Division<0)
          {
          Division=Division+255;
             }
                                    }
                            return AntLog(Division);
                        }
             return 0;
}

//*****************************************************************************
//*****************************************************************************
void dispersa(char *file)
{
   char c1, c2;
   int i, j, t, size;

   if ((in=fopen(file, "rb"))==NULL)
   {  printf("no puedo abrir el archivo fuente\n");
      exit(1);
   }

   if ((out0=fopen("s0", "wb"))==NULL)
   {  printf("no puedo crear un disperso\n");
      exit(1);
   }

   if ((out1=fopen("s1", "wb"))==NULL)
   {  printf("no puedo crear un disperso\n");
      exit(1);
```

```
}

if ((out2=fopen("s2", "wb"))==NULL)
{  printf("no puedo crear un disperso\n");
   exit(1);
}

if ((out3=fopen("s3", "wb"))==NULL)
{  printf("no puedo crear un disperso\n");
   exit(1);
}

if ((out4=fopen("s4", "wb"))==NULL)
{  printf("no puedo crear un disperso\n");
   exit(1);
}



for (i=0; i<5; i++)
   switch (i)
   {  case 0:  putc(' ', out0);
               for (j=0; j<3; j++) putc(a[i][j], out0);
               break;
      case 1:  putc(' ', out1);
               for (j=0; j<3; j++) putc(a[i][j], out1);
               break;
      case 2:  putc(' ', out2);
               for (j=0; j<3; j++) putc(a[i][j], out2);
               break;
      case 3:  putc(' ', out3);
               for (j=0; j<3; j++) putc(a[i][j], out3);
               break;
      case 4:  putc(' ', out4);
               for (j=0; j<3; j++) putc(a[i][j], out4);
               break;
   };


size=0;
while (!feof(in))
{
   for (j=0; j<3; j++)
      b[j]=0;

   j=0;
   do
   {  b[j++]=getc(in);

      if (!feof(in))
      {
```

```
            size++;
         };


      }
      while ((!feof(in))&&(j<3));

      if (j>0)
      for (i=0; i<5; i++)
      {  t=0;
         for (j=0; j<3; j++)
            t=Alu('+',t, Alu('*',a[i][j],b[j]));
         c2=t;

         switch (i)
         {  case 0:  putc(t, out0);
                     break;
            case 1:  putc(t, out1);
                     break;
            case 2:  putc(t, out2);
                     break;
            case 3:  putc(t, out3);
                     break;
            case 4:  putc(t, out4);
                     break;
         };

      };

   };

   fclose(in);

   rewind(out0);    putc(size%3, out0);   fclose(out0);
   rewind(out1);    putc(size%3, out1);   fclose(out1);
   rewind(out2);    putc(size%3, out2);   fclose(out2);
   rewind(out3);    putc(size%3, out3);   fclose(out3);
   rewind(out4);    putc(size%3, out4);   fclose(out4);
}


//@
//--------------------------------------------------------------------
main(int argc, char **argv)
{
   int Multiplicacion=0;
   int Suma = 0;
   float Aux_A = 0,Aux_B = 0,Aux_C = 0;
   clock_t start_A, end_A,start_B, end_B,start_C, end_C;

   //************************************************************
```

```
 if ((out5=fopen("Tiempo_Disperso", "wb"))==NULL)
{  printf("no puedo crear  Disperso\n");
   exit(1);
}

if ((out6=fopen("Tiempo_Multiplicacion", "wb"))==NULL)
{  printf("no puedo crear Multiplicacion\n");
   exit(1);
}

if ((out7=fopen("Tiempo_Suma", "wb"))==NULL)
{  printf("no puedo crear Suma\n");
   exit(1);
}
//*****************************************************************

//*****************************************************************
start_A = clock();
if (argc!=2)
{  printf("olvido ingresar el nombre del archivo fuente\n");
   exit(1);
}
 dispersa(argv[1]);
 end_A = clock();
printf("El tiempo de dispersion es :
%f\n", ((end_A - start_A) / CLK_TCK));//el tiempo en segundos
//*****************************************************************

//*****************************************************************
start_B = clock();
for(int r=0;r<300000;r++)
{
     Multiplicacion = Alu('*',33,25);
}
end_B = clock();
printf("El tiempo empleado en la Multiplicacion es :
%f\n", ((end_B- start_B) / CLK_TCK));//el tiempo en segundos
//cout<<"El tiempo en la Multiplicacion es : "<<Aux_B<<endl;
//*****************************************************************

//*****************************************************************
start_C = clock();
for(int o=0;o<1200000;o++)
{
     Suma = Alu('+',33,25);
}
end_C = clock();
printf("El tiempo empleado en la Suma es :
%f\n", ((end_C- start_C) / CLK_TCK));//el tiempo en segundos
//*****************************************************************
```

```
}
```

Referencias

[1] **Health Tech Publishing Co. inc.,*Medical Imaging*, 17:110-114, 2002.**

[2] **Strang G. y Nguyen T. *Wavelets and Filter Banks*. Wellesley-Cambridge Press, EUA, 1997. Pp.2-7.**

[3] **Klappenecker A., May F.U. y Nückel A. *Lossless Compression using Wavelets over Finite Rings and Related Architectures*, en Wavelet Applications in Signal and Image Processing V, Proc. SPIE, 3169:139-147, 1997.**

[4] **Kurosch A.G. *Curso de Álgebra Superior*, Editorial Mir, Moscú, 1994, Pp.271-319.**

[5] **Swanson M.D.*A Binary Wavelet Decomposition of Binary Images*,IEEE TRANSACTIONS ON IMAGE PROCESSING, 5:1637-1650, 1996.**

[6] **Rabin M.O. *Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance.* JACM, 36:335-348, 1989.**

[7] **Tanenbaum A.S., *Sistemas Operativos Distribuidos*, Prentice-Hall Hispanoamericana, S.A., 1996, pp.2-3.**

[8] **http://www.digilentinc.com**

[9] **Charte F. *Programación con C++ Builder*, Ediciones Anaya Multimedia, S.A.,1997, ISBN: 84-415-0203-X.**

[10] **Tapia H.*Introducción a Campos Finitos con Aplicación a Códigos y Criptografía*,Universidad Autónoma Metropolitana, México, 2008, pp. 5-15.**

[11] **McEliece R.J. *Finite fields for computer scientists and engineers*. Kluwer academic publishers, Boston EUA, 1986 1998. Pp. 147-156. 1998, pp. 147-156.**

[12] **Klindworth A.*FPLD-implementation ofcomputations over finite fields $GF(2^m)$ with applications to error control coding*, en Field-programmable logic and applications: Proceedings of the 5th international workshop FPL '95, 975:261-271, 1995.**

[13] **Daemen J. y Rijmen V. *The Design of Rijndael*. AES - The Advanced Encryption Standard. Springer, 2001. 248 pp. 1965.**

[14] **The Mathworks Inc. *Signal Processing Operations in Galois Fields*. Página Web de ayuda del Toolbox de Comunicaciones de Matlab: http://www.mathworks.com/access/helpdesk/help/toolbox/comm/index.html? /access/helpdesk/help/toolbox/comm/ug/fp6075.html& http://www.google.com.mx /search?hl=es&sa=X&oi=spell&resnum=0 &ct=result&cd=1&q=finite+field+ processing+signal&spell=1**

[15] **Herstein I.N. *Álgebra moderna*. Trillas, 1980. 200 Pp.**

[16] **MacWilliams F.J. y Sloane N.J.A. *The Theory of Error-Correcting Codes*. 2a edición. North-Holland Pub. Co., 1977. 110 pp.**

[17] **Mullen G.L.** ***Finite Fields with Applications to Coding Theory, Cryptography and Related Areas*****. Springer, ACM Press, 1999, pp. 164-176.**

[18] **Shannon C. E.** ***A mathematical theory of communication*****, Bell System Technical Journal, vol. 27, pp. 379-423 and 623-656, July and October, 1948.**

[19] **Sayood K.** ***Introduction to data Compression*****. 2a edición. Morgan Kaufmann, 2000, Pp.1-10, ISBN:1-55860-558-4.**

[20] **Mallat S.*****A theory for multiresolution signal decomposition: the wavelet representation*****. IEEE Pattern Anal. and Machine Intell., 11:674-693, 1989.**

[21] **Misiti M., Misiti Y., Oppenheim G., Poggi J.M.** ***Wavelet TOOLBOX For Use with MATLAB*****. The Math Works Inc., 1996, Pp.1-1 - 1-29.**

[22] **Kouro S. y Musalem R.** ***Tutorial introductorio a la teoría de wavelet*****. Archivo disponible en: http://www.elo.utfsm.cl/ elo377/documentos/Wavelet.pdf**

[23] **Gonzáles C.R., Woods E.R.** ***Tratamiento digital de imágenes*****. Addison-Wesley/Diaz de Santos, 1996, ISBN: 0-201-62576-8, Pp.6-7.**

[24] **Proakis J.G., Manolakis D. G.*****Tratamiento digital de señales*****. 3ra edición, Prentice hall, Madrid, 1998, ISBN: 84-8322-000-8, Pp. 565-596.**

[25] **Klappenecker A., May F.U. y Nückel A.** ***Lossless Image Compression Using Wavelets Over Finite Rings and Related Architectures*****. Wavelet Applications in Signal and Image Processing V, Proc. SPIE 3169:139-147, 1997.**

[26] **Abramson N.** ***Information Theory and Coding*****. McGraw-Hill, 1963, Pp. 201.**

Manipulación de imágenes médicas sobre campos finitos

Tesis que presenta
José Francisco Rodríguez arellano
Para obtener el grado de
Maestro en Ciencias en Ingeniería Biomédica

Asesores:
M. en C. Óscar Yáñez Suárez
Dr. Ricardo Marcelín Jiménez

Jurado Calificador:
Presindete: M. en C. Óscar Yáñez Suárez
Secretario: Dr. Adriano de Luca Penachia
Vocal: Dr. Miguel Ángel Ruiz Sánchez

23 de noviembre de 2010