PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

IMPLEMENTACIÓN DE UN CODIFICADOR/DECODIFICADOR WAVELET PARA LA COMPRESIÓN DE IMÁGENES SOBRE UN FPGA

Tesis para Optar el Título de:

INGENIERO ELECTRÓNICO

Presentado por:

CHRIS DENNIS TOMÁS HORNA

CHRISTIAN ALEXIS HUERTAS SAONA

Lima – Perú

2005

RESUMEN

En los últimos años, los esquemas de compresión de imágenes basados en la transformada de Wavelet han ido remplazando a los esquemas clásicos basados en la transformada de Fourier, ya que son más eficientes y ofrecen una mayor posibilidad de análisis debido a su naturaleza multiresolución[1]. La compresión de imágenes actualmente juega un papel muy importante en el desarrollo de equipos portátiles o de telecomunicaciones, ya que estos buscan almacenar grandes volúmenes de información en el menor espacio posible o transmitir información a altas velocidades. Por lo tanto, esto implica migrar de la solución *software* originalmente concebida, hacia su implementación en procesadores de aplicación específica (*hardware*), la cual ofrece una mayor flexibilidad y la posibilidad de procesar los datos en tiempo real. Este trabajo presenta la implementación de una versión optimizada del algoritmo de codificación/decodificación SPIHT (Set Partitioning In Hierarchical Trees) sobre un arreglo de puertas programables por campo (FPGA), para lograr una reducción considerable del tiempo de procesamiento. Para ello, se propone una metodología de diseño digital Top-Down capaz de adaptar el estado del arte de un algoritmo específico a su equivalente en hardware programable. Los resultados de las pruebas experimentales demuestran que el diseño alcanza un reducido tiempo de procesamiento, logrando codificar una imagen transformada al dominio Wavelet de 256x256 píxeles en 50 milisegundos y realizar la decodificación de las misma en menos de un milisegundo. Además posee un bajo consumo de recursos, ocupando un 27% del FPGA Stratix EP1S25F1020C5 de Altera para dicha configuración. El sistema esta configurado para comunicarse con una interfaz de usuario visual para la transferencia de imágenes y visualización de resultados por la PC a través del Bus PCI.

ÍNDICE

Pág.

PLANOS

ANEXOS (ver CD-ROM adjunto)

A. Archivos VHDL del proyecto

A-1. Archivos de la descripción VHDL del procesador GMS

A-2. Archivos de la descripción VHDL del procesador CODEC

A-3. Archivos de la descripción VHDL del diseño de referencia

A-4. Archivos VHDL necesarios para la implementación del sistema total

B. Simulaciones por bloques

C. Pseudo-Códigos del algoritmo GMS y del algoritmo CODEC

D. Diagramas de Estado

E. Código de los programas en MATLAB

F. Códigos fuente del software de interfaz de usuario

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. LA CODIFICACIÓN WAVELET

1.1. LA TRANSFORMADA DISCRETA DE WAVELET

Esta sección describe algunas de las propiedades de la transformada discreta de Wavelet (DWT) que son utilizadas en la compresión de imágenes. La forma más sencilla y general de la transformada de Wavelet unidimensional (DWT-1D) es mostrada en la figura 1.1. Se puede observar como una señal *X* pasa a través de un filtro paso-bajo y un filtro paso-alto (h y g respectivamente), para luego ser sub-muestreada por un factor de dos; construyendo de esta manera el primer nivel de transformación. Los múltiples niveles o escalas de la transformada de Wavelet son construidos por una repetición del proceso de filtraje y decimación sobre las salidas paso-bajo únicamente. Este proceso es típicamente efectuado hasta un número finito "K" de niveles, dando como resultado un juego de coeficientes $d_{i1}(n)$, i $\in$ {1,...,K} y $d_{K0}(n)$ llamados coeficientes Wavelet [1].

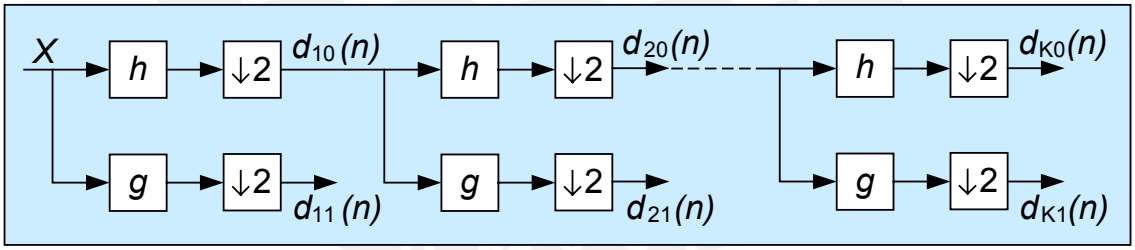


FIGURA 1.1.- UNA DESCOMPOSICIÓN WAVELET 1-D DE K NIVELES

Además, ésta transformación puede ser extendida hacia la transformada bidimensional de wavelet (DWT-2D), al aplicar la DWT-1D sobre las filas de la matriz de entrada, y luego repetir el mismo proceso sobre las columnas. Esto es valido, debido a la característica de superposición que posee la DWT [2]. La figura 1.2 muestra el procesamiento efectuado sobre las filas y las columnas de una imagen de tamaño N x N para la obtención de la transformada bidimensional de Wavelet de un

nivel (K=1) o de una octava. Como resultado, son obtenidas cuatro sub-bandas: la primera (LL) se debe al filtraje paso-bajos de las filas y las columnas, la segunda (LH) se debe al filtraje paso-bajo de las filas y al filtraje paso-alto de las columnas, la tercera (HL) se debe al filtraje paso-alto de las filas y al filtraje paso-bajos de las columnas y la cuarta (HH) se debe al filtraje paso-alto de las filas y columnas.

Estos cuatro grupos de coeficientes proporcionan: La aproximación (Sub-banda LL), los detalles horizontales (Sub-banda LH), los detalles verticales (Sub-banda HL) y los detalles diagonales (Sub-banda HH) de la imagen original. La aproximación contiene la mayor parte de la energía de la imagen, es decir, la información más importante, mientras que los detalles tienen valores próximos a cero.

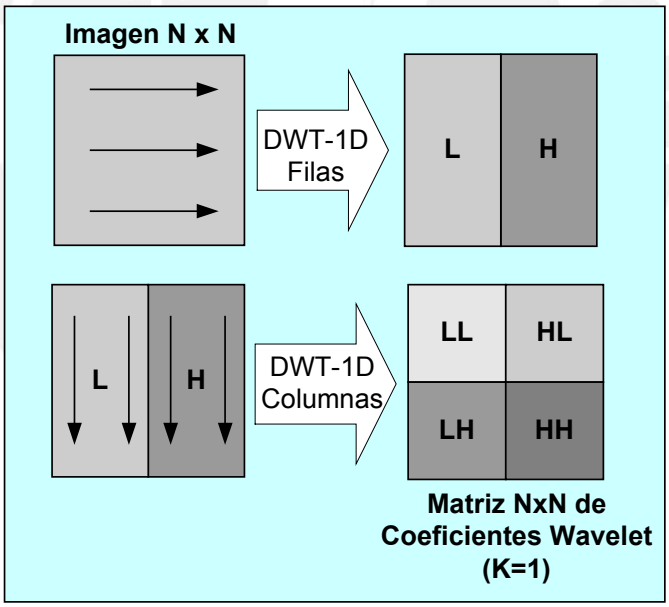


FIGURA 1.2.-. TRANSFORMADA BIDIMENSIONAL DE WAVELET PARA UN NIVEL

En la figura 1.3, se puede apreciar el resultado de la aplicación de la transformada directa de Wavelet (FWT) en descomposición completa (número máximo de octavas), realizada sobre una imagen estándar haciendo uso del software MATLAB. El segundo, tercer y cuarto cuadrante de la imagen transformada, contienen los

detalles obtenidos por el primer nivel de descomposición, y representan las bandas de más alta frecuencia y de mayor resolución. A la inversa, el último nivel de descomposición corresponde a la banda de frecuencia más baja, donde la resolución es más burda. Así, al desplazarse de los últimos niveles a los primeros, se observa una disminución de la energía contenida en las sub-bandas recorridas.

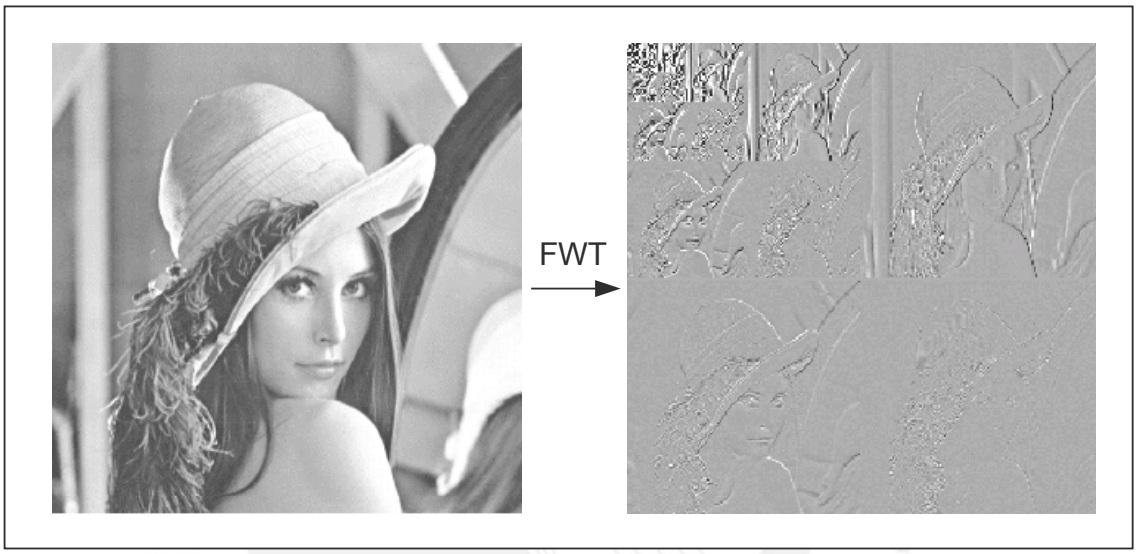


FIGURA 1.3.- TRANSFORMADA DIRECTA DE WAVELET (FWT) EN DESCOMPOSICIÓN COMPLETA

1.2. COMPRESIÓN DE IMÁGENES

Una característica común en las imágenes es la correlación entre píxeles vecinos o entre sus componentes espectrales, lo cuál acredita la existencia de información redundante. La tarea fundamental en los esquemas de compresión de imágenes es reducir esta redundancia, sin afectar la calidad de la imagen. En general, se pueden identificar dos tipos de redundancia:

- **Redundancia espacial**: Se dan correlaciones entre los píxeles vecinos de una imagen. Por lo tanto, se puede utilizar el valor de un píxel para predecir el valor de sus vecinos.
- **Redundancia espectral**: Se dan correlaciones entre las bandas espectrales de la imagen. Esto es muy usado en los esquemas de compresión donde se hace uso de una etapa de análisis/síntesis, que realiza una transformación lineal basada en la transformada de Fourier, en la transformada de Wavelet o en sus derivados.

Es así como surgen dos métodos de compresión que manejan la redundancia en las imágenes [3]. Uno de estos métodos es llamado "**Lossless**" o Compresión sin pérdida de información, y se caracteriza porque la tasa de compresión está limitada por la entropía (redundancia de datos) de la señal original. Entre las técnicas que usan éste método, destacan las que hacen uso de métodos estadísticos, tales como: Codificación Huffman, Codificación Aritmética, Lempel-Ziv, entre otros.

El otro método es llamado "**Lossy**" o Compresión con pérdida de información y se caracteriza por lograr las tasas de compresión más elevadas, a costa de sufrir una degradación relativa en el proceso de reconstrucción de la imagen. Los sistemas de compresión que utilizan este método, por lo general cuentan con una etapa de transformación lineal. Entre ellos podemos destacar: El estándar JPEG, basado en la transformada discreta de coseno (DCT) y el estándar JPEG2000, basado en la transformada discreta de wavelet (DWT) [3].

1.3. ESQUEMA GENERAL DE UN SISTEMA DE COMPRESIÓN WAVELET DE IMÁGENES

Un sistema de compresión de imágenes por transformación lineal para aplicaciones no críticas se resume de la siguiente manera: Se aplica una transformación lineal a los datos de la imagen de entrada (dando como resultado un conjunto de coeficientes), para luego aplicar una codificación con pérdida sobre dichos coeficientes resultantes haciendo uso de un cuantizador. Y por último, se aplica una codificación sin pérdida a los valores cuantizados para elevar el rendimiento del sistema.

Un típico sistema de compresión Wavelet de imágenes es mostrado en la figura 1.4. El proceso de compresión está formado por tres bloques fundamentales, llamados: (a) Etapa de Análisis (b) Cuantizador, y (c) Codificador de entropía.

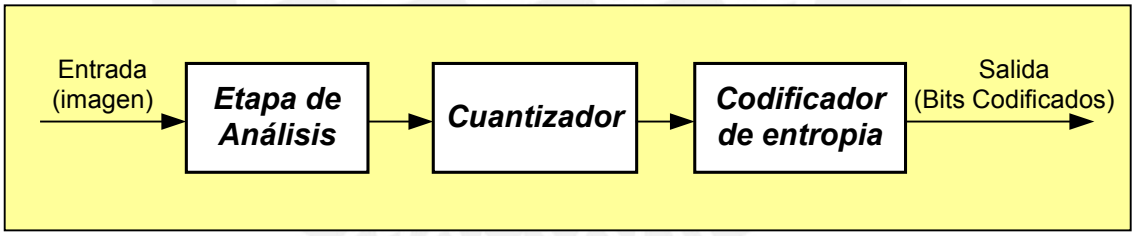


FIGURA 1.4.- SISTEMA DE COMPRESIÓN WAVELET DE IMÁGENES

(a) Etapa de Análisis:

Corresponde al procesamiento de la transformada discreta bidimensional de Wavelet (DWT-2D), para el cual se debe hacer una correcta elección de las Wavelets analizadoras. Este detalle juega un papel muy importante en los resultados finales. Las Wavelets ortogonales y biortogonales son las más eficientes y las más usadas para un posterior proceso de compresión [4].

(b) Cuantizador:

Su función principal es reducir el número de bits necesarios para almacenar los coeficientes transformados. Los sistemas de compresión con pérdida basados en el análisis Wavelet emplean Cuantización Escalar (SQ), siendo ésta técnica la base para la elaboración de los codificadores y decodificadores Wavelet [3].

(c) Codificador de Entropía:

Codifica los valores cuantizados con pérdida de información y es empleado para elevar el desempeño del sistema de compresión. Utiliza un modelo para determinar las probabilidades para cada valor cuantizado, produciendo un código apropiado basado en esas probabilidades, de modo que la trama codificada resultante final sea más pequeña que la trama que arroja el cuantizador. Los codificadores de entropía más usados son el codificador Huffman y el codificador Aritmético [3].

Para el proceso inverso (descompresión), se debe contar con una Etapa de Síntesis (transformación inversa) y una etapa de decodificación que sea simétrica con el proceso de codificación. El presente trabajo se centra en el desarrollo e implementación Hardware de un Cuantizador Escalar que permita realizar el proceso de codificación y decodificación sobre imágenes transformadas al dominio Wavelet.

2. ELECCIÓN DEL ALGORITMO DE CODIFICACIÓN A IMPLEMENTAR EN HARDWARE

2.1. INTRODUCCIÓN

Este capítulo presentará una breve descripción de las técnicas de codificación Wavelet más importantes en los últimos años, así como las consideraciones necesarias para la elección del algoritmo a implementar en Hardware. Finalmente se analizará a detalle el algoritmo escogido, y luego se planteará una versión alternativa del mismo para un procesamiento más eficiente.

2.2. TÉCNICAS DE CODIFICACIÓN WAVELET SOBRE IMÁGENES

A continuación se describen las características más importantes de tres algoritmos de codificación Wavelet, que fueron escogidos por su importancia y popularidad en el quehacer actual de la compresión de imágenes [3]. Son presentados por orden de antigüedad, para apreciar la evolución de uno con respecto a otro.

2.2.1. EMBEDDED ZEROTREE WAVELET

El algoritmo EZW (**E**mbedded **Z**erotree **W**avelet) fue introducido por Shapiro en 1993 [5]. Esta propuesta inicia la era de los algoritmos de codificación Wavelet con pérdida, donde la idea central, se fundamenta en el aprovechamiento de la organización espacial de las imágenes descompuestas al dominio Wavelet. Emplea una codificación por Mapa de significancias, el cual es generado en base a un valor umbral, dando como resultado 4 posibles etiquetas para representar a cada uno de los coeficientes: *significant positive (sp)*, *significant negative (sn)*, *zerotree root (zr) y isolated zero (iz).* Basándose en esta clasificación, es que el algoritmo codifica con

mayor prioridad a los coeficientes de mayor energía (bandas de baja frecuencia), ya que en ellos se encuentra la información más importante de la imagen a comprimir. Sus principales características son las siguientes:

- Permite una compresión progresiva de la imagen.
- Alta fidelidad de imagen, debido a la adición de bits de refinamiento.
- Es más eficiente que las técnicas basadas en la transformada discreta de cosenos (DCT).

2.2.2. SET PARTITIONING IN HIERARCHICAL TREES

El algoritmo SPIHT (**S**et **P**artitioning **I**n **H**ierarchical **T**rees) es una generalización del algoritmo EZW y fue propuesto por Amir Said y William Pearlman en 1996 [6]. Al igual que EZW, éste aprovecha la orientación espacial de los árboles de coeficientes Wavelet, pero los agrupa jerárquicamente en grandes sub-conjuntos. Esta nueva organización no solo proporciona una codificación por significancia de coeficiente, sino también por significancia de un grupo de ellos; siendo ésta última la diferencia y la mejora con respecto a EZW. Este tipo de codificación resulta tan robusta, que al incluirle una etapa de codificación sin perdida de información (Lossless), no se aprecia una gran variación en la relación Señal a Ruido del sistema de compresión. Además posee excepcionales características [7], tales como:

- Buena calidad de imagen con una elevada Relación Señal a Ruido Pico (PSNR).
- Rápida codificación y decodificación. Posee una alta simetría entre ambos.
- Habilidad de codificar para un Bit Rate o PSNR específico.

2.2.3. EBCOT

Este algoritmo denominado EBCOT (**E**mbedded **B**lock **C**oding with **O**ptimal **T**runcation), fue presentado por David Taubman en el año 2000 [8], siendo elegido como el Codec núcleo para el nuevo estándar JPEG2000. En contraste con el algoritmo SPIHT, éste reparte los coeficientes wavelets en bloques de 64 por 64 píxeles (Code Blocks) que luego son codificados independientemente uno del otro. Posteriormente se divide progresivamente cada Code Block en secuencias de sub-bloques de hasta 4x4, los cuales son analizados y procesados en base a cuatro operaciones llamadas: Zero Coding, Run-length Coding, Sign Coding y Magnitude Refinement. Por último, EBCOT organiza la trama de bits en capas en donde aplica un truncamiento eficiente para lograr una resolución y un Relación Señal a Ruido (SNR) escalable. Además de las características anteriormente mencionadas, posee las siguientes:

- Excelente calidad visual para reconstrucción de imágenes de muy bajos Bit rates.
- Codificación de Región de interés.
- Resistencia al error (en canales de ruido).

2.3. CONSIDERACIONES PARA LA ELECCIÓN DEL ALGORITMO

Según lo mostrado anteriormente se puede deducir qué algoritmo posee un mayor desempeño; siendo los más actuales, los más eficientes pero a la vez los más complicados de implementar en Hardware. En el caso de EZW y SPIHT, en [6] se

demuestra que SPIHT ofrece un mayor PSNR para bit-rates mayores a 0.2, lo cual implica un mayor nivel de compresión con la menor distorsión posible en la reconstrucción de la imagen.

El algoritmo EBCOT posee un mayor PSNR que SPIHT [8], pero las técnicas son bastante distintas una de la otra, ofreciendo SPIHT una mayor facilidad de implementación debido a la simetría que posee entre su etapa de codificación y decodificación. Además, se debe tener en cuenta que EBCOT fue desarrollado de una manera muy específica para encajar en el estándar JPEG2000, siendo este detalle crítico en la elección del algoritmo; ya que su implementación implica trabajar bajo las normas que establece el estándar y eso lo hace poco flexible.

Por lo tanto, se optó por la implementación hardware del algoritmo SPIHT porque rompe con la tendencia de complejidad de otros métodos, obteniendo resultados superiores utilizando métodos relativamente sencillos, como es una “Cuantización Escalar Uniforme”. La diferencia fundamental entre el algoritmo SPIHT optimizado para software y uno optimizado para hardware es que en el primero las instrucciones se ejecutan secuencialmente y son de propósito general, mientras que en el segundo caso el procesamiento puede darse en paralelo, a fin de ejecutar la mayor cantidad posible de operaciones al mismo tiempo, sin que haya dependencias entre ellas.

Previas implementaciones sobre arreglos de puertas programables por campo (FPGAs) han planteado modificaciones del algoritmo SPIHT como una alteración en el orden de procesamiento de los coeficientes de la imagen transformada con el fin

de incrementar el grado de paralelismo en la codificación [9]; o se han basado en un enfoque de particionamiento de imágenes [10,11] para minimizar la memoria necesaria. El presente trabajo propone una implementación sobre un FPGA con el propósito fundamental de reducir al máximo el tiempo de procesamiento del algoritmo de codificación SPIHT mediante la inserción de un algoritmo dedicado denominado: Generador de Mapas de Significancias (GMS) que permitirá evitar la redundancia en el procesamiento de datos del algoritmo. Esto fue introducido inicialmente en [12].

2.4. ANALISIS DEL ALGORITMO SPIHT

2.4.1. ESTRUCTURA DE DATOS USADA POR EL ALGORITMO SPIHT

El algoritmo SPIHT se encarga de procesar los datos provenientes de la descomposición Wavelet de una imagen, es decir los “coeficientes Wavelet”. Sin embargo, hace uso de la palabra “píxel” cuando se trata de ubicar un coeficiente dentro de la imagen transformada.

Los conjuntos o sets de coeficientes Wavelet se representan, en base a una estructura de datos llamada “Árboles de orientación espacial” [6]. Esta estructura se caracteriza por explotar las relaciones espaciales entre los coeficientes Wavelet de los diferentes niveles de la pirámide de sub-bandas. La figura 2.1 muestra cómo se define la pirámide. Las flechas señalan la dependencia espacial entre píxeles, y los bloques coloreados muestran a todos los descendientes para un píxel específico en cada nivel de descomposición.

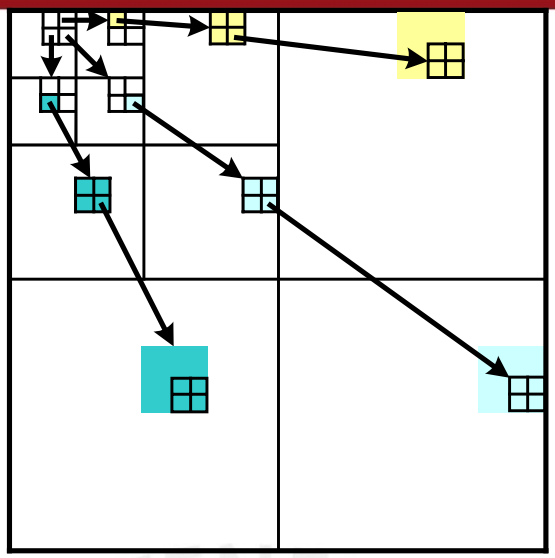


FIGURA 2.1.- ÁRBOLES DE ORIENTACIÓN ESPACIAL

Los descendientes directos de un píxel son llamados píxeles “hijos” y se componen de cuatro píxeles ubicados en la misma orientación espacial, dentro de una sub-banda ubicada en el siguiente nivel de descomposición Wavelet (ver figura 2.2). Además, se tiene que tomar en cuenta que los píxeles correspondientes a la escala más fina o de mayor resolución (altas frecuencias) de la imagen transformada son las hojas del árboles y éstos no tienen hijos.

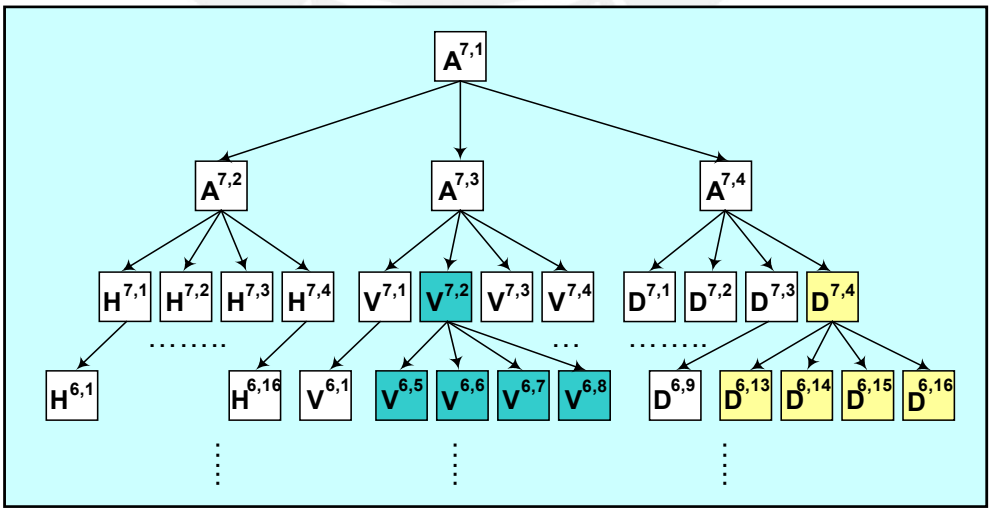


FIGURA 2.2.- COEFICIENTES WAVELET ORGANIZADOS EN ÁRBOLES JERÁRQUICOS

2.4.2 NOTACIÓN

Los árboles se clasifican en cuatro tipos de conjuntos, que almacenan las coordenadas de los coeficientes o las coordenadas de los sets de coeficientes Wavelet. Estos son:

- ***O***(*i*, *j*) : Es el set de coordenadas de los hijos inmediatos del coeficiente Wavelet "$C_{i,j}$". Este puede ser de tamaño cero o cuatro. El primer caso se dará cuando el coeficiente no tenga ningún descendiente y el segundo, en caso contrario.
- ***D***(*i*, *j*) : Es el set de coordenadas de los descendientes del coeficiente Wavelet "$C_{i,j}$" de tamaño cero o una suma de potencias de cuatro. Los descendientes son los hijos, nietos, y demás generaciones del coeficiente.
- ***H***(*i*, *j*) : Es el set de coordenadas de las raíces de todos los árboles de orientación espacial.
- ***L***(*i*, *j*) : Es el set de coordenadas que se obtiene de: ***D***(*i*, *j*) – ***O***(*i*, *j*). Contiene a todos descendientes sin considerar a los hijos inmediatos.

Además, se define como valor umbral *T* de los coeficientes Wavelet a un número potencia de dos menor al valor máximo de los coeficientes wavelet de una imagen transformada. Es decir: $T = 2^{[\mathrm{Log}_2(\max(i,j)\{|C_{i,j}|\})]}$

2.4.3. ALGORITMO DE CODIFICACIÓN

El algoritmo utiliza tres listas para clasificar los coeficientes Wavelet de una imagen transformada y estas son:

1. Lista de píxeles insignificantes (LIP): Contiene las coordenadas de aquellos coeficientes que son insignificantes con respecto al valor umbral *T*.
2. Lista de píxeles significantes (LSP): Contiene las coordenadas de aquellos coeficientes que son significantes con respecto al valor umbral *T*.
3. Lista de sets insignificantes (LIS): Contiene las coordenadas de las raíces de los sub-árboles insignificantes. Posee ya sea los sets $D(i, j)$ (también llamados sets de Tipo A) o los sets $L(i, j)$ (también llamados sets de Tipo B).

Antes de introducir el algoritmo, se deben conocer las reglas que se establecen sobre los sets. Y son las siguientes:

- Si el set $D(i, j)$ es significante entonces este es particionado en sets $L(i, j)$, que poseen los coeficientes $(k, l) \in O(i, j)$.
- Si el set $L(i, j)$ es significante entonces este es particionado en cuatro sets $D(k, l)$, con $(k, l) \in O(i, j)$.
- En caso los sets $D(i, j)$ o $L(i, j)$ sean insignificantes, estos son eliminados de la lista LIS y enviados a la lista LIP.

Además, una de las características más importantes aprovechadas por el algoritmo, es que el set $D(i, j)$ o $L(i,j)$ puede ser significante si al menos uno de sus coeficientes es significante. La notación usada para representar la significancia de un set es: $S_n(\text{Set}(i,j))$, donde $n = \text{Log}_2(T)$.

A continuación se muestra el algoritmo propuesto por Said y Pearlman [6], el cual involucra los siguientes procesos: Inicialización, Paso de Clasificación (Procesamiento de LIP y Procesamiento de LIS), Paso de Refinamiento (Procesamiento de LSP) y Paso de cuantización.

1. Inicialización:

1.1 Transmitir n = [$Log_2(max_{(i,j)}\{|C_{i,j}|\})$]. % $T = 2^n$ %

1.2 Inicializar LSP como una lista vacía.

1.3 Añadir a LIP las coordenadas de todas las raíces $(i, j) \in H$.

1.4 Añadir a LIS las coordenadas de las raíces $(i, j) \in H$ que tienen descendientes, como tipo A.

2. Paso de Clasificación:

2.1 Para cada entrada (i, j) en LIP hacer:

2.1.1 Enviar $S_n(i, j)$;

2.1.2 Si $S_n(i, j) = 1$, mover (i, j) a LSP. Enviar el signo de $C_{i,j}$;

2.2 Para cada entrada (i, j) en LIS hacer:

2.2.1 Si la entrada es de tipo D, entonces:

- Enviar $S_n(D(i, j))$;
- Si $S_n(D(i, j)) = 1$, entonces

  * Para cada $(k, l) \in O(i, j)$ hacer:

    · Enviar $S_n(k, l)$;

    · Si $S_n(k, l) = 1$, agregar (k, l) a LSP. Enviar el signo de $C_{k,l}$;

    · Si $S_n(k, l) = 0$, agregar (k, l) al final de LIP;

  * Si $L(i, j) \neq 0$, mover (i, j) al final de LIS como entrada de tipo L e ir al paso 2.2.2; Caso contrario, remover entrada (i, j) de LIS.

2.2.2 Si la entrada es de tipo L, entonces:

- Enviar $S_n(L(i, j))$;
- Si $S_n(L(i, j)) = 1$, entonces

  * Añadir cada $(k, l) \in O(i, j)$ al final de LIS como entrada tipo D;

  * Remover (i, j) de LIS;

3. **Paso de Refinamiento**:

   Para cada entrada (*i*, *j*) del LSP, excepto los que han sido incluidos en la última pasada de clasificación, enviar el *n-th* bit más significativo de $|C_{i,j}|$.

4. **Paso de Cuantización**:

   Decrementar "n" en uno (equivalente a dividir el umbral entre 2) e ir al paso 2 si es necesario.

Notas importantes para el desarrollo del algoritmo de decodificación SPIHT [6] :

- Los bits enviados a la trama, son usados por el decodificador para reconstruir los coeficientes wavelet $C'_{i,j}$, actualizándolos en cada iteración. El decodificador restablece el orden del procesamiento de las listas LIP, LSP y LIS, ya que éste trabaja de la misma forma que el codificador.

- En cada iteración, cuando las coordenadas (*i*, *j*) de un coeficiente $C_{i,j}$ son movidas a LSP como una entrada, se sabe que $2^n \leq |C_{i,j}| \leq 2^{n+1}$. Como resultado, el mejor valor que el decodificador puede dar al coeficiente $C'_{i,j}$ que está siendo reconstruido está entre 2^n y 2^{n+1}. Así, el decodificador debe poner $C'_{i,j} = \pm 1.5 \times 2^n$ (el signo de $C'_{i,j}$ es recibido como entrada por el decodificador justo después de la inserción). Durante la pasada de refinamiento, cuando el decodificador recibe como entrada el valor actual del *n*-th bit de $C'_{i,j}$, se mejora el valor 1.5×2^n sumándole 2^{n-1} (si el bit de entrada es 1) o restándole 2^{n-1}(si el bit de entrada es 0). De esta manera, el decodificador puede mejorar la apariencia de la imagen (o, equivalentemente, reducir la distorsión) durante las pasadas de clasificación y refinamiento.

2.5. MODIFICACIÓN EN EL PROCESAMIENTO DEL ALGORITMO SPIHT

2.5.1 INTRODUCCIÓN

El objetivo de esta sección, es modificar el procesamiento del algoritmo SPIHT para minimizar el volumen de accesos a la imagen transformada en el proceso de evaluación de significancias de los sets, permitiendo de ésta manera *una reducción del tiempo de procesamiento* con respecto a la propuesta original. Además trabajará con imágenes en el dominio Wavelet en descomposición completa donde el número de niveles de descomposición Wavelet es igual al número máximo de octavas menos uno.

2.5.2 MARCO TEÓRICO

El cálculo de significancia de un coeficiente wavelet "$C_{i,j}$" se define en base a una función S_n, que se encarga de comparar el valor absoluto del coeficiente con un determinado valor umbral *T*. Si este resulta mayor, se dice que es un coeficiente "significante" y se entrega el valor "1". En caso contrario, cuando el coeficiente es menor al valor umbral establecido, se dice que es "insignificante" y se arroja "0".

$$S_n(c_{i,j}) = \begin{cases} 1, \max(i,j) \in T \geq 2^n \\ 0, otro caso \end{cases}$$

Esta función básica es aplicada en el tratamiento de las listas LIP (*List of Insignificants Pixels*), y para ello tan solo basta hacer la lectura de la imagen transformada y hacer una comparación con el umbral establecido. Por lo tanto, para

el calculo de la significancia de coeficientes individuales, no es necesario implementar algún algoritmo dedicado, ya que se trata de una operación simple. Sin embargo, para el análisis de las listas LIS (*List of Insignificants Sets*) esto resulta más complicado ya que se trata de analizar la significancia de los sets (tipo D ó tipo L), lo cual implica analizar a todos los descendientes que estos posean. En este procedimiento, se accede redundantemente a algunos píxeles, ya que los sets con mayor descendencia comparten sus píxeles con los sets de menor descendencia.

2.5.3. GENERACIÓN DE DIRECCIONES DE DESCENDIENTES Y CÁLCULO DE SIGNIFICANCIAS DE SETS

En esta sección se propone un algoritmo que se encargará de generar las significancias de los sets (agrupadas en dos mapas de significancias), a partir del acceso a las direcciones de los descendientes de cada set y a su evaluación con respecto a un valor umbral.

En principio, se llegó a observar de que existe una relación espacial entre el píxel raíz y sus descendientes (offprings), y es que: Cualquiera que sea el píxel raíz, éste siempre proporcionara un grupo de 12 píxeles “Hijos” agrupados en tres cuartetos, donde cada cuarteto proviene de un píxel “Padre”. Además, las coordenadas de los cuatro hijos inmediatos de un píxel “Padre” cuya posición es (i, j) son ($2i$, $2j$), ($2i$+1, $2j$), ($2i$, $2j$+1), y ($2i$+1, $2j$+1). La figura 2.3 muestra la relación espacial “Raíz-Padre-Hijos” que existe entre píxeles y los niveles de descomposición que posee una imagen transformada de 16x16.

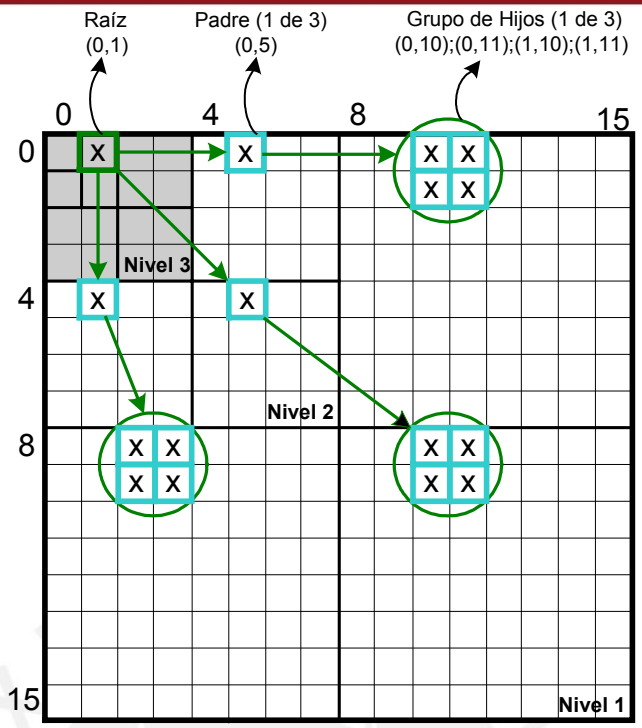


FIGURA 2.3.- RELACIÓN ESPACIAL "RAÍZ-PADRE-HIJOS" ENTRE PÍXELES

A) Generación del Mapa de significancias de los sets de tipo D:

Se sabe que un set ubicado en un nivel superior contiene a los coeficientes de los sets de niveles inferiores, por lo tanto, el cálculo de la significancia de los sets en los niveles más altos se realizará en base a la significancia de los sets de los niveles más bajos, evitando así, el análisis redundante de los descendientes. Por lo tanto, el algoritmo propuesto sólo necesita calcular la significancia de los píxeles "Hijos" inmediatos de un píxel "Padre" (que representa al set a analizar) y leer la significancia de los sets que representan cada uno de esos cuatro hijos. Es decir, se hacen cuatro comparaciones con el umbral y cuatro lecturas de mapa de significancias tipo D, para determinar la significancia final de un set. Según lo descrito anteriormente, para generar el mapa de significancias de los sets de tipo D de una imagen de 16x16 píxeles, se realizaran las siguientes tres etapas de procesamiento:

Etapa 1: Análisis de la significancia de los sets representados por los píxeles "Padres", ubicados en el nivel 2. La figura 2.4 muestra la generación de direcciones de los píxeles "Padres" y píxeles "Hijos" correspondientes a la Etapa 1.

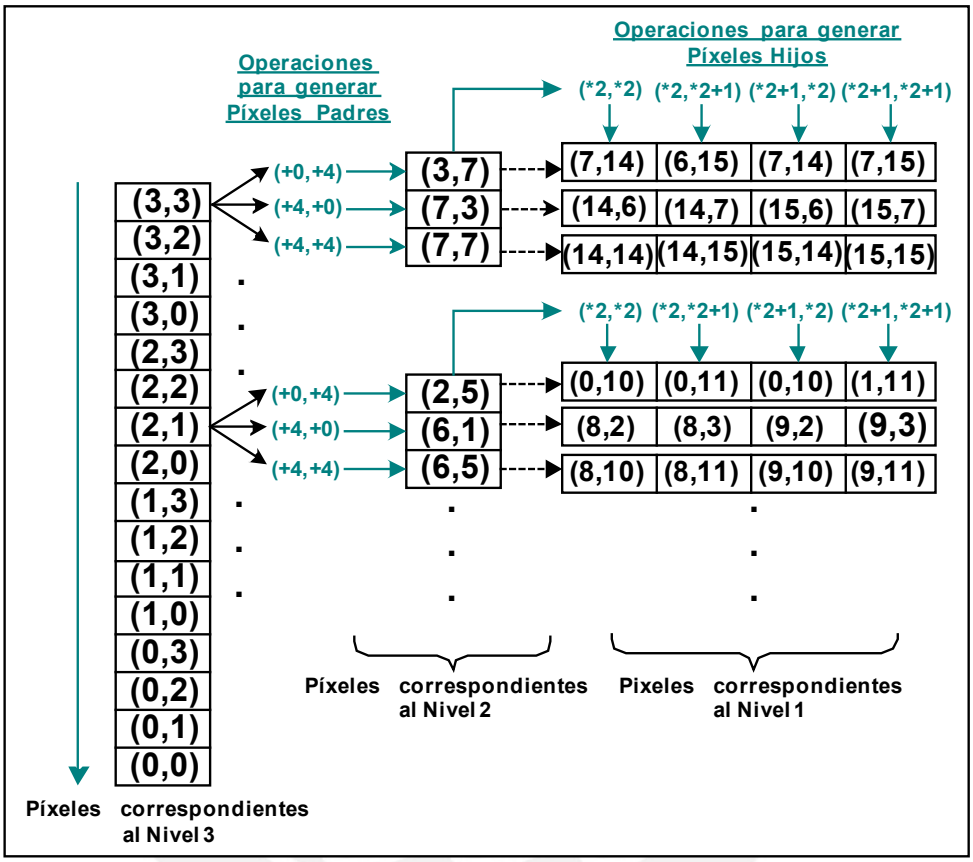


FIGURA 2.4.- GENERACIÓN DE DIRECCIONES (ETAPA 1)

Para el análisis de significancias, se toman los píxeles correspondientes al nivel 1, los cuales son comparados con el valor umbral. En función a las significancias generadas por estos grupos de cuatro píxeles, se determina la significancia del píxel "Padre", la cual se escribe en el mapa de significancias tipo D.

Etapa 2: Análisis de la significancia de los sets representados por los píxeles "Padres", ubicados en el nivel 3. La figura 2.5 muestra la generación de direcciones de los píxeles "Padres" y píxeles "Hijos" correspondientes a la Etapa 2.

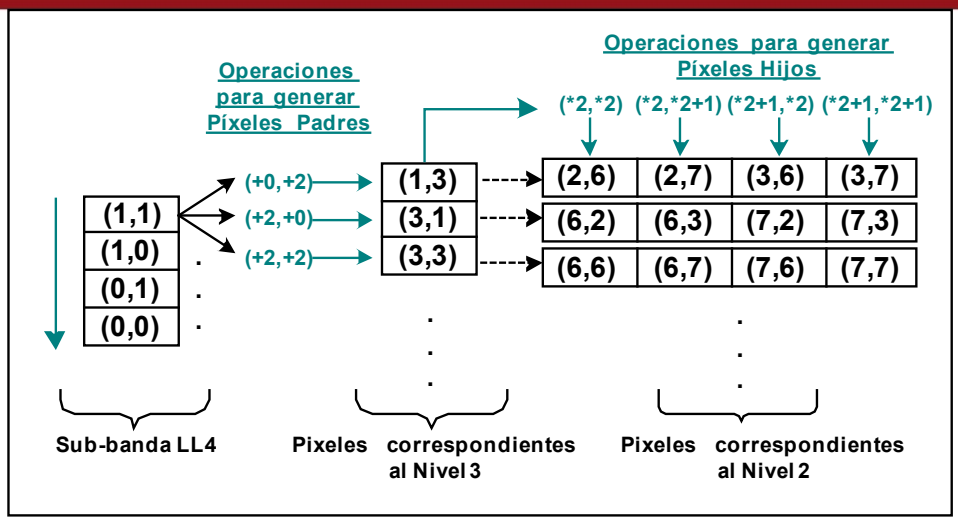


FIGURA 2.5. GENERACIÓN DE DIRECCIONES (ETAPA 2)

La significancia de los sets ubicados en el nivel 3 dependen de la significancia de cada uno de los hijos y de las significancias de los sets que representan cada uno de los píxeles "Hijos", las cuales fueron almacenadas en el mapa de significancias en la etapa anterior.

Etapa 3: Análisis de la significancia de los sets representados por los píxeles "Padres", ubicados de la Sub-banda LL4. La figura 2.6 muestra la generación de direcciones de los píxeles "Padres" y píxeles "Hijos" correspondientes a la Etapa 3.

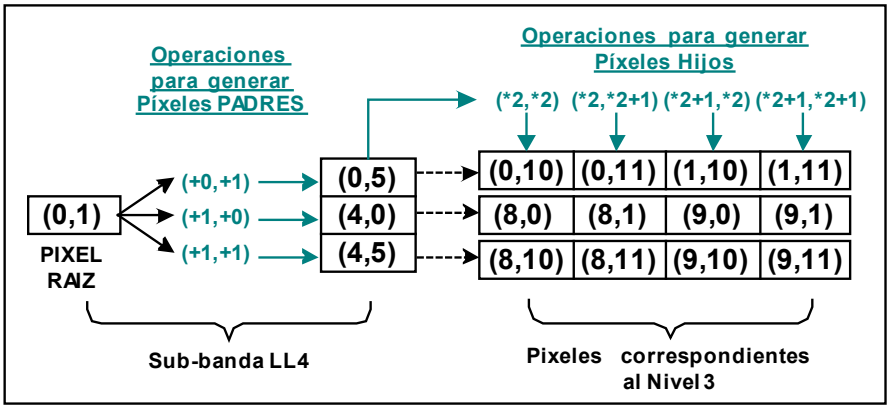


FIGURA 2.6.- GENERACIÓN DE DIRECCIONES (ETAPA 3)

En general, se realizan tantas etapas de generación de descendientes como número de niveles se tenga en la imagen transformada. Para el ejemplo se realizan “Tres Etapas de generación de descendientes” ya que posee 3 niveles de descomposición o 4 octavas; de tal forma que en la primera etapa se generan todos los píxeles que se ubican en el nivel 1, la segunda etapa genera los píxeles correspondientes al nivel 2 y por último la tercera etapa que trabaja con la última octava (correspondiente al píxel (0,0)) genera los píxeles que se ubican en el nivel 3.

Es importante destacar que no se analiza la significancia de los sets del primer nivel, ya que, dado que los píxeles de este nivel no tienen hijos, tampoco pueden representar sets. Por lo tanto, el tamaño del mapa de significancias generado para almacenar los sets de tipo D, será siempre la cuarta parte del tamaño de la imagen transfomada.

B) Generación del Mapa de significancias de los Sets de tipo L:

Está basado en el mapa de significancias de los sets de tipo D. Para el cálculo de la significancia de los sets de tipo L se tomarán en cuenta la significancias de los descendientes del píxel “Padre” sin contar los hijos inmediatos del mismo. Para ello, sólo se leerá la significancia de los sets de tipo D representados por los píxeles “Hijos”, sin analizar la significancia de los mismos. El tamaño del mapa de significancias generado para almacenar los sets de tipo L, será siempre la dieciseisava parte del tamaño de la imagen transformada.

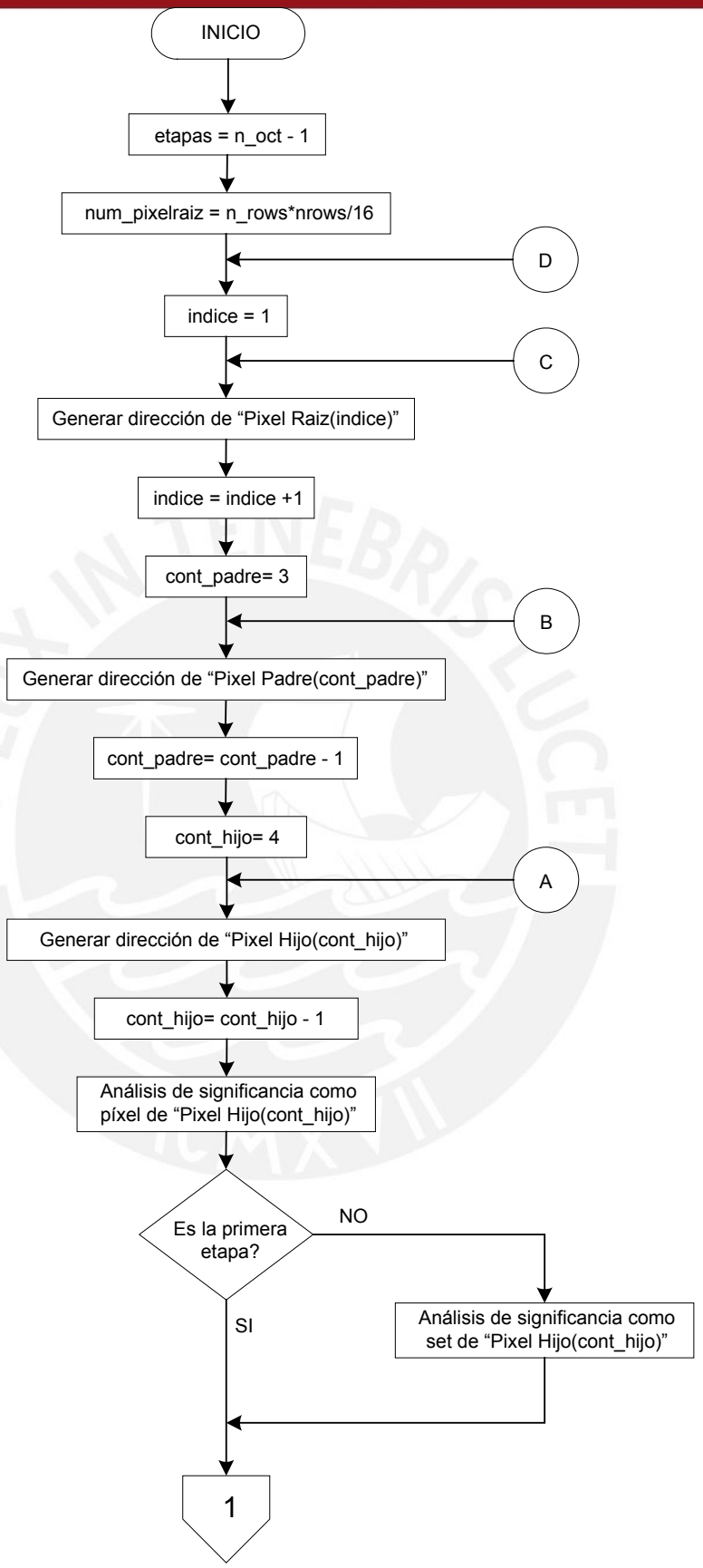
Por lo tanto, el presente trabajo propone un algoritmo denominado Generador de Mapas de Significancias (GMS) que realizará todo lo anteriormente expuesto. La figura 2.7 muestra el diagrama de flujo del algoritmo GMS:

Entradas:

- n_rows : Indica el número de filas o columnas. Para este caso son iguales ya que se trabajan con imágenes cuadradas.
- n_oct : Indica el número de octavas a procesar. El número de octavas es la cantidad de niveles Wavelet que posee la imagen transformada, para las pruebas se utiliza una descomposición Wavelet completa.
- T: Valor umbral de la imagen transformada.

Salidas:

- Mapa D / Mapa L: Son los mapas de significancias generados y que serán usado por el codificador SPIHT .



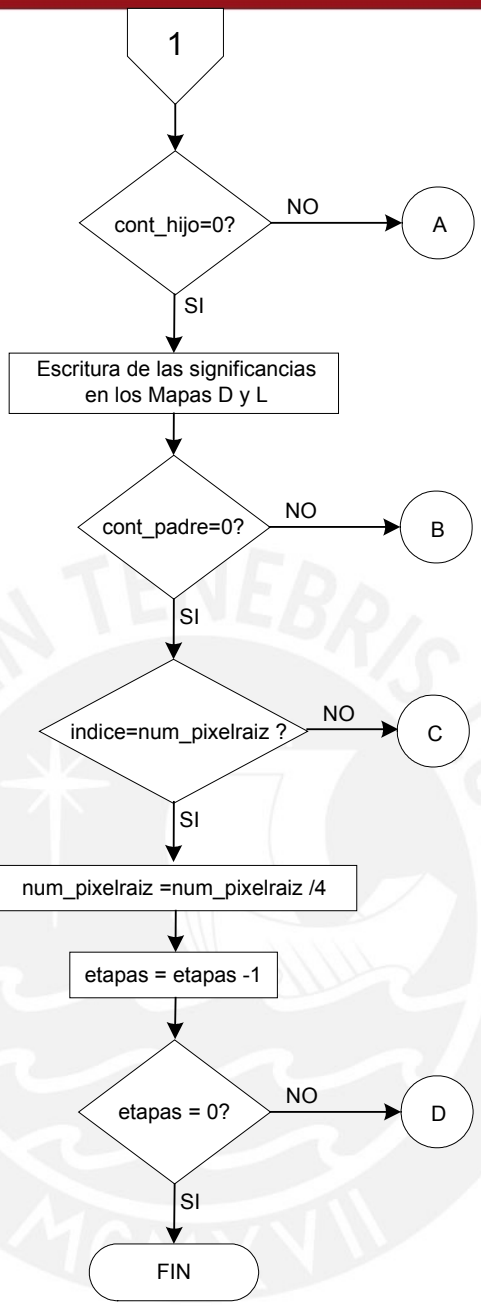


FIGURA 2.6.- DIAGRAMA DE FLUJO DEL ALGORITMO GMS

En base a éste algoritmo, se diseñará un núcleo de procesamiento llamado procesador GMS, el cual trabajará conjuntamente con otro núcleo de procesamiento llamado Procesador CODEC que realizará la Codificación/Decodificación SPIHT. Ambos serán implementados sobre un arreglo de puertas programables por campo (FPGA) e interactuarán con distintos bloques de memoria. La distribución de los módulos participantes se muestra en el diagrama de bloques de la figura 2.7.

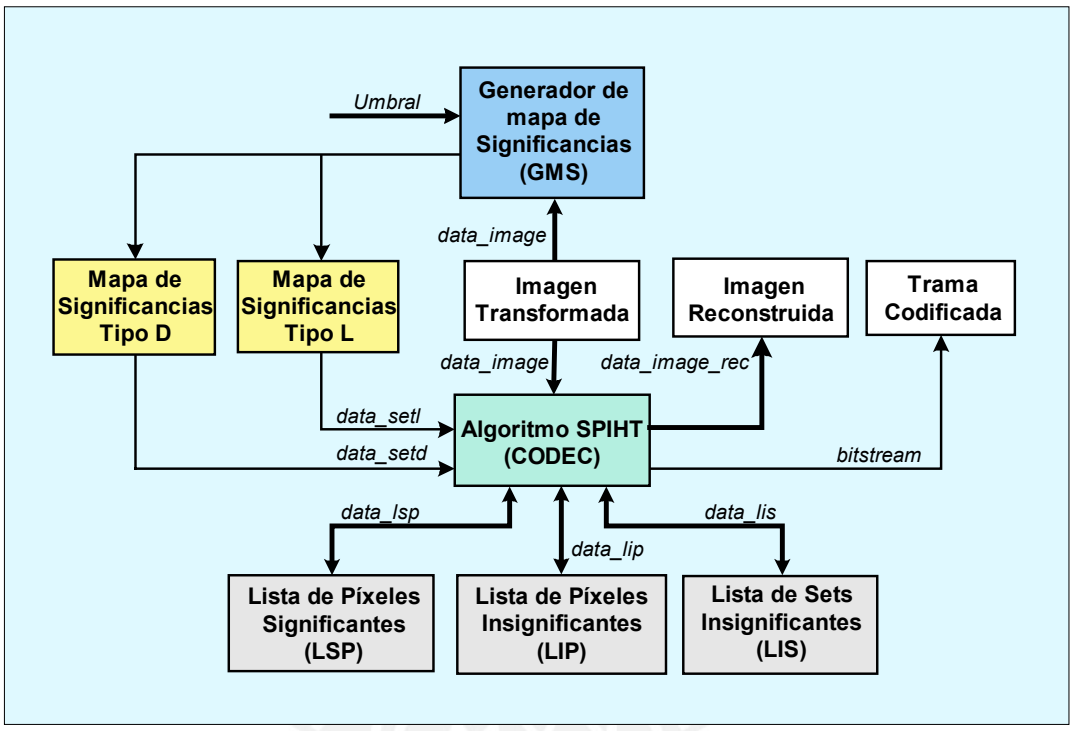


FIGURA 2.7.- DIAGRAMA DE BLOQUES DEL SISTEMA A IMPLEMENTAR EN HARDWARE

3. METODOLOGÍA DE DISEÑO TOP-DOWN APLICADA AL ALGORITMO ESCOGIDO

3.1. METODOLOGÍA DE DISEÑO

3.1.1 INTRODUCCION

La metodología propuesta en el presente capítulo, corresponde a un diseño del tipo Top-Down para la implementación de algoritmos de Procesamiento Digital de Señales sobre una plataforma hardware. Esto significa, empezar por una descripción totalmente algorítmica del diseño a realizar para luego descender progresivamente a niveles de descripción más físicos. Este desarrollo es usado sobre el algoritmo GMS y sobre el algoritmo SPIHT (escogido en el capitulo anterior). La figura 3.1. muestra el flujo de diseño de la metodología Top-Down.

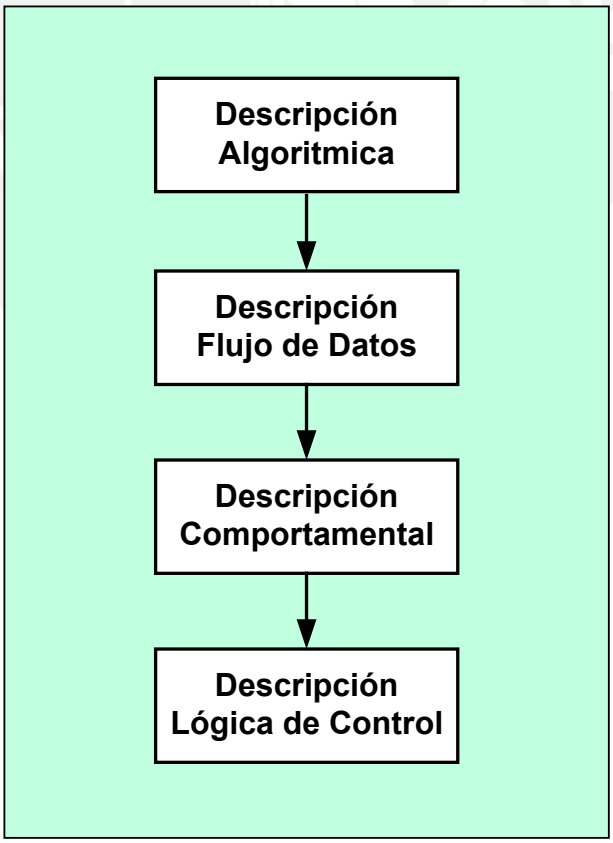


FIGURA 3.1. FLUJO DE DISEÑO DE LA METODOLOGÍA TOP- DOWN

A continuación se describe cada una de las secciones correspondientes al flujo de diseño. Posteriormente se mostrará como se aplica la metodología a modo de ejemplo, para la elaboración del procesador GMS propuesto en el capitulo anterior.

1) DESCRIPCIÓN ALGORITMICA:

Involucra la mención de las secuencias de operaciones a realizar con los datos. Esta descripción no contempla los tiempos de ejecución de las operaciones, ni la concurrencia a aplicar en las mismas.

2) DESCRIPCIÓN FLUJO DE DATOS:

Se obtiene el conjunto de operaciones a realizar con los datos en el dominio del tiempo, considerando las latencias involucradas en cada operación. Todas las consideraciones se toman en cuenta en base a una unidad de tiempo que determina el tiempo de procesamiento neto de cada dato o conjunto de datos. La descripción flujo de datos es independiente del grado de paralelismo aplicado en la arquitectura final. Para llevar a cabo dicha descripción con éxito, es necesario contar con el comportamiento de los bloques conformantes en el dominio del tiempo.

3) DESCRIPCIÓN COMPORTAMENTAL:

Trata con la especificación detallada del procesamiento de los datos, considerando los casos de concurrencia planteados en el diseño de la arquitectura. La unidad de tiempo básica en este caso es el ciclo de reloj y como resultado se debe obtener la arquitectura básica del diseño.

4) DESCRIPCIÓN LOGICA DE CONTROL:

Hace uso del algoritmo creado en la primera descripción y de las señales de control de la arquitectura generada en la descripción comportamental. Esta sección plantea el diseño de la lógica de control haciendo uso de una máquina de estados o un grupo de ellas jerarquizadas en el caso de un control complejo.

Los cuatro niveles de descripción serán aplicados en el presente trabajo para la implementación de los núcleos de procesamiento de función específica que ejecutaran la versión modificada del algoritmo SPIHT. Estos núcleos van a contener una arquitectura (camino de datos) y una lógica de control basada en máquinas de estados.

3.2 DEMOSTRACION DE LA METODOLOGÍA TOP-DOWN PARA EL DISEÑO DEL PROCESADOR GMS

En esta sección se muestra a modo de ejemplo la aplicación de la metodología sobre el procesador GMS. Además, se expondrá en cada una de las descripciones los detalles que fueron usados para la concepción de este procesador y de los demás bloques circuitales.

1) Descripción algorítmica: En base al diagrama de flujo mostrado en la sección 2.5.3. del capitulo 2 del presente trabajo, se debe describir el algoritmo GMS en forma detallada a modo de pseudo-código:

```
Inicio

% Inicialización de variables
rows_p ← n_rows/4;
cols_p ← n_rows/4;
father_value ←rows_p;
signif_set ← 0;
stages ← n_oct –1;
Threshold ← T;

Mientras stages ≥ 0 Hacer
    addr_rows_roots ← rows_p – 1;
    Mientras addr_rows_roots ≥ 0 Hacer
        addr_cols_roots ← cols_p – 1;
        Mientras addr_cols_roots ≥ 0 Hacer
            cont_father ← 3;
            Mientras cont_father ≥ 0 Hacer
                smD_data_wr ← 0;
                smL_data_wr ← 0;

                %  Generación de direcciones de los pixels “Padres”
                Caso (cont_father)
                  1:
                     addr_rows_father ← addr_rows_roots;
                     addr_cols_father  ← addr_cols_roots + father_value;

                  2:
                     addr_rows_father ← addr_rows_roots + father_value;
                     addr_cols_father  ← addr_cols_roots;

                  3:
                     addr_rows_father ← addr_rows_roots + father_value;
                     addr_cols_father  ← addr_cols_roots + father_value;
                 Fin Caso

                addr_father ← addr_rows_father & addr_cols_father;
                addr_rows_son_pre ← addr_rows_father x 2;
                addr_rows_son_pre ← addr_rows_father x 2;
                cont_son ← 3;

                  %  Generación de direcciones de los pixels “Hijos”
                  Mientras cont_son ≥ 0 Hacer
                      addr_rows_sons ← addr_rows_son_pre + cont_son(0);
                      addr_cols_sons ← addr_cols_son_pre + cont_son(1);
                      addr_sons ← addr_rows_sons & addr_cols_sons;
                      reg_temp ← Leer Imagen_t (addr_sons);

                          % Análisis de significancia como píxel de cada píxel “Hijo”
                          Si reg_temp ≥ Threshold Entonces
                               SmD_data_wr ← 1;
                          Fin si
```

```
                    % Análisis de significancia como Set de cada píxel "Hijo"
                    Si stages ≠ n_oct –1 Entonces
                        smD_data_rd ← Leer Mapa D (addr_sons);
                        Si smD_data_rd = '1' Entonces
                            SmD_data_wr ← 1;
                            SmL_data_wr ← 1;
                        Fin si
                    Fin si

                cont_son ← cont_son – 1;
                Fin Mientras cont_son

                % Escritura de las significancias en los Mapas D y L
                Escribir Mapa D (addr_father) ← smD_data_wr;

                Si stages ≠ n_oct –1 Entonces
                    Escribir  Mapa L (addr_father) ← smL_data_wr;
                Fin si

            cont_father ← cont_father – 1;
            Fin Mientras cont_father

        addr_cols_roots ← adr_cols_roots – 1;
        Fin Mientras addr_cols_roots

    addr_rows_roots ← addr_rows_roots – 1;
    Fin Mientras addr_rows_roots

% Reajuste de parámetros para la siguiente etapa de procesamiento
father_value ← father_value / 2;
rows_p ← rows_p/2;
cols_p ← cols_p /2;
stages ← stages – 1;
Fin Mientras stages

Fin
```

2) Descripción Flujo de Datos: En esta sección se debe descomponer el algoritmo en tareas o procesos, los cuales se caracterizan por realizar una operación determinada con lo datos. Por ejemplo, la escritura de significancia en el Mapa D corresponde al proceso: "Escribir Mapa D(addr_father) ← smD_data_wr" y la lectura de la imagen transformada corresponde al proceso: "reg_temp ← Leer Imagen_t (addr_sons)". A continuación son listados todos los procesos involucrados en el algoritmo GMS.

LISTA DE PROCESOS:

Proceso 1: rows_p ← n_rows/4

Proceso 2: father_value ← n_rows/4

Proceso 3: stages ← n_oct - 1

Proceso 4: Threshold ← T

Proceso 5: addr_rows_roots ← rows_p - 1

Proceso 6: addr_cols_roots ← cols_p - 1

Proceso 7: cont_father ← 3

Proceso 8: smd_data_wr ← 0

Proceso 9: sml_data_wr ← 0

Proceso 10: addr_rows_father ← addr_rows_roots

Proceso 11: addr_cols_father ← addr_cols_roots + father_value

Proceso 12: addr_rows_father ← addr_rows_roots + father_value

Proceso 13: addr_cols_father ← addr_cols_roots

Proceso 14: addr_father ← addr_rows_father & addr_cols_father

Proceso 15: addr_rows_son_pre ← addr_rows_father x 2

Proceso 16: addr_cols_son_pre ← addr_cols_father x 2

Proceso 17: cont ← 3

Proceso 18: addr_rows_sons ← addr_rows_son_pre + cont_son(0)

Proceso 19: addr_cols_sons ← addr_cols_son_pre + cont_son(1)

Proceso 20: addr_sons ← addr_rows_sons & addr_cols_sons

Proceso 21: reg_temp ← Leer Imagen Transformada (addr_sons)

Proceso 22: smd_data_wr ← 1

Proceso 23: sml_data_wr ← 1

Proceso 24: smd_data_rd ← Leer Mapa D (addr_sons)

Proceso 25: cont_son ← cont_son - 1

Proceso 26: Escribir Mapa D (addr_fathers) ← smd_data_wr

Proceso 27: Escribir Mapa L (addr_fathers) ← sml_data_wr

Proceso 28: cont_father ← cont_father - 1

Proceso 29: addr_cols_roots ← addr_cols_roots - 1

Proceso 30: addr_rows_roots ← addr_rows_roots - 1

Proceso 31: father_value ← father_value / 2

Proceso 32: rows_p ← rows_p / 2

Proceso 33: stages ← stages - 1

La primera aproximación a la descripción flujo de datos consiste en la disposición de los procesos involucrados en un diagrama de tiempos. Dicho diagrama debe mostrar la ejecución de los procesos según la secuencia de los mismos y la dependencia de datos existente. En la figura 3.2 el diagrama de procesos muestra la generación de direcciones de 4 píxeles Hijos, sus comparaciones con el valor umbral y la escritura de la signifiancia de éste grupo de píxeles en el Mapa D.

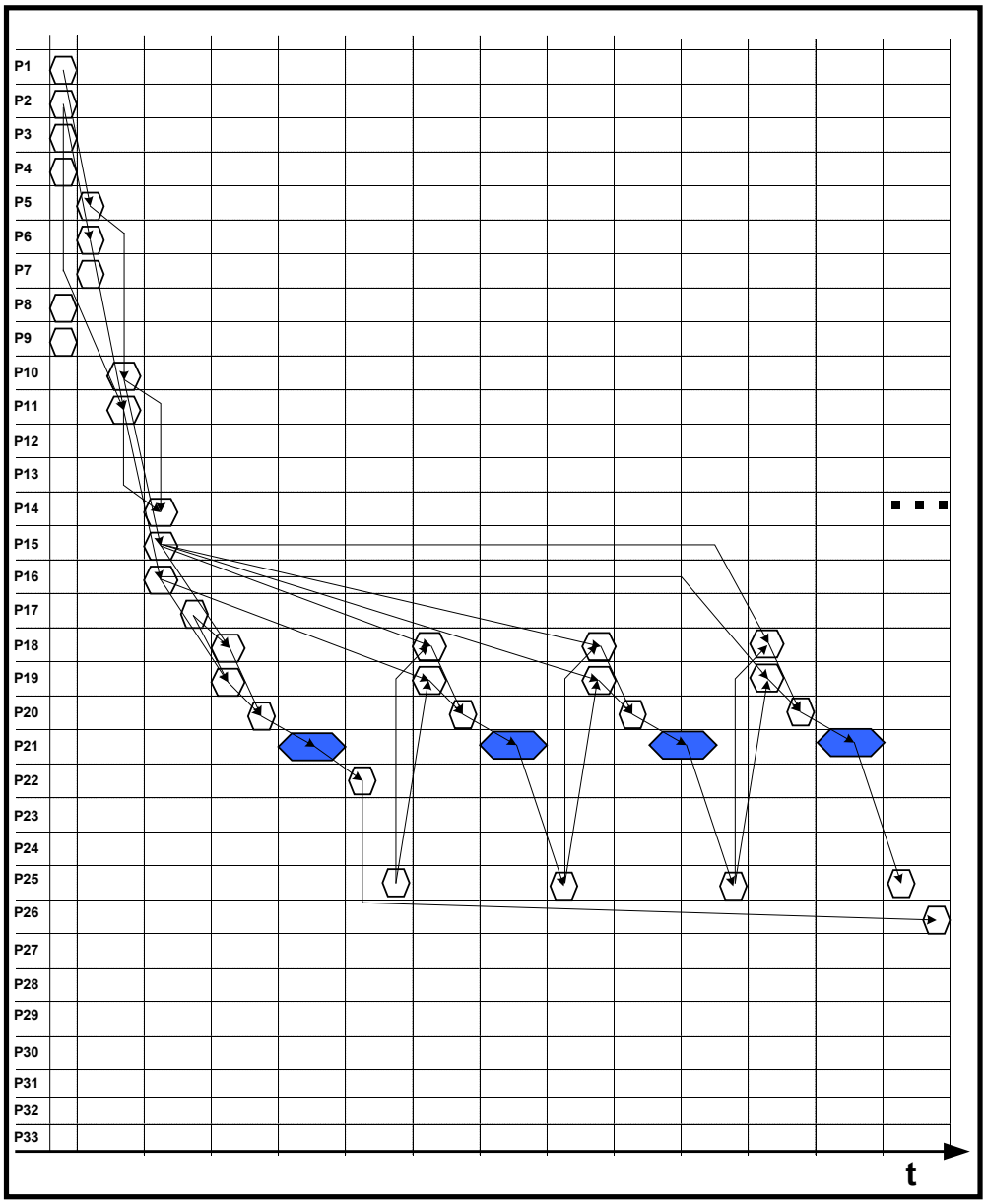


FIGURA 3.2.- DIAGRAMA DE PROCESOS PARA EL PROCESADOR GMS

3) Descripción Comportamental: En esta sección se deben enumerar los bloques ejecutores (recursos hardware) en base a los procesos mencionados y a la dependencia que existe entre procesos. Además es necesario exponer los siguientes conceptos:

- Dependencia de datos: Existe dependencia de datos entre dos procesos cuando uno de éstos tiene como operandos uno o más resultados provenientes de otro. En el diagrama de procesos, el proceso dependiente puede estar ubicado en el mismo intervalo de tiempo que ocupa otro proceso, o en uno posterior, pero no en uno anterior.

- Compartición de recursos: Existe compartición de recursos entre dos procesos cuando éstos emplean dentro de alguna fase de los procesamientos a efectuar, un mismo bloque de hardware. En el diagrama de procesos se observa que dichos procesos no pueden estar ubicados en el mismo intervalo de tiempo.

En base a lo anteriormente descrito, consideremos entonces los siguientes recursos con los que dispondrá el Procesador GMS:

- 4 registros de desplazamiento [SR1...SR4]: Para la división por dos de rows_p y father_value, y para la multiplicación por dos utilizada en la generación de los píxeles “Hijos”.
- 5 contadores [CONT1...CONT5]: Para los bucles que posee el algoritmo.

- 6 registros [REG1...REG6]: Para el almacenamiento de resultados temporales tales como operaciones aritméticas o lecturas de memoria.
- 2 flip-flops [FFD1, FFD2]: Son banderas usadas para la escritura de datos en los mapas de significancias tipo D y L.
- 2 sumadores [SUM1,SUM2]: Para la generación de los píxeles “Padres”.
- 2 bloques de Memoria RAM [RAM1,RAM2]: Para el almacenamiento de los dos mapas de significancia (Tipo D y Tipo L)
- Una RAM externa [DDR1]: Para el acceso a la imagen transformada.

Con estas consideraciones y la información acumulada, se elabora una Tabla de compartición de Recursos (Tabla 1), la cual es la base para la elaboración de la arquitectura o el camino de datos.

Procesos	SR1	SR2	SR3	SR4	CONT1	CONT2	CONT3	CONT4	CONT5	REG1	REG2	REG3	REG4	REG5	REG6	FFD1	FFD2	SUM1	SUM2	RAM1	RAM2	DDR1
1	X																					
2		X																				
3					X																	
4										X												
5						X																
6							X															
7								X														
8																X						
9																	X					
10											X											
11												X						X				
12											X								X			
13												X										
14											X	X										
15			X																			
16				X																		
17									X													
18													X									
19														X								
20													X	X								
21															X							X
22																X						
23																	X					
24																				X		
25									X													
26																				X		
27																					X	
28								X														
29							X															
30						X																
31		X																				
32	X																					
33					X																	

Tabla 1. Tabla de Compartición de Recursos del Procesador GMS

Por ejemplo, en la tabla anterior se muestra que existe compartición de recursos entre los procesos 24 y 26 (comparten la RAM1); asimismo, los procesos 3 y 33, 5 y 30, 6 y 29, 7 y 28 (comparten contadores). Es decir, a través de esta tabla se puede establecer la interconexión entre los recursos hardware anteriormente mencionados, estableciendo de esta manera una arquitectura inicial; a partir de la cual el diseñador realizará las optimizaciones y mejoras pertinentes. La interconexión no es mostrada en este capitulo, ya que se deja al diseñador la vía libre para la interconexión, optimización y mejoras que pueda sufrir la arquitectura. Es en el siguiente capitulo, donde se presentan todos los detalles correspondientes a la arquitectura final.

4) Descripción Lógica de Control : Según el algoritmo se puede apreciar que hay dos tareas a realizar y son: Carga del valor umbral y generación de mapas de significancias. Por lo tanto se necesitan cuatro estados: el primero espera una señal de habilitación para el inicio del procesamiento, el segundo genera un pulso necesario para la carga del valor umbral en el registro REG1, el tercero es un estado que se mantiene activo mientras se generen los mapas de significancia y el cuarto es un estado de fin de procesamiento.

4. DISEÑO DE LA ARQUITECTURA DE HARDWARE

4.1 INTRODUCCIÓN

La metodología de diseño Top-Down ha sido aplicada sobre los dos núcleos de procesamiento que gobiernan al sistema, es decir, sobre el procesador GMS y sobre el procesador CODEC (introducidos en la sección 2.5 del capítulo 2); ya que ambos están relacionados directamente con un algoritmo especifico. Además, se desarrolló un módulo denominado árbitro, el cual posee el control de mayor nivel del sistema y se basa en una simple máquina de estados. Sin embargo, todos los bloques fueron re-diseñados y optimizados para que funcionen de manera correcta y eficiente sobre un FPGA, utilizando las siguientes consideraciones:

- El diseño debe trabajar con imágenes transformadas al dominio Wavelet en descomposición completa No-estándar (número máximo de octavas), cuyas dimensiones sean potencia de dos y simétricas (imágenes cuadradas), de hasta 512x512 píxeles.
- El presente trabajo no aporta una mejora algorítmica, lo cual implica, que no es necesaria una elevada precisión. Por esa razón, el formato de los coeficientes Wavelet que utilizará la arquitectura del sistema será de punto fijo (truncamiento de decimales) y en complemento a dos. Además, según el análisis realizado en [9], una asignación de 16 bits es suficiente para poder representar la parte entera de cada coeficiente en cualquier nivel de transformación. Por lo tanto, los resultados de compilación deben ser obtenidos fijando el parámetro de ancho de bus a 16 bits.
- La memoria externa DDR-SDRAM y los diseños de propiedad intelectual de Altera que serán usados para la implementación del sistema trabajan a una

frecuencia de 133 MHz. Lo cual implica que todos los bloques diseñados (principalmente los procesadores) deben funcionar a una frecuencia mayor o igual a 133MHz.

Los diseños expuestos en el presente capítulo cumplen con las consideraciones anteriormente mencionadas. A continuación se describirá el comportamiento funcional, desarrollo, resultados independientes y consideraciones adicionales para el Procesador GMS, el Procesador CODEC y el árbitro del sistema.

Todos los bloques fueron descritos utilizando VHDL. Fueron compilados y simulados bajo el entorno Quartus II 2.2, fijando el dispositivo FPGA STRATIX EP1S25F1020C5 de Altera para el análisis de los resultados; ya que las pruebas experimentales se realizarán sobre una tarjeta de desarrollo basada en este FPGA. El parámetro data_width que representa el ancho del bus de datos de los coeficientes de la imagen, se fijó en 16 tomando en cuenta las consideraciones mencionadas líneas arriba.

4.2 PROCESADOR GMS

4.2.1 COMPORTAMIENTO FUNCIONAL

Es la unidad de procesamiento que se encarga de realizar la Generación de Mapas de Significancias a partir de los coeficientes Wavelet almacenados en la memoria externa. Los mapas generados por este procesador son almacenados en la memoria interna del FPGA (Mapas D y L), para luego ser utilizados por el Procesador CODEC. La arquitectura de esta unidad de procesamiento está fundamentada en lo

presentado en el capítulo 3 (ejemplo de la metodología Top-down), y ha sido re-diseñada tanto en su camino de datos como en su unidad de control, ya que tiene que adaptarse a los requerimientos temporales que impone el controlador de memoria externa DDR-SDRAM que será usado para la implementación.

4.2.2 DESARROLLO

Este procesador posee 4 módulos que trabajan en conjunto, y cada de uno de ellos es completamente parametrizable. El módulo encargado de generar las direcciones de los píxeles “raíces” está descrito en PROC_GMS_GENROOTS.VHD (ver sección Anexos A-1), y cuenta con un banco de registros y contadores para realizar dicha función. Además, posee circuitos de sincronización, debido a que este módulo interactúa con las señales provenientes del controlador de memoria externa.

La unidad aritmético-lógica está encargada de la generación de las direcciones de los píxeles “Padres” y de los píxeles “Hijos” y está descrita en PROC_GMS_ALU.VHD (ver sección Anexos A-1). Para cumplir dichas funciones, utiliza operaciones básicas de suma y multiplicación por dos. El módulo que realiza el análisis de significancias se encuentra descrito en PROC_GMS_ANALYSIS.VHD (ver sección Anexos A-1), el cual, se encarga de hallar la significancia como píxel y la significancia como set. Para hallar la significancia como píxel posee un comparador que compara el valor umbral con el valor absoluto del coeficiente Wavelet leído de la memoria externa. Para hallar la significancia como set, lee las significancias antes analizadas y almacenadas en el Mapa D. Finalmente, la unidad de control del procesador GMS se encuentra descrita en PROC_GMS_CONTROL.VHD (ver sección Anexos A-1), y se basa en una máquina

de estados como la mostrada en la figura 4.1, que sigue el comportamiento algorítmico correspondiente al pseudo-código del procesador GMS descrito en la sección 3.2. del capítulo 3. Todos los módulos se encuentran interconectados en la descripción PROC_GMS_TOP.VHD (ver sección Anexos A-1). La tabla 4.1. muestra los parámetros, señales de entrada y señales de salida del Procesador GMS, el cual se aprecia en el plano 1-1 de la sección Planos.

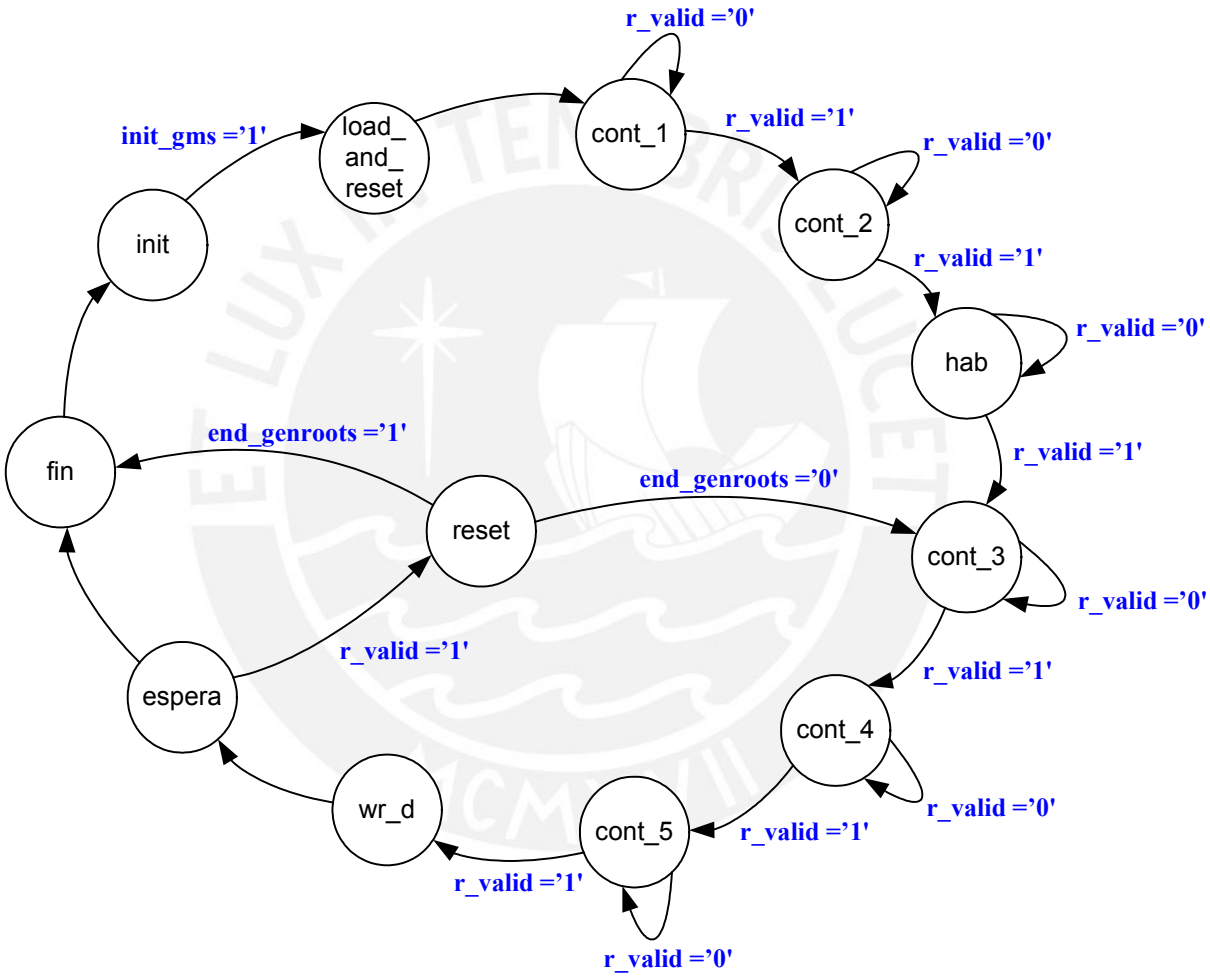


FIGURA 4.1.- DIAGRAMA DE ESTADOS DE LA UNIDAD DE CONTROL DEL PROCESADOR GMS

Parámetro	**Función**	
addr_width	Indica el ancho de bus de direcciones del controlador de memoria externa.	
isize_width	Indica el número de bits necesarios para representar el número de columnas ó filas que posee la imagen transformada (imagen cuadrada).	
data_width	Indica el tamaño en bits del dato a procesar.	
Señal de entrada	**Función**	**Número de bits**
im_data_rd	Bus de datos leídos desde memoria externa.	data_width

init_gsm	Señal de inicio del procesamiento GMS.	1
smd_data_rd	Bus de datos de lectura del Mapa D.	1
rw_ack_gsm	Señal de reconocimiento de dirección desde el controlador de memoria externa para la lectura de la imagen transformada.	1
r_valid_gsm	Señal que indica cuando el controlador de memoria externa tiene el dato listo en la lectura de la imagen transformada.	1
rows_p	Número de columnas (ó filas) de la imagen.	isize_width-2
Threshold	Valor Umbral para la codificación.	data_width
Stages	Número de etapas para el procesamiento GMS.	4
Clock	Señal de sincronismo del módulo.	1
Señal de salida	**Función**	**Número de bits**
smd_data_wr	Bus de datos de escritura en el Mapa D.	1
sm_write	Señal de habilitación de escritura en los Mapas.	1
sml_data_wr	Bus de datos de escritura en el Mapa L.	1
r_req_gsm	Señal que indica al controlador de memoria externa la petición de lectura de la imagen transformada.	1
smd_addr_rd	Bus de direcciones de lectura del Mapa D.	2*isize_width-2
smd_addr_wr	Bus de direcciones de escritura del Mapa D.	2*isize_width-2
im_addr_rd	Bus de direcciones de lectura de memoria externa.	addr_width
sml_addr_wr	Bus de direcciones de escritura del Mapa L.	2*isize_width-4
end_gsm	Fin del procesamiento de GMS.	1

TABLA 4.1. PARÁMETROS Y SEÑALES DE ENTRADA/SALIDA DEL PROCESADOR GMS.

4.2.3 RESULTADOS INDEPENDIENTES

El bloque fue compilado utilizando los siguientes valores de los parámetros: isize_width = 7,8,9 (para imágenes de 128x128, 256x256 y 512x512 píxeles respectivamente); data_width = 16; addr_width = 25 (ancho de bus de direcciones del controlador de memoria). Se realizaron compilaciones para los 3 tamaños de imágenes y los resultados son mostrados en la tabla 4.2. La simulación de este procesador para una imagen de 8 x 8 es mostrada en el Anexo B.

	TAMAÑO DE IMAGEN	LEs	%LEs	fmax (MHz)
GMS	128 x 128	213	< 1	157.28
	256 x 256	227	< 1	154.8
	512 x 512	238	< 1	154.05

TABLA 4.2.- RESULTADOS OBTENIDOS PARA EL BLOQUE PROCESADOR GMS

4.3 PROCESADOR CODEC

4.3.1 INTRODUCCIÓN

Esta es la unidad de procesamiento más importante de la arquitectura del sistema, debido a que realiza la codificación y decodificación de imágenes en el dominio Wavelet basándose en el algoritmo SPIHT[6]. Todo este núcleo ha sido integrado en un mismo bloque hardware, de tal manera que basta manipular una señal binaria llamada *cod_decod* para cambiar el modo del procesador CODEC; ya sea para la codificación (Modo C) o para la decodificación (Modo D). Se caracteriza por ejecutar de manera secuencial cuatro procesos (en cualquiera de sus dos modos), y bajo las reglas que establece el algoritmo original. Los procesos son los siguientes:

i) Etapa de Inicialización (general y por pasada).

ii) Procesamiento de la Lista de Píxeles Insignificantes (LIP).

iii) Procesamiento de la Lista de Conjuntos Insignificantes de Píxeles (LIS).

iv) Etapa de refinamiento y actualización del valor umbral.

En el caso de la codificación, este procesador se activa cuando finaliza el procesador GMS, para luego hacer uso de los mapas de significancias en el proceso que corresponde al procesamiento de LIS. En el caso de la decodificación, el procesador GMS no es activado porque para reconstruir la imagen transformada, el procesador CODEC sólo necesita la trama codificada. Además, se debe tener en cuenta, que el número de pasadas debe ser igual tanto en el Modo C como en el Modo D. La figura 4.2 muestra un diagrama temporal donde se muestran los

procesos que son ejecutados por el procesador CODEC con respecto al procesador GMS para dos pasadas del algoritmo SPIHT.

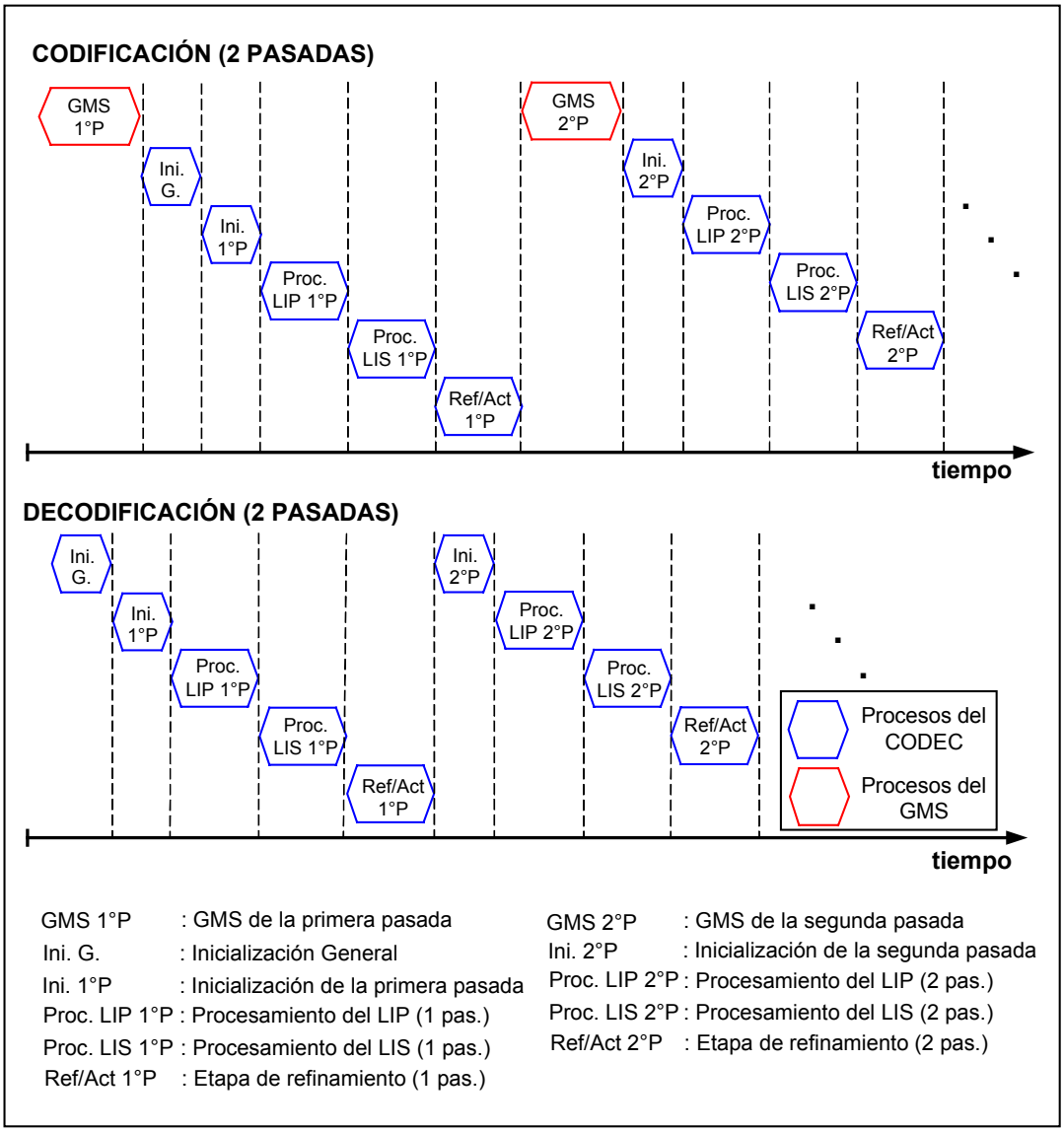


FIGURA 4.2.- DIAGRAMA TEMPORAL DE LOS DOS PROCESADORES EN EL DOMINO DEL TIEMPO PARA DOS PASADAS DEL ALGORITMO SPIHT

Según lo anteriormente descrito, se puede notar que el núcleo de procesamiento CODEC, puede ser dividido en módulos funcionales capaces de realizar cada uno de los procesos mencionados. Para ello, se propuso seguir la metodología de diseño Top-down y como primer paso se planteó la descripción algorítmica o pseudo-código

de cada uno de los procesos del algoritmo SPIHT, para finalmente mostrar en esta sección el diseño de cada módulo. De esta forma, los módulos desarrollados para realizar las tareas del núcleo de procesamiento CODEC son los siguientes:

a.- Módulo de Inicialización.

b.- Módulo de Procesamiento de LIP.

c.- Módulo de Procesamiento de LIS.

d.- Módulo de Procesamiento de LSP.

e.- Módulo de Punteros.

f.- Módulo Interfaz CODEC-Controlador de Memoria.

La lista LIP, la lista LIS y la lista LSP ocupan una zona específica de la memoria externa (los detalles de organización de memoria externa serán dados en el capítulo 5). Además, todos los módulos anteriormente mencionados realizan un acción sobre las listas, lo cual implica, que tendrán que efectuar operaciones de lectura y escritura en memoria externa. El último módulo es el responsable de la comunicación entre el Procesador CODEC y la memoria externa; y con él se presentará el hardware adicional para la interconexión de todo el Procesador CODEC.

4.3.2 MÓDULO DE INICIALIZACIÓN

4.3.2.1 COMPORTAMIENTO FUNCIONAL

La etapa de inicialización se encarga de colocar valores iniciales en las listas y de acondicionar los punteros de las mismas para un posterior proceso de codificación o decodificación. Esta etapa se divide en dos procesos:

- Inicialización general: Este proceso se ejecuta una única vez durante la codificación ó decodificación, y se encarga de cargar en la lista LIP las coordenadas de todas las raíces (i,j) $\in$ *H*, es decir: {(0,0), {(0,1), (1,0) y (1,1)}. Además, en la lista LIS se deben cargar las coordenadas de las raíces (i,j) $\in$ *H* que tienen descendientes, por lo tanto, tan solo se trata de las tres coordenadas: {(0,1), (1,0) y (1,1)} pero almacenadas como tipo D. Y por último, la constante "n" que representa la potencia de dos del valor umbral inicial; debe ser almacenada a modo de cabecera en la trama codificada cuando se realice un proceso de codificación y en el caso de la decodificación, debe ser leída de la trama codificada para reconstruir el valor umbral.

- Inicialización por pasada: Este proceso se activa al finalizar el proceso de inicialización general y se ejecuta al inicio de cada pasada de clasificación. Se encarga de poner a "cero" los índices de lectura y escritura de cada una de las listas. En el caso de la decodificación, se encarga de calcular los múltiplos del valor umbral que posteriormente serán usados para la reconstrucción de la imagen transformada.

4.3.2.2 DESARROLLO

El proceso correspondiente a la inicialización general, consta de dos sub-módulos: uno referido al camino de datos y otro a la unidad de control (ver PROC_CODEC_ARCH_INIG.VHD y PROC_CODEC_CTRL_INIG.VHD en sección Anexos A-2). El referido al camino de datos ha sido diseñado en base a contadores

y registros de desplazamiento. Este bloque consta de: Un circuito para la transmisión/recepción serial de la constante “n”, un bloque reconstructor del valor umbral inicial y circuitos contadores para la generación de las coordenadas a escribir en las listas LIP y LIS.

El formato a seguir para los datos de entrada de la lista LIP se forman a partir de la concatenación de dos grupos de bits de tamaño parametrizable; siendo el grupo de bits más significantes, el que representa la posición de las filas de la imagen transformada y el segundo grupo de bits (menos significantes) el que representa la posición de las columnas de la imagen transformada. Por otro lado, el formato para los datos de entrada de la lista LIS, además de lo anteriormente mencionado, poseen un bit adicional (el bit menos significativo) que indica el tipo de set (Tipo D o Tipo L). La figura 4.3 muestra una representación gráfica del formato usado.

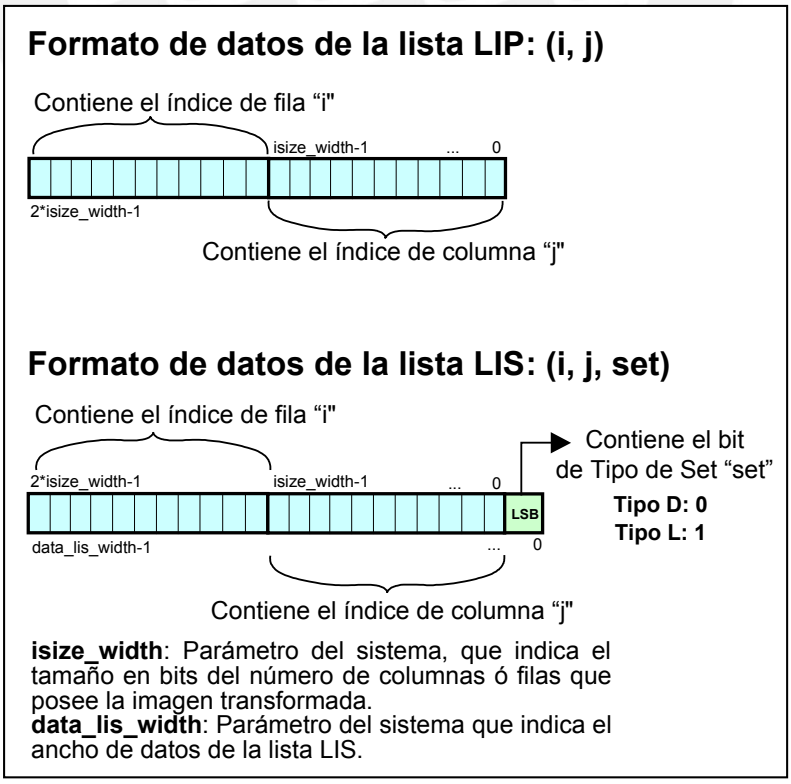


FIGURA 4.3.- FORMATO DE DATOS DE LIP Y LIS UTILIZADO EN EL DISEÑO

La unidad de control del sub-módulo de inicialización general se basa en una máquina de estados como la mostrada en la figura 4.4, que sigue el comportamiento algorítmico correspondiente al pseudo-código de la inicialización general (ver Anexo C).

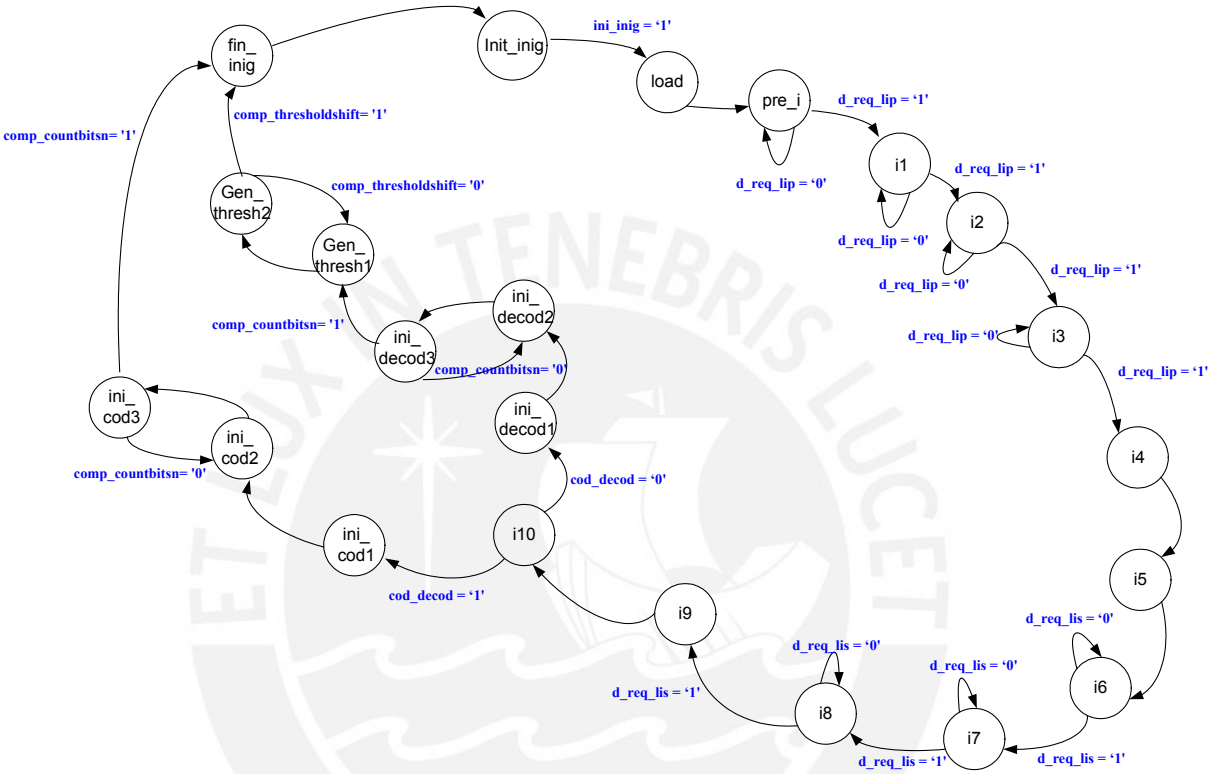


FIGURA 4.4.- DIAGRAMA DE ESTADOS DE LA INICIALIZACIÓN GENERAL

El proceso correspondiente a la inicialización por pasada, consta de dos sub-módulos: el primero referido al camino de datos y el segundo a la unidad de control (ver PROC_CODEC_ARCH_INIPAS.VHD y PROC_CODEC_CTRL_INIPAS.VHD en sección Anexos A-2). El camino de datos consta de un circuito que calcula el complemento a dos del valor umbral, y de un circuito que calcula múltiplos del valor umbral, tales como: valor umbral/2, -valor umbral/2, 1.5*valor umbral y -1.5*valor umbral. La unidad de control del sub-módulo de inicialización por pasada se basa en una máquina de estados como la mostrada en la figura 4.5, que sigue el

comportamiento algorítmico correspondiente al pseudo-código de la inicialización por pasada (ver Anexo C).

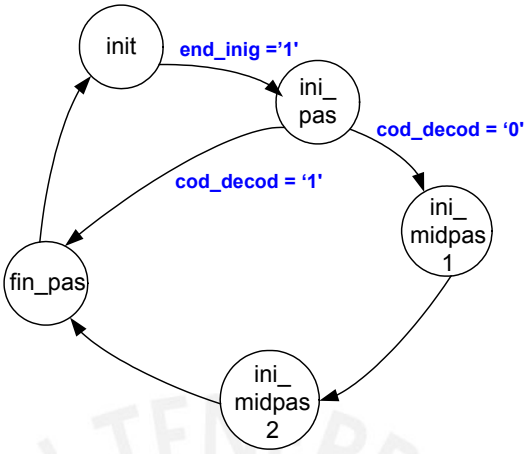


FIGURA 4.5- DIAGRAMA DE ESTADOS DE LA INICIALIZACIÓN POR PASADA

El módulo de inicialización se encuentra descrito en INICIALIZACION.VHD (ver sección Anexos A-2) y consta de la interconexión de los dos sub-módulos anteriormente descritos, de un circuito para dividir entre dos el valor umbral reconstruido para una siguiente pasada y compuertas lógicas para evitar la colisión de las señales de control. La tabla 4.3. muestra los parámetros, señales de entrada y señales de salida del módulo de inicialización, el cual se aprecia en el plano 2-1 de la sección planos.

Parámetro	**Función**	
isize_width	Indica el tamaño en bits del número de columnas ó filas que posee la imagen transformada (imagen cuadrada).	
data_width	Indica el tamaño en bits del dato a procesar.	
data_lis_width	Indica el tamaño en bits del dato perteneciente a la lista LIS	
Señal de entrada	**Función**	**Número de bits**
cod_decod	Señal que indica si se Codifica (cod_decod = '1') o Decodifica (cod_decod = '0').	1
ini_inig	Señal que activa el sub-bloque de inicialización general.	1
init_inipas	Señal que activa el sub-bloque de inicialización por pasada.	1
N_stream_decod	Dato serial recepcionado de la trama codificada.	1
d_req_lip	Señal que indica la petición desde el controlador de memoria externa para la escritura en LIP.	1

racklip_wr	Señal de reconocimiento de dirección desde el controlador de memoria externa para la escritura en LIP.	1
w_valid_lip	Señal que indica cuando el dato ha sido aceptado por el controlador de memoria externa para la escritura en LIP.	1
d_req_lis	Señal que indica la petición desde el controlador de memoria externa para la escritura en LIS.	1
racklis_wr	Señal de reconocimiento de dirección desde el controlador de memoria externa para la escritura en LIS.	1
w_valid_lis	Señal que indica cuando el dato ha sido aceptado por el controlador de memoria externa para la escritura en LIS.	1
Clock	Señal de sincronismo del bloque.	1
Señal de salida	**Función**	**Número de bits**
sel_lis	Selector del multiplexor mux_LIS.	1
ini_codec	Señal que habilita el flujo de datos para la inicialización general.	1
en_lip_wr_inig	Señal de habilitación de escritura en LIP por parte del sub-bloque de inicialización general.	1
en_lis_wr_inig	Señal de habilitación de escritura en LIS por parte del sub-bloque de inicialización general.	1
en_lip_rd_inipas	Señal de habilitación de escritura en LIP por parte del sub-bloque de inicialización por pasada.	1
data_ini_lip	Dato a escribir en LIP en la inicialización general	2*isize_width
data_ini_lis	Dato a escribir en LIS en la inicialización general	data_lis_width
w_req_lip	Señal que indica la petición de escritura hacia el controlador de memoria externa para escritura en LIP.	1
w_req_lis	Señal que indica la petición de escritura hacia el controlador de memoria externa para escritura en LIS.	1
threshold_rec	Dato que entrega el valor umbral reconstruido desde la trama codificada	data_width
threshold_05	Dato que entrega el valor umbral multiplicado por 0.5	data_width
threshold_15	Dato que entrega el valor umbral multiplicado por 1.5	data_width
threshold_neg05	Dato que entrega el negativo del valor umbral multiplicado por 0.5	data_width
threshold_neg15	Dato que entrega el negativo del valor umbral multiplicado por 1.5	data_width
end_inig	Señal de fin del sub-bloque de inicialización general	1
end_inipas	Señal de fin del sub-bloque de inicialización por pasada	1
aclr_indicelip_wr	Señal de reinicio del puntero de escritura de LIP.	1
aclr_indicelip_rd	Señal de reinicio del puntero de lectura de LIP.	1
aclr_indicelis_wr	Señal de reinicio del puntero de escritura de LIS.	1
aclr_indicelis_wr_fix	Señal de reinicio para el borrado del puntero en el rellenado de huecos de LIS	1
aclr_indicelis_rd	Señal de reinicio del puntero de lectura de LIS.	1
aclr_indicelsp_wr	Señal de reinicio del puntero de escritura de LSP.	1
aclr_indicelsp_rd	Señal de reinicio del puntero de lectura de LSP.	1
aclr_indice_trama_wr	Señal de reinicio del puntero de escritura de la trama codificada.	1
aclr_indice_trama_rd	Señal de reinicio del puntero de lectura de la trama codificada.	1
ld_lip_size	Señal de carga del registro que indica el tamaño de LIP.	1
ld_lis_size	Señal de carga del registro que indica el tamaño de LIS.	1
ld_lsp_pasada_anterior	Señal de carga del registro que indica el tamaño de LSP.	1
ld_indicelis_wr	Señal de carga el puntero de escritura de LIS.	1
aux_racklip_wr		1
aux_racklis_wr		1
clr_hab_lip_wr	Señal de borrado del flip-flop que habilita el rellenado de huecos en LIP.	1

clr_hab_lis_wr_fix	Señal de borrado del flip-flop que habilita el rellenado de huecos en LIS.	1
en_indice_ trama_wr	Señal de habilitación del puntero de escritura de la trama codificada.	1
en_indice_ trama_rd	Señal de habilitación del puntero de lectura de la trama codificada.	1
en_trama_wr	Señal de habilitación para la escritura en memoria interna que representa a la trama codificada.	1
en_trama_rd	Señal de habilitación para la lectura en memoria interna que representa a la trama codificada.	1
sel_lis_wr	Selector del multiplexor mux_lis_wr	1
N_stream_cod	Salida serial transmitida hacia la memoria interna	1

TABLA 4.3. PARÁMETROS Y SEÑALES DE ENTRADA/SALIDA DEL MÓDULO INICIALIZACIÓN

4.3.2.3 RESULTADOS INDEPENDIENTES

El bloque fue compilado utilizando los siguientes valores de los parámetros: isize_width = 7,8,9 (para imágenes de 128x128, 256x256, y 512x512 píxeles respectivamente); data_width = 16; data_lis_width = 2*isize_width + 1(ver figura 4.2). Se realizaron compilaciones para los 3 tamaños de imágenes obteniéndose en los tres casos los mismos resultados. El módulo diseñado ocupa 170 elementos lógicos (< 1% de LE´s del FPGA utilizado), y alcanza una frecuencia máxima de trabajo de 236.46 MHz. Simulaciones de este módulo, tanto en modo codificación como en modo decodificación, son mostradas en el Anexo B.

4.3.3 MÓDULO DE PROCESAMIENTO DE LIP

4.3.3.1 COMPORTAMIENTO FUNCIONAL

El módulo de procesamiento de LIP se encarga del ordenamiento de las coordenadas de los coeficientes anteriormente escritos en esta lista; ya sea por el

proceso de inicialización general, si es la primera pasada del algoritmo, o por el proceso de LIS, si estos coeficientes fueron considerados insignificantes en pasadas previas del algoritmo. Estos coeficientes son analizados en la pasada actual para determinar su significancia. Si el elemento analizado es significante, este pasa a la lista LSP y es eliminado de la lista LIP, pero si es insignificante, permanece en la lista LIP para ser analizado nuevamente en la próxima pasada considerando un nuevo valor umbral equivalente a la mitad del actual. Esta operación de ordenamiento no depende del modo de operación del procesador CODEC.

Además, si el procesador CODEC está trabajando en modo de codificación (Modo C), el módulo de procesamiento de LIP almacena bits en la trama codificada. Si el coeficiente analizado es significante, almacena un '1' en la trama seguido por su bit de signo ('0' si es positivo, '1' si es negativo). Si es insignificante, almacena un '0' en la trama. Por otro lado, si el procesador CODEC está trabajando en modo de decodificación (Modo D), el módulo de procesamiento de LIP lee bits de la trama codificada. Si el bit correspondiente al elemento de LIP analizado es '1', escribe en la coordenada de la imagen reconstruida que corresponde a este elemento el valor umbral multiplicado por 1.5, si es positivo, o multiplicado por –1.5 si es negativo.

4.3.3.2 DESARROLLO

El módulo de procesamiento de LIP se compone de un sub-módulo referido al camino de datos y otro referido a su respectiva unidad de control (ver PROC_CODEC_ARCH_LIP.VHD y PROC_CODEC_CTRL_LIP.VHD en la sección Anexos A-2). El camino de datos está conformado principalmente por un comparador, el cual compara el valor absoluto del coeficiente de entrada con el valor

umbral para determinar su significancia y dos flip-flop's que almacenan el valor de la significancia del coeficiente y su signo. La unidad de control del módulo de procesamiento de LIP se basa en una máquina de estados como la mostrada en la figura 4.6, que sigue el comportamiento algorítmico correspondiente al pseudo-código del procesamiento de LIP (ver Anexo C). La interconexión de la arquitectura y su correspondiente control, se muestra en el plano 2-2 de la sección Planos. La tabla 4.4 muestra los parámetros, señales de entrada y señales de salida del módulo de procesamiento LIP.

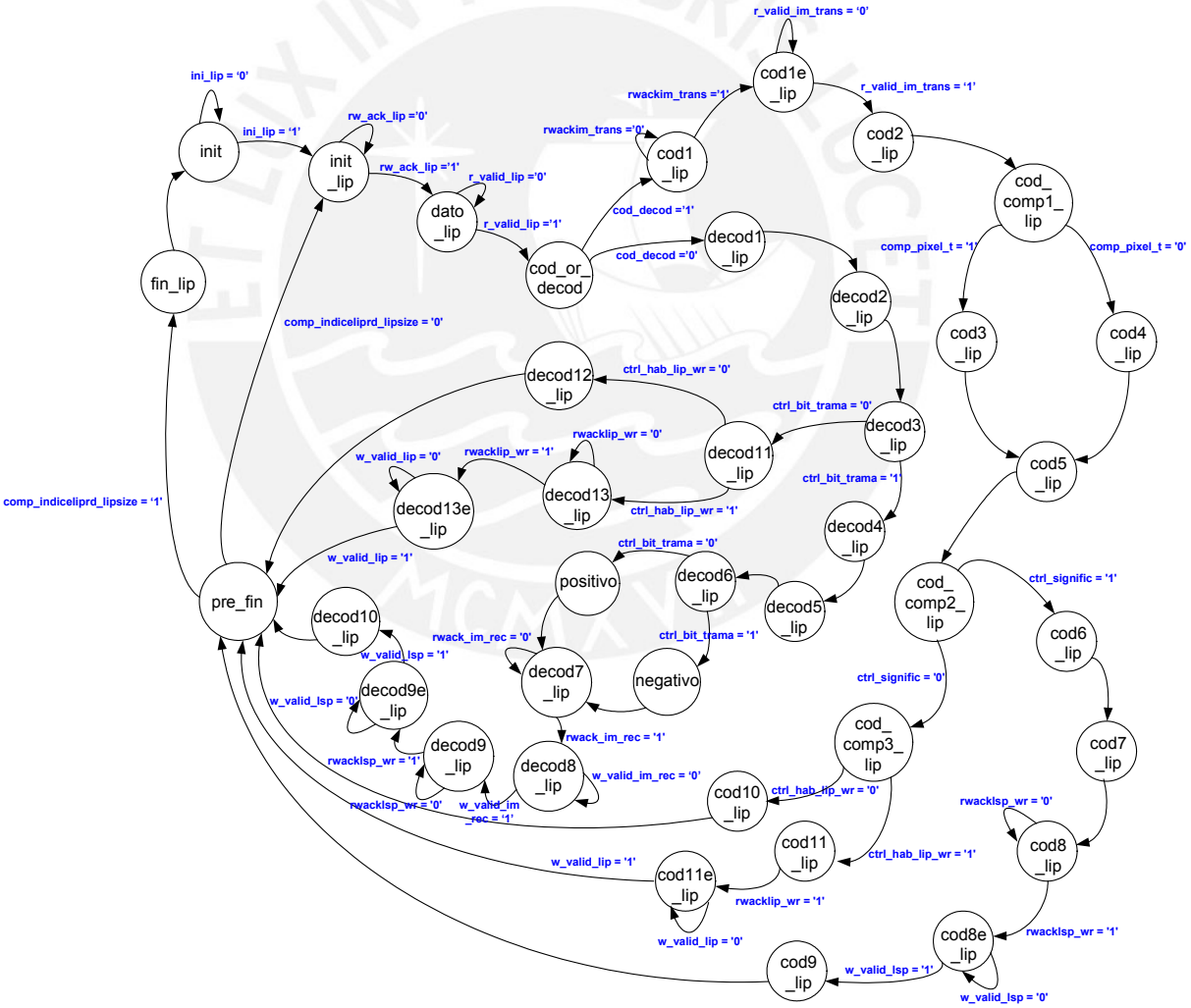


FIGURA 4.6.- DIAGRAMA DE ESTADOS DE LA UNIDAD DE CONTROL DEL MÓDULO DE PROCESAMIENTO DE LIP

Parámetro	**Función**	
data_width	Indica el tamaño en bits del dato a procesar.	
Señal de entrada	**Función**	**Número de bits**
threshold_cod	Valor umbral para la codificación.	data_width
data_im_pixel	Valor del coeficiente leído de memoria externa.	data_width
ini_lip	Señal de inicio del procesamiento LIP.	1
Cod_decod	Señal que indica si se Codifica (cod_decod = '1') o Decodifica (cod_decod = '0').	1
Ctrl_bit_trama	Señal que indica si el bit leído de la memoria interna trama pertenece a un coeficiente significante.	1
Ctrl_hab_lip_wr	Señal que habilita el llenado de huecos en LIP.	1
comp_indiceliprd _lipsize	Señal que indica si el puntero de lectura llegó al final de la lista LIP.	1
rw_ack_lip	Señal de reconocimiento de dirección desde el controlador de memoria externa para la lectura de LIP.	1
R_valid_lip	Señal que indica cuando el controlador de memoria externa tiene el dato listo en la lectura de LIP.	1
Rwacklip_wr	Señal de reconocimiento de dirección desde el controlador de memoria externa para la escritura en LIP.	1
w_valid_lip	Señal que indica cuando el dato ha sido aceptado por el controlador de memoria externa para la escritura en LIP.	1
rwacklsp_wr	Señal de reconocimiento de dirección desde el controlador de memoria externa para la escritura en LSP.	1
w_valid_lsp	Señal que indica cuando el dato ha sido aceptado por el controlador de memoria externa para la escritura en LSP.	1
Rwackim_trans	Señal de reconocimiento de dirección desde el controlador de memoria externa para la lectura de la imagen transformada.	1
R_valid_im_trans	Señal que indica cuando el controlador de memoria externa tiene el dato listo en la lectura de la imagen transformada.	1
rwack_im_rec	Señal de reconocimiento de dirección desde el controlador de memoria externa para la escritura de la imagen reconstruida.	1
w_valid_im_rec	Señal que indica cuando el dato ha sido aceptado por el controlador de memoria externa para la escritura de la imagen reconstruida.	1
en_sign_or	Señal de carga del flip-flop que almacena el signo del dato leído.	1
en_signific_or	Señal de carga del flip-flop que almacena la significancia del dato leído respecto al valor umbral	1
Clock	Señal de sincronismo del módulo.	1
Señal de salida	**Función**	**Número de bits**
data_im_pixel_abs	Valor absoluto del dato leído.	data_width
en_im_trans	Señal de habilitación de lectura de la imagen transformada.	1
en_im_rec	Señal de habilitación de escritura de la imagen reconstruida.	1
en_lsp_wr	Señal de habilitación de escritura en LSP.	1
en_lip_wr	Señal de habilitación de escritura en LIP.	1
en_lip_rd_	Señal de habilitación de lectura de LIP.	1
r_req_lip	Señal que indica al controlador de memoria externa la petición de lectura de LIP.	1
w_req_lip	Señal que indica al controlador de memoria externa la petición de escritura en LIP.	1
en_indicelip_rd	Señal de incremento del puntero de lectura de LIP.	1
en_indicelip_wr	Señal de incremento del puntero de escritura de LIP.	1

en_hab_lip_wr	Señal de carga del flip-flop que habilita el rellenado de huecos en LIP.	1
en_indice_trama _wr	Señal de incremento del puntero de escritura de la memoria interna Trama.	1
en_indice_trama_ rd	Señal de incremento del puntero de lectura de la memoria interna Trama.	1
en_escribir_trama	Habilitación de escritura en la memoria interna Trama.	1
w_req_lsp	Señal que indica al controlador de memoria externa la petición de escritura en LSP.	1
en_indicelsp_wr	Señal de incremento del puntero de escritura de LSP.	1
en_bit_trama	Señal de carga del flip-flop que almacena el bit leído de la memoria interna Trama.	1
en_sign	Señal de carga del flip-flop que almacena el signo del dato leído entregada por el control de LIP.	1
en_signific	Señal de carga del flip-flop que almacena la significancia del dato leído respecto al valor umbral, entregada por el control de LIP.	1
r_req_im_trans	Señal que indica al controlador de memoria externa la petición de lectura de la imagen transformada.	1
w_req_im_rec	Señal que indica al controlador de memoria externa la petición de escritura de la imagen reconstruida.	1
ld_píxel_dec	Señal de carga del registro que almacena la el coeficiente reconstruido a escribirse en la memoria externa.	1
sel_muxdata_im _rec	Selector del multiplexor muxdata_im_rec.	2
Sel_mux_trama	Selector del multiplexor mux_trama.	2
Sel_premux_itrans	Selector del multiplexor pre_mux_itrans.	1
Sel_mux_itrans_rd	Selector del multiplexor mux_itrans	1
Sign	Flip-flop que almecena el signo del dato leído.	1
End_lip	Señal de fin del procesamiento de LIP.	1

TABLA 4.4. PARÁMETROS Y SEÑALES DE ENTRADA/SALIDA DEL MÓDULO DE PROCESAMIENTO DE LIP

4.3.3.3 RESULTADOS INDEPENDIENTES

El bloque fue compilado utilizando los siguientes valores de los parámetros: isize_width = 7,8,9 (para imágenes de 128x128, 256x256, y 512x512 píxeles respectivamente); data_width = 16.

Fueron realizadas compilaciones para los 3 tamaños de imágenes obteniéndose en los todos los casos los mismos resultados. El módulo diseñado ocupa 87 elementos lógicos (< 1% de LE´s del FPGA utilizado), y alcanza una frecuencia máxima de

trabajo de 375.09 MHz. Simulaciones de este módulo, tanto en modo codificación como en modo decodificación, son mostradas en el Anexo B.

4.3.4 MÓDULO DE PROCESAMIENTO DE LIS

4.3.4.1 COMPORTAMIENTO FUNCIONAL

El módulo de procesamiento de LIS se encarga del ordenamiento de las coordenadas de los coeficientes que conforman los sets que pertenecen a dicha lista. Los sets de esta lista fueron escritos anteriormente ya sea por el proceso de inicialización general ó por el mismo proceso de LIS (en la pasada actual o en pasadas anteriores).

Los sets de la lista LIS son leídos secuencialmente, y cuando uno es encontrado significante, es eliminado de la lista. Existen dos tipos de sets: Los sets de tipo D y los sets de tipo L. Si un set significante es de tipo D sus cuatro hijos son analizados, y si tiene nietos, el set es escrito al final de la lista LIS como un set de tipo L. Los hijos que sean encontrados significantes, pasan a la lista LSP y los que sean encontrados insignificantes, pasan a la lista LIP. Si el set que fue encontrado significante es de tipo L, las coordenadas de sus cuatro hijos son escritas al final de la lista LIS como entradas de tipo D. Cabe resaltar que cada vez que un set es añadido al final de la lista LIS, este es analizado en la misma pasada del algoritmo. Para conocer la significancia de un set, el módulo de procesamiento de LIS lee el mapa D (si es un set de tipo D) o lee el mapa L (si es un set de tipo L) en la posición del mapa que corresponde a las coordenadas del set analizado.

Si el procesador CODEC está trabajando en modo de codificación (Modo C), el módulo de procesamiento de LIS almacena bits en la trama codificada. Si un set es significante escribe un '1' en la trama y si es de tipo D, a continuación escribe los bits que representan las significancias de sus hijos. Es decir, si un hijo es significante escribe un '1' seguido de su bit de signo, y si es insignificante, escribe un '0'. Si por el contrario el set es insignificante, solamente se escribe un '0'. Si el procesador CODEC está trabajando en modo decodificación (Modo D), el módulo de procesamiento de LIS lee bits de la trama codificada. Si un bit correspondiente al hijo de un set significante de tipo D es '1', entonces se escribe en la coordenada de la imagen reconstruida que corresponde a la coordenada de dicho hijo el valor umbral multiplicado por 1.5 si es positivo, o multiplicado por -1.5 si es negativo.

4.3.4.2 DESARROLLO

El módulo de procesamiento de LIS se compone de un sub-módulo referido al camino de datos y otro referido a la unidad de control (ver PROC_CODEC_ARCH_LIS.VHD y LIS_CTRL.VHD en la sección Anexos A-2). El sub-módulo referido al camino de datos fue diseñado en base a registros, contadores y sumadores. Está dividido en cuatro sub-módulos que se encargan de diferentes tareas: leer la significancia del set, cambiar el tipo del set, verificar la existencia de nietos, y generar las direcciones de los píxeles “Hijos”. El sub-módulo referido a la unidad de control está conformado por varias máquinas de estados, cuya organización jerárquica es mostrada en la figura 4.6. La descripción de cada una de estas máquinas se muestra en la tabla 4.5 y sus diagramas de estados en el Anexo D.

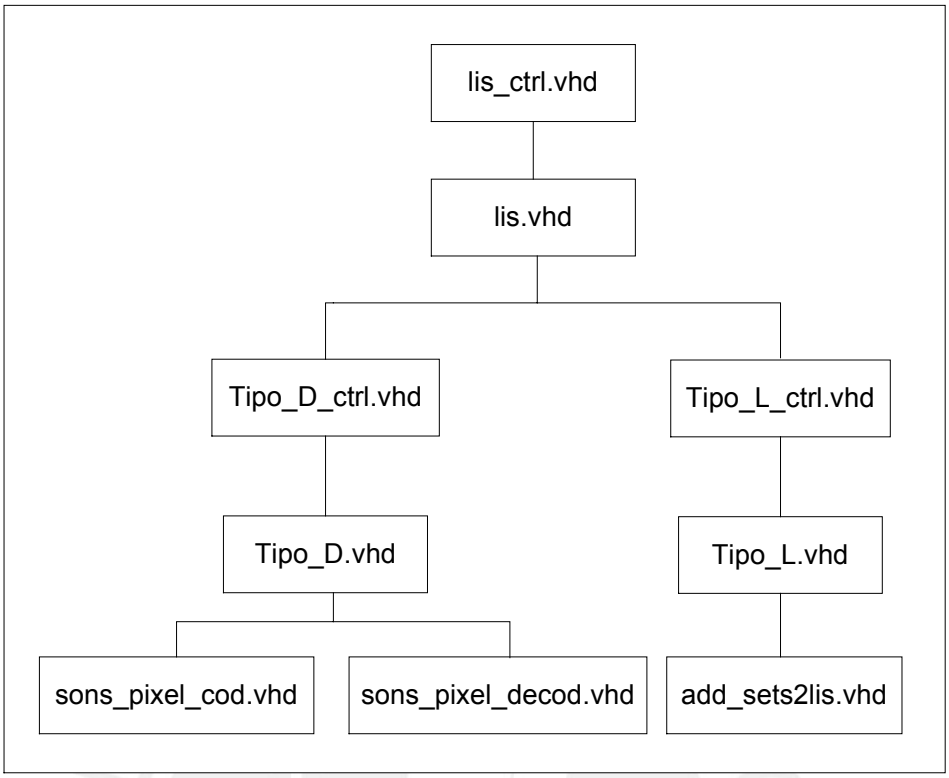


FIGURA 4.7.- JERARQUÍA DE LOS ARCHIVOS DEL CONTROL DE LIS

Módulo	**Descripción**
add_sets2lis.vhd	Maquina de estados que controla la escritura de los hijos de un set tipo L como sets tipo D.
Sons_píxel_cod.vhd	Maquina de estados que controla el análisis de los hijos de un set tipo D en modo codificación.
Sons_píxel_decod.vhd	Maquina de estados que controla el análisis de los hijos de un set tipo D en modo decodificación.
Tipo_D.vhd	Maquina de estados que controla el procesamiento de LIS cuando el set analizado es de tipo D.
Tipo_L.vhd	Maquina de estados que controla el procesamiento de LIS cuando el set analizado es de tipo L.
Tipo_D_ctrl.vhd	Agrupa los controles Tipo_D.vhd, sons_píxel_cod.vhd y Tipo_D_ctrl.vhd.
Tipo_L_ctrl.vhd	Agrupa los controles Tipo_L.vhd y add_sets2lis.vhd.
lis.vhd	Maquina de estados que representa el control general del procesamiento de LIS.
lis_ctrl..vhd	Agrupa los controles lis.vhd , Tipo_D_ctrl.vhd y Tipo_L_ctrl.vhd.

TABLA 4.5.- DESCRIPCIÓN DE LOS ARCHIVOS DE CONTROL DE LIS

La tabla 4.6 muestra los parámetros y señales de entrada/salida del módulo de procesamiento de LIS. Además, el camino de datos y el control de mayor nivel se muestra en el Plano 2-3.

Parámetro	**Función**	
isize_width	Indica el número de bits necesarios para representar el número de columnas ó filas que posee la imagen transformada (imagen cuadrada).	
Data_lis_width	Indica el tamaño en bits del dato perteneciente a la lista LIS.	
Señal de entrada	**Función**	**Número de bits**
Data_lis_rd	Datos leídos de LIS.	data_lis_width
n_cols	Número de columnas de la imagen.	isize_width
ini_lis	Señal de inicio del procesamiento de LIS.	1
Cod_decod	Señal que indica si se Codifica (cod_decod = '1') o Decodifica (cod_decod = '0').	1
Data_map_d_rd	Datos leídos del Mapa D.	1
Data_map_l_rd	Datos leídos del Mapa L.	1
bit_trama	Flip-flop que almacena el bit leído de la memoria interna Trama.	1
Signific	Flip-flop que almacena la significancia del dato leído respecto al valor umbral	1
Hab_lis_wr_fix	Flip-flop que habilita el rellenado de huecos en LIS.	1
go_lis	Señal que indica si el puntero de lectura llegó al final de la lista LIS.	1
r_valid_im_trans	Señal que indica cuando el controlador de memoria externa tiene el dato listo en la lectura de la imagen transformada.	1
r_valid_lis	Señal que indica cuando el controlador de memoria externa tiene el dato listo en la lectura de LIS.	1
w_valid_lis	Señal que indica cuando el dato ha sido aceptado por el controlador de memoria externa para la escritura en LIS.	1
w_valid_lsp	Señal que indica cuando el dato ha sido aceptado por el controlador de memoria externa para la escritura en LSP.	1
w_valid_lip	Señal que indica cuando el dato ha sido aceptado por el controlador de memoria externa para la escritura en LIP.	1
w_valid_im_rec	Señal que indica cuando el dato ha sido aceptado por el controlador de memoria externa para la escritura de la imagen reconstruida.	1
rwackim_trans	Señal de reconocimiento de dirección desde el controlador de memoria externa para la lectura de la imagen transformada.	1
rwacklis_rd	Señal de reconocimiento de dirección desde el controlador de memoria externa para la lectura de LIS.	1
racklis_wr	Señal de reconocimiento de dirección desde el controlador de memoria externa para la escritura de LIS.	1
Rwacklsp_wr	Señal de reconocimiento de dirección desde el controlador de memoria externa para la escritura de LSP.	1
racklip_wr	Señal de reconocimiento de dirección desde el controlador de memoria externa para la lectura LIP.	1
rwack_im_rec	Señal de reconocimiento de dirección desde el controlador de memoria externa para la escritura de la imagen reconstruida.	1
Clock	Señal de sincronismo del módulo.	1

Señal de salida	Función	Número de bits
in_mux_ini_lis	Salida del multiplexor mex_pre_ini_lis.	data_lis_width
address_sons	Dirección de los hijos de un coeficiente.	data_lis_width
addr_maps	Dirección leída de LIS.	data_lis_width
en_indice_lis_rd	Señal de incremento del puntero de lectura de LIS.	1
sel_lis_wr	Selector del multiplexor mux_lis_wr.	1
ld_lis_size	Señal de carga del registro que indica el tamaño de LIS.	1
ld_lip_size	Señal de carga del registro que indica el tamaño de LIP.	1
sel_mux_trama	Selector del multiplexor mux_trama.	2
en_indice_trama _rd	Señal de incremento del puntero de lectura de la memoria interna Trama.	1
en_indice_lis_wr _fix	Señal de incremento del puntero de rellenado de huecos de LIS.	1
sel_mux_itrans_rd	Selector del multiplexor mux_itrans	1
en_signific	Señal de carga del flip-flop que almacena la significancia del dato leído respecto al valor umbral.	1
en_indice_trama _wr	Señal de incremento del puntero de escritura de la memoria interna Trama.	1
en_indice_lip_wr	Señal de incremento del puntero de escritura de LIP.	1
en_indicelsp_wr	Señal de incremento del puntero de escritura de LSP.	1
en_indice_lis_wr	Señal de incremento del puntero de escritura de LIS.	1
en_sign	Señal de carga del flip-flop que almacena el signo del dato leído.	1
sel_lis	Selector del multiplexor mux_lis	1
en_bit_trama	Señal de carga del flip-flop que almacena el bit leído de la memoria interna Trama.	1
sel_pre_mux _itrans	Selector del multiplexor pre_mux_itrans.	1
sel_muxdata_im _rec	Selector del multiplexor muxdata_im_rec.	2
ld_pixel_dec	Señal de carga del registro que almacena la el coeficiente reconstruido a escribirse en la memoria externa.	1
en_lis_wr_fix	Señal de carga del flip-flop que habilita el rellenado de huecos en LIS.	1
en_trama_wr	Señal de incremento del puntero de escritura de la memoria interna Trama.	1
r_req_im_trans	Señal que indica al controlador de memoria externa la petición de lectura de la imagen transformada.	1
r_req_lis	Señal que indica al controlador de memoria externa la petición de lectura de LIS.	1
w_req_lis	Señal que indica al controlador de memoria externa la petición de escritura de LIS.	1
w_req_lsp	Señal que indica al controlador de memoria externa la petición de escritura de LSP.	1
w_req_lip	Señal que indica al controlador de memoria externa la petición de escritura de LIP.	1
w_req_im_rec	Señal que indica al controlador de memoria externa la petición de escritura de la imagen reconstruida.	1
en_lis_wr	Señal de habilitación de escritura en LIS.	1
en_im_rec	Señal de habilitación de escritura de la imagen reconstruida.	1
en_im_trans	Señal de habilitación de lectura de la imagen transformada.	1

en_lsp_wr	Señal de habilitación de escritura en LSP.	1
en_lip_wr	Señal de habilitación de escritura en LIP.	1
en_lis_rd	Señal de habilitación de lectura de LIS.	1
fin_lis	Señal de fin de procesamiento de LIS.	1

TABLA 4.6.- PARÁMETROS Y SEÑALES DE ENTRADA/SALIDA DEL MÓDULO DE PROCESAMIENTO DE LIS

4.3.4.3 RESULTADOS INDEPENDIENTES

El módulo fue compilado utilizando los siguientes valores de los parámetros: isize_width = 7,8,9 (para imágenes de 128x128, 256x256, y 512x512 respectivamente); data_lis_width = 2*isize_width + 1 (ver figura 4.2).

Fueron realizadas compilaciones para los 3 tamaños de imágenes y los resultados son mostrados en la tabla 4.7. Simulaciones de este módulo, tanto en modo codificación como en modo decodificación, son mostradas en el Anexo B.

	TAMAÑO DE IMAGEN	LEs	%LEs	Fmax (MHz)
LIS	128 x 128	166	< 1	422.12
	256 x 256	172	< 1	409.33
	512 x 512	178	< 1	381.66

TABLA 4.7.- RESULTADOS OBTENIDOS PARA EL MÓDULO DE PROCESAMIENTO LIS

4.3.5 MÓDULO DE PROCESAMIENTO DE LSP

4.3.5.1 COMPORTAMIENTO FUNCIONAL

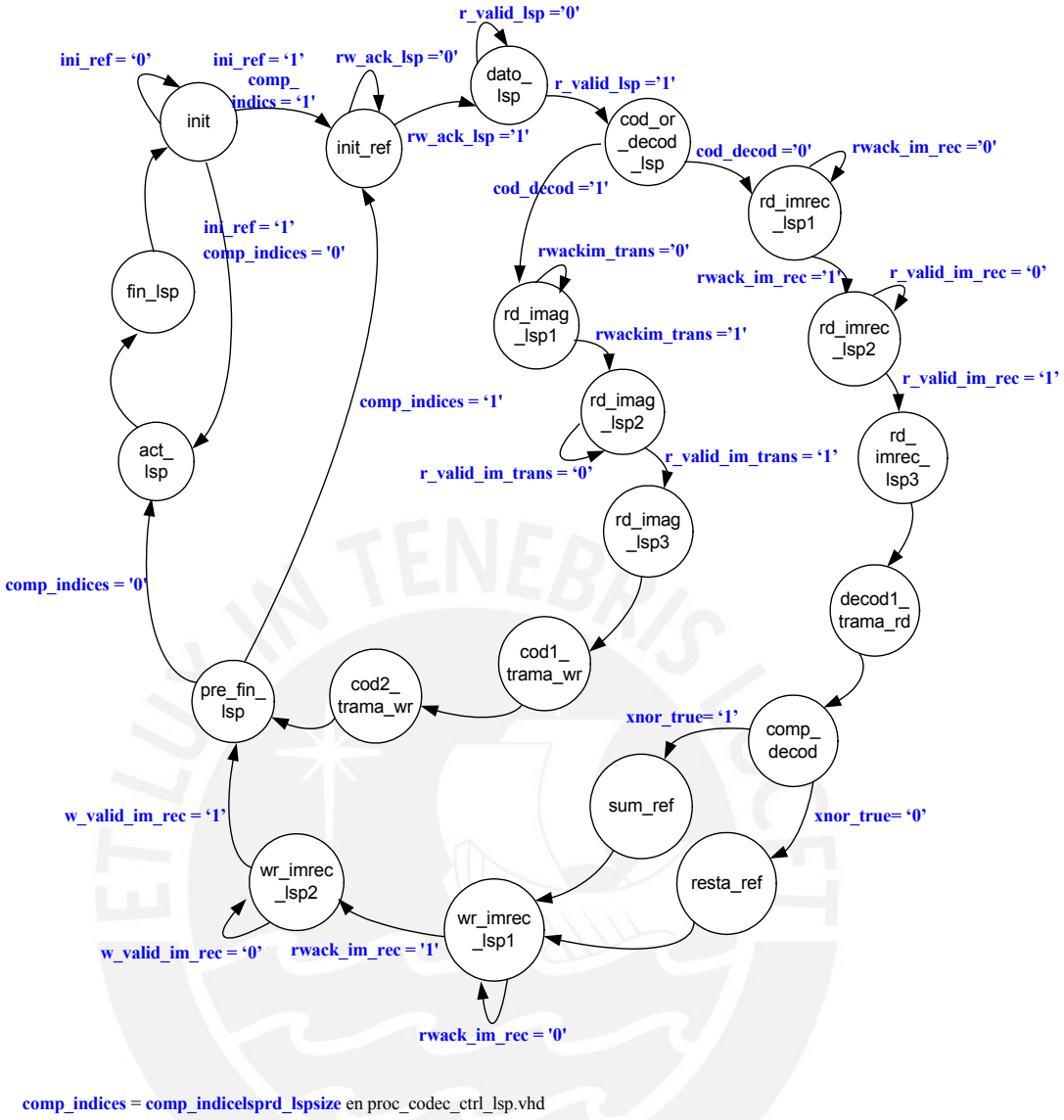
Este módulo se encarga de realizar la etapa de refinamiento sobre los elementos de la lista LSP, la cual contiene las coordenadas de los píxeles que fueron clasificados como “significantes” después de la pasada de clasificación. El refinamiento consiste

en transmitir en cada pasada los bits más significativos ubicados en la posición “n” de cada uno de los píxeles que posee la lista LSP, donde “n” es la potencia de dos del valor umbral. El formato usado para almacenar las coordenadas de los píxeles en las listas LSP es el mismo que el formato usado para la lista LIP (ver figura 4.2).

4.3.5.2 DESARROLLO

El módulo de procesamiento de LSP se compone de dos sub-módulos: uno referido al camino de datos y otro a su respectiva unidad de control (ver PROC_CODEC_ARCH_LSP.VHD y PROC_CODEC_CTRL_LSP.VHD en la sección Anexos A-2). El camino de datos está conformado principalmente una unidad aritmético lógica (ver ALU_LSP.VHD en la sección Anexos A-2) que se encarga de extraer el bit *n*-ésimo de coeficiente Wavelet significante, cuya coordenada se encuentra almacenado en la lista LSP y además posee una bandera de signo que sólo es usada cuando se encuentra en modo decodificación, ya que en este modo el módulo de procesamiento de LSP realiza una corrección de ±(valor umbral)/2 sobre cada coeficiente reconstruido. La unidad de control del módulo de procesamiento de LSP se basa en una máquina de estados como la mostrada en la figura 4.8, que sigue el comportamiento algorítmico correspondiente al pseudo-código del procesamiento de LSP (ver Anexo C).

La interconexión de los dos sub-módulos se muestra en el plano 2-4 en la sección Planos, además de los parámetros y señales de entrada/salida del módulo de procesamiento de LSP que se listan en la tabla 4.8.



comp_indices = comp_indicelsprd_lspsize en proc_codec_ctrl_lsp.vhd

FIGURA 4.8.- DIAGRAMA DE ESTADOS DE LA UNIDAD DE CONTROL DEL MÓDULO DE PROCESAMIENTO DE LSP

Parámetro	**Función**	
data_width	Indica el tamaño en bits del dato a procesar.	
Señal de entrada	**Función**	**Número de bits**
data_im_pixel_abs	Valor absoluto del dato leído.	data_width
threshold_cod	Valor Umbral para la codificación.	data_width
data_im_pixel	Valor del coeficiente leído de memoria externa.	data_width
ini_ref	Señal de inicio del procesamiento de LSP.	1
cod_decod	Señal que indica si se Codifica (cod_decod = '1') o Decodifica (cod_decod = '0').	1
bit_trama	Flip-flop que almacena el bit leído de la memoria interna Trama.	1
comp_indicelsprd_lspsize	Señal que indica si el puntero de lectura llegó al final de la lista LSP.	1
xnor_true	Si xnor_true='1' entonces píxel_ref=píxel + 0.5(Threshold) Si xnor_true='0' entonces píxel_ref=píxel - 0.5(Threshold)	1

rwacklsp_rd	Señal de reconocimiento de dirección desde el controlador de memoria externa para la lectura de LSP.	1
r_valid_lsp	Señal que indica cuando el controlador de memoria externa tiene el dato listo en la lectura de LSP.	1
rwackim_trans	Señal de reconocimiento de dirección desde el controlador de memoria externa para la lectura de la imagen transformada.	1
r_valid_im_trans	Señal que indica cuando el controlador de memoria externa tiene el dato listo en la lectura de la imagen transformada.	1
rwack_im_rec	Señal de reconocimiento de dirección desde el controlador de memoria externa para la escritura y lectura de la imagen reconstruida.	1
r_valid_im_rec	Señal que indica cuando el controlador de memoria externa tiene el dato listo en la lectura de la imagen reconstruida.	1
w_valid_im_rec	Señal que indica cuando el dato ha sido aceptado por el controlador de memoria externa para la escritura de la imagen reconstruida.	1
clock	Señal de sincronismo del módulo.	1
Señal de salida	**Función**	**Número de bits**
r_req_lsp	Señal que indica al controlador de memoria externa la petición de lectura de LSP.	1
w_req_im_rec	Señal que indica al controlador de memoria externa la petición de escritura de la imagen reconstruida.	1
r_req_im_trans	Señal que indica al controlador de memoria externa la petición de lectura de la imagen transformada.	1
r_req_im_rec	Señal que indica al controlador de memoria externa la petición de lectura de la imagen reconstruida.	1
en_im_trans	Señal de habilitación de lectura de la imagen transformada.	1
en_im_rec	Señal de habilitación de escritura y lectura de la imagen reconstruida.	1
en_lsp_rd	Señal de habilitación de lectura de LSP.	1
en_indicelsp_rd	Señal de incremento del puntero de lectura de LSP.	1
en_bit_trama	Señal de carga del flip-flop que almacena el bit leído de la memoria interna Trama.	1
en_escribir_trama	Habilitación de escritura en la memoria interna Trama.	1
en_indice_trama _rd	Señal de incremento del puntero de lectura de la memoria interna Trama.	1
en_indice_trama _wr	Señal de incremento del puntero de escritura de la memoria interna Trama.	1
ld_pixel_ref	Señal de carga del registro que almacena la el coeficiente reconstruido a escribirse en la memoria externa.	1
ld_pasada _anterior	Señal de carga del registro que indica el tamaño de LSP. Después de la última pasada.	1
ld_address_spixel	Señal de carga del registro que almacena la dirección leída de LSP.	1
sel_mux_data _imrec_lsp	Selector del multiplexor muxdata_im_rec.	2
sel_mux_trama	Selector del multiplexor mux_trama.	2
sel_mux_itrans_rd	Selector del multiplexor mux_itrans.	1
end_lsp	Señal de fin de procesamiento de LSP.	1

TABLA 4.8.- PARÁMETROS Y SEÑALES DE ENTRADA/SALIDA DEL MÓDULO DE PROCESAMIENTO DE LSP

4.3.5.3 RESULTADOS INDEPENDIENTES

El bloque fue compilado utilizando los siguientes valores de los parámetros: isize_width = 7,8,9 (para imágenes de 128x128, 256x256, y 512x512 respectivamente); data_width = 16.

Fueron realizadas compilaciones para los 3 tamaños de imágenes obteniéndose en los todos los casos los mismos resultados. El módulo diseñado ocupa 35 elementos lógicos (< 1% de LE´s del FPGA utilizado), y alcanza una frecuencia máxima de trabajo de 375.09 MHz. Simulaciones de este módulo, tanto en modo codificación como en modo decodificación, son mostradas en el Anexo B.

4.3.6 MÓDULO DE PUNTEROS

4.3.6.1 COMPORTAMIENTO FUNCIONAL

El módulo incluye todos los punteros, tanto de lectura como de escritura de las listas (LIP, LIS y LSP), de la imagen transformada, de la imagen reconstruida y de la trama codificada. Además, incluye los registros de tamaño de listas y habilitadores de rellenado de huecos.

El algoritmo SPIHT requiere que, en ciertos casos, sean eliminados elementos de las listas LIP y LIS. Para poder remover estos elementos sin dejar espacios vacíos, los módulos generadores de direcciones de dichas listas fueron diseñados empleando una técnica de punteros dobles.

En el caso de la lista LIP, los punteros de lectura y escritura se incrementan simultáneamente después del procesamiento de cada entrada de la lista LIP

mientras ningún elemento deba ser eliminado. Cuando se encuentra un elemento que debe ser eliminado de la lista, el puntero de escritura no se incrementa, pero el de lectura sí. A partir de ese momento, los elementos que deben permanecer en la lista LIP son reescritos en la posición a la que apunta el puntero de escritura, ocupando el lugar del elemento eliminado (ver figura 4.8).

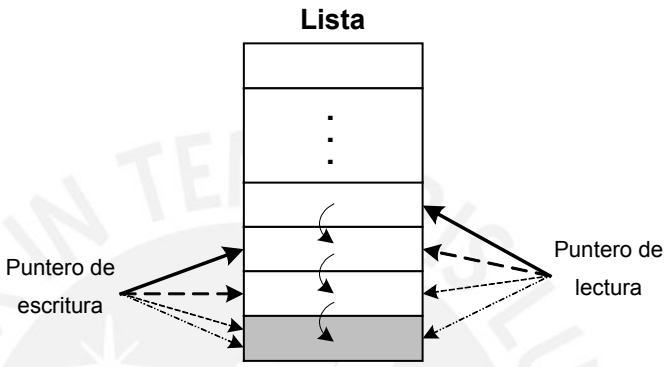


FIGURA 4.8.- FUNCIONAMIENTO DE LOS PUNTEROS DOBLES

En el caso de la lista LIS, además del puntero de lectura y escritura, existe un puntero de escritura especial para el rellenado de huecos. La razón de este puntero, es que durante el procesamiento de LIS, puede ser necesaria la escritura de algunas nuevas entradas al final de la lista, por lo que el puntero de escritura debe estar más adelantado que el puntero de lectura. De forma similar al caso de LIP, mientras ningún elemento deba ser eliminado de LIS, tanto el puntero de lectura como el puntero de rellenado de huecos son incrementados simultáneamente después del procesamiento de cada entrada de la lista LIS. Cuando se encuentra un elemento que debe ser eliminado de la lista, el puntero de rellenado de huecos no se incrementa y a partir de ese momento los elementos que deben permanecer en la lista LIS, son reescritos en la posición a la que apunta el puntero de rellenado de huecos, ocupando el lugar del elemento eliminado.

4.3.6.2 DESARROLLO

El módulo de punteros se encuentra descrito en PROC_CODEC_POINTERS.VHD (ver la sección Anexos A-2), y para su desarrollo se utilizan principalmente contadores (para crear los punteros requeridos). Además, se utilizan registros para almacenar el tamaño de las listas y flip-flop's para los habilitadores de relleno de huecos. El plano 2-5 de la sección Planos muestra los circuitos que conforman lo anteriormente mencionado.

Este módulo no presenta una unidad de control, ya que todas las señales necesarias para el control de estos elementos, vienen desde los distintos módulos del CODEC. La tabla 4.9 muestra los parámetros y señales de entrada/salida del módulo de punteros presentado en el plano 2-5 de la sección Planos.

Parámetro	**Función**	
lip_size_width	Indica el tamaño en bits de LIP. Por defecto es 19, lo cual representa 2^{19} elementos en LIP como máximo.	
lis_size_width	Indica el tamaño en bits de LIS. Por defecto es 19, lo cual representa 2^{19} elementos en LIS como máximo.	
lsp_size_width	Indica el tamaño en bits de LSP. Por defecto es 19, lo cual representa 2^{19} elementos en LSP como máximo.	
stream_size_width	Indica el tamaño en bits de la trama codificada. Por defecto es 20, lo cual representa 2^{20} posiciones de memoria interna para la trama codificada.	
Señal de entrada	**Función**	**Número de bits**
Aclr_indicelip_wr	Señal de reinicio del puntero de escritura de LIP.	1
en_indicelip_wr	Señal de incremento del puntero de escritura de LIP.	1
ld_lip_size	Señal de carga del registro que indica el tamaño de LIP.	1
Aclr_indicelip_rd	Señal de reinicio del puntero de lectura de LIP.	1
en_indicelip_rd	Señal de incremento del puntero de lectura de LIP.	1
en_hab_lip_wr	Señal de carga del flip-flop que habilita el rellenado de huecos en LIP.	1
clr_hab_lip_wr	Señal de borrado del flip-flop que habilita el rellenado de huecos en LIP.	1
ld_indicelis_wr	Señal de carga el puntero de escritura de LIS.	1
Aclr_indicelis_wr	Señal de reinicio del puntero de escritura de LIS.	1
en_indicelis_wr	Señal de incremento del puntero de escritura de LIS.	1
aclr_indicelis *wr*fix	Señal de reinicio del puntero de rellenado de huecos de LIS.	1
en_indicelis *wr*fix	Señal de incremento del puntero de rellenado de huecos de LIS.	1
ld_lis_size	Señal de carga del registro que indica el tamaño de LIS.	1

sel_mux_lis_wr	Selector del multiplexor mux_lis_wr.	1
Aclr_indicelis_rd	Señal de reinicio del puntero de lectura de LIS.	1
en_indicelis_rd	Señal de incremento del puntero de lectura de LIS.	1
en_hab_lis_wr_fix	Señal de carga del flip-flop que habilita el rellenado de huecos en LIS.	1
clr_hab_lis_wr_fix	Señal de borrado del flip-flop que habilita el rellenado de huecos en LIS.	1
Aclr_indicelsp_wr	Señal de reinicio del puntero de escritura de LSP.	1
en_indicelsp_wr	Señal de incremento del puntero de escritura de LSP.	1
ld_lsp_pasada _anterior	Señal de carga del registro que indica el tamaño de LSP para la próxima pasada.	1
Aclr_indicelsp_rd	Señal de reinicio del puntero de lectura de LSP.	1
en_indicelsp_rd	Señal de incremento del puntero de lectura de LSP.	1
aclr_indice_ trama_wr	Señal de reinicio del puntero de escritura de la memoria interna Trama.	1
en_indice *trama*wr	Señal de incremento del puntero de escritura de la memoria interna Trama.	1
aclr_indice *trama*rd	Señal de reinicio del puntero de lectura de la memoria interna Trama.	1
en_indice *trama*rd	Señal de incremento del puntero de lectura de la memoria interna Trama.	1
clock	Señal de sincronismo del módulo.	1
Señal de salida	**Función**	**Número de bits**
indice_lip_wr	Puntero de escritura de LIP.	lip_size_width
lip_size	Registro que indica el tamaño de LIP.	lip_size_width
indice_lip_rd	Puntero de lectura de LIP.	lip_size_width
Hab_lip_wr	Flip-flop que habilita el rellenado de huecos en LIP.	1
addr_lis_wr	Salida del multiplexor de punteros para escritura de LIS.	lis_size_width
lis_size	Registro que indica el tamaño de LIS.	lis_size_width
indice_lis_rd	Puntero de lectura de LIS.	lis_size_width
Hab_lis_wr_fix	Flip-flop que habilita el rellenado de huecos en LIS.	1
indice_lsp_wr	Puntero de escritura de LSP.	lsp_size_width
lsp_pasada _anterior	Registro que indica el tamaño de LSP para la próxima pasada.	lsp_size_width
indice_lsp_rd	Puntero de lectura de LSP.	lsp_size_width
indice_trama_wr	Puntero de escritura de la memoria interna Trama.	stream_size_ width
indice_trama_rd	Puntero de lectura de la memoria interna Trama.	stream_size_ width

TABLA 4.9.- PARÁMETROS Y SEÑALES DE ENTRADA Y SALIDA DEL MÓDULO DE PUNTEROS

4.3.6.3 RESULTADOS INDEPENDIENTES

El módulo fue compilado utilizando los valores 7, 8 y 9 para el parámetro isize_width (para imágenes de 128x128, 256x256, y 512x512 respectivamente).

Fueron realizadas compilaciones para los 3 tamaños de imágenes obteniéndose en todos los casos los mismos resultados. El módulo diseñado ocupa 196 elementos lógicos (< 1% de LE´s del FPGA utilizado), y alcanza una frecuencia máxima de trabajo de 375.09 MHz.

4.3.7 MÓDULO INTERFAZ CODEC-CONTROLADOR DE MEMORIA

4.3.7.1 COMPORTAMIENTO FUNCIONAL

El módulo interfaz CODEC-Controlador de Memoria administra los accesos al módulo de memoria externa por parte del CODEC. Se comporta a modo de un conmutador que crea enlaces entre los módulos que conforman el CODEC y el controlador de memoria, evitando de esta forma la colisión de datos y/o señales de control. Este conmutador está controlado por la señal “selector” proveniente del árbitro (ver CODEC_DDR_INTERFACE.VHD en la sección Anexos A-2). Con el fin de acceder a la sección apropiada del mapa de memoria, codifica las direcciones de lectura y escritura que los módulos del CODEC envían al controlador de memoria. Además, incluye ceros a los datos que serán escritos en la memoria externa, para completar los 128 bits del bus de datos del controlador de memoria.

Los seis módulos anteriormente descritos, son interconectados haciendo uso de multiplexores, comparadores, registros de desplazamiento y flip-flops; los cuales son mostrados en el plano 2-6 de la sección Planos y conforman el procesador CODEC, el cual se encuentra descrito en CODEC_TOP.VHD (ver en la sección Anexos A-2). Además, se realizó la compilación completa del procesador CODEC, para 3 tamaños de imágenes (128x128, 256x256 y 512x512 píxeles). La tabla 4.10 muestra los

resultados independientes de cada módulo anteriormente descrito y los resultados del Procesador CODEC completo:

	TAMAÑO DE IMAGEN	LEs	%LEs	fmax (MHz)
Inicialización	128 x 128	170	<1	236.46
	256 x 256	170	<1	236.46
	512 x 512	170	<1	236.46
LIP	128 x 128	87	<1	375.09
	256 x 256	87	<1	375.09
	512 x 512	87	<1	375.09
LIS	128 x 128	166	<1	422.12
	256 x 256	172	<1	409.33
	512 x 512	178	<1	381.66
LSP	128 x 128	35	<1	375.09
	256 x 256	35	<1	375.09
	512 x 512	35	<1	375.09
Punteros	128 x 128	196	<1	375.09
	256 x 256	196	<1	375.09
	512 x 512	196	<1	375.09
CODEC	128 x 128	1,182	< 4	181.82
	256 x 256	1,208	< 4	181.49
	512 x 512	1,234	< 4	180.45

TABLA 4.10.- RESULTADOS OBTENIDOS PARA EL BLOQUE PROCESADOR CODEC

4.4 ÁRBITRO DEL SISTEMA

4.4.1 COMPORTAMIENTO FUNCIONAL

El árbitro tiene entre sus funciones ser el control principal del sistema. Genera las señales de inicio y fin de todo el sistema, así como las señales de inicio de cada etapa del procesamiento. De esta forma controla qué módulos se activarán en

función al número de pasada del algoritmo y el modo de funcionamiento del sistema (codificación o decodificación). Además, interactúa con el controlador de memoria externa para la lectura de los parámetros de operación y la escritura de los resultados de ciclos de procesamiento y número de bits codificados. Por último, actúa como un codificador que recibe señales de control desde los módulos conformantes del CODEC generando las señales de selección que utiliza la Interfaz CODEC-Controlador de Memoria para gestionar los accesos a la memoria externa. Estas señales de control entrantes dependen del tipo de acceso a memoria externa (lectura o escritura) y de la sección del mapa de memoria a la cual los módulos intentan acceder.

4.4.2 DESARROLLO

El diseño del árbitro consta de una unidad de control y un codificador de 8 entradas. La unidad de control se basa en una máquina de estados como la mostrada en la figura 4.9. Durante los estados *rd_pasadas* y *rd_pasadas_e* se lee el número de pasadas que hará el sistema y se almacena en un contador. Durante los estados *rd_thres* y *ld_thres* se lee el valor umbral inicial de memoria externa y se almacena en un registro de desplazamiento. En el estado *thres_2_n* se convierte el valor umbral leído en la secuencia de bits “N” que encabezará la trama codificada. Durante todos los estados que se encuentran entre *thres_2_n* y *wr_bitcod* se da la codificación o decodificación en sí y se activa el contador de ciclos de procesamiento. Durante los estados *wr_bitcod y wr_bitcod_e* se escribe el número de bits de la trama codificada en la memoria externa. Durante los estados *wr_ciclos* y *wr_ciclos_e* se escribe el número de ciclos de procesamiento calculado en la

memoria externa. La tabla 4.11 muestra las señales de entrada y salida del módulo árbitro.

Señal de entrada	**Función**	**Número de bits**
en_lip_wr	Señal de habilitación de escritura en LIP.	1
en_lis_wr	Señal de habilitación de escritura en LIS.	1
en_lsp_wr	Señal de habilitación de escritura en LSP.	1
en_im_rec	Señal de habilitación de lectura y escritura en la imagen reconstruida.	1
en_lip_rd	Señal de habilitación de lectura de LIP.	1
en_lis_rd	Señal de habilitación de lectura de LIS.	1
en_lsp_rd	Señal de habilitación de lectura de LSP.	1
en_im_trans	Señal de habilitación de lectura de la imagen transformada.	1
Inicio	Señal inicio general del procesamiento.	1
cod_decod	Señal que indica si se Codifica (cod_decod = '1') o Decodifica (cod_decod = '0').	1
fin_gms	Señal de fin de procesamiento de GMS.	1
fin_pasada	Señal de fin de procesamiento de LSP.	1
hab_inipas	Flip-flop que habilita la ejecución de la Inicialización General.	1
fin_inig	Señal de fin de la Inicialización General.	1
pasadas_faltantes	Indica el número de pasadas del algoritmo que resta realizar.	4
r_valid	Señal que indica cuando el controlador de memoria externa tiene el dato listo para la lectura.	1
w_valid	Señal que indica cuando el dato ha sido aceptado por el controlador de memoria externa para la escritura.	1
rw_ack	Señal de reconocimiento de dirección desde el controlador de memoria externa.	1
hab_thres_2_n	Señal que indica el fin de la generación de N.	1
clock	Señal de sincronismo del módulo.	1
Señal de salida	**Función**	**Número de bits**
selector	Selector la sección de la memoria externa a la que se quiere acceder.	3
rst_hab_inipas	Señal de borrado del flip-flop que habilita la ejecución de la Inicialización General.	1
init_inipas	Señal de inicio de la Inicialización por Pasada.	1
init_inigen	Señal de inicio de la Inicialización General.	1
init_gms	Señal de inicio del procesamiento GMS.	1
ld_threshold_cod	Señal de carga del registro que almacena el valor umbral leído de memoria externa.	1
en_threshold_cod	Señal de desplazamiento del registro que almacena el valor umbral.	1
dec_pasada	Señal de habilitación del contador de pasadas.	1
gms_codec	Selector de los multiplexores que multiplexan el GMS y el CODEC.	1
sel_par	Selector de los multiplexores que multiplexan el procesamiento de la imagen y la lectura/escritura de parámetros y resultados.	1
en_par_addr	Señal que habilita el contador de direcciones de la lectura de parámetros.	1

en_pas	Señal de carga del registro que almacena el número de pasadas del algoritmo.	1
rd_req	Señal que indica al controlador de memoria externa la petición de lectura.	1
en_threshold _cod_n	Señal que habilita el desplazamiento del registro que genera N.	1
en_cont_ciclos	Señal que habilita la cuenta de ciclos de procesamiento.	1
w_req	Señal que indica al controlador de memoria externa la petición de escritura.	1
sel_resul	Selector del multiplexor que multiplexa los resultados a ser escritos en memoria externa.	1
en_resul_addr	Señal que habilita el contador de direcciones de la escritura de resultados.	1
hab_resul	Selector del multiplexor que multiplexan la dirección de lectura de parámetros y escritura de datos.	1
fin	Señal de finalización general del procesamiento.	1

TABLA 4.11.- PARÁMETROS Y SEÑALES DE ENTRADA Y SALIDA DEL ÁRBITRO

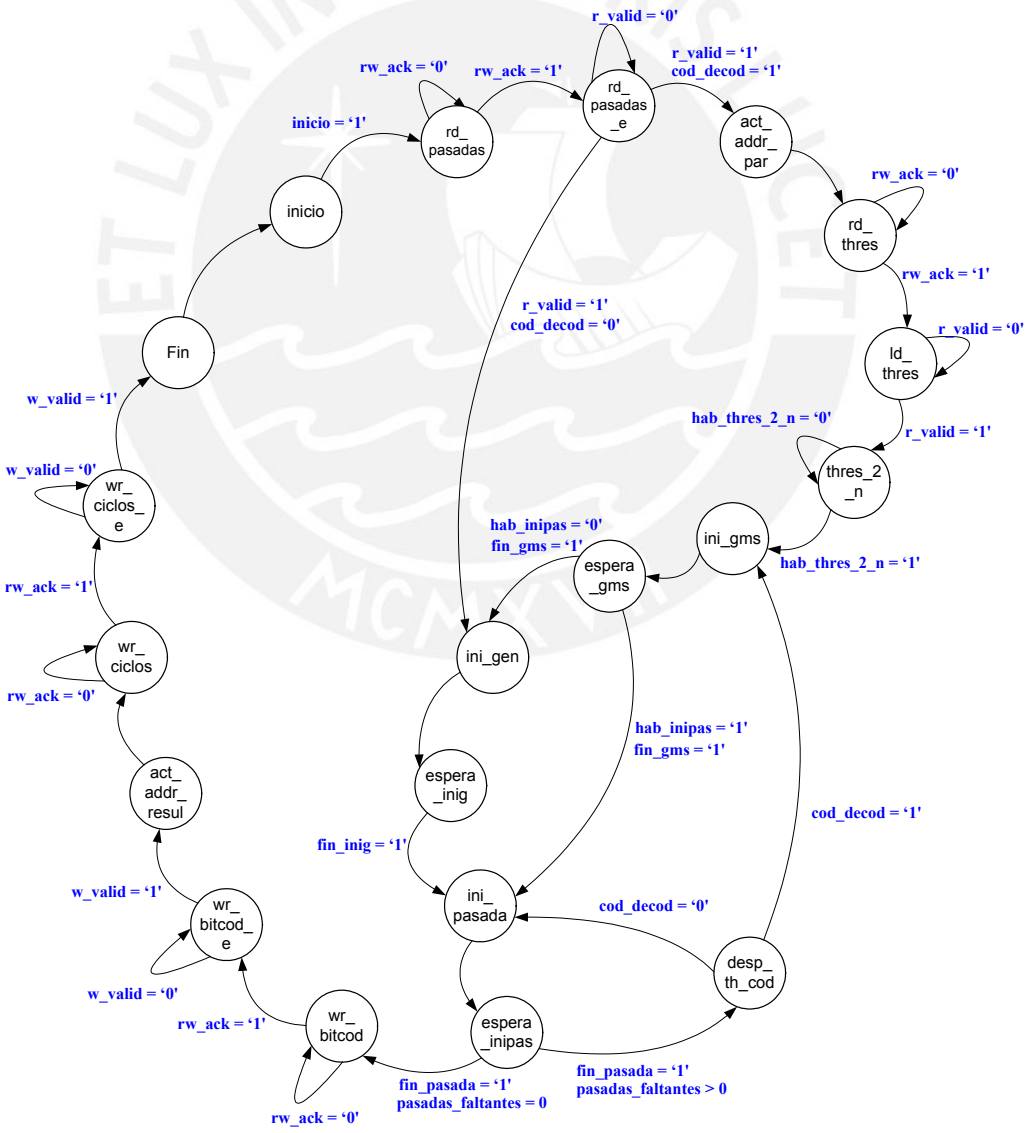


FIGURA 4.9.- DIAGRAMA DE ESTADOS DEL ÁRBITRO

4.4.3 RESULTADOS INDEPENDIENTES

El árbitro ocupa 45 elementos lógicos (<1% del total) en el FPGA utilizado, y alcanza una frecuencia máxima de trabajo de 375.09 MHz. Simulaciones de este módulo, tanto en modo codificación como en modo decodificación, son mostradas en el Anexo B.

5. IMPLEMENTACIÓN DE LA ARQUITECTURA

5.1 INTRODUCCIÓN

En el capítulo 4 fue descrito el diseño y funcionamiento de los módulos propios del presente trabajo de tesis, así como los resultados de las compilaciones de cada uno de ellos. En este capítulo es detallada la implementación de estos módulos sobre una plataforma específica, la cual también es descrita. En las secciones 5.2 y 5.3, son mencionadas las características más importantes de la tarjeta de desarrollo Stratix PCI y de los diseños de propiedad intelectual de Altera que fueron utilizados para la interacción con el bus PCI y con el módulo de memoria externa de la tarjeta de desarrollo. La sección 5.4 muestra aspectos importantes acerca de cómo fue implementado el sistema, teniendo en cuenta las características de la tarjeta de desarrollo.

5.2 RECURSOS HARDWARE

El diseño fue implementado y probado sobre la tarjeta de desarrollo Stratix-PCI de Altera, la cual cuenta con un FPGA STRATIX EP1S25F1020C5[13]. Esta tarjeta es una plataforma de evaluación y desarrollo para interfaces de alta velocidad como: PCI, PCI-X, (DDR) SDRAM, y 10/100 Ethernet. Además, posee tres fuentes de reloj: dos osciladores (33.33 MHz y 100 MHz) y una entrada de reloj SMA.

El dispositivo Stratix y el conector PCI de la tarjeta son compatibles con el estándar PCI revisión 2.3 y PCI-X revisión 2.0 modo 1. Además, son compatibles con las funciones MegaCore de Altera: *pci_mt64, pci_mt32, pci_t64* y *pci_t32*, las cuales

sirven de interfaz entre cualquier sistema implementado en el dispositivo Stratix y el bus PCI de 32 o 64 bits.

5.2.1 FPGA STRATIX EP1S25F1020C5

El dispositivo FPGA STRATIX EP1S25F1020C5 instalado en la tarjeta de desarrollo, se conecta a todos los componentes de la tarjeta a través de interfaces apropiadas. Este dispositivo es configurado a través de dos métodos: Configuración desde un dispositivo de memoria Flash contenido en la tarjeta de desarrollo y configuración JTAG a través del cable ByteBlaster II. Además, presenta entre sus características más importantes:

- 25 660 elementos lógicos.
- Memoria TriMatrix™ conformada por 3 bloques RAM para implementar memorias internas y buffers FIFO, ofreciendo hasta 1 944 576 bits de RAM disponibles sin reducir recursos lógicos.
- 6 PLLs (cuatro PLLs realzados y dos PLLs rápidos).
- 106 pines de usuario de E/S.

La estructura de memoria interna TriMatrix™ en un dispositivo Stratix, ofrece memoria RAM sin reducir recursos lógicos y hasta 12 Tbits/seg de ancho de banda de memoria de dispositivo. Este sistema de memoria consiste en 3 tipos de bloques RAM altamente flexibles: Bloques RAM M512 (de 512 bits cada bloque), Bloques RAM M4K (de 4K bits cada bloque) y Bloques M-RAM (de 512K bits cada bloque). El dispositivo FPGA STRATIX EP1S25F1020C5 tiene la organización de memoria que muestra la figura 5.1.

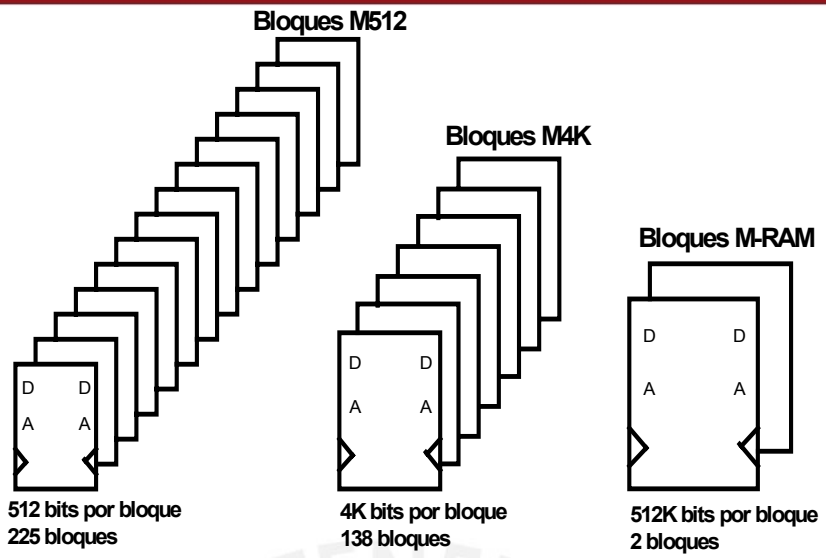


FIGURA 5.1. ORGANIZACIÓN DE BLOQUES DE MEMORIA INTERNA DEL FPGA STRATIX EP1S25F1020C5

Los bloques TriMatrix™ implementan tres modos de acceso a memoria:

- Memoria de Puerto Único: Este modo de acceso soporta escrituras y lecturas no simultaneas.
- Memoria Simple de Puerto Doble: Soporta escrituras y lecturas simultaneas a través de puertos de direccionamiento separados e independientes.
- Memoria True de Puerto Doble: Soporta combinaciones de dos operaciones: dos escrituras, dos lecturas, una lectura y una escritura simultáneamente a dos diferentes frecuencias de reloj.

Los recursos de memoria interna RAM del FPGA fueron utilizados para implementar los Mapas de Significancias generados por el procesador GMS, como será descrito en la sección 5.4.3, evitando de esta manera la utilización de recursos lógicos por parte de estos mapas.

Los dispositivos Stratix tienen múltiples PLLs con características avanzadas para una completa administración de reloj. Existen dos tipos de PLLs dentro del dispositivo Stratix: PLLs realzados (4 en el dispositivo EP1S25) y PLLs rápidos (2 en el dispositivo EP1S25).

Los PLLs rápidos son PLLs de propósito general que ofrecen administración de reloj, multiplicación y desplazamiento de fase y salidas de alta velocidad para administrar las “interfases de E/S diferencial de alta velocidad”. Los PLLs realzados, además de la administración de reloj, presentan posibilidades avanzadas como realimentación externa, conmutación entre relojes de entrada, control de retardo y fase, reconfiguración, espectro disperso y ancho de banda programable.

El software de diseño Quartus II de Altera incluye un conjunto de herramientas que permiten un rápido prototipado tanto de los requerimientos de memoria interna, como de los PLL’s a usar. La Megafunción *altsyncram* soporta las funciones de memoria ROM y RAM de uno o varios puertos para la arquitectura Stratix, e implementa estas funciones en los bloques de memoria TriMatrix™. Las memorias RAM y ROM de los bloques M512 y M4K pueden tener un contenido inicial, haciendo uso de los archivos de inicialización. Un archivo de inicialización es un archivo de texto ASCII con extensión .mif o .hex en el cual se especifica el valor contenido en cada posición de memoria. Este archivo es usado durante la compilación y/o simulación del proyecto. La Megafunción *altpll* implementa la descripción de un PLL en un dispositivo Stratix, en cualquiera de los dos tipos de PLL internos.

5.2.2 MÓDULO DE MEMORIA EXTERNA DDR SDRAM

La tarjeta de desarrollo posee un módulo de memoria DDR SDRAM PC333 de 256Mbytes, instalado en el conector de 200 pines SODIMM que a su vez esta conectado al dispositivo Stratix. La arquitectura de doble tasa de datos (DDR) del módulo de memoria, está diseñada para transferir dos palabras de datos por ciclo de reloj por sus pines de E/S. Opera con un reloj diferencial proveniente del dispositivo Stratix de la tarjeta de desarrollo, cuya frecuencia de 133.33 MHz es generada por un PLL interno.

Este módulo de memoria está configurado internamente como una DRAM de 4 bancos. A su vez, cada banco está dividido en 8192 filas y cada fila, en 1024 columnas de 64 bits cada una, como muestra la figura 5.2.

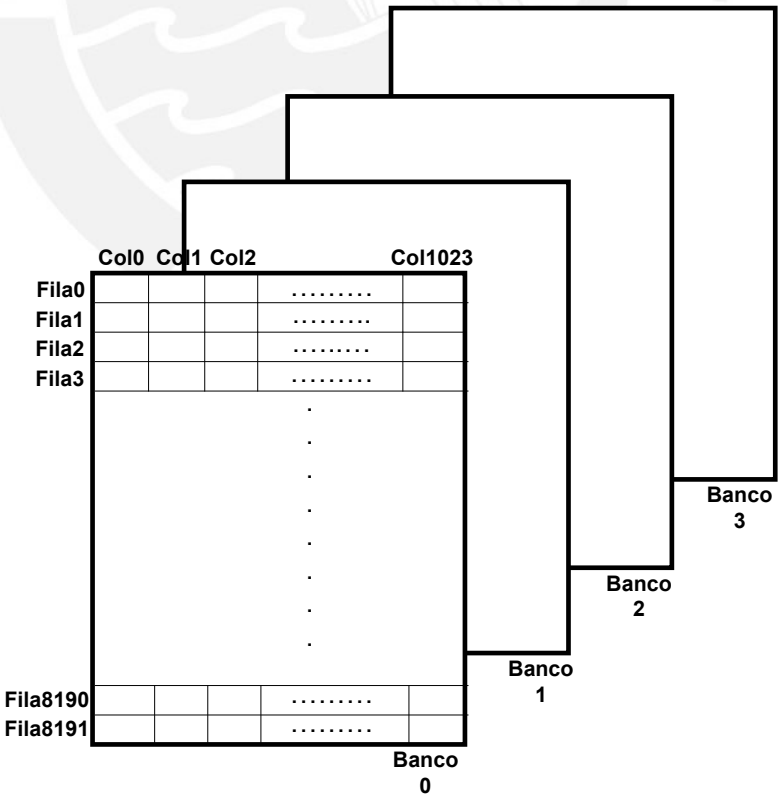


FIGURA 5.2. ORGANIZACIÓN INTERNA DEL MÓDULO DE MEMORIA EXTERNA

5.3 INTERFAZ PCI

La transmisión de datos entre la PC y la tarjeta de desarrollo, se realiza utilizando el diseño de referencia incluido en el kit de desarrollo Stratix PCI (Stratix PCI Development Kit Application). De esta forma, el origen o destino de los datos en la tarjeta es siempre el módulo de memoria DDR SDRAM. La interfaz de comunicación se completa con las funciones MegaCore de Altera: *pci_mt64* y "DDR SDRAM Controller".

5.3.1 FUNCIÓN MEGACORE PCI

La función MegaCore PCI de Altera [14] maneja el protocolo PCI y los requerimientos de tiempo internamente. La interfaz usada para la lógica de usuario, está diseñada para una fácil integración entre un sistema propio y el bus PCI de 32 o 64 bits. Existen cuatro funciones MegaCore de Altera: *pci_mt32, pci_t32, pci_t64* y *pci_mt64*, siendo esta última la utilizada en el presente trabajo de tesis.

En la función *pci_mt64*, las interfaces de maestro o target pueden operar independientemente, permitiendo la máxima eficiencia del bus PCI. La función permite transacciones de configuración, E/S y memoria. Pueden correr a las velocidades de reloj PCI de 33 y 66 MHz, alcanzando 132 Mbps en un sistema PCI de 32 bits/33 MHz y hasta 528 Mbps en un sistema PCI de 64 bits/66 MHz.

5.3.2 FUNCIÓN DDR-SDRAM MEGACORE CONTROLLER

La función MegaCore “DDR SDRAM Controller“ de Altera provee una interfaz simplificada a un módulo de memoria DDR SDRAM estándar. Maneja los complejos aspectos de utilizar una memoria DDR SDRAM tales como: inicializarla, administrar los bancos, y refrescarla a intervalos apropiados. Este controlador traduce las peticiones de lectura y escritura del lado local en los comandos necesarios para controlar la memoria.

El Controlador DDR SDRAM está formado por un módulo de control y por uno o más módulos de camino de datos, gobernados por una señal de reloj diferencial, como se observa en la figura 5.3.

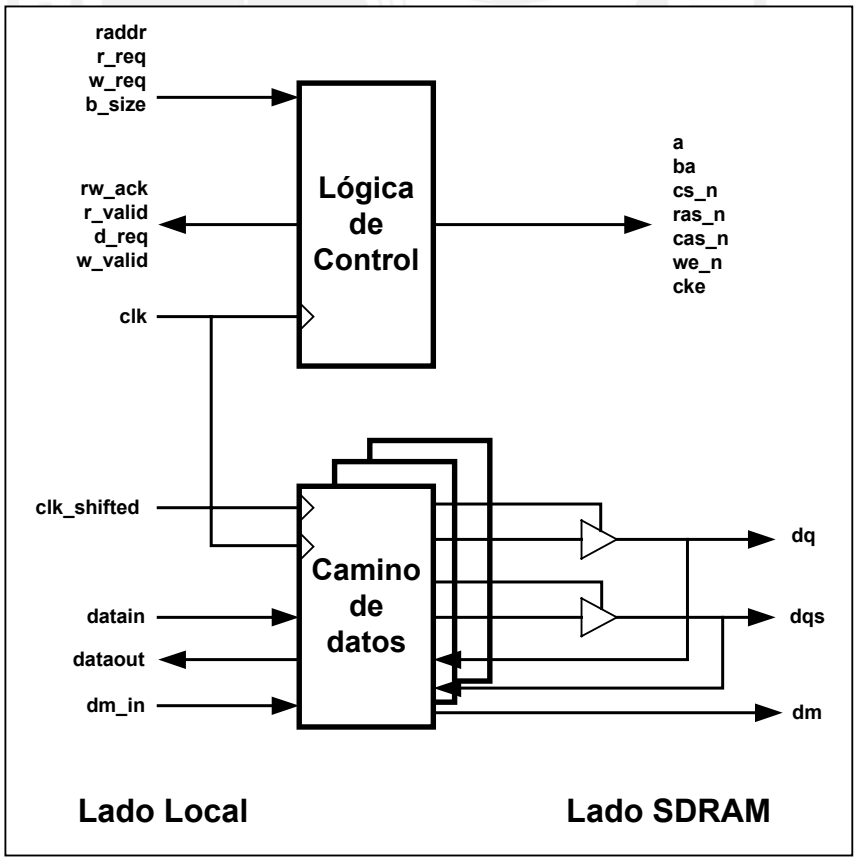


FIGURA 5.3. DIAGRAMA DE BLOQUES DEL CONTROLADOR DE MEMORIA

La tabla 5.1 muestra una descripción de las señales del controlador de memoria, tanto para el lado local como para el lado SDRAM.

Señales Locales		
Señal de entrada	**Función**	**Número de bits**
Clk	Reloj del sistema.	1
clk_shifted	Versión desplazada 90° de Clk que es usada para la escritura de datos SDRAM.	1
reset_n	Reset del sistema	1
Raddr	Dirección de lectura / escritura. Tiene toda la información acerca de el banco, fila, columna y chip donde debe apuntar el controlador, y esta ordenado de la siguiente forma: **raddr = (chip, fila, banco, columna)**	25
b_size	Tamaño de ráfaga. Por esta señal el lado local especifica el tamaño de su transmisión tipo ráfaga.	3
r_req	Petición de lectura.	1
wr_req	Petición de escritura.	1
Datain	Bus de datos de escritura	128
dm_in	Máscara de datos. Enmascara bytes durante la escritura de datos.	16
Señal de salida	**Función**	**Número de bits**
rw_ack	Reconocimiento de lectura/escritura. Reconoce la presente petición de lectura o escritura.	1
r_valid	Datos de lectura validos.	1
w_valid	Datos de escritura validos.	1
d_req	Petición de datos. Indica al bus del lado local que debe presentar datos en el siguiente flanco de reloj.	1
Dataout	Bus de datos de lectura.	128
Señales del Lado SDRAM		
Señal bidireccional	**Función**	**Número de bits**
Dq	Bus de datos SDRAM.	64
Dqs	Habilitador de datos SDRAM.	8
Señal de Salida	**Función**	**Número de bits**
A	Bus de direcciones.	13
Ba	Dirección de banco.	2
cs_n	Selección de chip.	2
ras_n	Comando habilitador de dirección de fila.	1
cas_n	Comando habilitador de dirección de columna.	1
we_n	Comando de habilitación de escritura.	1
Cke	Habilitación de reloj SDRAM.	1
Dm	Mascara de datos SDRAM.	8

TABLA 5.1. SEÑALES DEL CONTROLADOR DE MEMORIA

Los módulos SDRAM son controlados utilizando comandos, los cuales son interpretados a partir de la combinación de las señales *ras_n*, *cas_n* y *we_n*. La tabla 5.2 muestra los comandos SDRAM típicos.

Comando	Acrónimo	ras_n	cas_n	we_n
Sin operación	NOP	H	H	H
Activación	ACT	L	H	H
Lectura	RD	H	L	H
Escritura	WR	H	L	L
Fin de Ráfaga	BT	H	H	L
Pre-Carga	PCH	L	H	L
Auto-Refresco	ARF	L	L	H
Carga del Registro de Modo	LMR	L	L	L

TABLA 5.2. COMANDOS DE MEMORIA DDR SDRAM

El módulo de memoria DDR SDRAM debe ser activado e inicializado de una manera especifica [15]. De esta tarea se encarga el controlador de memoria debiendo esperar cualquier otro sistema a que termine el proceso de inicialización para acceder al módulo de memoria.

5.3.3 DISEÑO DE REFERENCIA PCI A DDR-SDRAM

El diseño de referencia PCI a DDR SDRAM de Altera[16], provee una interfaz entre la función MegaCore *pci_mt64* y un módulo DDR SDRAM de 256MB y 64 bits, incluyendo la función MegaCore “DDR SDRAM Controller“. Este diseño presenta entre sus principales características:

- Soporte para transacciones PCI (master y target) de 32 y 64 bits.
- Uso de la función FIFO de puerto doble de la librería de módulos parametrizables (LPM)

- Uso de la función MegaCore "DDR SDRAM Controller"
- Incluye una lógica de medición del rendimiento PCI
- Además, sirve de interfaz entre PCI y la memoria Flash

El diseño de referencia está incluido en el Kit de desarrollo Stratix PCI, como un proyecto cuyo archivo de mayor nivel incluye instancias de la funciones MegaCore *pci_mt64* y DDR-SDRAM Controller. La figura 5.4 muestra un diagrama de bloques de la interfaz PCI contenida en este proyecto.

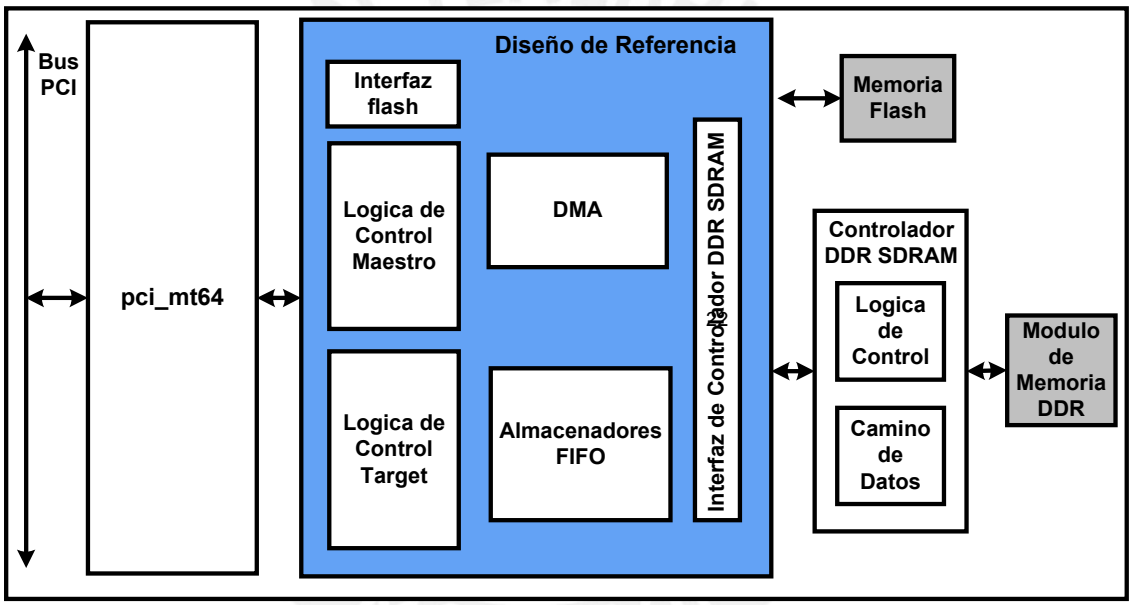


FIGURA 5.4. DIAGRAMA DE BLOQUES DE LA INTERFAZ PCI A DDR SDRAM

El diseño de referencia está constituido por los siguientes bloques:

- **Lógica de control de Modo Master:** La lógica de control de modo master controla la operación de la función *pci_mt64* en modo master, interactuando con el bloque DMA.
- **Lógica de control de Modo Target:** La lógica de control de modo target controla la operación de la función *pci_mt64* en modo target.

- **DMA:** El DMA interactúa con la lógica de control de modo master, los buffers FIFO en el camino de datos y la interfaz de controlador DDR SDRAM para coordinar transferencias DMA hacia y desde la memoria DDR.
- **Buffers FIFO en el camino de datos:** Sirven como un espacio de almacenamiento para los datos transferidos entre la memoria DDR SDRAM y el bus PCI, y sincronizan los datos que cruzan del dominio de reloj PCI al dominio de reloj DDR.
- **Interfaz de controlador DDR SDRAM:** Es la conexión hacia el lado local de la función MegaCore "DDR SDRAM Controller". El módulo de memoria DDR es de 64 bits de datos y requiere que la interfaz de usuario al controlador de memoria sea de 128 bits. Sin embargo el diseño de referencia trabaja con un bus PCI de 64 bits, por esto, sólo están conectados los 32 bits menos significativos de datos del módulo de memoria DDR. Lo que quiere decir que el diseño de referencia sólo usa 128MB de los 256MB del módulo de memoria DDR. En cada posición de memoria, 32 de los 64 bits son completados con ceros.

Este diseño original fue modificado con la finalidad que el controlador de memoria pueda ser utilizado también por la lógica diseñada para el procesamiento de la imagen. La modificación consistió en la separación del controlador de memoria y el corte de las señales y buses que lo unen al diseño de referencia. Entre estos dos bloques se introdujo un bloque llamado Mux/Demux para el Controlador de Memoria, el cual será descrito en la sección 5.4, encargado de la multiplexación de los accesos a memoria externa, cediendo el uso del controlador a uno de estos bloques. Un diagrama de bloques de la interfaz PCI con esta modificación es mostrado en la figura 5.5

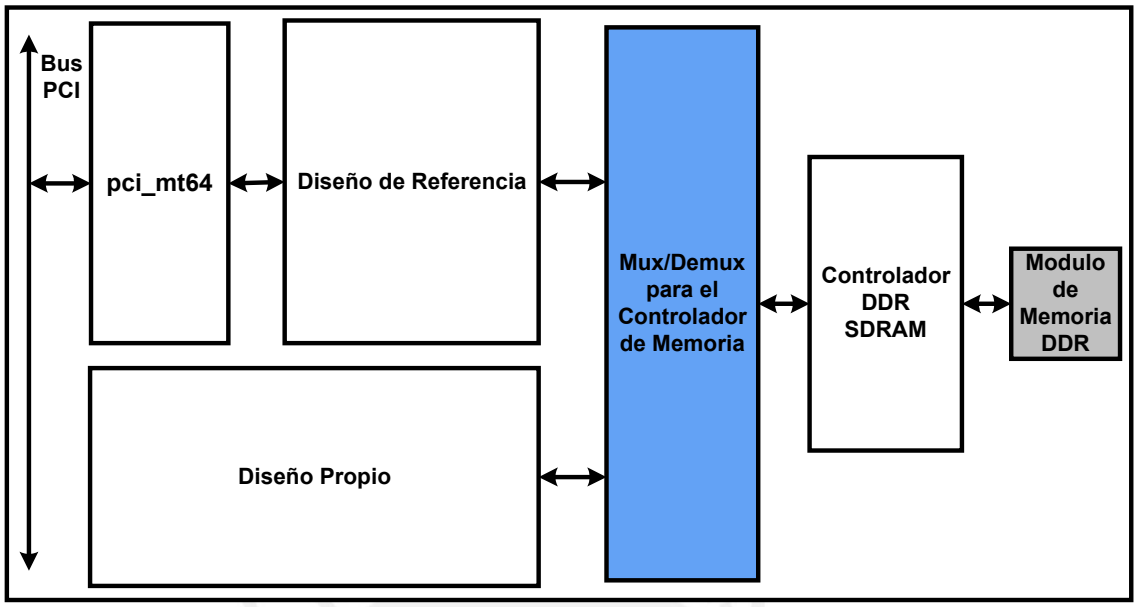


FIGURA 5.5. DIAGRAMA DE BLOQUES DEL DISEÑO DE REFERENCIA MODIFICADO

La función *pci_mt64* que utiliza el diseño de referencia implementa dos registros de dirección base (BAR): BAR0, que reserva 1MB del espacio de direcciones del sistema; y BAR1, que reserva 128MB del espacio de direcciones del sistema. BAR0 apunta a los registros de configuración internos del diseño de referencia, mientras que BAR1 apunta al módulo de memoria DDR SDRAM. La tabla 5.3 detalla este espacio de direcciones de memoria.

Región de Memoria	Tamaño del Bloque	Desplazamiento de Dirección	Descripción
BAR0	1Mbyte	00000h-FFFFFh	Registros de configuración internos del diseño de referencia
BAR1	128Mbyte	0000000h-7FFFFFFh	Modulo de Memoria DDR

TABLA 5.3. MAPEO DE MEMORIA DE LA FUNCIÓN PCI_MT64

5.4 IMPLEMENTACIÓN DEL SISTEMA

5.4.1 MUX/DEMUX PARA EL CONTROLADOR DE MEMORIA

Debido a que se cuenta con sólo un módulo de memoria externa que debe ser compartido por la interfaz PCI y por los diseños propios del codificador/ decodificador, se optó por diseñar una interfaz que sea capaz de gestionar el uso del controlador de memoria externa para evitar la colisión de datos y señales de control.

Este módulo se encarga de multiplexar los accesos por parte de los procesadores desarrollados y el diseño de referencia al módulo de memoria externa DDR SDRAM. Es decir, dependiendo de la señal *sel* (ver MUX_DEMUX.VHD), todas las señales del controlador de memoria estarán conectadas ya sea a la interfaz PCI o a los procesadores diseñados. Las señales que entran al controlador de memoria se multiplexan mientras que las que salen se demultiplexan.

5.4.2 ORGANIZACIÓN DE MEMORIA EXTERNA

La memoria externa fue dividida en seis bloques para cumplir con los requerimientos de almacenamiento de las listas de coordenadas (LIP, LIS y LSP) y las imágenes, de la siguiente forma:

I. Almacenamiento de la Imagen transformada al dominio wavelet (Imagen a codificar).

II. Almacenamiento de la Imagen reconstruida (resultado de la decodificación).
III. Lista de Píxeles Insignificantes (LIP).
IV. Lista de Píxeles Significantes (LSP).
V. Lista de Sets Insignificantes (LIS).
VI. Almacenamiento de parámetros (valor umbral y número de pasadas) y de resultados (tiempo de procesamiento y bits codificados).

Las dos primeras zonas de memoria dependen de un parámetro del sistema llamado *isize_width*, que indica la potencia de dos que da como resultado una de las dimensiones de la imagen transformada o de la imagen reconstruida. Es decir, el tamaño de las imágenes tanto de entrada como de salida son: 2^{isize_width} x 2^{isize_width} píxeles (ya que son imágenes cuadradas). Por ejemplo, si se procesa una imagen transformada de 512x512 píxeles, ésta y su correspondiente imagen reconstruida tendrán un consumo total de 512K posiciones de memoria.

El tamaño máximo de imagen a procesar es de 512x512, lo cual implica que *isize_width* máximo es 9 ($isize\_width_{max}$). Además, debido al análisis realizado en MATLAB del algoritmo a implementar se pudo ver que el tamaño de la listas podía llegar a ser considerable; asi que para asegurar que no suceda el desbordamiento de dichas listas, se optó por usar el doble de memoria que consume la imagen transformada.

Por lo tanto, la cantidad de Memoria Externa DDR SDRAM a usar por el sistema desarrolado es de $7 \times 2^{2*isize\_width}$ posiciones de memoria. Para el caso máximo, la cantidad de memoria a usar es 14 Mbytes de los 256Mbytes disponibles, además de

los 40 bytes reservados para los parámetros y resultados (Bloque de memoria externa VI). Sin embargo, se debe considerar que debido a especificaciones del controlador de memoria, cada posición es de 16 Bytes (128 bits). En la figura 5.6 se muestra el mapa de memoria externa.

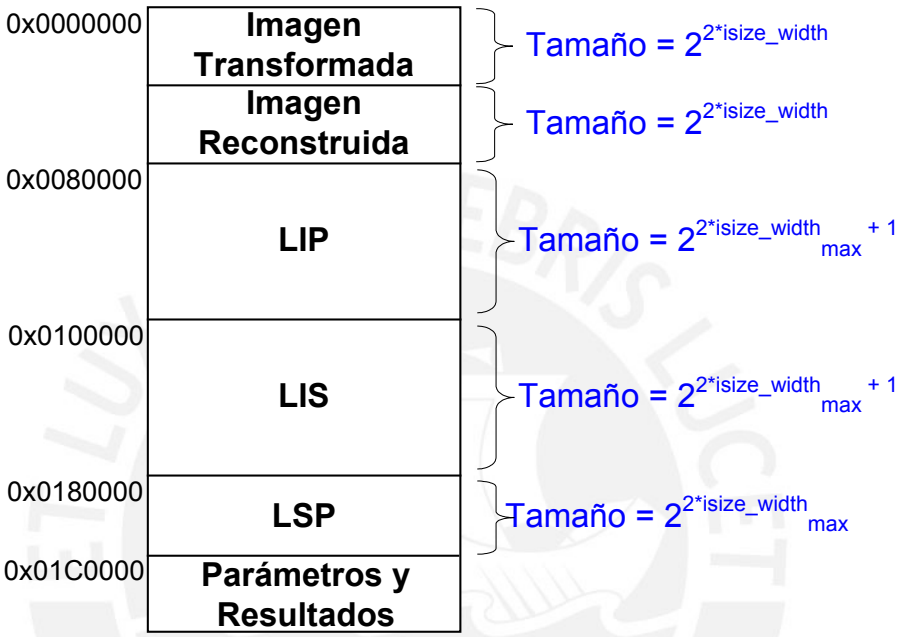


FIGURA 5.6. MAPA DE MEMORIA EXTERNA

5.4.3 UNIDADES DE MEMORIA INTERNA

El diseño hace uso de 3 unidades de memoria interna. En principio, el procesador Generador de Mapas de Significancias (GMS) entrega su resultado a dos bloques de memoria interna llamados Mapa D y Mapa L (con bus de datos de 1 bit). Y en el caso del procesador Codificador/Decodificador (CODEC) este hace uso de un bloque de memoria interna llamado trama (que posee la trama de bits codificados, es decir, con un bus de datos de un bit).

El tamaño de las memorias Mapa D y Mapa L dependen directamente del tamaño de la imagen "*isize_width"*, siendo el tamaño del Mapa D, la cuarta parte del número de coeficientes de la imagen transformada y el tamaño del Mapa L la dieciseisava parte del número de coeficientes de la imagen transformada (según análisis del algoritmo GMS). El tamaño de la trama se ha elegido en función a un parámetro llamado "*stream_size_width"*, que según resultados de pruebas experimentales fue fijado en 10 bits. Estos parámetros, definen el ancho del bus de direcciones.

Las memorias usadas para el Mapa D y la trama fueron del tipo "Memoria True de Puerto Doble" y la usada para el Mapa L fue del tipo "Memoria Simple de Puerto Doble", y ambas, fueron creadas utilizando la Megafunción *lpm_altsyncram*. La figura 5.7 muestra los bloques de memoria para los Mapas de Significancias tal como fueron implementados.

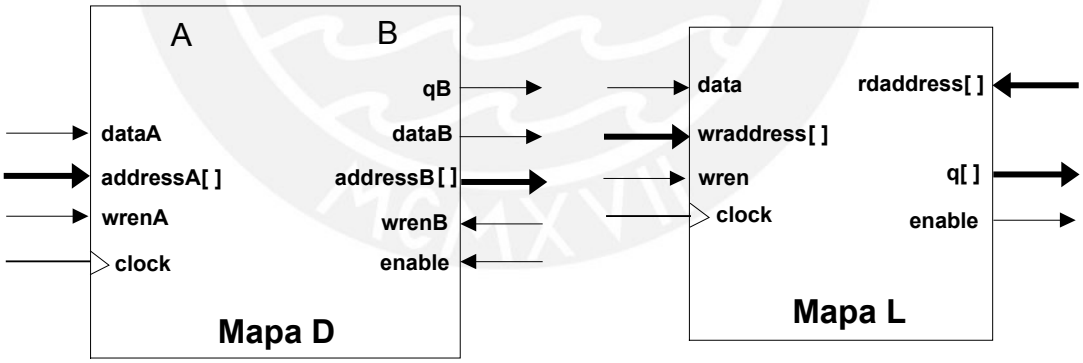


FIGURA 5.7 MEMORIAS IMPLEMENTADAS PARA LOS MAPAS DE SIGNIFICANCIAS

5.4.4 GENERACIÓN DE LA SEÑAL DE RELOJ DEL SISTEMA

Dado que el oscilador de la tarjeta de desarrollo entrega una señal de reloj de 33.33 MHz, el diseño de referencia genera una frecuencia de reloj de 133.33 MHz haciendo uso de un PLL implementado con la Megafunción *altpll*. Esta es la frecuencia de reloj principal del diseño de referencia y fue tomada como frecuencia de trabajo del sistema total (según los resultados experimentales mostrados en el capitulo 4). La figura 5.8 muestra el PLL implementado para generar las señales de reloj necesarias a partir de la señal de reloj proveniente del oscilador de la tarjeta de desarrollo.

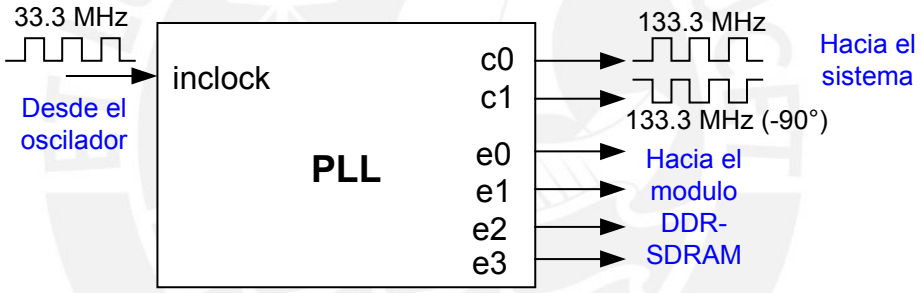


FIGURA 5.8. PLL IMPLEMENTADO EN EL SISTEMA

5.4.5 CONSUMO DE RECURSOS LÓGICOS DEL SISTEMA

La implementación del sistema se completa con algunos componentes adicionales incluidos en el archivo STRATIX_TOP.VHD, el cual es mostrado en la sección ANEXOS. Estos componentes adicionales incluyen multiplexores 2 a 1 para algunas señales del controlador de memoria que comparten los procesadores diseñados y los circuitos de lectura de parámetros y escritura de resultados en memoria externa. También incluyen un contador que se encarga de la cuenta de ciclos de

procesamiento y un circuito anti-rebote (ver PULSO_SW.VHD en la sección Anexos) conectado a la entrada del pulsador de inicio del sistema.

El consumo de celdas del sistema total, se obtuvo haciendo compilaciones para 3 tamaños de imagen (128x128, 256x256 y 512x512 píxeles). Estos resultados son mostrados en la tabla 5.4.

	TAMAÑO DE IMAGEN	LEs	%LEs	Bits de memoria utilizados	Número de pines utilizados
SISTEMA COMPLETO	128 x 128	6,944	< 27	23612(1%)	277
	256 x 256	6,987	< 27	38982(2%)	277
	512 x 512	7,076	< 27	100432(5%)	277

TABLA 5.4 CONSUMO TOTAL DE RECURSOS DEL SISTEMA

La figura 5.9 muestra una comparación del consumo de recursos entre cada procesador diseñado y el sistema total implementado. Se observa que los procesadores poseen un bajo porcentaje de consumo de celdas lógicas, siendo el diseño de referencia y los MegaCores, los bloques de más consumo en el sistema.

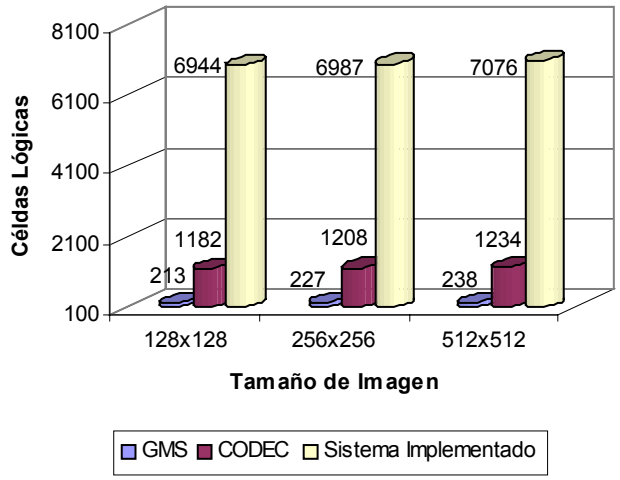


FIGURA 5.9.- GRÁFICO COMPARATIVO DEL CONSUMO DE CELDAS VS. TAMAÑO DE IMAGEN

6. RESULTADOS EXPERIMENTALES

6.1 INTRODUCCIÓN

Para realizar las pruebas del sistema se desarrolló una interfaz de usuario que permite la comunicación entre la PC y la tarjeta de desarrollo a través del Bus PCI. Esta interfaz es necesaria para establecer los parámetros iniciales del sistema y para visualizar los resultados entregados por el diseño realizado. Además, en esta sección se presentan los modos de operación del sistema desarrollado, los circuitos de prueba usados y el rendimiento obtenido por el sistema implementado.

6.2 INTERFAZ DE USUARIO

Este software fue desarrollado en Visual Basic 6.0 y posee un módulo controlador que gobierna la interacción con la tarjeta de desarrollo Stratix-PCI. El código del controlador fue generado automáticamente por un software que está incluido en el kit de desarrollo llamado Jungo-WinDriver, el cual evita al programador la complejidad del control temporal del protocolo PCI. Además, el software de interfaz de usuario posee una interfaz gráfica amigable y entre sus principales funciones, se encuentran:

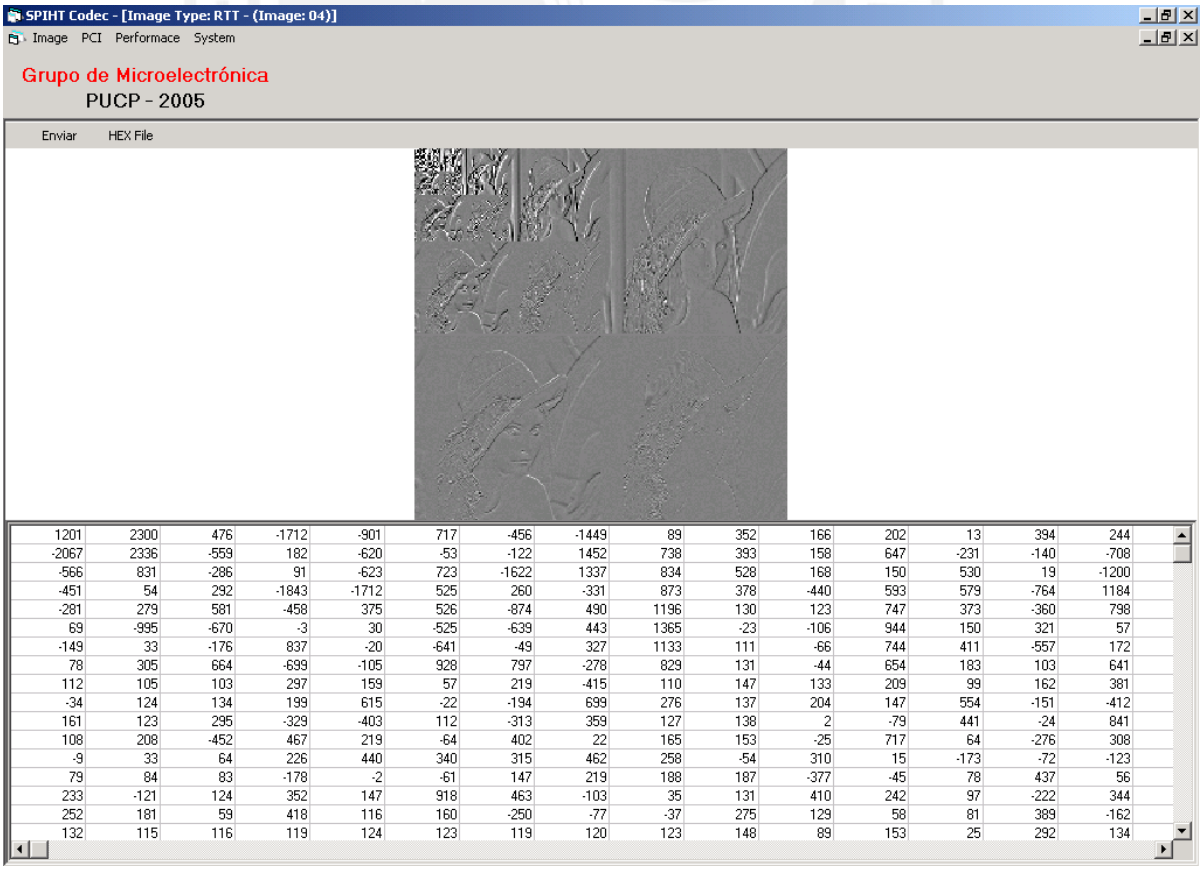
- Visualización de archivos de imágenes.
- Envío de la imagen transformada al dominio wavelet hacia el sector apropiado del mapa de memoria externa.
- Lectura del sector de la memoria externa donde se almacena la imagen reconstruida.
- Envío de parámetros de operación del sistema.

- Lectura de resultados de número de ciclos de procesamiento y número de bits codificados desde el sector de memoria externa reservado para estos valores.
- Calculo de rendimiento del sistema basándose en la imagen enviada y recibida.

La interfaz de usuario presenta una barra de menús donde es posible especificar la tarea a realizar. Esta barra de menús muestra las siguientes opciones:

- Menú "Image" : Contiene las opciones "Open RAW" y "Open RAW TT". La primera de ellas permite elegir una imagen original (sin transformar) almacenada en la memoria de la PC en formato *.raw* para ser abierta y visualizada en una ventana hija (ventana de imagen RAW). La segunda, permite elegir una imagen transformada al dominio wavelet, que se encuentra almacenada en la memoria de la PC en un formato propio cuya extensión es *.rtt* (raw de transformación truncada). Este formato ha sido creado en MATLAB (ver código en el Anexo E) y se caracteriza porque almacena en un archivo la imagen transformada con los coeficientes wavelet truncados. Esta imagen es visualizada en una segunda ventana hija (ventana de imagen RTT) junto a un visualizador de matriz que muestra los valores de los coeficientes wavelet truncados (ver figura 6.1).
- Menú "PCI" : Contiene la opción "Read PCI", la cual permite la lectura de la imagen reconstruida almacenada en la memoria externa de la tarjeta de desarrollo. Esta imagen es visualizada en una ventana hija (ventana de imagen Read PCI) junto a un visualizador de matriz que muestra los valores de los coeficientes Wavelet reconstruidos.

- Menú “Performance” : Contiene la opción “MSE & PRNR”, la cual abre un cuadro de diálogo, en el que se puede elegir dos de las ventanas hijas abiertas en un momento dado para compararlas (ver el cuadro de dialogo Select Matrix en la figura 6.2) y arrojar resultados de Error Cuadrático Medio (MSE) y Relación Señal a Ruido Pico (PSNR).
- Menú “System” : Contiene la opción “Config”, la cual abre el cuadro de diálogo “System Config”. En este cuadro de diálogo se pueden ingresar los parámetros de operación del sistema (valor umbral y número de pasadas), para luego escribirlos en la posición de la memoria externa reservada para este propósito. Además, se puede leer los resultados de número de ciclos de procesamiento tanto para codificación como para decodificación, y el número de bits codificados (ver figura 6.3).



1201	2300	476	-1712	-901	717	-456	-1449	89	352	166	202	13	394	244
-2067	2336	-559	182	-620	-53	-122	1452	738	393	158	647	-231	-140	-708
-566	831	-286	91	-623	723	-1622	1337	834	528	168	150	530	19	-1200
-451	54	292	-1843	-1712	525	260	-331	873	378	-440	593	579	-764	1184
-281	279	581	-458	375	526	-874	490	1196	130	123	747	373	-360	798
69	-995	-670	-3	30	-525	-639	443	1365	-23	-106	944	150	321	57
-149	33	-176	837	-20	-641	-49	327	1133	111	-66	744	411	-557	172
78	305	664	-699	-105	928	797	-278	829	131	-44	654	183	103	641
112	105	103	297	159	57	219	-415	110	147	133	209	99	162	381
-34	124	134	199	615	-22	-194	699	276	137	204	147	554	-151	-412
161	123	295	-329	-403	112	-313	359	127	138	2	-79	441	-24	841
108	208	-452	467	219	-64	402	22	165	153	-25	717	64	-276	308
-9	33	64	226	440	340	315	462	258	-54	310	15	-173	-72	-123
79	84	83	-178	-2	-61	147	219	188	187	-377	-45	78	437	56
233	-121	124	352	147	918	463	-103	35	131	410	242	97	-222	344
252	181	59	418	116	160	-250	-77	-37	275	129	58	81	389	-162
132	115	116	119	124	123	119	120	123	148	89	153	25	292	134

FIGURA 6.1.- VENTANA HIJA DE IMAGEN RTT EN LA INTERFAZ DE USUARIO

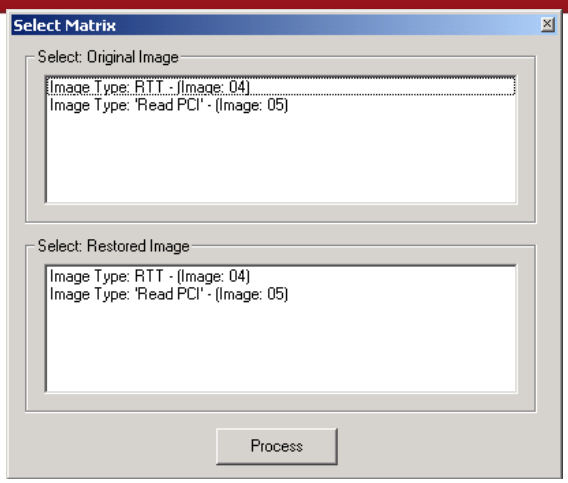


FIGURA 6.2.- CUADRO DE DIÁLOGO SELECT MATRIX

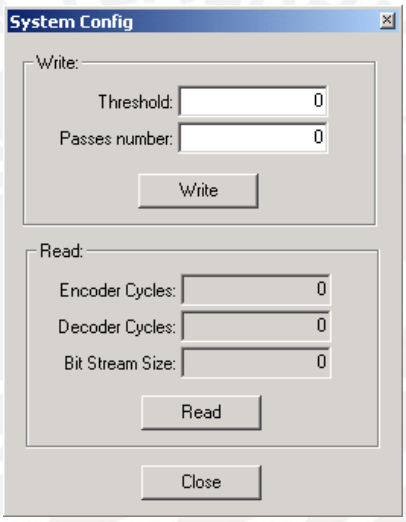


FIGURA 6.3.- CUADRO DE DIÁLOGO SYSTEM CONFIG

Tanto la ventana de imagen RTT como la ventana de imagen Read PCI presentan un botón llamado “Enviar” con el que se inicia la escritura de la imagen sobre la memoria externa de la tarjeta de desarrollo. Además, presentan un botón llamado “HEX File” con el que se puede visualizar los datos en formato hexadecimal, en el orden en que serán escritos en la memoria externa.

6.3 PROCESAMIENTO DE DATOS EN EL SISTEMA

Habiendo ya descrito cada uno de los bloques conformantes del sistema, en esta sección se muestran los modos de operación del sistema, es decir, el flujo de datos tanto para el proceso de codificación como para el proceso de decodificación.

6.3.1 CODIFICACIÓN

Para el proceso de codificación, una imagen transformada al dominio wavelet truncada debe ser procesada por la arquitectura del sistema, para entregar como resultado una trama de bits. Para ello, la secuencia de pasos es presentada a continuación:

i. La imagen a procesar es enviada desde la memoria del sistema de la PC hacia la tarjeta de desarrollo utilizando el software de interfaz de usuario. Esta imagen es almacenada en el sector reservado para la imagen transformada en la memoria externa de la tarjeta de desarrollo (ver sección 5.4.2 del capitulo 5).
ii. Una vez terminada la carga de la imagen, puede darse inicio al procesamiento de la misma. Para esto, antes de la primera pasada del algoritmo SPIHT, el sistema lee de memoria externa los valores de valor umbral y número de pasadas a realizar, los cuales almacena en un registro de desplazamiento y en un contador decreciente respectivamente.
iii. Luego, se inicia la primera pasada del algoritmo SPIHT con la generación de los mapas de significancia D y L, por parte del Procesador GMS.

iv. Seguidamente, el procesador CODEC en “Modo C” usa los mapas generados, iniciándose la codificación propiamente dicha. La imagen codificada se almacena como una trama de bits en la memoria interna para su posterior decodificación.

v. Al finalizar el procesamiento del CODEC, se da por concluida la primera pasada del algoritmo. Si el contador de pasadas no ha llegado a cero, se decrementa el valor umbral, y se inicia la siguiente pasada del algoritmo, retornando al paso iii. Si el contador de pasadas ha llegado a cero, se escribe el número de ciclos de procesamiento y el número de bits codificados en la memoria externa, dándose por terminado el procesamiento de codificación.

La figura 6.4 muestra el flujo de datos para cada pasada cuando el sistema se encuentra en el Modo Codificación.

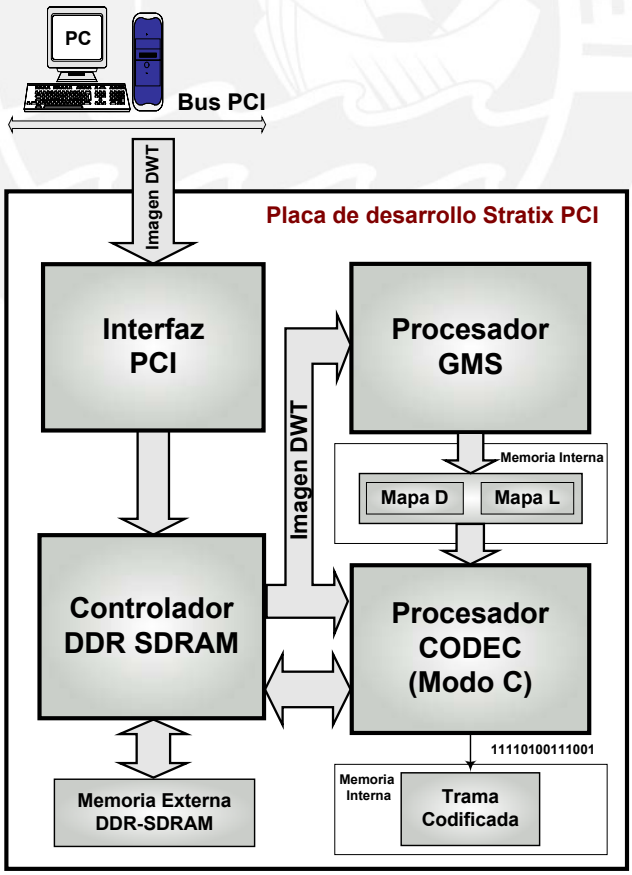


FIGURA 6.4. FLUJO DE DATOS DEL SISTEMA EN MODO CODIFICACIÓN

6.3.2 DECODIFICACIÓN

Para el proceso de decodificación, la trama de bits codificada almacenada en memoria interna es procesada por la arquitectura del sistema, para reconstruir la imagen transformada al dominio wavelet, siguiendo la secuencia de pasos que es presentada a continuación:

i. Al iniciarse el proceso de decodificación, y antes de la primera pasada del algoritmo, el sistema lee de la memoria externa el valor del número de pasadas a realizar y lo almacena en un contador decreciente.
ii. El procesador CODEC, trabajando en “Modo D”, procesa la trama codificada de la memoria interna y reconstruye continuamente la imagen transformada al dominio wavelet, almacenando este resultado en la memoria externa.
iii. Concluido el procesamiento del CODEC, se da fin a la primera pasada del algoritmo. Si esta no es la última pasada, el valor umbral es decrementado, volviendo al paso ii. Si el contador de pasadas llegó a cero, el número de ciclos de procesamiento es escrito en la memoria externa y dándose por terminado el proceso de decodificación.

La figura 6.5 muestra el flujo de datos para cada pasada cuando el sistema se encuentra en el Modo Decodificación.

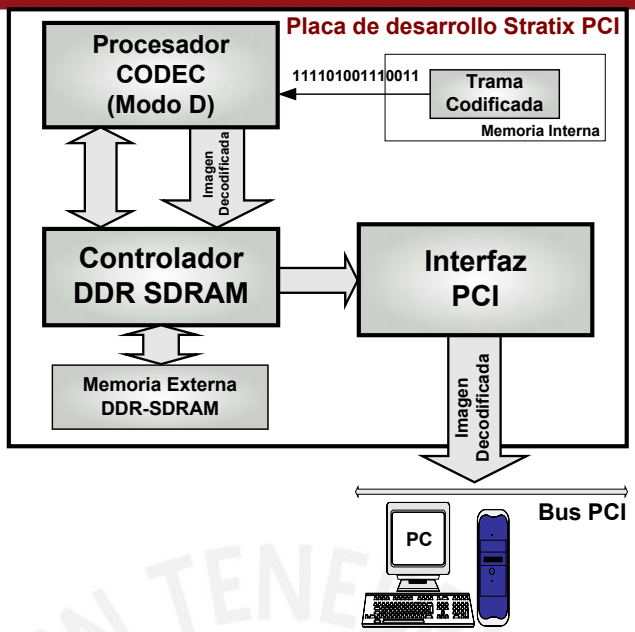


FIGURA 6.5. FLUJO DE DATOS DEL SISTEMA EN EL MODO DECODIFICACIÓN

6.4 PRUEBAS POR ETAPAS

Para validar el funcionamiento de cada una de las etapas del procesamiento en el FPGA, fueron hechas distintas pruebas con implementaciones parciales de dichas etapas. Para esto, fue utilizada la aplicación software del Kit de Desarrollo Stratix PCI de Altera (Stratix PCI Development Kit Application), la cual permite la escritura y lectura de cualquier grupo de posiciones de la memoria externa de la tarjeta de desarrollo. Esto fue particularmente útil debido a que las listas de coeficientes fueron implementadas en la memoria externa, y por tanto, pueden ser leídas en cualquier etapa del procesamiento para verificar la correcta generación de éstas. Todas las pruebas previas a la implementación final fueron hechas con imágenes ficticias, es decir, con una matriz de datos aleatorios, del mismo formato que las imágenes reales.

Inicialmente fue probado el acceso a la memoria externa de la tarjeta de desarrollo desde la PC. Para ello, utilizando la aplicación software del kit, fueron hechas escrituras de conjuntos de datos conocidos a ciertas posiciones de memoria, para luego, esas mismas posiciones ser leídas y comparar los datos obtenidos con los originales. Con esta prueba fue comprobado el correcto funcionamiento de la interfaz PCI con la modificación del Diseño de Referencia.

Las pruebas del procesador GMS fueron realizadas enviando el contenido de los dos Mapas de Significancias generados por este procesador hacia la memoria externa de la tarjeta, para luego ser leídos por la aplicación software del kit de desarrollo. Para ello se creó un módulo de pruebas capaz de escribir en la memoria externa los Mapas de Significancias conforme los va leyendo. La figura 6.6 muestra el circuito de pruebas para el procesador GMS, el cual sólo fue utilizado en la etapa de validación, ya que luego fue eliminado de la implementación final del sistema.

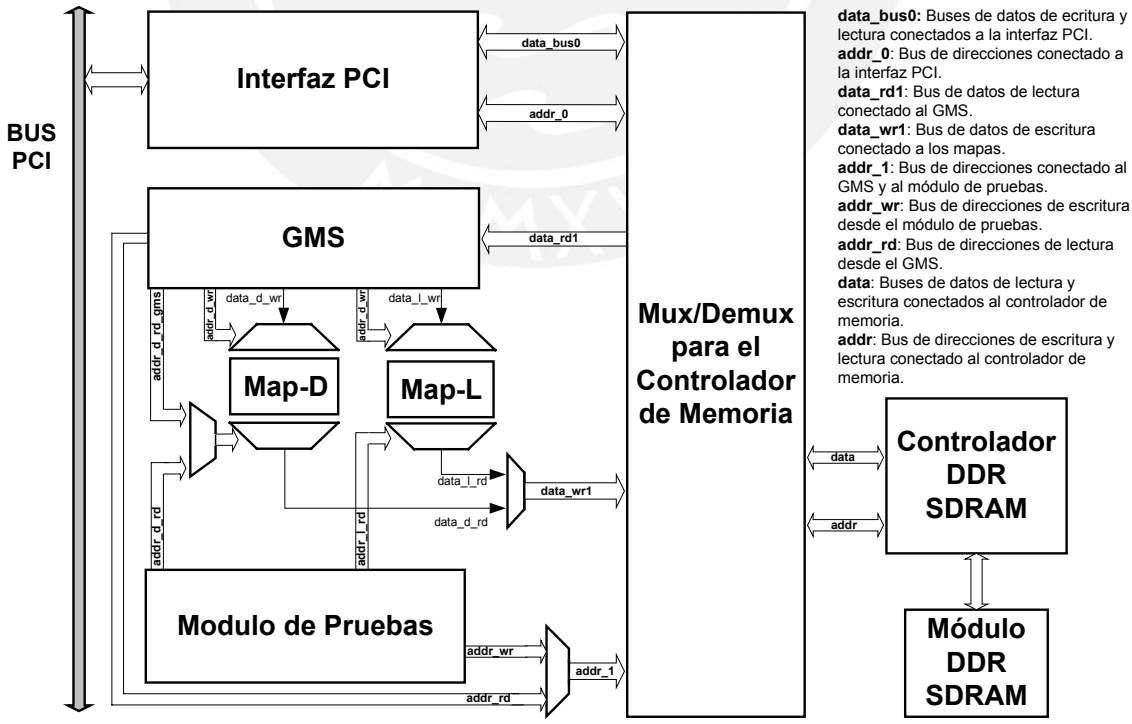


FIGURA 6.6.- CIRCUITO DE PRUEBAS PARA EL PROCESADOR GMS

Para las pruebas del procesador CODEC, un circuito de pruebas similar al anteriormente descrito, fue creado para que lea los valores de la memoria interna que almacena la trama codificada (ver figura 6.7). Estos bits codificados son enviados a la memoria externa para que, junto con las listas de coeficientes, puedan ser leídos por la aplicación software del kit de desarrollo. De esta manera, la prueba del procesador CODEC fue hecha por etapas; desde la validación del módulo inicialización, hasta el correcto funcionamiento del módulo encargado del Procesamiento de LSP. Con la adición de cada uno de los módulos, pudo observarse el correcto crecimiento de la trama codificada.

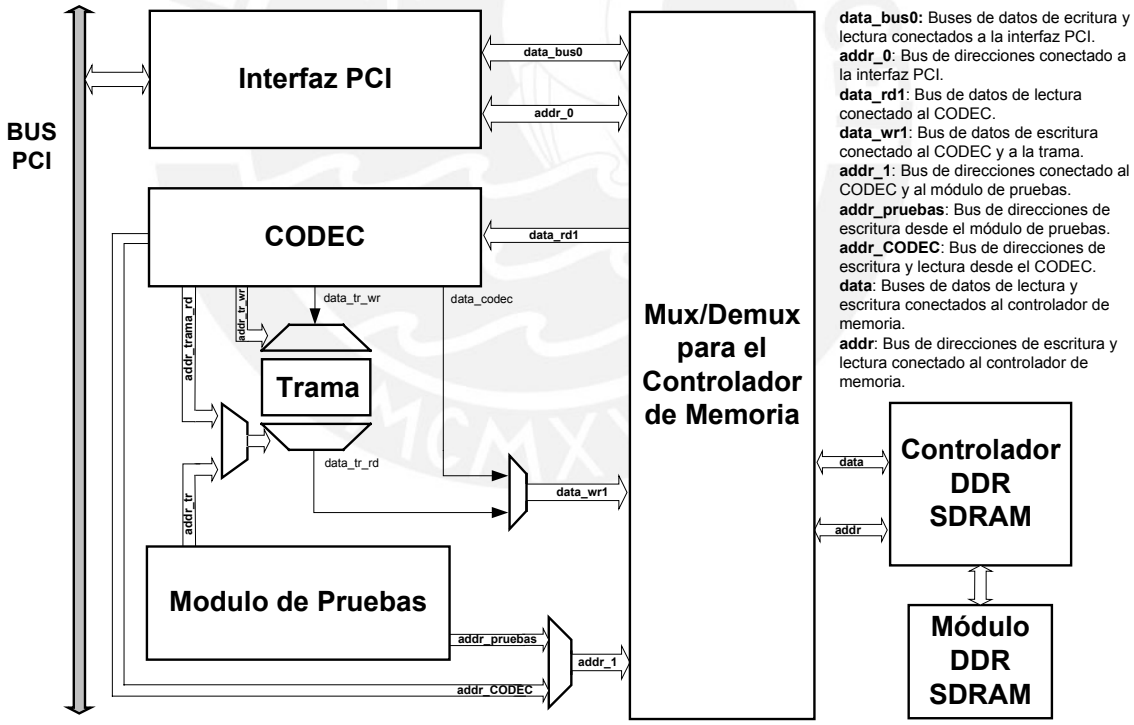


FIGURA 6.7.- CIRCUITO DE PRUEBAS PARA EL PROCESADOR CODEC

Para comprobar la validez de los resultados de las etapas de procesamiento realizadas por los procesadores GMS y CODEC, se comparó los resultados

obtenidos con los resultados que provee la implementación en MATLAB del algoritmo SPIHT y el algoritmo GMS propuesto en el presente trabajo (los códigos se encuentran en el Anexo E). La ejecución de este programa desarrollado en MATLAB puede ser interrumpida en cualquier punto del procesamiento, con la finalidad de comparar sus resultados con los obtenidos en la implementación Hardware.

Cuando todas las partes del sistema fueron debidamente probadas, fueron hechas pruebas con el sistema completo y utilizando imágenes reales. La figura 6.8 muestra, a manera de ejemplo, la reconstrucción de una imagen “Lena” estándar en el dominio wavelet de tamaño 8x8. La diferencia entre las imágenes es imperceptible visualmente y la diferencia entre los valores de los coeficientes es muy pequeña. Esta reconstrucción fue hecha con seis pasadas del algoritmo SPIHT, siendo el número máximo de pasadas igual a siete (dado que el valor umbral inicial es 64). El Error Cuadrático Medio (MSE) es de 0.515 y la Relación Señal a Ruido Pico (PSNR) es de 51 dB. Cabe resaltar que un PSNR superior a 35 dB es considerado aceptable en la mayoría de los casos y constituye un adecuada calidad de reconstrucción. El bit-rate obtenido para seis pasadas es de 1.9531 bits por píxel (bpp), el cual indica el rango de compresión que logra el sistema desarrollado.

FIGURA 6.8 RECONSTRUCCIÓN DE LA IMAGEN “LENA” DE 8x8 PARA 6 PASADAS DEL ALGORITMO

La tabla 6.1 muestra las medidas de MSE, PSNR y bit-rate para diferentes números de pasadas para una imagen “Lena” de 8x8. Es evidente que se obtiene un buen rendimiento del sistema con una configuración de número de pasadas mayor o igual a cinco.

Número de Pasadas	MSE	PSNR (dB)	bit-rate (bpp)
2	202.39	25.06	1.23
3	42.64	31.83	2.60
4	7.76	39.23	3.78
5	2.01	45.08	4.85
6	0.51	51.05	5.95
7	0.65	49.96	7.01

TABLA 6.1. MEDIDAS DE PERFORMANCE DEL SISTEMA PARA DIFERENTES PASADAS DEL ALGORITMO

6.5 RENDIMIENTO DEL SISTEMA

El rendimiento del sistema completo se evaluó en base a pruebas con imágenes en el dominio wavelet de tamaños considerables, tales como: 128x128, 256x256 Y 512x512 píxeles y para una configuración de cinco pasadas del algoritmo SPIHT. La tabla 6.2 muestra los niveles de error obtenidos.

Tamaño de Imagen	MSE	PSNR (dB)
128x128	201.68	25.08
256x256	284.23	23.59
512x512	301.89	23.33

TABLA 6.2.- NIVELES DE ERROR DEL SISTEMA PARA TRES DISTINTOS TAMAÑOS DE IMAGEN

El tiempo de procesamiento del sistema implementado sobre el FPGA, sin considerar los tiempos que involucra la carga y descarga de imágenes a través del Bus PCI, se muestra en la tabla 6.3. La unidad de tiempo está dada en milisegundos y cabe resaltar que el tiempo que involucra la codificación es la suma del tiempo que invierte el Procesador GMS y el Procesador CODEC en “Modo C”.

Tamaño de Imagen	GMS (ms)	CODEC (ms)		Codificación (ms)	Bit-Rate (bits/pixel)
		Modo C	Modo D		
128x128	3.070	0.967	0.712	4.037	0.281
256x256	12.285	1.012	0.713	13.297	0.069
512x512	49.150	1.018	0.904	50.168	0.016

TABLA 6.3.- TIEMPOS DE PROCESAMIENTO DEL SISTEMA PARA TRES DISTINTOS TAMAÑOS DE IMAGEN

7. CONCLUSIONES

- La metodología de diseño Top-Down propuesta en el presente trabajo, abre un camino alternativo para la concepción y desarrollo de arquitecturas VLSI relacionadas con algoritmos de Procesamiento Digital de Señales. Sin embargo, cabe resaltar que ésta es una propuesta inicial, y que puede ser optimizada durante el desarrollo de otras implementaciones hardware de algoritmos.

- El sistema implementado mostró un reducido consumo de área del FPGA STRATIX EP1S25F1020C5, menor al 27% del total de elementos lógicos (LEs). Debe considerarse que sólo los diseños de propiedad intelectual de Altera usados en la arquitectura del sistema, consumen el 19% de LEs del FPGA, mientras que menos del 8% son utilizados por diseños propios.

- Para el diseño de la arquitectura del sistema, se planteó que cada módulo encargado de realizar alguna tarea del algoritmo SPIHT, sea lo más independiente posible (implementación modular). Esto significa que cada módulo posee su propia unidad de control, y solo interactúa con otros módulos a través de señales de inicio o de fin de procesamiento. Bajo esta consideración, se hace más sencilla la supervisión del diseño, lográndose así, una reducción en el tiempo necesario para la detección y corrección de errores.

- No hay una fuerte dependencia frente a la variación de los parámetros (tamaño de palabra y/o tamaño de imagen), ya que el consumo de

celdas lógicas y la frecuencia máxima alcanzada por cada procesador diseñado, no se ven afectadas notablemente. Por lo tanto, esto permite la implementación del sistema para imágenes de tamaño considerable sin riesgo a limitación de recursos del FPGA.

- El uso de parámetros y el reducido consumo de recursos otorga una gran flexibilidad al diseño realizado en el presente trabajo, ya que permite una elevada adaptabilidad con otros diseños, tales como: módulos que realicen la transformada discreta bidimensional de Wavelet u otros módulos que sean necesarios para el desarrollo de un sistema de compresión integrado en un FPGA.

- La alta velocidad de procesamiento del sistema implementado permite alcanzar el requerimiento de tiempo real (<33ms) para imágenes de un tamaño menor a 512x512 píxeles. El motivo de esta restricción, es el elevado tiempo de procesamiento que invierte el procesador GMS (aproximadamente el 98% del tiempo total de procesamiento para tal tamaño de imágenes) por sus múltiples accesos a memoria externa.

- El presente trabajo de tesis permitió por primera vez en la Pontificia Universidad Católica del Perú, experimentar con módulos hardware capaces de interactuar con el Bus PCI. Estos módulos realzan la productividad en el diseño, ya que permiten al diseñador centrar sus esfuerzos en la lógica principal que rodea a la interfaz PCI; logrando con ello, reducir el tiempo de desarrollo del sistema.

- El consumo de memoria interna y externa se encuentra controlado por el parámetro del sistema *isize_width*, lo cual implica que el consumo de memoria se ajusta a las necesidades del usuario. Por lo tanto, esto puede ser muy beneficioso ya que para requerimientos bajos (imágenes pequeñas) no se desperdiciará la memoria asignada.

- El algoritmo SPIHT no realiza operaciones aritméticas complejas, tan sólo operaciones básicas, tales como: sumas, multiplicaciones y divisiones por dos. Por lo tanto, no es necesario el uso de los bloques DSP del dispositivo Stratix.

- La arquitectura desarrollada constituye un aporte significativo a la investigación actual, ya que la compresión Wavelet está siendo recientemente incorporada en los estándar de compresión de imágenes y video. Actualmente se sigue investigando acerca de las implementaciones hardware (FPGA y ASIC) más óptimas, siendo el presente trabajo un paso más en la línea de investigación iniciada con los proyectos [17], [18], [12], los cuales fueron desarrollados en el área de Lógica Programable del Grupo de Microelectrónica de la Pontificia Universidad Católica del Perú.

8.- RECOMENDACIONES

- El consumo de memoria externa puede ser reducido considerablemente (aproximadamente a la sexta parte del consumo de memoria actual) si en cada posición de memoria externa se almacena más de un dato. Esto es posible, debido a que el diseño presentado utiliza como máximo 19 bits efectivos, de un total de 128 bits que posee cada posición de memoria. Sin embargo, esta consideración trae consigo una elevada complejidad en el control de los accesos a memoria.

- Como un trabajo futuro para continuar el presente trabajo de tesis, se plantea que para elevar el tiempo de procesamiento del sistema, existe la posibilidad de desarrollar una versión paralela del procesador GMS. Esto es, en lugar de calcular un nuevo mapa de significancia por cada pasada del algoritmo, todos los mapas pueden ser generados simultáneamente en una única pasada del algoritmo. Para esto, se agregan tantas comparaciones (con distintos umbrales) en el proceso de análisis de significancias como mapas se quieran generar. Por ejemplo, si el sistema está configurado para un total de cinco pasadas, el tiempo de procesamiento que consume el procesador GMS en su versión original es “x”, mientras que en su versión paralela consumiría “x/5”. Pero el consumo de memoria interna para almacenar los mapas, se elevaría a 5 veces el consumo original; lo cual no es crítico considerando la capacidad de memoria interna que posee el FPGA utilizado.

REFERENCIAS

[1] Mallat, S. G., "A theory for multiresolution signal decomposition: the wavelet representation", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 11, No. 7, págs. del 674 al 693, 1989.

[2] Daniel Iparraguirre Cárdenas y Gerard Santillán Quiñonez, "Implementación de la Transformada Discreta de Wavelet de Una y Dos Dimensiones Usando la Técnica Dígito - Serial", *Tesis de Licenciatura, Pontificia Universidad Católica del Perú*, Abril 2001.

[3] K. Sayood. Introduction to Data Compression. Morgan Kauffman, San Francisco, CA, 1996.

[4] Geoff Davis y Aria Nosratinia, "Wavelet-Based Image Coding: An Overview", *Applied and Computational Control, Signals, and Circuits*, Vol. 1, No. 1, Mayo 1998.

[5] Shapiro, J. M., "Embedded image coding using zerotrees of wavelet coefficients", *IEEE Trans. on Signal Processing*, Vol. 41, No. 12, págs. del 3445 al 3462, Diciembre 1993.

[6] A. Said and W. A. Pearlman, "A new fast and efficient image codec based on set partitioning in hierarchical trees", *IEEE Trans. on Circuits and Systems for Video Technology*, Vol. 6, No.3, págs. del 243 al 250, Junio 1996.

[7] A. Said, W. A. Pearlman, "SPIHT Image Compression: Properties of the Method", http://www.cipr.rpi.edu/research/SPIHT/spiht1.html

[8] D. Taubman, "High performance scalable image compression with EBCOT", *IEEE Trans. on Image Processing*, Vol. 9, págs. del 1158 al 1170, Julio 2000.

[9] W. Fry, “Hyperspectral image compression on reconfigurable platforms”. Master Thesis, University of Washington, Seattle, Washington 2001.

[10] J. Ritter, G. Fey, P. Molitor. SPIHT implemented in a XC4000 device. MWSCAS 2002.

[11] J. Ritter. Wavelet based image compression using FPGAs. Master Thesis, Martin-Luther-University Halle Germany, 2002.

[12] Chris Tomás Horna, Manuel Paredes Castro, Daniel Iparraguirre Cárdenas, “Arquitectura flexible para la codificación wavelet SPIHT de imágenes sobre FPGAs”, *Anales del X Workshop Iberchip*, Cartagena de Indias – Colombia, Marzo 2004.

[13] Altera Corporation, Stratix Device Handbook, (V.3.1), Setiembre 2004.

[14] Altera Corporation, PCI MegaCore Function User Guide, *pci_ug* (V.2.2), Febrero 2003.

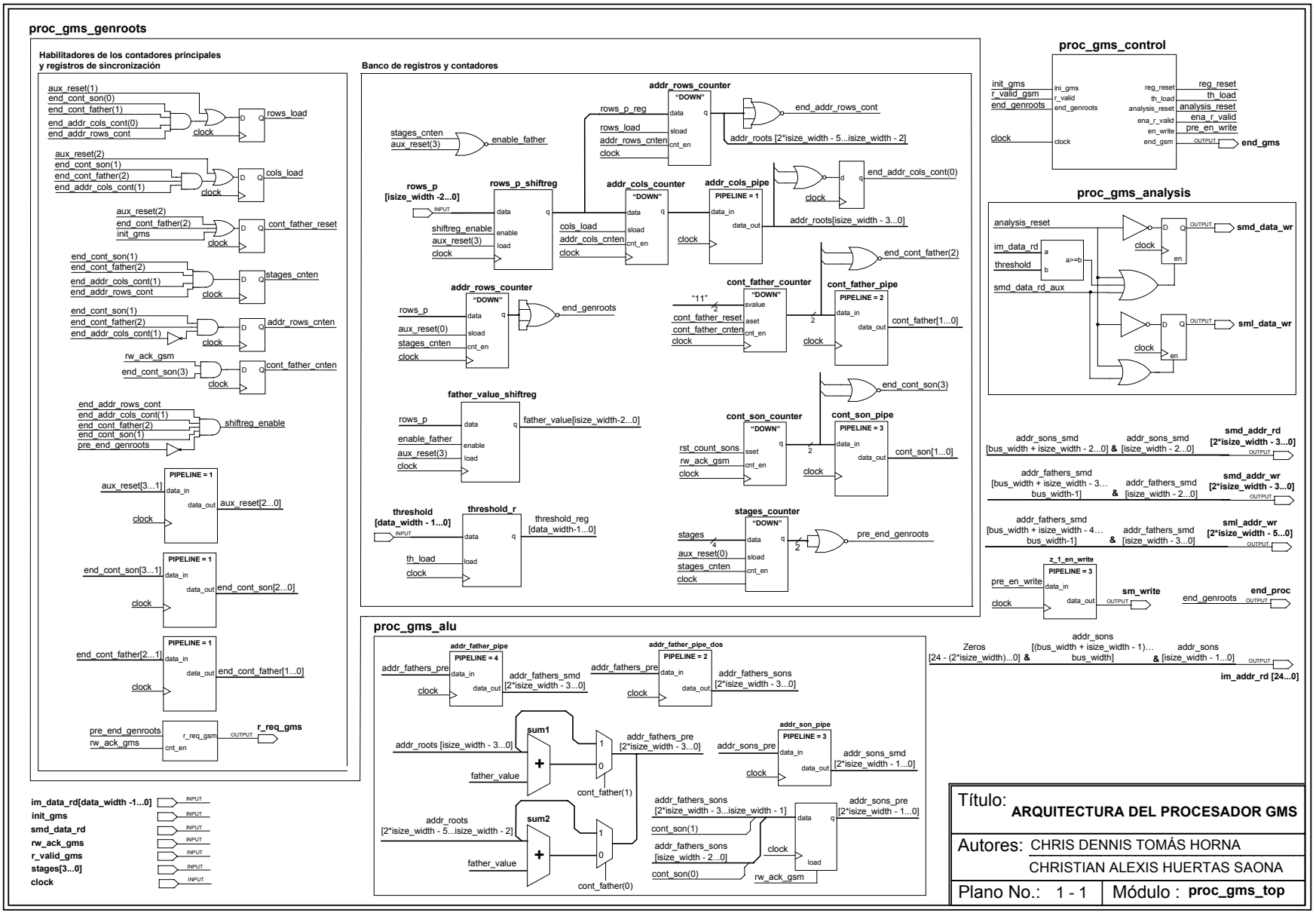
[15] Micron Technology, Inc., “*256Mb: x8 (DDR) SDRAM MT46V32M8 Datasheet*”, Marzo 2004.

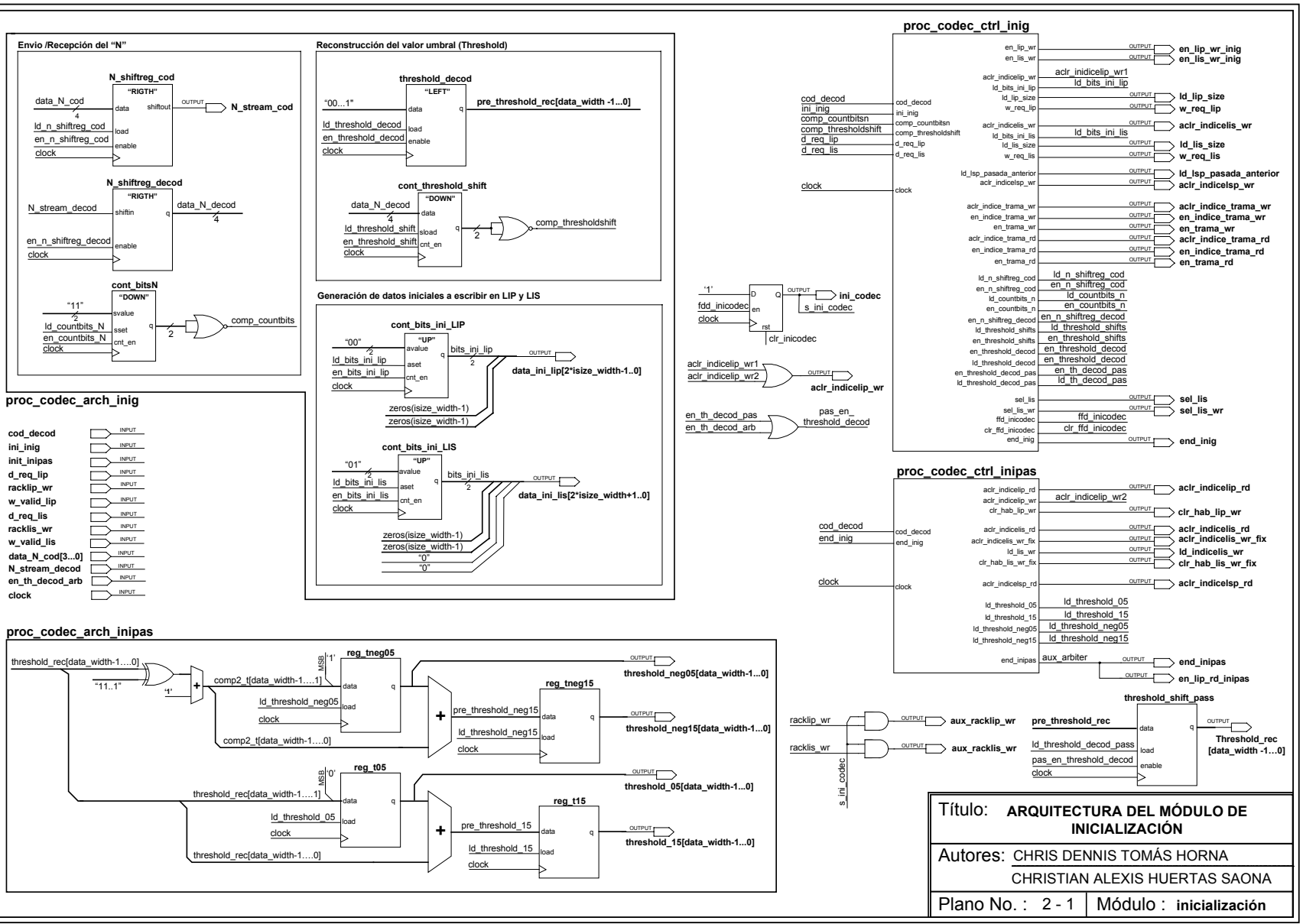
[16] Altera Corporation, PCI-to-DDR SDRAM Reference Design, *an223* (V.1.0), Mayo 2003.

[17] D. Iparraguirre Cárdenas, C. Tomás Horna, M. Paredes Castro, “Arquitectura Flexible y Configurable de la Transformada Discreta Bidimensional de Wavelet implementada en FPGAs”. *Anales del IX Workshop Iberchip*, La Habana–Cuba, Marzo 2003.

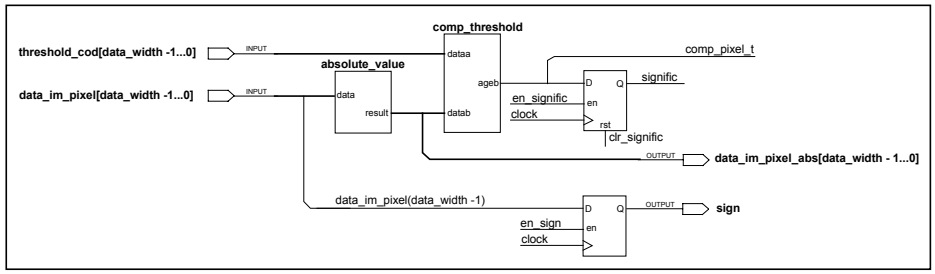
[18] Chris Tomás Horna, Christian Huertas Saona, Carlos Silva Cárdenas. “Implementación de un Codificador/Decodificador Wavelet SPIHT para la Compresión de Imágenes sobre FPGA”. *Anales del XI Workshop Iberchip*, Salvador de Bahia – Brasil, Marzo 2005.

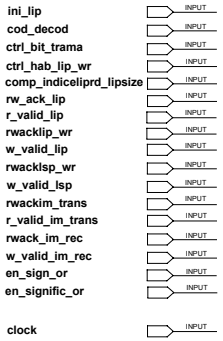
PLANOS



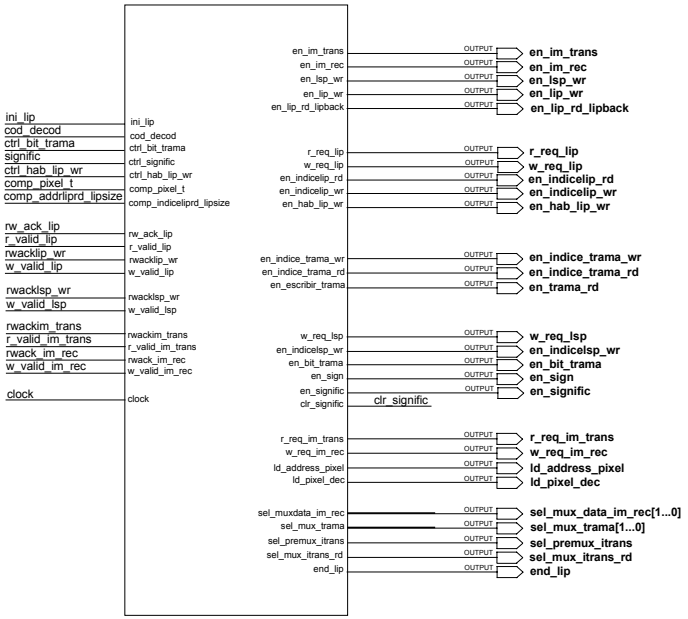


proc_codec_arch_lip

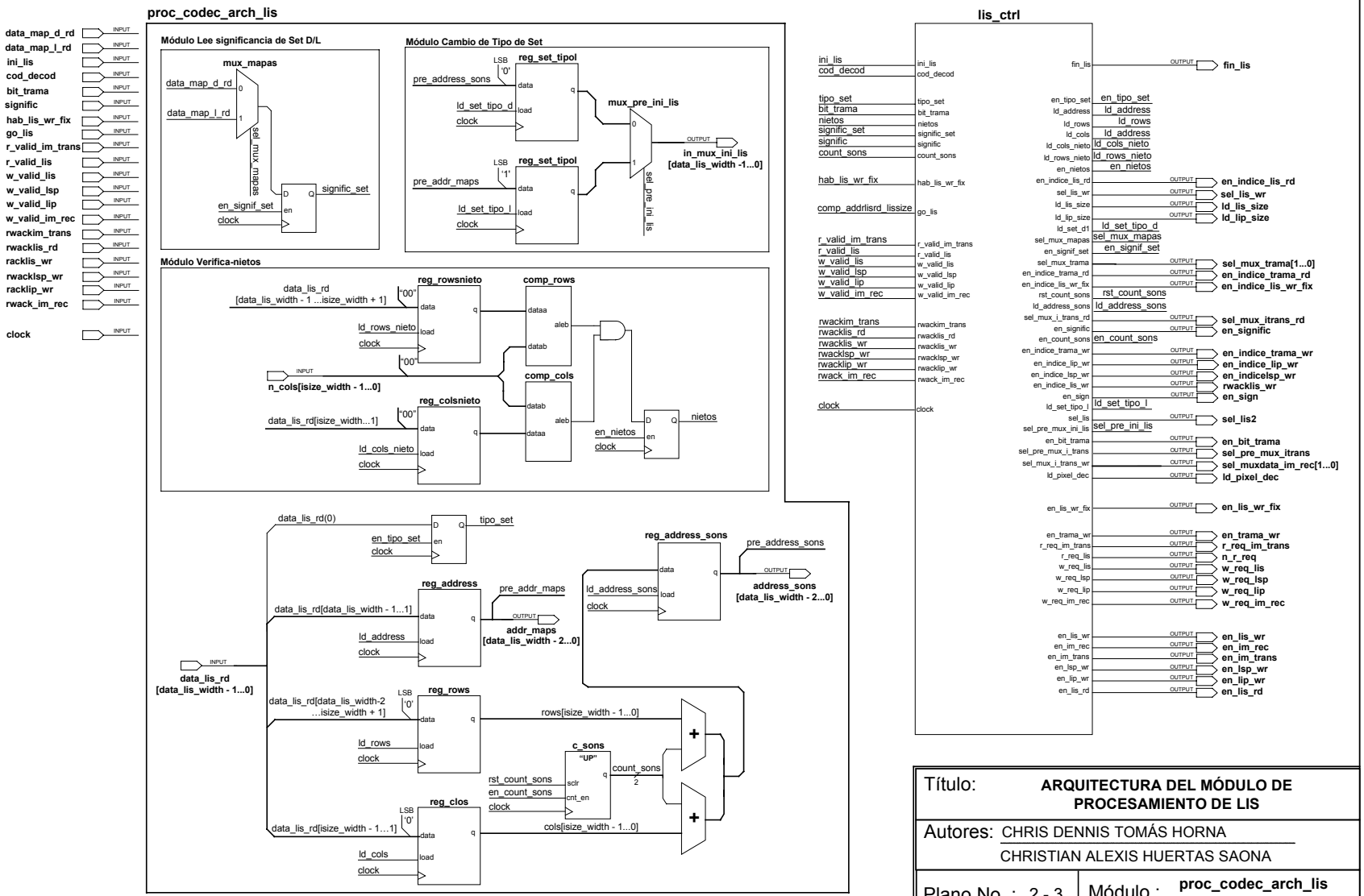




proc_codec_ctrl_lip

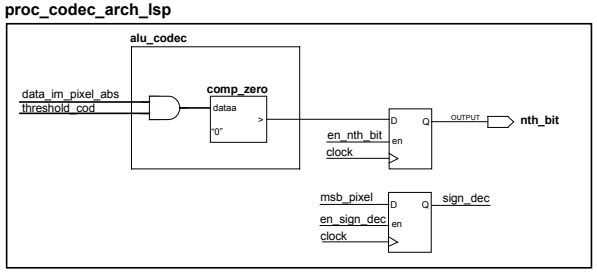


Título:	**ARQUITECTURA DEL MÓDULO DE PROCESAMIENTO DE LIP**
Autores:	CHRIS DENNIS TOMÁS HORNA CHRISTIAN ALEXIS HUERTAS SAONA
Plano No. : 2 - 2	Módulo : **proc_codec_arch_lip** **proc_codec_ctrl_lip**



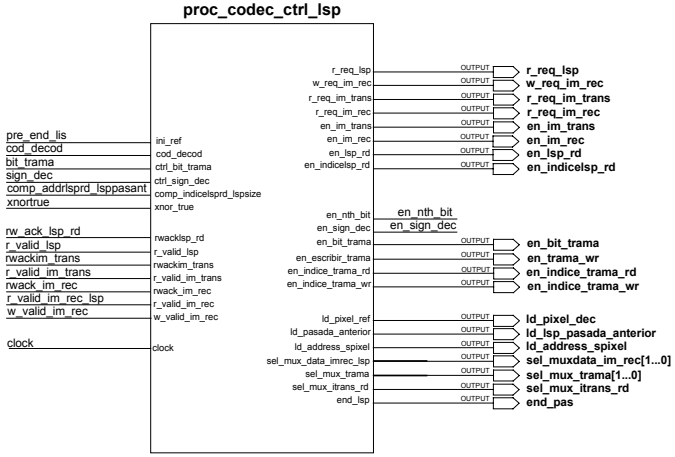
Título:	**ARQUITECTURA DEL MÓDULO DE PROCESAMIENTO DE LIS**
Autores:	CHRIS DENNIS TOMÁS HORNA CHRISTIAN ALEXIS HUERTAS SAONA
Plano No. : 2 - 3	Módulo : **proc_codec_arch_lis lis_ctrl**

proc_codec_arch_lsp

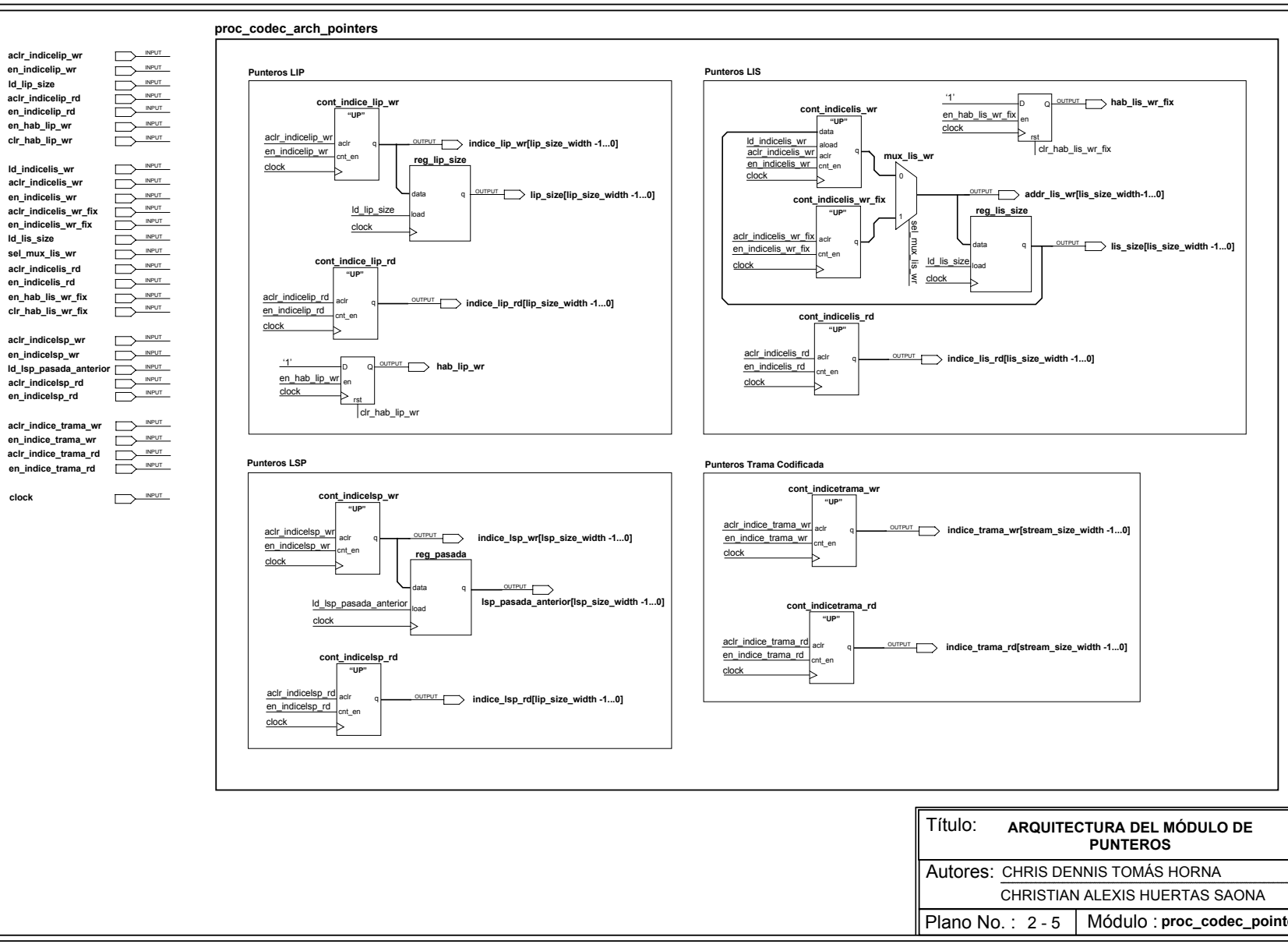


data_im_pixel_abs[data_width -1...0]	INPUT
threshold_cod[data_width -1...0]	INPUT
data_im_pixel[data_width -1...0]	INPUT
ini_ref	INPUT
cod_decod	INPUT
bit_trama	INPUT
comp_indicelsprd_lspsize	INPUT
xnortrue	INPUT
rw_ack_lsp_rd	INPUT
rwackim_trans	INPUT
r_valid_lsp	INPUT
r_valid_im_trans	INPUT
rwack_im_rec	INPUT
r_valid_im_rec	INPUT
w_valid_im_rec	INPUT
clock	INPUT

proc_codec_ctrl_lsp



Título:	**ARQUITECTURA DEL MÓDULO DE PROCESAMIENTO DE LSP**
Autores:	CHRIS DENNIS TOMÁS HORNA CHRISTIAN ALEXIS HUERTAS SAONA
Plano No. : 2 - 4	Módulo : **proc_codec_arch_lsp** **proc_codec_ctrl_lsp**



Título:	**ARQUITECTURA DEL MÓDULO DE PUNTEROS**
Autores:	CHRIS DENNIS TOMÁS HORNA CHRISTIAN ALEXIS HUERTAS SAONA
Plano No. : 2 - 5	Módulo : **proc_codec_pointers**