ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE ADQUISICIÓN, COMPRESIÓN Y ALMACENAMIENTO DE IMÁGENES EMPLEANDO VHDL (VHSIC HARDWARE DESCRIPTION LANGUAGE) Y FPGAs (FIELD PROGRAMMABLE GATE ARRAYs)

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

PABLO XAVIER JIMÉNEZ ESPINOSA

pablo_pxje85@hotmail.com

DIRECTOR: Ph.D. IVÁN BERNAL

imbernal@mailfie.epn.edu.ec

Quito, Enero 2011

DECLARACIÓN

Yo, Pablo Xavier Jiménez Espinosa, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

_____

Pablo Xavier Jiménez Espinosa

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Pablo Xavier Jiménez Espinosa, bajo mi supervisión.

_____

Ph.D. Iván Bernal
DIRECTOR DEL PROYECTO

AGRADECIMIENTOS

A mis padres sin cuyo amor y apoyo incondicional no hubiese concretado las metas hoy alcanzadas.

A mis amigos, con los que compartimos buenos y malos momentos en las aulas de la Escuela Politécnica Nacional, por su confianza y apoyo.

Al Dr. Iván Bernal, por su confianza, su apoyo y su dedicado tutelaje, sin los cuales este proyecto no hubiese salido adelante.

Al Dr. Daniel Cárdenas por su valiosa ayuda en el momento más crítico de este proyecto.

DEDICATORIA

Con infinito amor y gratitud a mis padres
Antonio y Marcia

Pablo Jiménez

CONTENIDO

ÍNDICE DE FIGURAS

CAPÍTULO 1

CAPÍTULO 2

CAPÍTULO 3

ANEXO 4

ANEXO 5

ÍNDICE DE TABLAS

CAPÍTULO 1

CAPÍTULO 2

CAPÍTULO 3

ANEXO 4

ANEXO 7

ANEXO 8

ANEXO 9

RESUMEN

Los FPGAs actuales han sufrido cambios drásticos desde su aparición a mediados de la década de los ochenta, de simples sistemas de lógica de acoplamiento a verdaderos sistemas en chip (SoC), con sistemas de millones de compuertas, capaces de albergar bloques lógicos que pueden implementar casi cualquier función lógica, así como bloques embebidos de microprocesadores, DSPs, e interfaces de entrada/salida de muy alta velocidad, haciendo que estos dispositivos sean aptos para casi cualquier aplicación.

El presente trabajo pretende demostrar el potencial que poseen los FPGAs actuales y el lenguaje de descripción de hardware VHDL para sintetizar, a nivel de hardware, algoritmos complejos; tal es el caso del esquema de codificación *Baseline* del estándar JPEG para la compresión de imágenes, mediante el uso de los estilos descriptivos que posee el lenguaje VHDL. Para ello se utiliza el módulo de desarrollo Spartan-3A *Starter Kit Board*, que posee un FPGA XC3S700A de la Familia Spartan-3A de Xilinx y la herramienta de desarrollo ISE *Foundation* 10.1.

El sistema diseñado admite los datos de una imagen monocromática de 160x120 pixeles sin comprimir, y devuelve los datos de la imagen comprimida con el formato de archivo JFIF para imágenes. Además se emplea una interfaz gráfica de usuario, desarrollada con el entorno de programación gráfica GUIDE de Matlab, para la interacción con el sistema diseñado y la visualización de resultados.

PRESENTACIÓN

Este proyecto presenta, en su parte fundamental, la descripción VHDL, la síntesis y la implementación sobre un FPGA de los procedimientos que se llevan a cabo en el esquema de codificación *Baseline* del estándar JPEG para la compresión de imágenes, los cuales son: la transformada discreta del coseno, la cuantización, la exploración en zig-zag y la codificación de entropía de Huffman, con el fin de obtener un compresor de imágenes de calidad variable a través de diferentes niveles de cuantización. Para la descripción de estos procedimientos en el lenguaje VHDL se ha hecho uso de los estilos descriptivos estructural y algorítmico que posee este lenguaje.

El desarrollo de este proyecto se divide en cuatro capítulos:

Capítulo 1: Fundamentos Teóricos. En este capítulo se hace una breve descripción de la arquitectura básica de un FPGA, para después profundizar en el estudio de la arquitectura de los FPGAs de la plataforma Spartan 3 de Xilinx. También, se hace una revisión del módulo de desarrollo Spartan-3A *Starter Kit Board* y del software de desarrollo *ISE Foundation* 10.1 de Xilinx. Además, se describe brevemente los procesos que intervienen en el esquema de codificación *Baseline* del estándar JPEG, como también el formato de archivo JFIF para imágenes comprimidas con JPEG.

Capítulo 2: Diseño del Sistema de adquisición, compresión y almacenamiento de imágenes. El capítulo inicia con una descripción global de los módulos constitutivos del sistema de adquisición, compresión y almacenamiento de imágenes. Se presenta la interfaz gráfica de control y se centra en la descripción del diseño de los módulos que conforman el sistema planteado. Se da una explicación de los procedimientos que se llevan a cabo en cada módulo a través de diagramas de bloques y diagramas ASM para máquinas de estados, y se presentan los resultados de la simulación de los bloques de hardware diseñados.

Capítulo 3: Implementación y pruebas del sistema. El capítulo comienza con la exposición de los resultados obtenidos durante la síntesis de las secciones de hardware del sistema sobre el FPGA XC3S700A de la familia Spartan-3A de Xilinx; se muestran los resultados de los recursos lógicos utilizados en el diseño, la frecuencia máxima de operación del sistema y los retardos por lógica combinacional e interconexión. Además, se describe brevemente el procedimiento realizado para configurar el módulo de desarrollo Spartan-3A *Starter Kit Board*. Finalmente, se exponen las imágenes comprimidas obtenidas con el sistema en dos escenarios de prueba y se hace la verificación de los parámetros de compresión.

Capítulo 4: Conclusiones y recomendaciones. Se expone las conclusiones y recomendaciones obtenidas durante el desarrollo del proyecto.

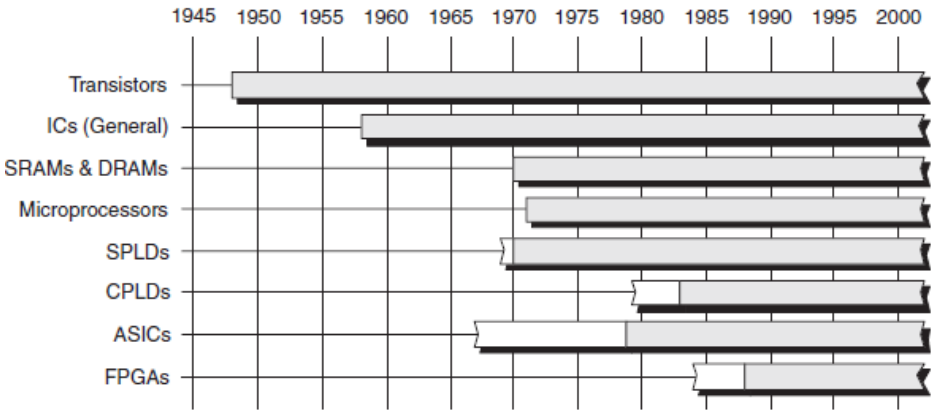
Por último, se presentan anexos con información adicional sobre los temas tratados en el desarrollo del proyecto, y se incluye el código fuente de la interfaz gráfica de control y el código VHDL de las secciones de hardware del sistema.

CAPÍTULO I

FUNDAMENTOS TEÓRICOS

1.1 ORIGEN DE LOS DISPOSITIVOS FPGAs

Hablar de FPGAs no es hablar de una tecnología nueva; estos dispositivos hacen su aparición a mediados de la década de los ochentas como sencillos sistemas digitales de lógica de acoplamiento (*glue logic*), los cuales tenían un limitado número de recursos lógicos y su función primordial era interconectar grandes bloques lógicos o dispositivos [1]. A inicios de la década de los noventas, el tamaño y sofisticación de los FPGAs fueron creciendo y ganando terreno en campos como las telecomunicaciones y las redes de datos, y no es sino hasta inicios del 2000 cuando se llegó a tener FPGAs de alto rendimiento con sistemas de millones de compuertas y bloques embebidos de microprocesadores, DSPs, e interfaces de entrada/salida de muy alta velocidad (pudiendo sobrepasar los 5 Gbps), haciendo que estos dispositivos sean aptos para casi cualquier aplicación, como por ejemplo sistemas de comunicaciones, procesamiento de imágenes y video, redes neuronales artificiales, procesamiento digital de señales, por mencionar unos pocos. En su conjunto, los FPGAs han llegado a ser considerados como sistemas en chip (SoC), en los cuales se tiene tanto elementos de hardware como de software (programas ejecutándose sobre los microprocesadores embebidos). La Figura 1-1 ilustra una línea de tiempo con la aparición de dispositivos que en las diferentes décadas sirvieron (y siguen sirviendo) para diseñar sistemas electrónicos.



Fuente [3]

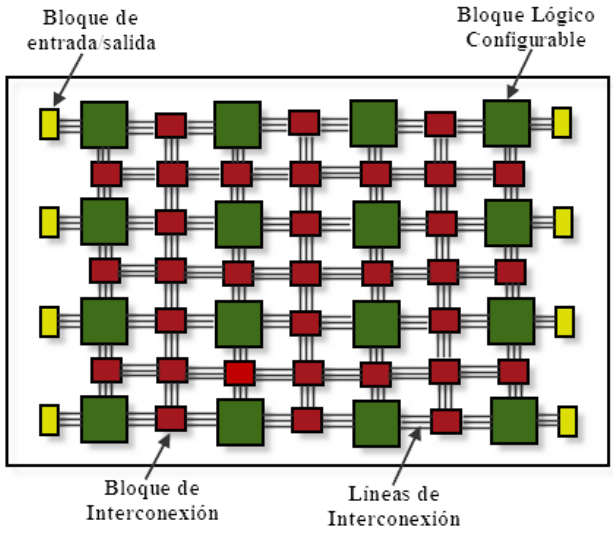
Figura 1- 1. Línea de tiempo con la aparición de diferentes dispositivos electrónicos

El éxito que han tenido los FPGAs se debe en gran medida al hecho de que, a pesar de sus vastos recursos, su valor comercial no se ha incrementado en la misma proporción que éstos [1] [2].

1.2 CONCEPTOS GENERALES SOBRE FPGAs

Un FPGA es un dispositivo lógico programable compuesto por un conjunto de bloques lógicos comunicados a través de conexiones programables, en el cual se puede implementar tanto circuitos combinacionales como secuenciales. En un concepto más coloquial se puede definir a un FPGA como un circuito integrado en blanco o virgen, en cuyo interior se encuentran elementos que nos permitirán implementar un circuito integrado personalizado que realice una función dedicada.

En su arquitectura más básica, un FPGA consta al menos de tres bloques funcionales: *Bloques Lógicos Configurables*, *Bloques de Entrada/Salida* y una *Matriz de Interconexiones Programables*. En la Figura 1-2 se muestra la arquitectura básica de un FPGA constituida por los tres bloques antes mencionados.

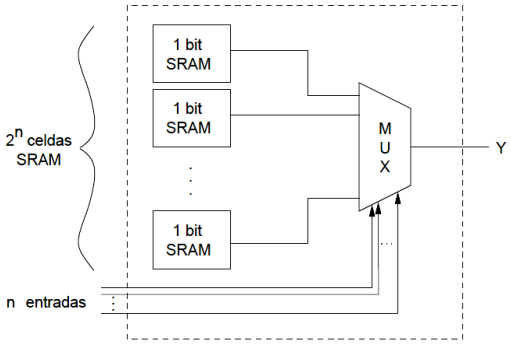


Fuente [4]

Figura 1- 2. Arquitectura básica de un FPGA

1.2.1 LOS BLOQUES LÓGICOS CONFIGURABLES

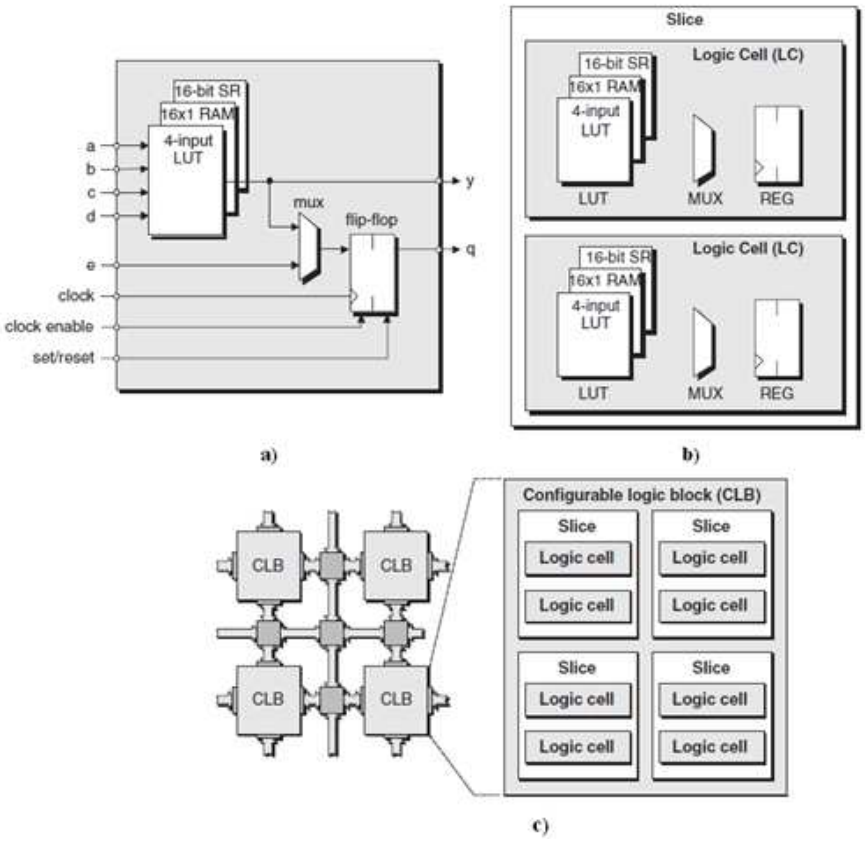
La parte más importante de un FPGA son los bloques lógicos configurables (CLB, por sus siglas en inglés), estos están constituidos por *Look-Up Tables* (LUTs) y por elementos de almacenamiento tipo flip-flops que permiten sincronizar la salida del CLB con una señal de reloj. Los LUTs son componentes de memoria RAM donde se almacena una tabla de verdad que define una función booleana y cuyas entradas de dirección son las variables de entrada de dicha función; en general con un LUT de nx1 se puede implementar cualquier función booleana de n entradas. En la actualidad se pueden tener FPGAs con LUTs de cuatro a seis entradas [3]. La Figura 1-3 indica la estructura básica de un LUT.



Fuente [4]

Figura 1- 3. Estructura básica de un LUT

Hay que mencionar que los distintos fabricantes de FPGAs tienen diferentes denominaciones para establecer los niveles jerárquicos en los que se agrupan los recursos lógicos disponibles en sus dispositivos. Por ejemplo, Xilinx establece una celda lógica (*Logic Cell*), la cual está conformada por un LUT, multiplexores y un *flip-flop* como lo indica la Figura 1-4a. Las celdas lógicas luego se agrupan en *slices* que contienen dos celdas lógicas como lo muestra la Figura 1-4b, y por último los slices se agrupan en los Bloques Lógicos Configurables (CLB), los cuales se encuentran formados por cuatro slices, como lo indica la Figura 1-4c. Otros fabricantes como Altera, Actel, Lattice, Atmel, entre otros, poseen sus propias denominaciones para los recursos lógicos en sus FPGAs. Se ha hecho mención sólo de la denominación de los recursos lógicos disponibles en los dispositivos de Xilinx, ya que para el presente proyecto se utilizará un FPGA de este fabricante.



Fuente [3]

Figura 1- 4. Jerarquía de agrupación de recursos lógicos en FPGAs de Xilinx: a) Celda lógica, b) Slice formado por dos Celdas Lógicas, c) Bloques Lógicos Configurables formados por cuatro slices

1.2.2 LOS BLOQUES DE ENTRADA/SALIDA

Los bloques de Entrada/Salida definen la interfaz del FPGA con el exterior, permitiendo el paso de señales hacia adentro y hacia afuera del dispositivo. La mayoría de fabricantes agrupa los pines de entrada/salida en bancos. Esto permite que los pines del circuito integrado tengan lógica de diferentes estándares eléctricos de manera independiente y programada por el usuario, además permite que se divida el dominio de reloj, ya que las entradas de reloj están asociadas a diferentes bancos [4]. Los bloques de entrada/salida constan de al menos tres elementos básicos: buffers de tres-estados, *resistencias pull-down/pull-up* y registros de entrada y salida.

1.2.3 LA MATRIZ DE INTERCONEXIÓN

Para utilizar los recursos lógicos mencionados anteriormente en conjunto, los FPGAs poseen una matriz de conexiones programables, la cual une los bloques funcionales a través de canales de líneas de conexión distribuidos de forma vertical y horizontal. Existen varias clases de líneas de conexión dependiendo de la distancia que cubran y su capacidad de transmitir señales de alta frecuencia.

1.2.4 LOS BLOQUES EMBEBIDOS

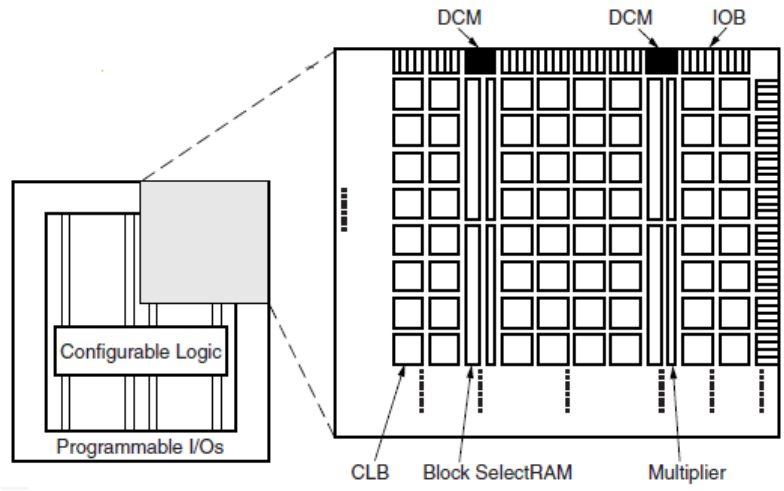
Las nuevas generaciones de dispositivos FPGAs presentan, además de los bloques tradicionales expuestos anteriormente, bloques embebidos que realizan una función específica. Estos bloques son añadidos con el propósito de evitar el consumo excesivo de los recursos lógicos de los CLBs debido a la implementación de funciones complejas. Los bloques embebidos han ampliado el rango de aplicación de los FPGAs. Entre los bloques embebidos más comunes se pueden mencionar:

- ***RAM Embebida.*** Estos bloques son los más comunes en los FPGAs actuales y, dependiendo de la arquitectura de éstos, pueden estar organizados como columnas en medio del arreglo de CLBs, o dispersos a lo largo de la superficie del dispositivo o en la periferia de éste. Los bloques de RAM embebida pueden servir para implementar memorias de tipo RAM, ROM, FIFOs, entre otros.
- ***Multiplicadores.*** Estos bloques permiten obtener el producto de dos números binarios de manera rápida, en muchos de los casos en un solo ciclo de reloj. Los multiplicadores embebidos, por lo general, se encuentran cerca de los bloques de memoria RAM, puesto que usualmente procesan los datos almacenados en estos bloques.
- ***Bloques de DSP.*** Estos bloques permiten implementar ciertas operaciones específicas y típicas que se dan en el procesamiento digital de señales. Por lo general, estos bloques contienen recursos lógicos para implementar circuitos multiplicadores-acumuladores como son multiplicadores, sumadores y registros.
- ***Administradores de Reloj (Digital Clock Managers – DCM).*** Estos bloques permiten generar distintas señales de reloj a partir de la proveniente del oscilador del FPGA. Mediante estos bloques es posible sintetizar nuevas frecuencias que pueden ser múltiplos o submúltiplos de la frecuencia de la señal de reloj original, así como manipular la fase de dicha señal de reloj.
- ***Microprocesadores Embebidos.*** Algunos FPGAs poseen microprocesadores embebidos que se interconectan con la lógica programable del resto del dispositivo. Esto permite añadir a sistemas basados en microprocesadores, la flexibilidad de diseñar sus periféricos con lógica programable [5]. Los bloques de microprocesadores que se encuentran integrados en el silicio del FPGA se conocen como *hard-processors*[1].

[1] Además de los hard-processors algunos fabricantes de FPGAs configuran un grupo de los recursos lógicos de sus FPGAs para actuar como un microprocesador, a ellos se los denomina soft-processors.

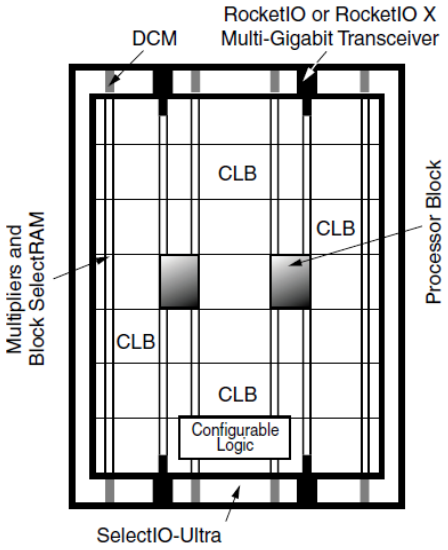
- ***Transceivers de alta Velocidad*.** Los FPGAs de última generación poseen bloques especiales para transmisión y recepción de datos a gran velocidad, pudiendo sobrepasar los 5Gbps. Los *transceivers* utilizan señales diferenciales para la transmisión y recepción de los datos, además sólo pueden establecer comunicaciones punto a punto.

En las Figuras 1-5 y 1-6 se ilustran algunos ejemplos de arquitecturas de FPGAs que incorporan los bloques embebidos mencionados anteriormente.



Fuente [6]

Figura 1- 5. Arquitectura de una Virtex II de Xilinx



Fuente [7]

Figura 1- 6. Arquitectura de una Virtex II Pro de Xilinx

Existen varias tecnologías de programación para los bloques antes mencionados. En el Anexo 1 se describen brevemente las más comunes actualmente.

1.3 LA PLATAFORMA DE FPGAs SPARTAN 3 DE XILINX

La plataforma de FPGAs Spartan 3 ofrece una densidad de sistema de compuertas desde las 50000 hasta los 5 millones. La plataforma Spartan 3 incluye tres familias, las cuales se mencionan en la Tabla 1-1.

Tabla 1- 1. Familias de FPGAs de la plataforma Spartan 3

Familia	Sub-familia
Spartan-3	N/A
Spartan-3A	Spartan-3A
	Spartan-3AN
	Spartan-3A DSP
Spartan-3E	N/A

En su mayoría, los dispositivos de las familias de esta plataforma comparten una arquitectura común, la cual se detalla más adelante; sin embargo, en algunos elementos o bloques funcionales habrá sutiles diferencias entre dichas familias, sobre todo respecto al rendimiento y, en casos muy particulares, cambios en un bloque funcional de una familia a otra.

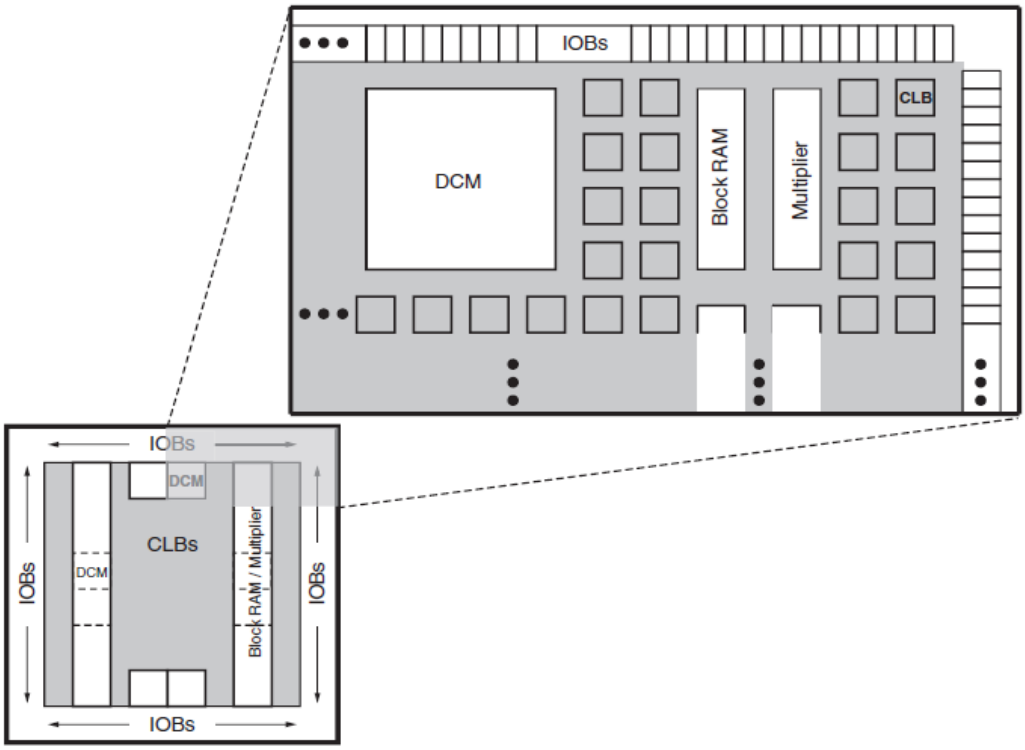
1.4 ARQUITECTURA DE LOS FPGAs DE LA PLATAFORMA SPARTAN 3

La arquitectura de los FPGAs de la plataforma Spartan 3 está compuesta de seis bloques funcionales programables, los cuales son:

1. Bloques Lógicos Configurables (CLBs)
2. Bloques de entrada/salida (IOBs)
3. Bloques de RAM embebida

4. Multiplicadores embebidos
5. Administrador Digital del Reloj (Digital Clock Manager, DCM)
6. Matriz de Interconexión

La descripción de la arquitectura presentada a continuación es basada en la Familia Spartan-3A, ya que el dispositivo utilizado en este proyecto es parte de esta familia. En el caso de que exista alguna diferencia en los bloques funcionales en las otras familias de la plataforma se hará la respectiva aclaración. La Figura 1-7 muestra la arquitectura de los FPGAs de la familia Spartan-3A.



Fuente [8]

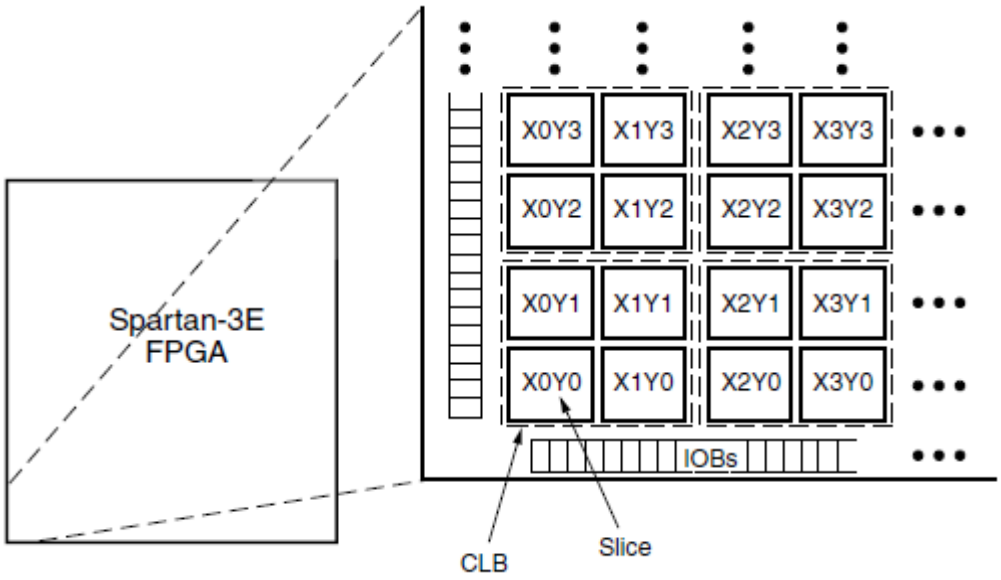
Figura 1- 7. Arquitectura de la plataforma Spartan-3A

1.5 DESCRIPCIÓN DE LOS BLOQUES FUNCIONALES

1.5.1 LOS BLOQUES LÓGICOS CONFIGURABLES (CLBS)

Los Bloques Lógicos Configurables (CLBs) son el principal recurso del FPGA para implementar tanto lógica combinacional como lógica secuencial. Su estructura interna se encuentra conformada por cuatro *slices*, los cuales a su vez están constituidos por dos *Look-Up Tables* (LUTs) y dos elementos de almacenamiento (*flip-flop*). Todas las familias dentro de la plataforma Spartan 3 comparten la estructura de CLBs anteriormente descrita.

Como se indica en la Figura 1-8, los CLBs están organizados en un arreglo de filas y columnas. Resulta obvio pensar que el número total de CLBs esté dado por el producto del número de filas y número de columnas, sin embargo éste es algo menor debido a que, como se ilustró en la Figura 1-7, los DCMs y Bloques de RAM se encuentran embebidos en el arreglo de CLBs.



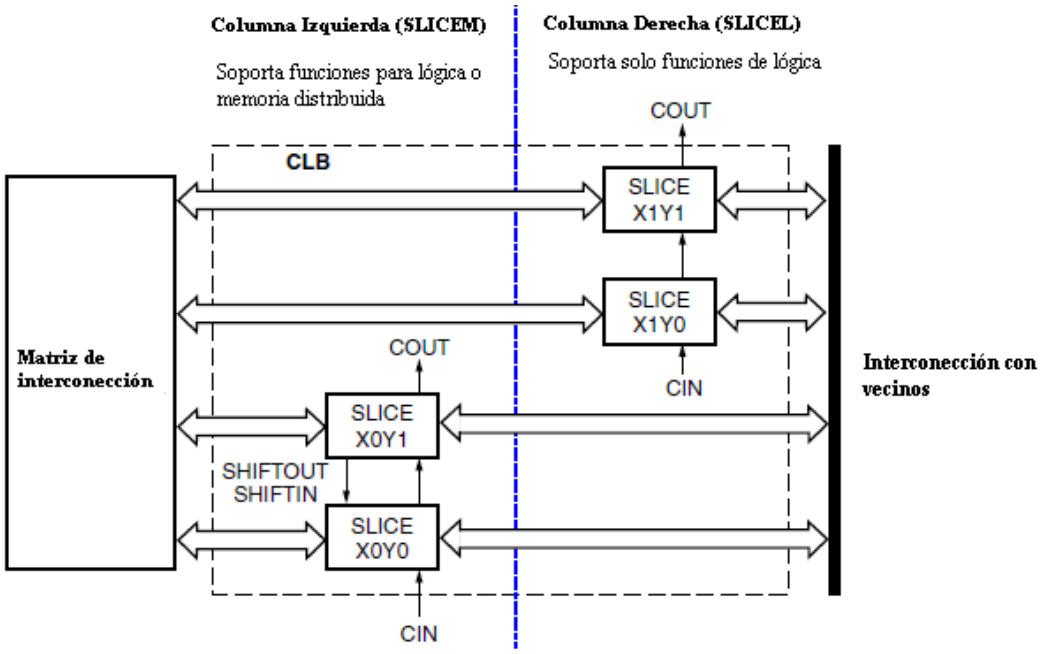
Fuente [8]

Figura 1- 8. Organización de CLBs en un Dispositivo Spartan 3

1.5.2 SLICES

Dentro del CLBs, los cuatro *slices* se dividen en dos pares organizados en dos columnas: derecha e izquierda. Los *slices* de la columna izquierda, denominados SLICEM, soportan funciones de lógica (combinacional y secuencial) y de memoria mientras que los de la columna derecha, llamados SLICEL, soportan solo funciones de lógica. La Figura 1-9 ilustra esta disposición.

Como se pudo observar en la Figura 1-8, la ubicación de un *slice* se determina mediante sus coordenadas (X, Y), siendo X el número de columna y Y el número de fila, cuyos valores inician en la columna inferior izquierda y se incrementan de izquierda a derecha y de arriba hacia abajo, respectivamente.



Fuente [8]

Figura 1- 9. Disposición de slices dentro de un CLB

Tanto los SLICEM como los SLICEL tienen en común los siguientes elementos:

- Dos LUTs de 4 entradas, denominados F y G
- Dos elementos de almacenamiento
- Dos multiplexores denominados F5MUX y FiMUX

- Lógica aritmética

Por otro lado los SLICEM soportan dos funciones específicas:

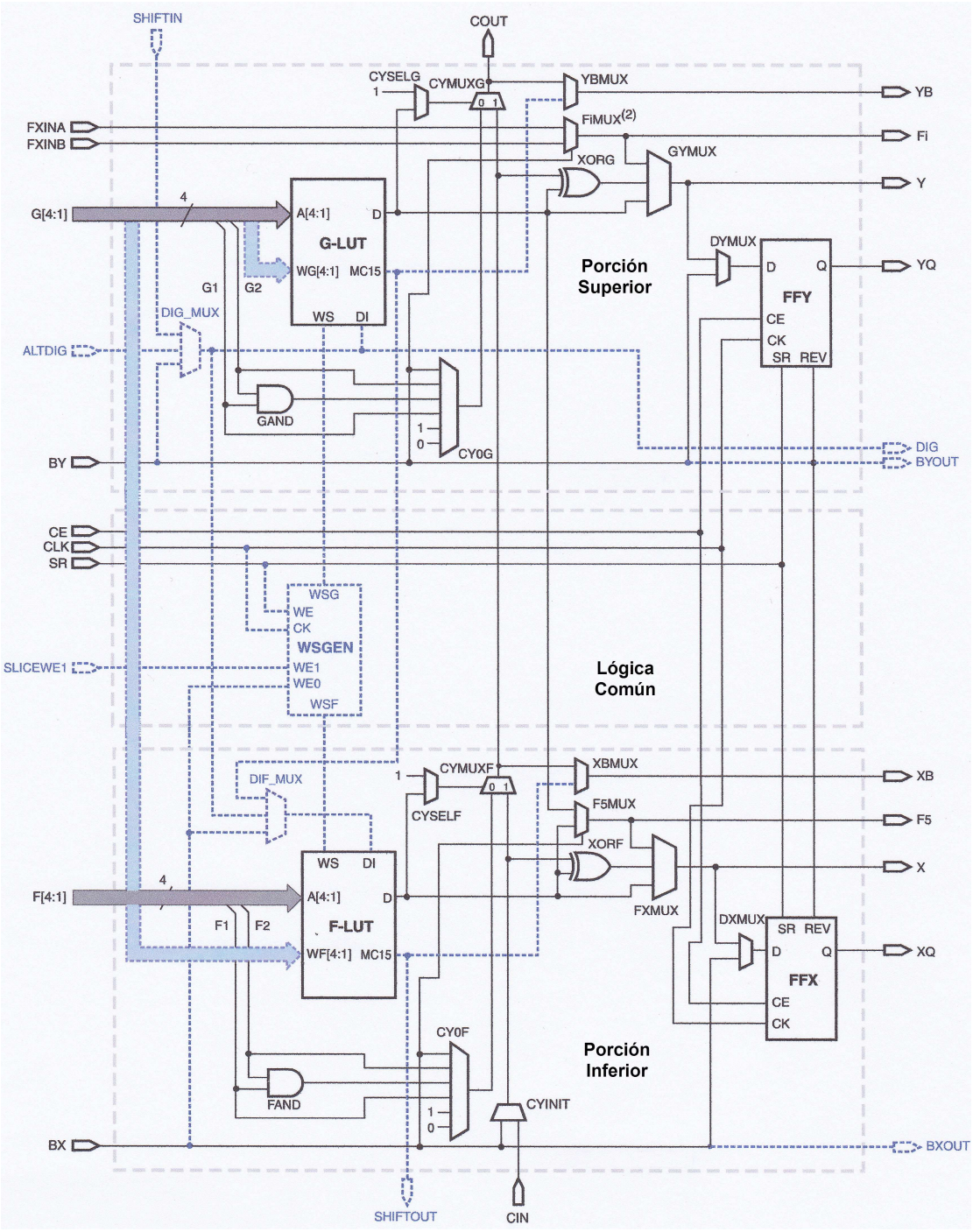
- Dos bloques de memoria RAM distribuida de 16x1.
- Dos registros de desplazamiento de 16 bits.

En la Figura 1-10 se puede observar el diagrama de la estructura de un *slice*. El *slice* se divide en una porción superior y otra inferior, ambas divisiones (porciones) comparten cuatro señales de control: reloj (CLK), habilitación del reloj (CE), habilitación de escritura en el *slice* (SLICEWE1) y Reset/Set (RS). Las líneas entre cortadas de color azul indican los recursos y conexiones que se encuentran disponibles sólo en *slices* de tipo M (SLICEM). En cada porción del *slice* se tiene un LUT, un elemento de almacenamiento (flip-flop), un multiplexor dedicado y elementos lógicos para realizar operaciones aritméticas.

1.5.3 DESPLAZAMIENTO DE DATOS DENTRO DE LOS SLICES (VÍAS LÓGICAS)

Dentro de un *slice* los datos de entrada, así como los datos procesados pueden seguir varias vías lógicas dependiendo de la lógica que se infiera en la descripción de hardware del sistema. La salida “D” de cada uno de los LUTs puede seguir cinco vías lógicas (Ver la Figura 1-10):

1. Salir del *slice* a través de las líneas X (para la porción inferior) o Y (para la porción superior).
2. Dentro del *slice*, a través de las líneas X e Y, alimentar la entrada de datos de los *flip-flops* FFX (para la porción inferior) y FFY (para la porción superior) por medio de los multiplexores DXMUX y DYMUX de la porción inferior y superior, respectivamente. La salida “Q” de los *flip-flops* abandona el *slice* a través de las líneas XQ y YQ.



Fuente [8]

Figura 1- 10. Diagrama simplificado de un SLICE

La Tabla 1-2 describe las señales de entrada y salida que poseen los *slices*.

Tabla 1- 2. Descripción de las señales de entrada y salida de un *slice*

Nombre	Tipo de slice/Porción del slice	Dirección	Descripción
F[4:1]	SLICEL/M Inferior	Entrada	Entrada de datos al F-LUT y a la compuerta FAND (solo F1 y F2)
G[4:1]	SLICEL/M Superior	Entrada	Entrada de datos al G-LUT y a la compuerta GAND (solo G1 y G2). También proporciona la dirección de escritura en memoria solo en SLICEM
BX	SLICEL/M Inferior	Entrada	Entrada de bypass hacia la salida BXOUT (SLICEM), entrada al elemento de almacenamiento, entrada de control del F5MUX, entrada hacia la lógica de carry, o entrada de datos a la RAM (SLICEM)
BY	SLICEL/M Superior	Entrada	Entrada de bypass hacia la salida BYOUT (SLICEM), entrada al elemento de almacenamiento, entrada de control del FiMUX, entrada hacia la lógica de carry, o entrada de datos a la RAM (SLICEM)
BXOUT	SLICEM Inferior	Salida	Salida directa de la entrada de bypass BX
BYOUT	SLICEM Superior	Salida	Salida directa de la entrada de bypass BY
ALTDIG	SLICEM Superior	Entrada	Entrada de datos alterna hacia la RAM
DIG	SLICEM Superior	Salida	Salida de bypass para las señales de entrada ALTDIG o SHIFTIN
SLICEWE1	SLICEM Común	Entrada	Habilitación de escritura en la RAM
F5	SLICEL/M Inferior	Salida	Salida del multiplexor F5MUX. Realimenta a un FiMUX
FXINA	SLICEL/M Superior	Entrada	Entrada hacia FiMUX desde un multiplexor F5MUX u otro FiMUX
FXINB	SLICEL/M Superior	Entrada	Entrada hacia FiMUX desde un multiplexor F5MUX u otro FiMUX
Fi	SLICEL/M Superior	Salida	Salida del multiplexor FiMUX
CE	SLICEL/M Común	Entrada	Habilitación del reloj para los flip-flops FFX y FFY
SR	SLICEL/M Común	Entrada	Set o Reset para los flip-flops FFX/Y o habilitación para escritura en memoria RAM (SLICEM)
CLK	SLICEL/M Común	Entrada	Reloj para los flip-flops FFX/Y o reloj para la memoria RAM en SLICEM
SHIFTIN	SLICEM Superior	Entrada	Entrada de datos a G-LUT cuando es usado como memoria RAM
SHIFTOUT	SLICEM Inferior	Salida	Salida de datos desplazados de F-LUT cuando es usado como memoria RAM
CIN	SLICEL/M Inferior	Entrada	Entrada a la cadena de acarreo
COUT	SLICEL/M Superior	Salida	Salida de la cadena de acarreo
X	SLICEL/M Inferior	Salida	Salida Combinatoria
Y	SLICEL/M Superior	Salida	Salida Combinatoria
XB	SLICEL/M Inferior	Salida	Salida combinatoria desde la lógica de Carry o el F-LUT
YB	SLICEL/M Superior	Salida	Salida combinatoria desde la lógica de Carry o el G-LUT
XQ	SLICEL/M Inferior	Salida	Salida del flip-flop FFX
YQ	SLICEL/M Superior	Salida	Salida del flip-flop FFX

3. Servir como entradas de control de los multiplexores CYMUXF y CYMUXG de la cadena de acarreo.
4. En la cadena de acarreo, servir como entrada para las compuertas XORF y XORG de la porción inferior y superior, respectivamente.
5. Ambas salidas de los LUTs G-LUT y F-LUT pueden alimentar el multiplexor F5MUX para implementar funciones lógicas.

Además de las vías lógicas antes mencionadas, los *slices* poseen dos vías de *bypass* denominadas BY (para la porción superior) y BX (para la porción inferior). Las vías de *bypass* pueden seguir una de siete vías lógicas descritas como siguen:

1. Salir directamente del *slice* mediante las salidas BYOUT (para la porción superior) y BXOUT (para la porción inferior) sin entrar a los LUTs.
2. Llegar hasta la entrada "D" del elemento de almacenamiento (*flip-flops*) sin entrar a los LUTs.
3. Entrar como señal de control para los multiplexores F5MUX o FiMUX.
4. Servir como entrada a la cadena de acarreo a través de los multiplexores de vía (CY0F y CY0G).
5. Manejar las entradas DI de los LUTs.
6. La entrada BY (de la porción superior) puede controlar las entradas de inversión REV de los *flip-flops* FFY y FFX.
7. A través del multiplexor DIG_MUX, la línea de entrada BY de la porción superior puede salir del *slice* mediante la línea DIG.

1.5.4 ELEMENTOS DE ALMACENAMIENTO

Dentro de un *slice* existen dos elementos de almacenamiento de datos denominados FFY y FFX, los cuales pueden trabajar como *latch* o como *flip-flops* tipo D. La entrada de estos elementos está controlada por un multiplexor, DYMUX para FFY y DXMUX para FFX (ver Figura 1-10), los cuales enrutan hacia la

entrada del elemento de almacenamiento señales provenientes del resto de componentes del *slice*.

1.5.5 LOOK-UP TABLES (LUTs)

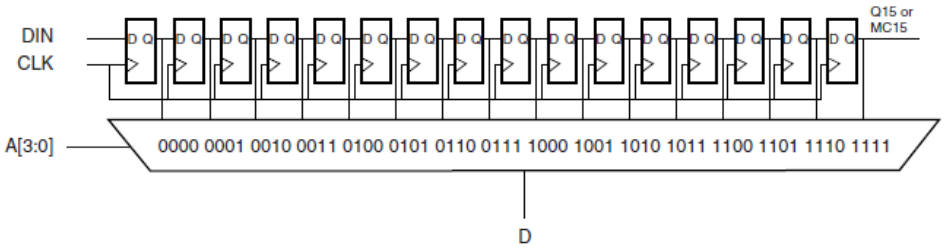
Los LUTs son generadores de funciones basados en una memoria RAM. Éstos son el principal recurso que tiene el FPGA para implementar funciones lógicas, además de servir como RAM distribuida o como registros de desplazamiento de 16 bits en los *slices* tipo M (SLICEM).

Los LUTs (sean G-LUTs o F-LUTs) tienen cuatro entradas y una salida. Con éstos se puede implementar cualquier función lógica de cuatro variables. Para funciones lógicas de más de cuatro variables los LUTs se agrupan en cascada mediante los multiplexores dedicados que para este fin posee cada *slice*.

1.5.5.1 LUTs como registros de desplazamiento de 16 bits

Los LUTs de un SLICEM pueden ser configurados como registros de desplazamiento de 16 bits. Esto implica que en cada CLB existirán cuatro LUTs que pueden ser configurados como registros de desplazamiento.

Cuando el LUT es utilizado como registro de desplazamiento, su arquitectura convencional se modifica para que se comporte como un registro de desplazamiento direccionable como el que indica la Figura 1-11. Esto hace que se puede utilizar toda la longitud del registro de desplazamiento (16 bits) o configurar una longitud deseada a través de las líneas de selección (dirección) A[3:0]. El dato de entrada se desplazará a través de una cadena de registros hacia la salida Q15/MC15 o a la salida D del LUT.

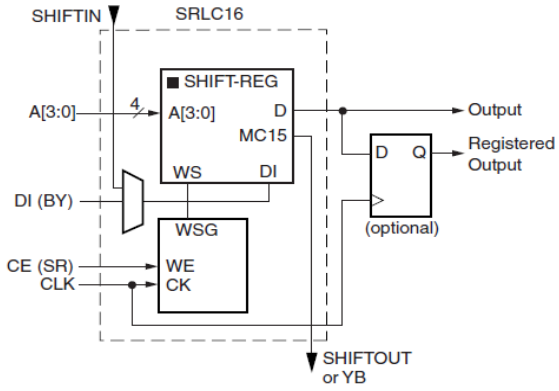


Fuente [8]

Figura 1- 11. LUT configurado como registro de desplazamiento direccionable

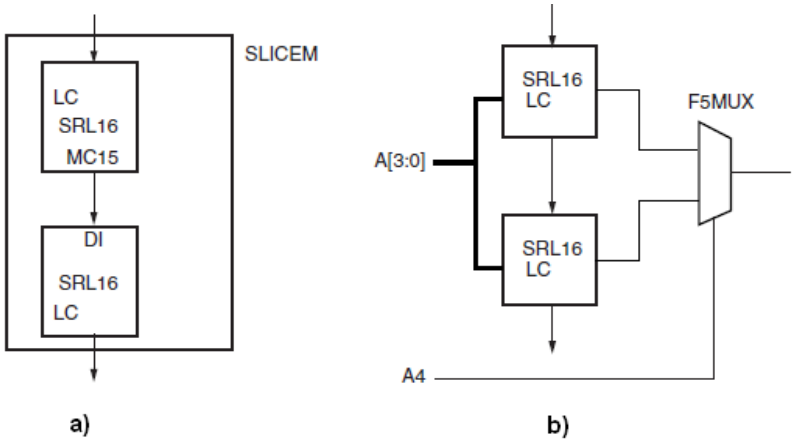
En el caso de que se necesite almacenar la salida D del registro de desplazamiento, se puede utilizar el *flip-flop* asociado al LUT como lo muestra la Figura 1-12. Para esto se debe considerar a dicho *flip-flop* como el último registro en la cadena mostrada en la Figura 1-11.

Los LUTs en un SLICEM pueden agruparse en cascada para implementar registros de desplazamiento de mayor longitud, esto se hace conectando la salida MC15 de un LUT a la entrada DI del otro LUT en el mismo *slice*, con lo cual se obtiene un registro de desplazamiento de longitud fija de 32 bits como lo indica la Figura 1-13a. De forma similar, cuando se requiere un registro de desplazamiento de longitud variable, el multiplexor F5MUX es utilizado para ampliar la selección de cualquiera de los 32 bits del registro a través de las líneas de selección A[4:0], como se muestra en la Figura 1-13b.



Fuente [8]

Figura 1- 12. Estructura de una celda lógica como Registro de Desplazamiento

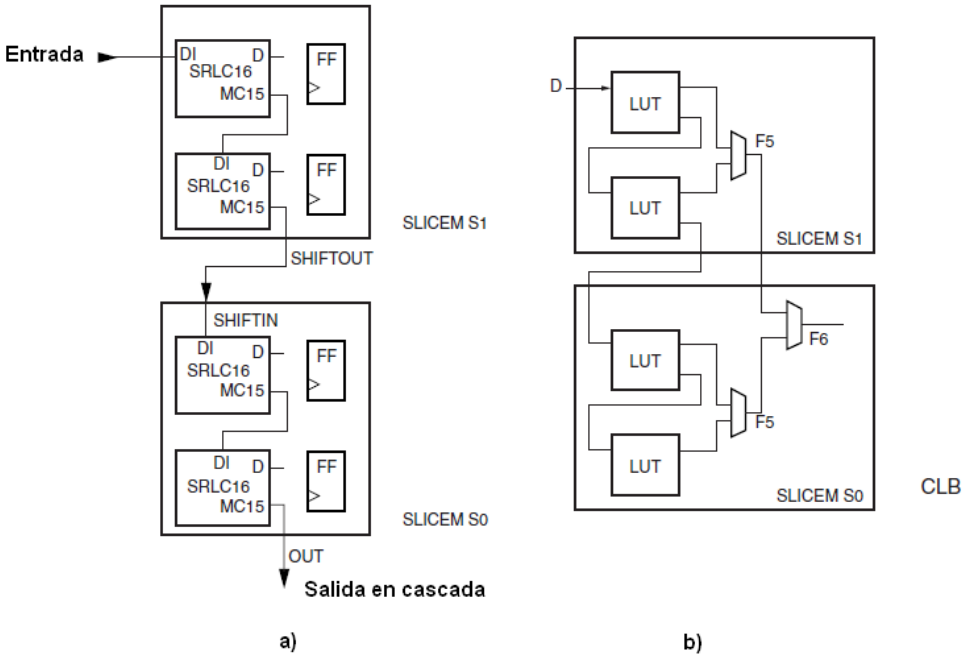


Fuente [8]

Figura 1- 13. Configuración en cascada de LUTs como registros de desplazamiento en un SLICEM: a) Registro de desplazamiento de longitud fija (32bits), b) registro de desplazamiento de longitud variable

Dentro de un CLB, todos los LUTs de los dos SLICEM pueden conectarse en cascada para tener un registro de desplazamiento de hasta 64 bits. Para ello la salida SHIFTOUT del primer SLICEM se conecta a la entrada SHIFTIN del segundo SLICEM. Al igual que con los LUTs dentro de un SLICEM, un multiplexor, el F6MUX, es utilizado para tener un registro de desplazamiento de longitud variable, mediante el cual se puede obtener cualquiera de los 64 bits del registro en el CLB. La Figura 1-14 muestra estas dos configuraciones para los dos SLICEM de un CLB.

La disposición en cascada de los LUTs en los SLICEM, es transparente para el diseñador, y se lleva a cabo en el proceso de síntesis, cuando se tenga una descripción funcional de un registro de desplazamiento, en un lenguaje HDL, cuyas características demanden una disposición de LUTs como la que se mencionó anteriormente.

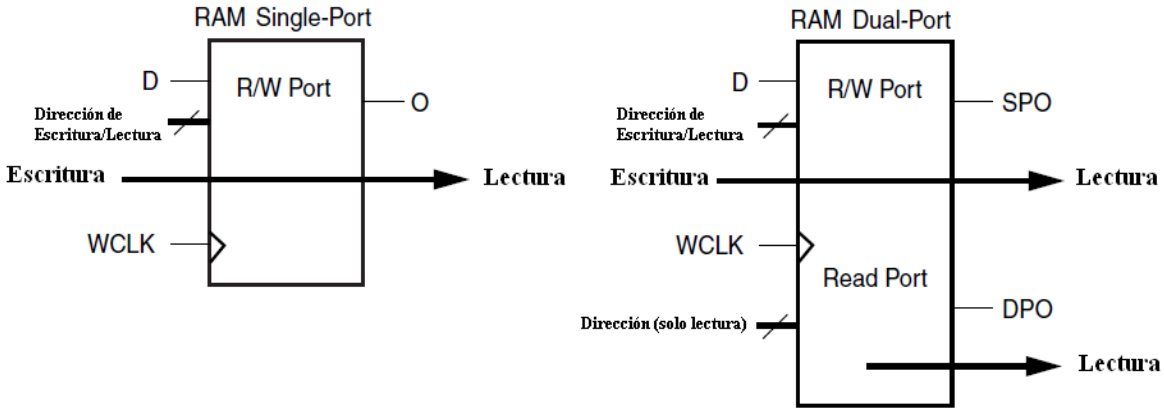


Fuente [8]

Figura 1- 14. Configuración en cascada de LUTs como registros de desplazamiento dentro de un CLB: a) Registro de desplazamiento de longitud fija de 64 bits, b) registro de desplazamiento de longitud variable

1.5.5.2 LUTs como memoria RAM distribuida

Cada LUT de un SLICEM puede ser configurado como una RAM sincrónica de 16x1 (16 palabras de un bit), así en un CLB se puede tener un total de 64 bits de RAM distribuida. En un *slice* tipo M se pueden implementar dos tipos de memoria RAM basadas en LUTs: memoria RAM *single-port* y memoria RAM *dual-port*. La diferencia entre estas memorias radica en que mientras una memoria *single-port* tiene un solo puerto para operaciones de lectura y escritura, en una memoria RAM *dual-port* se tiene un puerto para la operación de lectura/escritura y otro puerto exclusivo para la operación de lectura, como lo indica la Figura 1-15. Las memorias *single-port* son implementadas utilizando un LUT del SLICEM al contrario de las RAM *dual-port* que utilizan los dos LUTs del SLICEM.



Fuente [8]

Figura 1- 15. RAM distribuida en modalidades single-port y dual-port

En la implementación de una memoria RAM *dual-port*, los mismos datos son escritos en ambos LUTs del SLICEM, y pueden ser procesados desde dos líneas independientes de acceso; es decir, un LUT tiene el puerto para lectura/escritura, y el otro LUT tiene el puerto para lectura exclusivamente. Cuando se tenga una operación de escritura, el primer puerto escribirá los datos en ambos LUTs. Con este tipo de memoria se puede tener operaciones de lectura/escritura en el primer puerto (SPO) y simultáneamente una operación de lectura en el segundo puerto (DPO) de manera independiente una de la otra. Las 4 entradas del LUT son las líneas de dirección de la memoria RAM distribuida o LUT RAM.

La operación de escritura de una RAM distribuida es sincrónica y efectuada durante el flanco positivo de la señal de reloj siempre que la señal de habilitación de escritura (WE) esté activa. Por defecto, esta señal es activa en alto aunque se puede invertir su valor dentro de la RAM distribuida. Cuando sucede una transición positiva en la señal de reloj, y la señal WE está activa, la LUT RAM almacena el bit presente en el puerto D en la localidad de memoria que señala las líneas de dirección.

La operación de lectura, por su parte, es asincrónica y utiliza solamente lógica combinacional para acceder al contenido de la localidad de memoria deseada, lo cual implica un tiempo de acceso equivalente al retardo que infiere la lógica del

LUT. En el caso que se necesite lectura sincrónica de los contenidos de memoria, se puede utilizar el *flip-flop* tipo D asociado con cada LUT.

Los LUTs de los *slices* tipo M de un CLB pueden agruparse de diferentes maneras para tener RAM distribuida de mayor capacidad de palabras y mayor longitud en cada palabra. Es así que en un CLB se puede llegar a tener RAM distribuida de 16x4, 32x1, 32x2, o 64x1 configurando los LUT en cascada y/o en paralelo. Al igual que en el caso de los registros de desplazamiento con LUTs, la agrupación de LUTs para formar RAM distribuida de mayor capacidad, es transparente para el diseñador y se lleva a cabo en el proceso de síntesis.

1.5.6 LOS MULTIPLEXORES DEDICADOS DEL SLICE (F5MUX Y FIMUX)

Los multiplexores dedicados mejoran el rendimiento en diseños que requieran selección de datos o la implementación de funciones booleanas de varias variables. El rendimiento se ve mejorado debido a las conexiones dedicadas con cero retardos entre un LUT y el multiplexor, y entre cada uno de estos multiplexores. Esto contrasta con la implementación de multiplexores con LUTs, cuyos retardos de lógica y de conexión acumulados en cada bloque, disminuyen el rendimiento del sistema.

Básicamente, los multiplexores dedicados que poseen los *slices* cumplen dos funciones:

- Ampliar la densidad de entradas para funciones de selección de datos (multiplexores de mayor tamaño).
- Ampliar el rango de variables de entrada en funciones booleanas, a más de cuatro que es el número de variables que un LUT puede aceptar.

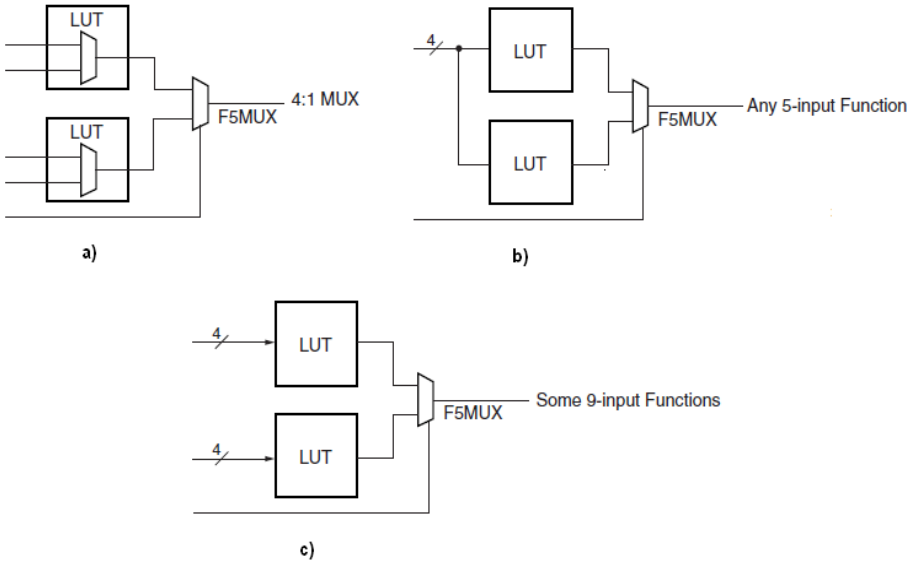
Para esto, los *slices* tienen dos multiplexores: F5MUX y FiMUX. Cabe mencionar que todos los *slices* (sean estos SLICEM o SLICEL) son idénticos con respecto a los multiplexores dedicados.

1.5.6.1 El Multiplexor F5MUX

Este multiplexor siempre combina los dos LUTs en un *slice* y recibe esta denominación debido a que con esta combinación puede generar cualquier función booleana de cinco entradas. Con la combinación de los dos LUTs y el multiplexor F5MUX se puede implementar:

- Un multiplexor 4:1.
- Un generador de cualquier función booleana de cinco entradas.
- Un generador parcial de funciones booleanas de nueve variables.

La Figura 1-16 ilustra los usos del multiplexor F5MUX en un *slice*.



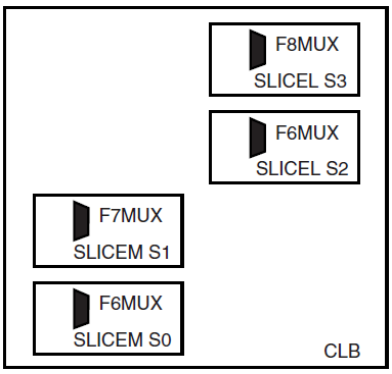
Fuente [8]

Figura 1- 16. Usos del multiplexor F5MUX: a) un multiplexor 4:1 b) función booleana de cinco variables c) función booleana de nueve variables.

1.5.6.2 El Multiplexor FiMUX

Este multiplexor puede tomar la denominación F6MUX, F7MUX, y F8MUX según su localización dentro del CLB. En general, cada FiMUX recibe en su entrada la salida del multiplexor del inmediato número inferior; por ejemplo, dos

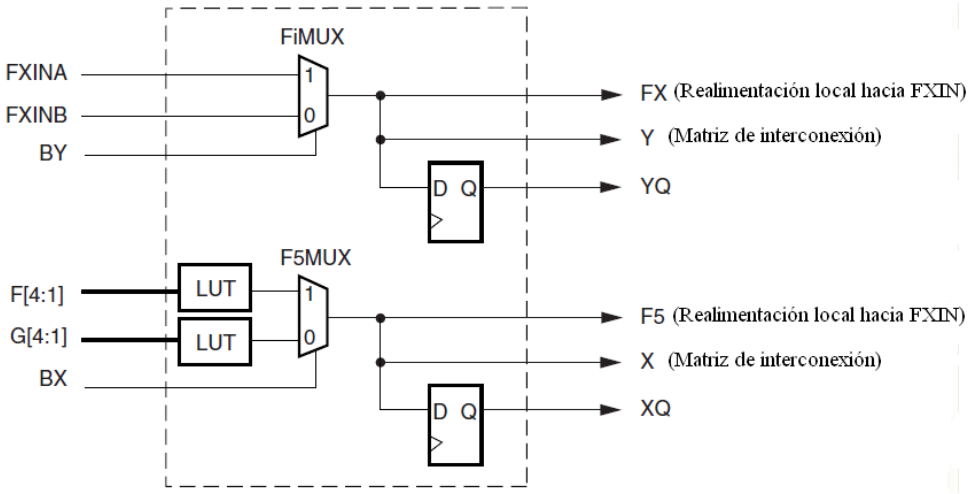
multiplexores F6MUX serán las entradas del multiplexor F7MUX. Estos multiplexores reciben las denominaciones F6MUX, F7MUX y F8MUX debido a que combinados con los LUTs de cada *slice* pueden implementar cualquier función de seis, siete y ocho variables, respectivamente. En la Figura 1-17 se puede observar la localización de los FiMUX dentro del CLB.



Fuente [8]

Figura 1- 17. Posición de los multiplexores FiMUX dentro del CLB

Los multiplexores F5MUX y FiMUX manejan las salidas del *slice* X e Y respectivamente, o a la entrada D de los *flip-flops*, como lo indica la Figura 1-18. También pueden manejar las salidas para realimentación local FX y F5, con las cuales se pueden agrupar estos elementos en cascada dentro del CLB.



Fuente [8]

Figura 1- 18. Salidas de los Multiplexores dedicados en un *slice*

En la Tabla 1-3 se presenta un resumen de la capacidad de cada multiplexor dedicado en un CLB para implementar multiplexores de más entradas así como funciones booleanas.

Tabla 1- 3. Capacidades de los multiplexores dedicados F5MUX y FiMUX

Mux	Mux Utilizado	Entradas	Número de entradas por función		
			Para cualquier función booleana	Para multiplexores	Para funciones booleanas parciales
F5MUX	F5MUX	LUTs	5	6 (mux 4:1)	9
FiMUX	F6MUX	F5MUX	6	11 (mux 8:1)	19
	F7MUX	F6MUX	7	20 (mux 16:1)	39
	F8MUX	F7MUX	8	37 (mux 32:1)	79

En un *slice* existen también multiplexores estáticos que se encargan de conducir las señales internas a través de los recursos lógicos en el *slice*. Por ejemplo, el multiplexor FXMUX conduce la señal de salida del F-LUT hacia la salida X del CLB, como se puede observar en la Figura 1-10. La línea de selección de estos multiplexores permanece estática y su valor es asignado durante el proceso de configuración del dispositivo.

1.5.7 CADENA DE ACARREO Y LÓGICA ARITMÉTICA

Los LUTs del *slice* pueden implementar funciones aritméticas, sin embargo esta opción no es óptima debido al retardo acumulado por la lógica de los LUTs. Para evitar este inconveniente, los FPGAs de la plataforma Spartan 3 poseen un conjunto dedicado de compuertas, multiplexores y conexiones para implementar algunas funciones aritméticas y mejorar el rendimiento.

1.5.7.1 Sumadores y Cadena de Acarreo

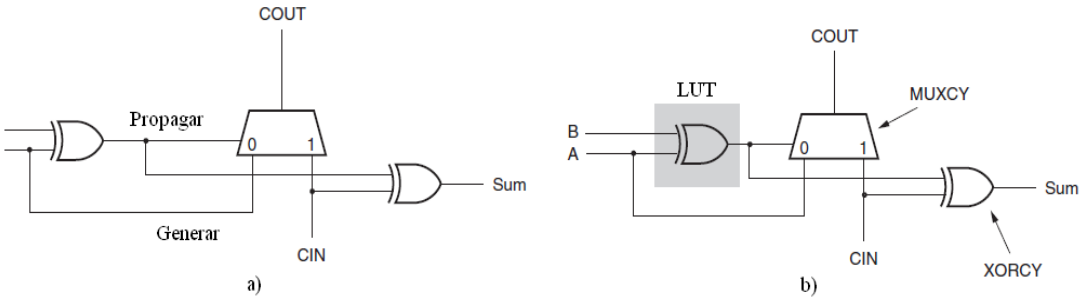
Los recursos lógicos que posee cada *slice* para implementar un sumador se basa en la necesidad de generar o propagar la señal de *carry* (sumador con acarreo anticipado), tal como lo indica la Tabla 1-4. Este esquema hace posible

implementar un sumador utilizando la lógica que se ilustra en la Figura 1-19a, en la cual la salida de *carry* puede ser definida por un multiplexor controlado por la señal “propagar”, la cual selecciona la entrada de *carry* (CIN) cuando es alta o seleccione A (o B) cuando es baja.

Tabla 1- 4. Generación y propagación de la señal de carry

A	B	Propagar	Generar
0	0	0	0
0	**1**	**1**	0
1	**0**	**1**	0
1	**1**	0	**1**

Con el propósito de implementar un sumador mediante los recursos lógicos de la Figura 1-19a cada *slice* en un CLB posee una compuerta XOR denominada XORCY y un multiplexor denominado MUXCY, en sus dos porciones (superior e inferior). La ventaja de esta estructura es que proporciona una rápida propagación de la señal de acarreo (*carry*) ya que solamente requiere el retardo de un multiplexor 2:1 y utiliza sólo un LUT del *slice* como se muestra en la Figura 1-19b. Estos recursos son idénticos en ambos *slices* (SLICEL y SLICEM).

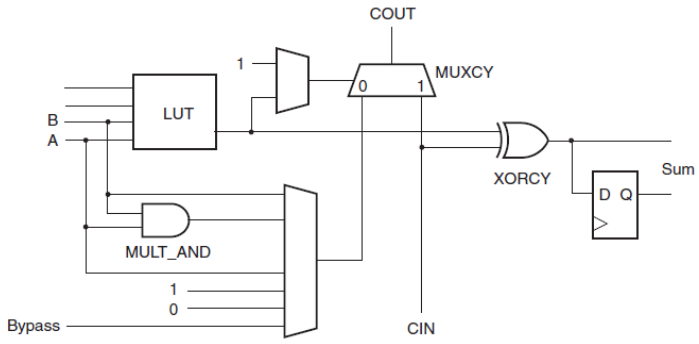


Fuente [8]

Figura 1- 19. Sumador con acarreo anticipado: a) Recursos lógicos para generar un sumador completo b) Recursos lógicos disponibles en un *slice* para generar un sumador completo

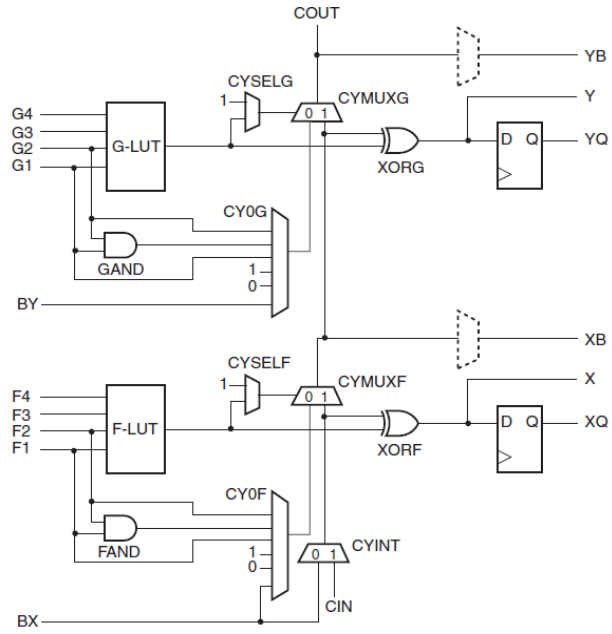
En los FPGAs de Xilinx, la combinación del LUT con el *flip-flop*, los multiplexores estáticos y la lógica aritmética se denomina celda lógica. La Figura 1-20 muestra la disposición de los recursos lógicos en una celda lógica de las FPGAs de la plataforma Spartan 3.

En la Figura 1-21 se muestra la disposición de la lógica aritmética en un *slice*. Los multiplexores estáticos denominados CY0F/G, CYSELF/G y CYINT permiten inicializar la cadena de acarreo con valores fijos (‘1’ o ‘0’) o con valores provenientes de la lógica interna del *slice* o desde cualquier parte del dispositivo.



Fuente [8]

Figura 1- 20. Esquema simplificado de una celda lógica en las FPGAs de la plataforma Spartan 3

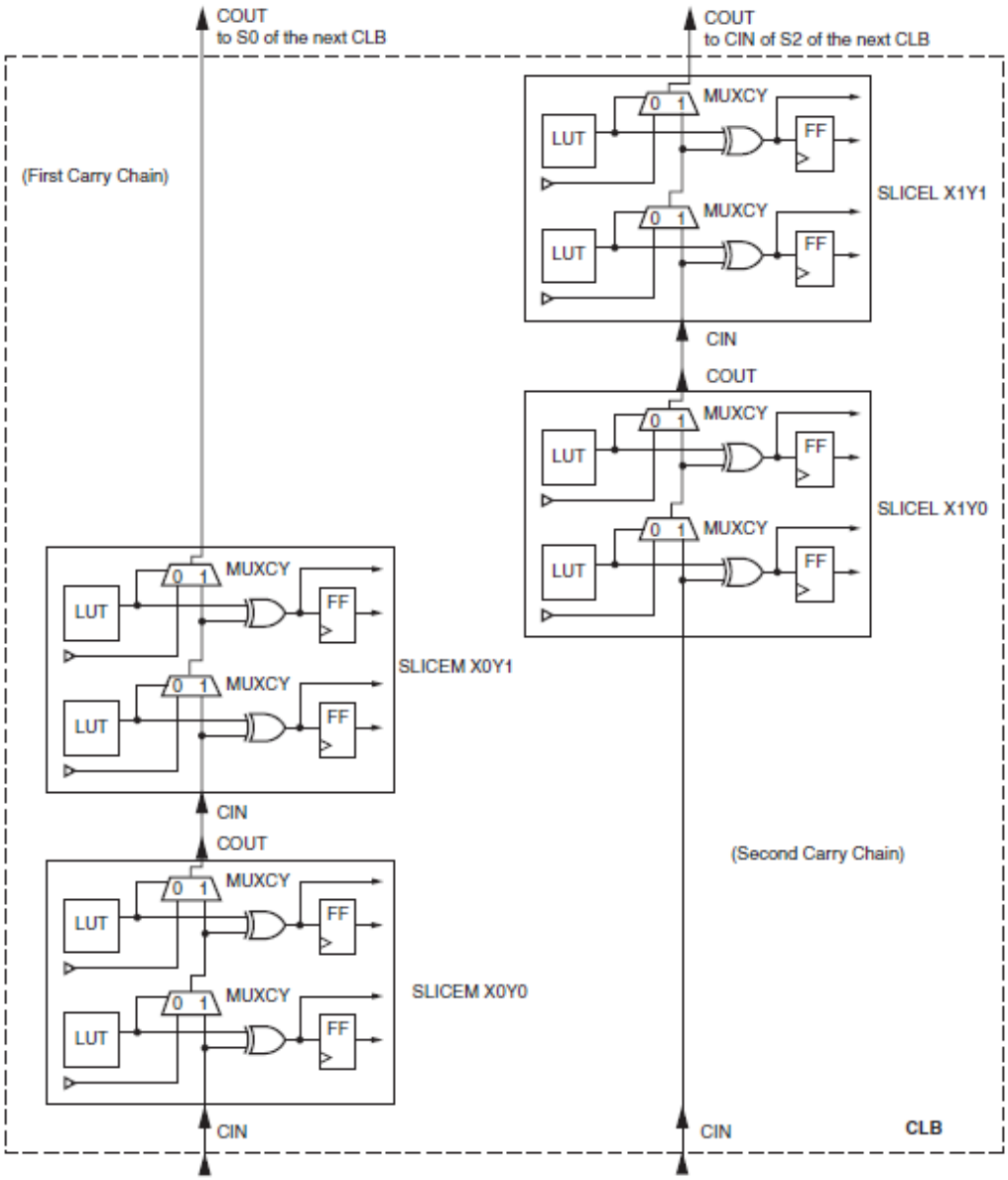


Fuente [8]

Figura 1- 21. Esquema simplificado de la lógica aritmética en un *slice*

Internamente, en un *slice*, la salida de *carry* de la porción inferior se conecta directamente a la entrada de *carry* de la porción superior a través del multiplexor CYMUXF como lo muestra la Figura 1-21. En un CLB, las cadenas de acarreo se concatenan entre *slices* de un mismo lado (SLICEM en el izquierdo y SLICEL en

el derecho), conectando la salida de *carry* (COUT) del *slice* inferior a la entrada de *carry* (CIN) del *slice* superior, como lo indica la Figura 1-22. La salida de *carry* (COUT) del *slice* superior del CLB se conecta a la entrada de *carry* (CIN) del *slice* inferior del siguiente CLB. Con esta estructura se tendrán dos cadenas de acarreo que se ejecutarán verticalmente, desde abajo hacia arriba en una columna de CLBs. El número total de cadenas de acarreo en un FPGA de la plataforma Spartan 3 será el doble del número de columnas de CLBs.

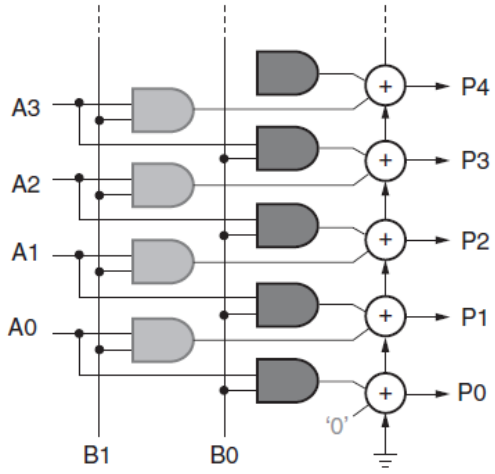


Fuente [8]

Figura 1- 22. Conexiones de la cadena de acarreo dentro de un CLB

1.5.7.2 Recursos para multiplicación

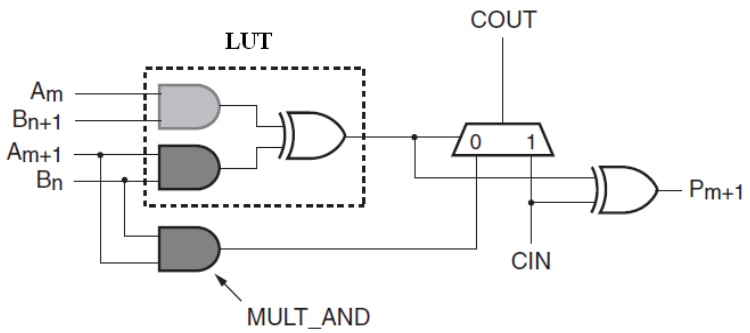
La multiplicación binaria de un bit por un bit es equivalente a la función AND de los dos operandos. Tomando en cuenta esto, una multiplicación de operandos de más bits puede realizarse generando productos parciales, esto es multiplicando un operando por cada bit del otro operando, y luego sumando estos resultados parciales para obtener el producto total, tal como se ilustra en la Figura 1-23.



Fuente [8]

Figura 1- 23. Multiplicación por productos parciales

Para implementar un multiplicador mediante el esquema de productos parciales, cada porción de un *slice* cuenta con una compuerta AND, la cual se asocia con la lógica de acarreo y el LUT, como lo indica la Figura 1-24.



Fuente [8]

Figura 1- 24. Multiplicación mediante productos parciales en un *slice*

La compuerta dedicada MULT_AND toma el nombre de FAND en la porción inferior del *slice* y GAND en la porción superior.

1.5.8 BLOQUES EMBEBIDOS

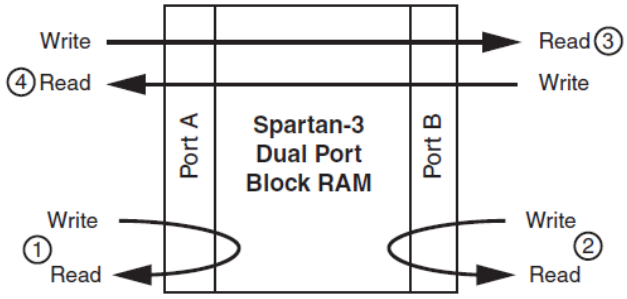
Los FPGAs de la plataforma Spartan 3 poseen tres bloques embebidos dentro del arreglo de CLBs: Bloques de RAM embebida, Administradores digitales de reloj (DCMs) y Multiplicadores.

1.5.8.1 Bloques de RAM embebida

Los bloques de memoria RAM embebida en los FPGAs de la plataforma Spartan 3 están organizados en columnas en medio del arreglo de CLBs. Dependiendo de la densidad del dispositivo dentro de cada familia, éstos pueden tener de una a cinco columnas de bloques de RAM embebida. Cada bloque de memoria RAM es físicamente una memoria RAM *dual-port* que puede ser configurada como una memoria de puerto simple. Los dos puertos independientes de los bloques de RAM tienen acceso al mismo rango de memoria, y cada uno tiene sus propias señales de control, su línea de datos y señal de reloj.

Cada bloque de RAM embebida contiene 18432 bits (18 Kbits) para almacenamiento de datos. La relación de aspecto[2] de cada bloque es configurable y los bloques pueden combinarse en cascada para formar memorias de mayor capacidad. Los dos puertos independientes en los bloques de RAM embebida se denominan A y B, y pueden soportar el flujo de datos que se observan en la Figura 1-25 para implementar memorias RAM *single-port* y *dual-port*.

[2] **Relación de Aspecto:** La relación entre el bus de datos y el bus de direcciones.



Fuente [8]

Figura 1- 25. Flujo de datos posibles en un Bloque de RAM embebida

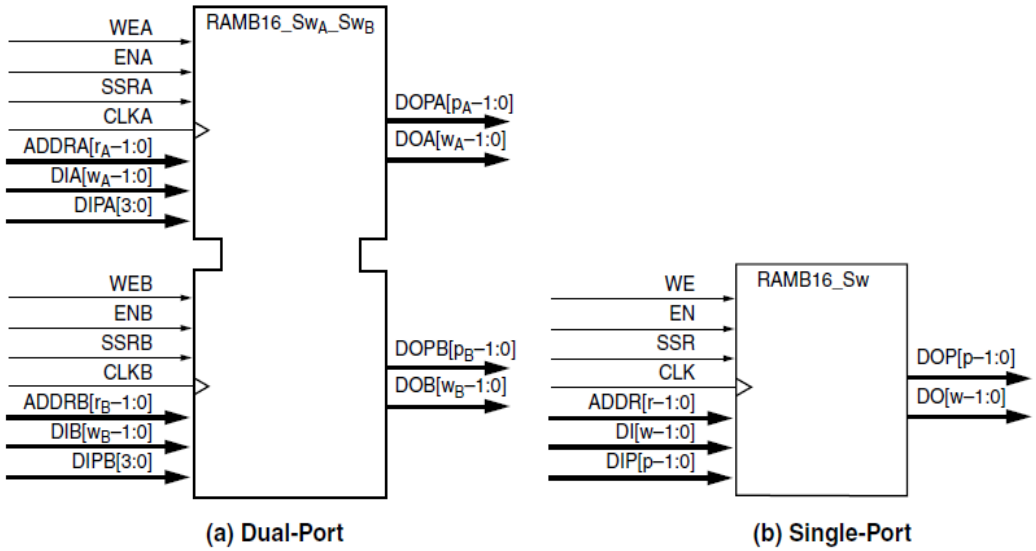
1. Con el puerto A, el bloque de RAM embebida se comporta como una RAM *single-port* independiente, que realiza tanto operaciones de lectura como escritura en el mismo puerto y utiliza un simple conjunto de líneas de dirección.

2. Con el puerto B, el bloque de RAM embebida se comporta como una RAM *single-port* independiente que realiza tanto operaciones de lectura como escritura en el mismo puerto y utiliza un simple conjunto de líneas de dirección.

3. El puerto A es el puerto de escritura y tienen su propio conjunto de líneas de dirección para escritura, el puerto B es el puerto de lectura y tienen su propio conjunto de líneas de dirección de lectura. El ancho de datos para los puertos A y B puede ser diferente (en términos de relación de aspecto).

4. El puerto B es el puerto de escritura y tienen su propio conjunto de líneas de dirección para escritura, el puerto A es el puerto de lectura y tienen su propio conjunto de líneas de dirección de lectura. El ancho de datos para los puertos A y B puede ser diferente (en términos de relación de aspecto).

La Figura 1-26 ilustra un bloque de memoria RAM embebida en sus modalidades *dual-port* y *single-port*. En la Tabla 1-5 se presenta una breve descripción de las señales de interfaz que poseen los bloques de RAM embebida.



Fuente [8]

Figura 1- 26. Bloque de RAM embebida: a) Dual-port y b) Single-port

Tabla 1- 5. Señales de la interfaz de un bloque de RAM embebida

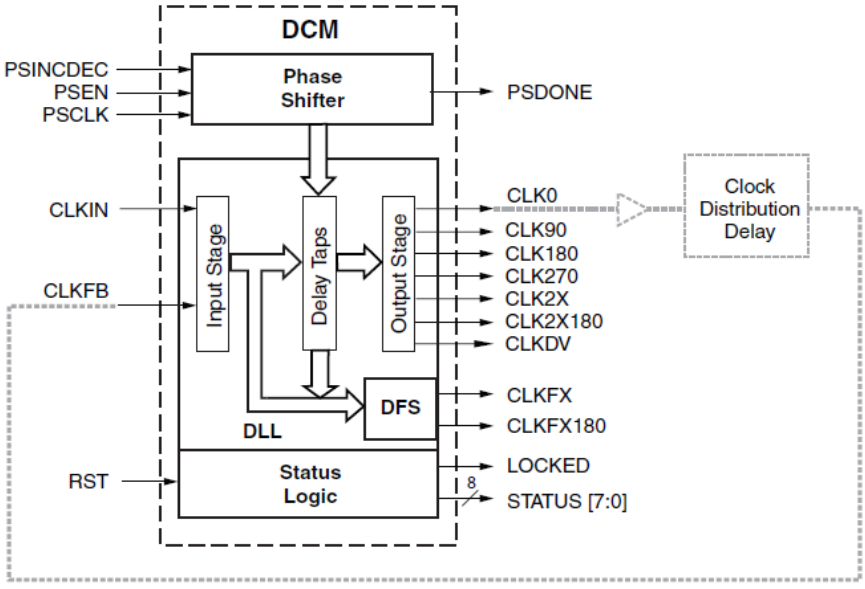
Single-port	Dual-port		Dirección	Descripción
	Puerto A	Puerto B		
DI	DIA	DIB	Entrada	Bus de datos de entrada
DIP	DIPA	DIPB	Entrada	Bus de entrada de datos de paridad[3]
DO	DOA	DOB	Salida	Bus de salida de datos
DOP	DOPA	DOPB	Salida	Bus de salida de datos de paridad
ADDR	ADDRA	ADDRB	Entrada	Bus de dirección
WE	WEA	WEB	Entrada	Señal de habilitación de escritura
EN	ENA	ENB	Entrada	Señal de habilitación del reloj
SSR	SSRA	SSRB	Entrada	Reset/Set sincrónico
CLK	CLKA	CLKB	Entrada	Señal de reloj

La operación de escritura es habilitada mediante la señal WE; cuando esta señal está activa, los datos presentes en el puerto DI son almacenados en la localidad especificada por el bus de direcciones ADDR, siempre que la señal de habilitación EN esté activa. Si la señal de habilitación de escritura WE no está activa, se realizará una operación de lectura en la celda especificada por el bus de direcciones ADDR, siempre que la señal de habilitación EN esté activa. Cuando la señal de habilitación EN no está activa ningún dato es escrito en la memoria y el bus de datos de salida permanece en su último estado (con el último dato leído).

[3] Los datos de este bus, no hacen referencia al concepto típico de paridad como método de detección de errores; este bus puede ser usado como bits de datos adicionales o para proporcionar información adicional de la palabra de dato almacenada en el bloque de RAM embebida.

1.5.8.2 Administradores Digitales de Reloj (DCMs)

Un bloque de gran utilidad que poseen los FPGAs de la plataforma Spartan 3 es el Administrador Digital del Reloj, con el cual el diseñador es capaz de manipular los parámetros de frecuencia y fase de la señal de reloj del sistema, pudiendo sintetizar nuevas señales de reloj. Además de esta utilidad, los DCMs son utilizados para eliminar algunos problemas de sincronismo en aplicaciones de alta frecuencia y alto rendimiento como lo es el sesgado de la señal de reloj (*skew*). Un DCM está constituido por cuatro bloques funcionales: un bloque de Lazo de Seguimiento de Retardo (*Delay-Locked Loop – DLL*), el Sintetizador Digital de Frecuencia (*Digital Frequency Synthesizer – DFS*), el bloque de desplazamiento de fase (*Phase Shifter – PS*) y un bloque de lógica de estado, como lo indica la Figura 1-27.

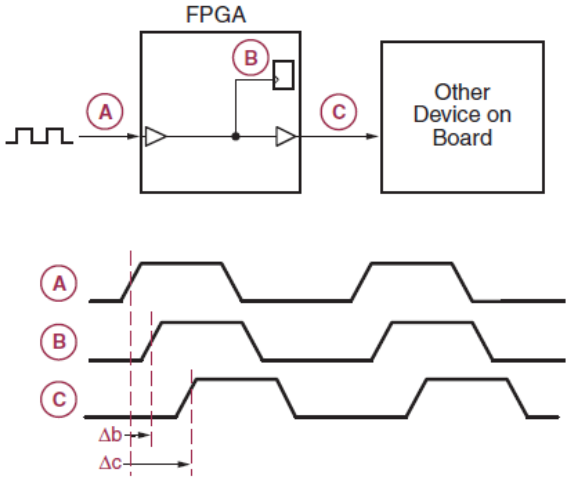


Fuente [8]

Figura 1- 27. Diagrama de Bloques de un DCM

El bloque de lazo de seguimiento de retardo (DLL) es el bloque encargado de eliminar el sesgado de la señal de reloj. El sesgado de la señal de reloj es un problema inherente de los sistemas sincrónicos y consiste en la desviación del alineamiento a la fase cero debido a retardos en las rutas que toma la señal de reloj dentro del dispositivo, causando que dicha señal llegue a diferentes puntos

del sistema en tiempos diferentes, como se describe en la Figura 1-28. El sesgado de reloj (*skew*) hace que se requieran parámetros de *time set-up*[4] *y time hold*[5] mayores, lo cual provoca que se requiera un período de reloj mayor, disminuyendo el rendimiento global del sistema.

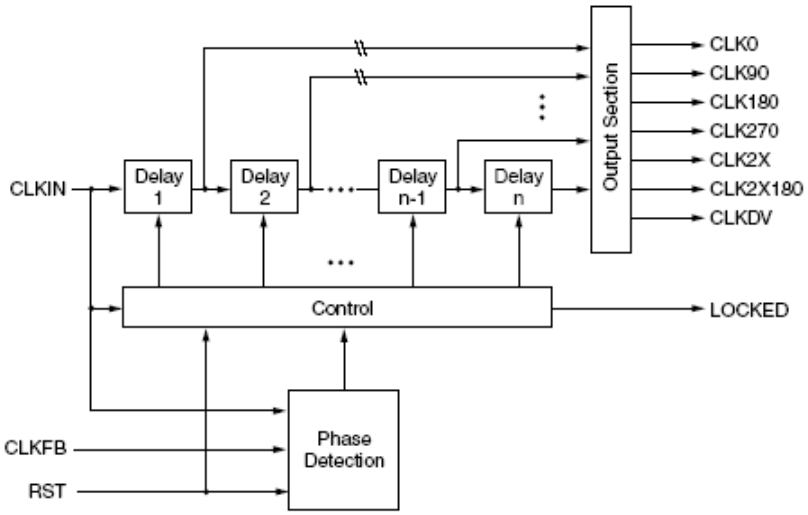


Fuente [8]

Figura 1- 28. Sesgado de la señal de reloj en sistemas sincrónicos

El lazo de seguimiento de retardo (DLL) es un sistema retroalimentado, como lo indica la Figura 1-29. Al inicio de la operación del FPGA, la salida de reloj (CLK0) es distribuida, a través de la red de distribución de reloj, hacia los diferentes elementos sincrónicos del sistema y luego de pasar por todas las rutas utilizadas en la red de reloj, retorna hacia el bloque DLL a través de la entrada CLKFB. Mediante la señal CLKFB, el bloque DLL calcula el desfase con respecto a la señal de reloj de entrada (CLKIN), y activa el número de elementos de retardo suficientes para compensar el desfase y alinear la señal de salida CLK0 a la fase cero. Una vez que el desfase está compensado, el bloque de control activa la señal LOCKED.

[4] **Time Set-up:** Es el tiempo mínimo que una señal debe mantenerse estable antes de un evento de la señal de reloj para ser considerado confiable durante el muestreo.
[5] **Time Hold:** Es el tiempo mínimo que debe retenerse una señal después del evento de la señal de reloj para ser considerado confiable durante el muestreo.



Fuente [9]

Figura 1- 29. Diagrama de bloques de un DLL

Mediante el sintetizador digital de frecuencia, un DCM puede generar diferentes señales de reloj, las cuales se derivan de la señal de entrada y un coeficiente resultante de la división de dos números enteros, tal como lo muestra la Ecuación 1. El sintetizador digital de frecuencia puede trabajar en conjunto o separado del bloque de lazo de seguimiento de retardo (DLL). En el caso de no trabajar con el bloque DLL las señales de salida del sintetizador digital de frecuencia no tienen ninguna relación de fase con la señal de entrada de reloj.

$$F_{CLKFX} = F_{CLKIN} \bullet \frac{CLKFX\_MULTIPLY}{CLKFX\_DIVIDE} \qquad (1)$$

El factor de multiplicación CLKFX_MULTIPLY puede estar en el rango de 2 a 32 y el factor de división CLKFX_DIVIDE en el rango de 1 a 32.

El bloque de desplazamiento de fase controla la relación de fase existente en todas las nueve salidas del DCM con respecto a la fase de la señal de reloj de entrada. Con este bloque es posible realizar desfases controlados de la señal de reloj en una fracción fija o variable de su periodo. El bloque de desplazamiento de fase puede generar señales de reloj con cuatro diferentes tipos de desfase:

Desplazamientos de fase fijos

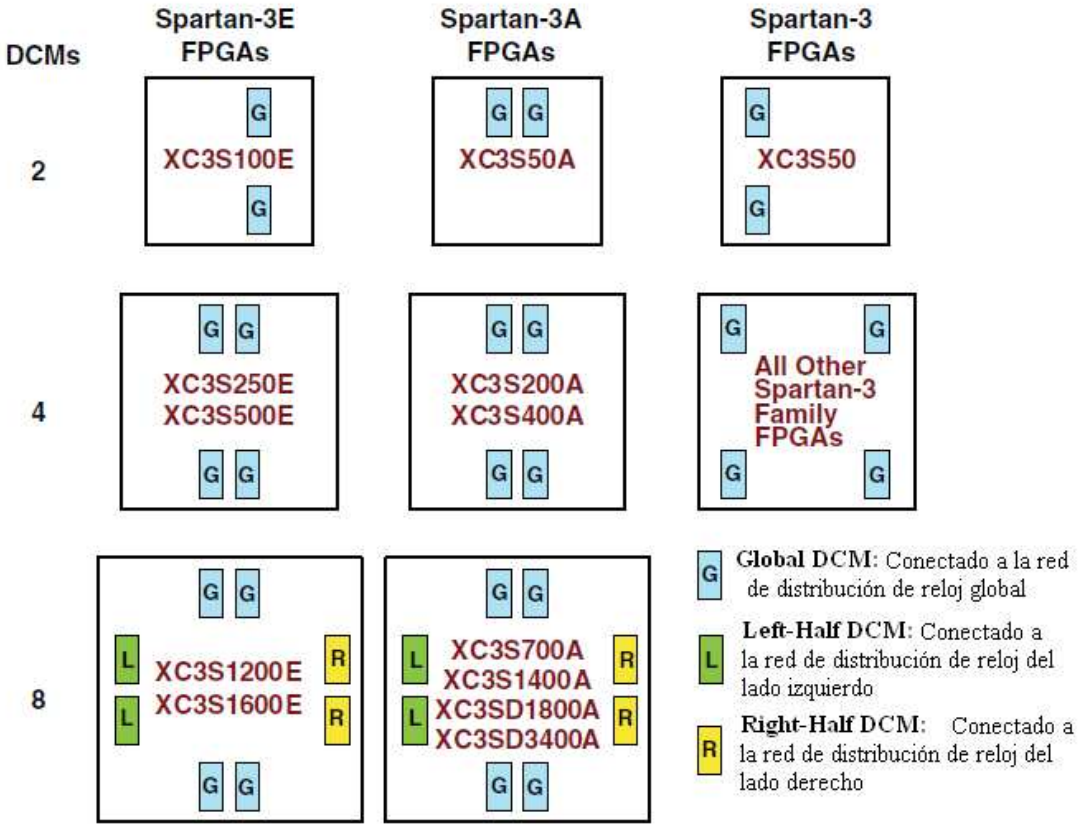
1. Salidas desfasadas medio periodo, con desfases de 0º y 180º.
2. Salidas con desfase en cuadratura, con desfases de 0º, 90º, 180º y 270º.
3. Desplazamientos de fase finos, con una resolución de desfase de 1/256 del período de la señal de reloj de entrada.

Desplazamientos de fase variable

4. Desplazamientos de fase variables, son controlados en la aplicación desarrollada en el FPGA; pueden variar en pasos de 1/256 del período de la señal de reloj de entrada en la familia Spartan-3 y en pasos determinados por elementos de retardo embebidos en el silicio del dispositivo (independientes de la frecuencia del reloj de entrada) en las familias Spartan-3A y Spartan-3E. Las señales PSEN, PSINCDEC, PSCLK y PSDONE son las interfaces del bloque de desplazamiento de fase cuando se utiliza un esquema de desplazamiento de fase variable.

El bloque de Lógica de Estado, indica el estado actual del DCM mediante cuatro señales: los tres bits menos significantes del bus STATUS que indican el estado de las operaciones de los bloques DLL y PS, y la señal LOCKED.

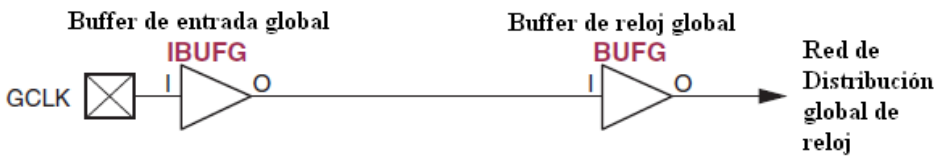
Dependiendo de la familia y el tamaño del dispositivo (en celdas lógicas) el número y localización de los DCMs puede variar. En la Figura 1-30 se muestra el número y la localización relativa de los DCMs en todas las familias dentro de la plataforma de FPGAs Spartan 3. Los DCMs ubicados en la parte inferior y superior de la oblea del FPGA tienen conexiones dedicadas a la red de distribución de reloj global; los ubicados en el lado derecho del dispositivo tienen conexiones dedicadas a la red de distribución de reloj de la mitad derecha del FPGA y los ubicados en el lado izquierdo a la red de distribución de reloj de la mitad izquierda del dispositivo. Más adelante se describe con más detalle la red de distribución de reloj.



Fuente [8]

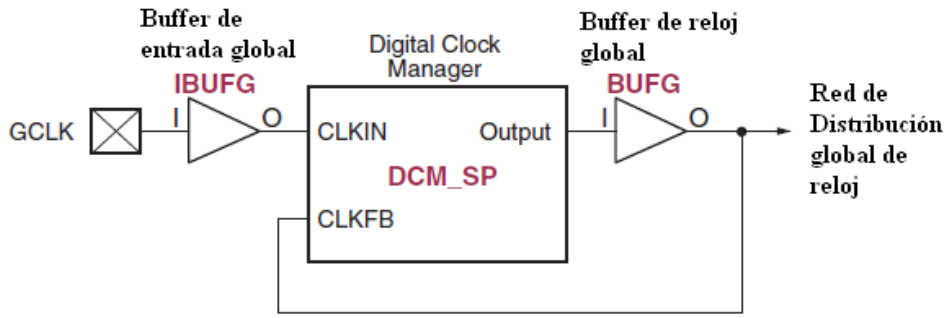
Figura 1- 30. Número y localización relativa de los DCMs en las FPGAs de la plataforma Spartan 3

Un DCM es un elemento opcional en la red de distribución de reloj. Normalmente, la señal de reloj, que proviene del oscilador externo, ingresa a través de un buffer de entrada global y va hacia un buffer de reloj global después del cual se propaga sobre la red de distribución de reloj, como lo muestra la Figura 1-31. Cuando se utiliza el DCM, éste se ubica entre la trayectoria del buffer de entrada global y el buffer de reloj global como lo ilustra la Figura 1-32.



Fuente [8]

Figura 1- 31. Trayectoria de la señal de reloj sin usar DCMs



Fuente [8]

Figura 1- 32. Trayectoria de la señal de reloj usando DCMs

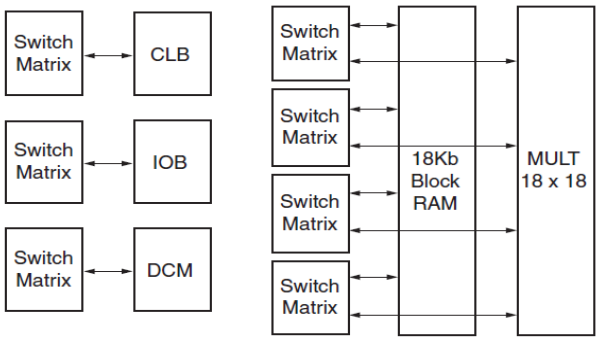
1.5.8.3 Multiplicadores Embebidos

Los multiplicadores embebidos son bloques dedicados a obtener el producto de dos números de 18 bits en complemento de dos. Estos bloques proporcionan una alternativa más eficiente de realizar la operación de multiplicación en términos de recursos lógicos en comparación con una solución que utiliza CLBs. Los multiplicadores embebidos pueden realizar aplicaciones tales como multiplicaciones de dos números con signo, multiplicación de dos números sin signo, además pueden ser utilizados para obtener el complemento de dos de un número así como su magnitud. Los multiplicadores embebidos físicamente se encuentran junto a los bloques de RAM embebida y comparten ciertos recursos de interconexión con el fin de obtener un manejo adecuado de los datos.

1.5.9 MATRIZ DE INTERCONEXIÓN

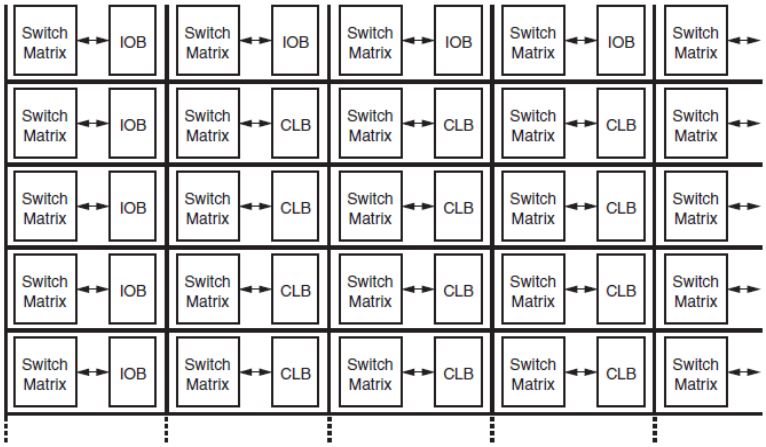
La matriz de interconexiones programables es la encargada de trazar las rutas entre las señales de entrada y salida de los elementos funcionales del FPGA como son los CLBs, los bloques de I/O, y los bloques embebidos. Para realizar la interconexión entre estos elementos, los FPGAs de la plataforma Spartan 3 cuenta con cuatro tipos de conexiones: líneas largas, líneas hex, líneas dobles y líneas directas. Estas líneas de interconexión se conectan a los elementos funcionales del FPGA mediante un elemento denominado matriz de conmutación; más de una matriz de conmutación puede estar asociada a un elemento funcional,

como muestra la Figura 1-33. En su conjunto las matrices de conmutación forman un arreglo en todo el dispositivo como lo indica la Figura 1-34.



Fuente [8]

Figura 1- 33. Matrices de conmutación asociadas a los bloques funcionales del FPGA



Fuente [8]

Figura 1- 34. Arreglo de matrices de conmutación en el FPGA

Las líneas largas están agrupadas en grupos de 24 y distribuidas tanto de manera vertical como horizontal en el FPGA. Estas líneas conectan uno de cada seis bloques funcionales y por su baja capacitancia son ideales para conducir señales de alta frecuencia. Estas líneas son una alternativa confiable para llevar señales de reloj en el caso de que las líneas globales de reloj estén ocupadas.

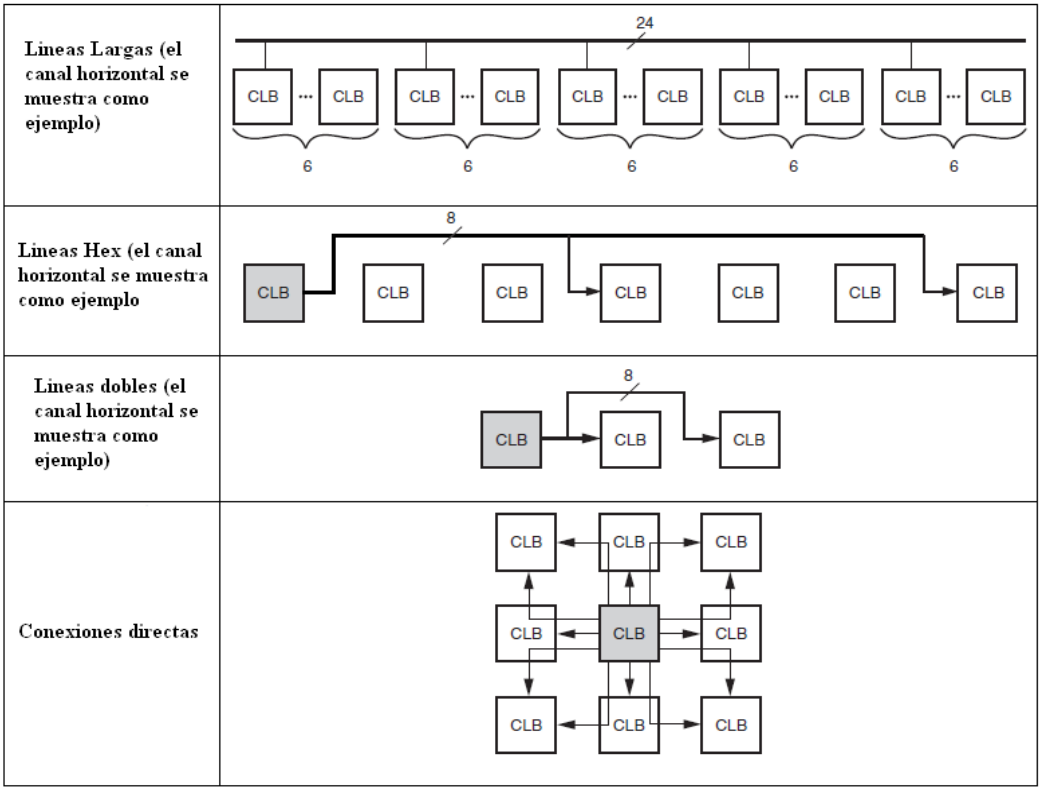
Las líneas Hex están agrupadas en grupos de 8 y conectan uno de cada tres bloques funcionales, se encuentran distribuidas de manera horizontal y vertical en el dispositivo. Estas líneas cubren el desfase existente entre las líneas largas y las

líneas dobles [5]. La mayor conectividad que ofrecen estas líneas se ve contrastada con su menor capacidad de conducir señales de alta frecuencia.

Las líneas dobles están agrupadas en grupos de 8 y conectan un bloque funcional con otro y se encuentran distribuidas de manera vertical y horizontal en el dispositivo. Esta línea de interconexión ofrece más flexibilidad y mayor conectividad que las líneas largas y líneas hex.

Las líneas directas conectan un bloque funcional con sus vecinos tanto en sentido vertical, horizontal y diagonal. En total estas líneas de interconexión pueden conectar un bloque funcional con sus ocho bloques adyacentes.

La Figura 1-35 ilustra los tipos de interconexiones que se tienen en la matriz de interconexión de propósito general en los FPGA de la plataforma Spartan 3.



Fuente [8]

Figura 1- 35. Líneas de interconexiones en la matriz de interconexión de propósito general

Además de los tipos de interconexiones antes mencionadas, la matriz de interconexión posee dos señales de control global:

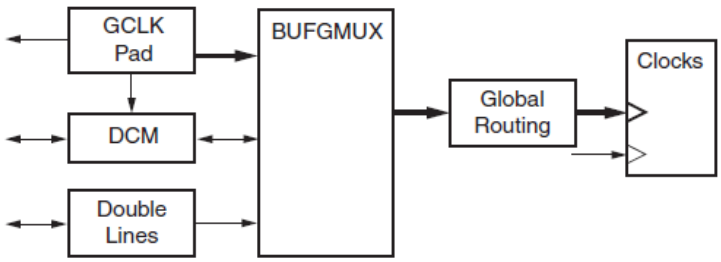
GSR (Global Set/Reset) cuando está en alto pone de manera asincrónica todos los registros y *flip-flops* en su estado inicial.

GTS (Global Three-State) cuando está en alto forza a todos los pines de entrada/salida al estado de alta impedancia.

1.5.10 RECURSOS DE SINCRONISMO

La infraestructura de reloj que poseen los FPGAs de la plataforma Spartan 3 proporciona una red adecuada para la distribución de señales de reloj a los elementos sincrónicos que posee el FPGA. Entre sus características más importantes, la red de distribución posee líneas de interconexión de baja capacitancia y bajo sesgado de la señal de reloj, lo cual la hace adecuada para la conducción de señales de alta frecuencia.

Los recursos de sincronismo son tres componentes conectados entre sí: los pines de entrada para la señal de reloj (incluyendo el buffer global de entrada), un multiplexor global de reloj, y la red de distribución de señales de reloj, tal como lo muestra la Figura 1-36. Los dispositivos de la plataforma Spartan 3 poseen ocho entradas globales de reloj, aunque existen algunas diferencias en las familias Spartan-3E y Spartan-3A las cuales poseen también ocho entradas de reloj adicionales en cada lado del dispositivo.



Fuente [8]

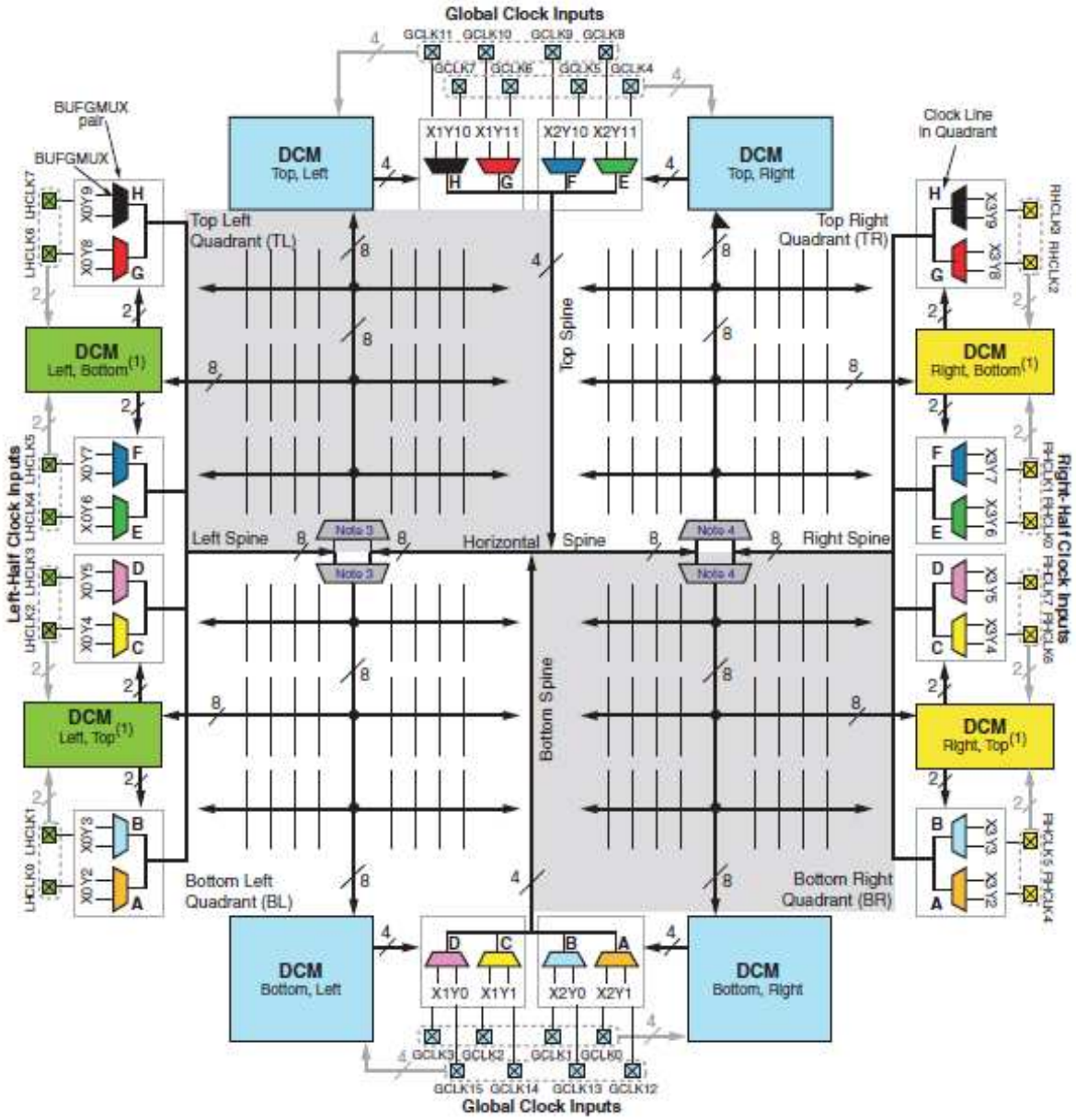
Figura 1- 36. Recursos de sincronismo en los FPGAs de la plataforma Spartan 3

Los multiplexores de reloj son multiplexores 2 a 1 y pueden conmutar dos señales de reloj distintas hacia la red de distribución de reloj o enviar directamente una señal de reloj actuando sólo como un buffer. Las señales que ingresan al multiplexor de reloj pueden provenir desde un pin de entrada de reloj, un DCM o desde una línea doble de la matriz de interconexión.

Con respecto a la red de distribución de señales de reloj, los FPGAs de la familia Spartan-3A y Spartan-3E difieren de la familia original Spartan-3 en que su arquitectura en la red de distribución de reloj está basada en un esquema de cuadrantes, como se indica en la Figura 1-37; en esta figura se indica también la localización de los multiplexores de reloj en forma de coordenadas. Cada cuadrante en la matriz puede soportar hasta ocho señales de reloj, a través de líneas de reloj, etiquetadas con las letras A hasta H.

Cabe mencionar que una entrada global de reloj se distribuye por todos los cuatro cuadrantes y una entrada de lado derecho o izquierdo se distribuye solo en los dos cuadrantes, sean derechos o izquierdos, respectivamente.

La matriz de distribución de reloj, en la familia original de FPGAs Spartan-3, tiene ocho entradas globales de reloj ubicadas en la parte inferior y superior de la oblea del dispositivo. Mediante la red de distribución de reloj cualquiera de las entradas globales puede conducir la señal de reloj hacia cada uno de los CLBs en el dispositivo, por medio de las líneas principales de reloj. Después, mediante líneas secundarias, la señal de reloj llega a los elementos sincrónicos de cada *slice* [9].



Fuente [8]

Figura 1- 37. Red de distribución de señales de reloj basado en un esquema de cuadrantes para las familias de FPGAs Spartan-3A y Spartan-3E

1.5.11 BLOQUE DE ENTRADA/SALIDA (IOB)

Todas las familias de la plataforma Spartan 3 comparten una estructura básica en los bloques de entrada/salida que permite definir la dirección de la señal en el IOB, la cual puede seguir una de tres vías (ver Figura 1-38):

- La ruta de entrada: lleva la señal desde el pin del FPGA a través de un elemento de retardo programable hacia la línea I. Después de los elementos de retardo, la señal puede tomar dos rutas alternativas a través de dos elementos de almacenamiento (IFF1 e IFF2) hacia las líneas IQ1 e IQ2. Las salidas I, IQ1 e IQ2 ingresan hacia la lógica interna del FPGA.

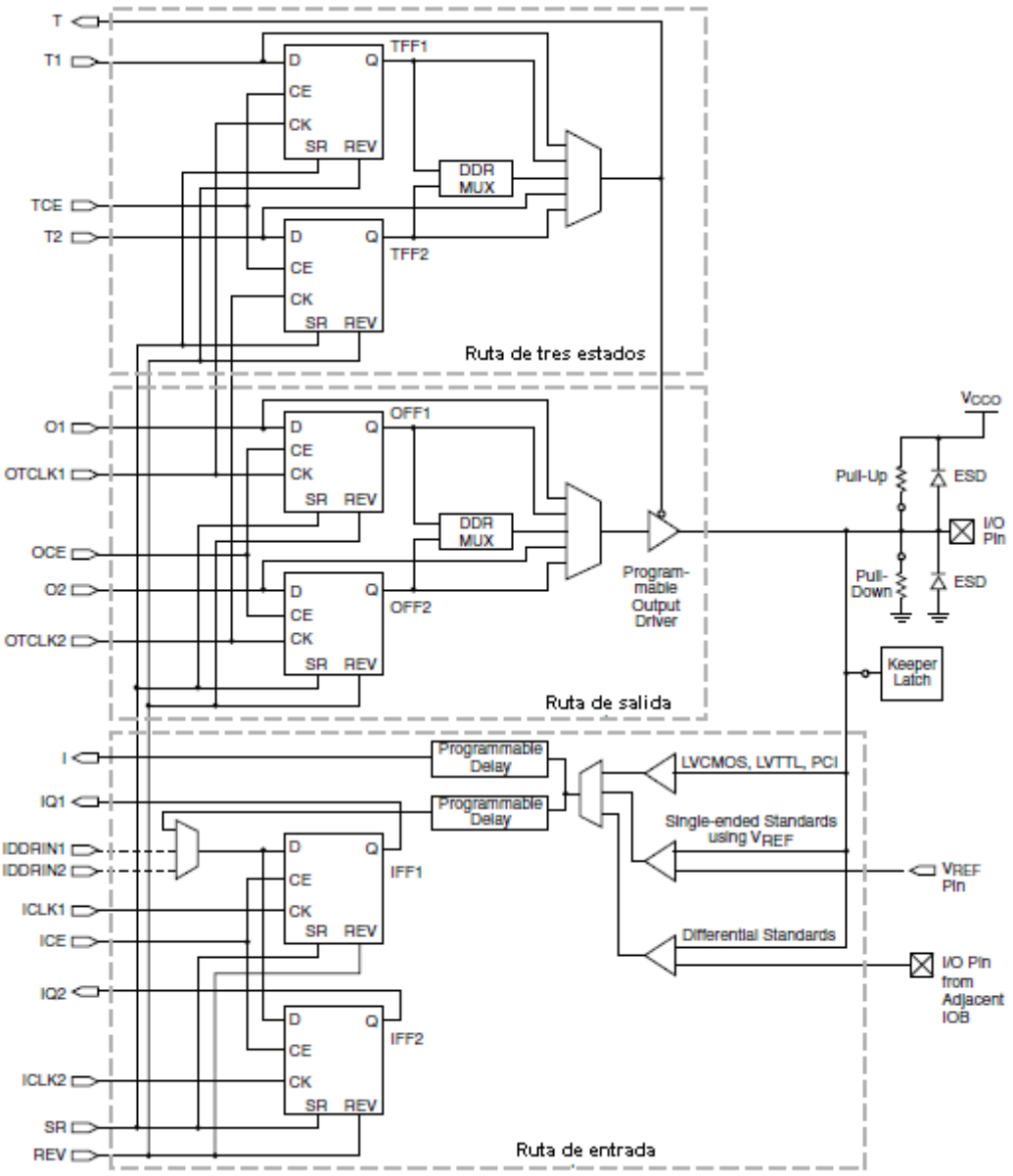
- La ruta de salida: empieza con las líneas O1 y O2 que provienen de la lógica interna del FPGA, y son llevadas a través de un multiplexor hacia un *driver* de tres-estados y después hacia el pin del FPGA. Alternativamente, el multiplexor puede insertar dos elementos de almacenamiento (OFF1 y OFF2) en esta ruta.

- La ruta de tres estados: determina cuándo el *driver* de salida está en alta impedancia. Las líneas T1 y T2, que provienen de la lógica interna del FPGA, se conducen a través de un multiplexor hacia la salida del *driver*. El multiplexor puede añadir dos elementos de almacenamiento (TFF1 y TFF2) en esta ruta.

Los tres pares de elementos de almacenamiento que existen en el IOB (dos por cada ruta) pueden ser configurados como *flip-flops tipo D* o como *latches* disparados por nivel. Estos elementos sincrónicos están controlados por la red de distribución de reloj.

Los elementos de almacenamiento en la ruta de salida y en la ruta de tres-estados pueden ser usados en conjunto, con un multiplexor especial para producir transmisión de doble tasa de datos (DDR). Esto se logra tomando datos sincronizados con el flanco de subida del reloj y convirtiéndolos en bits sincronizados tanto con el flanco de subida como con el de bajada. A la combinación de los dos registros y el multiplexor se le llama *flip flop* tipo D de doble tasa de datos (FDDR).

Los IOBs poseen resistencias de *pull-up* y *pull-down* configurables, las cuales establecen niveles altos o bajos, respectivamente, en las salidas del IOB; por lo general, estas resistencias son aplicadas a los bloques de entrada/salida no utilizados. También estos bloques poseen un circuito de retención (Keeper) que mantiene el último nivel lógico después de que el driver de salida ha sido puesto en alta impedancia.



Fuente [8]

Figura 1- 38. Estructura de un IOB en dispositivos de la plataforma Spartan 3

Los bloques de entrada/salida cuentan con circuitos de protección para descargas electro-estáticas basados en diodos, que pueden soportar hasta 2000V de descarga [10]. Además, los IOBs poseen un control de tasa de transición de voltaje (*slew rate*) que puede definir qué tan veloz puede ser una señal que ingresa o sale del IOB. Los IOBs poseen también un control de corriente de salida para los estándares LVTTL y LVCMOS, lo que permite adaptar los IOBs para manejar dispositivos que necesitan mayores corrientes para su activación. Los IOBs cuentan con un sistema para controlar la impedancia en los terminales del FPGA denominado Control Digital de Impedancia (DCI, por sus siglas en inglés), el cual permite adaptar la impedancia del pin del FPGA a la de la línea de transmisión que llega a éste; esto se logra mediante la configuración de un arreglo de resistencias internas. El Control Digital de Impedancia puede tener hasta cinco tipos de terminaciones, definidos por el arreglo de resistencias [11]. La adaptación de impedancias entre el FPGA y la línea de transmisión evita problemas de señal reflejada.

Los IOBs de los FPGAs de la plataforma Spartan 3 soportan varios estándares eléctricos para lógica de terminación simple y diferencial. En el *Data Sheet* de los FPGAs de la familia Spartan-3A, que se muestra en el Anexo 2, se enlistan los estándares eléctricos de lógica soportados por esta familia de la plataforma Spartan 3.

1.6 DEDUCCIÓN DE LOS BLOQUES FUNCIONALES

La mayor parte de los elementos lógicos de los CLB, así como los bloques embebidos expuestos en la sección anterior, pueden ser deducidos en el proceso de Síntesis de tres formas diferentes: mediante sentencias de códigos HDL que describan su comportamiento, mediante primitivas encontradas en librerías especiales, o mediante la generación de *IP Cores*[6].

[6] **IP Core**: Bloques de propiedad intelectual, son un conjunto de funciones lógicas predefinidas, optimizadas para las FPGAs de un fabricante determinado.

1.7 EL DISPOSITIVO XC3S700A

Para el presente proyecto se utiliza el FPGA XC3S700A perteneciente a la familia Spartan-3A, que posee un sistema de 700 mil compuertas lógicas. Las características de los bloques funcionales de este FPGA se resumen en la Tabla 1-6.

Tabla 1- 6. Características del dispositivo XC3S700A

FPGA XC3S700A		
Sistema de compuertas		700 K
Celdas Lógicas		13248
Arreglo de CLBs	**Filas**	48
	Columnas	32
	CLBs totales	1472
	Slices Totales	5888
RAM Distribuida		92 Kbits
Bloque de memoria RAM		360 Kbits
Multiplicadores dedicados		20
DCMs		8
Entradas/Salidas de usuario		372

1.8 MÓDULO DE DESARROLLO SPARTAN-3A *STARTER KIT*

Xilinx, al igual que otras compañías dedicadas a la fabricación de FPGAs, posee su propio software para desarrollar aplicaciones con sus FPGAs, así como también módulos de desarrollo que incluyen sus dispositivos con una amplia gama de periféricos que expande el campo de aplicación de estos módulos. En el presente proyecto se utiliza el módulo de desarrollo *Spartan-3A Starter Kit,* el cual posee las siguientes características [12]:

- El módulo posee un FPGA XC3S700A de la familia Spartan-3A que posee un sistema de compuertas de 700K equivalente a 13248 celdas lógicas, en el paquete tipo FG484, el cual posee 372 pines.

- El módulo cuenta con cuatro memorias de configuración:
    1. Una memoria flash PROM de Xilinx de 4Mb de capacidad.

2. Una memoria flash paralela de 32Mb de capacidad.
3. Dos memorias de configuración serial flash SPI, ambas de 16 Mb de capacidad.

- Tiene un controlador JTAG embebido, el cual facilita la tarea de configuración del FPGA y la programación de las memorias de configuración a través del puerto USB de un PC sin requerir un cable JTAG especial.

- Posee un oscilador de 50 MHz y un socket para un oscilador auxiliar. También el módulo tiene un conector tipo SMA[7] que puede ser utilizado como entrada o salida de señales de reloj.

- Con respecto a los periféricos, el módulo de desarrollo Spartan-3A Starter Kit tiene un puerto VGA con una resolución de 12 bits para generar 640x480 pixeles. Posee dos puertos RS-232 (DCE y DTE), un puerto Ethernet RJ-45, un puerto PS2 para teclado o mouse. El módulo también cuenta con un conversor analógico-digital (ADC) y un conversor digital-analógico (DAC) ambos con interfaz SPI[8], con una resolución de 14 bits y 12 bits, respectivamente. El módulo tiene una salida tipo jack para audio en estéreo. Posee además un LCD de dos filas de 16 caracteres. La tarjeta de desarrollo también tiene una memoria tipo SDRAM DDR2 de 512Mb de capacidad. Posee, además, cuatro *switches* de deslizamiento y cuatro *switches* tipo pulsador, como también un *encoder* de rotación (*Rotary Shaft Encoder*). El módulo de desarrollo tiene ocho diodos tipo led. Todos estos dispositivos están debidamente conectados a los pines del FPGA en la placa PCB. La asignación de pines del FPGA, asociados a los dispositivos antes mencionados, se encuentra documentada en el manual del módulo de desarrollo Spartan-3A *Starter Kit*, el cual se presenta en el Anexo 3; la

[7] **SMA**: SubMiniature version A, tipo de conector roscado para cable coaxial, utilizado para altas frecuencias, posee una impedancia característica de 50 ohms.
[8] **SPI**: Serial Peripheral Interface, es un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos.

información sobre las asignaciones deben incluirse en el archivo de restricciones del usuario (UCF) para la implementación final.

- El módulo de desarrollo también dispone de algunos puertos de expansión, los cuales son utilizados para conectar otros periféricos o interconectar dos módulos de desarrollo. Del mismo modo posee puertos para entradas con estándares diferenciales así como un puerto JTAG dedicado para operaciones de configuración, programación y depuración en tiempo real.

En la Figura 1-39 se muestra el módulo de desarrollo Spartan-3A *Starter Kit* y la localización de los elementos antes mencionados.

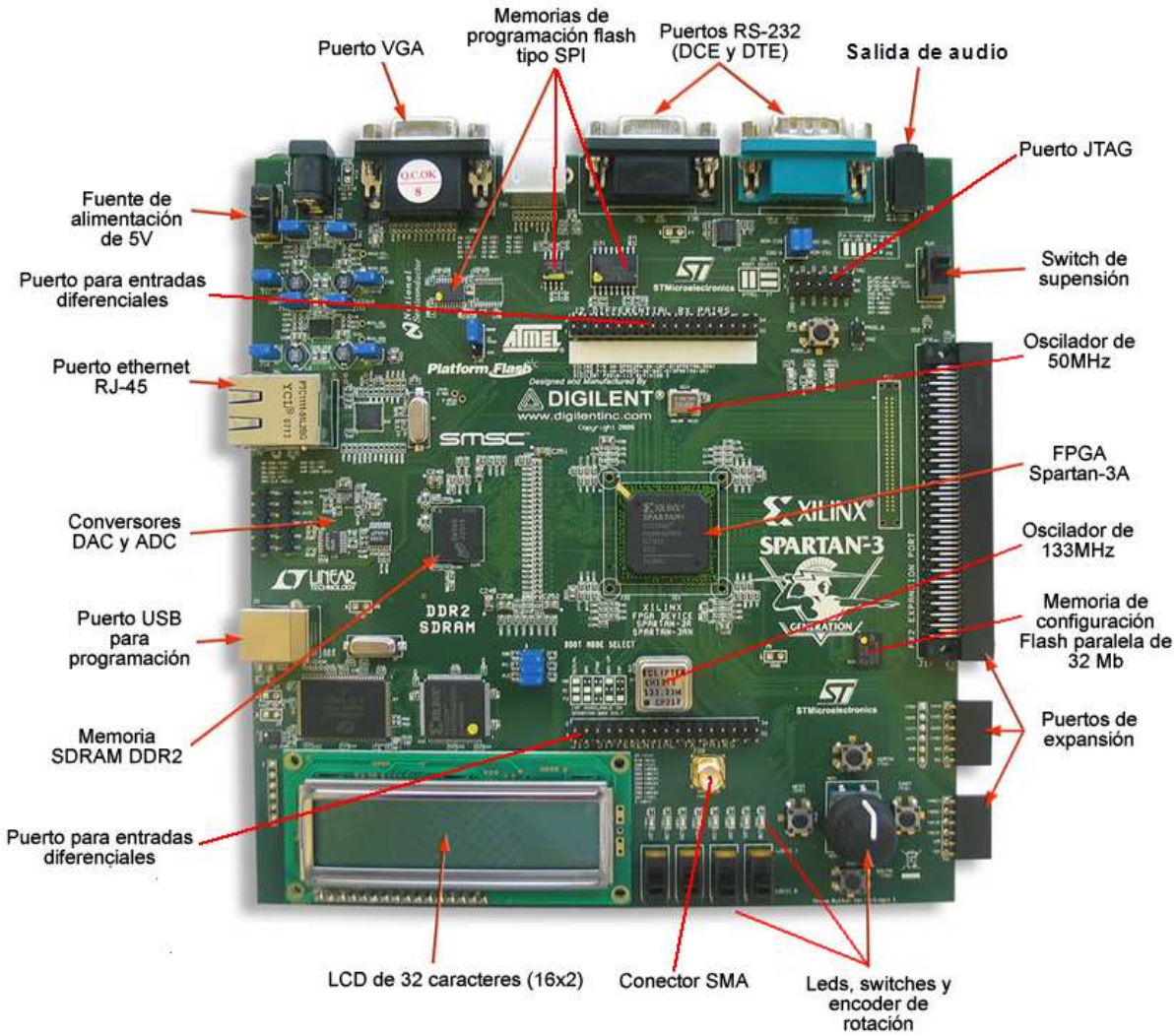


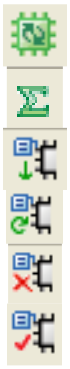
Figura 1- 39. Módulo de desarrollo Spartan-3A Starter Kit de Xilinx

El FPGA del módulo de desarrollo puede ser programado de dos formas: programando directamente el FPGA o programando una de las memorias de configuración. En el Anexo 4 se describen los modos de configuración del módulo de desarrollo *Spartan-3A Starter Kit* de Xilinx.

1.9 SOFTWARE DE DESARROLLO DE XILINX Y FLUJO DE DISEÑO

El software que Xilinx posee para desarrollar aplicaciones en sus FPGAs se denomina *Integrated Software Environment* (ISE), el cual posee un conjunto de herramientas y aplicaciones que intervienen en cada uno de los procesos del flujo de diseño descrito más adelante. El paquete de software ISE tiene dos versiones denominadas *Foundation y WebPack,* la última de las cuales es una versión gratuita y puede ser descargada desde el portal web de Xilinx. Para el presente proyecto se utilizará el paquete de software *ISE Foundation 10.1*, cuyo entorno de trabajo se indica en la Figura 1-40. El entorno de trabajo de esta herramienta se divide en cinco secciones:

1. **Barra de herramientas.** Aquí se encuentran disponibles los comandos frecuentemente usados en un diseño, dispuestos como botones, tanto aquellos que son estándar en cualquier barra de herramientas (Abrir, Imprimir, Pegar, etc.) como aquellos específicos del software de desarrollo, los cuales se muestran a continuación:



Ejecuta todo los procesos implicados en un diseño.

Abre en el espacio de trabajo una ventana con un resumen del diseño.

Ejecuta el proceso seleccionado en la Ventana de Procesos.

Vuelve a ejecutar el proceso seleccionado.

Detiene el proceso que se esté ejecutando.

Abre el cuadro de diálogo para editar las propiedades del proceso seleccionado en la Ventana de Procesos.

2. **Ventana de Fuentes.** En esta sección se indica los archivos fuentes que se han creado y añadido al proyecto, los cuales pueden estar asociados a la implementación como a la simulación del diseño. El usuario puede desplegar el menú de opciones '*Sources for:'* de esta ventana, donde puede escoger las opciones *Implementation, Behavioral Simulation* o *Post-Route Simulation* según se requiera ver los archivos fuentes asociados a la implementación o los archivos fuentes asociados a la simulación.

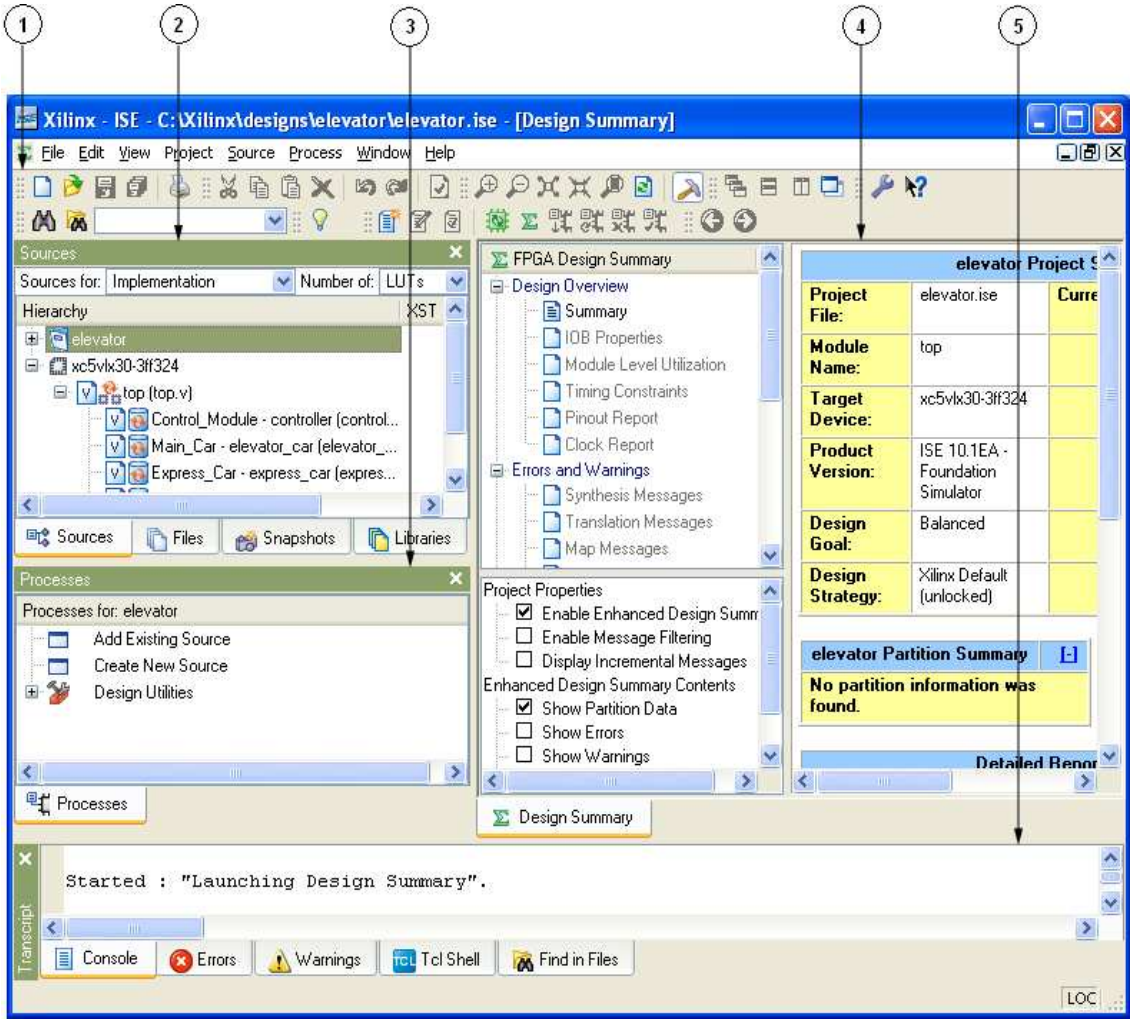


Figura 1- 40. Entorno de trabajo del paquete de software *ISE Foundation 10.1* de Xilinx

3. **Ventana de Procesos.** En esta sección del entorno de trabajo se puede ejecutar cada uno de los procesos que tiene lugar en el flujo de diseño que se detalla más adelante.

4. **Espacio de trabajo.** En esta sección el usuario puede abrir los archivos que se encuentran en la *Ventana de Fuentes* y editarlos, también aparecen los reportes generados al finalizar un proceso, así como ciertas ayudas como son las plantillas de lenguajes HDLs, entre otras.
5. **Ventana de Consola.** En esta sección del entorno de trabajo aparecen los mensajes de salida que generan los procesos que se están ejecutando. En esta sección es posible también ingresar comandos que realicen diversas acciones, como por ejemplo ejecutar procesos, operaciones del proyecto (como añadir fuentes), tareas de análisis de tiempos, entre otras.

1.9.1 FLUJO DE DISEÑO CON EL PAQUETE ISE FOUNDATION

Para realizar un proyecto de diseño e implementación de sistemas digitales basados en FPGAs y utilizando el paquete de software *ISE Foundation* de Xilinx, se recomienda seguir el flujo de diseño que se indica en la Figura 1-41. El flujo de diseño empieza con el *Diseño Lógico*, este consiste en describir mediante un lenguaje HDL nuestro circuito o sistema digital. La herramienta de software *ISE* también permite realizar esta descripción mediante esquemáticos, que después son traducidos a un lenguaje HDL previamente seleccionado.

Después de haber descrito el sistema digital, éste debe pasar por el proceso de *Síntesis*, el cual consiste en transformar el sistema diseñado desde un nivel de abstracción alto a un nivel de abstracción bajo, en el cual se especifican los componentes de hardware y sus interconexiones. El proceso de *Síntesis* lo realiza el software de manera automática, y en el caso del paquete de software *ISE Foundation* el resultado obtenido consiste en un *Netlist*, que es un archivo que contiene los elementos de hardware y sus interconexiones. El *Netlist* viene a ser la representación a nivel de compuertas del sistema descrito en el lenguaje HDL seleccionado. El archivo generado en el proceso de *Síntesis* tiene el formato NGC (*Native Generic Circuit*).

Una vez que se ha generado el archivo NGC comienzan los procesos de implementación, el primero de los cuales es la *Optimización*. En este proceso se minimizan los retardos de interconexión y retardos relacionados con los niveles lógicos. Después de este proceso se ejecuta el proceso de *Translate* en el cual el archivo de netlist original con formato NGC se convierte a un formato estándar llamado NGD (*Native Generic Database*). Luego los procesos de *Mapping, Placement* y *Routing* son ejecutados. El proceso de *Mapping* consiste en mapear el contenido del archivo NGD en los componentes reales que posee el FPGA seleccionado para el diseño. El proceso de *Placement* se encarga de posicionar los componentes mapeados en el proceso previo en lugares específicos del FPGA y el proceso de *Routing* se encarga de interconectarlos. Los procesos de *Placement* y *Routing* producen una carga computacional considerable por lo que su tiempo de finalización puede durar, dependiendo de la complejidad del diseño y las características del PC, hasta decenas de minutos [9].

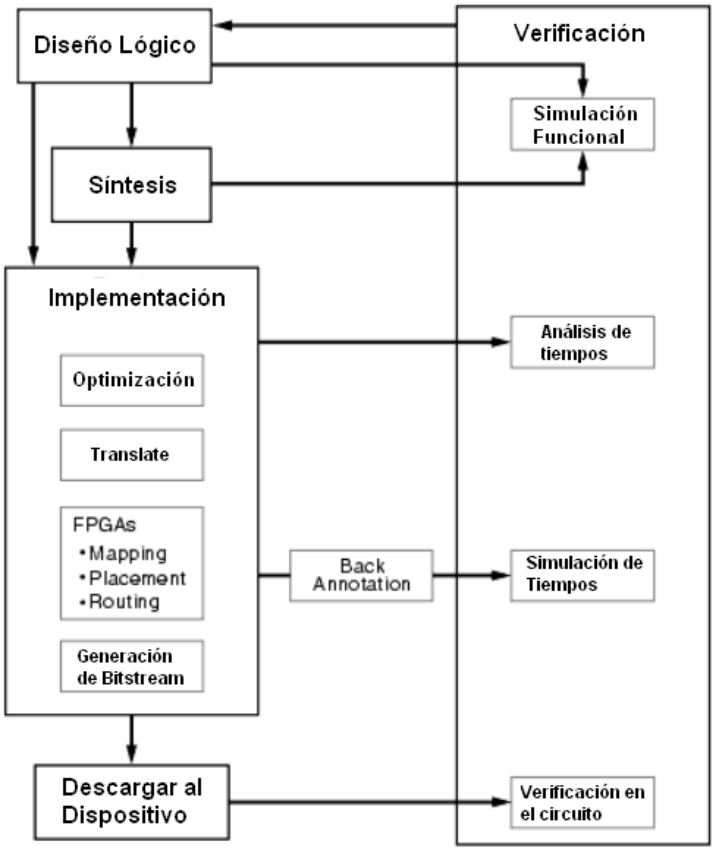


Figura 1- 41. Flujo de Diseño con FPGAs

Después de los procesos de *Placement* y *Routing* se obtendrá un archivo de formato NCD (*Native Circuit Description*), el cual contiene los componentes del FPGA localizados e interconectados.

La verificación puede darse en varias etapas del diseño. Durante los procesos de *Diseño Lógico* y *Síntesis,* la Simulación Funcional ayuda a verificar el comportamiento del sistema descrito y detectar errores en las etapas tempranas del diseño. Una vez que se han finalizado los procesos de *Placement* y *Routing*, es posible realizar un análisis de tiempos que determine los retardos presentados y las violaciones a las restricciones de tiempos (en el caso de que hayan sido usadas). También existe la posibilidad de simular el sistema diseñado incluyendo todos los factores de retardo existentes en la implementación física mediante el proceso de verificación denominado *Simulación de Tiempos*. En este proceso se puede verificar si el sistema bajo diseño funciona correctamente a la velocidad deseada en el dispositivo seleccionado. Como lo indica la Figura 1-41, antes de poder hacer una *Simulación de Tiempos* se requiere de un proceso denominado *Back Annotation*, el cual convierte la información del diseño físico de nuevo en un diseño lógico, con el cual se puede realizar la simulación.

Después de haber realizado los procedimientos de *Placement* y *Routing*, y hacer las verificaciones previas, se genera un *bitstream* basándose en el archivo NCD generado al finalizar los procedimientos antes mencionados. El *bitstream* es el archivo de configuración que contiene la trama de bits que produce la configuración del FPGA.

En el Anexo 5 se presenta un ejemplo de diseño con la herramienta de desarrollo *ISE Foundation 10.1*, en el cual se incluye los procedimientos a seguir para ejecutar cada uno de los procesos descritos en el flujo de diseño.

1.10 EL ESTÁNDAR JPEG

Oficialmente el estándar JPEG está definido en las recomendaciones ISO/IEC IS 10918-1 de la Organización Internacional para la Estandarización (ISO) y en la ITU-T T.81 de la Unión Internacional de Telecomunicaciones (ITU); el estándar establece un conjunto de técnicas de compresión de imágenes mediante algunos procedimientos de codificación y métodos de transformación. El objetivo principal que persigue el estándar es obtener codificadores y decodificadores con tasas de compresión y calidades de imagen explotando al máximo la tecnología actual disponible, como también obtener algoritmos de compresión de complejidad computacional manejable, con el fin de obtener implementaciones prácticas de éstos [13][14].

El estándar JPEG soporta imágenes monocromáticas, imágenes a color, o imágenes multi-componente, independientemente del tamaño o el espacio de color que utilicen, y puede alcanzar factores de compresión de 10:1 hasta 20:1 [15].

El estándar JPEG define cuatro modos de compresión:

1. *Modo de Codificación secuencial.* En este modo cada componente de la imagen es codificada en una sola exploración siguiendo un orden de izquierda a derecha y de arriba abajo. El modo secuencial soporta los métodos de codificación de entropía de Huffman y Aritmética.

2. *Modo de Codificación Progresivo*. La imagen en el modo Progresivo es codificada en múltiple exploraciones, pudiendo ser de 2 a 896. La imagen es también decodificada en múltiples exploraciones con cada una de las cuales se obtendrá una imagen de mejor calidad. El modo progresivo soporta la codificación de entropía de Huffman y Aritmética.

3. *Modo de codificación sin pérdidas*. El modo de codificación sin pérdidas preserva exactamente la imagen original. Este modo se basa en la codificación diferencial y en predictores, y puede alcanzar tasas de compresión bajas en el orden de 2:1.

4. *Modo de codificación jerárquico*. En este modo la imagen es codificada en un número de subimágenes de diferente resolución denominadas tramas. En el decodificador, la primera trama que se ensambla es la de menor resolución, y a partir de ésta, se pueden ir decodificando los datos hasta obtener la resolución de imagen deseada.

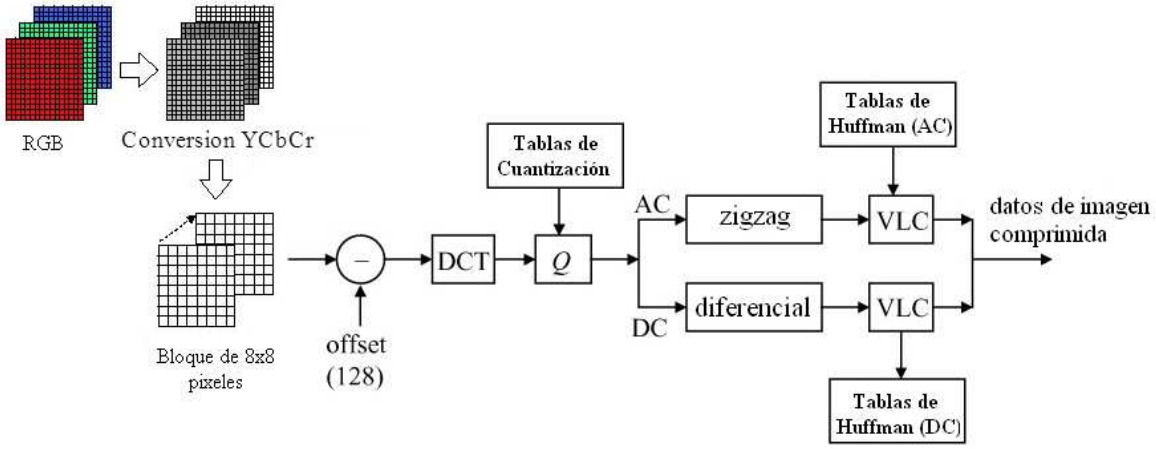
Los dos primeros modos de codificación (secuencial y progresivo), pueden soportar imágenes con 8 y 12 bits de precisión.

1.10.1 ESQUEMA DE CODIFICACIÓN SECUENCIAL BASELINE

El modo de codificación secuencial con pérdidas más ampliamente difundido se denomina *Baseline*, el cual se caracteriza por los siguientes parámetros:

- Puede comprimir imágenes con 1 a 4 componentes y una precisión de 8 bits por componente. Cada componente es codificado de manera independiente.
- La imagen a comprimir puede estar en el espacio de color RGB o YCbCr, aunque se obtienen mejores resultados de compresión en el espacio YCbCr.
- Solamente admite codificación de entropía de Huffman.
- Se basa en la transformada discreta del coseno.

El diagrama de bloques que se indica en la Figura 1-42 representa el procedimiento básico para comprimir una imagen con el esquema Baseline JPEG.



Fuente [16]

Figura 1- 42. Esquema de compresión Baseline del estándar JPEG

Como se puede ver en el esquema anterior, el primer paso que se sigue en el esquema Baseline es la transformación del espacio de color RGB al YCbCr, esto se logra mediante las siguientes ecuaciones:

$$Y = 0.299R + 0.587G + 0.114B$$
$$Cb = -0.1687R - 0.3313G + 0.5B + 2^{\text{Precisión muestreo}/2}$$
$$Cr = 0.5R - 0.4187G - 0.0813B + 2^{\text{Precisión muestreo}/2}$$

Al tener la imagen en YCbCr, ésta debe dividirse en bloques (subimágenes) de 8x8 pixeles no superpuestos (*non-overlapping*). Estos bloques son la entrada a la DCT-2D.

1.10.1.1 La transformada discreta coseno a dos dimensiones

La Transformada discreta coseno (DCT-2D) permite transformar datos de una representación espacial a una representación frecuencial. Esta transformada se utiliza en muchos de los estándares de compresión de imágenes y video, entre otras cosas por las siguientes características [14]:

1. *Capacidad de compactación de la energía en el dominio transformado*. La DCT-2D permite concentrar la mayor parte de la información en pocos

coeficientes, por lo que bastará codificar con mayor precisión éstos para obtener una buena representación de todo el bloque de la imagen. Cabe mencionar que esta característica es un parámetro estadístico por lo que en casos muy raros la energía puede estar dispersa en todos los coeficientes.

2. *El algoritmo es independiente de los datos de la imagen.*
3. *Errores reducidos en los contornos de los bloques*. La DCT-2D permite obtener errores de codificación pequeños en los límites de la subimagen evitando así el efecto de bloques en la imagen reconstruida.
4. *Identificación frecuencial de los coeficientes transformados.* Los coeficientes de la DCT-2D pueden interpretarse desde un punto de vista frecuencial, lo que permite obtener mejores esquemas de codificación basados en conceptos psico-visuales.

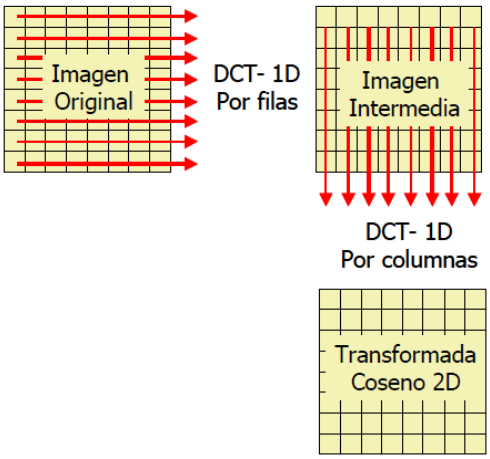
La definición matemática de la DCT-2D para un bloque de 8x8 está dada por:

$$F(u,v) = \frac{1}{4}C(u)C(v)\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)Cos\left[\frac{\pi(2x+1)u}{16}\right]Cos\left[\frac{\pi(2y+1)u}{16}\right]$$

Para u=0,1,…,7 y v=0,1,...,7, donde

$$C(k) = \begin{cases} \frac{1}{\sqrt{2}} & para\ k = 0 \\ 1 & para\ el\ resto \end{cases}$$

A pesar de que la ecuación que define la transformada discreta coseno parecería difícil de implementar en aplicaciones prácticas, existen algoritmos de optimización que permiten desarrollar la DCT-2D, reduciendo el número de operaciones a ejecutar. Uno de estos algoritmos se basa en la transformada discreta coseno unidimensional y consiste en aplicar la transformada unidimensional a las filas del bloque de la imagen y, posteriormente, sobre el resultado obtenido, volver a aplicar la transformada unidimensional a las columnas, tal como lo ilustra la Figura 1-43.



Fuente [14]

Figura 1- 43. Obtención de la DCT-2D en base a la DCT-1D

Este algoritmo puede representarse de manera matricial como se muestra a continuación:

$$Y = CXC^T$$

Donde X representa el bloque de 8x8 pixeles, Y los coeficientes DCT-2D, y C está dada por

$$C = \begin{bmatrix} 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 \\ 0.4904 & 0.4157 & 0.2778 & 0.0975 & -0.0975 & -0.2778 & -0.4157 & -0.4904 \\ 0.4619 & 0.1913 & -0.1913 & -0.4619 & -0.4619 & -0.1913 & 0.1913 & 0.4619 \\ 0.4157 & -0.0975 & -0.4904 & -0.2778 & 0.2778 & 0.4904 & 0.0975 & -0.4157 \\ 0.3536 & -0.3536 & -0.3536 & 0.3536 & 0.3536 & -0.3536 & -0.3536 & 0.3536 \\ 0.2778 & -0.4904 & 0.0975 & 0.4157 & -0.4157 & -0.0975 & 0.4904 & -0.2778 \\ 0.1913 & -0.4619 & 0.4619 & -0.1913 & -0.1913 & 0.4619 & -0.4619 & 0.1913 \\ 0.0975 & -0.2778 & 0.4157 & -0.4904 & 0.4904 & -0.4157 & 0.2778 & -0.0975 \end{bmatrix}$$

Este algoritmo será el que se sintetice a nivel de hardware en el siguiente capítulo.

La DCT-2D permitirá obtener una matriz de 8x8 con los 64 coeficientes transformados del bloque de 8x8 pixeles. El primer coeficiente se denomina DC y corresponde al promedio de los restantes 63, denominados coeficientes AC.

1.10.1.2 Cuantización

La cuantización consiste en dividir cada coeficiente DCT por una constante que se encuentra en una tabla denominada *Tabla de Cuantización.* La Tabla de Cuantización, en general, define la importancia que tiene cada coeficiente transformado dentro del bloque.

El estándar JPEG sugiere dos Tablas de Cuantización, una para luminancia y otra para crominancia (Cb y Cr), las cuales se presentan en la Figura 1-44. La relación de los coeficientes DCT y los elementos de la tabla de cuantización está definida por la siguiente expresión:

$$F^q(u,v) = \frac{F(u,v)}{Q(u,v)}$$

Matriz de cuantificación de luminancia

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Matrices de cuantificación de crominancia

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Fuente [14]

Figura 1- 44. Tablas de Cuantización recomendadas en el estándar JPEG

Como se mencionó anteriormente, los coeficientes de la DCT-2D pueden ser interpretados desde un punto de vista frecuencial, siendo así que en la esquina superior izquierda se encuentran los coeficientes de baja frecuencia que concentran la mayor energía. Ya que el sistema visual humano es susceptible a detectar pérdidas en las componentes de baja frecuencia espacial de una imagen, es necesario que estos coeficientes tengan un menor grado de cuantización que los coeficientes que se alejan del origen de la matriz de coeficientes. Tomando

esto en cuenta, se puede realizar una cuantización mucho mayor a los coeficientes de alta frecuencia que se encuentran en la región inferior derecha de la matriz de coeficientes; esto provocará que muchos de los coeficientes AC formen cadenas de ceros, lo cual es un factor de redundancia que puede ser aprovechado en el esquema de codificación de la entropía.

1.10.1.2.1 Factor de Calidad

El Factor de Calidad de una imagen comprimida en JPEG está ligado directamente con los niveles de cuantización establecidos en la tabla de cuantización. Se puede variar la calidad de una imagen modificando los niveles de cuantización de las tablas de cuantización; el método establecido para obtener tablas de cuantización (y por ende diferentes factores de calidad) consiste en definir un factor de calidad de 50% para las tablas de cuantización recomendadas en el estándar y, en base a éstas, obtener nuevas tablas, escalando las tablas originales por un factor que está en función del factor de calidad deseado [16], como se indica a continuación:

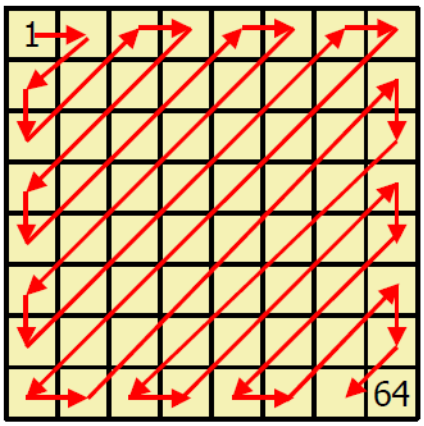
$$\alpha = \frac{50}{factor\_calidad} \qquad Si \quad 1 \leq factor\_calidad \leq 50$$

$$\alpha = 2 - \frac{2 \times factor\_calidad}{100} \qquad Si \quad 50 < factor\_calidad \leq 99$$

$$Q_{factor\_calidad} = \alpha \times Q_{JPEG}$$

1.10.1.3 Exploración en zig-zag

La exploración en zig-zag permite maximizar el número de ceros consecutivos que se obtienen después de realizar el proceso de cuantización. Al disponer los coeficientes en el orden que se indica en la Figura 1-45, los coeficientes de baja frecuencia son los primeros en ser codificados.



Fuente [14]

Figura 1- 45. Exploración en zig-zag

1.10.1.4 Codificación de entropía

Para la codificación de los coeficientes cuantizados se utiliza una variación de los códigos de Huffman, y se codifican por separado los coeficientes AC y DC.

Para los coeficientes AC (sean de luminancia o crominancia) ordenados en zig-zag, cada coeficiente no nulo se codifica mediante dos palabras de longitud variable, una denominada A y la otra B. La palabra A está definida en función de la categoría (C) a la que pertenezca el coeficiente y la longitud de series de ceros (R) que antecede a dicho coeficiente (relación R/C). La categoría del coeficiente se establece según el valor que posea éste, y para los coeficientes AC se establecen 10 categorías las cuales se muestran en la Tabla 1-7. La categoría define la cantidad de bits con los que se puede representar el valor del coeficiente no nulo; debido a que los coeficientes AC, que se obtienen de la DCT-2D, están en el rango de -1023 a 1023, se tiene que la máxima categoría para los coeficientes AC es 10.

Con la longitud de series de ceros y la categoría del coeficiente establecidas, la palabra código A puede ser consultada en la tabla de códigos que se propone en el estándar JPEG, las cuales se presentan en el Anexo 6 de este proyecto. Estas tablas establecen la palabra código A y su longitud en bits. Existen dos tablas de

códigos para la palabra A, una para la componente de luminancia y otra para las componentes de crominancia (Cb y Cr).

Tabla 1- 7. Categorías para los coeficientes AC

Categoría	Rango de valores de coeficientes
1	-1, 1
2	-3 a -2, 2 a 3
3	-7 a -4, 4 a 7
4	-15 a -8, 8 a 15
5	-31 a -16, 16 a 31
6	-63 a -32, 32 a 63
7	-127 a -64, 64 a 127
8	-255 a -128, 128 a 255
9	-511 a -256, 256 a 511
10	-1023 a -512, 512 a 1023

Por ejemplo, si se tiene la secuencia de coeficientes para un bloque de 8x8 de la componente de luminancia, **12 -3 0 0 0 0 5** (ceros), se consideran las siguientes observaciones para determinar la palabra código A de los coeficientes AC no nulos:

- El coeficiente en rojo es el coeficiente DC del bloque de 8x8, y no se codifica de la forma antes mencionada.
- El primer coeficiente AC no nulo, -3, pertenece a la categoría 2 según la Tabla 1-7, no tienen ninguna serie de ceros que le preceda por tanto el valor de la relación R/C es 0/2, y para esta relación la palabra código A es “01”.
- El siguiente coeficiente AC no nulo, 5, pertenece a la categoría 3 según la Tabla 1-7, a este coeficiente le preceden cuatro ceros, por lo que la relación R/C es 4/3, y para esta relación, según el Anexo 6, la palabra código A es “1111111110010110”.

Para la palabra código A existen algunas consideraciones especiales con respecto a la longitud de series de ceros que puede preceder a un coeficiente no nulo, con el objeto de optimizar los códigos de longitud variable utilizados para la codificación. En el estándar JPEG se establece que la longitud máxima antes de un coeficiente no nulo debe ser 15 ceros consecutivos. Cuando existe una cadena

de 16 ceros se añade una palabra código especial para indicar este hecho denominada ZRL (*zero-run-length*); pueden concatenarse varias de estas palabras hasta obtener una cadena de ceros menor a 16 para poder codificar la palabra A mediante las relaciones R/C que se especifican en el estándar.

Existe una palabra código A especial para indicar el fin de un bloque de 8x8 o en su defecto que el resto de coeficientes son ceros, y se denomina EOB (End of Block). En el caso especial en que el último coeficiente (el coeficiente 64) no sea nulo, no se codifica la palabra código EOB sino la que corresponda para ese coeficiente no nulo [17].

La palabra código B refleja directamente el valor del coeficiente cuantizado no nulo, se expresa en binario en *complemento a uno* y su longitud está definida por la categoría a la que pertenezca dicho coeficiente. Por ejemplo, para la secuencia de coeficientes **12 -3 0 0 0 0 5** (ceros) la palabra código B para sus coeficientes AC no nulos sería:

- Para el primer coeficiente AC no nulo, -3, su *complemento a uno* es “00”, y su longitud en bits es dos ya que este coeficiente pertenece a la categoría 2, como lo muestra la Tabla 1-7.
- Para el siguiente coeficiente AC no nulo, 5, es el mismo coeficiente expresado en binario (ya que es positivo), “101”, y su longitud en bits es tres ya que este coeficiente pertenece a la categoría 3, como se puede observar en la tabla de categorías.

La codificación de los coeficientes DC es algo parecido a la codificación en AC. De la misma manera se definen dos palabras código de longitud variable denominadas A’ y B’. La codificación no se realiza directamente sobre el coeficiente DC sino sobre un coeficiente diferencial que se define como la diferencia entre el coeficiente DC del bloque actual y el coeficiente DC del bloque precedente, el cual puede estar en el rango de -2047 a 2047 y puede ser cero.

La definición de la palabra A' es más sencilla que su similar para los coeficientes AC. Simplemente se basa en la categoría a la que pertenezca el coeficiente DC diferencial, la cual tiene el mismo significado que en el caso de los coeficientes AC. En la Tabla 1-8 se presenta la definición de las categorías de los coeficientes DC diferenciales. Establecida la categoría del coeficiente DC diferencial se puede consultar la palabra código A' que corresponda a dicha categoría en las tablas que en el estándar se recomienda y que se muestran en el Anexo 6. Al igual que en el caso AC, existen dos tablas de códigos, una para cada componente, que indican la palabra código del coeficiente y su longitud en bits.

Tabla 1- 8. Categorías para los coeficientes DC diferenciales

Categoría	Rango de valores de coeficientes
0	0
1	-1, 1
2	-3 a -2, 2 a 3
3	-7 a -4, 4 a 7
4	-15 a -8, 8 a 15
5	-31 a -16, 16 a 31
6	-63 a -32, 32 a 63
7	-127 a -64, 64 a 127
8	-255 a -128, 128 a 255
9	-511 a -256, 256 a 511
10	-1023 a -512, 512 a 1023
11	-2047 a -1024, 1024 a 2047

Cuando se inicia la codificación de los coeficientes DC, se considera que el coeficiente DC que precede al coeficiente DC del primer bloque de 8x8 de la imagen es cero. Por ejemplo, si se considera a la secuencia **12 -3 0 0 0 0 5** (ceros) como los coeficientes ordenados en zig-zag del primer bloque de 8x8, el coeficiente DC diferencial sería DC_{diff} = 12 – 0 = 12, y este coeficiente diferencial pertenece a la categoría 4 según la Tabla 1-8, para la cual la palabra código A' es "101" según las tablas definidas en el estándar (Anexo 6) para la componente de luminancia.

La palabra B' se obtiene de la misma forma que con los coeficientes AC, es decir es igual al *complemento a uno* del valor del coeficiente diferencial y su longitud es igual a la categoría a la que pertenezca dicho coeficiente. Por ejemplo, para el

coeficiente diferencial 12 su palabra código B’ es “1100” y su longitud es de 4 bits ya que el coeficiente diferencial pertenece a la categoría 4.

De esta manera, la codificación de la secuencia de ejemplo **12 -3 0 0 0 0 5** (ceros) correspondiente a un bloque de 8x8 de la componente de luminancia resultaría como muestra en la siguiente tabla:

Tabla 1- 9. Resultados de la compresión de un bloque de 8x8 pixeles

DC		**AC**		**AC**		**AC**
12		**-3**		**5**		**EOB**
C=4		**R/C=**0/2		**R/C=**4/3		**---**
A’	**B’**	**A**	**B**	**A**	**B**	**A**
101	1100	01	00	1111111110010110	101	1010

De esta manera se ha reducido la cantidad de bits necesarios para representar un bloque de 8x8 pixeles, de 512 a tan solo 34 bits.

1.11 FORMATO DE INTERCAMBIOS DE ARCHIVOS JFIF

Inicialmente, el estándar JPEG no recomendaba ningún formato para archivos de imagen comprimidos con este estándar. Esto produjo que se diseñara un formato de intercambio de imágenes JPEG, fuera del estándar, denominado JFIF (*JPEG File Interchange Format*). A pesar que tiempo después de establecido el estándar, el grupo de trabajo de JPEG definió el formato de archivo SPIFF (*Still Picture Interchange File Format*), éste no logró sustituir al formato JFIF que para ese entonces se encontraba muy difundido en los sistemas informáticos.

El formato de archivos JFIF se base en marcadores, los cuales dividen todo el *stream* del archivo en bloques de datos que ayudan al decodificador a obtener la información necesaria para la decodificación de la imagen JPEG. Cada marcador consta de dos bytes, siempre el primer byte empieza con el valor hexadecimal xFF, y el segundo byte especifica el contenido del marcador. El valor hexadecimal xFF puede ser utilizado como byte de relleno entre los marcadores del archivo, ya

que el decodificador al encontrar un byte xFF, seguido por otro byte xFF, lo ignora.

El orden en que pueden establecerse los marcadores dentro del archivo no es estricto y sólo debe cumplir con dos reglas:

1. El archivo siempre debe iniciar con un marcador de Inicio de Imagen (**SOI**[9]) y terminar con un marcador de Fin de Imagen (**EOI**[10]).
2. Si la información de un marcador es usada por un segundo marcador, el primer marcador debe aparecer antes que el segundo.

Los marcadores pueden ser de dos tipos: autónomos y no autónomos. Los marcadores autónomos son definidos sólo con los dos bytes que especifican el marcador. Los marcadores no autónomos son acompañados de datos para la decodificación de la imagen, por lo que también consta, además de los dos bytes del marcador, de dos bytes de longitud que definen el número de bytes de datos que acompañan al marcador. La longitud del contenido del marcador toma en cuenta los datos del marcador y los dos bytes de longitud, pero no los dos bytes que especifican el marcador.

En el Anexo 7 se presentan todos los marcadores utilizados en el formato de archivo JFIF. A continuación se describen los marcadores necesarios para generar un archivo de imagen con el estilo de codificación Secuencial *Baseline* del estándar JPEG, que es el objeto de estudio en este proyecto.

Marcador SOI (Inicio de Imagen)

Este marcador indica el inicio del archivo con formato JFIF, siempre aparece al inicio del archivo y sólo puede existir un marcador SOI por archivo.

[9] **SOI: Start Of Image**
[10] **EOI: End Of Image**

Marcador EOI (Fin de Imagen)

Este marcador indica el término del archivo JFIF, siempre aparece al final del archivo y sólo puede existir un marcador EOI por archivo.

Marcador APP_n

Los marcadores APP_0-APP_{15} son utilizados para incluir en el archivo datos específicos de la aplicación que realiza el procesamiento de la imagen, además de los especificados en el estándar. Los marcadores APP_n pueden aparecer en cualquier parte del archivo; para el formato JFIF se establece el marcador APP_0, el cual siempre debe ir después de un marcador de inicio de imagen (SOI), y posee los campos de datos que se indican en la Tabla 1-10.

Tabla 1- 10. Campos de datos del marcador APP_0

Nombre del campo	Tamaño del campo	Descripción
Identificador	5 bytes	Cadena de caracteres *JFIF* terminados con el valor 00_{16}.
Identificador principal de versión	1 byte	Primer número identificador de la versión. Para la versión 1.1 este valor es 1.
Identificador secundario de versión	1 byte	Segundo número identificador de la versión. Para la versión 1.1 este valor es 1.
Unidades	1 byte	Unidades para la densidad de pixel. Una valor de 0 significa que ninguna densidad es usada. Un valor de 1 significa que los dos siguientes campos son expresados en pixeles por pulgada, un valor de 2 especifica pixeles por cm.
Xdensity	2 bytes	Densidad de pixel horizontal.
Ydensity	2 bytes	Densidad de pixel vertical.
Xthumbnail	1 byte	Ancho de la imagen en miniatura (opcional). Este valor puede ser 0.
Ythumbnail	1 byte	Alto de la imagen en miniatura (opcional). Este valor puede ser 0.
Thumbnail	variable	Una imagen miniatura (opcional). El tamaño de este campo es 3 x *Xthumbnail* x *Ythumbnail*. La imagen es almacenada con 3 bytes por pixel en el espacio de color RGB.

Marcador DHT[11] (Definición de Tablas de Huffman)

Este marcador define (o redefine) las Tablas de Huffman, las cuales son identificadas por una clase (AC y DC) y un número. Para el modo de codificación Baseline están definidas dos tablas de cada tipo (para luminancia y crominancia). El marcador DHT es un marcador no autónomo y presenta un encabezado con los campos de datos que se indican en la Tabla 1-11.

Tabla 1- 11. Campos de datos del marcador DHT

Tamaño del Campo	Descripción
1 byte	Los 4 bits más significativos definen la clase de tabla. Un valor de 0 significa una tabla para coeficientes DC, un valor de 1 significa una tabla para coeficientes AC. Los 4 bits menos significantes definen el identificador de la tabla. Para Baseline se definen los identificadores 0 (para luminancia) y 1 (para crominancia).
16 bytes	Enumeración de códigos de Huffman según su longitud. Cada enumeración se almacena como un entero sin signo de 1 byte.
Variable	Símbolos abreviados de los códigos de Huffman

Los símbolos abreviados de los códigos de Huffman son de 1 byte de longitud. Para los coeficientes diferenciales DC, este byte representa la magnitud de la diferencia de coeficientes DC, indicando la longitud en bits de su magnitud. Para los coeficientes AC los cuatro bits más significativos indican la longitud de series de ceros que precede al coeficiente AC no nulo, y los cuatro bits menos significativos, la longitud en bits de la magnitud de éste. En el Anexo K del estándar JPEG, se indican los valores para la enumeración de los códigos de Huffman según su longitud y los símbolos abreviados de éstos (que serán usados en el siguiente capítulo), para las tablas recomendadas en el estándar JPEG.

Marcador DQT[12] (Definición de Tablas de Cuantización)

Define (o redefine) las Tablas de Cuantización usadas en la imagen. Se pueden definir hasta cuatro tablas de cuantización. El marcador DQT no es autónomo y presenta los datos que se muestran en la Tabla 1-12.

[11] **DHT: Define Huffman Table**
[12] **DQT: Define Quantization Table**

Tabla 1- 12. Campos de datos del marcador DQT

Tamaño del campo	Descripción
1 byte	Los 4 bits menos significativos son los identificadores de la tabla (0, 1, 2 o 3). Los 4 bits más significativos especifican el tamaño de los valores de la Tabla de Cuantización (0 = 1 byte, 1 = 2 bytes)
64 o 128 bytes	Valores sin signo de 1 o 2 bytes de la Tabla de Cuantización.

Cuando un valor de la Tabla de Cuantización es mayor a 255 se hace necesario definir una Tabla de Cuantización con valores de 2 bytes. El orden en que se definen los valores de la Tabla de Cuantización en el marcador DQT es el mismo orden de la exploración en zig-zag. El valor del campo de longitud para este marcador será igual al tamaño de las tablas (64 o 128) más dos.

Marcador SOF_n[13] (Inicio de Trama)

Este marcador define una trama (de imagen). El marcador SOF_n consiste de una porción de datos fija, seguida por una estructura para definir cada componente de color usada en la trama. Los campos de datos que acompañan a este marcador se describen en la Tabla 1-13.

Tabla 1- 13. Campos de datos de marcador SOF

Porción Fija	
Tamaño del Campo	**Descripción**
1 byte	Precisión de muestreo en bits (puede ser 8 o 12)
2 bytes	Alto de la imagen en pixels
2 bytes	Ancho de la imagen en pixels
1 byte	Número de componentes de la imagen
Especificación de componentes (para cada componente)	
Tamaño del Campo	**Descripción**
1 byte	Identificador de componente. Para JFIF los identificadores son 1 (Y), 2(Cb) y 3 (Cr).
1 byte	Los 4 bits más significativos especifican la frecuencia de muestreo horizontal de la componente y los 4 bits menos significativos especifican la frecuencia de muestreo vertical. Valores permitidos son 1, 2, 3 o 4.
1 byte	Identifica la tabla de cuantización para la componente. Los valores corresponden a los de identificación del marcador DQT.

Sólo puede existir un marcador SOF en el archivo JFIF.

[13] **SOF: Start Of Frame**

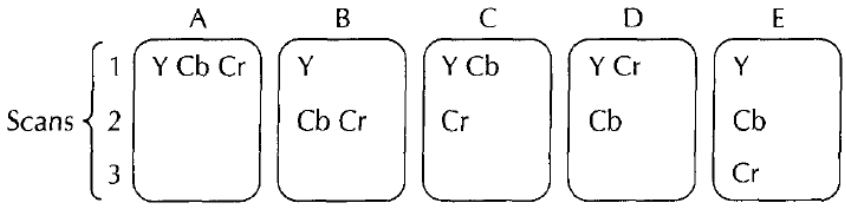
Marcador SOS[14] (Inicio de exploración)

Este marcador indica el inicio de los datos comprimidos de una exploración. Su estructura de datos se divide como se indica en la Tabla 1-14.

Tabla 1- 14. Campos de datos del marcador SOS

Estructura del marcador SOS	
Tamaño del campo	**Descripción**
1 byte	Contador de componente
2 bytes	Descriptor de componente
1 byte	Inicio de la selección espectral (0-63)
1 byte	Fin de la selección espectral (0-63)
1 byte	Aproximación sucesiva (dos campos de 4 bits, cada uno con un valor en el rango de 0-13)
Descriptor de Componente	
Tamaño de campo	**Descripción**
1 byte	Identificador de la componente
1 byte	Los 4 bits más significativos especifican la Tabla de Huffman para los coeficientes diferenciales DC y los 4 bits menos significativos especifican la Tabla de Huffman para los coeficientes AC

Existe un descriptor por cada componente y debe aparecer en el orden en el cual estén las componentes dentro del MCU, a su vez este orden depende del tipo de exploración que se haga [18], como lo ilustra la Figura 1-46. Es posible que no todas las componentes definidas en el marcador SOF estén en el marcador SOS, pero es necesario que tengan el mismo orden en ambos marcadores. También es requisito que el identificador de componente en el descriptor sea el mismo que el que se definió en el marcador SOF, como también que los identificadores de tablas de Huffman sean los mismos que los del marcador DHT.



Fuente [18]

Figura 1- 46. Tipos de Exploración que admite JFIF

[14] **SOS: Start Of Scan**

En una exploración secuencial el inicio de la selección espectral[15] debe ser cero; el término de la selección espectral, 63, y la aproximación sucesiva[16], cero.

El marcador SOS debe aparecer siempre después de un marcador SOF_n, y le deben preceder los marcadores DQT y DHT.

[15] **Selección Espectral:** Hace referencia a un método de dividir las componentes de una imagen en bandas de coeficientes DCT dentro de la MCU. Se utiliza sobre todo en la codificación progresiva JPEG.
[16] **Aproximación Sucesiva:** Es otro método de división de componentes de imagen utilizado en la codificación progresiva JPEG.

REFERENCIAS

[1] Woods, Roger; McAllister, John; Lightbody, Gaye; Yi, Ying. *FPGA-based Implementation of Signal Processing Systems*. 1 ed.. Editorial John Wiley & Sons Ltd, 2008.

[2] Intel Corporation, 40 aniversario de la Ley de Moore.
Obtenido de:
http://www.intel.com/cd/corporate/techtrends/EMEA/spa/209840.htm
(Último acceso: 12/09/2009)

[3] Maxfield, Clive. *The Design Warrior's Guide to FPGA*. 1 ed.. Burlington: Editorial Newnes, 2004.

[4] Alpago, Octavio y Sagreras, Miguel. *Lógica Programable: FPGA*. Universidad de Buenos Aires, 2007.
Obtenido de:
http://cactus.fi.uba.ar/6633/material/FPGA_Tutorial/FPGA.pdf
(Último acceso: 16/09/2009)

[5] Güichal, Guillermo. *Diseño Digital Utilizando Lógicas Programables*. Universidad Tecnológica Nacional, Facultad Regional Bahía Blanca, 2005.
Obtenido de:
http://fpga.com.ar/notas/NotasCompletas.htm
(Último acceso: 16/09/2009)

[6] Xilinx, Inc. *Virtex-II Platform FPGAs: Complete Data Sheet,* Noviembre 2007, versión 3.5.
Obtenido de:
http://www.xilinx.com/support/documentation/data_sheets/ds031.pdf
(Último acceso: 20/09/2009)

[7] Xilinx, Inc. *Virtex-II Pro and Virtex-II Pro X Platform FPGAs:Complete Data Sheet,* Noviembre 2007, versión 4.7.
Obtenido de:
http://www.xilinx.com/support/documentation/data_sheets/ds083.pdf
(Último acceso: 20/09/2009)

[8] Xilinx, Inc. *Spartan-3 Generation FPGA User Guide*, Enero 2009, versión 1.5.
Obtenido de:
http://www.xilinx.com/support/documentation/user_guides/ug331.pdf
(Último acceso: 25/09/2009)

[9] Alarcón, Claudio. *Arquitectura de la FPGA Spartan III de Xilinx*. Universidad de Chile, Marzo 2006.
Obtenido de:
https://www.u-cursos.cl/ingenieria/2006/1/EL693/1/material_docente/objeto/86389
(Último acceso: 29/09/2009)

[10] Xilinx, Inc. *Spartan-3A FPGA Family: Data Sheet*. Julio 2007, versión 1.4.1.
Obtenido de:
http://www.xilinx.com/support/documentation/data_sheets/ds529.pdf
(Último acceso: 05/10/2009)

[11] Xilinx, Inc. *Spartan-3 FPGA Family: Data Sheet*. Diciembre 2009, versión 1.4.1.
Obtenido de:
http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf
(Último acceso: 09/01/2010)

[12] Xilinx, Inc. *Spartan-3A FPGA Starter Kit Board User Guide*. Junio 2007, version 1.3.

Obtenido de:
http://www.xilinx.com/support/documentation/boards_and_kits/ug330.pdf
(Último acceso: 11/01/2010)

[13] Acharya, Tinku; Tsai, Ping-Sing. *JPEG2000, Standard for Image Compression: concepts, algorithms and VLSI architecture*. 1 ed.. Hoboken: Editorial John Wiley & Sons Ltd, 2005.

[14] Tarrés, Francesc. *Sistemas Audiovisuales: Televisión analógica y digital.*. 1 ed.. Barcelona: Editorial UPC, 2000.

[15] Bernal, Iván. *Compresión de Imágenes con JPEG*. Quito: Escuela Politécnica Nacional, Septiembre 2004.

[16] Ghanbari, Mohammed. *Standard Codecs: Image Compression to Advanced Video Coding*. 2 ed. Londres: The Institution of Electrical Engineers, 2003.

[17] Pennebaker, William; Mitchell, Joan. *JPEG still image data compression standard*. 3 ed.. Editorial Van Nostrand Reinhold, 1993.

[18] Miano, John. *Compressed Image File Formats: JPEG, PNG, GIF, XBM, BMP*. 1 ed.. Reading, Massachusetts: Editorial ACM Press, 1999.

CAPÍTULO 2

DISEÑO DEL SISTEMA DE ADQUISICIÓN, COMPRESIÓN Y ALMACENAMIENTO DE IMÁGENES

2.1 GENERALIDADES

Este capítulo se centra en el diseño de un Sistema de Adquisición, Compresión y Almacenamiento de Imágenes, utilizando un FPGA de la familia Spartan-3A de la plataforma Spartan 3 y el entorno de programación Visual GUIDE de Matlab, tal como se indica en el diagrama de bloques que se muestra en la Figura 2-1. La parte medular del sistema, que es la implementación del estilo de codificación secuencial *Baseline* del estándar JPEG, es diseñada e implementada sobre el dispositivo FPGA XC3S700A de la familia Spartan-3A cuyas características y arquitectura se describieron en el Capítulo 1. Esta implementación destaca y demuestra el potencial que poseen los FPGAs en el desarrollo de aplicaciones complejas que requieren gran requerimiento de cómputo y recursos lógicos.

Algunas funciones de los módulos del sistema, representados en el diagrama de bloques de la Figura 2-1, son desarrolladas como rutinas dentro de un programa en lenguaje Matlab. Esto se ha realizado con el fin de solventar algunas limitaciones, en lo que respecta a capacidad de recursos lógicos (CLBs y bloques embebidos), que se encontrarían si dichas funciones fueran realizadas en hardware, así como la realización de funciones que no pueden hacerse netamente sobre hardware (como la dotación de extensiones a archivos). Por este motivo, estos módulos poseen una sección de hardware (implementación sobre el FPGA) y una sección de software (rutinas en lenguaje de Matlab).

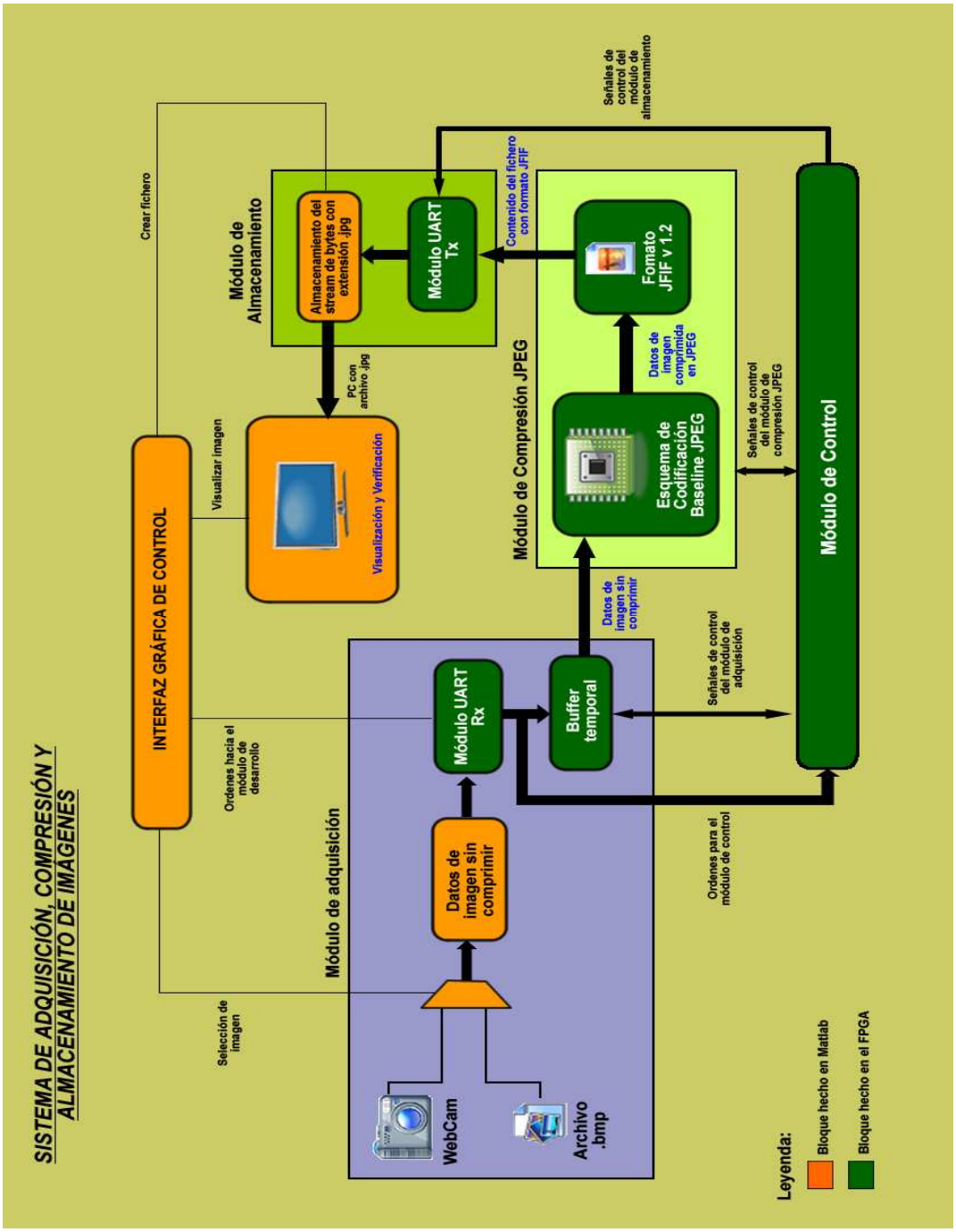


Figura 2- 1. Diagrama de Bloques del Sistema de Adquisición, Compresión y Almacenamiento de Imágenes

Como se observa en la Figura 2-1, los módulos del sistema son:

- *Módulo de Adquisición*. Consta de una sección de hardware y una sección de software. Este módulo envía los datos de una imagen monocromática sin comprimir, previamente adquirida mediante funciones de Matlab, hacia

el módulo de desarrollo *Spartan 3A Starter Kit*, donde son procesados para obtener los datos comprimidos de la imagen.

- *Módulo de Compresión*. En este módulo se sintetiza a nivel de hardware el esquema de codificación *Baseline* del estándar JPEG descrito en el Capítulo 1 y se dispone de los datos comprimidos para que cumplan con el formato de intercambio de archivos JFIF, anteriormente expuesto. Este módulo es implementado completamente sobre el FPGA Spartan-3A XC3S700A.
- *Módulo de Transferencia y Almacenamiento de archivos*. Consta de una sección de hardware y una sección de software. Este módulo se encarga de transmitir los datos del archivo de imagen JPEG desde el módulo de desarrollo *Spartan 3A Starter Kit* hacia el computador que ejecuta la aplicación de Interfaz Gráfica de Control, para posteriormente, mediante ésta, recibir los bytes formateados en JFIF y dar la extensión *.jpg* para que el sistema operativo de la máquina reconozca el archivo de imagen.
- *Módulo de Control*. Este módulo se encarga de habilitar las secciones de hardware del sistema, como también de controlar el flujo de datos entre éstas.
- *Interfaz Gráfica de Control*. Este módulo es desarrollado mediante el entorno de programación Visual GUIDE disponible en el paquete computacional Matlab. Con esta interfaz, el usuario interactúa con el sistema para poder controlar las funciones de adquisición, compresión y almacenamiento, así como la visualización de resultados.

Las secciones de cada módulo desarrolladas sobre el FPGA, son descritas con el lenguaje de descripción de hardware VHDL. Una breve reseña del lenguaje VHDL se encuentra en el Anexo 8 de este proyecto.

A continuación se detalla el diseño de cada uno de los módulos expuestos anteriormente.

2.2LA INTERFAZ GRÁFICA DE CONTROL

La interfaz gráfica de control del sistema, es realizada mediante el entorno de programación Visual GUIDE de Matlab. La herramienta GUIDE permite realizar interfaces gráficas de usuario con muchas de las características básicas que poseen los programas de desarrollo de entornos de interfaz gráfica como lo son Visual Basic o Visual C++ [1].

En la Figura 2-2 se puede observar la interfaz de control gráfica del sistema, desarrollada mediante la herramienta GUIDE. La interfaz básicamente consiste de botones (*push button*) que al presionarlos (dar *click*) ejecutarán la función seleccionada del sistema. La interfaz gráfica de control posee cuatro secciones: las funciones del sistema, los controles del puerto serial, la sección de estado y la sección de menús.

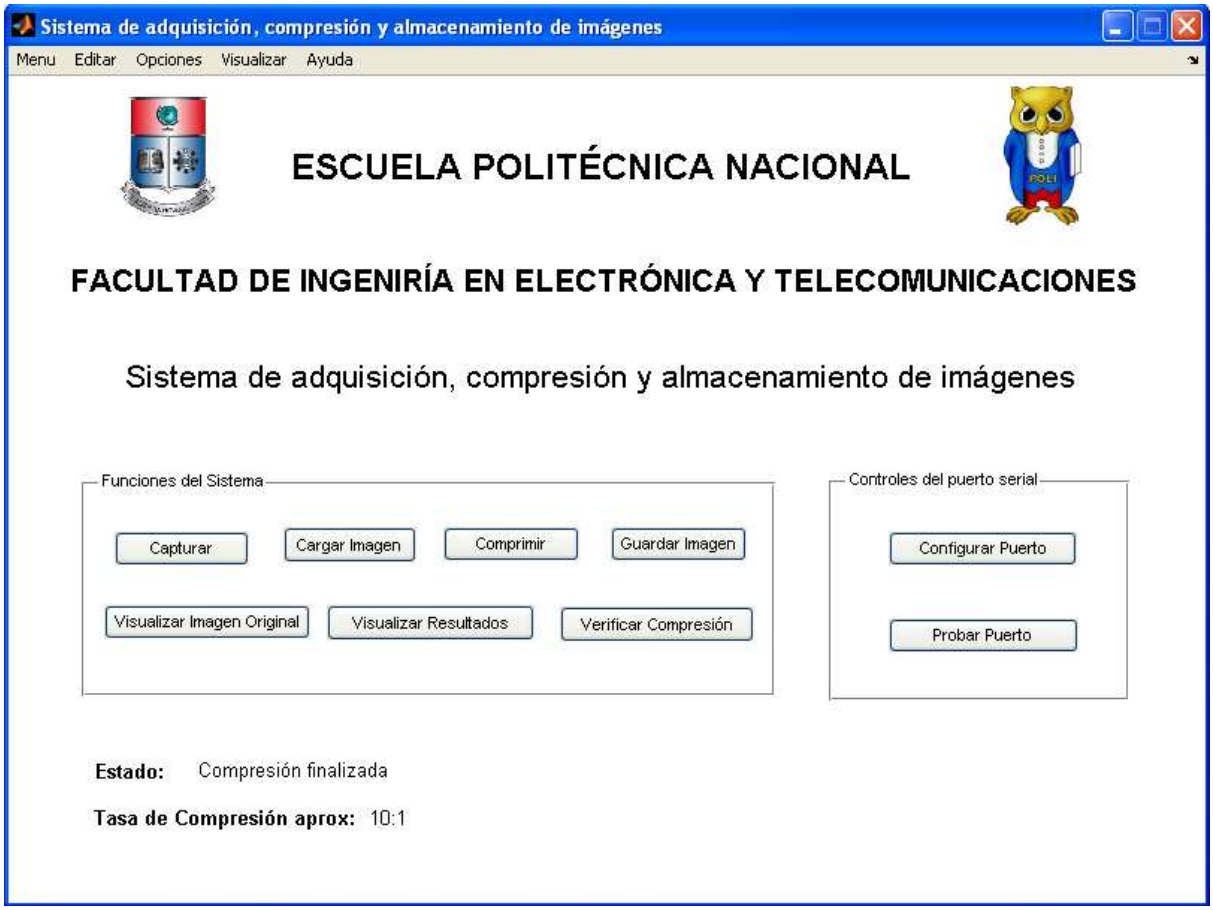


Figura 2- 2. Interfaz Gráfica de Control del sistema

En la sección de funciones del sistema se agrupa un conjunto de botones que activan la ejecución de las acciones que realiza el sistema de adquisición, compresión y almacenamiento de imágenes, las cuales se detallan a continuación:

- **Capturar:** Crea un objeto de video en Matlab, adquiere una trama desde una cámara web, la redimensiona a 160x120 pixeles y obtiene los datos de la componente de luminancia (la imagen monocromática) para posteriormente enviarla, a través del puerto serial, al módulo de desarrollo para su compresión en JPEG.
- **Cargar Imagen:** Adquiere una imagen desde un archivo con formato bmp, la redimensiona a 160x120 pixeles y obtiene los datos de la componente de luminancia, para su posterior envío hacia el módulo de desarrollo, para su compresión con el estándar JPEG.
- **Comprimir:** Envía los datos obtenidos mediante cualquiera de las dos opciones antes mencionadas hacia el módulo de desarrollo, donde el módulo de compresión albergado en el FPGA, realiza la compresión con el estilo de codificación *Baseline* del estándar JPEG. Después de enviar los datos de la imagen sin comprimir hacia el módulo de desarrollo, el programa que ejecuta esta acción espera la recepción del *stream* de bytes del archivo de imagen JFIF que se envía desde el módulo de desarrollo, luego de realizada la compresión, hacia la computadora donde se ejecuta la interfaz gráfica de control, donde es almacenado en un archivo temporal con extensión *.jpg*.
- **Guardar Imagen:** Guarda la imagen adquirida y comprimida, en cualquier localidad del sistema de archivos del sistema operativo y con el nombre de archivo que el usuario del sistema escoja.
- **Visualizar imagen original:** Visualiza en una nueva ventana la imagen monocromática adquirida anteriormente, mediante las opciones capturar o cargar imagen. En la Figura 2-3 se muestra esta interfaz gráfica.

Figura 2- 3. Visualización de imagen original

- **Visualizar Resultados:** Despliega en una nueva ventana la imagen comprimida con el esquema de codificación *Baseline* del estándar JPEG, realizado por el módulo de compresión implementado en el FPGA. También despliega, conjuntamente con la imagen comprimida, la tasa de compresión obtenida. En la Figura 2-4 se indica esta ventana.

Figura 2- 4. Visualización de la imagen comprimida

- **Verificar Compresión:** Mediante esta opción se ejecuta la aplicación denominada *JPEGsnoop*, la cual es un analizador de ficheros de fotogramas, como los archivos de imagen *jpg*. *JPEGsnoop* permite verificar y analizar los campos de datos que poseen estos ficheros. Para el caso concreto de ficheros de imágenes comprimidas con JPEG, la aplicación proporciona información sobre los siguientes campos [2]:

    - Tablas de cuantización de las componentes de luminancia y crominancia.
    - Frecuencias de muestreo de las componentes de crominancia.
    - Tablas de Huffman.
    - Estimación del factor de calidad de la imagen.
    - Información de la cabecera JFIF.

La sección de controles del puerto serial agrupa dos botones para la selección del puerto COM del computador, así como para probar la conectividad con el módulo de desarrollo. Estos botones son:

- **Configurar puerto**: Mediante esta opción se elije el número de puerto COM con el que se realiza la comunicación hacia el módulo. Al dar *click* sobre este botón se despliega una lista para poder escoger el puerto, tal como lo indica la Figura 2-5. Al presionar el botón “Seleccionar”, la interfaz gráfica de control creará, mediante un programa realizado en Matlab, un objeto serial con el número de COM seleccionado y que tendrá las siguientes características:

    - Bits por segundo: 57600
    - Bits de datos: 8
    - Bits de paridad: ninguno
    - Bits de parada: 1

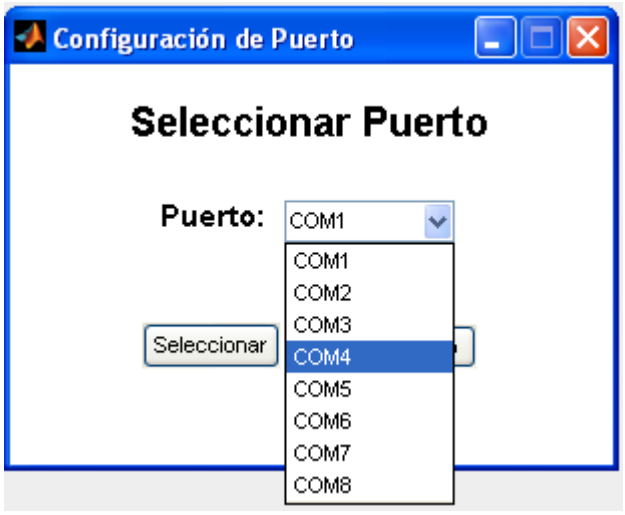


Figura 2- 5. Selección del puerto serial

- **Probar puerto**: Mediante esta opción se comprueba la comunicación serial entre la interfaz gráfica de control y el módulo de desarrollo *Spartan 3A Starter Kit*; para ello, el programa de la interfaz gráfica de control envía un carácter de prueba hacia el módulo de desarrollo, donde el sistema que se ejecuta en el FPGA reenvía el carácter más uno. El carácter de prueba es el ASCII 84_{10} correspondiente a la letra “T”, por lo que si la conexión entre el módulo de desarrollo y su interfaz de control gráfica es correcta, el módulo debe devolver el carácter ASCII 85_{10} correspondiente a la letra “U”. Si la conexión es favorable, se informa al usuario del sistema a través de la sección de estado (al igual que en el caso opuesto).

En la sección de estados, se indica al usuario del sistema si la ejecución de la acción escogida ha finalizado satisfactoriamente. También para el caso de la opción *Comprimir*, indica la tasa de compresión aproximada alcanzada para la imagen adquirida.

En la sección de menús se agrupan las opciones de las secciones de funciones del sistema y de controles del puerto serial, en los siguientes menús:

- Menú:
    - Guardar Imagen
    - Salir
- Editar:
    - Seleccionar puerto
    - Probar Puerto
- Opciones:
    - Cargar Imagen
    - Capturar Imagen
    - Comprimir Imagen
    - Verificar Compresión
- Visualizar:
    - Imagen Original
    - Imagen Comprimida
- Ayuda:
    - Acerca de

Como se mencionó en la introducción de este capítulo, el Sistema de Adquisición, Compresión y Almacenamiento de Imágenes consta de cinco módulos, algunos de los cuales poseen secciones de hardware y software. A continuación se presenta el proceso de diseño de cada uno de los módulos que conforman el sistema.

2.3 MÓDULO DE ADQUISICIÓN DE IMÁGENES

Este módulo se encarga de adquirir los datos de imágenes monocromáticas sin compresión. Como se puede observar en la Figura 2-1, las imágenes pueden provenir desde dos fuentes: archivos de imágenes con formato bmp y desde un cuadro adquirido de una cámara web. En este módulo se tiene una sección hecha en software y otra en hardware.

2.3.1 SECCIÓN DE SOFTWARE DEL MÓDULO DE ADQUISICIÓN DE IMÁGENES

En la sección de software de este módulo, se tienen rutinas escritas en lenguaje Matlab, que permiten obtener los datos de la imagen sin comprimir de las formas que se enunciaron anteriormente y que a continuación se detallan.

2.3.1.1 Obtención de imágenes mediante la cámara web

Una de las ventajas que presenta el lenguaje de programación de Matlab, es la facilidad con que se pueden manejar dispositivos de hardware de la computadora como objetos dentro de un programa; en el caso de una cámara web: un objeto de video. Un objeto de video representa la conexión entre Matlab y un dispositivo de adquisición de imágenes particular.

Para crear un objeto de video, el lenguaje de programación de Matlab tiene el siguiente formato:

```
obj = videoinput(adaptorname,deviceID,format)
```

Donde:

obj: representa el nombre que se le desee dar al objeto de video.
videoinput: es la sentencia del lenguaje de programación de Matlab para crear un objeto de video.
adaptorname: es la cadena de caracteres que especifica el nombre del adaptador usado para comunicarse con el dispositivo.
deviceID: es un valor numérico escalar que especifica un dispositivo particular disponible a través del adaptador *adaptorname*.
format: especifica el formato (o formatos) de video que puede manejar el dispositivo en cuestión.

El dispositivo de adquisición que se utiliza para el desarrollo de este módulo es una cámara web de marca Logitech, cuyo objeto dentro del programa en Matlab se define como sigue:

```
vid = videoinput('winvideo',1,'RGB24_352x288');
```

Con esta sentencia se ha creado un objeto de video denominado *vid*, cuyo controlador (adaptador), es un controlador genérico de Windows, y su formato de video está en el espacio de color RGB de 24 bits con una resolución de imagen de 352x288 pixeles.

Una vez que se crea el objeto de video, se puede obtener un cuadro desde la cámara mediante la siguiente sentencia:

```
frame = getsnapshot(vid);
```

Después de obtener el cuadro (imagen) desde la cámara web, es necesario redimensionarlo a 160x120 pixeles y obtener su versión monocromática (componente de luminancia). Esto se hace debido a que la capacidad de memoria embebida (RAM embebida) del FPGA XC3S700A de la Familia Spartan-3A, no permite almacenar imágenes de mayor resolución con sus tres componentes; esto será más evidente cuando se dimensione más adelante el buffer temporal. Para lograr obtener la imagen en escala de grises y redimensionarla, se utiliza la siguiente sentencia en Matlab:

```
image = rgb2gray(imresize(frame, [120, 160]));
```

Con la función *imresize* se realiza el dimensionamiento de la imagen *frame* a 160x120 pixeles y la función *rgb2gray* convierte la imagen redimensionada a escala de grises para ser almacenada en la variable *image*.

Para finalizar la adquisición de la imagen, ésta debe almacenarse en una variable global para poder ser utilizada en los siguientes procesos del sistema. Para ello se utilizan las siguientes sentencias:

```
handles.metricdata.image=image;
guidata(hObject,handles);
```

2.3.1.2 Obtención de imágenes mediante archivos BMP

La otra opción que posee el sistema, es obtener los datos de la imagen sin comprimir, a partir de un archivo bmp ubicado en cualquier localidad del sistema de archivos. Para esto se debe obtener el nombre y dirección del archivo bmp elegido, con la siguiente función de Matlab:

```
[filename1, pathname1] = uigetfile('imagenes\*.bmp','Seleccione un archivo *.bmp');
```

Con esta función se despliega una ventana que permite seleccionar el archivo bmp dentro del sistema de archivos y almacenar en dos variables tanto el nombre del archivo (*filename1*) como su localización (*pathname1*).

Después de realizada la localización del archivo de imagen bmp, con la función *imread* de Matlab se puede obtener los datos de las componentes de dicha imagen como se muestra a continuación:

```
a=imread(direccion1);
```

En donde *dirección1* representa la concatenación de las variables *filename1* y *pathname1* en un vector fila; la concatenación se obtiene con la siguiente función:

```
direccion1=cat(2,pathname1,filename1);
```

Una vez obtenidos los datos de la imagen, se debe realizar los procesos de redimensionamiento, obtención de la componente de luminancia y almacenamiento en una variable, como se indica a continuación:

```
image = rgb2gray(imresize(a, [120, 160]));
handles.metricdata.image=image;
guidata(hObject,handles);
```

Las secciones de código de lenguaje Matlab, de las dos opciones de adquisición que se presentaron anteriormente, son las sentencias subyacentes que se ejecutan en los botones *Capturar* y *Cargar Imagen* de la Interfaz Gráfica de Control.

Una vez que se poseen los datos de la imagen sin comprimir, es necesario ordenar estos datos de forma tal que la imagen quede dividida en bloques de 8x8 pixeles. A pesar que esto puede resolverse en hardware mediante un esquema de direccionamiento en el buffer temporal, éste consume demasiados bloques de RAM embebida, más de los que se pueden utilizar para este fin, por lo que esta tarea se la realizó en Matlab mediante lazos *for* anidados como se muestra a continuación:

```
a=1;
b=1;
c=1;
      for i=1:15
          for j=1:20
              block8x8=image(a:a+7,b:b+7);
              for k=1:8
                  for l=1:8
                      image_vector(c)=block8x8(k,l);
                      c=c+1;
                  end
              end
              b= b+8;
          end
          b=1;
          a=a+8;
      end
```

De esta manera, el vector fila *image_vector* contiene los datos de la imagen dividida en bloques de 8x8 y ordenados como se muestra en la Figura 2-6.

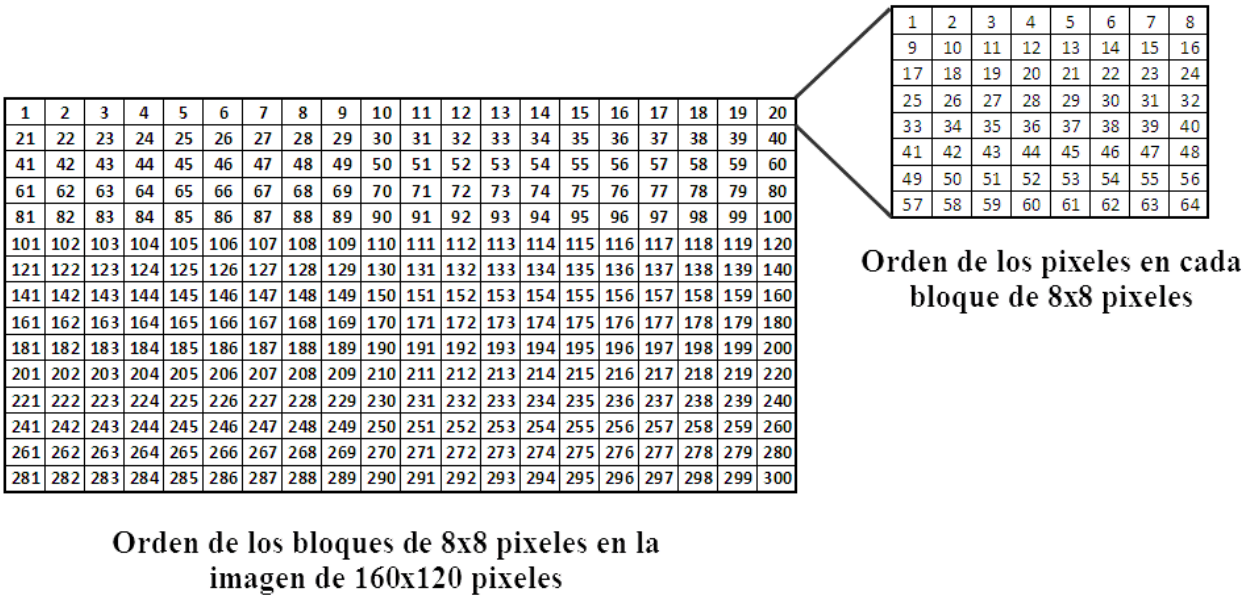


Figura 2- 6. Orden de los bloques 8x8 de la imagen de 160x120 pixeles

Concluido el proceso de la división de la imagen en bloques de 8x8 pixeles, se requiere enviar estos datos hacia el módulo de desarrollo, para que sean comprimidos con el esquema de codificación *Baseline*. Para ello, es necesario realizar dos tareas: crear y manejar un objeto serial en el programa de la interfaz gráfica de control y sintetizar un módulo UART-RS232 sobre el FPGA, para poder realizar la comunicación serial entre la interfaz de control y el módulo de desarrollo.

2.3.1.3 Configuración del puerto serial en la interfaz de control

La configuración del puerto serial en el programa de la interfaz gráfica de control, se realiza con la creación de un objeto serial, el cual permite la comunicación del puerto serial con Matlab. Para crear un objeto serial se utiliza la siguiente sentencia:

```
s=serial('COM1');
```

Una vez que el objeto serial es creado, es necesario definir sus propiedades de la siguiente manera:

```
s.BaudRate=57600;
s.DataBits=8;
s.Parity='none';
s.StopBits=1;
s.Terminator={'CR/LF',''};
s.Timeout=16;
s.InputBufferSize=30000;
s.OutputBufferSize=30000;
```

en donde:

BaudRate: establece la velocidad del puerto serial en bits por segundo (bps).

DataBits: establece el número de bits de datos.

Parity: determina el tipo de paridad utilizada.

StopBits: determina el número de bits de parada.

Terminator: establece un carácter ASCII para terminar la transmisión.

Timeout: estable un límite de tiempo para terminar la transmisión en caso de no recibir respuesta del otro terminal.

InputBufferSize: determina el tamaño en bytes del buffer para los datos recibidos.

OutputBufferSize: determina el tamaño en bytes del buffer para los datos que van a ser transmitidos.

Las sentencias para la creación y configuración del objeto serial en Matlab, se llevan a cabo cuando el usuario presiona el botón *Configurar Puerto* en la Interfaz Gráfica de Control.

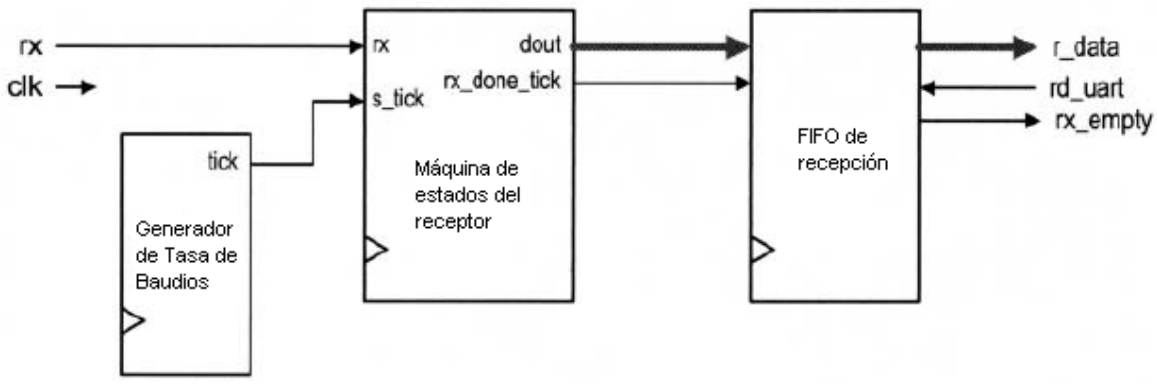
2.3.2 SECCIÓN DE HARDWARE DEL MÓDULO DE ADQUISICIÓN DE IMÁGENES

Después de realizar la configuración del objeto serial en el programa de la interfaz gráfica de control, se debe realizar la síntesis en hardware de un sistema de recepción UART compatible con las características del objeto serial que se hizo

en Matlab. La descripción del sistema de recepción UART en VHDL, ha sido tomada de [3][17], y su funcionalidad se describe a continuación.

2.3.2.1 Sistema UART: subsistema de recepción

El subsistema de recepción UART consta de tres etapas, como lo muestra la Figura 2-7.



Fuente [3]

Figura 2- 7. Etapas del subsistema de recepción UART

2.3.2.1.1 Generador de Tasa de Baudios

Mediante este bloque se genera una señal (*s_tick* en la Figura 2-7) que indica el momento que se debe tomar una muestra del bit recibido; en un intervalo de bit se toman 16 muestras. Para esto, se debe poner en 1L una señal cada vez que un contador alcance el valor 54 $\left(\frac{50\times10^6}{16\times57600}\right)$, para el caso de una velocidad de 57600 bps y un oscilador de 50MHz en el FPGA. Cabe mencionar que esta señal no actúa como otra señal de reloj en la máquina de estados del receptor, sino como una señal de habilitación para los registros que actúan como contadores en este módulo.

[17] Esto se señaló en el plan de tesis aprobado.

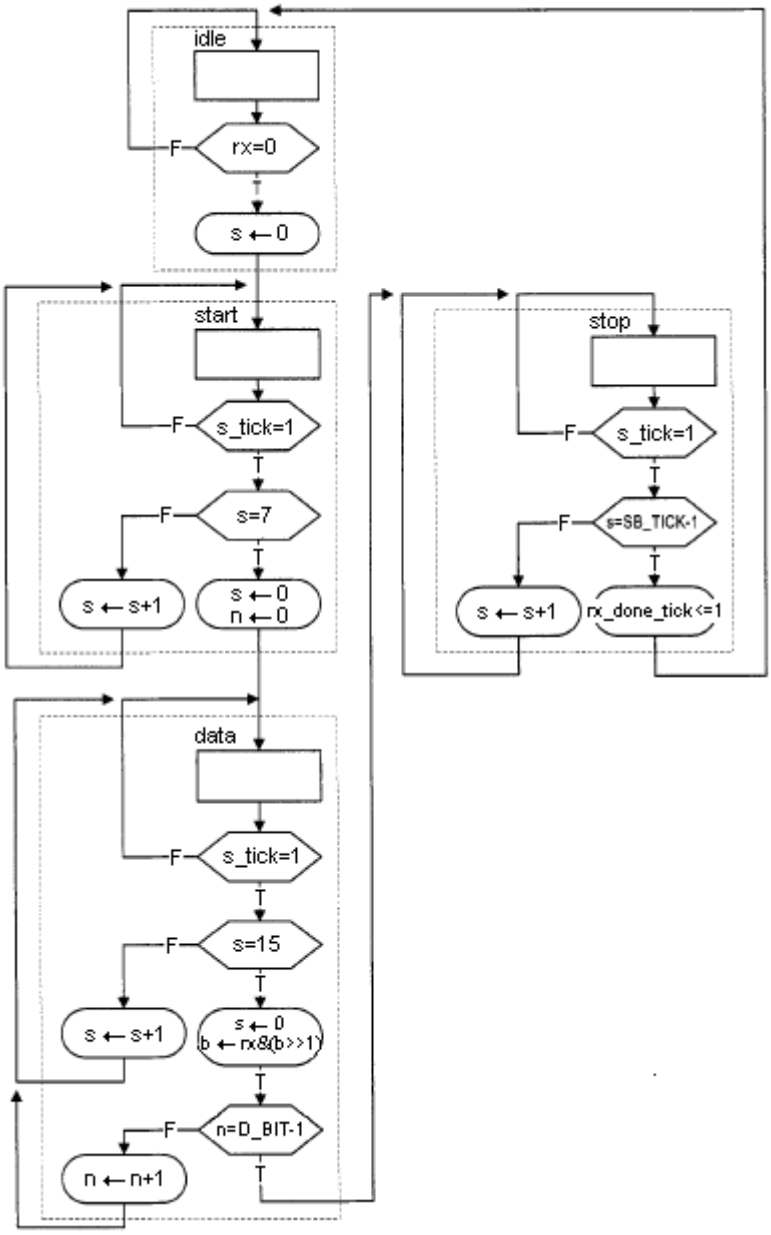
2.3.2.1.2 Máquina de estados del receptor

La máquina de estados del receptor, se basa en un esquema de muestreo, en el cual cada bit es muestreado 16 veces (16 veces la velocidad de transmisión), con el fin de poder obtener su punto medio. Si para la comunicación se asume N bits de datos y M bits de parada, el procedimiento de muestreo es el siguiente:

1. Se espera a que la señal de entrada en el receptor sea '0' (el bit de inicio), y luego se inicia un contador de muestras, que se incremente cada vez que la señal *s_tick* del generador de tasa de baudios sea 1L.
2. Cuando el contador alcanza 7, la señal de entrada ha alcanzado el punto medio del bit de inicio. El contador se reinicia a 0.
3. Cuando el contador alcanza 15, la señal de entrada alcanza el punto medio del primer bit de datos; se recupera este valor, se lo desplaza en un registro y se reinicia el contador.
4. Se repite el paso 3, N-1 veces, hasta recuperar el resto de bits de datos.
5. Si se utiliza un bit de paridad, el paso 3 se repite una vez más.
6. Repetir el paso 3, M veces más, hasta obtener los bits de parada.

La descripción en VHDL de este procedimiento corresponde a una máquina de estados, cuyo diagrama ASM[18] se indica en la Figura 2-8. Como se observa en la Figura 2-8, se definen 4 estados para el procedimiento de muestreo: *idle*, *start*, *data* y *stop*. La máquina de estados se mantiene en el estado *idle* hasta que la señal de recepción *rx* se vuelva cero (inicio de la transmisión). Los estados *start, data* y *stop* representan el procesamiento de los bits de inicio, datos y parada, respectivamente, acorde con el procedimiento de muestreo antes expuesto. En la descripción en VHDL de la máquina de estados, se incluyen la constante *D_BIT*, que indica el número de bits de datos, y la constante *SB_TICK,* que indica el número de muestras necesarias para los bits de parada, que para el caso del diseño son 8 y 16, respectivamente.

[18] **ASM**: *Algorithmic state machine*, es un diagrama especifico para describir el comportamiento de máquinas de estados finitos y consta de tres elementos: estados, condicionales y salidas condicionales.



Fuente [3]

Figura 2- 8. Diagrama de flujo de la máquina de estados del receptor

La máquina de estados del receptor posee dos contadores representados por dos registros denominados s y n. El registro *s* cuenta el número de muestras que deben tomarse en cada estado, a saber 7 en el estado *start,* 15 en el estado *data* y *SB_TICK* en el estado *stop*. El registro n lleva la cuenta de los bits de datos recibidos en el estado *data*, es decir cuenta hasta *D_BIT*.

Los bits recibidos son desplazados en un registro denominad *b*, el cual después es almacenado en el FIFO de recepción a través de la señal de salida *dout* de la máquina de estados del receptor.

Se incluye la señal *rx_done_tick*, la cual es puesta en 1L por un ciclo de reloj, después de que el proceso de recepción ha finalizado, esta señal también sirve para habilitar la escritura del registro *b* en el FIFO de recepción.

2.3.2.1.3 FIFO de recepción

El FIFO de recepción debe retener los datos que han sido obtenidos desde la máquina de estados del receptor, para su posterior lectura y procesamiento en los siguientes módulos del sistema.

El FIFO de recepción ha sido implementado mediante un *IP Core*, el cual tiene las siguientes características:

- Capacidad de 64 palabras de 8 bits.
- Implementación basada en LUTs.
- Sincronización común (señal de reloj común) para operaciones de lectura y escritura.
- Señales de estado *full* (lleno) y *empty* (vacío).
- *Reset* Asincrónico.
- Señales de habilitación de lectura y escritura.

2.3.2.1.4 Señales de interfaz del subsistema de recepción UART

En el subsistema de recepción UART que se indica en la Figura 2-7, la entidad VHDL de más alto nivel, que contiene los tres componentes presentados en la figura, posee los puertos de interfaz con los cuales el subsistema de recepción UART interactúa con el resto del sistema. Las señales externas que tiene el subsistema de recepción son *rx y clk*, las cuales están conectadas, a través de la

entidad de más alto nivel de todo el sistema, a los pines del FPGA que se conectan al pin 3 del puerto RS-232 (recepción) del módulo de desarrollo y al oscilador del FPGA, respectivamente.

Las señales internas son las que sirven de interconexión con el resto del sistema de adquisición, compresión y almacenamiento de imágenes; estas señales son:

- *rx_empty*: indica si existe en el FIFO de recepción un dato que ha sido recibido.
- *rd_uart*: es la señal de habilitación de lectura para el FIFO de recepción.
- *r_data*: es el bus de 8 bits que lleva el dato leído desde el FIFO de recepción.

2.3.2.2 Buffer temporal

El buffer temporal consiste de una memoria RAM donde se almacenan los datos de la imagen sin comprimir que son recibidos a través del subsistema de recepción UART, como lo indica la Figura 2-1. La resolución en pixeles de la imagen sin comprimir, ha sido tomada en base a la capacidad de almacenamiento en los bloques de RAM embebida que tiene el FPGA XC3S700A de la familia Spartan-3A. El FPGA utilizado posee 360 kbits distribuidos en 20 bloques de RAM embebida de 18 kbits cada uno. Una imagen monocromática de 160x120 pixeles puede ser almacenada en 150 kbits $\left(\frac{160\times120\times8}{1024}\right)$, para lo cual se usan 9 bloques de RAM embebida. La memoria RAM que va almacenar la imagen monocromática sin comprimir, por tanto, debe tener una capacidad de almacenamiento de 19200 palabras de 8 bits (160x120 bytes).

Se ha implementado esta memoria utilizando un IP Core con las siguientes características:

- Memoria RAM dual port, con puerto de lectura/escritura y puerto de solo lectura con sincronización común.
- Habilitación de lectura en el puerto de solo lectura.
- Capacidad de 19200 bytes.

La memoria RAM creada para funcionar como buffer temporal, es de tipo dual port, esto es para tener un mejor control sobre las operaciones de lectura y escritura en puertos distintos; además, se maximiza la funcionalidad de los bloques de RAM embebida, ya que como se mencionó en el Capítulo 1, estos bloques físicamente son dual port, por lo cual esta implementación no consume recursos lógicos adicionales.

Ya que la división de la imagen monocromática de 160x120 pixeles en bloques de 8x8 y posterior reordenamiento, se realizó mediante una rutina en el programa en Matlab de la interfaz gráfica de control, el esquema de direccionamiento del buffer temporal es un simple contador ascendente tanto para la operación de lectura como para la de escritura. El generador de direcciones, tanto de lectura como de escritura, es descrito en VHDL mediante un segmento de código secuencial, como el que se muestra a continuación:

- Para escritura:

```
process (clk)
begin
   if clk'event and clk = '1' then
      if rst = '1' then
         address_ram_w <= (others =>'0');
      else
         -- Se verifica la habilitación de generación de direcciones
         -- de escritura
         if wr_en_ram = '1' then
            -- Se verifica si se ha alcanzado las 19200
            -- direcciones de escritura
            if address_ram_w < "100101011111111" then
               address_ram_w <= address_ram_w + "0000000000000001";
            -- Si se alcanza las 19200 direcciones se
            -- reinicia el contador
            else
               address_ram_w <= (others => '0');
            end if;
         -- Si no se habilita la generación de direcciones
         -- se mantiene el valor en el puerto de dirección
         else
            address_ram_w <= address_ram_w;
         end if;
      end if;
   end if;
end process;
```

- Para lectura:

```
process (clk)
begin
   if clk'event and clk = '1' then
      if rst = '1' then
         address_ram_r <= (others => '0');
      else
         -- Se verifica la habilitación de generación de direcciones
         -- de lectura
         if ena_jpeg = '1' then
            -- Se verifica si se ha alcanzado las 19200
            -- direcciones de lectura
            if address_ram_r < "100101011111111" then
               address_ram_r <= address_ram_r + "000000000000001";
            -- Si se alcanza las 19200 direcciones se
            -- reinicia el contador
            else
               address_ram_r <= (others => '0');
            end if;
         -- Si no se habilita la generación de direcciones
         -- se reinicia el contador
         else
            address_ram_r <= (others => '0');
         end if;
      end if;
   end if;
end process;
```

En ambos esquemas de direccionamiento se posee una señal que habilita la generación (cuenta) de direcciones; en el caso del direccionamiento de escritura la señal habilitante es *wr_en_ram*, y en el caso de lectura la señal habilitante es *ena_jpeg*. Hay que destacar que estas mismas señales son las que habilitan las operaciones de lectura y escritura de la memoria RAM que almacena la imagen monocromática. La asignación de los valores activos de estas señales, es controlada por el módulo de control a continuación descrito.

2.4MÓDULO DE CONTROL

El módulo de control es la entidad VHDL del sistema de adquisición, compresión y almacenamiento de imágenes que administra la habilitación de los procesos que se cumplen en las secciones de hardware de los otros módulos, como también el flujo de datos entre éstos.

En la Figura 2-9, se muestra una representación de la entidad VHDL de este módulo. Básicamente, el módulo de control es una máquina de estados, cuyo diagrama ASM se muestra en la Figura 2-10.

En el diagrama ASM de la Figura 2-10, sólo los puertos que cambian su contenido a su valor activo son presentados, el resto de puertos no son incluidos en la figura y debe entenderse que permanecen con su valor inicial. En la Tabla 2-1 se indican la dirección, la longitud en bits y el valor activo de los puertos que posee la entidad VHDL del módulo de control.

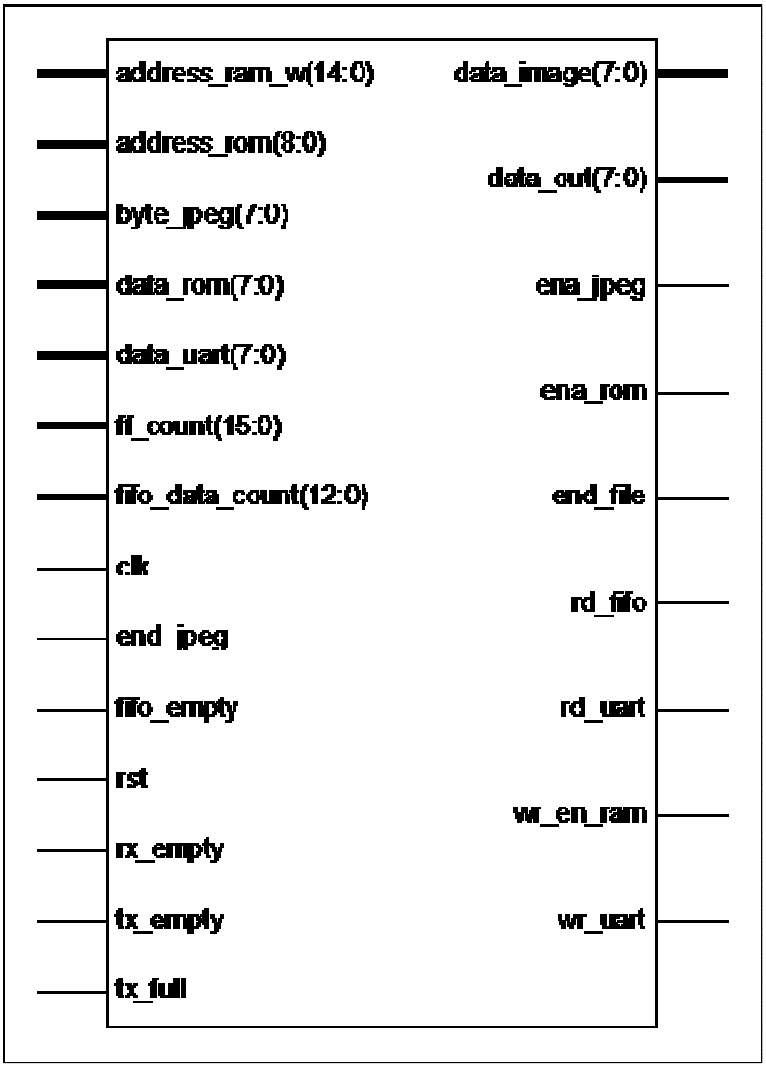


Figura 2- 9. Representación de la entidad VHDL del módulo de control

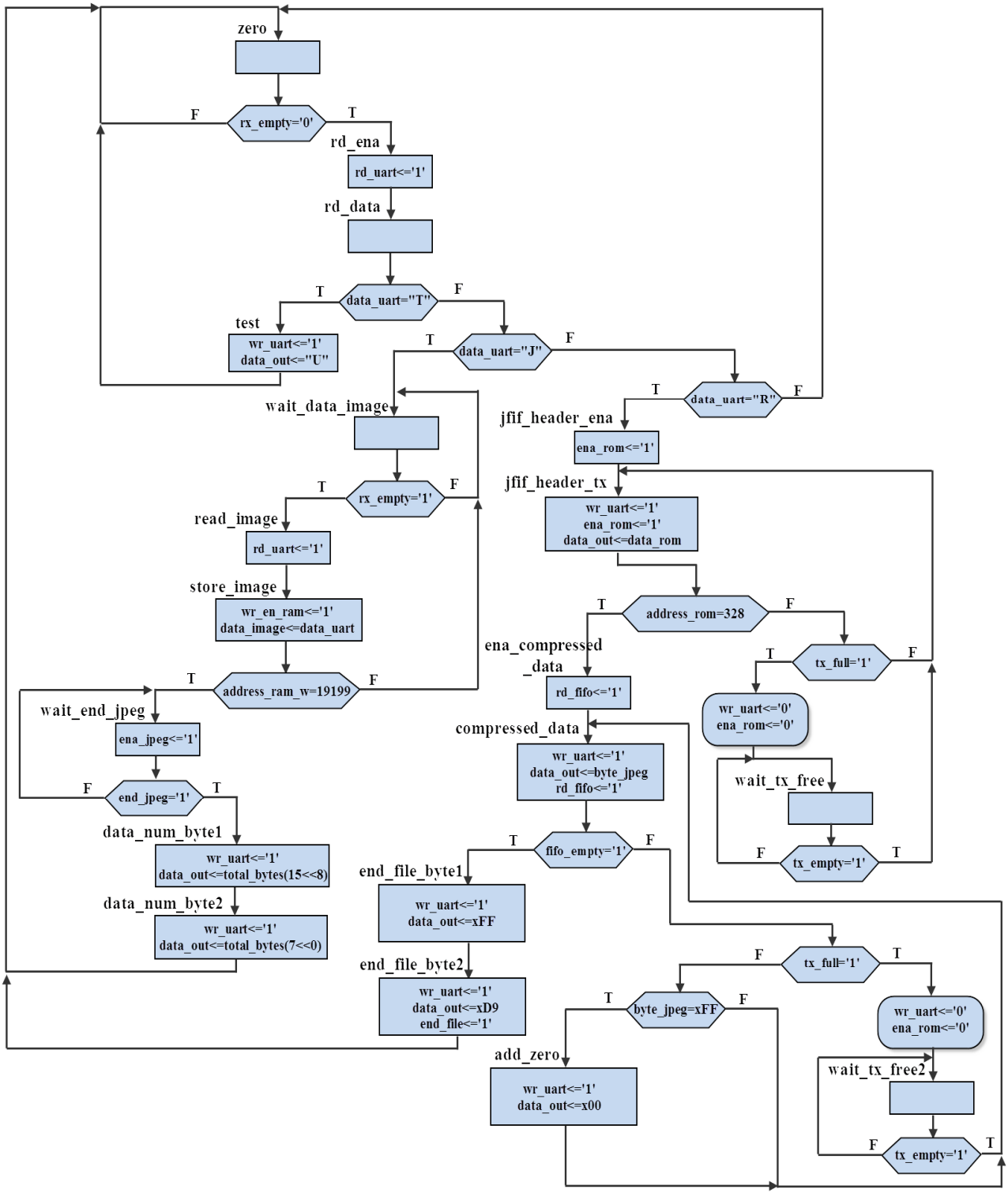


Figura 2- 10. Diagrama ASM del módulo de control

Tabla 2- 1. Puertos de la entidad VHDL del módulo de control

Puerto	Dirección	Longitud en bits	Valor activo	Descripción
rx_empty	Entrada	1	1L	Indica si no existen datos en el FIFO de recepción.
tx_empty	Entrada	1	1L	Indica si no existen datos en el FIFO de transmisión.
tx_full	Entrada	1	1L	Indica si el FIFO de transmisión está lleno.
data_uart	Entrada	8	N/A	Es el puerto de entrada para los datos que han sido recibidos a través del sistema UART.
rd_uart	Salida	1	1L	Puerto que habilita la lectura del FIFO de recepción.
wr_uart	Salida	1	1L	Puerto que habilita la escritura en el FIFO de transmisión.
data_out	Salida	8	N/A	Puerto de salida para los datos que se van a transmitir a través del sistema UART.
address_ram_w	Entrada	15	N/A	Puerto para el monitoreo del bus de dirección de la memoria RAM de almacenamiento temporal.
address_rom	Entrada	9	N/A	Puerto para el monitoreo del bus de dirección de la memoria ROM que posee la cabecera de archivo JFIF.
byte_jpeg	Entrada	8	N/A	Puerto que recoge los bytes de la imagen comprimida desde el FIFO de almacenamiento del módulo de compresión.
data_rom	Entrada	8	N/A	Puerto que recoge los bytes que provienen de la memoria ROM que posee los datos de la cabecera de archivo de imagen JFIF.
fifo_data_count	Entrada	13	N/A	Indica el número de bytes almacenados en el FIFO del módulo de compresión.
ff_count	Entrada	16	N/A	Indica el número de bytes con valor xFF, que existe en los datos de la imagen comprimida.
end_jpeg	Entrada	1	1L	Indica cuándo ha finalizado el proceso de compresión JPEG en el módulo de compresión.
fifo_empty	Entrada	1	1L	Indica si el FIFO de almacenamiento en el módulo JPEG está vacío.
data_image	Salida	8	N/A	Puerto de salida para los bytes de datos de la imagen sin comprimir.
ena_jpeg	Salida	1	1L	Señal de habilitación del módulo de compresión.
ena_rom	Salida	1	1L	Señal de habilitación de lectura de la memoria ROM que contiene la cabecera del archivo de imagen JFIF.
end_file	Salida	1	1L	Indica cuándo se ha transmitido el último byte del archivo de imagen JFIF.
rd_fifo	Salida	1	1L	Señal de habilitación de lectura del FIFO de almacenamiento del módulo de compresión.

wr_en_ram	Salida	1	1L	Señal de habilitación de escritura de la memoria RAM de almacenamiento temporal.
clk	Entrada	1	N/A	Señal de reloj del módulo de control
rst	Entrada	1	1L	Señal de reset del módulo de control

2.4.1 TAREAS DEL MÓDULO DE CONTROL

Las tareas de control, que se ejecutan en este módulo, son clasificadas de la siguiente manera:

2.4.1.1 Control del sistema UART

En cada uno de los estados que posee el módulo de control, se procede a manejar las señales de habilitación del sistema UART. Los valores de asignación de cada señal de control del sistema UART, están en función de los procesos realizados en cada estado.

2.4.1.2 Lectura de la acción a realizar enviada desde la interfaz gráfica de control

Desde la interfaz gráfica de control se envían las acciones que debe realizar el módulo de desarrollo, en forma de caracteres ASCII. Se definieron tres caracteres ASCII para las tres operaciones que realiza el módulo de desarrollo, a saber:

- El carácter ‘T’ para realizar una prueba de conexión.
- El carácter ‘J’ para habilitar la recepción de datos de la imagen y su posterior compresión.
- El carácter ‘R’ para habilitar la transmisión del *stream* de bytes de datos de la imagen comprimida en JPEG, con el formato JFIF.

En la Figura 2-11, se indica la sección del diagrama ASM del módulo de control que realiza el proceso indicado. Cuando se inicia el módulo de desarrollo, el módulo de control se encuentra en su estado inicial *zero*; cuando la señal *rx_empty* indica que un dato ha sido recibido desde la interfaz gráfica de control

(*rx_empty* = 0L), se produce la transición al estado *rd_ena* y posteriormente a *rd_data*. En el estado *rd_ena*, se habilita la lectura del FIFO de recepción a través de la señal de control *rd_uart*. En el estado *rd_data* se recepta el dato proveniente del FIFO de recepción a través del puerto *data_uart*, y se hace la comparación con los tres caracteres ASCII correspondientes a las acciones que se desea que haga el módulo de desarrollo. Si existe coincidencia con uno, se hace la transición al correspondiente estado: *test* si el carácter recibido es “T”, *wait_data_image* si es “J” y *jfif_header_ena* si es “R”. Si no existe coincidencia se hace la transición al estado inicial *zero*.

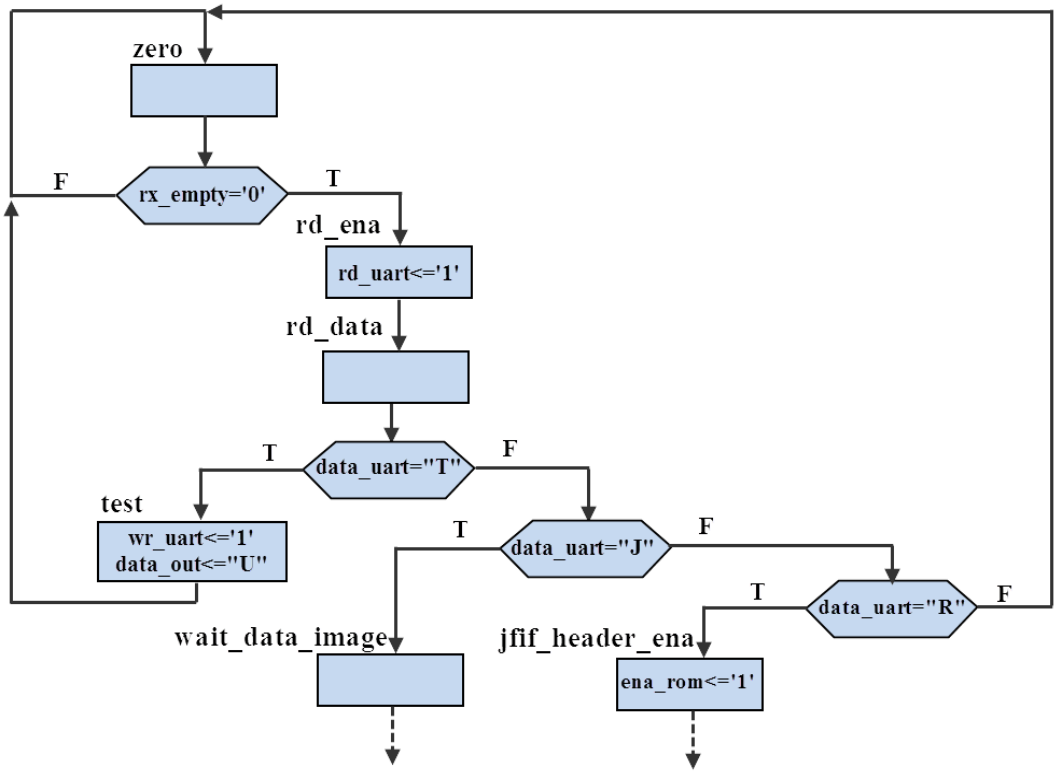


Figura 2- 11. Estados para la definición de tarea del módulo de desarrollo

En el estado *test*, se realiza una prueba de conexión que consiste en enviar a través del sistema UART, el código ASCII del carácter “T” más uno, correspondiente al carácter “U”. Esto se logra asignando el código ASCII del carácter “U” al puerto de salida *data_out* y activando la señal de escritura del FIFO de transmisión, *wr_uart*. Después de hacer estas acciones, se hace la transición al estado inicial *zero*. En el programa de la interfaz gráfica de control se

verifica el carácter que envía el módulo de desarrollo para comprobar la conexión, y se informa al usuario del sistema sobre el estado de la conexión, como ya se explicó anteriormente.

En la Figura 2-12 se indican los resultados de la simulación realizada para esta tarea del módulo de control.

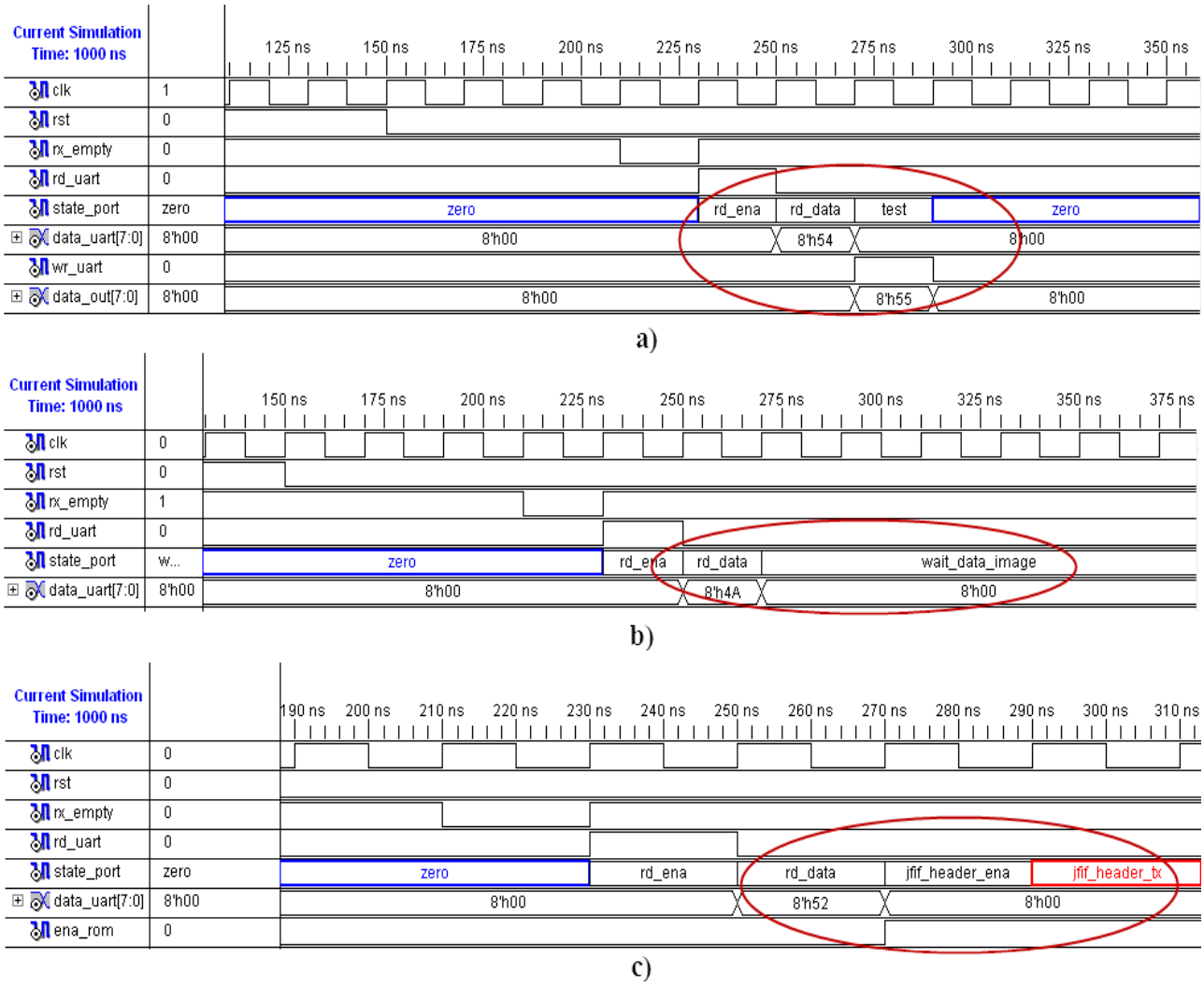


Figura 2- 12. Simulación de la tarea de lectura de la acción a realizar en el módulo de control: a) transición al estado test b) transición al estado wait_data_image c) transición al estado jfif_header_ena

Como se puede observar en la Figura 2-12, la descripción VHDL del módulo de control cumple con el comportamiento que se planificó en el diagrama ASM de la

Figura 2-11 que corresponde a la tarea de lectura de la acción que debe realizar el módulo de desarrollo.

2.4.1.3 Control de escritura del buffer de almacenamiento temporal

En la Figura 2-13 se indica la sección del diagrama ASM correspondiente a esta tarea. En el estado *wait_data_image*, se espera la recepción de los datos de la imagen monocromática sin comprimir, esto se logra monitoreando el valor de la señal *rx_empty* mediante un proceso (*process*) sensible a esta señal; cuando la señal está en estado bajo (0L) se ha recibido un dato de la imagen sin comprimir y se hace la transición al estado *read_image*.

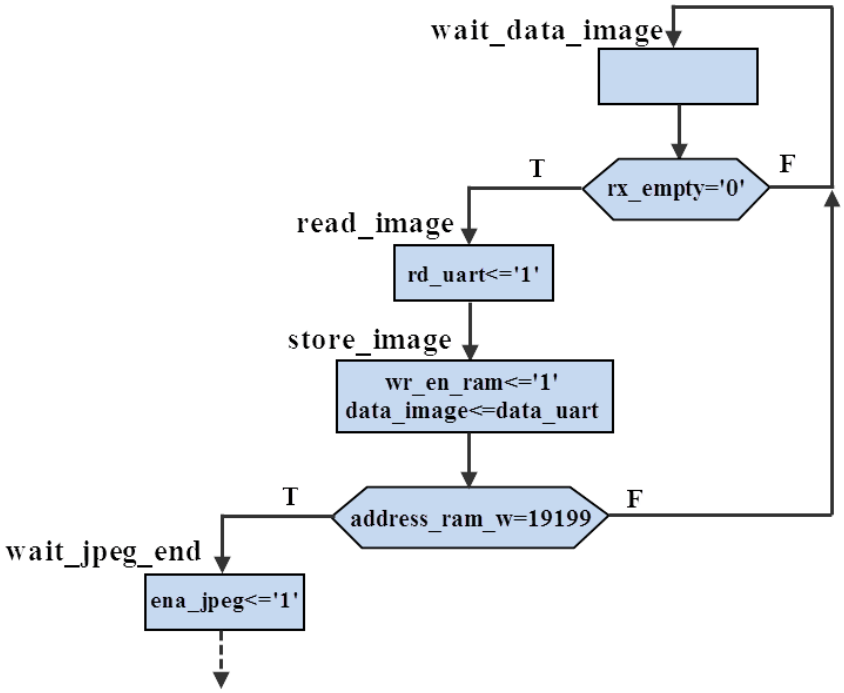


Figura 2- 13. Estados para el control del buffer temporal

En el estado *read_image* se activa la señal de lectura del FIFO de recepción a través del puerto *rd_uart* del módulo de control, y se hace la transición al estado *store_image*, en el cual se recibe el byte de dato de la imagen, a través del puerto data_uart; a su vez este mismo dato es asignado al puerto de salida *data_image*, el cual está conectado con el puerto de lectura/escritura de la memoria RAM del

buffer de almacenamiento temporal. Al mismo tiempo, en el estado *store_image*, se activa la señal de habilitación de escritura, *wr_en_ram*, para la memoria RAM y para el generador de direcciones de escritura, que se mencionó anteriormente. Mientras no se complete la recepción y almacenamiento de los 19200 bytes de la imagen monocromática, se mantienen los procesos realizados en el lazo de estados *wait_image – read_image – store_image*, como lo indica la Figura 2-13; para este fin, en el estado *store_image*, se monitorea el valor que está asignado en el puerto de direcciones de escritura de la memoria RAM, a través del puerto de entrada *address_ram_w* del módulo de control. Cuando se ha alcanzado las 19200 direcciones de escritura, se produce la transición al estado *wait_jpeg_end*.

En la Figura 2-14 se indican los resultados de la simulación para esta tarea del módulo de control.

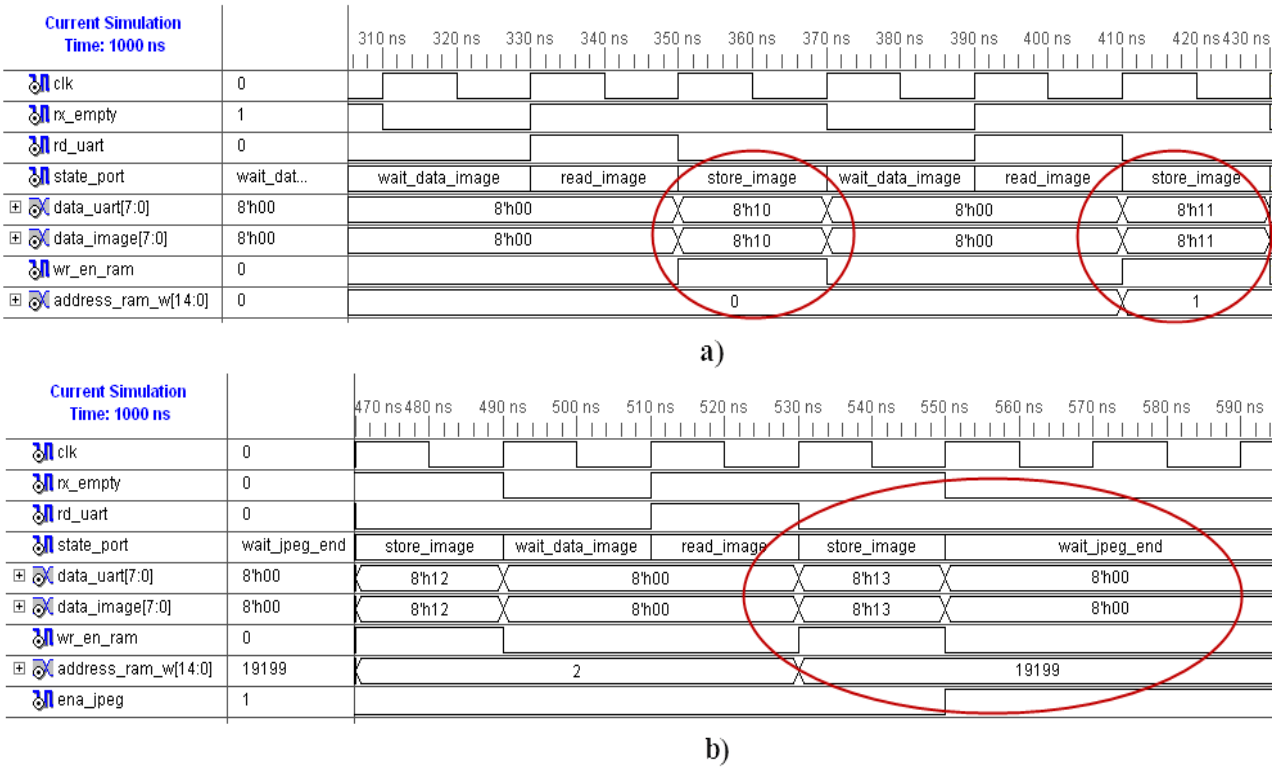


Figura 2- 14. Resultados de la simulación para la tarea de control de escritura del buffer de almacenamiento temporal: a) Procedimiento de escritura en el buffer b) finalización del proceso de almacenamiento

Como se observa en la simulación de la Figura 2-14, la descripción VHDL del módulo de control cumple con el comportamiento planteado en el diagrama ASM de la Figura 2-13, correspondiente a la acción de control de escritura del buffer de almacenamiento temporal.

2.4.1.4 Habilitación del módulo de compresión JPEG

En el estado *wait_jpeg_end* se realizan dos procesos, como se indica en la Figura 2-15. El primer proceso consiste en activar la señal de habilitación para el módulo de compresión JPEG. Esto se hace asignando un 1L al puerto de salida *ena_jpeg* del módulo de control; esta señal sirve para deshabilitar el *reset* del módulo de compresión JPEG.

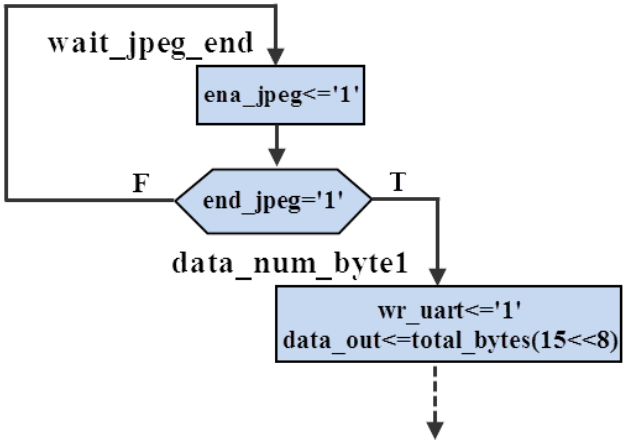


Figura 2- 15. Estados para la habilitación del proceso de compresión

El segundo proceso realizado en el estado *wait_jpeg_end*, es monitorear el fin del procedimiento de compresión realizado por el módulo de compresión JPEG, esto se logra a través del puerto de entrada *end_jpeg*, el cual está conectado a un puerto de salida del módulo de compresión que sirve como bandera para indicar que el último byte de los datos de la imagen comprimida ha sido creado. Cuando se detecta un 1L en este puerto (valor activo del puerto), se procede a la transición al estado *data_num_byte1*.

En la Figura 2-16 se indican los resultados de la simulación para esta tarea del módulo de control.

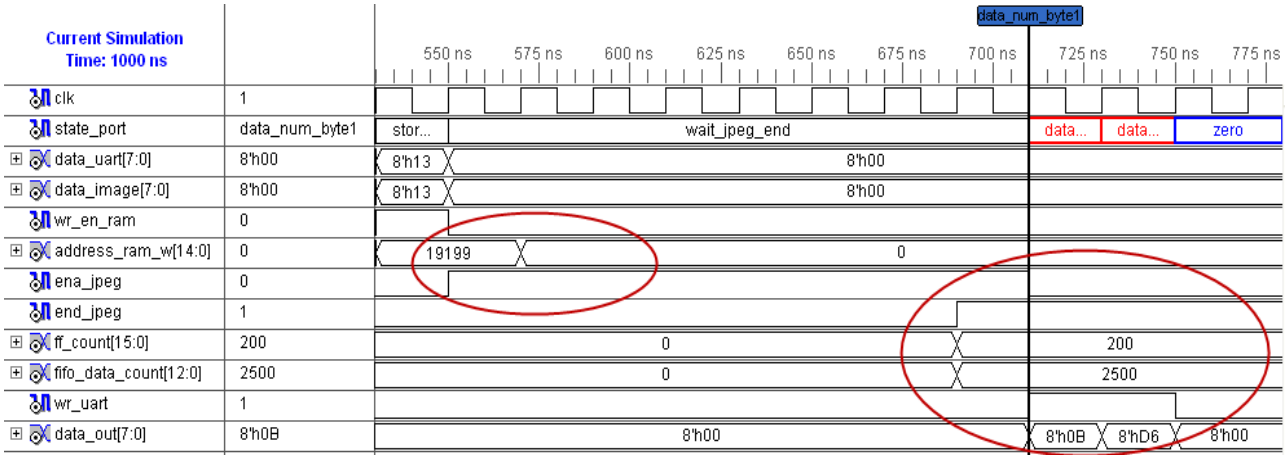


Figura 2- 16. Resultados de la simulación para la tarea de habilitación del módulo de compresión JPEG

Como se observa en la Figura 2-16, la descripción VHDL de la máquina de estados del módulo de control, cumple con el comportamiento planificado en el ASM de la Figura 2-15 que se describió anteriormente.

2.4.1.5 Envío de información previa para la transmisión del *stream* de datos hacia la PC

En los dos estados que efectúan esta tarea (*data_num_byte1* y *data_num_byte2*), se envía la información del número total de bytes que posee el archivo de la imagen generada. El número total de bytes del archivo está en función de los siguientes parámetros:

- El número de bytes de la cabecera JFIF del archivo, el cual para una imagen monocromática, con tablas de cuantización de un byte y las tablas de Huffman definidas en el estándar JPEG, posee 328 bytes.
- El número de bytes generados para los datos comprimidos de la imagen monocromática, el cual es un número variable que depende de la tasa de compresión seleccionada.

- El número de bytes con el valor x00, que se añaden a los bytes de datos, cuando se encuentra un byte con valor xFF (condición discutida en el Capítulo 1).
- Los dos bytes del marcador EOI (xFF y xD9).

Estos valores son sumados en una señal interna de 16 bits, en el módulo de control, como se indica a continuación:

```
total_bytes <= ("000" & fifo_data_count) + ("00000000" & jfif_hearder_bytes) + ff_count + "0000000000000010";
```

El puerto de entrada *fifo_data_count* corresponde al número de bytes de datos de la imagen comprimida, el puerto de entrada *ff_count*, corresponde al número de bytes x00 añadidos a los bytes de datos. Se definió la constante *jfif_header_bytes* para especificar los 328 bytes de la cabecera de archivo de la imagen. El valor “0000000000000010” definido en la expresión anterior corresponde a los dos bytes del marcador EOI.

La señal *total_bytes* es transmitida en dos bytes a través del sistema UART. En el estado *data_num_byte1* se transmite el bytes más significativo de esta señal, y en el estado *data_num_byte2* el menos significativo, luego de lo cual se pasa al estado inicial *zero*, como lo muestra la Figura 2-17, en espera de recibir la orden desde la interfaz gráfica de control, de transmitir el *stream* de datos del archivo de imagen generado. En el programa de la interfaz gráfica de control, se reciben estos dos bytes y se reconstruye el número total de bytes del archivo de imagen generado por el módulo de desarrollo; con este número se especifica al objeto serial de la interfaz gráfica de control, cuántos bytes debe recibir, después de enviar la orden de transmisión del *stream* de datos, a través del carácter ASCII ‘R’.

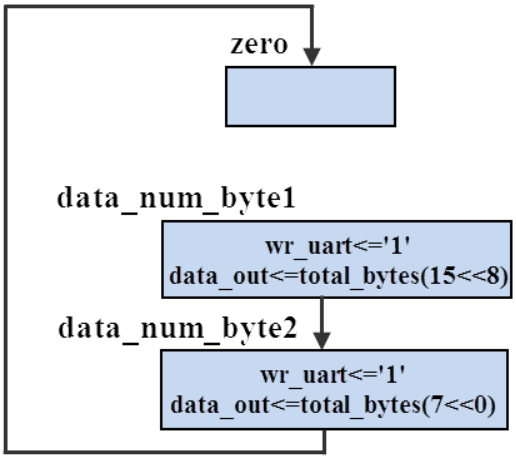


Figura 2- 17. Estados para el envío del número de bytes del archivo de imagen

En la Figura 2-18 se indica los resultados de la simulación para esta tarea del módulo de control.

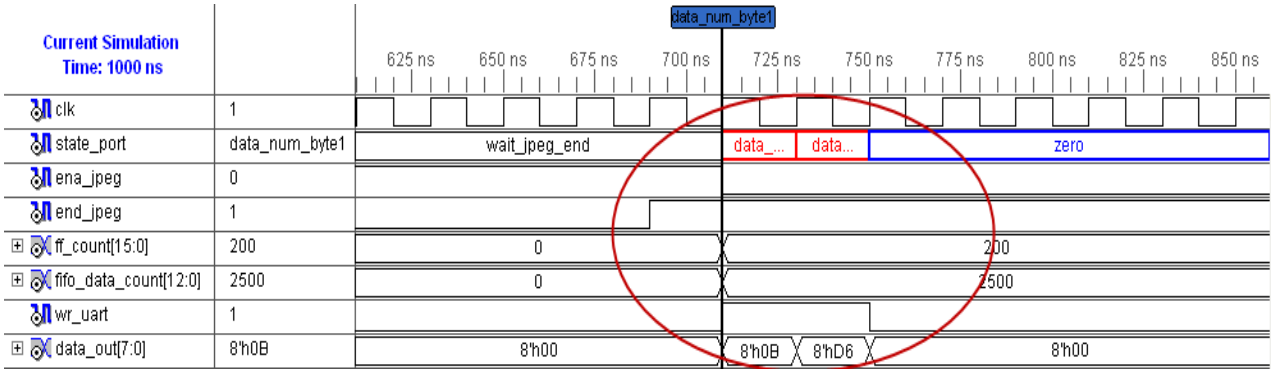


Figura 2- 18. Resultados de la simulación para la tarea de envío de información previa para la transmisión del *stream* de datos del módulo de control

Como se observa en la simulación de la Figura 2-18, la descripción VHDL del módulo de control cumple con el comportamiento planteado en el diagrama ASM de la Figura 2-17, correspondiente a la acción de envío de información previa para la transmisión del *stream* de datos hacia la PC.

2.4.1.6 Transmisión de la cabecera JFIF

Después de recibir el número de bytes que contiene el archivo de la imagen generada en el módulo de desarrollo, el programa de la interfaz gráfica de control envía al módulo de desarrollo, la orden de transmisión del *stream* de bytes del archivo de imagen; dicha transmisión se realiza en dos partes: la transmisión de la cabecera de archivo JFIF, y la transmisión de los datos de la imagen monocromática comprimida. En la primera parte de la transmisión están involucrados los estados que se muestran en la Figura 2-19. En el estado *jfif_header_ena* se procede a activar la señal de habilitación de lectura de la memoria ROM, que contiene los bytes con la información de los marcadores que posee la cabecera de archivo con formato JFIF, esto se logra asignando el valor 1L al puerto de salida *ena_rom*. Después de esto, se realiza la transición al estado *jfif_header_tx*, donde los datos provenientes de la memoria ROM son transmitidos a través del sistema UART; para esto, los bytes de datos de la memoria ROM, que posee la cabecera JFIF, ingresan al módulo de control a través del puerto de entrada *data_rom*, y son asignados al puerto de salida *data_out*, para su transmisión. Al mismo tiempo, en este estado se vuelve a activar la señal de habilitación de escritura del FIFO de transmisión del sistema UART, *wr_uart*. En el estado *jfif_header_tx*, se deben transmitir los 328 bytes de datos que posee la cabecera de archivo, para verificar esta condición, se monitorea el bus de direcciones de la memoria ROM, a través del puerto de entrada *address_rom*. Cuando se termina de transmitir toda la cabecera de archivo, se procede a la transición al estado *ena_compressed_data* para comenzar la transmisión de los datos de la imagen comprimida.

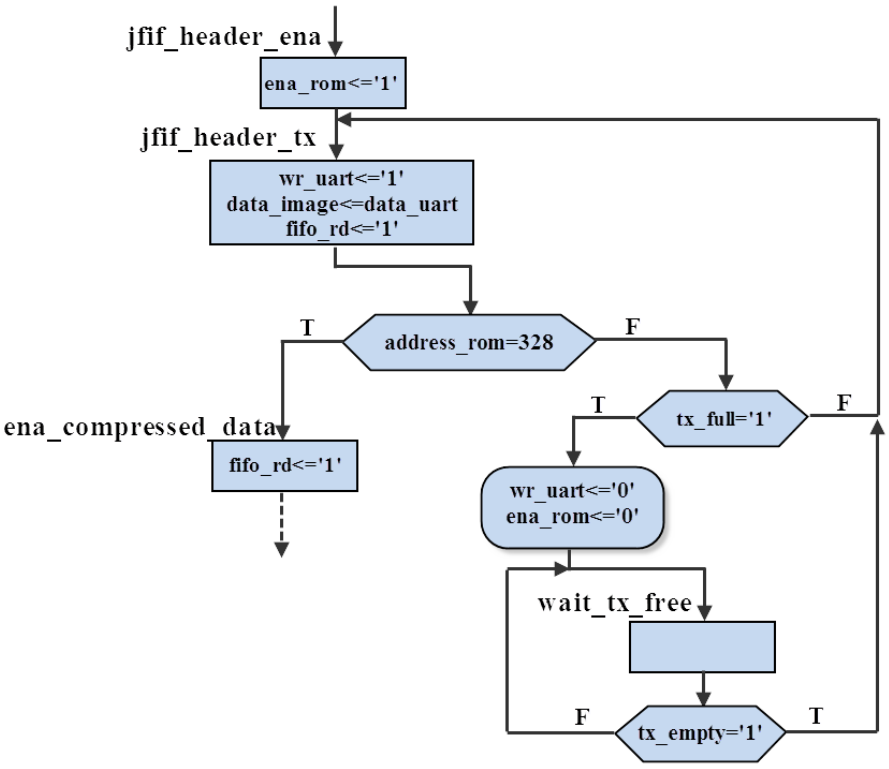


Figura 2- 19. Estados para la transmisión de la cabecera del archivo JFIF

En caso de que el FIFO de transmisión se llene, se añadió un estado auxiliar, denominado *wait_tx_free*, cuya transición se hace siempre que, estando en el estado *jfif_header_tx*, se detecte que el puerto de entrada *tx_full* está activo (con valor 1L). En este estado se permanece hasta que el puerto de entrada *tx_empty* esté en su valor activo, indicando que el FIFO de transmisión se encuentra vacío nuevamente, luego de lo cual se hace la transición de nuevo al estado *jfif_header_tx* para terminar la transmisión de la cabecera del archivo JFIF.

En la Figura 2-20 se indican los resultados de la simulación para esta tarea del módulo de control.

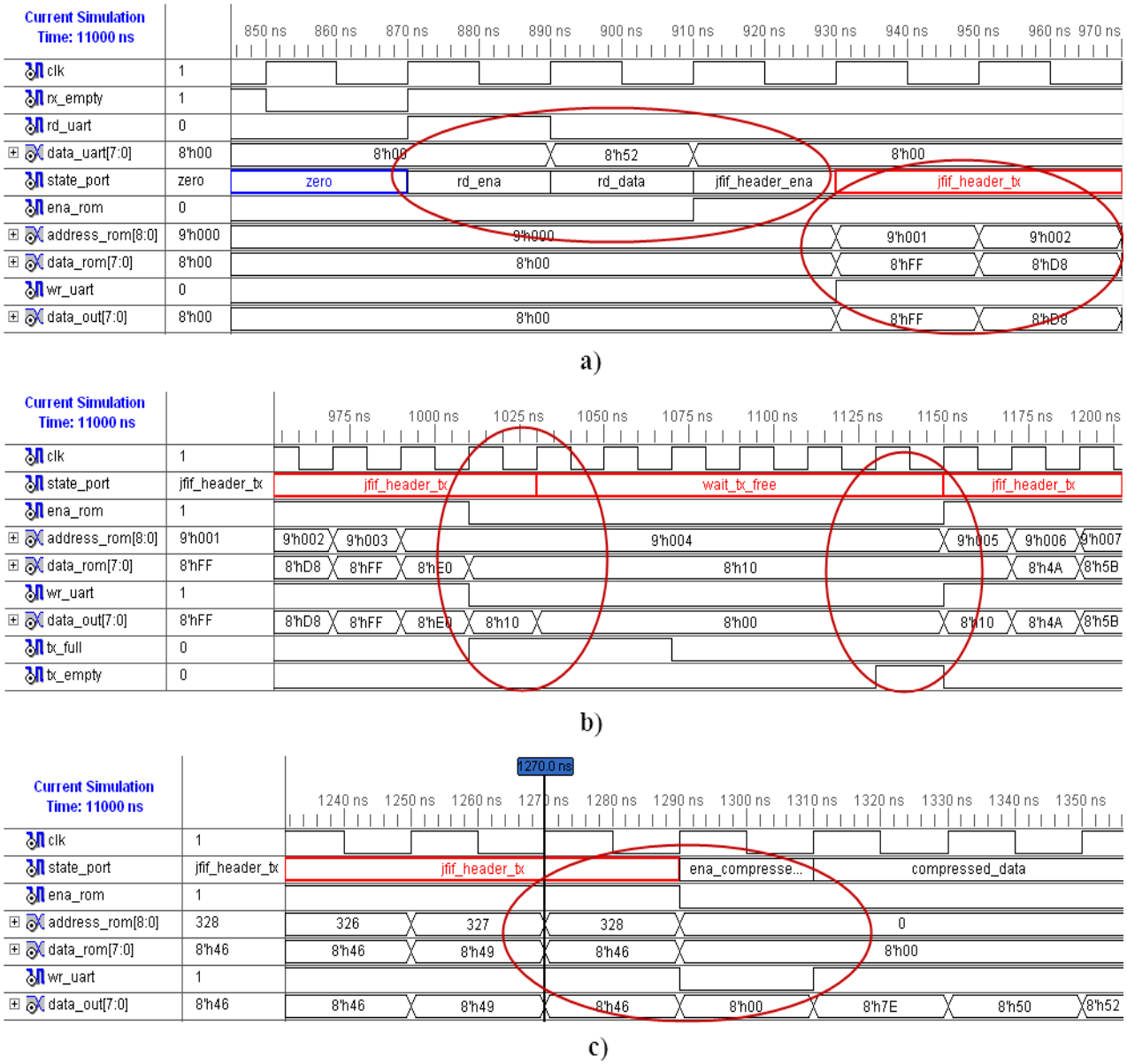


Figura 2- 20. Resultados de la simulación para la tarea de transmisión de la cabecera JFIF: a) habilitación de la transmisión de la cabecera JFIF b) procedimiento para la condición lleno en el FIFO de transmisión c) finalización del proceso de transmisión de la cabecera JFIF

Como se observa en la Figura 2-20, la descripción VHDL del módulo de control responde correctamente al comportamiento planteado en el diagrama ASM de la Figura 2-19 que corresponde a la acción de transmisión de la cabecera de archivo JFIF

2.4.1.7 Envío de los datos de la imagen comprimida

Cuando se termina de transmitir los bytes de datos de la cabecera de archivo, lo siguiente a transmitirse son los bytes de datos de la imagen comprimida propiamente dichos, los cuales se encuentran almacenados en un FIFO que es parte del módulo de compresión, como más adelante se detallará. La transmisión de los datos comprimidos empieza en el estado *ena_compressed_data*, como se muestra en la Figura 2-21. En este estado se activa la señal de habilitación de lectura del FIFO que almacena los bytes de datos comprimidos, a través del puerto de salida *rd_fifo*. Una vez activada la señal de habilitación de lectura del FIFO, se hace la transición al estado *compressed_data*.

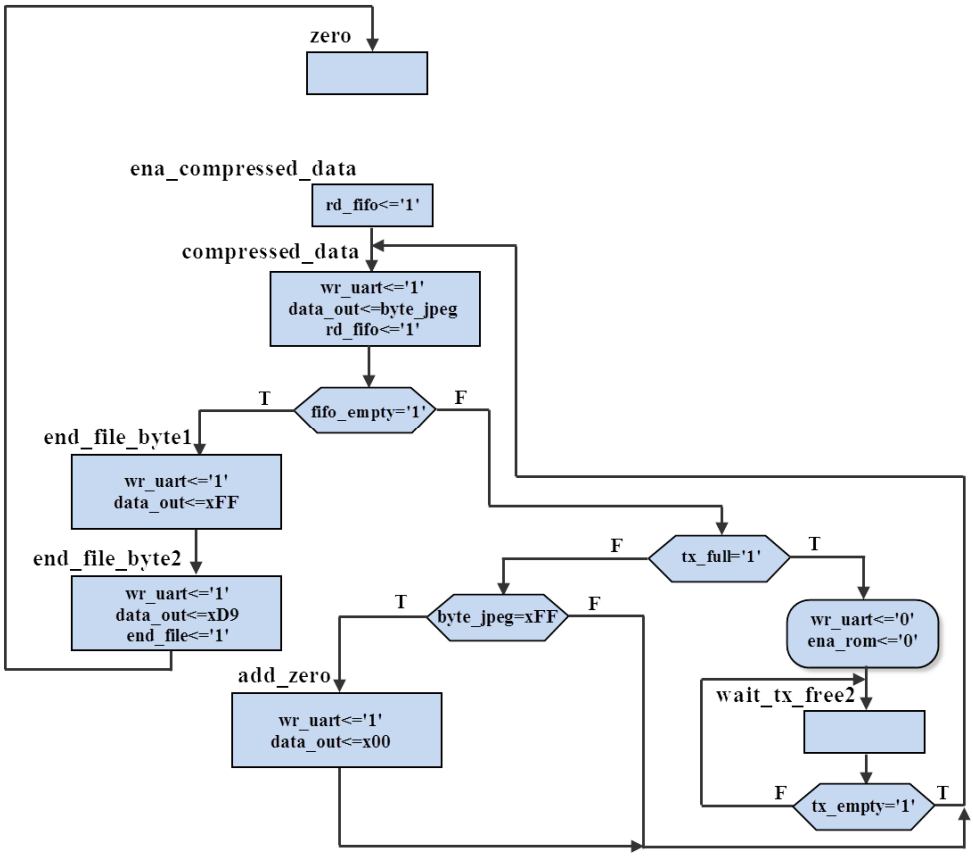


Figura 2- 21. Estados para la transmisión de los datos de la imagen comprimida

En el estado *compressed_data* se transmiten los bytes de la imagen comprimida, a través del sistema UART; los datos provenientes del FIFO, que contienen los datos de la imagen comprimida, ingresan al módulo de control a través del puerto

de entrada *byte_jpeg*, y su valor es asignado al puerto de salida *data_out.* Se permanece en este estado hasta que todos los bytes de datos de la imagen comprimida se transmitan; esta condición se verifica monitoreando el puerto de entrada *fifo_empty*, el cual indica si el FIFO que posee los bytes de la imagen comprimida, está vacío. Una vez que el FIFO esté vacío, se procede a la transición de los últimos dos estados *end_file_byte1* y *end_file_byte2*, en donde se transmiten a través del sistema UART los dos últimos bytes del archivo de imagen, correspondientes al marcador EOI, a saber xFF y xD9. En el estado *end_file_byte2*, también se activa una bandera que indica el final del proceso de transmisión del archivo de imagen, denominada *end_file.*

En el estado *compressed_data* también se realiza el control sobre los bytes con valor xFF, que puedan encontrarse en los datos de la imagen comprimida, por las razones que fueron discutidas en el Capítulo 1. Cada vez que se detecta que un byte de dato proveniente del FIFO posee el valor xFF, se hace la transición a un estado auxiliar, denominado *add_zero*, en donde se transmite a través del sistema UART, un byte con el valor x00; después de transmitir ese byte, se vuelve al estado *compressed_data* para seguir con la transmisión de los datos de la imagen comprimida, hasta que se detecte otro byte con el valor xFF.

Cuando se transmiten los bytes de datos de la imagen comprimida a través del sistema UART, éstos pueden llenar el FIFO de transmisión, dependiendo de la tasa de compresión alcanzada; cuando se indica a través del puerto de entrada *tx_full*, que el FIFO de transmisión está lleno, se hace la transición al estado *wait_tx_free2*. En este estado se permanece hasta que el FIFO de transmisión vuelva a estar vacío. Una vez que se detecta que el FIFO de transmisión está vacío, a través del puerto de entrada *tx_empty*, se hace la transición al estado *compressed_data* para continuar con la transmisión de los datos de la imagen comprimida.

En la Figura 2-22 se indica el resultado de la simulación para la tarea de la transmisión de los datos de la imagen comprimida, en el módulo de control. En la Figura 2-23 se muestra la simulación del proceso de finalización de la transmisión de los datos de la imagen comprimida.

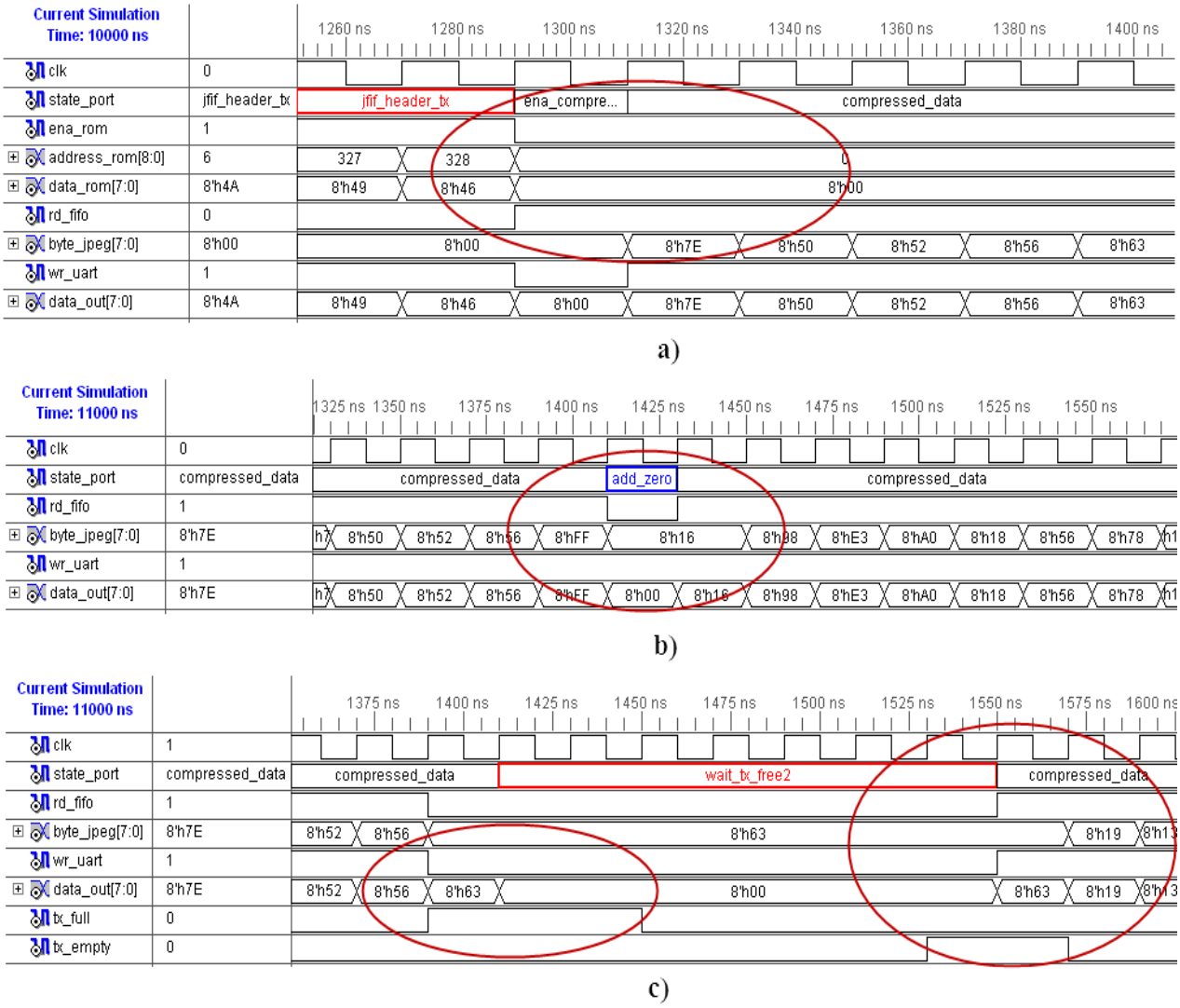


Figura 2- 22. Resultado de la simulación para la tarea de transmisión de los datos de la imagen comprimida: a) transmisión en condiciones normales b) detección de un byte con valor xFF c) procedimiento para la condición *full* en el FIFO de transmisión

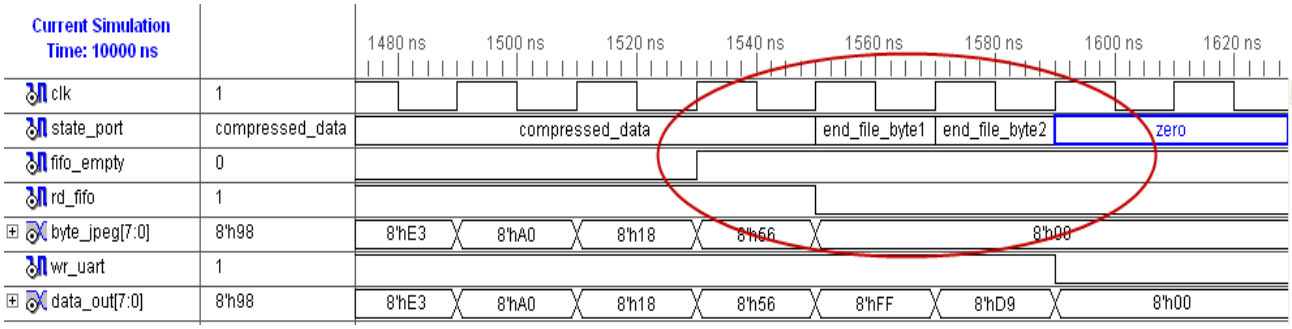
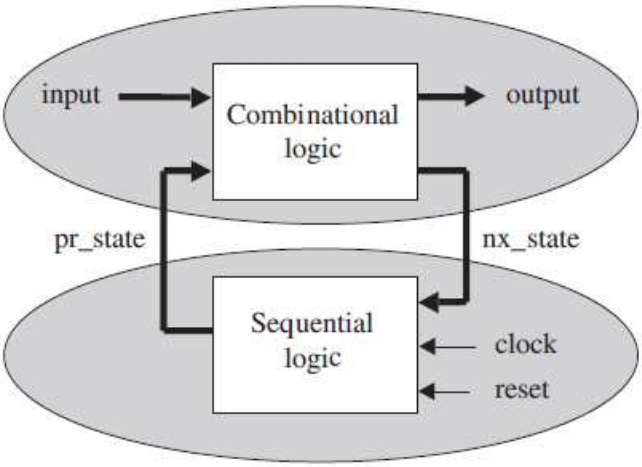


Figura 2- 23. Finalización de la transmisión de los datos de la imagen comprimida

Como se observa en la simulación de la Figura 2-22 y de la Figura 2-23, la descripción VHDL del módulo de control cumple con el comportamiento planteado en el diagrama ASM de la Figura 2-21, correspondiente a la acción de transmisión de los datos de la imagen comprimida.

Cabe mencionar que los datos que ingresan al módulo de control no son procesados por éste. Los datos que ingresan al módulo de control sirven para monitorear el estado de una determinada acción que se realiza en el módulo de desarrollo, y para enrutar datos de entrada a través de puertos dedicados, conectados a los puertos de entrada de los módulos del sistema.

Como se mencionó anteriormente, la descripción VHDL del módulo de control corresponde a una máquina de estados. La estructura más básica de una máquina de estados descrita en VHDL, consta de dos secciones: una sección de lógica secuencial y una sección de lógica combinacional, como se ilustra en la Figura 2-24.



Fuente [4]

Figura 2- 24. Estructura básica de una máquina de estados descrita en VHDL

La sección de lógica secuencial se encarga de sincronizar los cambios de estado con una señal de reloj. La sección de lógica combinacional se encarga de asignar los valores a las salidas de la máquina de estados y establecer el siguiente estado, en función del estado actual (máquina de Moore) o de los valores del estado actual y las señales de entrada (máquina de Mealy). En la Figura 2-25 se indica la plantilla VHDL que se ha utilizado en el proyecto, para describir en VHDL una máquina de estados finitos.

El código de la descripción VHDL de la máquina de estados del módulo de control se encuentra debidamente comentado en el Anexo 11.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-------------------------------------------------------
ENTITY <entity_name> IS
   PORT ( input: IN <data_type>;
          reset, clock: IN STD_LOGIC;
          output: OUT <data_type>);
END <entity_name>;
-------------------------------------------------------
ARCHITECTURE <arch_name> OF <entity_name> IS
   TYPE state IS (state0, state1, state2, state3, ...);
   SIGNAL pr_state, nx_state: state;
BEGIN
--------------------Sección de lógica secuencial--------------------
    PROCESS (reset, clock)
    BEGIN
       IF (reset='1') THEN
          pr_state <= state0;
       ELSIF (clock'EVENT AND clock='1') THEN
          pr_state <= nx_state;
       END IF;
    END PROCESS;
--------------------Sección de lógica combinacional---------------
    PROCESS (input, pr_state)
    BEGIN
       CASE pr_state IS
          WHEN state0 =>
             IF (input = ...) THEN
                output <= <value>;
                nx_state <= state1;
             ELSE ...
             END IF;
          WHEN state1 =>
             IF (input = ...) THEN
                output <= <value>;
                nx_state <= state2;
             ELSE ...
             END IF;
          WHEN state2 =>
             IF (input = ...) THEN
                output <= <value>;
                nx_state <= state3;
             ELSE ...
             END IF;
          ...
       END CASE;
    END PROCESS;
END <arch_name>;
```

Fuente [4]

Figura 2- 25. Plantilla VHDL para describir una máquina de estados

2.5 MÓDULO DE COMPRESIÓN

En este módulo se sintetiza a nivel de hardware el esquema de codificación *Baseline* del estándar JPEG, para una imagen monocromática de 160x120 pixeles. Como se mencionó en el Capitulo 1, el esquema de codificación *Baseline* del estándar JPEG, está constituido por cuatro bloques funcionales: Transformada Discreta del Coseno, Cuantización, Exploración en zig-zag y la Codificación de la Entropía de Huffman. Adicionalmente, en este módulo se ha diseñado y sintetizado un generador de bytes, el cual agrupa los códigos de longitud variable generados en bytes de datos; también se ha añadido un FIFO para almacenar los bytes de datos generados. En la Figura 2-26 se ilustra el diagrama de bloques del módulo de compresión.

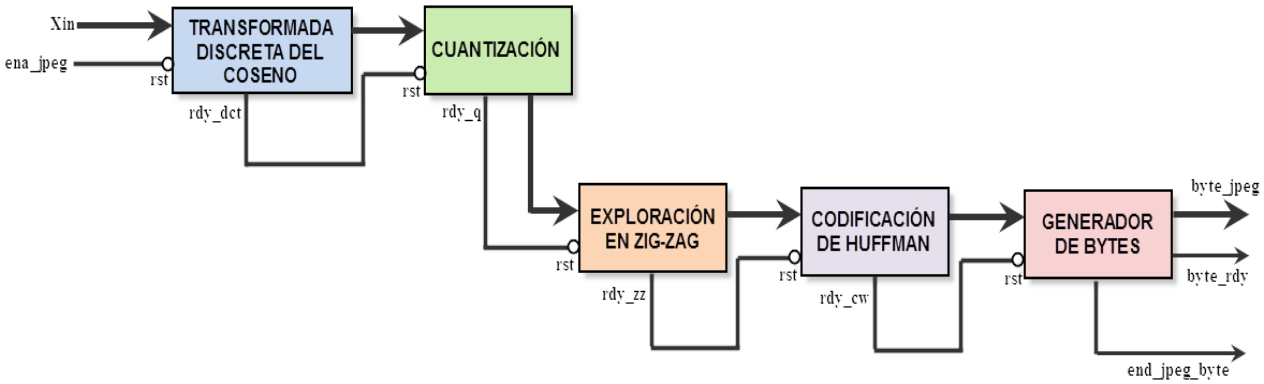


Figura 2- 26. Diagrama de bloques del módulo de compresión

Una vez que los datos de la imagen monocromática de 160x120 pixeles, son transmitidos y almacenamos en el buffer temporal, el módulo de control habilita el módulo de compresión, como ya se mencionó en el apartado anterior. La señal de habilitación, e*na_jpeg*, sirve internamente en el módulo de compresión, como señal de *reset* para el primer bloque funcional, es decir la DCT. Ya que la señal de *reset* en el bloque DCT es activa en 1L, se requiere un inversor en la señal *ena_jpeg* antes de ingresar al módulo de compresión, tal como se muestra en la Figura 2-26. En la salida de cada bloque funcional se tiene dos clases de puertos primarios; un puerto (o puertos) contiene los datos procesados del bloque en cuestión, y el otro es una bandera que indica cuándo el dato de salida del otro puerto es válido, dicho de otro manera, cuándo está listo (*ready-rdy*) el dato

procesado. Esta bandera toma las siguientes denominaciones dependiendo del bloque funcional:

rdy_dct en el bloque de la transformada discreta del coseno.
rdy_q en el bloque de cuantización.
rdy_zz en el bloque de exploración en zig-zag.
rdy_cw en el bloque de codificación de la entropía.
byte_rdy en el bloque generador de bytes.

Estas señales de bandera, invertidas, sirven como *reset* del siguiente bloque funcional en cada etapa del proceso de compresión, como se indica en la Figura 2-26. Esto produce que se tenga en cada bloque un *reset* local; tener en cada bloque un *reset* local en lugar de uno global (para todos los bloques) permite mejorar el diseño en términos de recursos lógicos utilizados, reduciendo el número de recursos de interconexión y LUTs [5].

En la Figura 2-27 se ilustra una representación de la entidad del módulo de compresión, y en la Tabla 2-2 se indican la dirección, la longitud en bits y el valor activo de los puertos que posee la entidad VHDL del módulo de compresión.

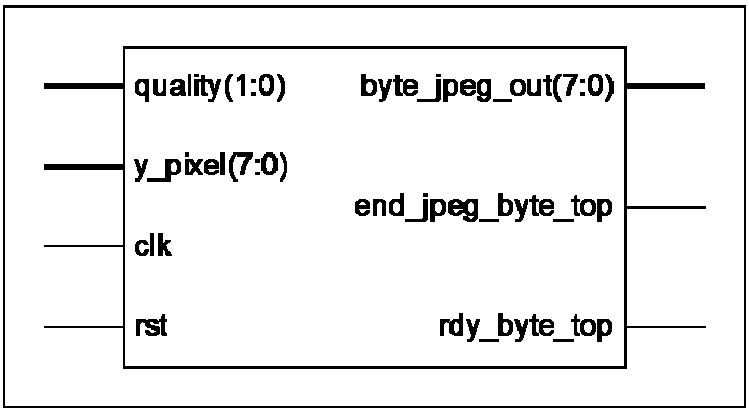


Figura 2- 27. Representación de la entidad VHDL del módulo de compresión

Tabla 2- 2. Puertos de la entidad VHDL del módulo de compresión

Puerto	Dirección	Longitud en bits	Valor activo	Descripción
clk	Entrada	N/A	N/A	Señal de reloj.
rst	Entrada	1	1L	Señal de *reset.*
y_pixel	Entrada	8	N/A	Byte de dato de la componente de luminancia de la imagen sin comprimir.
quality	Entrada	2	N/A	Selector del factor de calidad de la imagen comprimida.
byte_jpeg_out	Salida	8	N/A	Puerto de salida con los bytes de datos de la imagen comprimida.
rdy_byte_top	Salida	1	1L	Indica cuándo se tiene en el puerto *byte_jpeg_out* un byte de dato válido.
end_jpeg_byte_top	Salida	1	1L	Indica si el byte presente en el puerto *byte_jpeg_out* es el último del *stream*.

A continuación se describe la estructura de los bloques funcionales del módulo de compresión.

2.5.1 TRANSFORMADA DISCRETA DEL COSENO

Como se mencionó en el Capítulo 1, existen procedimientos para desarrollar la DCT-2D, reduciendo el número de operaciones a ejecutar. Con estos procedimientos se puede deducir la implementación de un sistema digital que realice esta transformada, mediante la correspondencia directa de algunos procedimientos del algoritmo con los elementos de hardware de un FPGA, así como su facilidad de descripción mediante un lenguaje de descripción de hardware, como el VHDL.

2.5.1.1 Descripción del Algoritmo

El procedimiento que se escogió, por ser el más simple en su implementación, es el que se discutió en el Capítulo 1, que se basa en la transformada unidimensional

del coseno y consiste en aplicar la transformada unidimensional a las filas del bloque de la imagen, y luego sobre los resultados obtenidos, volver a aplicar la transformada unidimensional sobre las columnas de éste.

La transformada DCT-2D, definida por:

$$F(u,v) = \frac{1}{4}C(u)C(v)\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)Cos\left[\frac{\pi(2x+1)u}{16}\right]Cos\left[\frac{\pi(2y+1)u}{16}\right]$$

se puede expresar en forma matricial, como se indicó en el capitulo anterior, y a continuación se muestra:

$$Y = CXC^T$$

Donde *X* representa el bloque de 8x8 pixeles, *Y* los coeficientes DCT-2D, y *C* es la matriz de transformación que está dada por:

$$C = \begin{bmatrix} 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 \\ 0.4904 & 0.4157 & 0.2778 & 0.0975 & -0.0975 & -0.2778 & -0.4157 & -0.4904 \\ 0.4619 & 0.1913 & -0.1913 & -0.4619 & -0.4619 & -0.1913 & 0.1913 & 0.4619 \\ 0.4157 & -0.0975 & -0.4904 & -0.2778 & 0.2778 & 0.4904 & 0.0975 & -0.4157 \\ 0.3536 & -0.3536 & -0.3536 & 0.3536 & 0.3536 & -0.3536 & -0.3536 & 0.3536 \\ 0.2778 & -0.4904 & 0.0975 & 0.4157 & -0.4157 & -0.0975 & 0.4904 & -0.2778 \\ 0.1913 & -0.4619 & 0.4619 & -0.1913 & -0.1913 & 0.4619 & -0.4619 & 0.1913 \\ 0.0975 & -0.2778 & 0.4157 & -0.4904 & 0.4904 & -0.4157 & 0.2778 & -0.0975 \end{bmatrix}$$

La DCT-2D, en su definición normal para un bloque de 8x8, requiere de 4096 multiplicaciones y 4096 sumas, pero con el método basado en la DCT-1D se requieren de 1024 multiplicaciones y 1024 sumas [6].

La deducción de la arquitectura del sistema digital, para implementar el procedimiento antes descrito para la DCT-2D, ha sido tomada de [7] y a continuación se describe.

Primera Transformada discreta del coseno unidimensional (sobre las filas)

Si $Z = XC^t$, se tiene que los elementos de esta matriz se definen como se indica a continuación:

$$Z = \begin{bmatrix} X_{00} & X_{01} & X_{02} & X_{03} & X_{04} & X_{05} & X_{06} & X_{07} \\ X_{10} & X_{11} & X_{12} & X_{13} & X_{14} & X_{15} & X_{16} & X_{17} \\ X_{20} & X_{21} & X_{22} & X_{23} & X_{24} & X_{25} & X_{26} & X_{27} \\ X_{30} & X_{31} & X_{32} & X_{33} & X_{34} & X_{35} & X_{36} & X_{37} \\ X_{40} & X_{41} & X_{42} & X_{43} & X_{44} & X_{45} & X_{46} & X_{47} \\ X_{50} & X_{51} & X_{52} & X_{53} & X_{54} & X_{55} & X_{56} & X_{57} \\ X_{60} & X_{61} & X_{62} & X_{63} & X_{64} & X_{65} & X_{66} & X_{67} \\ X_{70} & X_{71} & X_{72} & X_{73} & X_{74} & X_{75} & X_{76} & X_{77} \end{bmatrix} \times$$

$$\begin{bmatrix} 0.3536 & 0.4904 & 0.4619 & 0.4157 & 0.3536 & 0.2778 & 0.1913 & 0.0975 \\ 0.3536 & 0.4157 & 0.1913 & -0.0975 & -0.3536 & -0.4904 & -0.4619 & -0.2778 \\ 0.3536 & 0.2778 & -0.1913 & -0.4904 & -0.3536 & 0.0975 & 0.4619 & 0.4157 \\ 0.3536 & 0.0975 & -0.4619 & -0.2778 & 0.3536 & 0.4157 & -0.1913 & -0.4904 \\ 0.3536 & -0.0975 & -0.4619 & 0.2778 & 0.3536 & -0.4157 & -0.1913 & 0.4904 \\ 0.3536 & -0.2778 & -0.1913 & 0.4904 & -0.3536 & -0.0975 & 0.4619 & -0.4157 \\ 0.3536 & -0.4157 & 0.1913 & 0.0975 & -0.3536 & 0.4904 & -0.4619 & 0.2778 \\ 0.3536 & -0.4904 & 0.4619 & -0.4157 & 0.3536 & -0.2778 & 0.1913 & -0.0975 \end{bmatrix}$$

Para la primera fila de la matriz Z, mediante algunas agrupaciones de términos, se obtiene lo siguiente:

$$\begin{aligned}
Z_{00} &= 0.3536(X_{00} + X_{01} + X_{02} + X_{03} + X_{04} + X_{05} + X_{06} + X_{07}) \\
Z_{01} &= 0.4904(X_{00} - X_{07}) + 0.4157(X_{01} - X_{06}) + 0.2778(X_{02} - X_{05}) + 0.0975(X_{03} - X_{04}) \\
Z_{02} &= 0.4619(X_{00} + X_{07}) + 0.1913(X_{01} + X_{06}) - 0.1913(X_{02} + X_{05}) - 0.4619(X_{03} + X_{04}) \\
Z_{03} &= 0.4157(X_{00} - X_{07}) - 0.0975(X_{01} - X_{06}) - 0.4904(X_{02} - X_{05}) - 0.2778(X_{03} - X_{04}) \\
Z_{04} &= 0.3536(X_{00} + X_{07}) - 0.3536(X_{01} + X_{06}) - 0.3536(X_{02} + X_{05}) + 0.3536(X_{03} + X_{04}) \\
Z_{05} &= 0.2778(X_{00} - X_{07}) - 0.4904(X_{01} - X_{06}) + 0.0975(X_{02} - X_{05}) + 0.4157(X_{03} - X_{04}) \\
Z_{06} &= 0.1913(X_{00} + X_{07}) - 0.4619(X_{01} + X_{06}) + 0.4619(X_{02} + X_{05}) - 0.1913(X_{03} + X_{04}) \\
Z_{07} &= 0.0975(X_{00} - X_{07}) - 0.2778(X_{01} - X_{06}) + 0.4157(X_{02} - X_{05}) - 0.4904(X_{03} - X_{04})
\end{aligned}$$

o en general, para cualquier fila mediante las mismas agrupaciones, se llega a lo siguiente:

$$Z_{k0} = 0.3536(X_{k0} + X_{k1} + X_{k2} + X_{k3} + X_{k4} + X_{k5} + X_{k6} + X_{k7})$$
$$Z_{k1} = 0.4904(X_{k0} - X_{k7}) + 0.4157(X_{k1} - X_{k6}) + 0.2778(X_{k2} - X_{k5}) + 0.0975(X_{k3} - X_{k4})$$
$$Z_{k2} = 0.4619(X_{k0} + X_{k7}) + 0.1913(X_{k1} + X_{k6}) - 0.1913(X_{k2} + X_{k5}) - 0.4619(X_{k3} + X_{k4})$$
$$Z_{k3} = 0.4157(X_{k0} - X_{k7}) - 0.0975(X_{k1} - X_{k6}) - 0.4904(X_{k2} - X_{k5}) - 0.2778(X_{k3} - X_{k4})$$
$$Z_{k4} = 0.3536(X_{k0} + X_{k7}) - 0.3536(X_{k1} + X_{k6}) - 0.3536(X_{k2} + X_{k5}) + 0.3536(X_{k3} + X_{k4})$$
$$Z_{k5} = 0.2778(X_{k0} - X_{k7}) - 0.4904(X_{k1} - X_{k6}) + 0.0975(X_{k2} - X_{k5}) + 0.4157(X_{k3} - X_{k4})$$
$$Z_{k6} = 0.1913(X_{k0} + X_{k7}) - 0.4619(X_{k1} + X_{k6}) + 0.4619(X_{k2} + X_{k5}) - 0.1913(X_{k3} + X_{k4})$$
$$Z_{k7} = 0.0975(X_{k0} - X_{k7}) - 0.2778(X_{k1} - X_{k6}) + 0.4157(X_{k2} - X_{k5}) - 0.4904(X_{k3} - X_{k4})$$

donde k = 0 a 7.

Mediante este procedimiento se obtiene la transformada discreta del coseno unidimensional de las filas de la matriz de pixeles de 8x8 (bloque de 8x8 pixeles). En el procedimiento anterior, existen algunas peculiaridades que sirven para la deducción de la arquitectura del sistema digital, que implementa el primer bloque de la DCT-1D, por ejemplo:

- Para hallar los elementos de cada fila de la matriz Z, siempre se suman (o restan) los mismos pares de pixeles, a saber: X_{k0} con X_{k7}, X_{k1} con X_{k6}, X_{k2} con X_{k5}, y X_{k4} con X_{k3}.
- Dependiendo de la columna a la que pertenezca cada elemento, de cualquier fila de la matriz Z, la suma (o resta) de los pares de elementos de la matriz de pixeles, se multiplica siempre por un conjunto determinado de coeficientes de la matriz de transformación C, a saber:

    - En primera columna de la matriz Z:
        - 0.3536 para todos los pares
    - En la segunda columna de la matriz Z:
        - 0.4904 para el par X_{k0} y X_{k7}
        - 0.4157 para el par X_{k1} y X_{k6}
        - 0.2778 para el par X_{k2} y X_{k5}
        - 0.0975 para el par X_{k3} y X_{k4}
    - En la tercera columna de la matriz Z:
        - 0.4619 para el par X_{k0} y X_{k7}

- 0.1913 para el par X_{k1} y X_{k6}
- -0.1913 para el par X_{k2} y X_{k5}
- -04619 para el par X_{k3} y X_{k4}

o En la cuarta columna de la matriz Z:

- 0.4157 para el par X_{k0} y X_{k7}
- -0.0975 para el par X_{k1} y X_{k6}
- -0.4904 para el par X_{k2} y X_{k5}
- -0.2778 para el par X_{k3} y X_{k4}

o En la quinta columna de la matriz Z:

- 0.3536 para el par X_{k0} y X_{k7}
- -0.3536 para el par X_{k1} y X_{k6}
- -0.3536 para el par X_{k2} y X_{k5}
- 0.3536 para el par X_{k3} y X_{k4}

o En la sexta columna de la matriz Z:

- 0.2778 para el par X_{k0} y X_{k7}
- -0.4904 para el par X_{k1} y X_{k6}
- 0.0975 para el par X_{k2} y X_{k5}
- 0.4157 para el par X_{k3} y X_{k4}

o En la séptima columna de la matriz Z:

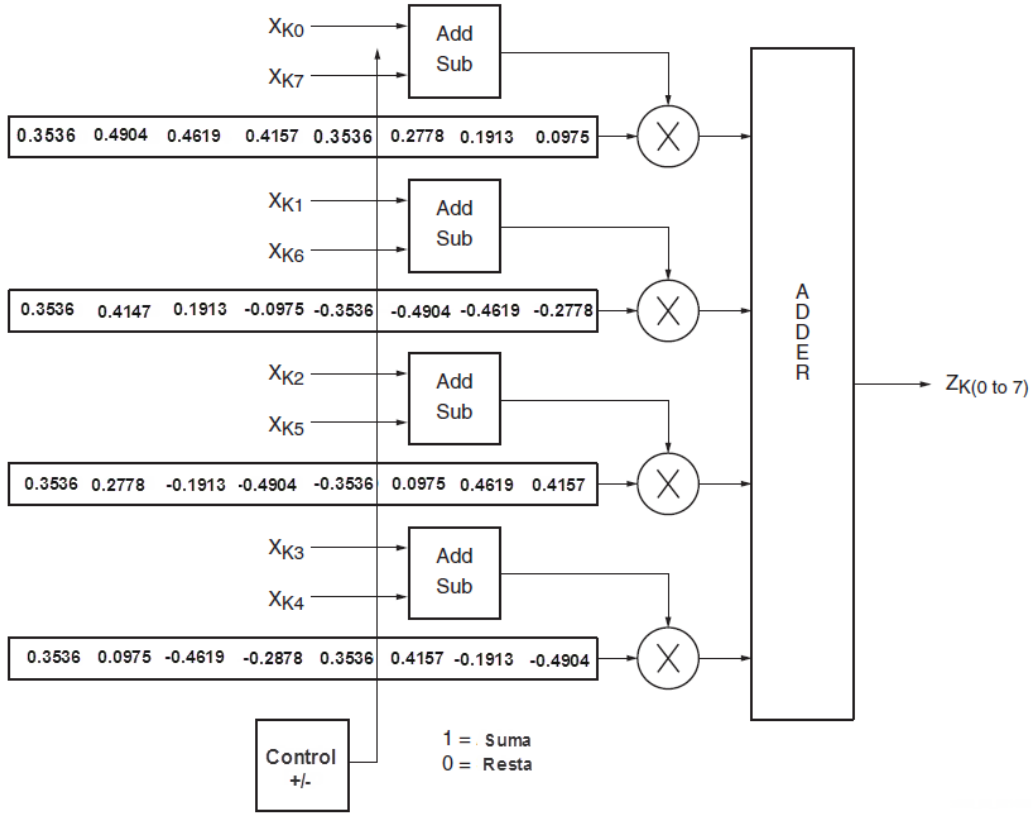
- 0.1913 para el par X_{k0} y X_{k7}
- -0.4619 para el par X_{k1} y X_{k6}
- 0.4619 para el par X_{k2} y X_{k5}
- -0.1913 para el par X_{k3} y X_{k4}

o En la octava columna de la matriz Z:

- 0.0975 para el par X_{k0} y X_{k7}
- -0.2778 para el par X_{k1} y X_{k6}
- 0.4157 para el par X_{k2} y X_{k5}
- -0.4904 para el par X_{k3} y X_{k4}

Estas características hacen posible la implementación de un sistema digital para la primera transformada discreta del coseno (DCT-1D), cuyo diagrama de bloques se muestra en la Figura 2-28. La señal de control en la figura, es la que indica la operación (suma o resta) a realizar sobre los pares de pixeles anteriormente

mencionados. Una memoria ROM almacena el conjunto de coeficientes que multiplica a la suma (o resta) de los pares de pixeles.



Fuente [7]

Figura 2- 28. Diagrama de bloques de la implementación de la DCT-1D

Con el sistema digital de la Figura 2-28, se puede obtener los elementos de cada fila de la matriz Z, en ocho ciclos de reloj, y los 64 elementos de la matriz Z en 64 ciclos de reloj. La matriz Z se obtiene fila por fila, y se almacena en una memoria RAM, para su posterior procesamiento.

Segunda transformada discreta del coseno unidimensional (sobre las columnas)

Una vez que se tiene la matriz Z, se puede proseguir con el proceso, ahora multiplicando C.Z.

En el caso de la multiplicación de C.Z, se pueden hacer agrupaciones tales como en el caso de $X.C^t$, que conlleven a una estructura de un sistema como el que se mostró en la Figura 2-28. Sin embargo, las operaciones ahora se llevan a cabo en las columnas de la matriz Z, lo que conlleva a la lectura columna por columna, de la memoria RAM donde se almacena la matriz Z. Esto produce un efecto de transposición, de suerte que se cumple la propiedad de las matrices:

$$\left(C.Z\right)^t = Z^t.C^t$$

con lo cual, la matriz final de los 64 coeficientes DCT estará en orden transpuesto. Con la lectura de la matriz Z por columnas, en realidad se está haciendo la operación $Z^t.C^t$.

Si Y = C.Z se tiene que los elementos de la matriz se definen como se indica en seguida:

$$Y = \begin{bmatrix} 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 \\ 0.4904 & 0.4157 & 0.2778 & 0.0975 & -0.0975 & -0.2778 & -0.4157 & -0.4904 \\ 0.4619 & 0.1913 & -0.1913 & -0.4619 & -0.4619 & -0.1913 & 0.1913 & 0.4619 \\ 0.4157 & -0.0975 & -0.4904 & -0.2778 & 0.2778 & 0.4904 & 0.0975 & -0.4157 \\ 0.3536 & -0.3536 & -0.3536 & 0.3536 & 0.3536 & -0.3536 & -0.3536 & 0.3536 \\ 0.2778 & -0.4904 & 0.0975 & 0.4157 & -0.4157 & -0.0975 & 0.4904 & -0.2778 \\ 0.1913 & -0.4619 & 0.4619 & -0.1913 & -0.1913 & 0.4619 & -0.4619 & 0.1913 \\ 0.0975 & -0.2778 & 0.4157 & -0.4904 & 0.4904 & -0.4157 & 0.2778 & -0.0975 \end{bmatrix} \times$$

$$\begin{bmatrix} Z_{00} & Z_{01} & Z_{02} & Z_{03} & Z_{04} & Z_{05} & Z_{06} & Z_{07} \\ Z_{10} & Z_{11} & Z_{12} & Z_{13} & Z_{14} & Z_{15} & Z_{16} & Z_{17} \\ Z_{20} & Z_{21} & Z_{22} & Z_{23} & Z_{24} & Z_{25} & Z_{26} & Z_{27} \\ Z_{30} & Z_{31} & Z_{32} & Z_{33} & Z_{34} & Z_{35} & Z_{36} & Z_{37} \\ Z_{40} & Z_{41} & Z_{42} & Z_{43} & Z_{44} & Z_{45} & Z_{46} & Z_{47} \\ Z_{50} & Z_{51} & Z_{52} & Z_{53} & Z_{54} & Z_{55} & Z_{56} & Z_{57} \\ Z_{60} & Z_{61} & Z_{62} & Z_{63} & Z_{64} & Z_{65} & Z_{66} & Z_{67} \\ Z_{70} & Z_{71} & Z_{72} & Z_{73} & Z_{74} & Z_{75} & Z_{76} & Z_{77} \end{bmatrix}$$

Para la primera columna de la matriz Y, mediante algunas agrupaciones de términos se obtiene lo siguiente:

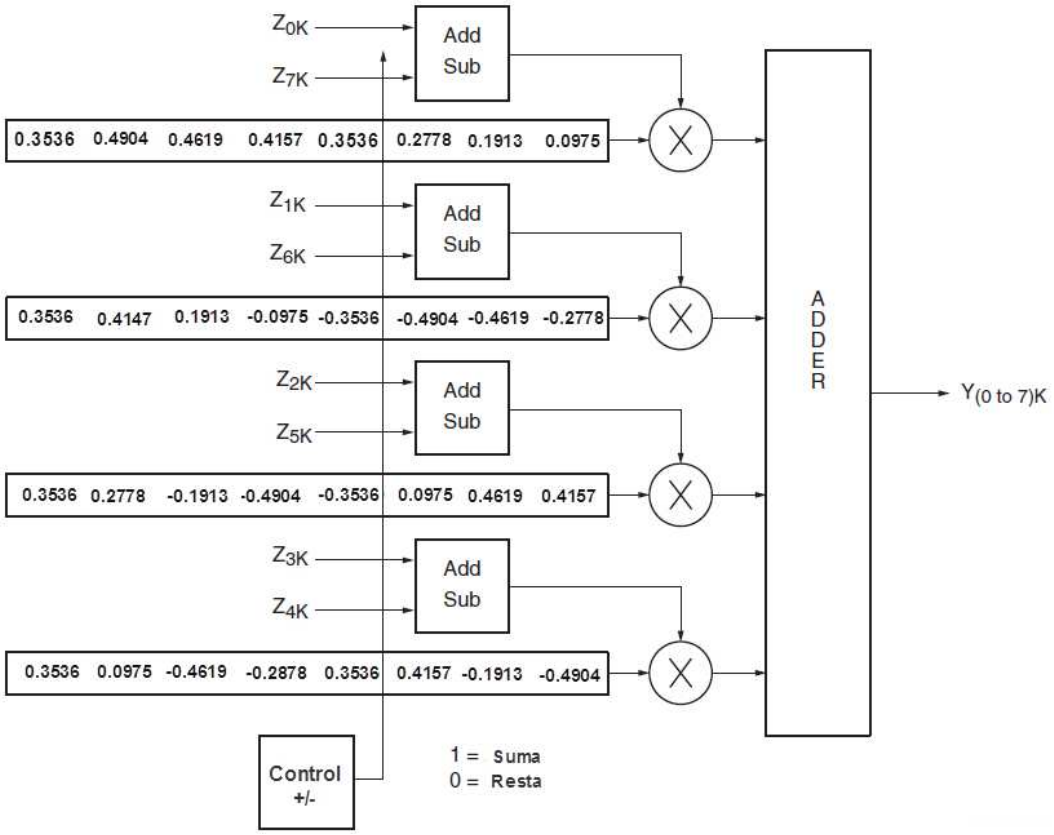
$$
\begin{aligned}
Y_{00} &= 0.3536\left(X_{00}+X_{10}+X_{20}+X_{30}+X_{40}+X_{50}+X_{60}+X_{70}\right)\\
Y_{01} &= 0.4904\left(X_{00}-X_{70}\right)+0.4157\left(X_{10}-X_{60}\right)+0.2778\left(X_{20}-X_{50}\right)+0.0975\left(X_{30}-X_{40}\right)\\
Y_{02} &= 0.4619\left(X_{00}+X_{70}\right)+0.1913\left(X_{10}+X_{60}\right)-0.1913\left(X_{20}+X_{50}\right)-0.4619\left(X_{30}+X_{40}\right)\\
Y_{03} &= 0.4157\left(X_{00}-X_{70}\right)-0.0975\left(X_{10}-X_{60}\right)-0.4904\left(X_{20}-X_{50}\right)-0.2778\left(X_{30}-X_{40}\right)\\
Y_{04} &= 0.3536\left(X_{00}+X_{70}\right)-0.3536\left(X_{10}+X_{60}\right)-0.3536\left(X_{20}+X_{50}\right)+0.3536\left(X_{30}+X_{40}\right)\\
Y_{05} &= 0.2778\left(X_{00}-X_{70}\right)-0.4904\left(X_{10}-X_{60}\right)+0.0975\left(X_{20}-X_{50}\right)+0.4157\left(X_{30}-X_{40}\right)\\
Y_{06} &= 0.1913\left(X_{00}+X_{70}\right)-0.4619\left(X_{10}+X_{60}\right)+0.4619\left(X_{20}+X_{50}\right)-0.1913\left(X_{30}+X_{40}\right)\\
Y_{07} &= 0.0975\left(X_{00}-X_{70}\right)-0.2778\left(X_{10}-X_{60}\right)+0.4157\left(X_{20}-X_{50}\right)-0.4904\left(X_{30}-X_{40}\right)
\end{aligned}
$$

o en general para cualquier columna mediante las mismas agrupaciones, se llega a lo siguiente:

$$
\begin{aligned}
Y_{0k} &= 0.3536\left(Z_{0k}+Z_{1k}+Z_{2k}+Z_{3k}+Z_{4k}+Z_{5k}+Z_{6k}+Z_{7k}\right)\\
Y_{1k} &= 0.4904\left(Z_{0k}-Z_{7k}\right)+0.4157\left(Z_{1k}-Z_{6k}\right)+0.2778\left(Z_{2k}-Z_{5k}\right)+0.0975\left(Z_{3k}-Z_{4k}\right)\\
Y_{2k} &= 0.4619\left(Z_{0k}+Z_{7k}\right)+0.1913\left(Z_{1k}+Z_{6k}\right)-0.1913\left(Z_{2k}+Z_{5k}\right)-0.4619\left(Z_{3k}+Z_{4k}\right)\\
Y_{3k} &= 0.4157\left(Z_{0k}-Z_{7k}\right)-0.0975\left(Z_{1k}-Z_{6k}\right)-0.4904\left(Z_{2k}-Z_{5k}\right)-0.2778\left(Z_{3k}-Z_{4k}\right)\\
Y_{4k} &= 0.3536\left(Z_{0k}+Z_{7k}\right)-0.3536\left(Z_{1k}+Z_{6k}\right)-0.3536\left(Z_{2k}+Z_{5k}\right)+0.3536\left(Z_{3k}+Z_{4k}\right)\\
Y_{5k} &= 0.2778\left(Z_{0k}-Z_{7k}\right)-0.4904\left(Z_{1k}-Z_{6k}\right)+0.0975\left(Z_{2k}-Z_{5k}\right)+0.4157\left(Z_{3k}-Z_{4k}\right)\\
Y_{6k} &= 0.1913\left(Z_{0k}+Z_{7k}\right)-0.4619\left(Z_{1k}+Z_{6k}\right)+0.4619\left(Z_{2k}+Z_{5k}\right)-0.1913\left(Z_{3k}+Z_{4k}\right)\\
Y_{7k} &= 0.0975\left(Z_{0k}-Z_{7k}\right)-0.2778\left(Z_{1k}-Z_{6k}\right)+0.4157\left(Z_{2k}-Z_{5k}\right)-0.4904\left(Z_{3k}-Z_{4k}\right)
\end{aligned}
$$

donde k = 0 a 7.

Como se puede observar, se obtienen las mismas características que en la multiplicación de XC^t, solo que en este caso, son las columnas de la matriz Z las que intervienen en las operaciones para obtener los elementos de la matriz Y, que es la que contiene los 64 coeficientes de la DCT-2D. Esto permite deducir el sistema digital cuyo diagrama de bloques se muestra en la Figura 2-29.



Fuente [7]

Figura 2- 29. Diagrama de bloques de la implementación de la DCT-2D

Se puede notar que el sistema digital para implementar la segunda DCT-1D (para las columnas de la matriz Z) tiene la misma estructura que el primer caso (para las filas de la matriz X), esto ayuda a reducir la dificultad en el diseño de la DCT-2D, el cual se describe a continuación.

2.5.1.2 Descripción del diseño

El código VHDL para el diseño de la entidad de la DCT-2D, ha sido tomado de [8][19] y modificado para obtener los resultados esperados según el algoritmo antes descrito. Las modificaciones consistieron en cambiar el tipo de aritmética utilizada, de complemento a 1, hacia complemento a 2, ya que como se explicó en el Capítulo 1, los multiplicadores embebidos trabajan con este tipo de complemento; los multiplicadores fueron incluidos explícitamente en el código como

[19] Esto se señaló en el plan de tesis aprobado

componentes, creados a través de *IP Cores*. También se modificó la precisión en bits de los coeficientes de la matriz de transformación de 8 a 13 bits, entre otras modificaciones de orden funcional[20].

En la Figura 2-30 se ilustra una representación de la entidad del bloque de la DCT-2D, y en la Tabla 2-3 se indican la dirección, la longitud en bits y el valor activo de los puertos que posee la entidad VHDL del bloque de la DCT-2D.

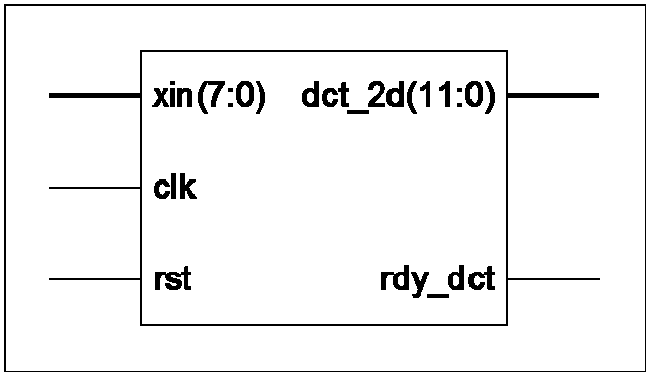


Figura 2- 30. Representación de la entidad del boque de la DCT-2D

Tabla 2- 3. Puertos de la entidad VHDL del bloque de la DCT-2D

Puerto	Dirección	Longitud en bits	Valor activo	Descripción
clk	Entrada	N/A	N/A	Señal de reloj.
rst	Entrada	1	1L	Señal de *reset.*
xin	Entrada	8	N/A	Bytes de datos de la componente de luminancia de la imagen sin comprimir.
dct_2d	Salida	12	N/A	Puerto de salida de los coeficientes de la DCT-2D.
rdy_dct	Salida	1	N/A	Indica cuándo el coeficiente presente en el puerto *dct_2d* es válido.

Todo el diseño de la DCT-2D es sincrónico, empieza cuando la señal de *reset rst*, se desactiva, con lo cual comienza el ingreso de los valores de los pixeles que conforman los bloques de 8x8, fila por fila, a través del puerto *xin*; los valores que

[20] El diseño original no proporcionó los resultados correctos de la DCT-2D, por lo que se realizó un análisis por medio de simulaciones para poder encontrar y corregir algunas secciones del código.

ingresan a través del puerto *xin* provienen de la memoria RAM que conforma el buffer temporal del sistema. A los datos que ingresan se les substrae el valor 128 para luego ser desplazados a través de un registro de desplazamiento, hasta obtener una fila del boque de 8x8, tal como lo indica el diagrama de bloques de la Figura 2-31. Ya que cada fila posee 8 elementos (pixeles), para obtener y restar el valor 128 de todos los elementos de una fila se requieren 8 ciclos de reloj.

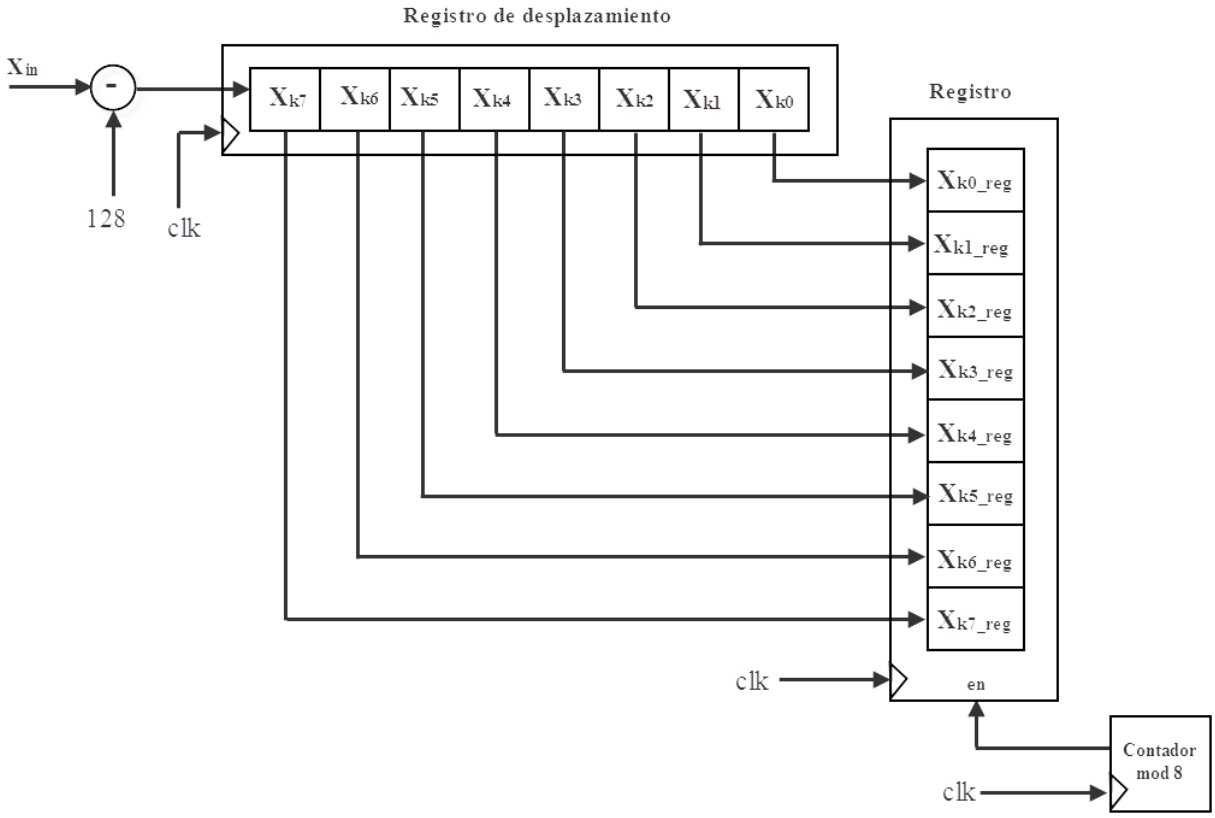


Figura 2- 31. Diagrama de bloques del proceso de adquisición de las filas del bloque de 8x8 pixeles

Después de obtener la fila del bloque de 8x8 pixeles, se debe registrar con flip-flops los elementos de la fila, como se indica en el diagrama de bloques de la Figura 2-31. El registro de la fila debe cumplir una particularidad, que es la de duplicar el bit más significativo del valor de cada pixel en la fila, esto con el propósito de evitar el *overflow*[21] en las operaciones de suma y resta que se realizan posteriormente. El registro se activa cuando un contador módulo 8,

[21] **Overflow**: o desbordamiento, sucede si al realizar una operación aritmética en binario, el bit más alto resulta conllevar un acarreo, y no hay forma de representar una cifra más para ese acarreo.

denominado *cntr8*, alcanza su valor máximo, esto con el fin de almacenar los pixeles, sólo cuando toda la fila ha ingresado a través del registro de desplazamiento.

El código VHDL, para realizar los procesos de obtención y registro de los pixeles de una fila, puede ser descrito como se indica a continuación:

- Registro de desplazamiento y substracción del valor 128:

```
process (clk)
   begin
      if clk'event AND clk = '1' then
               xk7 <= xin - "10000000";
               xk6 <= xk7;
               xk5 <= xk6;
               xk4 <= xk5;
               xk3 <= xk4;
               xk2 <= xk3;
               xk1 <= xk2;
               xk0 <= xk1;
      end if;
end process;
```

- Registro de la fila, duplicando el bits más significativo:

```
process (clk)
   begin
      if clk'event and clk = '1' then
         if (rst = '1') then
            xk0_reg <= "000000000";    xk1_reg <= "000000000";
            xk2_reg <= "000000000";    xk3_reg <= "000000000";
            xk4_reg <= "000000000";    xk5_reg <= "000000000";
            xk6_reg <= "000000000";    xk7_reg <= "000000000";
         else
            if (cntr8 = "1000") then
               xk0_reg <= xk0(7) & xk0;
               xk1_reg <= xk1(7) & xk1;
               xk2_reg <= xk2(7) & xk2;
               xk3_reg <= xk3(7) & xk3;
               xk4_reg <= xk4(7) & xk4;
               xk5_reg <= xk5(7) & xk5;
               xk6_reg <= xk6(7) & xk6;
               xk7_reg <= xk7(7) & xk7;
            else

            end if;
         end if;
      end if;
end process;
```

Cuando se tienen almacenados los elementos de la fila, como se indicó anteriormente, se comienza a realizar las operaciones de suma o resta de los pares de pixeles que se definieron en la descripción del algoritmo. Esta tarea está controlada por una señal que indica cuándo se deben sumar o restar dichos pixeles. Cuando esta señal es 1L se realiza la operación de suma, caso contrario se realiza la resta de los pares de pixeles. La implementación VHDL de este procedimiento es simplemente una señal, denominada *toggle*, que cambia su estado lógico en cada nuevo ciclo de reloj, con lo cual se alterna las operaciones (suma/resta); esto se logra haciendo la negación de la señal en cada ciclo de reloj como lo muestra el siguiente código VHDL:

```
process (clk)
begin
   if clk'event and clk = '1' then
      if (rst = '1') then
         toggle <= '0';
      else
         toggle <= not toggle;
      end if;
   end if;
end process;
```

En el noveno ciclo de reloj, las operaciones entre los pares de pixeles comienzan a ejecutarse, siendo la primera operación, la suma de éstos. Las operaciones (suma o resta) de los pixeles, también son sincrónicas (están dentro de un proceso disparado por la señal de reloj); por tanto, en el décimo ciclo de reloj se comienza a obtener los resultados de dichas operaciones. Las sumas (o restas) se van asignando a cuatro señales, a saber

- *add_sub1a*, para la suma o resta de *xk0_reg* y *xk7_reg*.
- *add_sub2a*, para la suma o resta de *xk1_reg* y *xk6_reg.*
- *add_sub3a*, para la suma o resta de *xk2_reg* y *xk5_reg.*
- *add_sub4a*, para la suma o resta de *xk3_reg* y *xk4_reg.*

Cada señal está conectada a un puerto de un multiplicador específico, y corresponde al primer operando de ese multiplicador. En el segundo puerto

(operando) de cada multiplicador, se asigna el correspondiente coeficiente que debe multiplicar la suma o resta de los pares de pixeles, como se mencionó en la descripción del algoritmo. La multiplicación de la suma (o resta) de los pares de pixeles se lleva a cabo en el décimo ciclo de reloj y los resultados se obtienen en el décimo primer ciclo de reloj.

Los coeficientes que multiplican a la suma (o resta) de los pares de pixeles, en la primera DCT-1D, son definidos mediante cuatro señales que cambian su valor cada ciclo de reloj. Estas señales son las siguientes:

- c1, que multiplica a *add_sub1a.*
- c2, que multiplica a *add_sub2a.*
- c3, que multiplica a *add_sub3a.*
- c4, que multiplica a *add_sub4a.*

En cada ciclo de reloj, las cuatro señales cambian su valor, a los cuatro conjuntos de valores que se determinaron mediante las agrupaciones de términos, en las multiplicaciones de matrices, para obtener las matrices Z e Y. En la Tabla 2-4 se muestran los valores asignados a estas señales en cada ciclo de reloj.

Para la representación de los coeficientes decimales, se usó un número de 13 bits en complemento a 2, cuyo bit más significativo representa la parte entera del coeficiente y el resto de bits, la parte fraccionaria del coeficiente, tal como se muestra en la Figura 2-32.

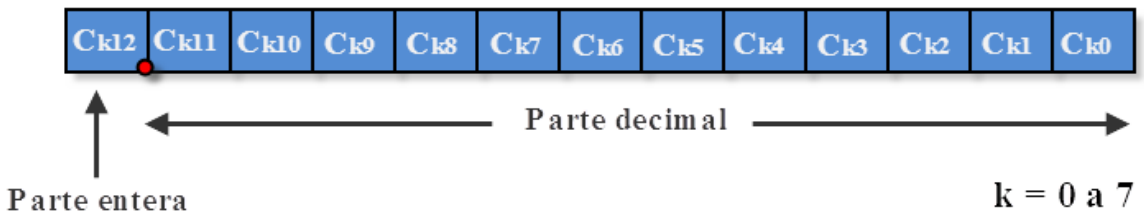


Figura 2- 32. Representación de los coeficientes mediante números de 13 bits en complemento a 2

La representación de los coeficientes mediante 12 bits en su parte fraccionaria, provoca cierta imprecisión en éstos, aunque ello, no causa una alteración significativa en los resultados obtenidos. En la Tabla 2-4 se indica la representación con un número de 13 bits en complemento a 2, para los coeficientes utilizados en el proceso de la DCT-1D, aplicada a las filas del bloque de la imagen.

Tabla 2- 4. Asignación y representación binaria de los coeficientes de la matriz de transformación en cada ciclo de reloj

Ciclo de reloj	Señal	Coeficiente	Representación con 13 bits	Equivalente en base 10
Primer ciclo	C1	0.3536	0010110101000_2	1448
	C2			
	C3			
	C4			
Segundo ciclo	C1	0.4904	0011111011001_2	2009
	C2	0.4157	0011010100111_2	1703
	C3	0.2778	0010001110010_2	1138
	C4	0.0975	0000110010000_2	400
Tercer ciclo	C1	0.4619	0011101100100_2	1892
	C2	0.1913	0001100010000_2	784
	C3	-0.1913	1110011110000_2	-784
	C4	-0.4619	1100010011100_2	-1892
Cuarto ciclo	C1	0.4157	0011010100111_2	1703
	C2	-0.0975	1111001110000_2	-400
	C3	-0.4904	1100000100111_2	-2009
	C4	-0.2778	1101110001110_2	-1138
Quinto ciclo	C1	0.3536	0010110101000_2	1448
	C2	-0.3536	1101001011000_2	-1448
	C3	-0.3536	1101001011000_2	-1448
	C4	0.3536	0010110101000_2	1448
Sexto ciclo	C1	0.2778	0010001110010_2	1138
	C2	-0.4904	1100000100111_2	-2009
	C3	0.0975	0000110010000_2	400
	C4	0.4157	0011010100111_2	1703
Séptimo ciclo	C1	0.1913	0001100010000_2	784
	C2	-0.4619	1100010011100_2	-1892
	C3	0.4619	0011101100100_2	1892
	C4	-0.1913	1110011110000_2	-784
Octavo ciclo	C1	0.0975	0000110010000_2	400
	C2	-0.2778	1101110001110_2	-1138
	C3	0.4157	0011010100111_2	1703
	C4	-0.4904	1100000100111_2	-2009

Por ejemplo, para el coeficiente 0.3536, la representación es 0010110101000_2, con lo que se obtiene los valores que se indican en la Figura 2-33:

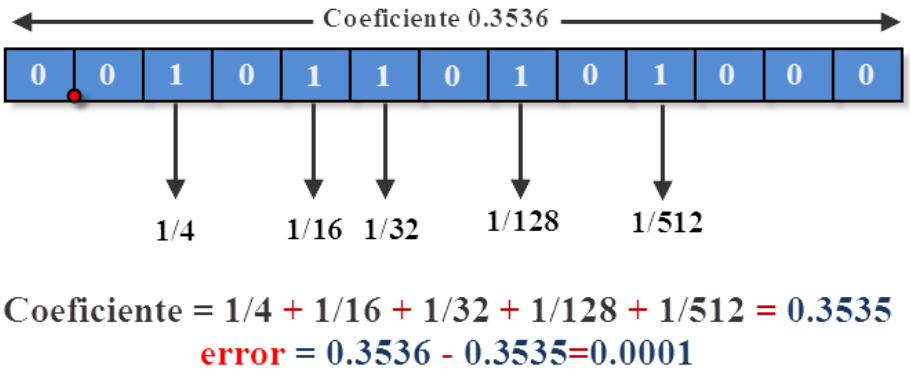


Figura 2- 33. Valores obtenidos con la representación de 13 bits de un coeficiente

Los valores de los coeficientes son asignados a las señales C1, C2, C3 y C4 mediante el procedimiento cuyo diagrama de bloques se muestra en la Figura 2-34. La señal *indexi* indica qué conjunto de coeficientes debe asignarse a las señales antes mencionadas. La señal *indexi* es la salida de un contador módulo 8, que se habilita cuando la señal de *reset* se desactiva.

La selección de los valores de los coeficientes también es sincrónica, por lo cual se tiene un retardo de un ciclo de reloj en tener los valores asignados a estas señales (C1, C2, C3 y C4). Esto hace que en el décimo ciclo de reloj coincidan los resultados de la suma (o resta) de los pares de pixeles con el correspondiente coeficiente para su multiplicación.

La multiplicación de los coeficientes con la suma (o resta) de los pares de pixeles, se lleva a cabo en el décimo ciclo de reloj, y su resultado se obtiene un ciclo después. Los cuatro multiplicadores se realizaron a través de IP Cores, cumpliendo con las siguientes características:

- Multiplicación de dos número con signo
- Primer operando de 10 bits (suma o resta de los pares de pixeles)
- Segundo operando de 13 bits (coeficiente de la matriz de transformación)
- Uso de multiplicadores embebidos
- Latencia del resultado de un ciclo de reloj

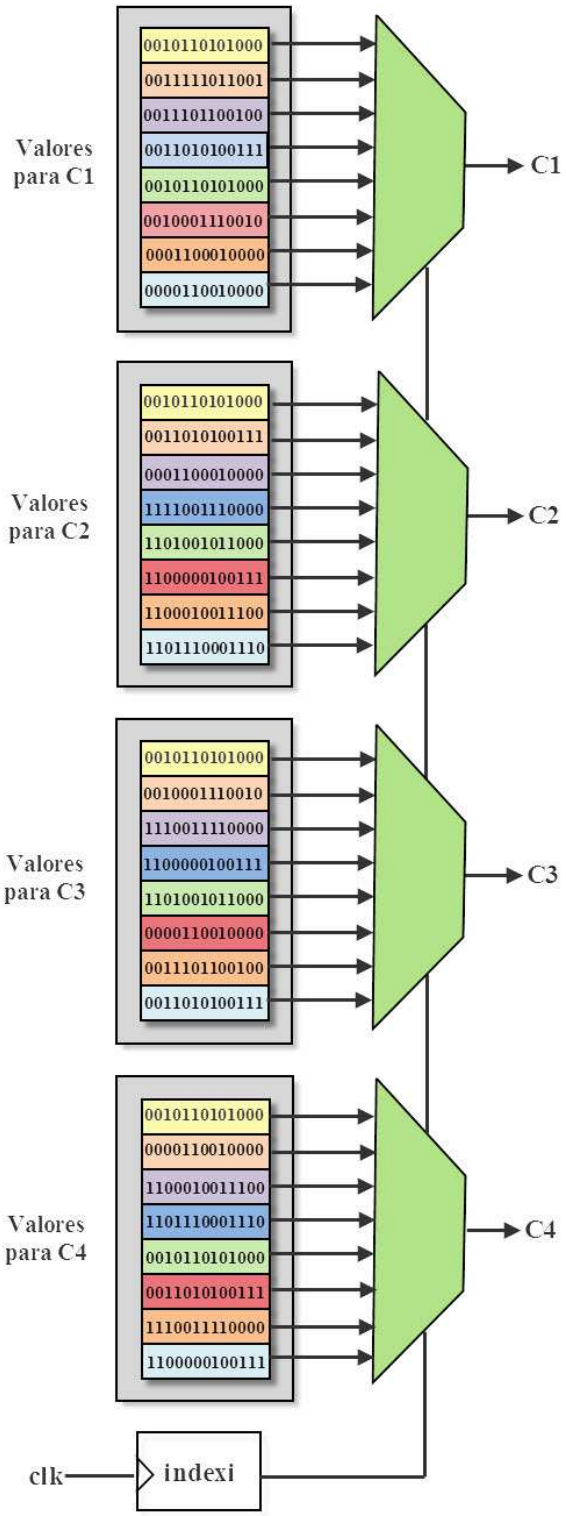


Figura 2- 34. Diagrama de bloques del proceso de asignación de coeficientes de la matriz de transformación

El resultado de las multiplicaciones son asignadas a cuatro señales de 22 bits cada una, *p1a_all*, *p2a_all*, *p3a_all* y *p4a_all*, que posteriormente son registradas,

duplicando su bit más significativo, para evitar el efecto de *overflow* en la suma posterior de estos productos, tal como se muestra a continuación:

```
P_prodA:process(clk)
begin
   if (rising_edge (clk)) then
      if (rst = '1') then
         p1a <= (others =>'0'); p2a <= (others =>'0');
         p3a <= (others =>'0'); p4a <= (others =>'0');
      else
            p1a <= p1a_all(21) & p1a_all;
            p2a <= p2a_all(21) & p2a_all;
            p3a <= p3a_all(21) & p3a_all;
            p4a <= p4a_all(21) & p4a_all;
     end if;
   end if;
end process P_prodA;
```

La suma de los productos representados por la señales *p1a*, *p2a*, *p3a* y *p4a*, se llevan a cabo en dos ciclos de reloj. En el primer ciclo se lleva a cabo la suma de *p1a* con *p2a*, y *p3a* con *p4a*, que son asignadas a las señales *z_out_int1* y *z_out_int2*, respectivamente. En el segundo ciclo se lleva a cabo la suma de las dos últimas señales y es asignada a la señal *z_out_int*. Con estas consideraciones el primer resultado de la primera DCT-1D (primer elemento de la matriz Z), aplicado a las filas del bloque de 8x8 pixeles, se obtiene en el 14avo ciclo de reloj.

La señal *z_out_int* es una señal de 24 bits, que maneja 12 bits de parte fraccionaria y 12 bits de parte entera. Antes de almacenar el resultado de la primera DCT-1D, se hace un redondeo de la parte entera, para así evitar tener que almacenar los 24 bits de todo el resultado. Para el redondeo es suficiente analizar el bit más significativo de la parte entera y los ocho bits más significativos de la parte fraccionaria de la suma de productos, representada por la señal *z_out_int*. Estos bits son asignados a una señal, denominada *fractional1,* como se indica a continuación:

```
fractional1 <= z_out_int(23) & z_out_int(11 downto 4);
```

Ya que las operaciones que se realizan en el procedimiento de la DCT-2D están en complemento a 2, se debe considerar para el proceso de redondeo que:

- Si el valor de la señal *fractional1* está comprendido entre los valores decimales 128 a 255, se debe sumar uno a la parte entera. Esto porque el número (*z_ount_int*) es positivo y su parte fraccionaria es mayor a +0.5.
- Si el valor de la señal *fractional1* está comprendido entre los valores decimales 257 a 384, se debe mantener la parte entera. Esto porque el número (*z_ount_int*) es negativo y su parte fraccionaria es menor a +0.5.
- Si el valor de la señal *fractional1* está comprendido entre los valores decimales 385 a 511, se debe sumar uno a la parte entera. Esto porque el número (*z_ount_int*) es negativo y su parte fraccionaria es mayor a +0.5.
- En cualquier otro caso de mantiene la parte entera.

La parte entera redondeada se almacena en un sistema de memorias, para su posterior procesamiento, a partir del 14avo ciclo de reloj. Un contador módulo 14 indica el momento en que debe ser habilitado el sistema de memorias, así como los generadores de direcciones de escritura y lectura.

El sistema de memorias, para almacenar los resultados de la primera DCT-1D (por filas), consiste de dos memorias RAM de 64 palabras de 11 bits. El sistema actúa de tal forma que mientras se almacenan los resultados obtenidos de la primera DCT-1D de un bloque de 8x8 pixeles, se lee los datos de la DCT-1D del bloque previo en la otra memoria, almacenados previamente. Este esquema, permite seguir el flujo de procesamiento de los datos de manera continua. La selección de la memoria para lectura o escritura se hace por medio de una señal de control.

El esquema de direccionamiento utilizado para almacenar los elementos de la primera DCT-1D, consiste en un generador de direcciones que es implementado

mediante un contador módulo 64 ascendente. Este esquema de direccionamiento permite almacenar los resultados de la matriz intermedia Z, fila por fila. El esquema de direccionamiento para la lectura de los datos, por su parte, se realiza columna por columna, para la aplicación de la segunda DCT-1D. Esto se logra, primero incrementando en cada ciclo, los tres bits más significativos de la señal de dirección de lectura, e incrementando cada 8 ciclos, los tres bits menos significativos de esta señal, como se indica en la siguiente sección de código:

```
process (clk)
begin
   if clk'event and clk = '1' then
      if (rst = '1') then
         rd_add(5 downto 3) <= "111";
      else
         if (ena_ramreg = '1') then
            rd_add(5 downto 3) <= rd_add(5 downto 3) + "001";
         end if;
      end if;
   end if;
end process;

process (clk)
begin
   if clk'event and clk = '1' then
      if (rst = '1') then
         rd_add(2 downto 0) <= "111";
      else
         if (ena_ramreg = '1' and rd_add(5 downto 3) = "111") then
            rd_add(2 downto 0) <= rd_add(2 downto 0) + "001";
         end if;
      end if;
   end if;
end process;
```

La señal de control indica en qué memoria se realiza la operación de escritura y en qué memoria la operación de lectura, conforme van ingresando los datos de la primera DCT-1D. La señal de control es de un solo bit, y el comportamiento de la asignación de operaciones (lectura o escritura) se basa en la siguiente tabla:

Tabla 2- 5. Señal de control para la habilitación de operaciones de lectura y escritura

Señal de control	Memoria A	Memoria B
0	escritura	lectura
1	lectura	escritura

La señal de control cambia su valor cada vez que se alcanza las 64 direcciones de lectura. Con esto se garantiza que mientras en una memoria se está escribiendo, en la otra se está leyendo los contenidos de la misma. La estructura del sistema de memorias se indica en el diagrama de bloques de la Figura 2-35.

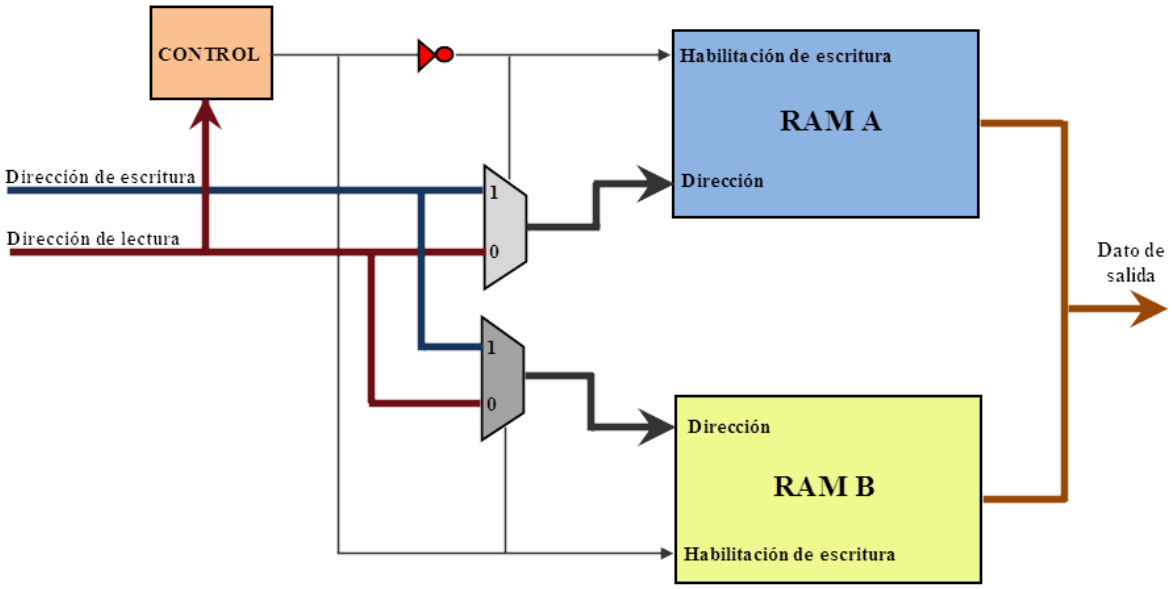


Figura 2- 35. Diagrama de bloques de la memoria de almacenamiento para la primera DCT-1D

En cada una de estas memorias se almacenan 64 coeficientes de la primera DCT-1D. Los primeros 64 coeficientes se terminan de almacenar en la memoria RAM en el ciclo 78 de la señal de reloj, luego de lo cual están disponibles para la aplicación de la segunda DCT-1D. Un contador módulo 80 indica el momento en que se habilita (se valida) la lectura de los coeficientes de la primera transformada discreta del coseno unidimensional, que se encuentran almacenados en una de las memorias RAM. A partir del ciclo 80 de la señal de reloj empieza el procesamiento de la segunda DCT-1D, esta vez a las columnas de la matriz de coeficientes de la primera DCT-1D (matriz Z en el algoritmo).

El procesamiento de la segunda DCT-1D es el mismo que en el primer caso, y empieza con la adquisición de una columna (en lugar de una fila) de la matriz intermedia de coeficientes de la primera transformada discreta del coseno unidimensional, desde una de las memorias RAM ; los elementos de la columna

de la matriz intermedia son desplazados a través de un registro de desplazamiento hasta alcanzar los ocho elementos, luego de lo cual, son registrados duplicando el bit más significativo para evitar el *overflow* en las operaciones posteriores. El mismo contador módulo 8 sirve para habilitar el registro de los elementos de cada columna (de la matriz intermedia Z), en los registros z0k_reg, z1k_reg, z2k_reg, z3k_reg, z4k_reg, z5k_reg, z6k_reg, z7k_reg, los cuales son de 12 bits (11 bits más el MSB duplicado), como lo muestra el siguiente código:

```
process (clk)
begin
   if clk'event and clk = '1' then
      if (rst = '1') then
            z0k_reg <= "000000000000";    z1k_reg <= "000000000000";
            z2k_reg <= "000000000000";    z3k_reg <= "000000000000";
            z4k_reg <= "000000000000";    z5k_reg <= "000000000000";
            z6k_reg <= "000000000000";    z7k_reg <= "000000000000";
      else
         if (cntr8 = "1000") then
            z7k_reg <= z7k(10) & z7k;
            z6k_reg <= z6k(10) & z6k;
            z5k_reg <= z5k(10) & z5k;
            z4k_reg <= z4k(10) & z4k;
            z3k_reg <= z3k(10) & z3k;
            z2k_reg <= z2k(10) & z2k;
            z1k_reg <= z1k(10) & z1k;
            z0k_reg <= z0k(10) & z0k;
         end if;
      end if;
   end if;
end process;
```

Los registros de los elementos de la columna de la matriz intermedia, comienzan a sumarse o restarse mediante el mismo mecanismo utilizado en la primera transformada discreta del coseno (DCT-1D). Las operaciones entre los pares de elementos, comienzan a ejecutarse, siendo la primera operación, la suma de éstos. Las sumas (o restas) se van asignando a cuatro señales, a saber

- *add_sub1b*, para la suma o resta de *z0k_reg* y *z7k_reg*.
- *add_sub2b*, para la suma o resta de *z1k_reg* y *z6k_reg.*
- *add_sub3b*, para la suma o resta de *z2k_reg* y *z5k_reg.*
- *add_sub4b*, para la suma o resta de *z3k_reg* y *z4k_reg.*

Debido al sincronismo guardado, en los procesos hasta el momento realizados, se puede utilizar las misma señales (el mismo procedimiento) para la asignación de los coeficientes, que multiplican a las sumas o restas de los pares de elementos de la matriz intermedia Z. Las señales que reciben los valores de los coeficientes son:

- c1, que multiplica a *add_sub1b.*
- c2, que multiplica a *add_sub2b.*
- c3, que multiplica a *add_sub3b.*
- c4, que multiplica a *add_sub4b.*

Los cuatro multiplicadores que realizan las multiplicaciones antes mencionadas, se generaron mediante IP Cores y cada uno tiene las siguientes características:

- Multiplicación de dos número con signo
- Primer operando de 12 bits (suma o resta de los pares de elementos de la matriz Z)
- Segundo operando de 13 bits (coeficiente)
- Uso de multiplicadores embebidos
- Latencia del resultado de un ciclo de reloj

El resultado de las multiplicaciones son asignadas a cuatro señales de 25 bits cada una: *p1b_all*, *p2b_all*, *p3b_all* y *p4b_all*, que, posteriormente, son almacenadas en cuatro registros de 26 bits, duplicando el valor del bit más significativo de estas señales, para evitar el efecto de *overflow* en la posterior suma de estos productos, tal como se muestra a continuación:

```
P_prodB:process(clk)
begin
   if (rising_edge (clk)) then
      if (rst = '1') then
         p1b <= (others =>'0'); p2b <= (others =>'0');
         p3b <= (others =>'0'); p4b <= (others =>'0');
      else
            p1b <= p1b_all(24) & p1b_all;
            p2b <= p2b_all(24) & p2b_all;
            p3b <= p3b_all(24) & p3b_all;
            p4b <= p4b_all(24) & p4b_all;
      end if;
   end if;
end process P_prodB;
```

Los registros de los productos obtenidos son sumados en dos ciclos de reloj: en el primer ciclo se lleva a cabo la suma de *p1b* con *p2b*, y *p3b* con *p4b*, que son asignadas a las señales *dct2d_int1* y *dct2d_int2*, respectivamente. En el segundo ciclo, se lleva a cabo la suma de las dos últimas señales y el resultado es asignado a la señal *dct_2d_int*. La señal *dct_2d_int*, es de 27 bits, y maneja 12 bits de parte fraccionaria. Para obtener el primer resultado de la DCT-2D del bloque de 8x8 pixeles, se realiza el procedimiento de redondeo de la parte entera de la señal *dct_2d_int*, tal como se hizo para la primera DCT-1D (la señal *z_out_int*), y considerando 12 bits para ésta.

De esta manera se obtiene el primer resultado de la DCT-2D en el ciclo 94 de la señal de reloj. Un contador módulo 94 fue incluido para habilitar una señal (*rdy_dct*) que indique cuando el primer resultado de la DCT-2D está listo, tal como se muestra a continuación:

```
process (clk)
begin
   if clk'event and clk = '1' then
      if (rst = '1') then
         cntr94 <= "0000000";
      else
            if (cntr94 < "1011110") then
               cntr94 <= cntr94 + "0000001";
            else
               cntr94 <= cntr94;
            end if;
      end if;
   end if;
end process;

rdy_dct <= '1' WHEN (cntr94 = "1011110") ELSE '0';
```

2.5.1.3 Simulación del bloque de la DCT-2D

Para la simulación del bloque de la DCT-2D, se ha usado como ejemplo el bloque de 8x8 pixeles de la componente de luminancia de una imagen, que se muestra en la fuente [9], y que se indica en la Figura 2-36.

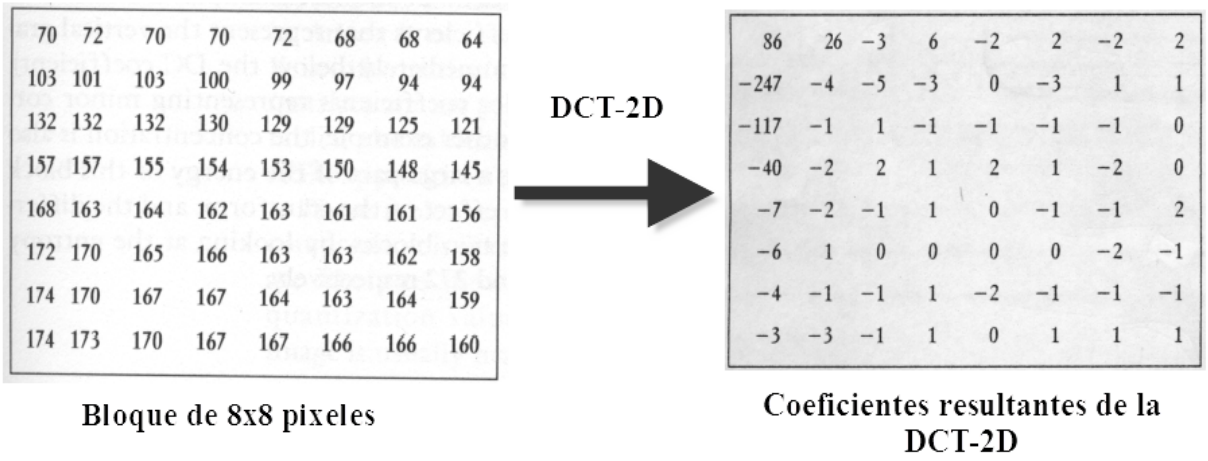


Figura 2- 36. Resultados teóricos de referencia para la simulación del bloque de la transformada discreta del coseno

El resultado de la simulación del bloque de la transformada discreta del coseno en dos dimensiones (DCT-2D) se indica en la Figura 2-37. Se ha incluido en la entidad del bloque DCT-2D, un puerto que cuenta los ciclos de reloj, solo con fines de simulación, para observar la latencia inicial en los resultados.

Como se puede observar en la Figura 2-37b, los efectos que producen las aproximaciones, tanto en los coeficientes de la matriz de transformación como en el redondeo de la parte entera de los resultados de las dos DCT-1D, hacen que en ciertos coeficientes el resultado final difiera en una unidad por encima o por debajo del resultado teórico de la DCT-2D, que se puede observar en la Figura 2-36. Sin embargo, este efecto se ve reducido por la cuantización posterior, en la cual esta variación no afecta significativamente el resultado de los coeficientes DCT-2D cuantizados.

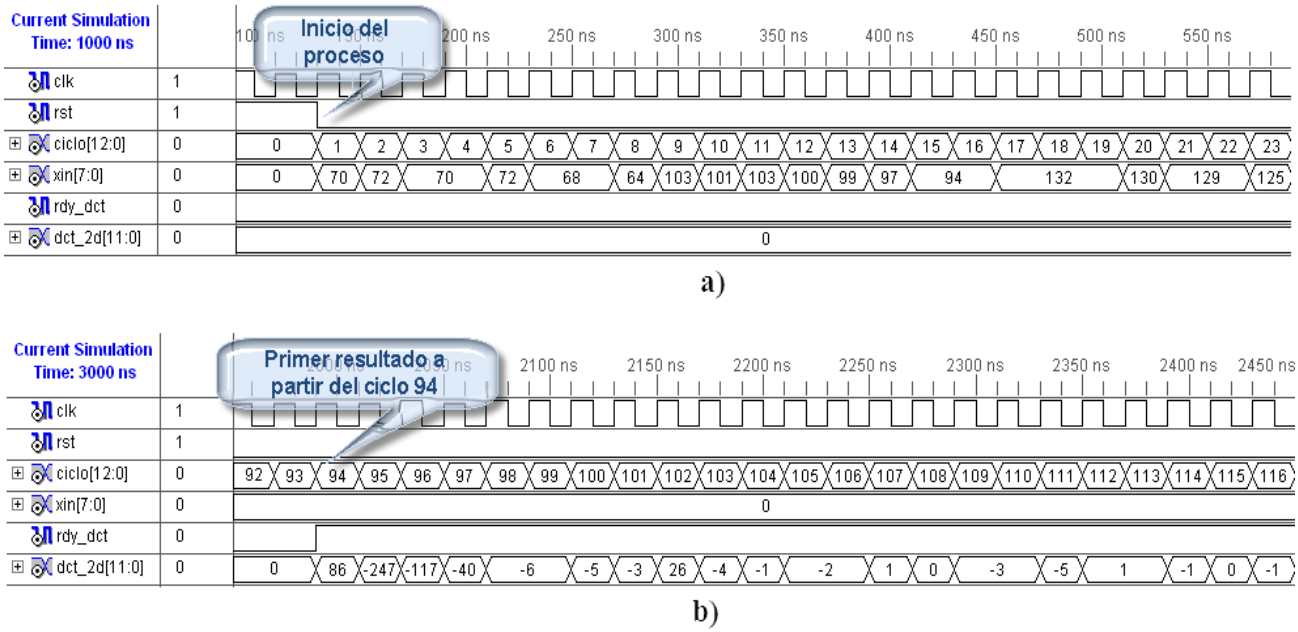


Figura 2- 37. Resultados de la simulación del bloque de la DCT-2D: a) Inicio del proceso de la DCT-2D b) Primer resultado de la DCT-2D

El código VHDL del bloque de la transformada discreta del coseno, se encuentra debidamente comentado en el Anexo 11 del proyecto.

2.5.2 CUANTIZACIÓN

Cuando se obtiene el primer coeficiente DCT-2D, en el ciclo de reloj 94, la señal *rdy_dct*, que indica cuando los coeficientes DCT-2D están listos, también deshabilita la señal de *reset* del bloque de cuantización, con lo cual, se empieza a procesar los datos que provienen del bloque de la transformada discreta del coseno en dos dimensiones.

En la Figura 2-38 se ilustra una representación de la entidad VHDL del bloque de cuantización, y en la Tabla 2-6 se indican la dirección, la longitud en bits y el valor activo de los puertos que posee la entidad VHDL del bloque de cuantización.

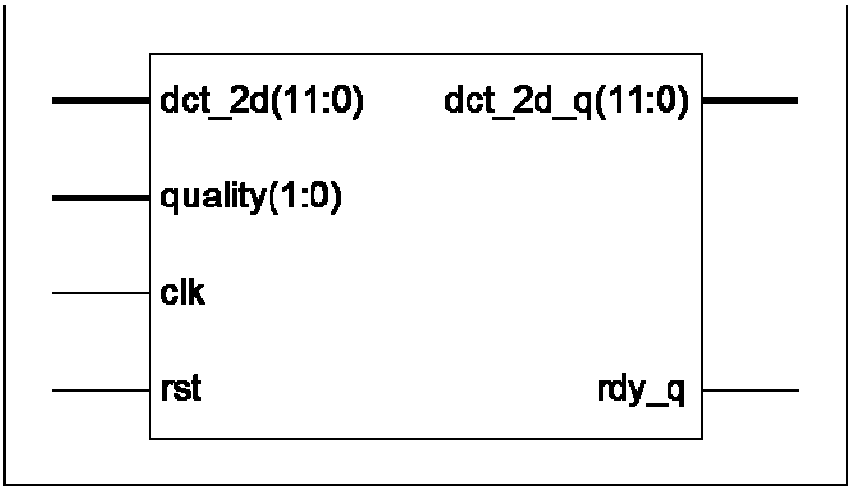


Figura 2- 38. Representación de la entidad VHDL del bloque de cuantización

Tabla 2- 6. Puertos de la entidad VHDL del módulo de cuantización

Puerto	Dirección	Longitud en bits	Valor activo	Descripción
clk	Entrada	N/A	N/A	Señal de reloj.
rst	Entrada	1	1L	Señal de *reset.*
dct_2d	Entrada	12	N/A	Puerto de entrada para los coeficientes del bloque de la DCT-2D.
quality	Entrada	2	N/A	Selector del factor de calidad de la imagen comprimida.
dct_2d_q	Salida	12	N/A	Puerto de salida con los coeficientes DCT-2D cuantizados.
rdy_q	Salida	1	1L	Indica cuándo se tiene en el puerto *dct_2d_q* un dato válido.

La entidad VHDL del bloque de cuantización está conformada por dos componentes: una memoria ROM, donde se almacenan las tablas de cuantización, y un divisor creado utilizando un *IP Core*, el cual se encarga de dividir el coeficiente DCT-2D, para su correspondiente elemento de la tabla de cuantización.

El módulo de compresión ha sido diseñado para proveer compresión de imágenes monocromáticas, con diferentes niveles de cuantización (calidad). Esto se logra mediante la definición de varias tablas de cuantización, con las cuales se obtiene

un factor de calidad diferente. Como se indicó en el Capítulo 1, se pueden generar varias tablas de cuantización en base a las tablas que se recomiendan en el estándar JPEG y en función del factor de calidad deseado. Aplicando el procedimiento de la Sección 1.11.2.1 del Capítulo 1, se obtienen las siguientes tablas de cuantización para cuatro factores de calidad de la componente de luminancia:

Factor de calidad de 25

$$\begin{bmatrix} 32 & 22 & 20 & 32 & 48 & 80 & 102 & 122 \\ 24 & 24 & 28 & 38 & 52 & 116 & 120 & 110 \\ 28 & 26 & 32 & 48 & 80 & 114 & 138 & 112 \\ 28 & 34 & 44 & 58 & 102 & 174 & 160 & 124 \\ 36 & 44 & 74 & 112 & 136 & 218 & 206 & 154 \\ 48 & 70 & 110 & 128 & 162 & 208 & 226 & 184 \\ 98 & 128 & 156 & 174 & 206 & 242 & 240 & 202 \\ 144 & 184 & 190 & 196 & 224 & 200 & 206 & 198 \end{bmatrix}$$

Factor de calidad de 35

$$\begin{bmatrix} 23 & 16 & 14 & 23 & 34 & 57 & 73 & 87 \\ 17 & 17 & 20 & 27 & 37 & 83 & 86 & 79 \\ 20 & 19 & 23 & 34 & 57 & 81 & 99 & 80 \\ 20 & 24 & 31 & 41 & 73 & 124 & 114 & 89 \\ 26 & 31 & 53 & 80 & 97 & 156 & 147 & 110 \\ 34 & 50 & 79 & 91 & 116 & 149 & 161 & 131 \\ 70 & 91 & 111 & 124 & 147 & 173 & 171 & 144 \\ 103 & 131 & 136 & 140 & 160 & 143 & 147 & 141 \end{bmatrix}$$

Factor de calidad de 50

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Factor de calidad de 75

$$\begin{bmatrix} 8 & 6 & 5 & 8 & 12 & 20 & 26 & 31 \\ 6 & 6 & 7 & 10 & 13 & 29 & 30 & 28 \\ 7 & 7 & 8 & 12 & 20 & 29 & 35 & 28 \\ 7 & 9 & 11 & 15 & 26 & 44 & 40 & 31 \\ 9 & 11 & 19 & 28 & 34 & 55 & 52 & 39 \\ 12 & 18 & 28 & 32 & 41 & 52 & 57 & 46 \\ 25 & 32 & 39 & 44 & 52 & 61 & 60 & 51 \\ 36 & 46 & 48 & 49 & 56 & 50 & 52 & 50 \end{bmatrix}$$

Estas tablas de cuantización son almacenadas en la memoria ROM que se mencionó anteriormente. Cabe mencionar que los elementos de cada tabla fueron almacenados columna por columna, ya que los resultados del bloque de la DCT-2D se obtienen en orden transpuesto.

La memoria ROM fue hecha mediante un *IP Core*, y cuenta con las siguientes características:

- Utilización de LUTs como recurso de almacenamiento.
- Capacidad de 512 palabras de 12 bits.
- Entradas y salidas registradas.

El esquema de direccionamiento utilizado para acceder a cada elemento de una tabla de cuantización, se basa en la división en dos secciones de la señal de dirección de la memoria ROM: los dos bits más significativos indican qué tabla de cuantización será utilizada para este proceso, y los seis bits menos significativos apuntan a los 64 valores de la tabla de cuantización escogida. Los seis bits menos significativos de la señal de dirección, son generados mediante un contador módulo 64, que comienza a contar cuando la señal de *reset* del módulo de cuantización se desactiva. Los dos bits más significativos están controlados directamente por dos de los cuatro *switches* de deslizamiento que posee el módulo de desarrollo, mediante los cuales el usuario del sistema puede seleccionar el factor de calidad a través de la combinación de estos *switches*, como se indica en la Tabla 2-7.

Tabla 2- 7. Combinación de los dos *switches* de deslizamiento para obtener los factores de calidad

Combinación de los *Switches*	**Factor de Calidad**
00	25
01	35
10	50
11	75

De esta manera, la distribución de la memoria ROM, que almacena las tablas de cuantización, resulta como lo indica la Figura 2-39.

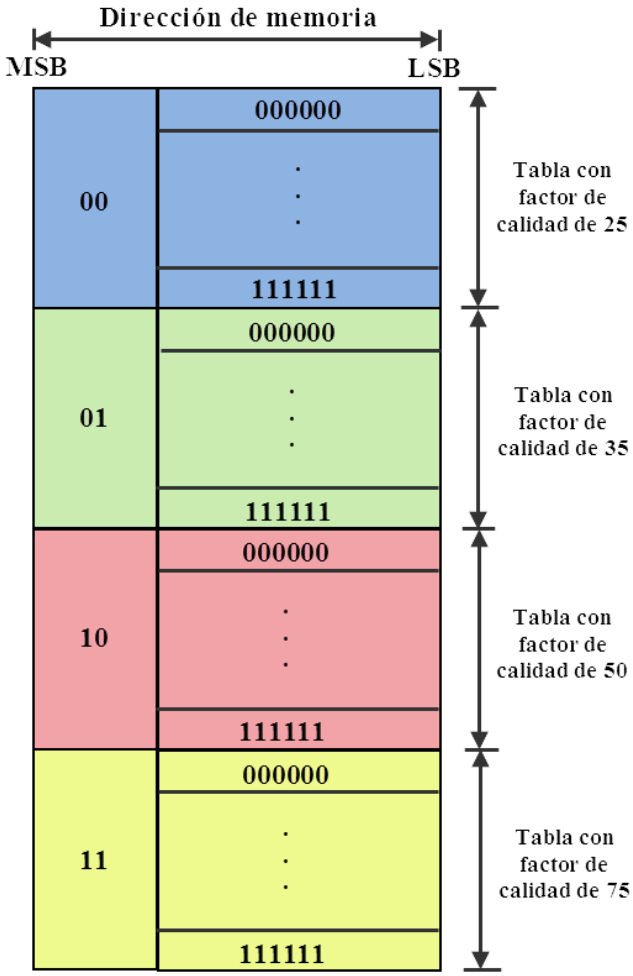


Figura 2- 39. Distribución de la memoria ROM que almacena las tablas de cuantización

En el transcurso del tiempo, desde que se desactiva la señal de *reset*, hasta la obtención del primer valor de la tabla de cuantización, transcurren tres ciclos de reloj (generación de la dirección, registro de la entrada y salida de la memoria ROM), por lo cual la entrada de los coeficiente DCT-2D debe ser retardada tres ciclos a través de un registro de desplazamiento, esto con el fin de que coincida con su correspondiente valor cuantizador.

En el tercer ciclo de reloj, los valores del coeficiente DCT-2D y del elemento de la tabla de cuantización ingresan a un divisor. El divisor fue creado utilizando un IP Core, con las siguientes características:

- Dividendo de 12 bits.

- Divisor de 12 bits.
- Cociente de 12 bits.
- Residuo fraccionario de 9 bits.
- Una división por ciclo de reloj.
- División de números con signo.
- Puerto de habilitación.

El divisor proporciona como resultados el cociente y el residuo en forma de fracción, con el cual se puede realizar un proceso de redondeo del cociente. El divisor proporciona una división por ciclo de reloj, con una latencia inicial que depende de la longitud en bits del dividendo, la longitud en bits del residuo y si la operación es con signo. Para este caso, la latencia inicial está dada por la fórmula:

$$Latencia_{(ciclos)} = M + F + 4$$

Donde, M es la longitud en bits del dividendo y F es la longitud en bits del residuo. Con esto se obtienen los resultados del divisor después de 25 ciclos de reloj. Los resultados de la división (cociente y residuo) son registrados y posteriormente se realiza el proceso de redondeo del coeficiente mediante el análisis del residuo en su forma fraccional. El cociente y el residuo son representados en complemento a 2, por lo que para el proceso de redondeo se debe considerar que:

- Si el valor decimal de la señal de residuo está entre 128 y 255, se debe sumar la unidad al cociente, ya que el cociente es positivo y su parte fraccional es mayor o igual a 0,5.
- Si el valor decimal de la señal de residuo está entre 257 y 384, se debe restar la unidad al cociente, ya que el cociente es negativo y su parte fraccional es mayor o igual a 0,5.
- En cualquier otro caso se mantiene el cociente.

La latencia inicial de todo el bloque de cuantización es de 30 ciclos de reloj, por lo que en ese ciclo se comienza a obtener los resultados de los coeficientes DCT-2D cuantizados. Con el fin de indicar este momento, se añadió una señal (*rdy_q*), que se habilita (cambia su valor a 1L) cuando un contador módulo 30 alcanza su máximo valor.

2.5.2.1 Simulación del bloque de cuantización

Para la simulación se ha tomado como ejemplo los resultados de la DCT-2D, obtenidos en la sección anterior y la tabla de cuantización con factor de calidad de 50, cuyo resultado teórico de la cuantización se muestra en la Figura 2-40. El resultado de la simulación del bloque de cuantización se indica en la Figura 2-41. Cabe mencionar que la simulación se lleva a cabo conjuntamente con el bloque de la transformada discreta coseno, para verificar la funcionalidad conjunta de estos dos bloques.

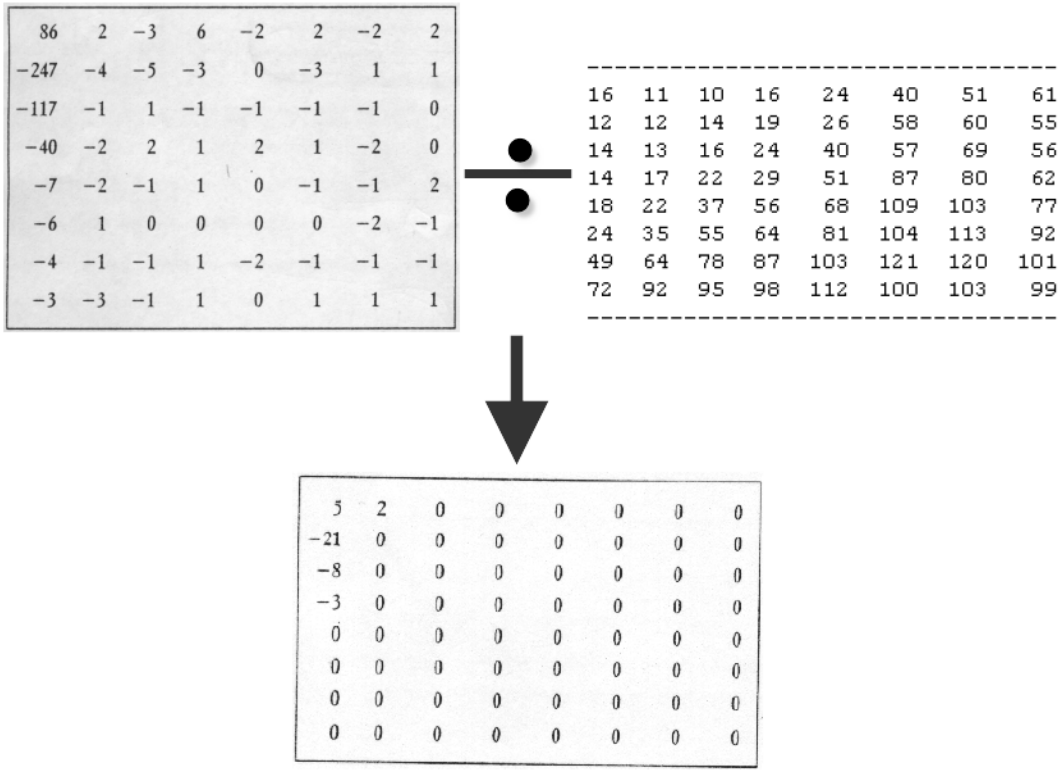


Figura 2- 40. Resultado teórico de referencia para la simulación del bloque de Cuantización

Como se puede observar en la Figura 2-40, los resultados teóricos coinciden con los del diseño VHDL de la entidad del bloque de cuantización. Una vez que los resultados de la cuantización se comienzan a obtener, éstos deben ser almacenados en una memoria RAM cumpliendo con un orden específico para la exploración en zig-zag.

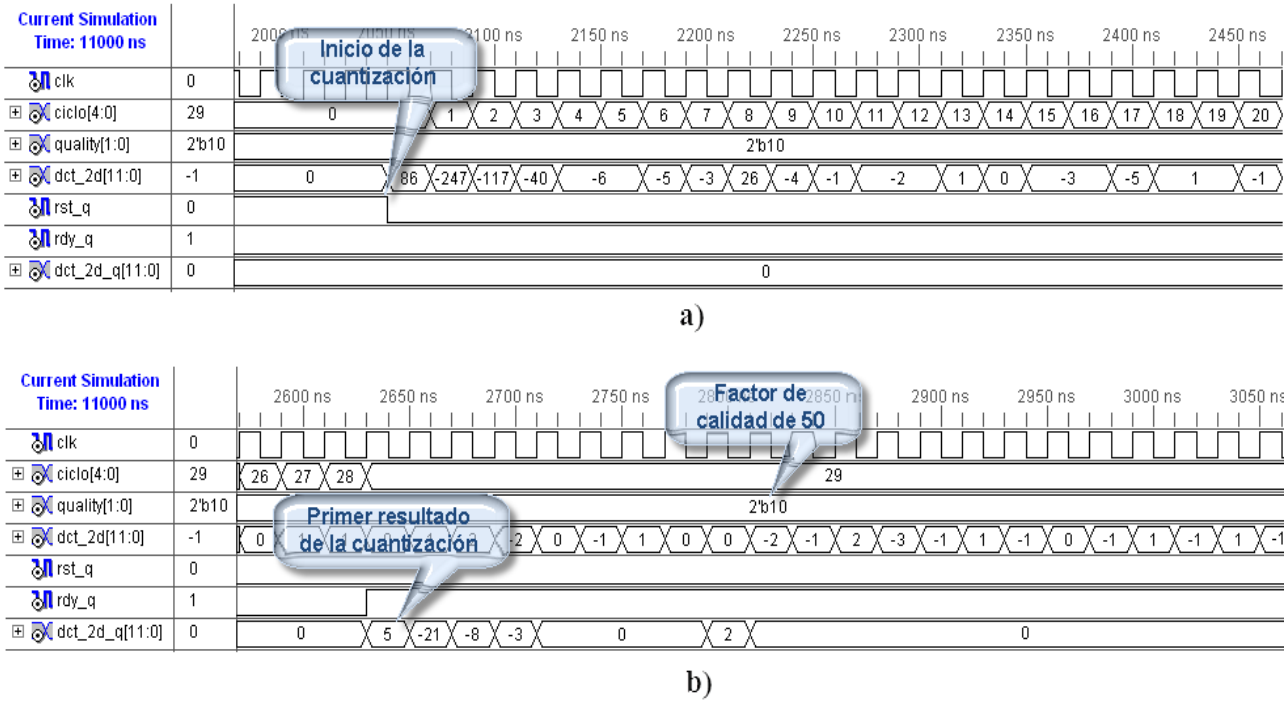


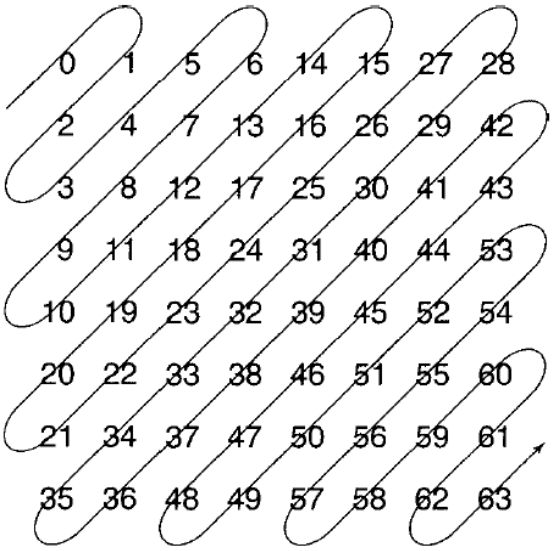
Figura 2- 41. Resultado de la simulación del bloque de cuantización: a) inicio del proceso b) resultado de la cuantización a partir del ciclo 30 de la señal de reloj

El código VHDL del bloque de cuantización, se encuentra debidamente comentado en el Anexo 11 del proyecto.

2.5.3 EXPLORACIÓN EN ZIG-ZAG

La señal *rdy_q*, que indica cuando los coeficientes DCT-2D cuantizados están listos, también deshabilita la señal de *reset* del bloque de exploración en zig-zag, con lo cual se empieza a ordenar los coeficientes en la disposición que se muestra en la Figura 2-42.

En la Figura 2-43 se ilustra una representación de la entidad VHDL del bloque de exploración en zig-zag, y en la Tabla 2-8 se indican la dirección, la longitud en bits y el valor activo de los puertos que posee la entidad VHDL del bloque de exploración en zig-zag.



Fuente [10]

Figura 2- 42. Orden en zig-zag para los coeficientes DCT-2D cuantizados

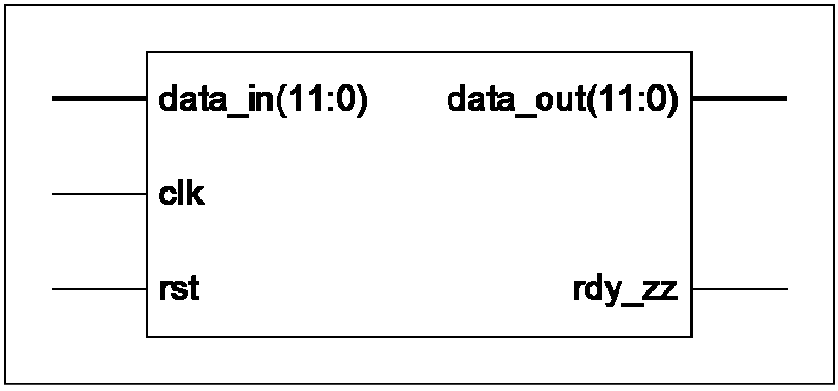


Figura 2- 43. Representación de la entidad VHDL del bloque de exploración en zig-zag

Tabla 2- 8. Puertos de la entidad VHDL del módulo de exploración en zig-zag

Puerto	Dirección	Longitud en bits	Valor activo	Descripción
clk	Entrada	N/A	N/A	Señal de reloj.
rst	Entrada	1	1L	Señal de *reset.*
data_in	Entrada	12	N/A	Puerto de entrada para los coeficientes DCT-2D cuantizados.
data_out	Salida	12	N/A	Puerto de salida con los coeficientes DCT-2D cuantizados en disposición de zig-zag.
rdy_zz	Salida	1	1L	Indica cuándo se tiene en el puerto *data_out* un dato válido.

El bloque de exploración en zig-zag consta de una memoria ROM donde se guarda el orden en que deben ser almacenados los coeficientes cuantizados, para poder tener su disposición en zig-zag, y un sistema de dos memorias RAM donde se almacenan los coeficientes ya ordenados. El diagrama de bloques para el bloque de la exploración en zig-zag se muestra en la Figura 2-44. Cuando se desactiva la señal de *reset* de este bloque, se empiezan a generar dos direcciones: la dirección de la memoria ROM y la dirección de lectura para las memorias RAM. El dato de salida de la memoria ROM, proporciona la dirección de escritura de los coeficientes cuantizados, para ser almacenados en una de las dos memorias RAM con la disposición en zig-zag. La selección de la memoria RAM se realiza mediante una señal de control.

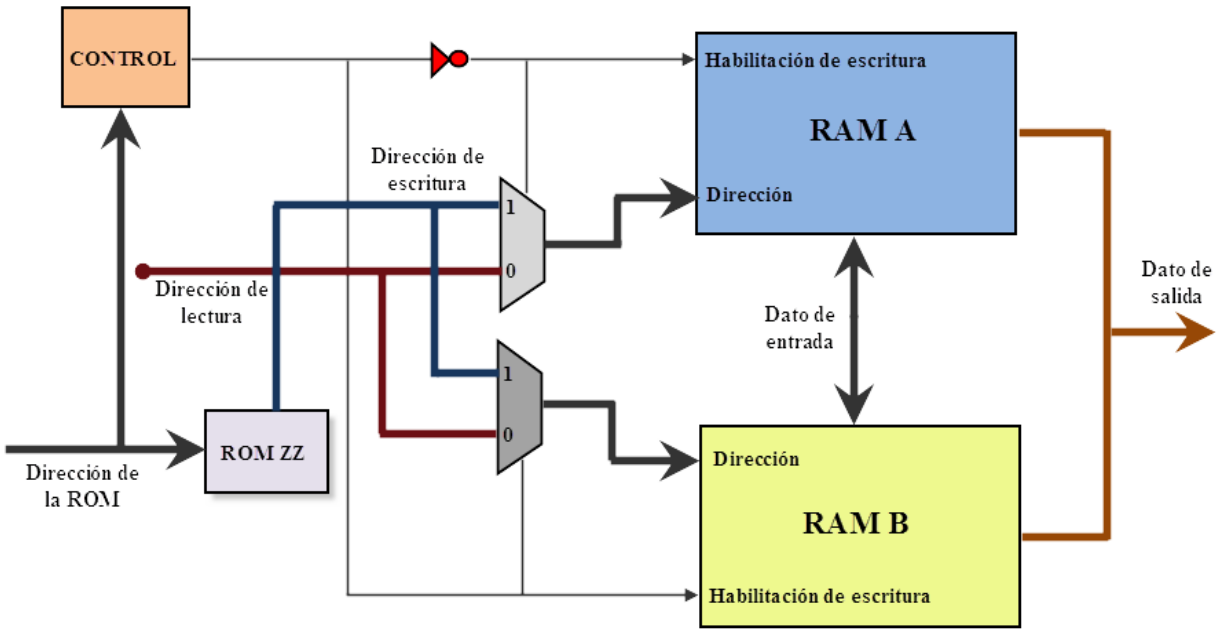


Figura 2- 44. Diagrama de bloques del bloque de exploración en zig-zag

El procedimiento de control es igual al sistema de memorias del bloque de la DCT-2D. La señal de control indica en qué memoria se realiza la operación de escritura y en qué memoria la operación de lectura, conforme van ingresando los datos de la primera DCT-1D. La señal de control es de un solo bit, y el comportamiento de la asignación de operaciones (lectura o escritura), se basa en la siguiente tabla:

Tabla 2- 9. Señal de control para la habilitación de operaciones de lectura y escritura, del bloque de exploración en zig-zag

Señal de control	Memoria A	Memoria B
0	escritura	lectura
1	lectura	escritura

La señal de control cambia su valor cada vez que se alcanza las 64 direcciones de lectura de la memoria ROM. Con esto se garantiza que mientras en una memoria se está escribiendo, en la otra se está leyendo los coeficientes DCT-2D cuantizados en disposición de zig-zag.

La memoria ROM fue creada mediante un IP Core, y consta de las siguientes características:

- Utilización de LUTs como recurso de almacenamiento.
- Capacidad de 64 palabras de 6 bits.
- Entradas y salidas registradas.

Ya que los resultados de los coeficientes cuantizados se obtienen en orden transpuesto, los valores mostrados en la Figura 2-42 deben ser almacenados en la memoria ROM, columna por columna.

La latencia inicial que se tiene en este módulo, para obtener el primer resultado, es de 67 ciclos de reloj. Se incluyó una señal (*rdy_zz*), que indica el momento en que los datos a la salida del bloque son válidos; esta señal se activa cuando un contador módulo 67 llega a su valor máximo.

2.5.3.1 Simulación del bloque de exploración en zig-zag

Para la simulación se ha tomado como ejemplo la exploración en zig-zag de los coeficientes cuantizados, obtenidos en la sección anterior, cuyo resultado teórico se muestra en la Figura 2-45. El resultado de la simulación del bloque de exploración en zig-zag, se indica en la Figura 2-46. Cabe mencionar que la simulación se lleva a cabo conjuntamente con el bloque de la transformada discreta del coseno y el bloque de cuantización, para verificar la funcionalidad conjunta de estos bloques.

(5)	**2**	**-21**	**-8**	**0**	**0**	**0**	**0**	**0**	**-3**	**(todos ceros)**

Fuente [9]

Figura 2- 45. Resultado teórico de referencia para la simulación del bloque de exploración en zig-zag

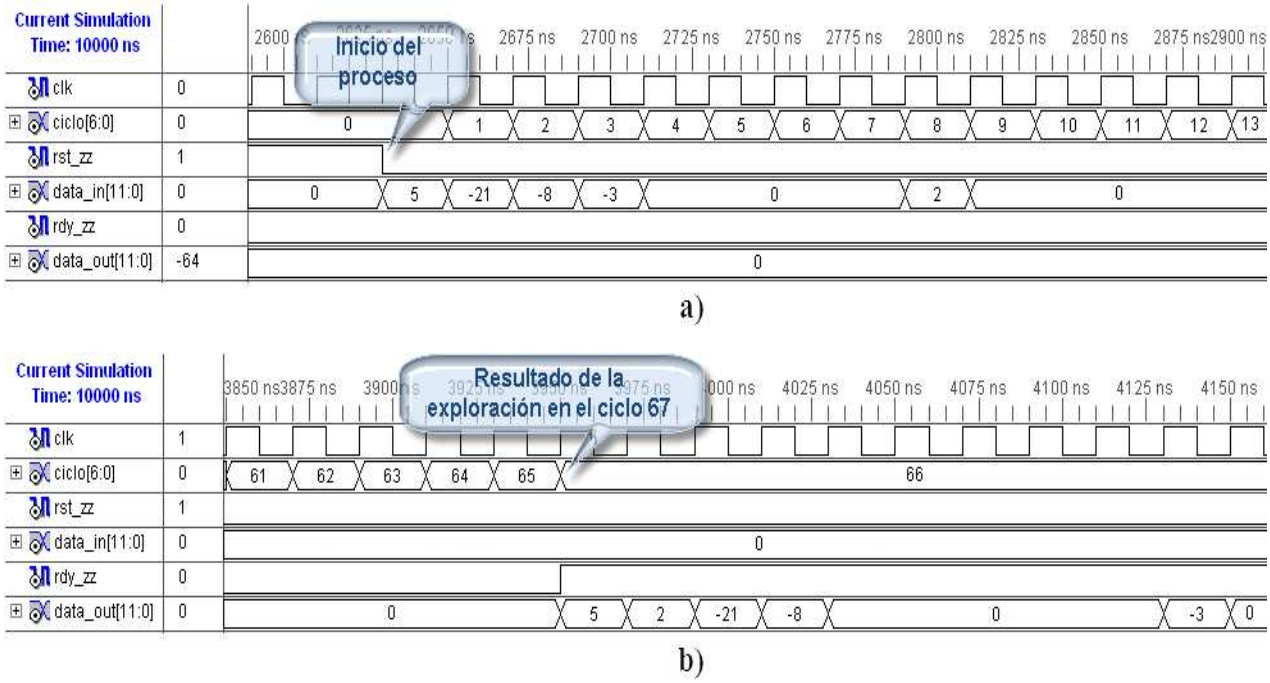


Figura 2- 46. Resultados de la simulación parea el bloque de exploración en zig-zag: a) inicio del proceso b) resultado de la exploración a partir del ciclo 67

Como se puede observar, los resultados teóricos coinciden con los del diseño VHDL de la entidad de exploración en zig-zag. Los coeficientes cuantizados, dispuesto de este modo (zig-zag), ingresan al bloque de codificación de entropía de Huffman, donde se les asignará palabras código, que producen la compresión propiamente dicha.

El código VHDL de la descripción del bloque de exploración en zig-zag, se encuentra debidamente comentado en el Anexo 11 de este proyecto.

2.5.4 CODIFICACIÓN DE LA ENTROPÍA DE HUFFMAN

En este bloque se realiza la codificación de los coeficientes ordenados en zig-zag, mediante los códigos de Huffman que se establecen en el estándar JPEG; este bloque define las palabras código A y B, que se discutieron en el Capítulo 1. En la Figura 2-47, se ilustra una representación de la entidad del bloque de codificación

de Huffman y en la Tabla 2-10 se indican la dirección, la longitud en bits y el valor activo de los puertos que posee esta entidad.

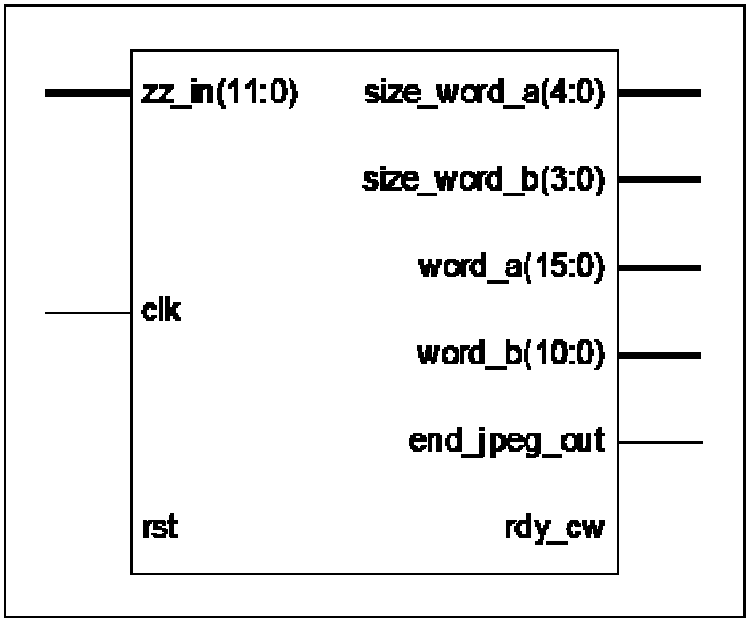


Figura 2- 47. Representación de la entidad VHDL del bloque de codificación de Huffman

Tabla 2- 10. Puertos de la entidad VHDL del módulo de codificación de Huffman

Puerto	Dirección	Longitud en bits	Valor activo	Descripción
clk	Entrada	N/A	N/A	Señal de reloj.
rst	Entrada	1	1L	Señal de *reset.*
zz_in	Entrada	12	N/A	Puerto de entrada para los coeficientes DCT-2D cuantizados, en disposición zig-zag.
size_word_a	Salida	5	N/A	Indica la longitud en bits de la palabra código A.
size_word_b	Salida	4	N/A	Indica la longitud en bits de la palabra código B.
word_b	Salida	11	N/A	Puerto de salida para la palabra código B.
word_a	Salida	16	N/A	Puerto de salida para la palabra código A.
end_jpeg_out	Salida	1	1L	Indica si los valores en los puertos *size_word_a, size_word_b, word_a, word_b* corresponden al último pixel de la imagen.
rdy_cw	Salida	1	1L	Indica si los valores en los puertos *size_word_a, size_word_b, word_a, word_b* son válidos.

El bloque de codificación de Huffman consta de cuatro componentes, como se puede observar en la Figura 2-48.

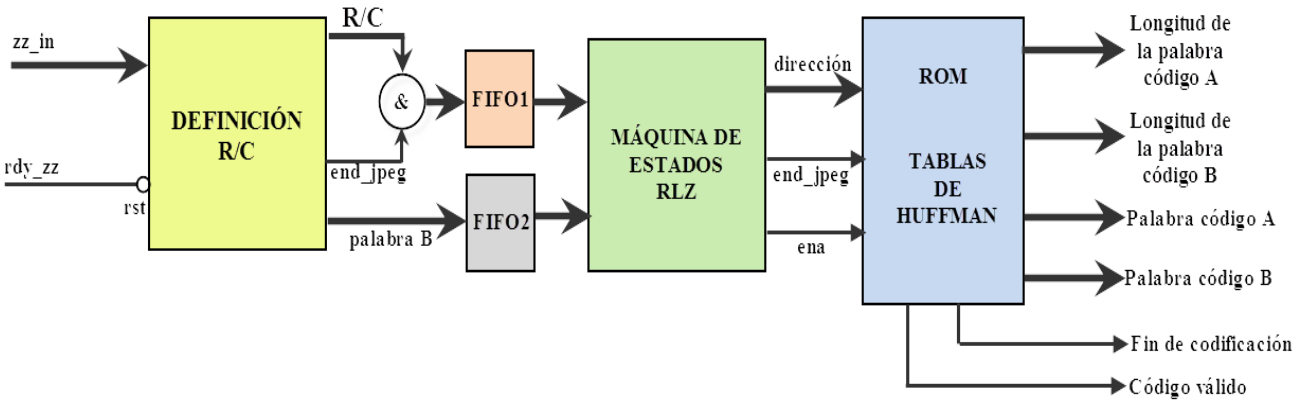


Figura 2- 48. Diagrama de bloques del bloque de codificación de Huffman

2.5.4.1 Primer Componente: Definición de categoría y longitud de series de ceros

La señal *rdy_zz*, que indica cuando los coeficientes DCT-2D cuantizados están ordenados en zig-zag, también deshabilita la señal de *reset* del bloque de codificación de Huffman. Los coeficientes cuantizados ingresan al primer componente de este bloque, en el cual, se define la categoría a la que pertenecen los coeficientes no nulos, y la longitud de series de ceros que preceden a éstos. En la Tabla 2-11 se indican los puertos que tiene este componente del bloque de codificación de Huffman.

Tabla 2- 11. Puertos del primer componente del bloque de codificación de Huffman

Puerto	Dirección	Longitud en bits	Valor activo	Descripción
clk	Entrada	N/A	N/A	Señal de reloj.
rst	Entrada	1	1L	Señal de *reset.*
zz_in	Entrada	12	N/A	Puerto de entrada para los coeficientes DCT-2D cuantizados en disposición zig-zag
word_b1	Salida	11	N/A	Puerto de salida del valor del coeficiente no nulo (palabra código B)
run	Salida	7	N/A	Puerto de salida para la longitud de la serie de ceros.
category	Salida	4	N/A	Puerto de salida para la categoría del coeficiente no nulo.
rdy	Salida	1	1L	Indica si los datos en los puertos *word_b*, *run* y *category* son válidos.
end_jpeg	Salida	1	1L	Indica si los valores en los puertos *word_b*, *run* y *category* corresponden al último pixel de la imagen

En la Figura 2-49 se ilustra una representación de la entidad VHDL del primer componente del bloque de codificación de Huffman.

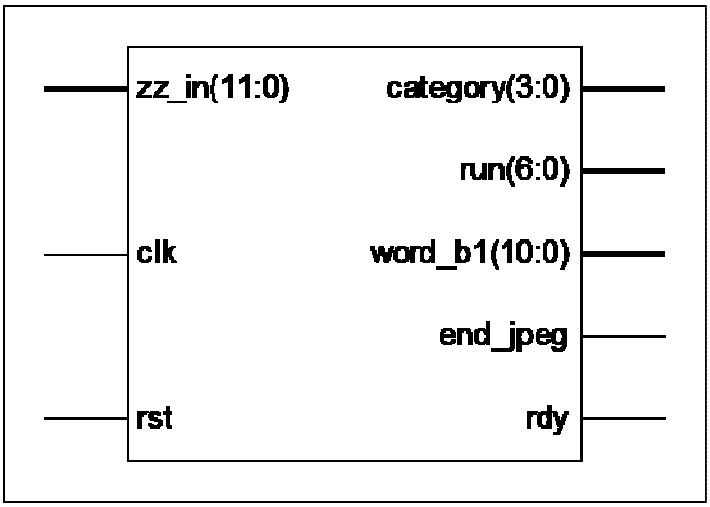


Figura 2- 49. Representación de la entidad VHDL del primer componente del bloque de codificación de Huffman

En este bloque se realiza, en primer lugar, la diferenciación entre el coeficiente DC y los 63 coeficientes AC. Para esto, cuando la señal de *reset* se desactiva, empieza a correr un contador módulo 64; cuando este contador indica 1, una señal auxiliar de un bit denominada *dc_rdy*, se habilita señalando que en ese momento ha ingresado el coeficiente DC a través del puerto de entrada *zz_in*. Esta señal sirve como habilitación de un registro de desplazamiento, como se muestra a continuación:

```
process (clk)
   begin
      if clk'event and clk = '1' then
         if rst = '1' then
            dc_old <= (others => '0');
            dc_new <= (others => '0');
         else
            if dc_rdy = '1' then
               dc_new <= zz_in_reg;
               dc_old <= dc_new;
            else
               dc_old <= dc_old;
               dc_new <= dc_new;
            end if;
         end if;
      end if;
end process;
```

Cuando la señal *dc_rdy* es activa, se produce un desplazamiento: en la señal *dc_new* ingresa el nuevo coeficiente DC, y en la señal *dc_old* se almacena el coeficiente DC del bloque previo. De esta manera se puede llegar a obtener el coeficiente DC diferencial, sincronizado con la señal *dc_rdy* desplazada un ciclo de reloj (mediante un registro), como se muestra a continuación:

```
dc_diff <= dc_new - dc_old when dc_rdy_reg = '1' else (others => '0');
```

Identificado el coeficiente DC, y obtenido el coeficiente DC diferencial, se prosigue con el análisis de los coeficientes. El coeficiente diferencial es el primero en entrar a una serie de comparadores a fin de obtener la categoría de éste, después prosiguen los coeficientes AC, en el orden en que ingresan al bloque de codificación de Huffman. Ya que las categorías de los coeficientes DC diferenciales y de los coeficientes AC se definen con los mismos rangos de valores, desde la categoría 1 hasta la categoría 10, se puede utilizar un solo comparador para determinar cualquier categoría desde la 0 hasta la 11. El comparador, consiste de un estamento ***case*** que compara el valor del coeficiente actual, con los rangos de valores de cada categoría, dentro de un bloque secuencial ***process*** como se muestra en el siguiente código:

```
--Definición de categoría

process (clk)
begin
   if rising_edge(clk) then
      category_s <= (others => '0');
      case conv_integer(coeff) is
         --Categoría 1
         when 4095 =>
            category_s <= "0001";
         when 1 =>
            category_s <= "0001";
         --Categoría 2
         when 4093 to 4094 =>
            category_s <= "0010";
         when 2 to 3 =>
            category_s <= "0010";
         --Categoría 3
         when 4089 to 4092 =>
            category_s <= "0011";
         when 4 to 7 =>
            category_s <= "0011";
                  .
                  .
                  .
                  .
         --Categoría 11
         when 2049 to 3072 =>
            category_s <= "1011";
         when 1024 to 2047 =>
            category_s <= "1011";
         when others =>
            null;
      end case;
   end if;
end process;

category <= category_s;
```

Dependiendo del caso se asigna a la salida *category*, el valor de la categoría del coeficiente no nulo*.*

Simultáneamente, después de obtenido el coeficiente DC diferencial, se empieza a comparar los coeficientes AC en busca de coeficientes nulos (series de ceros). Cuando se detecta un coeficiente nulo, se incrementa un contador; este procedimiento continua hasta encontrar un coeficiente no nulo, momento en el cual, el resultado del contador de series de ceros es asignado a la salida *run,* del

componente en cuestión. La salida *run* de este componente, tiene una particularidad, y es que, esta señal de salida indica también si los resultados obtenidos de la definición de categoría y la longitud de series de ceros, corresponden a un coeficiente diferencial DC o a un coeficiente AC. Esto se logra asignando al bit más significativo de este puerto de salida, la bandera que indica un coeficiente DC (*dc_rdy*). Cabe mencionar que la búsqueda de coeficientes nulos, se habilita solo cuando el contador módulo 64 es mayor a 1, para evitar así conflictos con los coeficientes DC diferenciales, los cuales pueden ser nulos.

Cuando un coeficiente no es nulo, se asigna a la salida *word_b1*, la correspondiente palabra código B, que representa el valor del coeficiente. Para esto, si el coeficiente no nulo es positivo, se asigna el mismo valor del coeficiente a la salida *word_b1*, pero si éste es negativo, se debe asignar el complemento a 1 del coeficiente. El complemento a 1 del coeficiente puede ser hallado mediante la negación del resultado de la suma de la negación del coeficiente con la unidad, como se indica a continuación:

$$coeficiente_{a1} = not\left(not(coeficiente) + 1\right)$$

Una vez que el contador módulo 64 ha alcanzado su máximo valor, se activa una señal auxiliar de un solo bit, denominada *eob*, la cual indica que se ha terminado de definir las categorías y la longitud de series de ceros, de los coeficientes no nulos, de un bloque de 8x8 pixeles. Esta señal indica, por lo tanto, que un fin de bloque ha sucedido, lo que implica también, que no se ha detectado otro coeficiente no nulo en el resto del bloque de 8x8 pixeles.

Cuando la señal de *reset* se desactiva, otro contador empieza a correr, y es usado para contar el número de pixeles procesados. Cuando el contador alcanza el número total de pixeles que posee la imagen procesada, que en este caso son 19200 pixeles (imagen de 160x120), se activa el puerto de salida *end_jpeg* para indicar que los datos presentes en los puertos *word_b1*, *run* y *category* corresponden al último pixel de la imagen.

El puerto de salida *rdy* indica si los datos en los puertos *word_b1*, *run* y *category* son válidos, y es activado por cualquiera de las siguientes circunstancias:

- Cuando se detecta un coeficiente no nulo
- Cuando se identifica el coeficiente DC
- Cuando se detecta el fin de bloque a través de la señal auxiliar *eob*

Los procedimientos de definición de categoría y longitud de series de ceros, están sincronizados (mediante registros) de tal forma que siempre coinciden a la salida, la categoría y la serie de ceros del mismo coeficiente.

2.5.4.2 Segundo componente: FIFOs de almacenamiento temporal

El segundo componente que tiene el bloque de codificación de Huffman, es un conjunto de dos FIFOs, los cuales almacenan temporalmente los resultados de la definición de categoría, longitud de series de ceros y la palabra código B. Los FIFOs sirven, principalmente, para evitar que se pierda información de datos, en caso de que se tenga que procesar las palabras código de un coeficiente no nulo, con una longitud de series de ceros de más de 15 ceros consecutivos. Por ejemplo, si se tiene un coeficiente con 36 ceros que lo preceden y a continuación de éste se tiene otro coeficiente no nulo, la información de este último se perdería debido a que, mientras se definen las dos palabras código ZRL (palabra especial para longitud de series de ceros de 16), y la palabra código A con la categoría y la nueva longitud de series de ceros (menor a 16) del coeficiente no nulo, el último coeficiente no nulo pasaría desapercibido. Para evitar esta circunstancia, se insertaron los dos FIFOs, con lo cual si una condición como la del ejemplo anterior sucediera, se detiene la lectura del FIFO hasta finalizar el procesamiento de las palabras código del coeficiente en cuestión, evitando así la pérdida de información de algún coeficiente no nulo.

Los FIFOS fueron implementados mediante IP Cores. El primer FIFO almacena la categoría, la longitud de series de ceros y la bandera que indica el último pixel de la imagen (*end_jpeg*), concatenadas; este FIFO tiene una capacidad de 64 palabras de 12 bits y utiliza los LUTs como elementos de almacenamiento. El segundo FIFO almacena la palabra código B, tiene una capacidad de 64 palabras de 11 bits y utiliza los LUTs como elementos de almacenamiento. La señal de habilitación de escritura de ambos FIFOs está controlada por el puerto *rdy* del componente anterior. De esta forma, cada vez que se valida los resultados, automáticamente se almacenan en los FIFOs.

2.5.4.3 Tercer Componente: Máquina de estados de control de palabras ZRL

El tercer componente del bloque de codificación de Huffman es una máquina de estados. En la Tabla 2-12 se indican los puertos que tiene este componente del bloque de codificación de Huffman.

Tabla 2- 12. Puertos del tercer componente del bloque de codificación de Huffman

Puerto	Dirección	Longitud en bits	Valor activo	Descripción
clk	Entrada	N/A	N/A	Señal de reloj.
rst	Entrada	1	1L	Señal de *reset.*
run	Entrada	7	N/A	Longitud de series de ceros del coeficiente no nulo obtenido desde el FIFO.
category	Entrada	4	N/A	Categoría del coeficiente no nulo obtenido desde el FIFO.
word_b1	Entrada	11	N/A	Valor del coeficiente no nulo (palabra código B) obtenido desde el FIFO.
empty	Entrada	1	1L	Indica si los FIFOs están vacíos.
end_jpeg	Entrada	1	1L	Entrada de la bandera que indica el último pixel de la imagen, obtenida desde el FIFO.
end_jpeg_out	Salida	1	1L	Salida de la bandera que indica el último pixel de la imagen.
word_b2	Salida	11	N/A	Salida del valor del coeficiente no nulo (palabra código B).
size_word_b2	Salida	4	N/A	Longitud en bits del valor del coeficiente no nulo.
add	Salida	9	N/A	Dirección de la memoria ROM con los códigos de Huffman.
rd	Salida	1	1L	Habilita de lectura de los FIFOs.
ena_rom	Salida	1	1L	Habilita la lectura de la memoria ROM de los códigos de Huffman.

La máquina de estados controla la habilitación de lectura de los FIFOs, como también de la memoria ROM que almacena los códigos de Huffman (palabras código A). La principal función es definir las direcciones de la memoria ROM de las Tablas de Huffman en función de la categoría y longitud de series de ceros del coeficiente. También, la máquina de estados controla que la longitud de series de ceros de los coeficientes no sea mayor a 15, produciendo la dirección de la palabra código ZRL (que define una longitud de series de ceros igual a 16), en caso de encontrar una cadena de ceros mayor a 15. El esquema de direccionamiento para la memoria ROM, que almacena los códigos de Huffman, es el que se indica en la siguiente figura:

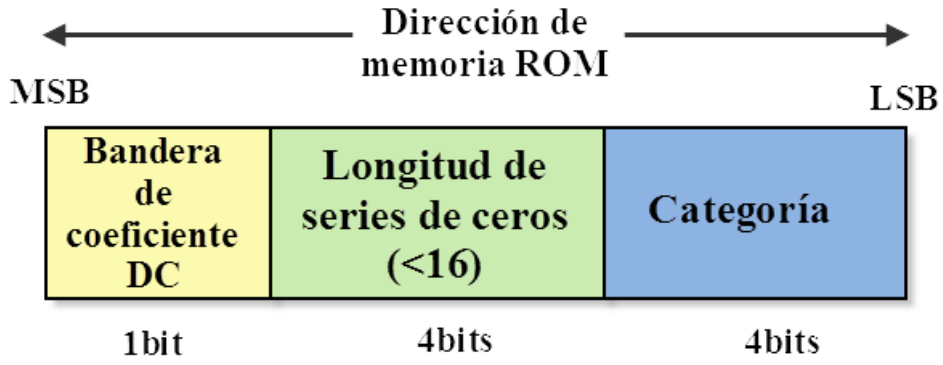


Figura 2- 50. Esquema de direccionamiento para la memoria ROM con las tablas de Huffman

La bandera de coeficiente DC, en la dirección de memoria, indica si el dato de la categoría, corresponde a un coeficiente DC (cuando es 1L). Como se indicó anteriormente, esta bandera está incluida en el bit más significativo de la señal de entrada *run*, la cual indica la longitud de series de ceros del coeficiente no nulo. La longitud de series de ceros, que se indica en la Figura 2-50, corresponde al resultado de extraer todas las cadenas de 16 ceros que se encuentren en la longitud de series de ceros de entrada, procedimiento que más adelante se detalla.

Con este esquema se garantiza la asignación de las palabras código para los coeficientes no nulos, acorde con la tabla de códigos que se recomienda en el estándar JPEG, tanto para los coeficientes DC como para los coeficientes AC.

En la Figura 2-51 se ilustra un diagrama ASM para la máquina de estados del bloque de codificación de Huffman. El procedimiento empieza cuando se ha detectado que un dato ha sido almacenado en los FIFOs, por medio del puerto de entrada *empty* (que está conectado al puerto de control *empty* de uno de los FIFOs). Esto produce la transición al estado *rd_fifo*, donde se activa la señal de habilitación de lectura de los FIFOs, a través del puerto de salida *rd*, el cual está conectado a los puertos de habilitación de lectura de ambos FIFOs. Después de esto, se hace la transición al estado *zrl0*, donde se verifica si la longitud de series de ceros es mayor a 15 ceros consecutivos.

El peor caso que se puede dar, en la longitud de series de ceros de un coeficiente no nulo, es si éste es el 64avo coeficiente (62 ceros consecutivos – el coeficiente DC no cuenta), con lo cual se tiene 3 cadenas de 16 ceros consecutivos. Pensando en este caso, como el peor, se definieron los estados: *zrl0, zrl1, zrl2, zrl3, zrl4*.

En el estado *zrl0*, primeramente, se habilita la lectura de la memoria ROM, a través del puerto de salida *ena_rom*, y si la longitud de series de ceros en el puerto de entrada *run* es menor a 16, se asigna a la señal de dirección de la ROM, *add*, el siguiente valor:

```
add <= run(6)& run (3 downto 0) & category;
```

Luego se verifica el estado de la señal *empty,* que indica si el FIFO está vacío; si esta señal es 0L se mantiene en el estado *zrl0*, caso contrario se hace la transición al estado *zero*.

Por el contrario, si el valor presente en el puerto *run* es mayor a 16, se asigna al puerto de dirección, el valor “011110000” que corresponde a la dirección de la palabra código ZRL, luego, se deshabilita la lectura de los FIFOs, y se hace la transición al estado *zrl1*.

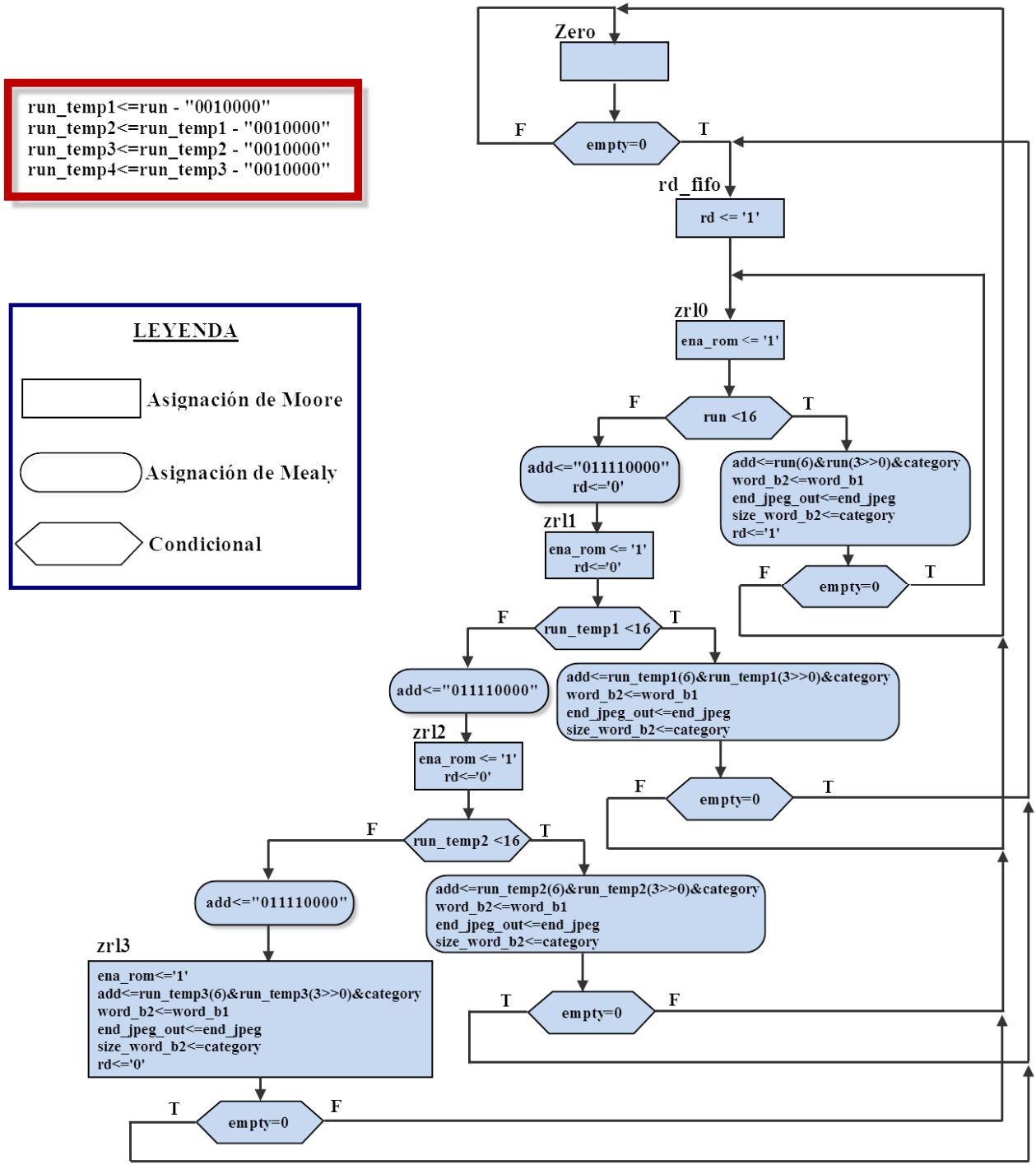


Figura 2- 51. Diagrama ASM de la máquina de estados del bloque de codificación de Huffman

En el estado *zrl1* se mantiene habilitada la memoria ROM de los códigos de Huffman, a través del puerto de salida *ena_rom*, y se verifica si la señal *run_temp1* es menor a 16. La señal *run_temp1* tiene asignada la diferencia de la longitud de series de ceros en el puerto *run* menos 16 (una cadena de 16 ceros); esta señal es calculada en el mismo instante en que ingresan los datos a través

del puerto *run*. Si el valor de la señal *run_temp1* es menor a 16, se asigna a la señal de dirección de la ROM, *add*, el siguiente valor:

```
add <= run_temp1 (6) & run_temp1 (3 downto 0) & category;
```

Luego se verifica el estado de la señal *empty* que indica si el FIFO está vacío; si esta señal es 0L se mantiene en el estado *rd_fifo*, caso contrario se hace la transición al estado *zero*.

Por el contrario, si el valor presente en la señal *run_temp1* es mayor a 16, se asigna al puerto de dirección, el valor “011110000” que corresponde a la dirección de la palabra código ZRL, y se hace la transición al estado *zrl2*. Se debe notar que en la permanencia en el estado *zrl1*, se mantuvo deshabilitada la lectura de los FIFOs.

En el estado *zrl2* se mantienen los valores de las señales de habilitación tanto de lectura de los FIFOs, así como de lectura de la ROM de los códigos de Huffman, a través de los puertos de salida correspondientes. Se verifica si la señal *run_temp2* es menor a 16. La señal *run_temp2* tiene asignada la diferencia de la longitud de series de ceros en el puerto *run* menos 32 (dos cadenas de 16 ceros); esta señal es calculada previamente. Si el valor de la señal *run_temp2* es menor a 16, se asigna a la señal de dirección de la ROM, *add*, el siguiente valor:

```
add <= run_temp2 (6) & run_temp2 (3 downto 0) & category;
```

Posteriormente, se verifica el estado de la señal *empty,* que indica si el FIFO está vacío; si esta señal es 0L se hace la transición al estado *rd_fifo*, caso contrario se hace la transición al estado *zero*.

Si el valor presente en la señal *run_temp2* es mayor a 16, se asigna al puerto de dirección, el valor “011110000” que corresponde a la dirección de la palabra código ZRL, y se hace la transición al estado *zrl3*.

En el estado *zrl3* se mantienen los valores de las señales de habilitación tanto de lectura de los FIFOs, como de lectura de la ROM de los códigos de Huffman, a través de los puertos de salida correspondientes. En este estado no se realiza ninguna verificación de la señal *run_temp3*, pues como se mencionó anteriormente, el peor caso de una cadena de ceros mayor a 16, es de 62 ceros consecutivos. La señal *run_temp3* tiene asignada la diferencia de la longitud de series de ceros en el puerto *run* menos 48 (tres cadenas de 16 ceros), por tanto se sabe que el valor de esta señal siempre será menor a 16 (14 en el peor de los casos). De esta manera, la dirección de la memoria ROM queda definida como:

```
add <= run_temp3 (6) & run_temp3 (3 downto 0) & category;
```

Finalmente, se verifica el estado de la señal *empty,* que indica si el FIFO está vacío; si esta señal es 0L se hace la transición al estado *rd_fifo*, caso contrario se hace la transición al estado *zero*.

En los cuatro estados (*zrl0, zrl1, zrl2, zrl3*), si la respectiva longitud de series de ceros (*run, run_temp1, run_temp2, run_temp3*) es menor a 16, los valores en los puertos de entrada *word_b1*, *end_jpeg* y *category* son transferidos (asignados) a los puertos de salida *word_b2*, *end_jpeg_out* y *size_word_b2* respectivamente, los cuales representan: al valor del coeficiente (palabra código B), la bandera que indica el último pixel de la imagen y el tamaño en bits del valor del coeficiente no nulo (palabra código B).

La salida de la máquina de estados *end_jpeg_out*, va conectada directamente con la salida *end_jpeg_out*, de la entidad de más alto nivel (la que contiene los componentes hasta ahora descritos – el bloque de codificación de Huffman).

2.5.4.4 Cuarto Componente: Memorias ROM de códigos de Huffman

El último componente del bloque de codificación de Huffman, consiste internamente de dos memorias ROM. En una de ellas se alberga los códigos de Huffman recomendados en el estándar JPEG, para la componente de luminancia.

En la otra memoria se almacena la longitud en bits de los códigos de Huffman. En la Tabla 2-13 se indican los puertos que tiene este componente del bloque de codificación de Huffman.

Tabla 2- 13. Puertos del cuarto componente del bloque de codificación de Huffman

Puerto	Dirección	Longitud en bits	Valor activo	Descripción
ena_rom	Entrada	1	1L	Señal de habilitación de lectura de las memorias ROM. Proviene de la máquina de estados.
add	Entrada	9	N/A	Dirección de las memorias ROM. Proviene de la máquina de estados.
word_b2	Entrada	11	N/A	Valor del coeficiente no nulo. Proviene de la máquina de estados.
size_word_b2	Entrada	4	N/A	Longitud en bits del coeficiente no nulo. Proviene de la máquina de estados.
rdy_cw	Salida	1	1L	Indica cuándo son válidos los datos en los puertos de salida.
word_b	Salida	11	N/A	Valor del coeficiente no nulo (palabra código B).
size_word_b	Salida	4	N/A	Longitud en bits del coeficiente no nulo.
size_word_a	Salida	5	N/A	Longitud en bits del código de Huffman.
word_a	Salida	16	N/A	Código de Huffman (palabra código A).

Cuando se habilita la lectura de la memoria ROM, a través del puerto *ena_rom* que es controlado por la máquina de estados, se realiza la lectura del contenido de las dos memorias ROM, especificado por la dirección presente en el puerto de entrada *add*, y que es generada por la máquina de estados. En la Figura 2-52 se ilustra un diagrama de bloques de este componente del bloque de codificación de Huffman. Como se observa en la figura, el procedimiento realizado para las obtenciones de la palabra código y su longitud, almacenadas en las memorias ROM, es completamente concurrente. Los contenidos de las memorias ROM se obtienen inmediatamente se habilita la lectura de las mismas (en teoría). En la práctica, se tiene el retardo de los niveles lógicos que se generen en la síntesis del bloque; sin embargo, este retardo no es significativo para la frecuencia de reloj de trabajo del módulo de desarrollo (50 MHz).

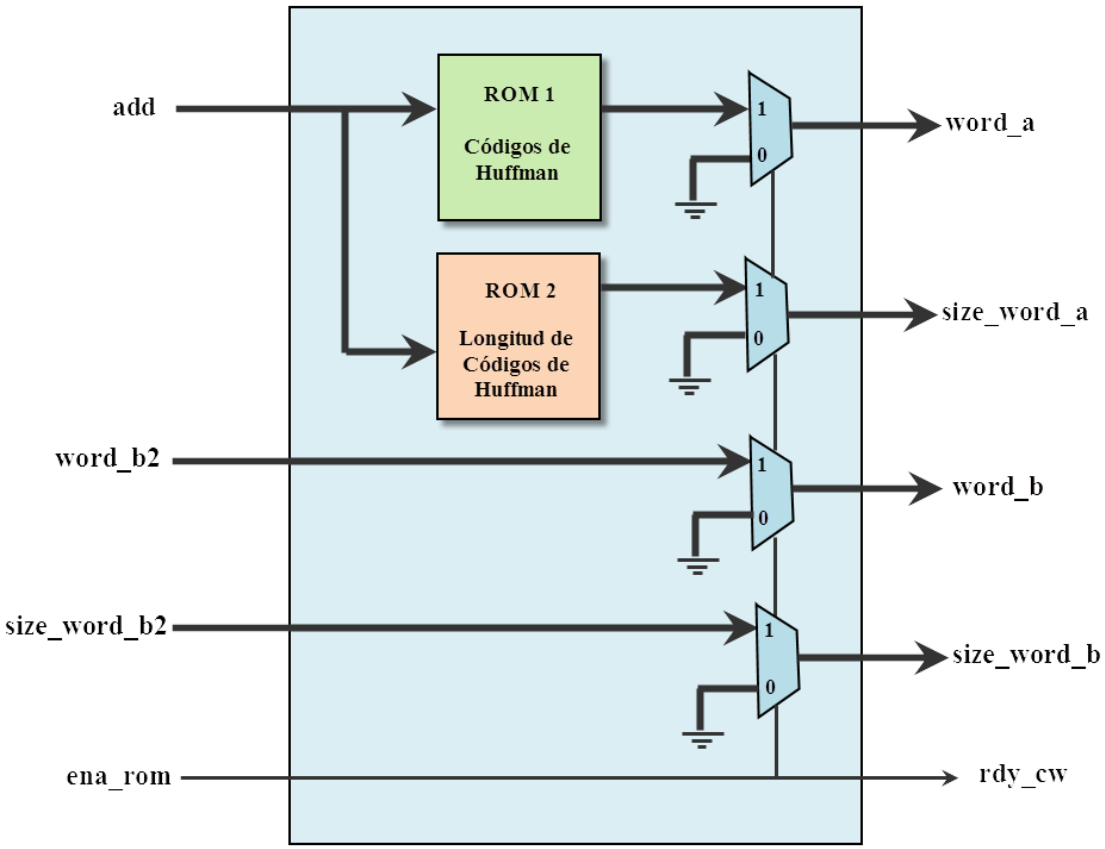


Figura 2- 52. Diagrama de bloques del componente de memoria para los códigos de Huffman

Los datos de las memorias ROM están organizados en dos bloques: en el primer bloque se encuentran almacenados los datos (códigos o longitud de códigos – dependiendo de la memoria) para los coeficientes AC, y en el segundo bloque los datos para los coeficientes DC. El bit más significativo del puerto de dirección establece de qué bloque hay que obtener los datos, como se indica en la Tabla 2-14.

Tabla 2- 14. Bloques de datos de las memorias ROM

Bit más significativo del puerto de dirección	Bloque de datos de las memorias ROM
0	Coeficientes AC
1	Coeficientes DC

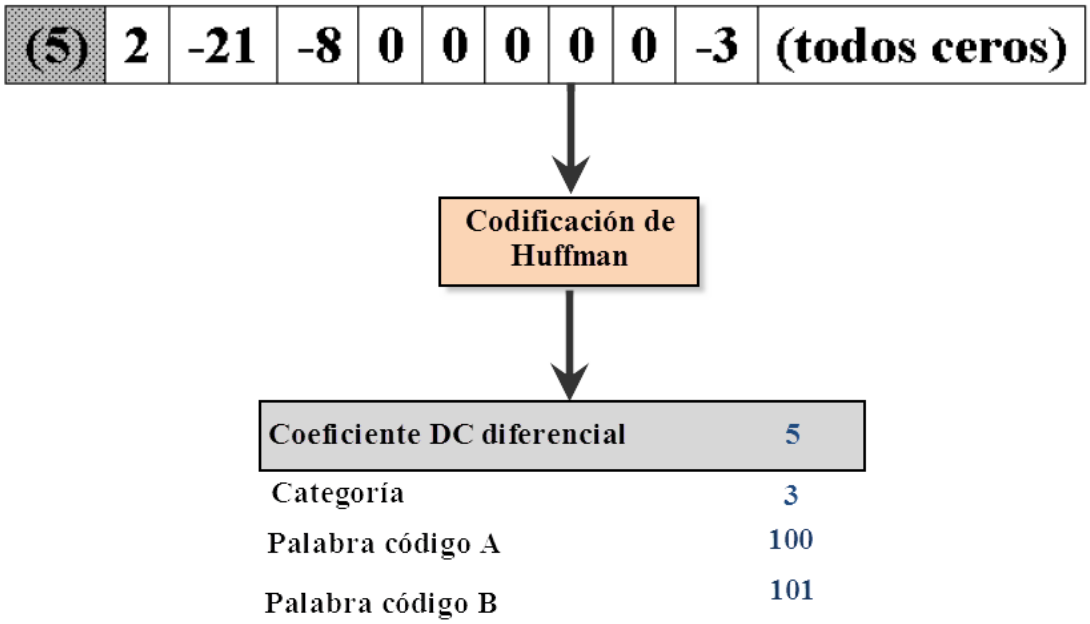
Los códigos de Huffman, para los coeficientes AC contenidos en la primera memoria ROM, están organizados en forma ascendente en función de la relación R/C (longitud de series de ceros y categoría), tal como se establecen en las tablas de códigos de Huffman, recomendadas en el estándar JPEG. Los códigos de Huffman para los coeficientes DC, están organizados de forma ascendente en función de la categoría del coeficiente. La longitud en bits, de los códigos de Huffman, sigue esta misma organización en la segunda memoria ROM.

Cuando se habilita la lectura de la memoria ROM, a través del puerto *ena_rom,* en el puerto de salida *word_a* se obtiene la palabra código para el coeficiente no nulo, y en el puerto *size_word_a* se obtiene la longitud en bits del código del puerto *word_a*. Simultáneamente, en el puerto de salida *word_b*, se asigna el valor del puerto de entrada *word_b2*, que corresponde al valor del coeficiente no nulo. En el puerto *size_word_b* se asigna el valor del puerto de entrada *size_word_b2*, que corresponde a la longitud en bits del valor del coeficiente no nulo. De esta manera se tiene en el puerto de salida *word_a*, la palabra código A, y en el puerto *word_b,* la palabra código B (conceptos discutidos en el Capítulo 1).

Los puertos de salida de este último componente, son asignados directamente a los correspondientes puertos de salida de la entidad de más alto nivel del bloque de codificación de Huffman y que se indican en la Tabla 2-10.

2.5.4.5 Simulación del bloque de codificación de Huffman

Para la simulación de este bloque, se han tomado como referencia los resultados teóricos de la codificación de Huffman de los coeficientes en disposición de zig-zag, que se obtuvieron en la sección anterior. Los resultados de esta codificación se muestran en la Figura 2-53.



Coeficiente DC diferencial	5
Categoría	3
Palabra código A	100
Palabra código B	101

Coeficientes AC	2	-21	-8	0 0 0 0 0 -3	todos ceros
R/C	0/2	0/5	0/4	5/2	EOB
Palabra código A	01	11010	1011	11111110111	1010
Palabra código B	10	01010	0111	00	---

Figura 2- 53. Resultados teóricos de referencia para la simulación del bloque de codificación de Huffman

En la Figura 2-54 se muestra el resultado de la simulación de este bloque. Cabe mencionar que la simulación se lleva a cabo conjuntamente con el bloque de la transformada discreta coseno, el bloque de cuantización y el bloque de exploración en zig-zag, para verificar la funcionalidad conjunta de estos bloques. Se puede observar la coincidencia de los resultados obtenidos de manera teórica y mediante el bloque de codificación de Huffman, diseñado en VHDL, para el coeficiente -3 que tiene una relación R/C de 5/2.

El código VHDL del bloque de codificación de Huffman, se encuentra debidamente comentado en el Anexo 11 de este proyecto.

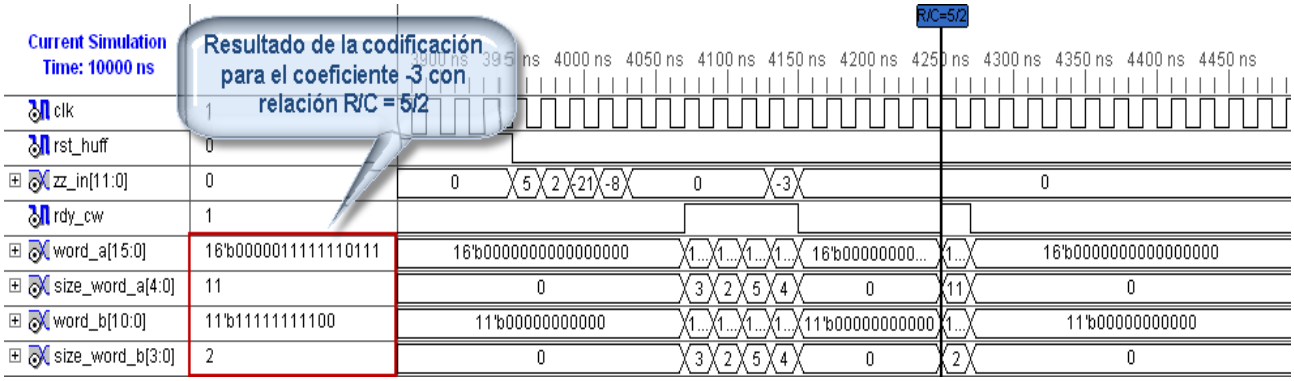


Figura 2- 54. Resultado de la simulación del bloque de codificación de Huffman

2.5.5 GENERADOR DE BYTES

En el bloque de codificación de Huffman, los resultados obtenidos son las palabras código A y B, las cuales son de longitud variable. El objetivo del bloque de generación de bytes, es agrupar la palabra código A y la palabra código B, solo con sus bits de datos, los cuales están explícitamente representados por la longitud de cada palabra código (que también es un resultado del bloque de codificación de Huffman), para dividir esa agrupación en bytes de datos, que serán parte del *stream* de bytes del archivo de imagen final.

En la Figura 2-55 se ilustra una representación de la entidad del bloque de generación de bytes y en la Tabla 2-15 se indican la dirección, la longitud en bits y el valor activo de los puertos que posee esta entidad.

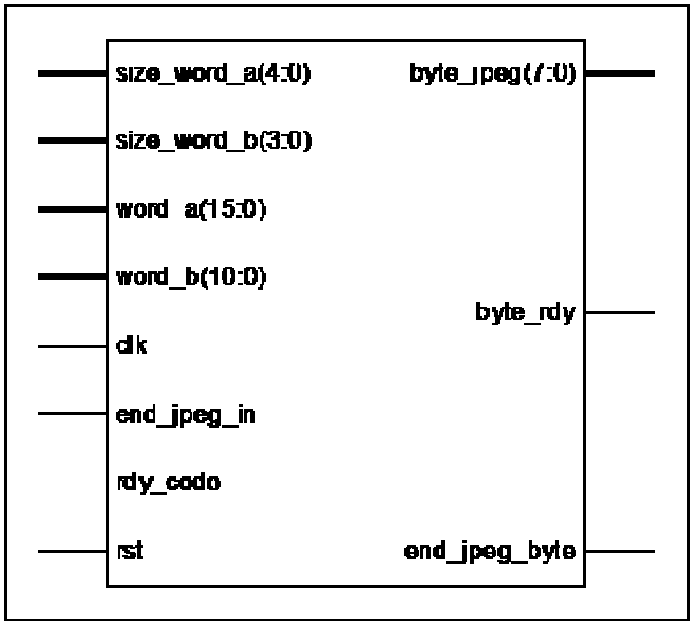


Figura 2- 55. Representación de la entidad VHDL del bloque de generación de bytes

Tabla 2- 15. Puertos del bloque de generación de bytes

Puerto	Dirección	Longitud en bits	Valor activo	Descripción
clk	Entrada	1	N/A	Señal de reloj.
rst	Entrada	1	1L	Señal de reset.
size_word_a	Entrada	5	N/A	Entrada de la longitud en bits de la palabra código A.
size_word_b	Entrada	4	N/A	Entrada de la longitud en bits de la palabra código B.
word_a	Entrada	16	N/A	Entrada de la palabra código A.
word_b	Entrada	11	N/A	Entrada de la palabra código B.
rdy_code	Entrada	1	1L	Indica cuándo se tiene una palabra código válida en los puertos de entrada *word_a* y *word_b.*
end_jpeg_in	Entrada	1	1L	Indica que las palabras código en los puertos *word_a* y *word_b* corresponden al último pixel de la imagen.
byte_jpeg	Salida	8	N/A	Salida del byte de dato de la imagen comprimida.
byte_rdy	Salida	1	1L	Indica cuándo se tiene un byte válido en el puerto *byte_jpeg.*
end_jpeg_byte	Salida	1	1L	Indica cuándo se tiene el último byte de dato de la imagen comprimida.

El proceso de generación de bytes empieza cuando se detecta, a través del puerto de entrada *rdy_code*, que un código válido ha ingresado al bloque en los puertos *word_a* y *word_b*. Lo primero que se hace en este bloque, es cancelar los bits que no son datos en la palabra código B (bits de relleno). Esto se logra

haciendo una operación AND entre la palabra código B y una constante de 11 bits, la cual indica con cada bit 1L los bits de datos de la palabra código B. El valor de la constante está en función de la longitud en bits de la palabra código B, en el puerto *size_word_b*, y proviene de una memoria ROM. Este procedimiento, para una palabra código B de 5 bits, se ilustra de manera gráfica en la Figura 2-56.

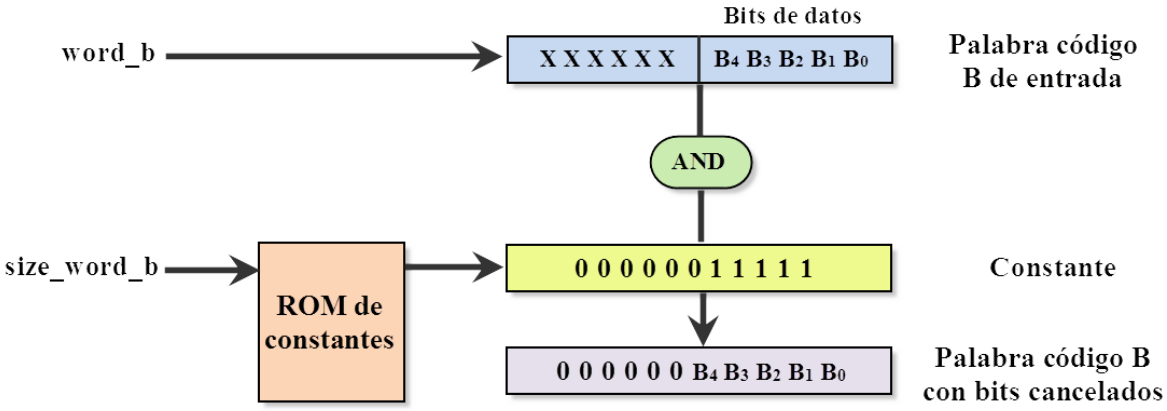


Figura 2- 56. Cancelación de los bits que no son datos en la palabra código B

El siguiente procedimiento es concatenar las palabras código una a continuación de la otra. Para esto, se asigna la palabra código A, a los 16 bits menos significantes de un registro de 27 bits, el cual después es desplazado a la izquierda un número de veces igual a la longitud en bits de la palabra código B. Con este registro se realiza una operación OR con la palabra código B (con los bits de relleno cancelados), de esta forma se tiene concatenadas las palabras código A y B sin bits de relleno. En la Figura 2-57 se ilustra este procedimiento.

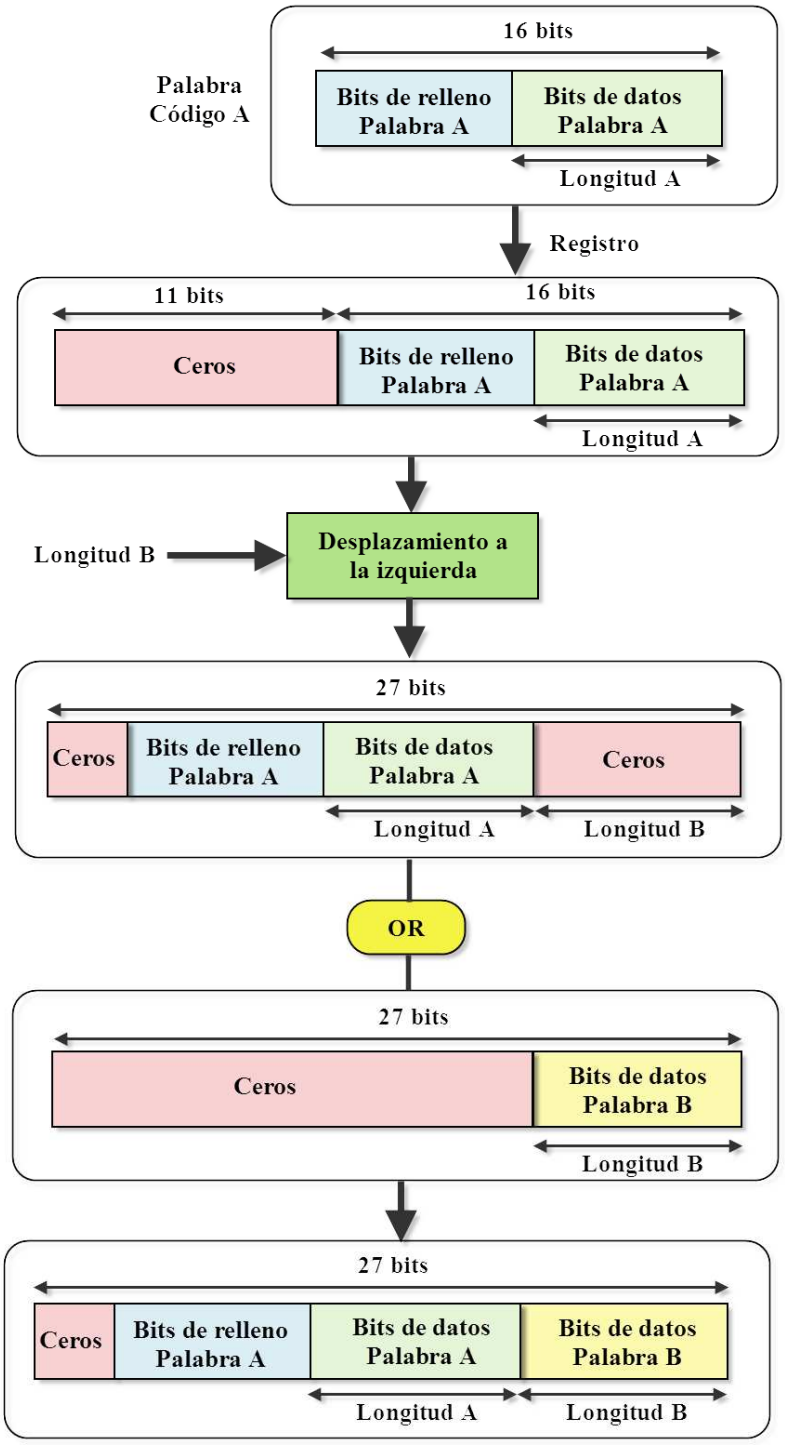


Figura 2- 57. Proceso de concatenación de las palabras código

Cuando se tiene la palabra código A concatenada con la palabra código B, se debe de alinear esta concatenación al bit más significativo del registro de 27 bits, para así eliminar los bits de relleno de la palabra código A y los posibles ceros del

desplazamiento anterior. Esto se logra, desplazando el registro a la izquierda el número de bits restantes que no son datos (27 menos la longitud de la palabra código A y B). Después de realizar este desplazamiento, el resultado se registra para posteriormente comenzar a generar los bytes de datos de la imagen comprimida. En la Figura 2-58 se indica el proceso antes mencionado.

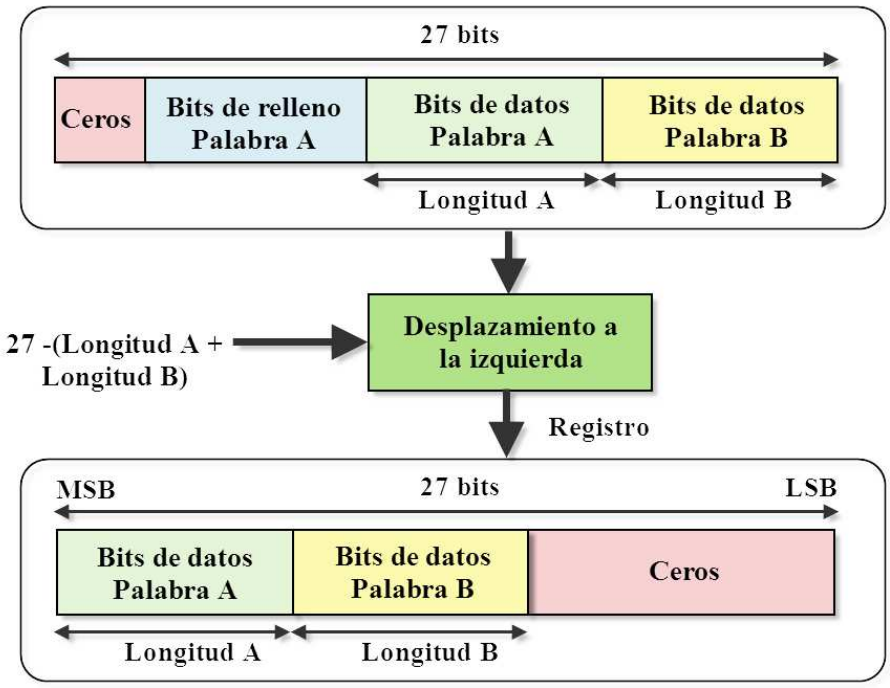


Figura 2- 58. Alineación de las palabras código al MSB

Los datos de este registro, conjuntamente con la longitud total de los bits de datos (longitud A + longitud B) y la bandera del último pixel (*end_jpeg_in*), son almacenados en un FIFO. La señal de habilitación de escritura del FIFO, está controlada por el puerto *rdy_code* que indica cuándo se tiene una palabra código válida en los puertos de entrada *word_a* y *word_b*.

La lectura de este FIFO está controlada por una máquina de estados, la cual es la encargada de generar los bytes de datos propiamente dicho. Cuando en la máquina de estados se detecta que un dato se encuentra almacenado, a través de la señal *empty* del FIFO, la máquina de estado activa la señal de habilitación de lectura. Antes de que los datos ingresen a la máquina de estados para la generación de los bytes, se hace un arreglo de los datos nuevos con los datos de

residuo que se hayan generado en el procesamiento previo (en caso de que existan). En un procesamiento previo, la máquina de estados enviará dos datos que deben ser incorporados a los nuevos datos de ingreso, estos son: los bits de datos que no alcancen a generar un byte, y cuántos bits son.

Los nuevos datos, conformados por las palabras código A y B (el registro de 27 bits), son asignados como los bits más significativos de un registro de 35 bits. Luego, este registro se desplaza a la derecha el número de bits de los datos de residuo que indica la máquina de estados. Con el registro desplazado, se hace una operación OR con los datos de residuo, los cuales ocupan los bits más significativos de una señal de 35 bits que proviene de la máquina de estados. Este nuevo resultado es el que ingresa a la máquina de estados para generar los bytes, conjuntamente con el número total de bits de datos (que son los bits de residuo y los bits de las nuevas palabras código). El procedimiento anterior se ilustra en la Figura 2-59.

En la Tabla 2-16 se muestran los puertos que posee la máquina de estados para la generación de los bytes.

El diagrama ASM de la máquina de estados que genera los bytes, se muestra en la Figura 2-60. Cuando se detecta que un nuevo dato ha sido almacenado en el FIFO de interfaz, a través del puerto de entrada *fifo_empty*, la máquina de estados habilita la señal de lectura del FIFO. Cuando los nuevos datos (*data_new*) y la longitud en bits de estos (*length_total*) ingresan a la máquina de estados, ésta empieza a verificar cuántos bytes se puede generar con éstos.

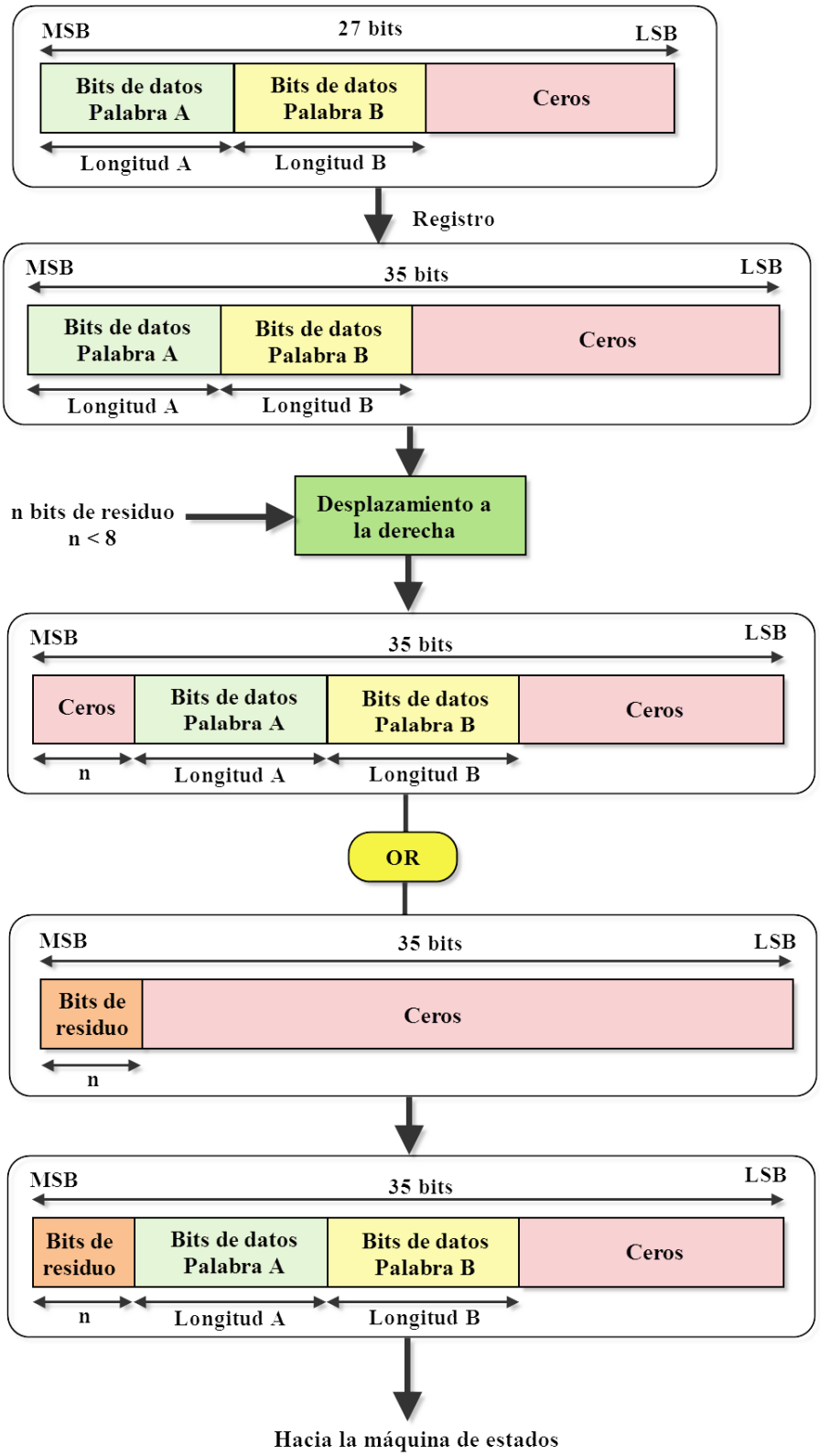


Figura 2- 59. Datos de ingreso para la máquina de estados

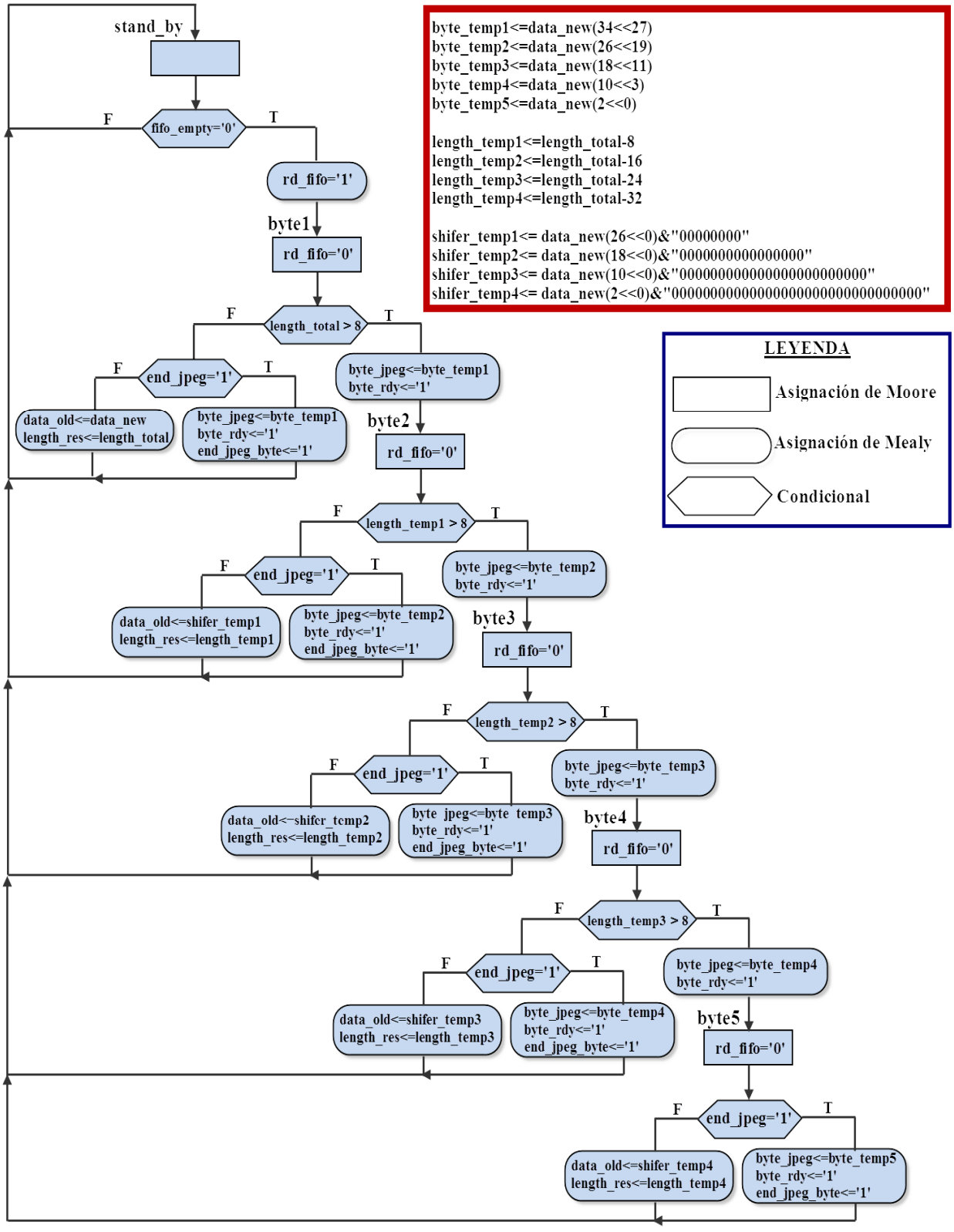


Figura 2- 60. Diagrama ASM de la máquina de estados del generador de bytes

Tabla 2- 16. Puertos de la máquina de estados para la generación de bytes

Puerto	Dirección	Longitud en bits	Valor activo	Descripción
clk	Entrada	1	N/A	Señal de reloj.
rst	Entrada	1	1L	Señal de reset.
fifo_empty	Entrada	1	1L	Indica si el FIFO se encuentra vacío.
data_new	Entrada	35	N/A	Entrada de nuevos datos (bits de residuo y palabras código).
length_total	Entrada	6	N/A	Número total de bits de datos en el puerto *data_new* (bits de residuo y palabras código).
end_jpeg	Entrada	1	1L	Entrada de bandera para el último pixel de la imagen.
fifo_rd	Salida	1	1L	Habilita la lectura del FIFO.
data_old	Salida	35	1L	Bits de datos de residuo.
length_res	Salida	6	N/A	Indica el número de bits de residuo.
byte_jpeg	Salida	8	N/A	Byte de datos de la imagen comprimida.
rdy_byte	Salida	1	1L	Indica cuándo se tiene un byte válido en el puerto *byte_jpeg*.
end_jpeg_byte	Salida	1	1L	Indica cuándo se tiene el último byte de dato de la imagen comprimida.

Cuando ingresa un dato nuevo a la máquina de estados, simultáneamente se crean las siguientes señales, para el desarrollo del algoritmo mostrado en la Figura 2-60:

- *byte_temp1*: a esta señal se asignan los 8 bits más significativos del puerto de entrada *data_new*.
- *byte_temp2*: a esta señal se asignan los 8 bits más significativos del puerto de entrada *data_new* desplazado 8 posiciones a la izquierda.
- *byte_temp3*: a esta señal se asignan los 8 bits más significativos del puerto de entrada *data_new* desplazado 16 posiciones a la izquierda.
- *byte_temp4*: a esta señal se asignan los 8 bits más significativos del puerto de entrada *data_new* desplazado 24 posiciones a la izquierda.
- *byte_temp5*: a esta señal se asignan los 8 bits más significativos del puerto de entrada *data_new* desplazado 32 posiciones a la izquierda.

- *length_temp1*: a esta señal se asigna el número de bits de los datos de entrada (*length_total*) menos 8.
- *length_temp2*: a esta señal se asigna el número de bits de los datos de entrada (*length_total*) menos 16.
- *length_temp3*: a esta señal se asigna el número de bits de los datos de entrada (*length_total*) menos 24.
- *length_temp4*: a esta señal se asigna el número de bits de los datos de entrada (*length_total*) menos 32.
- *shifter_temp1*: a esta señal se asigna el puerto de entrada *data_new* desplazado 8 posiciones a la izquierda.
- *shifter_temp2*: a esta señal se asignan los datos del puerto de entrada *data_new* desplazado 16 posiciones a la izquierda.
- *shifter_temp3*: a esta señal se asignan los datos del puerto de entrada *data_new* desplazado 24 posiciones a la izquierda.
- *shifter_temp4*: a esta señal se asigna los datos del puerto de entrada *data_new* desplazado 32 posiciones a la izquierda.

En el peor de los casos, los nuevos datos no superan los 34 bits (27 bits de las palabras código y 7 bits de residuo). Tomando en cuenta esto, se definieron cinco estados: *byte1*, *byte2*, *byte3*, *byte4* y *byte5*.

En el estado *byte1* se analiza el dato de ingreso a la máquina de estados; si el número de bits de datos, representados en el puerto *length_total* es mayor a 8, se activa la señal *byte_rdy*, luego se asigna el byte de dato de la señal *byte_temp1* al puerto *byte_jpeg* y se realiza la transición al estado *byte2.* Si el número de bits de datos es menor a 8, se analiza la bandera *end_jpeg* para verificar si los datos corresponden al último pixel de la imagen. Si la bandera es 0L, los datos que ingresaron en los puertos *data_new* y *length_total* son asignados a los puertos de salida de los bits de residuo y de la longitud de éstos, respectivamente (los puertos *data_old* y *length_res*), para luego volver al estado inicial (*stand_by*); si la bandera es 1L, se activa la señal *byte_rdy*, se asigna al puerto *byte_jpeg* la señal *byte_temp1* (aunque no todos son bits de datos) y se activa la bandera

end_jpeg_byte para indicar que es el último byte de datos de la imagen comprimida. Luego se hace la transición al estado inicial.

En el estado *byte2* se analiza la señal *length_temp1*, la cual tiene asignado el número de bits del dato de entrada (*length_total*) menos 8; si el número de bits de datos, indicados por esta señal, es mayor a 8, se activa la señal *byte_rdy*, luego se asigna el byte de dato de la señal *byte_temp2* al puerto *byte_jpeg*, y se realiza la transición al estado *byte3.* Si el número de bits de datos es menor a 8, se analiza la bandera *end_jpeg*. Si la bandera *end_jpeg* no está activada, se asigna a los puertos de salida de los bits de residuo y de la longitud de éstos, el dato de entrada (*data_new*) desplazado 8 posiciones a la izquierda (la señal *shifter_temp1*) y la señal *lenght_temp1*, respectivamente, y se hace la transición al estado inicial. Si la bandera está activada, se activa la señal *byte_rdy*, luego se asigna al puerto *byte_jpeg* la señal *byte_temp2* (aunque no todos los bits de esta señal sean de datos) y se activa la bandera *end_jpeg_byte*, para después hacer la transición al estado inicial.

En el estado *byte3* se analiza la señal *length_temp2*, la cual tiene asignado el número de bits del dato de entrada (*length_total*) menos 16; si el número de bits de datos indicados por esta señal, es mayor a 8, se activa la señal *byte_rdy*, luego se asigna el byte de dato de la señal *byte_temp3* al puerto *byte_jpeg* y se realiza la transición al estado *byte4.* Si el número de bits de datos es menor a 8, se analiza la bandera *end_jpeg*. Si la bandera *end_jpeg* no está activada, se asigna en los puertos de salida de los bits de residuo y de la longitud de éstos, el dato de entrada (*data_new*) desplazado 16 posiciones a la izquierda (la señal *shifter_temp2*) y la señal *lenght_temp2*, respectivamente, y se vuelve al estado inicial. Si la bandera está activada, se activa la señal *byte_rdy*, luego se asigna al puerto *byte_jpeg* la señal *byte_temp3* (aunque no todos los bits de esta señal sean de datos) y se activa la bandera *end_jpeg_byte* para indicar que es el último byte de datos de la imagen comprimida. Luego se hace la transición al estado inicial.

En el estado *byte4* se analiza la señal *length_temp3*, la cual tiene asignado el número de bits del dato de entrada (*length_total*) menos 24; si el número de bits de datos indicados por esta señal, es mayor a 8, se activa la señal *byte_rdy*, luego se asigna el byte de dato de la señal *byte_temp4* al puerto *byte_jpeg*, y se realiza la transición al estado *byte5.* Si el número de bits de datos es menor a 8, se analiza la bandera *end_jpeg*. Si la bandera *end_jpeg* no está activada, se asigna en los puertos de salida de los bits de residuo y de la longitud de éstos, el dato de entrada (*data_new*) desplazado 24 posiciones a la izquierda (la señal *shifter_temp3*) y la señal *lenght_temp3,* respectivamente, y se vuelve al estado inicial. Si la bandera está activada, se activa la señal *byte_rdy*, luego se asigna al puerto *byte_jpeg* la señal *byte_temp4* (aunque no todos los bits de esta señal sean de datos) y se activa la bandera *end_jpeg_byte* que indica el último byte de datos de la imagen comprimida. Luego se hace la transición al estado inicial.

En el estado *byte5* sólo se analiza la bandera *end_jpeg*, que indica el último pixel de la imagen comprimida, ya que se sabe de antemano que, si se llega a este estado, el número de bits de datos sobrante es siempre menor a 8, es decir, se trata de bits de residuo. Si la bandera *end_jpeg* no está activada, se asigna a los puertos de salida de los bits de residuo y de la longitud de éstos, el dato de entrada (*data_new*) desplazado 32 posiciones a la izquierda (la señal *shifter_temp4*) y la señal *lenght_temp4,* respectivamente, y se vuelve al estado inicial. Si la bandera está activada, se activa la señal *byte_rdy*, luego se asigna al puerto *byte_jpeg* la señal *byte_temp5* (aunque no todos los bits de esta señal sean de datos) y se activa la bandera *end_jpeg_byte*, para después hacer la transición al estado inicial.

Hay que mencionar que sólo en el estado *stand_by* se realiza la lectura de los datos del FIFO de interfaz.

2.5.5.1 Simulación del bloque de generación de bytes

Para la simulación de este bloque se ha considerado los datos de los códigos de Huffman que se generaron en la sección anterior. En la Figura 2-61 se muestran

los resultados teóricos de la agrupación en bytes de las palabras código. En la Figura 2-62 se ilustran los resultados de la simulación de este bloque.

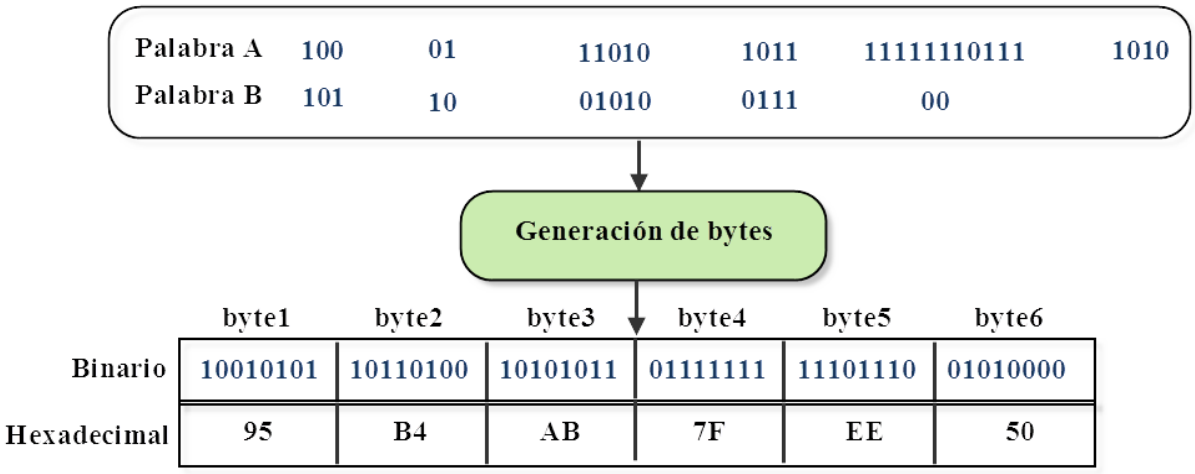


Figura 2- 61. Resultados teóricos de la generación de bytes

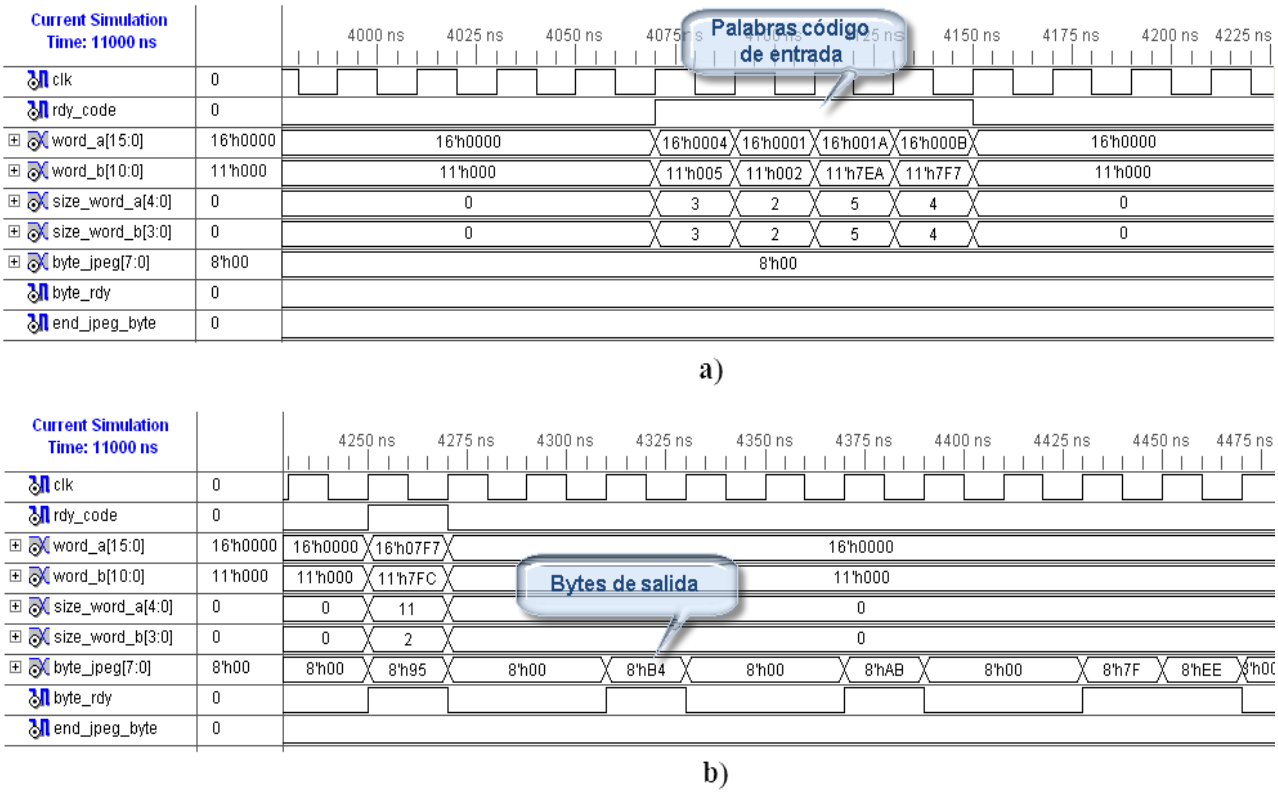


Figura 2- 62. Resultados de la simulación del bloque de generación de bytes: a) Ingreso de palabras código b) bytes generados

Como se observa en la Figura 2-62, los resultados de la simulación de la descripción VHDL del bloque de generación de bytes, coinciden plenamente con los resultados teóricos. Cabe mencionar que la simulación se lleva a cabo conjuntamente con el bloque de la transformada discreta del coseno, el bloque de cuantización, el bloque de exploración en zig-zag y el bloque de codificación de Huffman, para verificar la funcionalidad conjunta de estos bloques.

El código VHDL del bloque de generación de bytes, se encuentra debidamente comentado en el Anexo 11 de este proyecto.

2.5.6 BLOQUES ADICIONALES EN EL MÓDULO DE COMPRESIÓN

Adicional a los bloques que intervienen directamente en el esquema de codificación Baseline del estándar JPEG, se han diseñado dos bloques, los cuales se describen a continuación.

2.5.6.1 FIFO de almacenamiento de los bytes de la imagen comprimida

Los bytes de datos de la imagen comprimida, generados por el bloque de generación de bytes, se almacenan en un FIFO especial, para tener disponible el *stream* de estos bytes y conformar el archivo de imagen JFIF. El FIFO ha sido implementado mediante un IP Core con las siguientes características:

- Utilización de bloques de RAM embebida.
- Capacidad de 8192 bytes
- Contador de datos
- *Reset* asincrónico

Para escoger la capacidad del FIFO (entre las opciones disponibles), se ha considerado que se obtendrá una tasa de compresión mínima de 2:1, lo que equivale a una capacidad de 9600 bytes. El contador de datos es una salida especial del FIFO que indica cuántos datos se encuentran almacenados en éste.

Esta información es la que se envía al módulo de control, cuando se indica la finalización del proceso de compresión a través de la bandera *end_jpeg_byte*.

La escritura en el FIFO está controlada por el puerto de salida *byte_rdy* del módulo de generación de bytes; esto hace que automáticamente un byte de la imagen comprimida se almacene en el FIFO una vez que se haya generado.

El FIFO posee una pequeña máquina de estados, la cual realiza dos funciones:

- Cuenta los bytes de datos con valor xFF para enviar ese número al módulo de control, y
- Evita que se escriban en el FIFO datos espurios, una vez que haya terminado el proceso de compresión y generación de bytes[22].

El Diagrama ASM de la máquina de estados que controla la escritura del FIFO se indica en la Figura 2-63.

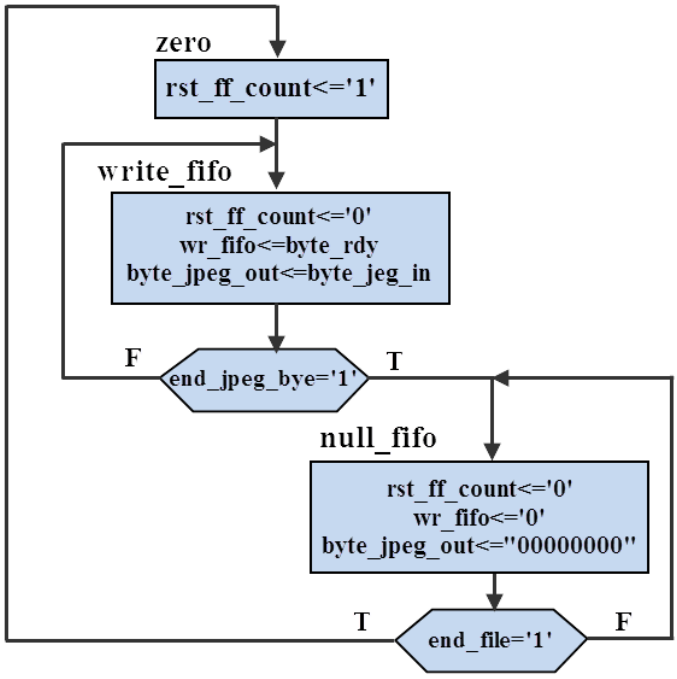


Figura 2- 63. Diagrama ASM de la máquina de estados de control del FIFO de almacenamiento

[22] Después del último byte de datos de la imagen comprimida, se siguen generando valores en los bloques del módulo de compresión (DCT, cuantización, exploración en zig-zag y codificación de Huffman) hasta que se vuelva a activar la señal de *reset* de estos bloques.

Como se puede observar en la Figura 2-60, el proceso de control del FIFO de almacenamiento, empieza en el estado *zero*, donde se habilita la señal de *reset* del contador de bytes con valor xFF (señal *rst_ff_count*), luego de lo cual se hace la transición al estado *write_fifo*. En este estado, se deshabilita la señal de *reset* del contador de bytes xFF, y el control de escritura del FIFO está dado por la señal *byte_rdy*, que proviene del generador de bytes. En el estado *write_fifo*, los datos de entrada al FIFO provienen del puerto de salida *byte_jpeg* del bloque de generación de bytes.

Cuando se detecta el final de la generación de los bytes de datos, a través de la bandera *end_jpeg_byte*, se hace la transición al estado *null_fifo*. En el estado *null_fifo*, se deshabilita permanentemente la señal de escritura en el FIFO, asignando un 0L al puerto *wr_fifo*, que está conectado al puerto de control de escritura del FIFO, evitando que cualquier otro byte, que no sea dato, se almacene en éste. En este estado se permanece hasta que se detecte que la bandera *end_file*, proveniente del módulo de control, se active, luego de lo cual se hace la transición al estado inicial *zero*.

Cuando se detecta que el byte de dato que ingresa al FIFO de almacenamiento, tiene el valor xFF, se incrementa en uno un contador. El dato del contador es enviado al puerto *ff_count* del módulo de control, para que conjuntamente con el número de bytes que almacena el FIFO y los bytes de la cabecera JFIF, se calcule el tamaño en bytes del archivo de imagen.

2.5.6.2 Memoria ROM de la cabecera JFIF

Para cumplir con el formato de archivo JFIF explicado en el Capítulo 1, es necesario almacenar en una memoria ROM, los marcadores que definen la información necesaria, para poder decodificar la imagen comprimida. Como se muestra en la Figura 2-64, el archivo de imagen está dividido, por los marcadores, en varios segmentos de datos, que albergan información relacionada con los parámetros de compresión.

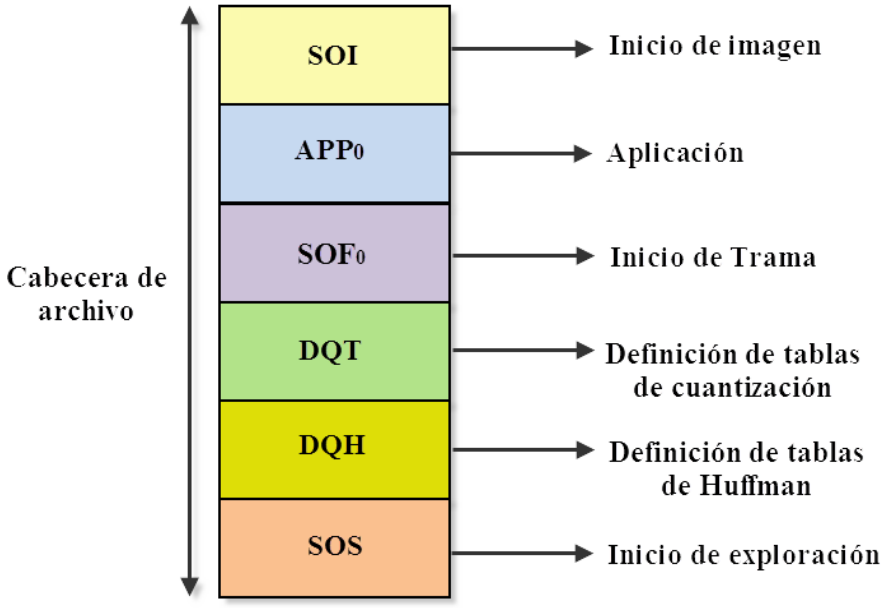


Figura 2- 64. Cabecera de archivo de imagen con formato JFIF

En la memoria ROM se almacena la cabecera de archivo, en la cual se encuentra información sobre los siguientes marcadores:

- Marcador de inicio de imagen (SOI)
- Marcador de aplicación para JFIF (APP_0)
- Marcador de inicio de Trama (SOF_0)
- Marcador de definición de tablas de cuantización (DQT)
- Marcador de definición de Tablas de Huffman para los coeficientes AC y DC (DHT)
- Marcador de inicio de exploración (SOS)

Los marcadores y sus datos, son almacenados en la memoria ROM cumpliendo con los formatos respectivos de cada marcador, que se indicaron en el Capítulo 1. Los valores de los marcadores y sus datos, para el caso de una imagen monocromática, se encuentra detallado en el Anexo 9 de este proyecto.

Debido a que se tienen cuatro diferentes tablas de cuantización, se requirió almacenar en la memoria ROM, cuatro cabeceras de archivo (una por cada

tabla). La selección de la cabecera que será transmitida, está definida por un mecanismo similar al utilizado en el bloque de cuantización, y que consiste en definir la cabecera de archivo con los dos bits más significativos del bus de direcciones de la memoria ROM, los cuales están controlados por los dos *switches* de deslizamiento que posee el módulo de desarrollo.

La memoria ROM, con el contenido de las cabeceras de archivo, fue implementada mediante un IP Core con las siguientes características:

- Capacidad de 2048 bytes
- Utilización de bloques de RAM embebida
- Memoria ROM *single port*

2.6 MÓDULO DE TRANSFERENCIA Y ALMACENAMIENTO DE ARCHIVOS

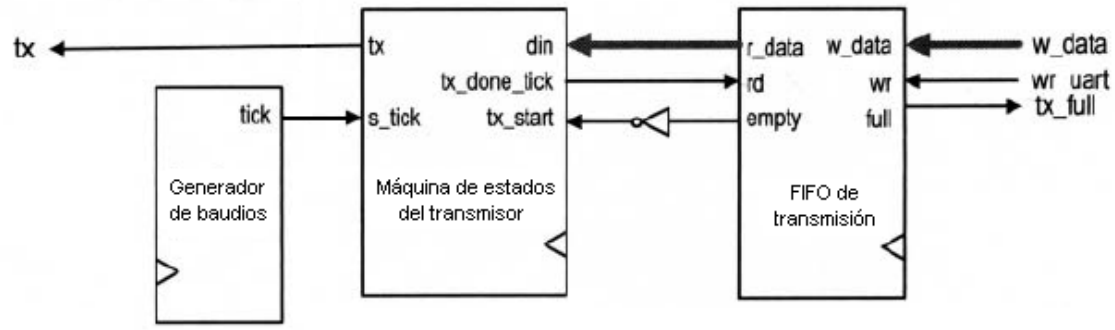
El módulo de transferencia y almacenamiento de archivos posee una sección de hardware y una sección de software. La sección de hardware es el subsistema de transmisión UART, con el cual se hará la transmisión por el puerto serial, desde el módulo de desarrollo, al computador que ejecute el programa de la interfaz gráfica de control. La sección de software consta de un conjunto de instrucciones en Matlab para recibir el *stream* de bytes desde el módulo de desarrollo y dar la extensión de archivo de imagen .jpg.

2.6.1 SECCIÓN DE HARDWARE DEL MÓDULO DE TRANSFERENCIA Y ALMACENAMIENTO DE ARCHIVOS

2.6.1.1 Sistema UART: subsistema de transmisión

En la Figura 2-65 se muestran las epatas del subsistema de transmisión. El subsistema de transmisión UART es similar al subsistema de recepción, aunque algunas condiciones cambian. A pesar que este subsistema también cuenta con

un generador de tasa de baudios, una máquina de estados del transmisor y un FIFO de transmisión, el comportamiento de los dos primeros difiere con su par en el subsistema de recepción.



Fuente [3]

Figura 2- 65. Subsistema de transmisión UART

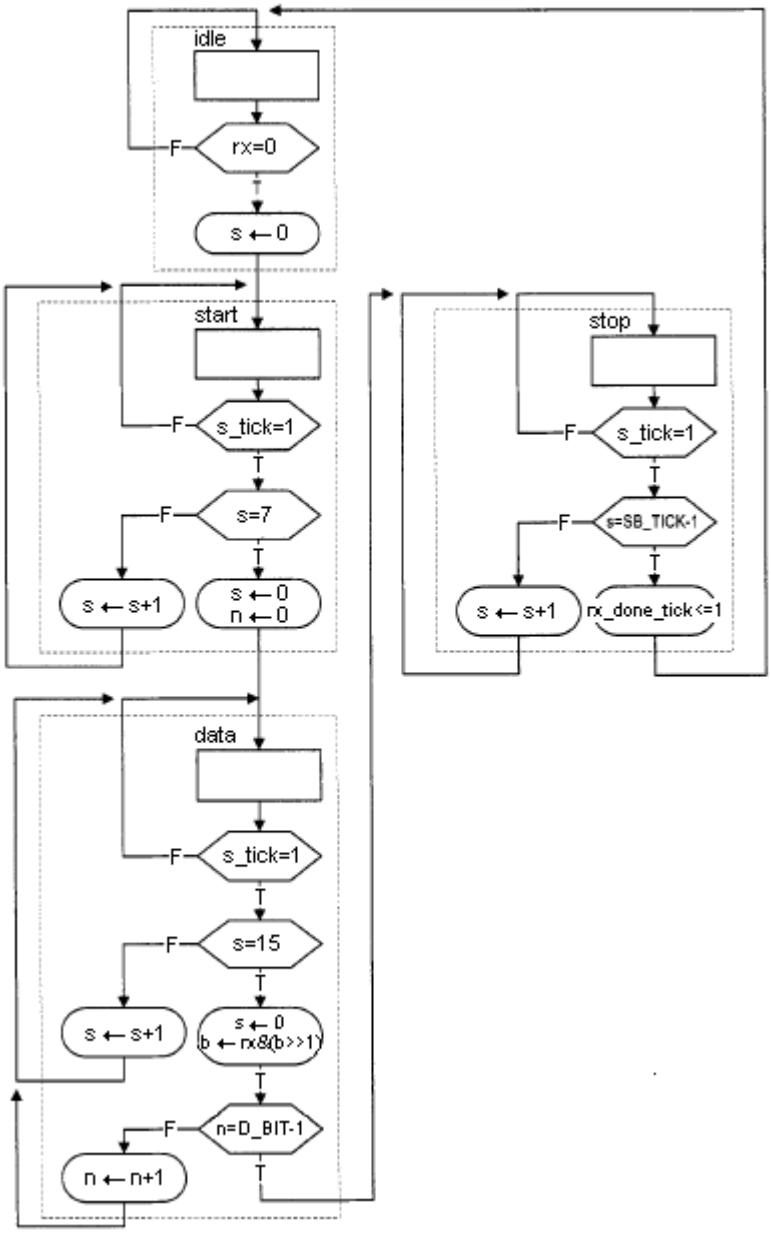
2.6.1.1.1 Generador de baudios para el transmisor

En el subsistema de transmisión no se necesita ningún método de muestreo como en el caso de la recepción de datos, es por eso que la señal de generación de baudios debe ser 16 veces más baja que en recepción. Aunque esta condición conlleva a crear un nuevo generador de baudios, se utiliza el mismo que en recepción, y se incluye el registro *s* en la máquina de estados de transmisión, que es utilizado como contador y que es habilitado por la señal *s_tick*; para la transmisión de un bit se requiere que el contador *s* llegue a 15 (contador mod-16).

2.6.1.1.2 Máquina de estados del transmisor

El diagrama ASM de la Figura 2-66 ilustra el comportamiento de la máquina de estados del transmisor. Como se observa en la figura, la máquina de estados del transmisor posee cuatro estados: *idle*, *start*, *data*, *stop*. En la descripción VHDL de la máquina de estados, se incluyen, al igual que en el caso de recepción, la constante *D_BIT*, que indica el número de bits de datos y la constante *SB_TICK* que indica el número de muestras necesarias para los bits de parada, que para el caso del diseño son 8 y 16, respectivamente.

La transición del estado *idle* al estado *start* solo se da cuando se detecta que el FIFO de transmisión no está vacío. Esto se logra chequeando si el valor de la señal *tx_start* es 1L, ya que a esta señal se le asigna la señal de estado *empty* del FIFO de transmisión, la cual es activa en alto.



Fuente [3]

Figura 2- 66. Diagrama ASM de la máquina de estados del transmisor

En la máquina de estados del transmisor se incluyen dos registros denominados *s* y *n*; el registro *s* cumple con la función que se indicó en el generador de baudios, y el registro *n* lleva la cuenta de los bits de datos que se transmiten. Además, se incluye un registro de desplazamiento denominado *b*, el cual recibe el dato a transmitir desde el FIFO de transmisión y desplaza hacia la derecha sus bits cada vez que uno de ellos es transmitido. En los estados *start*, *data* y *stop* se realiza la transmisión de los bits de inicio, datos y parada, respectivamente, a través de la señal de salida *tx.* El método de transmisión es el siguiente:

1. En el estado *start* se transmite el bit de inicio (0L) en la señal de salida *tx*, hasta que el registro *s* alcance el valor de 15, luego de lo cual se realiza la transición al estado *data.*
2. En el estado *data* se transmiten los bits de datos, para lo cual, se asigna cada bit del registro *b* a la señal de salida *tx* durante el tiempo que tarde el registro *s* en alcanzar las 15 muestras; cada vez que el registro *s* alcance el valor de 15, se incrementa el registro *n*, que cuenta el número de bits de datos transmitidos, y se desplaza el registro *b* una posición hacia la derecha. Cuando el registro *n* alcanza el valor de la constante D_BIT menos uno, es decir 7, se produce la transición al estado *stop*.
3. En el estado *stop* se transmite el bit de parada, para lo cual se asigna el valor 1L a la señal de salida *tx,* hasta que el registro *s* alcance el valor de la constante *SB_TICK* menos uno, es decir 15. Después de esto se produce la transición al estado inicial *idle*, hasta que se vuelva a escribir en el FIFO de transmisión.

Se incluye la señal *tx_done_tick*, la cual es puesta en 1L por un ciclo de reloj, después de que el proceso de transmisión ha finalizado (en el estado *stop*). Esta señal también sirve para habilitar la lectura de un nuevo dato del FIFO, para su posterior asignación en el registro *b* y volver a realizar una transmisión.

2.6.1.1.3 FIFO de transmisión

El FIFO de transmisión es el encargado de almacenar los bytes que se van a transmitir por el sistema UART. Este FIFO es creado mediante un *IP Core,* y sus características son similares al caso de recepción excepto en la capacidad del FIFO. La capacidad de este FIFO es 2048 palabras de 8 bits, por lo que en lugar de utilizar LUTs para su implementación se debe escoger los bloques de RAM. Esto se hace con el fin de transmitir la mayor cantidad de bytes antes de entrar en la condición *full* del FIFO, con la cual ya no se puede escribir en el FIFO hasta que se haya terminado de transmitir, al menos 1 byte.

2.6.1.1.4 Señales de interfaz del subsistema de transmisión UART

El subsistema de transmisión cuenta con cinco señales de interfaz: dos externas y tres internas. La señales externas que tiene el subsistema de transmisión, son *tx y clk*, las cuales están conectadas, a través de la entidad de más alto nivel de todo el sistema, a los pines del FPGA que conectan los pines 2 del puerto RS-232 (transmisión) del módulo de desarrollo y al oscilador del FPGA, respectivamente. Las señales internas de interfaz para la transmisión de datos son:

a. *tx_full*: es la señal que indica la condición *full* (lleno) de la memoria FIFO de transmisión.
b. *wr_uart*: es la señal de habilitación de escritura del FIFO de transmisión.
c. *w_data*: es el bus de 8 bits que lleva los datos que se desean escribir en el FIFO para su posterior transmisión.

2.6.2 SECCIÓN DE SOFTWARE DEL MÓDULO DE TRANSFERENCIA Y ALMACENAMIENTO DE ARCHIVOS

2.6.2.1 Recepción del *stream* de bytes de archivo de imagen y su almacenamiento

El proceso la recepción del *stream* de bytes del archivo de imagen, se lleva a cabo mediante los siguientes pasos:

- Cuando se envía el carácter ASCII “J”, para indicar el inicio de todo el proceso de compresión, el programa de la interfaz gráfica de control, en la siguiente línea, envía a través del puerto serial los datos de la imagen sin comprimir, ordenados como bloques de 8x8 pixeles; después, el programa espera recibir dos bytes provenientes del módulo de desarrollo. Los dos bytes corresponden al tamaño en bytes que tiene el archivo de imagen. Estos procesos se llevan a cabo mediante las siguientes instrucciones de Matlab:

```
fopen(s);
fwrite(s,'J','uint8');
fwrite(s,image_vector,'uint8');
total_bytes = fread(s,2,'uint8');
```

- Una vez que se ha calculado el tamaño total del archivo de imagen, el programa de la interfaz gráfica de control, envía a través del puerto serial el carácter ASCII “R” para indicar que se debe iniciar la transferencia del *stream* de bytes del archivo de imagen, desde el módulo de desarrollo al programa de la interfaz gráfica de control. Este proceso se hace mediante las siguientes instrucciones:

```
total_double = double(total_bytes);
MSB_total = dec2binvec(total_double(1));
LSB_total = dec2binvec(total_double(2),8);
total_binvec=[LSB_total MSB_total];
total = binvec2dec(total_binvec);
fwrite(s,'R','uint8');
```

- En una variable se almacenan los bytes recibidos por el puerto serial, hasta alcanzar el número de bytes especificados por los dos primeros bytes recibidos; esto se logra mediante la siguiente instrucción:

```
image_jpeg = fread(s,total,'uint8');
```

- El contenido de la variable que tiene el *stream* de bytes del archivo de imagen, es guardado en un archivo con extensión *.jpg*. Esto se logra mediante el siguiente conjunto de instrucciones:

```
image_trans = (image_jpeg)';
image_txt=fopen('image_jpeg.jpg','w');
fwrite(image_txt, image_trans);
fclose(image_txt);
```

El código fuente de la interfaz gráfica de control y de los módulos VHDL que posee este sistema, se muestran debidamente comentados, en el Anexo 10 y el Anexo 11, respectivamente.

2.7 LA ENTIDAD VHDL DE MÁS ALTO NIVEL DEL SISTEMA

Las distintas secciones de los módulos, que han sido descritos en VHDL, convergen en una sola entidad VHDL. Dicha entidad, tiene cada uno de los bloques VHDL descritos anteriormente como componentes (descripción estructural en VHDL) y mediante señales internas conecta estos bloques para que cumplan con la funcionalidad que se ha planteado para el sistema. Los puertos de entrada y salida de esta entidad, están conectados directamente con los pines del FPGA XC3S700A, y estos a su vez, a los periféricos del módulo de desarrollo utilizados por el sistema. En la Figura 2-67 se muestra la representación de la entidad VHDL de más alto nivel del sistema y en la Tabla 2-17 una descripción de los puertos de ésta.

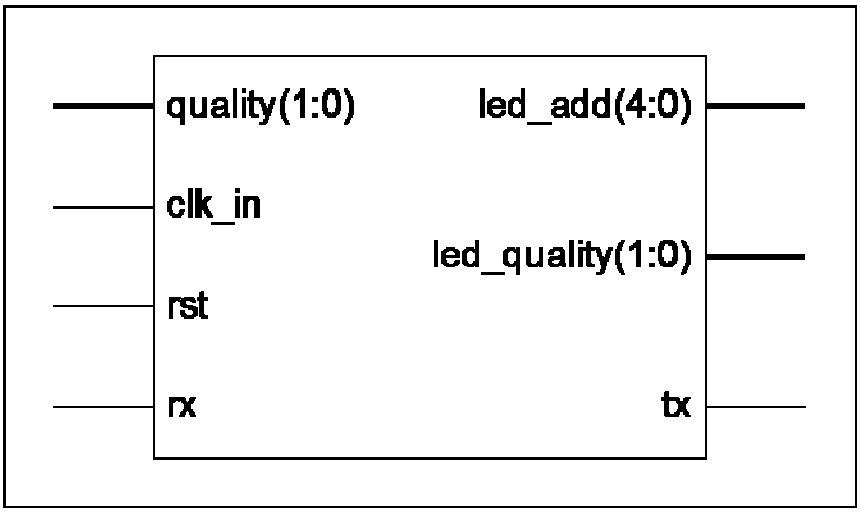


Figura 2- 67. Representación de la entidad VHDL de más alto nivel del sistema

Tabla 2- 17. Puertos de la entidad de más alto nivel del sistema

Puerto	Dirección	Longitud en bits	Valor activo	Descripción
clk_in	Entrada	1	N/A	Señal de reloj del sistema. Está conectada al oscilador del módulo de desarrollo.
rst	Entrada	1	1L	Señal de reset del sistema. Está conectada a un *switch* tipo pulsador del módulo de desarrollo.
rx	Entrada	1	N/A	Entrada de recepción UART. Está conectada al pin 3 del puerto RS-232 del módulo de desarrollo.
quality	Entrada	2	N/A	Selector del Factor de calidad de la imagen comprimida. Está conectado a dos *switches* de deslizamiento del módulo de desarrollo.
led_add	Salida	5	N/A	Indica la actividad de recepción de datos de la imagen sin comprimir. Están conectados a 5 leds del módulo de desarrollo.
led_quality	Salida	2	N/A	Indica el estado lógico de los selectores del factor de calidad. Están conectados a dos leds de módulo de desarrollo.
tx	Salida	1	N/A	Salida de transmisión UART. Está conectada al pin 2 del puerto RS-232 del módulo de desarrollo.

El puerto de salida *led_add* de la entidad de más alto nivel del sistema, indica la actividad de recepción de datos de la imagen sin comprimir, asignando los 5 bits más significativos de la señal de dirección de escritura del buffer temporal, tal como se indica en el siguiente código:

```
led_add <= address_ram_w(14 downto 10);
```

El puerto de salida led_quality indica el estado lógico de los selectores del factor de calidad de la imagen comprimida, asignando el mismo puerto de entrada *quality* a este puerto de salida.

REFERENCIAS

[1] Barragan, Diego. *Manual de interfaz gráfica de usuario en Matlab*. Universidad Técnica Particular de Loja, 2008.
Obtenido de:
http://www.mathworks.com/matlabcentral/fileexchange/12122?controller=file_infos&download=true
(Último acceso: 06/08/2010)

[2] Hass, Calvin. *JPEGsnoop - JPEG File Decoding Utility.* Descripción del software.
Obtenido de:
http://www.impulseadventure.com/photo/jpeg-snoop.html
(Último acceso: 16/08/2010)

[3] Chu, Pong**.** *FPGA Prototyping by VHDL Examples*. 1 ed. Editorial John Wiley & Sons Ltd, 2008.

[4] Pedroni, Volnei. *Circuit Design with VHDL*. Editorial MIT Press, 2004

[5] Chapman, Ken. *Get Smart About Reset: Think Local, Not Global*. Xilinx's white paper, 2008
Obtenido de:
http://www.xilinx.com/support/documentation/white_papers/wp272.pdf
(Último acceso: 12/08/2010)

[6] Madisetti, Vijay y Williams, Douglas. *Digital Signal Processing Handbook*. Editorial CRC Press, 1999.

[7] Pillai, Latha. Video Compression using DCT. Xilinx Application Note, 2002
Obtenido de:
http://www.cs.york.ac.uk/rts/docs/Xilinx-datasource-2003-q1/appnotes/xapp610.pdf
(Último acceso: 11/06/2010)

[8] Código VHDL para la implementación de la DCT-2D
Obtenido de:
ftp://ftp.xilinx.com/pub/applications/xapp/xapp610.zip
(Último acceso: 11/06/2010)

[9] Bernal, Iván. *Compresión de Imágenes con JPEG*. Quito: Escuela Politécnica Nacional, Septiembre 2004.

[10] Miano, John. *Compressed Image File Formats: JPEG, PNG, GIF, XBM, BMP*. 1 ed.. Reading, Massachusetts: Editorial ACM Press, 1999.

CAPÍTULO 3

IMPLEMENTACIÓN Y PRUEBAS DEL SISTEMA

3.1. GENERALIDADES

En este capítulo se describen los resultados de la síntesis e implementación de la entidad VHDL de más alto nivel, del sistema de adquisición, compresión y almacenamiento de imágenes. También se documentan los resultados de las imágenes comprimidas a través del sistema, en dos escenarios de prueba, y se hará la verificación de los parámetros de compresión, empleando la herramienta de software *JPEGsnoop*.

3.2. SÍNTESIS DE LA ENTIDAD VHDL DE MÁS ALTO NIVEL DEL SISTEMA

Siguiendo con los pasos del flujo de diseño en la herramienta de desarrollo ISE Foundation 10.1, se han obtenido los siguientes resultados, en la síntesis de la entidad VHDL de más alto nivel, del sistema de adquisición, compresión y almacenamiento de imágenes.

3.2.1. RECURSOS UTILIZADOS

En la Figura 3-1 se indican los recursos del FPGA utilizados para los módulos VHDL descritos en el Capítulo 2. Como se puede observar en la Figura 3-1, se han utilizado todos los 20 bloques de RAM embebida (*BRAM*); los bloques de memoria RAM fueron los principales limitantes para obtener un compresor JPEG de 3 componentes de una imagen mayor a 160x120 pixeles, como se discutió en el Capítulo 2.

También se utilizó 8 de los 20 multiplicadores embebidos, para los 8 multiplicadores de la DCT-2D.

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	3497	5888	59%
Number of Slice Flip Flops	4291	11776	36%
Number of 4 input LUTs	4094	11776	34%
Number of bonded IOBs	13	372	3%
Number of BRAMs	20	20	100%
Number of MULT18X18SIOs	8	20	40%
Number of GCLKs	1	24	4%
Number of DCMs	1	8	12%

Figura 3- 1. Recursos utilizados en el FPGA XC3S700A

Dentro de los recursos de sincronismo utilizados, se tiene una entrada global de reloj (GCLK[23]). Además, se utilizó un administrador digital de reloj (DCM[24]). El DCM no es un elemento esencial en el diseño de los módulos de hardware, y se utiliza únicamente con el fin de evitar el sesgado de reloj (*skew*), a través del bloque de lazo de seguimiento de retardo (DLL[25]), como se indicó en el Capítulo 1. Aunque a la frecuencia de operación de 50MHz, este problema (el *skew*) no es significativo, es una práctica de diseño común añadir un DCM para este fin.

En la implementación de los módulos de hardware, se utilizó el 34% de los LUTs disponibles en el FPGA, que corresponden a 4094 LUTs. Además, para los registros generados en la descripción VHDL de los módulos, se utilizó el 36% de los flip-flops disponibles, equivalente a 4291 flip-flops.

Se utilizaron 13 bloques de entrada/salida para las conexiones externas del FPGA, las cuales se detallan en el archivo de restricciones del usuario (UCF[26]), utilizado para la implementación del sistema.

[23] **GCLK**: Global Clock
[24] **DCM**: Digital Clock Manager
[25] **DLL**: Delay-Locked Loop
[26] **UCF**: User Constraints File

3.2.2. RETARDOS Y FRECUENCIA MÁXIMA

El informe sobre los valores de temporización es parte del reporte del proceso de síntesis, generado una vez que se ha finalizado la síntesis de la entidad VHDL de más alto nivel. El informe de temporización se indica a continuación:

```
Timing Summary:
---------------
Speed Grade: -4

   Minimum period: 12.092ns (Maximum Frequency: 82.702MHz)
   Minimum input arrival time before clock: 5.821ns
   Maximum output required time after clock: 6.111ns
   Maximum combinational path delay: 5.900ns
```

En el informe se establece, entre otros parámetros, la frecuencia máxima a la que puede operar la entidad VHDL de más alto nivel, en el FPGA XC3S700A. Para el diseño, con los recursos utilizados de la Figura 3-1, la máxima frecuencia a la que puede operar el sistema es de 82.7 MHz. Ya que el módulo de desarrollo *Spartan-3A Starter Kit*, posee un oscilador de 50 MHz y otro de 133.3 MHz, no hay ningún problema de sincronismo con el oscilador de 50 MHz, que es el utilizado para este proyecto. Cabe mencionar que los valores de frecuencia expuestos en el reporte del proceso de síntesis, pueden variar después de ejecutados los procesos de implementación (Translate, Map y Place and Route); por lo general estos valores disminuyen dependiendo del dispositivo. Para el dispositivo FPGA XC3S700A se obtuvo una frecuencia de operación real de 65.79 MHz.

En el informe también se detallan el retardo por lógica combinacional y los tiempos de *setup* y *hold* para el diseño. Se obtuvo un retardo de lógica combinacional (por los niveles de lógica) de 5.9ns, además, los tiempos de *setup* y *hold* de 5.821ns y 6.111ns, respectivamente.

3.3. IMPLEMENTACIÓN DE LA ENTIDAD VHDL DE MÁS ALTO NIVEL DEL SISTEMA

Siguiendo con los pasos del flujo de diseño en la herramienta de desarrollo ISE Foundation 10.1, una vez realizada la síntesis de la entidad VHDL de más alto nivel del sistema, se debe realizar los procesos de *Optimización*, *Translate*, *Mapping* y *Placement and Routing*. Para esto, se requiere un archivo de restricciones del usuario, en el cual se especifican las asignaciones de los pines del FPGA utilizados en el diseño, así como las condiciones de frecuencia y periodo de reloj con el que trabaja el sistema. En la Figura 3-2 se indica el archivo de restricciones del usuario utilizado en este sistema.

```
NET "clk_in" LOC = "E12"| IOSTANDARD = LVCMOS33 ;
NET "clk_in" PERIOD = 20.0ns HIGH 40%;

net "rst" LOC = "T15" | IOSTANDARD = LVTTL | PULLDOWN ;

NET "rx" LOC = "E16" | IOSTANDARD = LVTTL ;
NET "tx" LOC = "F15" | IOSTANDARD = LVTTL | DRIVE = 4 | SLEW = SLOW ;

NET "quality<0>" LOC = "V8" | IOSTANDARD = LVTTL | PULLUP ;
NET "quality<1>" LOC = "U10"| IOSTANDARD = LVTTL | PULLUP ;

NET "led_add<4>" LOC = "W21" | IOSTANDARD = LVTTL | SLEW = QUIETIO | DRIVE = 4 ;
NET "led_add<3>" LOC = "Y22" | IOSTANDARD = LVTTL | SLEW = QUIETIO | DRIVE = 4 ;
NET "led_add<2>" LOC = "V20" | IOSTANDARD = LVTTL | SLEW = QUIETIO | DRIVE = 4 ;
NET "led_add<1>" LOC = "V19" | IOSTANDARD = LVTTL | SLEW = QUIETIO | DRIVE = 4 ;
NET "led_add<0>" LOC = "U19" | IOSTANDARD = LVTTL | SLEW = QUIETIO | DRIVE = 4 ;

NET "led_quality<1>" LOC = "T19" | IOSTANDARD = LVTTL | SLEW = QUIETIO | DRIVE = 4 ;
NET "led_quality<0>" LOC = "R20" | IOSTANDARD = LVTTL | SLEW = QUIETIO | DRIVE = 4 ;
```

Figura 3- 2. Archivo de restricciones de usuario (UCF) del sistema diseñado

De la Figura se puede comentar lo siguiente:

- La restricción **NET** hace referencia a un puerto de la entidad de más alta jerarquía en el diseño conectado con un pin del FPGA.
- La restricción **LOC** especifica el pin del FPGA asociado al puerto de la entidad de mayor jerarquía.
- La restricción **IOSTANDARD** especifica el estándar eléctrico de la lógica utilizada por el pin del FPGA.

- La restricción **PULLDOWN** activa la resistencia interna de pull-down del pin seleccionado en el FPGA. En el caso de requerir una resistencia de pull-up se debe utilizar la restricción **PULLUP**.
- La restricción **SLEW** especifica el comportamiento de la velocidad de transición de voltaje (*slew rate*[27]) del pin y puede ser utilizado en pines de entrada, salida o bidireccionales. Las opciones disponibles para esta restricción son *SLOW* (para transiciones lentas)*, FAST* (para transiciones rápidas) *y QUIETIO* (para transiciones muy lentas).
- La restricción **DRIVE** define la corriente que maneja el buffer de entrada/salida del pin del FPGA cuando éste tiene un estándar eléctrico de tipo LVTTL, LVCMOS12, LVCMOS15, LVCMOS18, LVCMOS25, o LVCMOS33.
- La restricción **PERIOD** define la velocidad de la señal de reloj. Puede ser expresada mediante la duración del período (ps, ns, us, ms) o con un valor de frecuencia (GHz, MHz, KHz). También puede definir el ciclo de trabajo mediante la opción HIGH.

Cuando los procesos de implementación, mencionados anteriormente, terminan, se procede a generar el *bitstream* de configuración, para luego descargarlo en el FPGA. Para el caso del sistema que se ha diseñado, se ha programado la memoria de configuración correspondiente a la plataforma flash PROM de Xilinx, la cual tiene una capacidad de 4Mbits. Para programar esta memoria se debe disponer los *jumpers* de configuración, que posee el módulo de desarrollo, como lo indica la Figura 3-3.

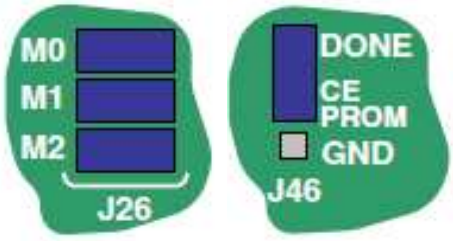
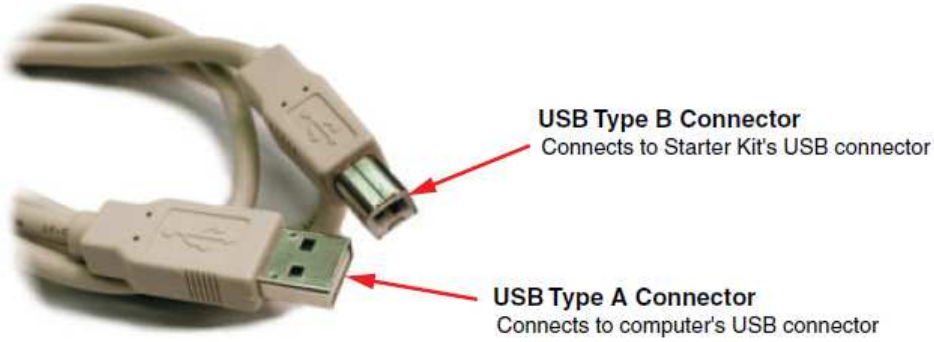


Figura 3- 3. Disposición de los *jumpers* de configuración para programación de la plataforma flash PROM de Xilinx

[27] **Slew Rate:** En sistemas digitales se define como el tiempo de transición de una señal digital entre dos puntos fijos de medida.

El proceso de programación de la plataforma flash PROM de Xilinx, se realiza mediante la herramienta *Impact* que se encuentra integrada en el paquete de software ISE Foundation. Como se indicó en el Capítulo 1, el módulo de desarrollo *Spartan-3A Starter Kit* tiene un controlador JTAG embebido, por lo cual, para la programación del módulo de desarrollo se requiere un cable USB estándar, como el que se muestra en la Figura 3-4. Este cable se conecta al módulo de desarrollo *Spartan-3A Starter Kit*, como lo indica la Figura 3-5.



Fuente [1]

Figura 3- 4. Cable USB estándar para la configuración del módulo de desarrollo

Fuente [1]

Figura 3- 5. Conexión del cable USB estándar al módulo de desarrollo

Cuando la programación de la plataforma flash PROM ha finalizado con éxito, la herramienta *Impact* despliega un mensaje como el que se muestra en la Figura 3-6.

Figura 3- 6. Programación exitosa del dispositivo

Un led en el módulo de desarrollo *Spartan-3A Starter Kit*, indica que el FPGA ha sido configurado con éxito, como lo muestra la Figura 3-7

Fuente [1]

Figura 3- 7. Led de estado de configuración del módulo de desarrollo

3.4. PRUEBAS CON EL SISTEMA DE ADQUISICIÓN, COMPRESIÓN Y ALMACENAMIENTO DE IMÁGENES

Para realizar las pruebas con el sistema diseñado, se debe contar con los siguientes elementos:

- Módulo de desarrollo *Spartan-3A Starter Kit*
- Adaptador de 5V, para el módulo de desarrollo
- Cable USB estándar, para la programación de la memoria de configuración.

- Cable RS-232 directo, con terminales macho y hembra para conectar el puerto RS-232 del módulo de desarrollo con el puerto serial del computador.
- Adaptador USB a RS-232, en caso de que el computador no posea puerto serial RS-232.
- Cámara Web con controlador estándar para Windows.

En la Figura 3-8 se muestran los elementos antes mencionados.

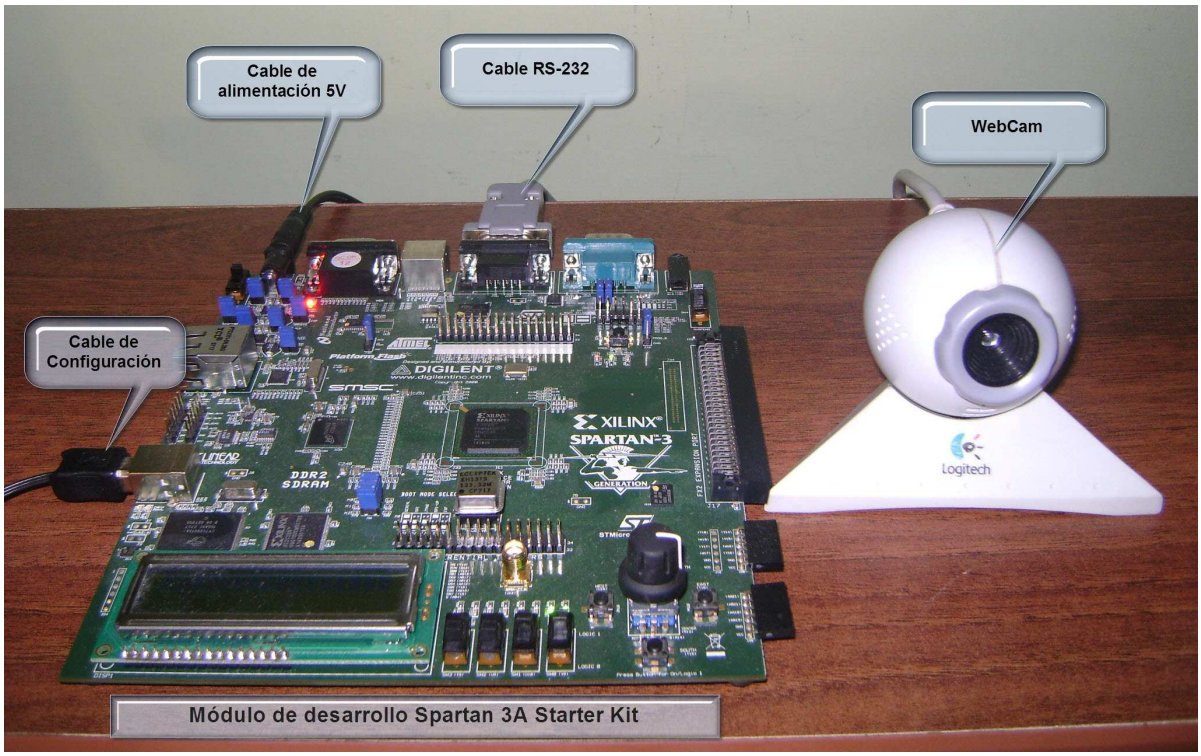


Figura 3- 8. Elementos necesarios para las pruebas del sistema

Adicionalmente, se requiere un computador (desktop o laptop) con Matlab 2007 (o superior) pre instalado, para poder ejecutar el programa de la interfaz gráfica de control. En la Figura 3-9 se muestra el módulo de desarrollo *Spartan-3A Starter Kit* conectado a la laptop que ejecuta la interfaz gráfica de control en Matlab 2007.

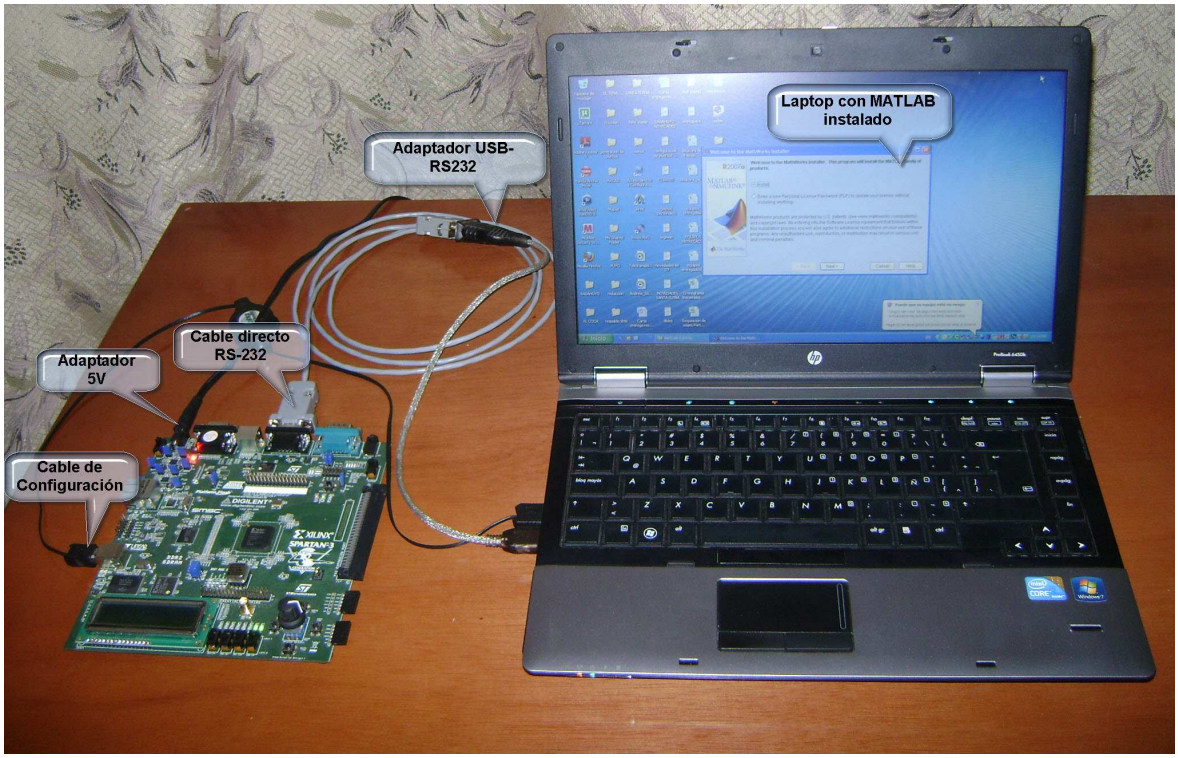


Figura 3- 9. Módulo de desarrollo conectado con una laptop

Para las pruebas del sistema de adquisición, compresión y almacenamiento de imágenes se han creado dos escenarios.

3.4.1. PRIMER ESCENARIO DE PRUEBAS

En el primer escenario de pruebas, se obtiene una imagen sin comprimir desde un archivo de imagen bmp, a través de la interfaz gráfica de control, y se realiza la compresión con los cuatro factores de calidad que se establecieron en el diseño. Los resultados de las imágenes comprimidas, en el primer escenario de pruebas, se muestran en la Tabla 3-1.

Tabla 3- 1. Resultados del primer escenario de pruebas del sistema

Factor de Calidad	Imagen JPEG
25	
35	
50	
75	

3.4.2. SEGUNDO ESCENARIO DE PRUEBAS

En el segundo escenario de pruebas, se obtiene una imagen sin comprimir desde una cámara Web, a través de la interfaz gráfica de control, y se realiza la compresión con los cuatro factores de calidad que se establecieron en el diseño. Los resultados de las imágenes comprimidas, en el segundo escenario de pruebas, se muestran en la Tabla 3-2.

Tabla 3- 2. Resultados del segundo escenario de pruebas del sistema

Factor de Calidad	Imagen JPEG
25	
30	
50	
75	

Se puede observar en las imágenes obtenidas, los efectos de la cuantización en la calidad final de la imagen. Como se esperaba, la calidad de la imagen va mejorando conforme el factor de calidad aumenta, ya que la cuantización es menos agresiva al incrementar el factor de calidad.

3.5. VERIFICACIÓN DE LOS PARÁMETROS DE COMPRESIÓN

La verificación de los parámetros de compresión, se ha realizado mediante la herramienta de software *JPEGsnoop*, la cual puede ser descargada libremente a través de la página Web http://www.impulseadventure.com/photo/jpeg-snoop.html. Este software permite analizar ficheros de fotogramas, como los archivos de imagen *jpg*; para el caso de la imagen comprimida con el sistema diseñado, se verificarán los siguientes parámetros:

- Marcador de inicio de imagen (SOI).
- Marcador de aplicación JFIF (APP_0).
- Marcador de inicio de trama (SOF).
- Marcador de definición de tablas de Huffman (DHT).
- Marcador de definición de tablas de cuantización (DQT).
- Marcador de inicio de exploración (SOS).
- Marcador de fin de imagen (EOI).

La interfaz de usuario, de la herramienta de software *JPEGsnoop,* se ilustra en la Figura 3-10. La interfaz de usuario se divide en dos secciones: en el área de parámetros del archivo, se despliega la información sobre la codificación de la imagen, debidamente tabulada. En el área de visualización se despliega la imagen decodificada. Para cargar un archivo, para ser analizado mediante el programa, se selecciona *File/Open Image…* y se selecciona el archivo de imagen que se quiera examinar.

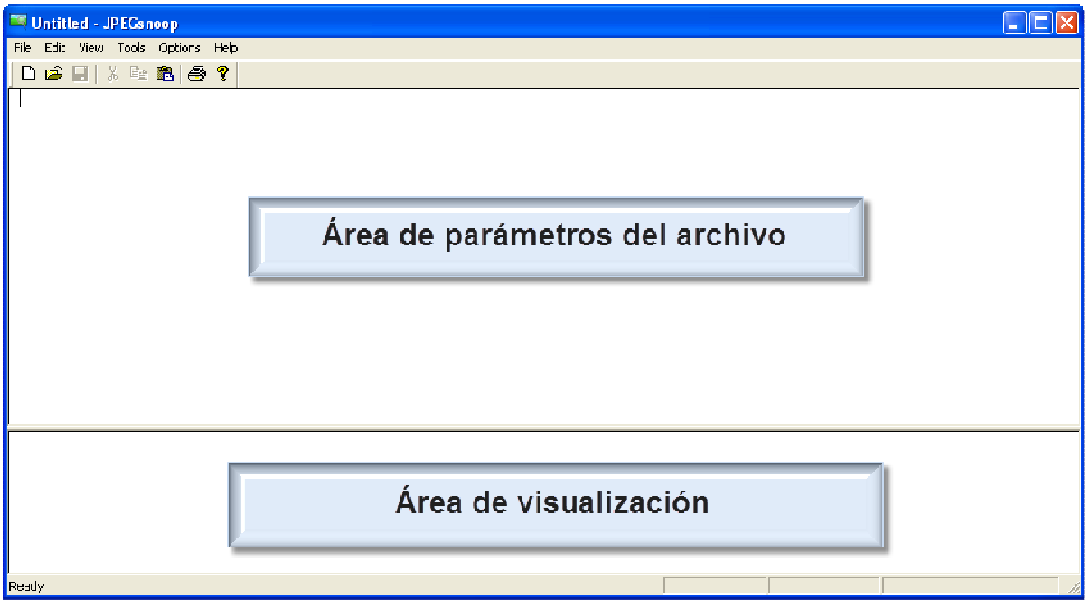


Figura 3- 10. Interfaz de usuario de la herramienta de software *JPEGsnoop*

Los resultados, presentados a continuación, corresponden, como se esperaba, a los contenidos que se definieron en la memoria ROM de la cabecera JFIF, en el módulo de compresión, y que se encuentran descritos en el Anexo 9.

3.5.1. MARCADOR DE INICIO DE IMAGEN Y MARCADOR DE APLICACIÓN JFIF

Como se mencionó en el Capítulo 1, el formato de archivo JFIF, requiere que, después de definido el marcador de inicio de imagen, se defina el marcador APP_0. Esta condición se cumple en todas las imágenes de prueba presentadas en la Tabla 3-1 y la Tabla 3-2. El resultado de esta verificación, a través de la herramienta de Software *JPEGsnoop*, se muestra en la Figura 3-11. Como se puede observar en la figura, se cumple con el identificador de *JFIF* y con la versión del formato de archivo.

```
Start Offset: 0x00000000
*** Marker: SOI (xFFD8) ***
  OFFSET: 0x00000000

*** Marker: APP0 (xFFE0) ***
  OFFSET: 0x00000002
  length      = 16
  identifier = [JFIF]
  version     = [1.1]
  density     = 1 x 1 (aspect ratio)
  thumbnail   = 0 x 0
```

Figura 3- 11. Resultado de la verificación de los marcadores de inicio de imagen y aplicación JFIF

3.5.2. MARCADOR DE INICIO DE TRAMA

El resultado de la verificación, a través de la herramienta de Software *JPEGsnoop*, para este marcador del archivo, se puede ver en la Figura 3-12. Como se puede observar en la figura, se ha cumplido con los parámetros establecidos para la imagen comprimida, los cuales son:

- Precisión de muestreo de 8 bits
- Dimensión de la imagen de 160x120 pixeles
- Una componente de imagen (luminancia)
- Frecuencia de muestro vertical y horizontal iguales
- Identificador de la componente de luminancia igual a 1 (valor recomendado en el estándar)
- Identificador de tabla de cuantización de la componente de luminancia igual a 0 (valor recomendado en el estándar)

```
*** Marker: SOF0 (Baseline DCT) (xFFC0) ***
  OFFSET: 0x00000014
  Frame header length = 11
  Precision = 8
  Number of Lines = 120
  Samples per Line = 160
  Image Size = 160 x 120
  Raw Image Orientation = Landscape
  Number of Img components = 1
    Component[1]: ID=0x01, Samp Fac=0x11 (Subsamp 1 x 1), Quant Tbl Sel=0x00 (Lum: Y)
```

Figura 3- 12. Verificación de los datos del marcador de inicio de trama

3.5.3. MARCADOR DE DEFINICIÓN DE TABLAS DE HUFFMAN

Como se indicó en el Capitulo 1, las tablas de Huffman se definen en la cabecera de archivo mediante dos parámetros: una enumeración de códigos según su longitud, y los códigos de Huffman abreviados. En el Anexo K del estándar JPEG, se indica está notación para los códigos de Huffman recomendados en el mismo estándar, y que se muestran en la Figura 3-13.

Los resultados de esta verificación se indican en la Figura 3-14 y en la Figura 3-15. En estas figuras se puede observar los códigos de Huffman abreviados, para los coeficientes DC y AC de la componente de luminancia. Se puede verificar en las figuras la completa coincidencia de los códigos definidos en el estándar y los encontrados en las imágenes comprimidas con el sistema. Además, se indica el identificador de la tabla de Huffman, el cual para la componente de luminancia es 0, y la clase de tabla: clase 0 para coeficientes DC y clase 1 para coeficientes AC.

DC-Y	Enumeración decódigos de Huffman	X'00	01	05	01	01	01	01	01	01	00	00	00	00	00	00	00'							
	Códigos de Huffman Abreviados	X'00	01	02	03	04	05	06	07	08	09	0A	0B'											
AC-Y	Enumeración de códigos de Huffman	X'00	02	01	03	03	02	04	03	05	05	04	04	00	00	01	7D'							
	Códigos de Huffman Abreviados	X'01	02	03	00	04	11	05	12	21	31	41	06	13	51	61	07							
		22	71	14	32	81	91	A1	08	23	42	B1	C1	15	52	D1	F0							
		24	33	62	72	82	09	0A	16	17	18	19	1A	25	26	27	28							
		29	2A	34	35	36	37	38	39	3A	43	44	45	46	47	48	49							
		4A	53	54	55	56	57	58	59	5A	63	64	65	66	67	68	69							
		6A	73	74	75	76	77	78	79	7A	83	84	85	86	87	88	89							
		8A	92	93	94	95	96	97	98	99	9A	A2	A3	A4	A5	A6	A7							
		A8	A9	AA	B2	B3	B4	B5	B6	B7	B8	B9	BA	C2	C3	C4	C5							
		C6	C7	C8	C9	CA	D2	D3	D4	D5	D6	D7	D8	D9	DA	E1	E2							
		E3	E4	E5	E6	E7	E8	E9	EA	F1	F2	F3	F4	F5	F6	F7	F8							
		F9	FA'																					

Figura 3- 13. Notación de los códigos de Huffman en la cabecera de archivo JFIF recomendados en el estándar JPEG

```
*** Marker: DHT (Define Huffman Table) (xFFC4) ***
  OFFSET: 0x00000066
  Huffman table length = 31
  ----
  Destination ID = 0
  Class = 0 (DC / Lossless Table)
    Codes of length 01 bits (000 total):
    Codes of length 02 bits (001 total): 00
    Codes of length 03 bits (005 total): 01 02 03 04 05
    Codes of length 04 bits (001 total): 06
    Codes of length 05 bits (001 total): 07
    Codes of length 06 bits (001 total): 08
    Codes of length 07 bits (001 total): 09
    Codes of length 08 bits (001 total): 0A
    Codes of length 09 bits (001 total): 0B
    Codes of length 10 bits (000 total):
    Codes of length 11 bits (000 total):
    Codes of length 12 bits (000 total):
    Codes of length 13 bits (000 total):
    Codes of length 14 bits (000 total):
    Codes of length 15 bits (000 total):
    Codes of length 16 bits (000 total):
    Total number of codes: 012
```

Figura 3- 14. Verificación de las Tablas de Huffman para los coeficientes DC de la Componente de luminancia

```
*** Marker: DHT (Define Huffman Table) (xFFC4) ***
  OFFSET: 0x00000087
  Huffman table length = 181
  ----
  Destination ID = 0
  Class = 1 (AC Table)
    Codes of length 01 bits (000 total):
    Codes of length 02 bits (002 total): 01 02
    Codes of length 03 bits (001 total): 03
    Codes of length 04 bits (003 total): 00 04 11
    Codes of length 05 bits (003 total): 05 12 21
    Codes of length 06 bits (002 total): 31 41
    Codes of length 07 bits (004 total): 06 13 51 61
    Codes of length 08 bits (003 total): 07 22 71
    Codes of length 09 bits (005 total): 14 32 81 91 A1
    Codes of length 10 bits (005 total): 08 23 42 B1 C1
    Codes of length 11 bits (004 total): 15 52 D1 F0
    Codes of length 12 bits (004 total): 24 33 62 72
    Codes of length 13 bits (000 total):
    Codes of length 14 bits (000 total):
    Codes of length 15 bits (001 total): 82
    Codes of length 16 bits (125 total): 09 0A 16 17 18 19 1A 25 26 27 28 29 2A 34 35 36
                                         37 38 39 3A 43 44 45 46 47 48 49 4A 53 54 55 56
                                         57 58 59 5A 63 64 65 66 67 68 69 6A 73 74 75 76
                                         77 78 79 7A 83 84 85 86 87 88 89 8A 92 93 94 95
                                         96 97 98 99 9A A2 A3 A4 A5 A6 A7 A8 A9 AA B2 B3
                                         B4 B5 B6 B7 B8 B9 BA C2 C3 C4 C5 C6 C7 C8 C9 CA
                                         D2 D3 D4 D5 D6 D7 D8 D9 DA E1 E2 E3 E4 E5 E6 E7
                                         E8 E9 EA F1 F2 F3 F4 F5 F6 F7 F8 F9 FA
    Total number of codes: 162
```

Figura 3- 15. Verificación de las Tablas de Huffman para los coeficientes AC de la Componente de luminancia

3.5.4. MARCADOR DE DEFINICIÓN DE TABLAS DE CUANTIZACIÓN

Mediante la herramienta de software *JPEGsnoop*, se puede extraer de la cabecera del archivo de imagen .jpg, las tablas de cuantización utilizadas en la compresión de la imagen, así como el factor de calidad que se ha alcanzado con dichas tablas. En las siguientes figuras se muestran las tablas de cuantización obtenidas mediante la herramienta de software, para los cuatro factores de calidad definidos en el diseño. Se puede observar en las figuras, que se ha logrado alcanzar los factores de calidad deseados mediante el método descrito en el Capítulo 1, y con el cual se definió las tablas en el bloque de cuantización del módulo de compresión.

```
*** Marker: DQT (xFFDB) ***
  Define a Quantization Table.
  OFFSET: 0x00000021
  Table length = 67
  ----
  Precision=8 bits
  Destination ID=0 (Luminance)
    DQT, Row #0:  32  22  20  32  48  80 102 122
    DQT, Row #1:  24  24  28  38  52 116 120 110
    DQT, Row #2:  28  26  32  48  80 114 138 112
    DQT, Row #3:  28  34  44  58 102 174 160 124
    DQT, Row #4:  36  44  74 112 136 218 206 154
    DQT, Row #5:  48  70 110 128 162 208 226 184
    DQT, Row #6:  98 128 156 174 206 242 240 202
    DQT, Row #7: 144 184 190 196 224 200 206 198
    Approx quality factor = 25.00 (scaling=200.00 variance=0.00)
```

Figura 3- 16. Tabla de cuantización para el factor de calidad de 25

```
*** Marker: DQT (xFFDB) ***
  Define a Quantization Table.
  OFFSET: 0x00000021
  Table length = 67
  ----
  Precision=8 bits
  Destination ID=0 (Luminance)
    DQT, Row #0:  23  16  14  23  34  57  73  87
    DQT, Row #1:  17  17  20  27  37  83  86  79
    DQT, Row #2:  20  19  23  34  57  81  99  80
    DQT, Row #3:  20  24  31  41  73 124 114  89
    DQT, Row #4:  26  31  53  80  97 156 147 110
    DQT, Row #5:  34  50  79  91 116 149 161 131
    DQT, Row #6:  70  91 111 124 147 173 171 144
    DQT, Row #7: 103 131 136 140 160 143 147 141
    Approx quality factor = 35.03 (scaling=142.73 variance=0.89)
```

Figura 3- 17. Tabla de cuantización para el factor de calidad de 35

```
*** Marker: DQT (xFFDB) ***
  Define a Quantization Table.
  OFFSET: 0x00000021
  Table length = 67
  ----
  Precision=8 bits
  Destination ID=0 (Luminance)
    DQT, Row #0:  16  11  10  16  24  40  51  61
    DQT, Row #1:  12  12  14  19  26  58  60  55
    DQT, Row #2:  14  13  16  24  40  57  69  56
    DQT, Row #3:  14  17  22  29  51  87  80  62
    DQT, Row #4:  18  22  37  56  68 109 103  77
    DQT, Row #5:  24  35  55  64  81 104 113  92
    DQT, Row #6:  49  64  78  87 103 121 120 101
    DQT, Row #7:  72  92  95  98 112 100 103  99
    Approx quality factor = 50.00 (scaling=100.00 variance=0.00)
```

Figura 3- 18. Tabla de cuantización para el factor de calidad de 50

```
*** Marker: DQT (xFFDB) ***
  Define a Quantization Table.
  OFFSET: 0x00000021
  Table length = 67
  ----
  Precision=8 bits
  Destination ID=0 (Luminance)
    DQT, Row #0:   8   6   5   8  12  20  26  31
    DQT, Row #1:   6   6   7  10  13  29  30  28
    DQT, Row #2:   7   7   8  12  20  29  35  28
    DQT, Row #3:   7   9  11  15  26  44  40  31
    DQT, Row #4:   9  11  19  28  34  55  52  39
    DQT, Row #5:  12  18  28  32  41  52  57  46
    DQT, Row #6:  25  32  39  44  52  61  60  51
    DQT, Row #7:  36  46  48  49  56  50  52  50
    Approx quality factor = 74.75 (scaling=50.51 variance=0.81)
```

Figura 3- 19. Tabla de cuantización para el factor de calidad de 75

3.5.5. MARCADOR DE INICIO DE EXPLORACIÓN

La herramienta de software *JPEGsnoop* permite decodificar las matrices DCT de cada MCU de la imagen, encontradas en los datos de la imagen comprimida, que acompañan al marcador de inicio de exploración. También, el software calcula la tasa de compresión alcanzada. En la Figura 3-20 se indica la información del marcador de inicio de exploración extraída por el software; en esta se verifican los siguientes parámetros, que corresponden al estilo de codificación secuencial *Baseline* del estándar JPEG:

- Contador de componentes igual a 1 (solo luminancia).
- Descriptor de la componente, en el cual se especifica el identificador de la componente de luminancia (el cual es 1) y el identificador de las tablas de Huffman (el cual es 0).
- Selección espectral con inicio en 0 y fin en 63.
- Aproximación sucesiva igual a 0.

```
*** Marker: SOS (Start of Scan) (xFFDA) ***
  OFFSET: 0x0000013E
  Scan header length = 8
  Number of img components = 1
    Component[1]: selector=0x01, table=0x00
  Spectral selection = 0 .. 63
  Successive approximation = 0x00
```

Figura 3- 20. Verificación de los datos del marcador de inicio de exploración

En la Tabla 3-3 se indican las tasas de compresión alcanzadas por las ocho imágenes, en los dos escenarios de prueba, mostradas en la Tabla 3-1 y la Tabla 3-2, que han sido extraídas mediante la herramienta de software *JPEGsnoop*.

3.5.6. MARCADOR DE FIN DE IMAGEN

El último marcador de un archivo JFIF es el de fin de imagen (EOI); con la herramienta de software *JPEGsnoop* se ha extraído la información de este marcador como se indica en la Figura 3-21.

```
*** Marker: EOI (End of Image) (xFFD9) ***
  OFFSET: 0x00000580
```

Figura 3- 21. Verificación de la información del marcador de fin de imagen

Tabla 3- 3. Tasas de compresión alcanzadas en los dos escenarios de pruebas

Primer escenario de pruebas	
Factor de Calidad	**Tasa de compresión**
25	Compression stats: Compression Ratio: 11.39:1 Bits per pixel: 0.70:1
35	Compression stats: Compression Ratio: 9.32:1 Bits per pixel: 0.86:1
50	Compression stats: Compression Ratio: 7.50:1 Bits per pixel: 1.07:1
75	Compression stats: Compression Ratio: 5.13:1 Bits per pixel: 1.56:1
Segundo escenario de pruebas	
Factor de Calidad	**Tasa de compresión**
25	Compression stats: Compression Ratio: 17.79:1 Bits per pixel: 0.45:1
35	Compression stats: Compression Ratio: 14.70:1 Bits per pixel: 0.54:1
50	Compression stats: Compression Ratio: 11.92:1 Bits per pixel: 0.67:1
75	Compression stats: Compression Ratio: 7.78:1 Bits per pixel: 1.03:1

REFERENCIAS

[1] Xilinx, Inc. *Spartan-3A/3AN Starter Kit Board: User Guide*. Mayo 2007, versión 1.0.

Obtenido de:

http://www.xilinx.com/support/documentation/boards_and_kits/ug334.pdf

(Último acceso: 17/12/2009)

CAPÍTULO 4

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

- Se verificó, en la práctica, el potencial que poseen los FPGAs actuales y, en caso particular, la arquitectura de los FPGAs de la plataforma Spartan 3, cuyos componentes (CLBs y bloques embebidos) han facilitado la descripción VHDL e implementación de funciones de gran complejidad de cómputo, como es el caso de la Transformada Discreta del Coseno y la codificación de entropía de Huffman, ambos bloques de suma importancia en el esquema de codificación *Baseline* del estándar JPEG.

- Se ha puesto en evidencia la gran versatilidad del lenguaje de descripción de hardware VHDL, con respecto a sus estilos descriptivos (algorítmico, flujo de datos y estructural), los cuales pueden convivir en un mismo diseño, facilitando el proceso de descripción y síntesis de algoritmos de alto nivel de abstracción, como es el esquema de codificación *Baseline* del estándar JPEG, implementado en este proyecto.

- El conocimiento de la arquitectura del FPGA Spartan-3A, fue un aspecto fundamental en el desarrollo del diseño del sistema propuesto en este proyecto, ya que se pudo mejorar las descripciones VHDL de los módulos del sistema, obteniendo una mejor correlación entre el código VHDL y los bloques funcionales que posee esta arquitectura; de esta manera, se optimizaron los recursos lógicos del FPGA y se evitaron posibles errores de implementación, debido a malas prácticas de descripción, como por ejemplo la implementación de latches, que producen errores de sincronismo en la implementación final.

- El diseño jerárquico y estructurado en el lenguaje VHDL, permitió la verificación de la funcionalidad de los bloques de cada módulo del sistema, tanto a nivel de simulación funcional como a nivel de implementación física; de este modo, se facilitó la ubicación puntual de un bloque del módulo de compresión, cuya implementación física no reflejaba el comportamiento de su simulación funcional, para posteriormente corregir su descripción VHDL.

- La simulación funcional fue un proceso útil para verificar el comportamiento de los bloques que integran los módulos del sistema diseñado, sin embargo, en los diseños implementados en el FPGA, la simulación funcional sólo reflejó el comportamiento inherente que se obtiene de la descripción VHDL, sin considerar parámetros reales de implementación como son los retardos de interconexión y los retardos por niveles de lógica utilizados, ni tampoco la implementación física de latches que generan problemas de sincronismo, lo cual, en su conjunto, produjo resultados contradictorios entre la simulación funcional y la implementación final del sistema, por lo cual un diseño con FPGAs debe contar también con la simulación de tiempos, la cual toma en cuenta parámetros reales de implementación específicos de cada dispositivo FPGA y garantiza resultados más fiables que la simulación funcional.

- La simulación funcional efectuada mediante editores de formas de onda en la herramienta de desarrollo ISE Foundation 10.1 de Xilinx, facilitó la creación de estímulos de prueba para los bloques de los módulos del sistema diseñado, sin embargo, el procedimiento para crearlos no tiene la misma flexibilidad de un *testbench* creado mediante sentencias VHDL para producir estímulos de simulación más acordes a la realidad de una implementación física, por lo cual un *testbench* desarrollado mediante sentencias VHDL contribuye mejor a la depuración de diseños de sistemas digitales descritos con este lenguaje.

- La descripción de máquinas de estados se convirtió en el procedimiento más favorable para implementar las funciones más complejas de las secciones de hardware del sistema (en términos de descripción), como la generación de bytes, el módulo de control y la generación de dirección de las palabras código

en la codificación de Huffman; esto porque la naturaleza de la máquina de estados permite la implementación de sistemas cuyas tareas forman una secuencia bien definida, como es el caso de los bloques antes mencionados.

- Los IP Cores que se encuentran disponibles en la herramienta de desarrollo ISE Foundation 10.1, constituyeron un importante apoyo en el diseño del sistema propuesto en este proyecto, ya que con éstos, se implementaron funciones específicas (como memorias RAM, memorias ROM, FIFOs y divisores), optimizando recursos del FPGA y evitando al diseñador su descripción, y más bien enfocándolo sólo a configurar parámetros en los asistentes de los IP Cores, y a desarrollar aspectos más relevantes del diseño, reduciendo el diseño en términos de líneas de código y tiempo de implementación.

- El diseño del módulo de compresión y las secciones de hardware de los otros módulos del sistema de adquisición, compresión y almacenamiento de imágenes, se realizaron en 4481 líneas comentadas de código VHDL distribuidas en 18 entidades de diseño.

- La interpretación de los reportes generados en cada etapa del flujo de diseño de la herramienta de desarrollo ISE Foundation 10.1 de Xilinx, fueron de utilidad para verificar los criterios de diseño del sistema propuesto en este proyecto, así como encontrar potenciales errores en la descripción VHDL.

- El lenguaje de programación de Matlab, utilizado en la interfaz gráfica de control, es muy versátil para adquirir datos desde periféricos y manipularlos dependiendo de las necesidades de la aplicación desarrollada, lo que contribuyó a adaptar las imágenes a comprimir, de acuerdo a los recursos de memoria del FPGA utilizado, así como controlar de una manera sencilla la transmisión y recepción de datos a través del puerto serial.

- Se obtuvieron los resultados de las imágenes comprimidas que se esperaban, visualizándose en el computador que aloja la interfaz gráfica de control de

usuario; se verificó el cumplimiento de los parámetros de compresión, así como el formato de archivo JFIF utilizado para generar los archivos de imagen. Las imágenes obtenidas son legibles; sin embargo, se evidencian problemas de codificación en algunos bloques, presumiblemente debido a los errores de redondeo en los bloques de la DCT-2D y cuantización, así como por los errores de aproximación de los coeficientes de transformación utilizados para la implementación de la DCT-1D, que son representados como números de 13 bits. Los errores de codificación son más notorios debido a la resolución de la imagen de prueba de 160x120, lo cual magnifica un error en un bloque de 8x8 pixeles.

RECOMENDACIONES

- Mejorar el diseño del sistema de adquisición, compresión y almacenamiento de imágenes. Empleando un módulo de desarrollo que posea un FPGA con mayores recursos lógicos (CLBs y bloques embebidos), se puede mejorar el diseño del sistema desde los siguientes puntos de vista:

  - ✓ *Realizar un compresor para una imagen a color y de mayor resolución*. Esto se puede lograr implementando tres módulos de compresión, como el diseñado en este proyecto, en paralelo y cambiar las tablas de cuantización y tablas de Huffman, por las correspondientes para las componentes de crominancia, en los dos módulos de compresión añadidos. Se puede hacer la compresión de imágenes de mayor resolución, dependiendo de la cantidad de bloques de RAM embebida que posea el FPGA.
  - ✓ *Realizar un compresor para una imagen a color y de mayor resolución utilizando un soft-processor.* Se puede modificar el módulo de compresión actual, para que tenga las dos tablas de cuantización y dos tablas de Huffman para las componentes de luminancia y crominancia, y su selección esté controlada por un programa ejecutado sobre un soft-processor, como por ejemplo el MicroBlaze de Xilinx; el soft-processor puede también controlar los elementos de almacenamiento masivo que contienen los módulos de desarrollo, como memorias SDRAM DDR2 y DDR3, de una manera más sencilla que implementando un controlador en VHDL, con lo cual se tendría disponible gran cantidad de memoria RAM para almacenar una imagen de mayor resolución, sin utilizar los bloques de RAM embebida del FPGA.
  - ✓ *Realizar un sistema de adquisición, compresión y almacenamiento de imágenes autónomo*. Se puede prescindir de la adquisición de la imagen a comprimir desde la interfaz gráfica de control, diseñada en Matlab, implementado un módulo de control que ejecute la adquisición de la imagen a comprimir desde un módulo de sensor de imagen, como

por ejemplo el C3188A, el cual posee un sensor de imagen CMOS OV7620 de Ovnivision que entrega una imagen a color de 640x480 pixeles y posee salidas digitales para las componentes YCbCr. Con un módulo de sensor de imagen como el descrito y cualquiera de las opciones del módulo de compresión antes mencionadas, se puede hacer al sistema de adquisición, compresión y almacenamiento de imágenes menos dependiente de una aplicación de software para su control, a modo de una cámara digital.

- Se pueden optimizar mejor los recursos lógicos y la frecuencia de trabajo utilizados en diseños complejos, como el que se realizó en el presente proyecto, mediante algunas prácticas de codificación HDL, entre las que podemos destacar:

  - ✓ La utilización de resets sincrónicos, y en lo posible con carácter local (no global) en los distintos componentes del diseño.
  - ✓ Evitar la generación de latches debido a instrucciones *when*, *if* o *case* con permutaciones de señales incompletas.
  - ✓ Registrar las entradas y salidas de los componentes del sistema.
  - ✓ Añadir en la medida de lo posible niveles Pipeline.
  - ✓ Si el diseño requiere señales de control en un proceso sincrónico, se debe respetar el siguiente orden: 1) reset sincrónico, 2) set sincrónico, 3) habilitación (enable) y 4) el proceso combinacional.

- Se recomienda seguir incentivando, desde los pensum de estudio, nuevas metodologías de diseño digital en la Facultad de Ingeniería Eléctrica y Electrónica de la Escuela Politécnica Nacional, ya que tecnologías como los FPGAs actuales, tienen un amplio campo de aplicación en muchas de las cátedras impartidas en nuestra Facultad.

ANEXO 1

TECNOLOGÍAS DE PROGRAMACIÓN MÁS COMUNES DE LOS FPGAs ACTUALES

Los CLB y las interconexiones dentro del FPGA son programadas por el usuario, por lo que es menester hacer una breve descripción de las tecnologías de programación de estos bloques. En la actualidad las principales tecnología de programación de FPGAs son las siguientes:

- **Basada en SRAM (RAM estática)**. Este tipo de tecnología de programación es la más común entre las FPGAs actuales. Una de las ventajas que presenta esta tecnología, es el hecho de poder programar el FPGA una y otra vez, pudiendo reutilizar el mismo dispositivo. Ya que el almacenamiento de datos en las celdas SRAM es volátil, la información de las celdas programas se perderá una vez que se retire la alimentación de voltaje del dispositivo; es por esto que los FPGAs basados en SRAM tienen asociado una memoria externa de tipo EEPROM, denominada memoria de configuración, en donde se almacena el *bitstream* de configuración del FPGA. El hecho de utilizar una memoria EEPROM externa implica una desventaja en cuestión de seguridad, ya que el *bitstream* almacenado puede ser extraído de este dispositivo y obtener, mediante procesos de ingeniería inversa, nuestro diseño. Los FPGAs más modernos tienen la posibilidad de almacenar el *bitstream* de configuración encriptado, por lo que estos dispositivos poseen recursos lógicos especiales para realizar esta operación. La desventaja de este tipo de FPGAs es la necesidad de una batería de respaldo externa para mantener almacenada la clave de encriptación en registros especiales, aún cuando se deje sin energía al dispositivo, incrementando el tamaño del sistema, su complejidad y su costo.

- **Basada en Anti fusible**. Esta tecnología se denomina de anti fusible debido a que, al contrario de un fusible donde existe una conexión previa que se destruye al pasar una corriente, la conexión en estos dispositivos se tiene que crear. Estos dispositivos son programables una sola vez (*One Time Programed*) y al contrario de los dispositivos basados en SRAM, no pueden ser programados mientras son residentes de un sistema, ya que requieren un dispositivo especial para programarlos. La tecnología anti fusible es no volátil por lo que la FPGA no perderá su configuración al desconectar la fuente de poder del dispositivo, y por lo tanto no requiere de una memoria externa de configuración.

  Un anti fusible es implementado mediante dos líneas perpendiculares de material conductor separadas por un dieléctrico. Cuando una tensión elevada es aplicada a las dos líneas que se cruzan, el campo eléctrico generado entre las líneas es lo suficientemente grande para romper la rigidez dieléctrica del material aislante, creando un pequeño arco (conexión) permanente entre las línea. Por lo general se utiliza como metal el aluminio y como dieléctrico algún óxido.

  La naturaleza misma de la implementación de un anti fusible, hace que los FPGAs basados en esta tecnología sean óptimos para aplicaciones en ambientes de alta radiación (*Rad-Hard*), como son el caso de aplicaciones militares y aeroespaciales, ya que altos niveles de radiación puede producir el borrado o modificación de celdas de memoria. Hay que aclarar que actualmente los FPGAs basados en celdas de memoria SRAM también pueden ser utilizados para aplicaciones de alta radiación, debido a que utilizan empaquetados especiales contra radiación y triple redundancia de diseño[28].

[28] **Triple Redundancia de Diseño:** Técnica de implementación empleada en FPGAs basadas en celdas SRAM que establece tres copias de cada celda, con el fin de que si una celda es afectada por la incidencia de radiación se puede recuperar su valor original comparando las otras dos.

- **Basada en EEPROM/FLASH**. Los dispositivos basados en esta tecnología presentan ciertas ventajas frente a su contraparte SRAM. La ventaja más notoria es que una celda FLASH es mucho más pequeña que una SRAM, esto implica que se tendrá una lógica más compacta y por lo tanto una reducción de retardos de interconexión; también estos dispositivos son no volátiles por lo que no requieren de una memoria externa de configuración. A los dispositivos basados en EEPROM/FLASH, se los puede ver como el punto intermedio entre los FPGAs basados en SRAM y los basados en anti fusible. Con respecto a la seguridad, algunos dispositivos EEPROM/FLASH presentan un sistema de protección basado en el concepto de llave multibit (*multibit key*), la cual puede estar en el rango de 50 a algunos cientos de bits. Para poder programar o leer el *bitstream* desde el dispositivo, el usuario deberá ingresar la llave (clave) antes mencionada de lo contrario cualquier intento de reutilizar el FPGA será vano. Una desventaja de este tipo de FPGAs es el tiempo de programación, que por lo general es tres veces el tiempo de programación de un dispositivo basado en SRAM. Los dispositivos EEPROM/FLASH no son Rad Hard.

ANEXO 2

SPARTAN-3A FPGA FAMILY: DATA SHEET

(El contenido de este documento se presenta en el CD adjunto)

ANEXO 3

SPARTAN-3A/3AN STARTER KIT BOARD USER GUIDE

(El contenido de este documento se presenta en el CD adjunto)

ANEXO 4

CONFIGURACIÓN DEL MÓDULO DE DESARROLLO

El FPGA en el módulo de desarrollo Spartan-3A *Starter Kit* puede ser configurado de dos formas diferentes: configurando de manera directa el FPGA o programando cualquiera de las cuatro memorias de configuración, ambas mediante el cable JTAG embebido en el módulo o utilizando el puerto JTAG dedicado. Cuando se utiliza la última opción el usuario del módulo debe de disponer de un cable JTAG especial.

Existen en la placa del módulo de desarrollo tres bloques de jumper que deben ser tomados en cuenta para efectuar la configuración del FPGA o programar una memoria específica de configuración. Estos jumperes, se encuentran debidamente etiquetados en el PCB del módulo.

El conjunto de jumperes etiquetados como J26, es el que establece el modo de configuración que el FPGA utilizará cuando se aplique energía por primera vez al módulo. En la Tabla A4-1 se indica los modos de configuración posibles. El jumper etiquetado como J46 permite la habilitación de la memoria flash PROM de Xilinx, como también el posterior uso de ésta en la aplicación diseñada, la Tabla A4-2 describe las opciones de disposición de este jumper. Por último existe un conjunto de jumperes etiquetado como J1 que permite escoger qué memoria de tipo SPI será programada, como se indica en la Tabla A4-3.

Tabla A4- 1. Modos de configuración del FPGA

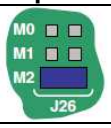
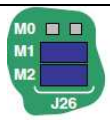
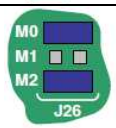
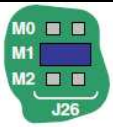
Modo de configuración	Fuente de la imagen de configuración	Disposición del Jumper J26	Disposición del Jumper J46
Internal Master SPI	Memoria Flash SPI interna. Solo disponible en FPGAs Spartan-3AN		
Master Serial	Memoria Flash PROM de Xilinx		
Master SPI	Memoria Flash PROM SPI especificada por el jumper J1		
Master BPI Up	Memoria NOR Flash PROM paralela		
JTAG	Bitstream transmitido desde el PC		

Tabla A4- 2. Disposición del jumper de habilitación de la plataforma Flash PROM de Xilinx

Plataforma Flash PROM	Disposición del Jumper J46	Modos de configuración permitidos	Observaciones
Deshabilitada		Cualquiera	La aplicación en el FPGA tiene completo acceso a las memorias Flash SPI y NOR Flash PROM Paralela, después de la configuración.
Habilitada durante la configuración		Master Serial o JTAG	La memoria Flash PROM es deshabilitada después de la configuración, la aplicación en el FPGA tiene completo acceso a las memorias Flash SPI y NOR Flash PROM Paralela, después de la configuración
Siempre habilitada		Master Serial o JTAG	La aplicación en el FPGA no tiene acceso a las memorias Flash SPI y NOR Flash PROM Paralela, después de la configuración

Las memorias Flash SPI que posee el módulo son: Atmel AT45DB161D de 16Mbit y STMicroelectronics M25P16 de 16Mbit. De las dos memorias flash SPI, solamente una puede ser configurada a la vez y, posteriormente, la aplicación puede tener acceso a una o ambas memorias, dependiendo de la disposición del conjunto de jumperes J1. En la Figura A4-1 se indica las conexiones entre las memorias Flash SPI y el FPGA Spartan-3A. En la figura se puede observar que existen dos señales de selección para las memorias, denominadas SPI_SS_B (pin Y4 del FPGA) y ALT_SS_B (pin Y5 del FPGA). Las entradas de selección de las dos memorias son activadas en bajo, por esto, durante el proceso de programación, la señal de selección SPI_SS_B estará en estado bajo y la señal de selección ALT_SS_B en estado alto, garantizando que solamente una memoria sea seleccionada para programarla y no existan conflictos durante el mencionado proceso. Por medio de la disposición del jumper J1 la señal SPI_SS_B es conducida hacia la entrada de selección de la memoria Flash SPI que se va a configurar y si se requiere, la señal ALT_SS_B a la otra. Después de la configuración del FPGA, la aplicación puede seleccionar cualquiera de las dos memorias mediante las señales antes mencionadas.

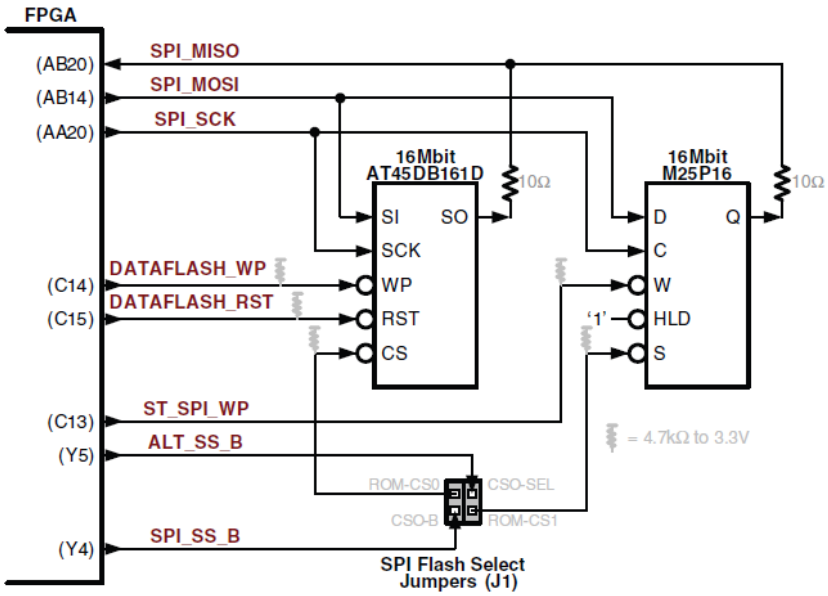
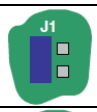
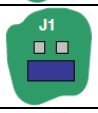
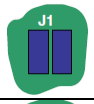
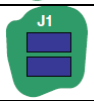
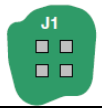


Figura A4- 1. Conexiones de las memorias Flash SPI con el FPGA Spartan-3A

Tabla A4- 3. Jumperes de selección de memorias Flash SPI

Disposición del Jumper J1	Fuente del modo de configuración SPI	Después de la configuración	
		Señal de selección para la memoria Atmel AT45DB161D	Señal de Selección para la memoria STMicro M25P16
	Atmel AT45DB161D	SPI_SS_B (Y4)	N/A
	STMicro M25P16	N/A	SPI_SS_B (Y4)
	Atmel AT45DB161D	SPI_SS_B (Y4)	ALT_SS_B (Y5)
	STMicro M25P16	ALT_SS_B (Y5)	SPI_SS_B (Y4)
	Ninguna	Ninguna	Ninguna

ANEXO 5

EJEMPLO DE DISEÑO CON ISE FOUNDATION 10.1

Para iniciar un proyecto, se selecciona del menú *File/New Project*, lo cual desplegará el cuadro de dialogo que se muestra en la Figura A5-1. En este cuadro de diálogo se debe escribir el nombre del proyecto y la ruta en donde se va a almacenar, también se debe escoger el tipo de fuente que va a contener el proyecto, pudiéndose escoger entre fuentes de tipo HDL, esquemático o archivos netlist de tipo EDIF (*Electronic Data Interchange Format*) o NGC/NGO. Continuando con el asistente se desplegará el cuadro de diálogo de la Figura A5-2. En este cuadro de diálogo se debe seleccionar la familia de FPGA o CPLD, el dispositivo dentro de la familia, el tipo de encapsulado del dispositivo, entre otros parámetros. También, se debe seleccionar el lenguaje HDL que se desea utilizar pudiendo escoger entre Verilog y VHDL, además del simulador a usar. Para el desarrollo de este ejemplo se utilizará el simulador *ISE Simulator*, que se encuentra incorporado en el paquete *ISE Foundation*. El FPGA que se utilizará en este ejemplo es de la familia Spartan-3A, el dispositivo XC3S700A con encapsulado FG484.

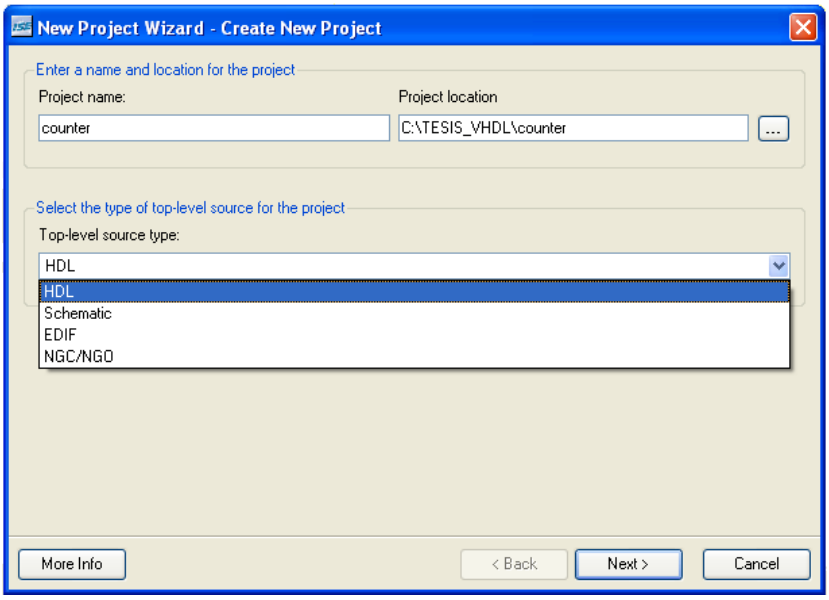


Figura A5- 1. Cuadro de Diálogo para generar un nuevo proyecto en ISE Foundation 10.1

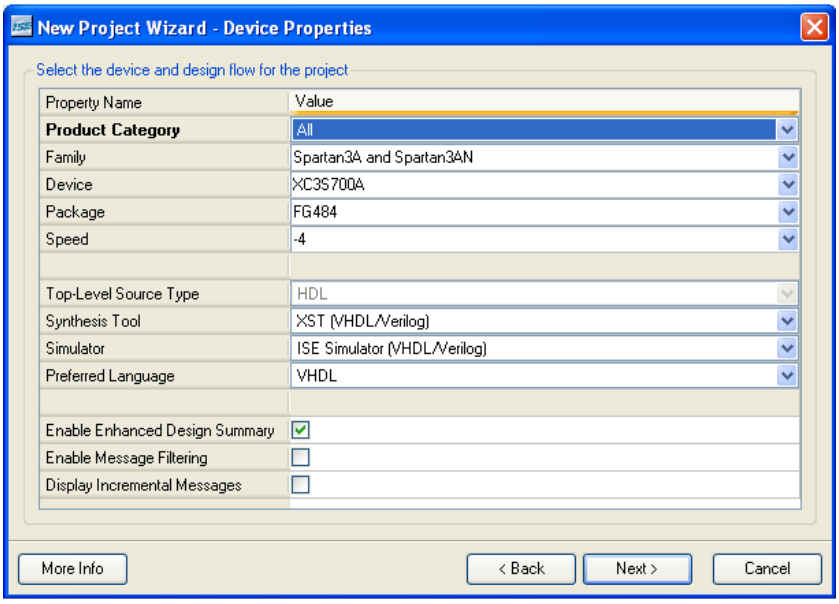


Figura A5- 2. Cuadro de Diálogo para seleccionar el FPGA en un nuevo proyecto en ISE Foundation 10.1

Una vez seleccionado el FPGA y el simulador a utilizar, se debe continuar con el asistente. El siguiente cuadro de diálogo, permitirá crear nuevos archivos fuentes al presionar *New Source*; con esto se desplegará el cuadro de dialogo que aparece en la Figura A5-3, en el cual se puede escoger varios tipos de archivos fuentes. En este ejemplo se utilizará para el diseño, Módulos VHDL (*VHDL Module*). Existen también otros tipos de archivos fuentes que están asociados a la simulación, como son *Test Bench Waveform, VHDL Test Bench* y *Verilog Test Fixture*. Para la implementación física, están asociados los archivos fuentes *Implementation Constraints File* y *I/O Pin Assignment*.

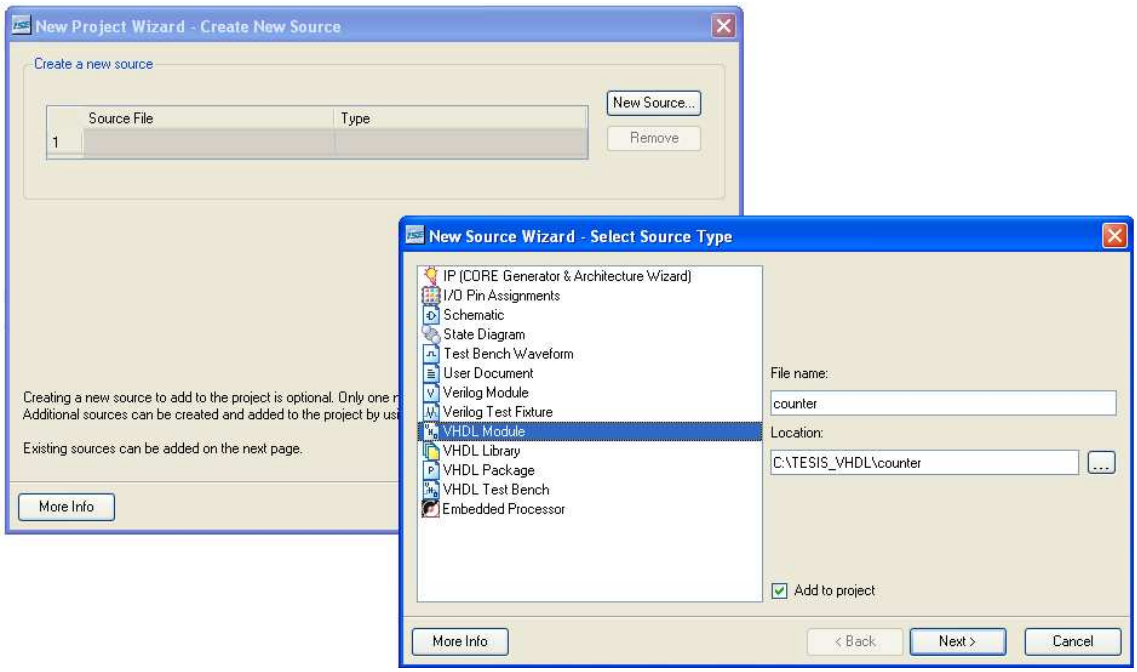


Figura A5- 3. Tipos de archivos fuentes disponibles en ISE

Una vez que se ha seleccionado un Módulo VHDL como archivo fuente para el diseño lógico y se haya elegido un nombre para el archivo, el siguiente cuadro de diálogo nos permite ingresar los puerto de entrada y salida que va a tener nuestra entidad, así como el nombre de la arquitectura (en el caso de un módulo VHDL), tal como lo indica la Figura A5-4. Después de este procedimiento, se desplegará un resumen de la entidad creada, en el que se resalta todos los parámetros configurados. Una vez finalizada la creación del archivo fuente, se prosigue con el asistente, cuyo siguiente cuadro de diálogo nos permite agregar al proyecto un archivo ya existente, en caso de necesitarlo; de no ser así, se prosigue con el asistente. Al final, se presenta un resumen con las características y parámetros que se han seleccionado, como lo indica la Figura A5-5; presionando *Finish* se habrá creado el proyecto en *ISE Foundation 10.1.*

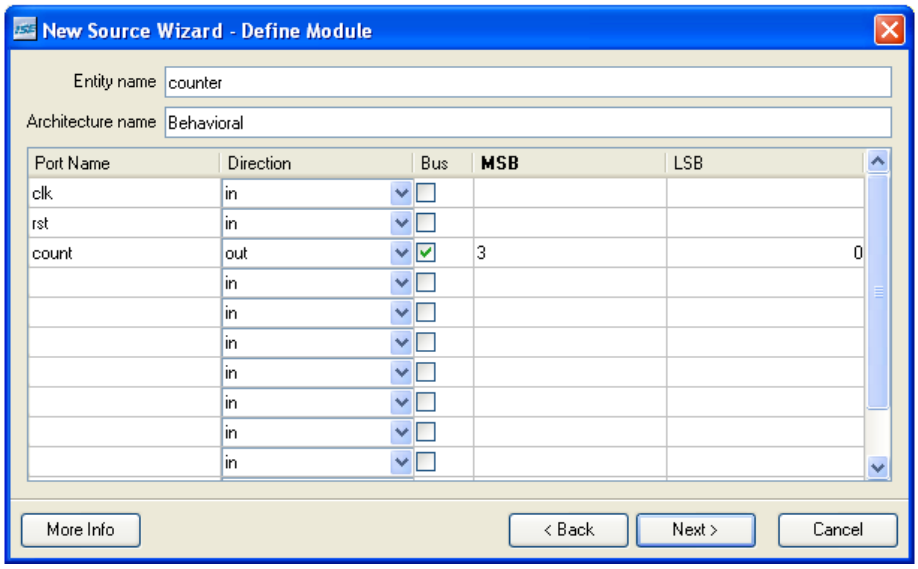


Figura A5- 4. Designación de puertos de entrada y salida para el módulo VHDL

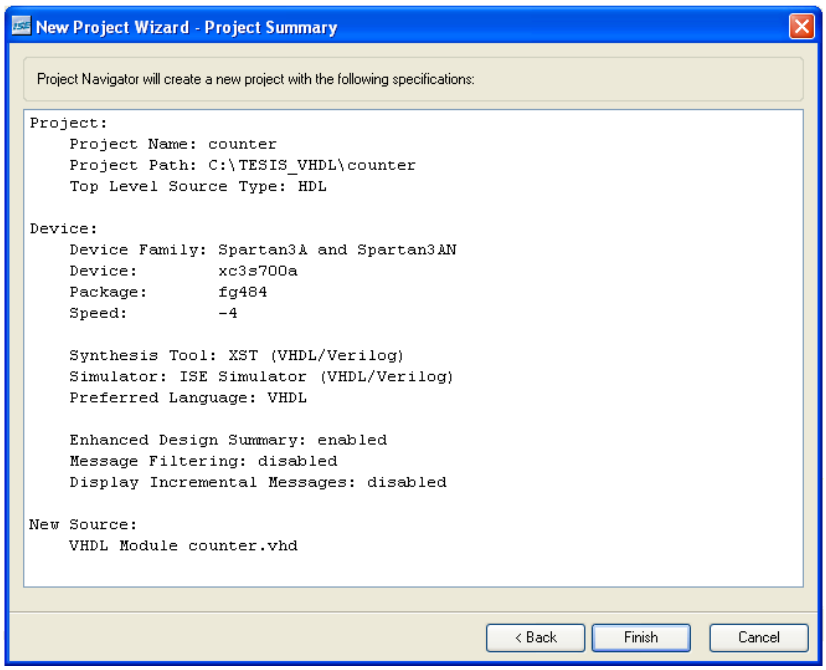


Figura A5- 5. Resumen del proyecto creado en ISE Foundation 10.1

En el espacio de trabajo, aparecerá el archivo fuente creado (en este caso un módulo VHDL) y, a su vez, el archivo se añadirá a la ventana de fuentes. En este punto del proyecto, el diseñador puede comenzar a describir el comportamiento del circuito en el archivo fuente desplegado en el espacio de trabajo. Como ejemplo, se muestra en la Figura A5-6 el diseño de un contador módulo 16, el cual se incrementa en cada transición positiva de la señal de reloj y consta de una

señal de reset asincrónica. Después de concluir con la descripción del sistema en un lenguaje HDL (etapa de Diseño Lógico), se debe ejecutar el proceso de *Síntesis* en la *Ventana de Procesos*, en el cual se verifica la sintaxis del lenguaje HDL y se creará el archivo *netlist* necesario para los procesos de implementación.

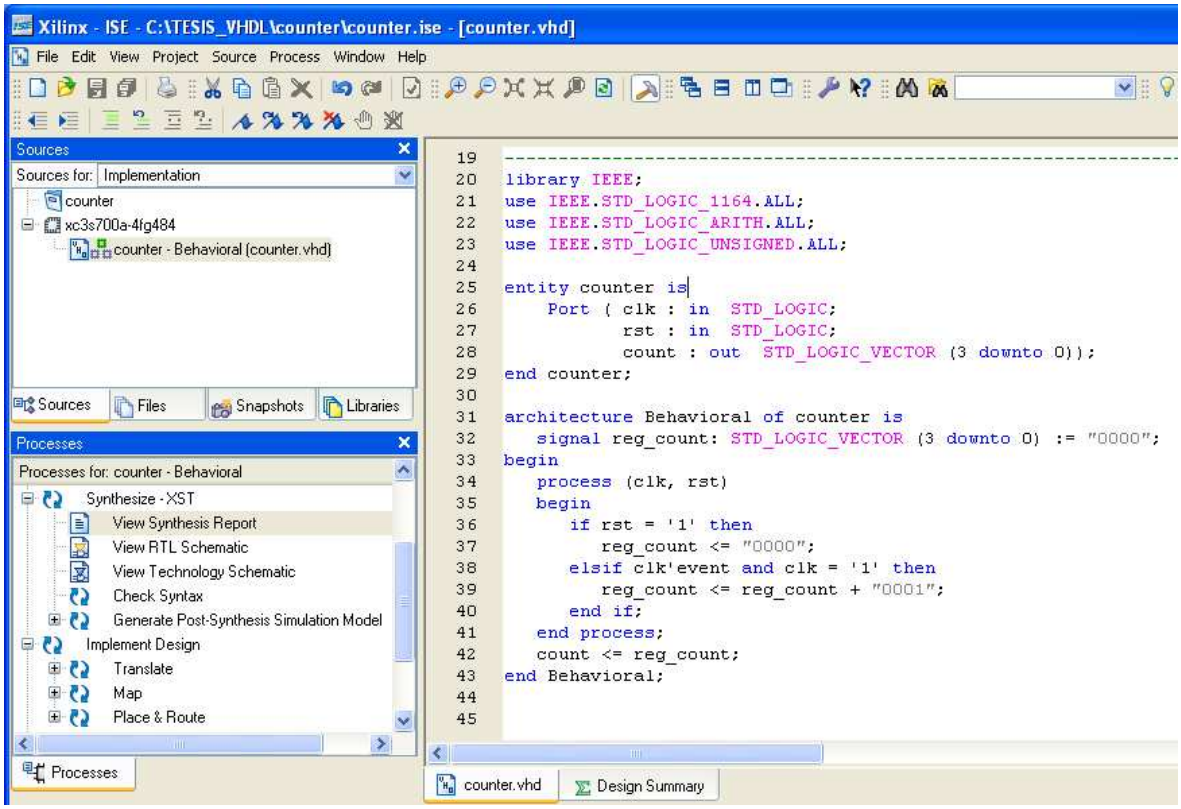


Figura A5- 6. Espacio de trabajo de ISE Foundation 10.1 con un archivo fuente VHDL

Para verificar el comportamiento del sistema diseñado, se debe crear un archivo fuente de tipo *Test Bench Waveform*, con el cual se puede especificar con formas de onda, los estímulos que van a actuar sobre el sistema que se ha diseñado. Este tipo de archivo fuente para simulación, es conveniente debido a que las formas de onda graficadas mediante el editor que se indica en la Figura A5-7, son después transformadas en las correspondientes sentencias de código HDL adecuadas para simulación, evitando así tener que realizar un *Test Bench* describiendo los estímulos mediante el uso directo de sentencias de un HDL. Para añadir este archivo fuente, se debe seleccionar *Project/New Source*, escoger *Test Bench Waveform* en el cuadro de diálogo que se despliega y asociarlo a un archivo fuente. Después de esto un cuadro de dialogo permitirá configurar el

tiempo de simulación y los parámetros de la señal de reloj, en caso de que el diseño posea elementos sincrónico; finalizando este asistente se habrá creado el archivo *Test Bench Waveform*. La simulación del diseño se realiza seleccionando la sección *Behavioral Simulation* de la *Ventana de Fuentes* y escogiendo el archivo *Test Bench Waveform* que en el paso previo se creó. Una vez seleccionado la fuente para la simulación, se ejecuta el proceso *Simulate Behavioral Model* en la *Ventana de Procesos*, obteniéndose los resultados de la simulación del contador módulo 16 como lo indica la Figura A5-8.

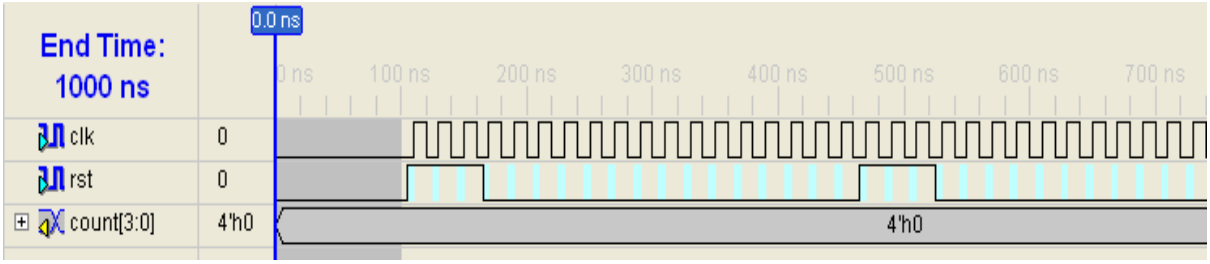


Figura A5- 7. Editor de formas de onda para un Test Bench Waveform

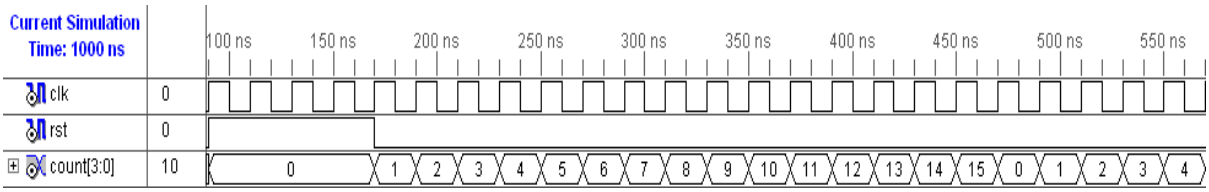


Figura A5- 8. Resultados de la Simulación del contador módulo 16

Terminados los procesos de verificación, se debe realizar la implementación del sistema diseñado, para lo cual, primero se necesita crear un archivo de restricciones de usuario (*User Constraints File*), en el cual, se especifica las asignaciones de los pines del FPGA utilizados en el diseño, así como las condiciones de frecuencia y periodo de reloj (en caso de ser necesario) con el que trabajará el sistema. Un archivo de restricciones de usuario (UCF por sus siglas en inglés) puede ser añadido al proyecto seleccionando del menú *Project/New Source*… y escogiendo en el asistente desplegado el tipo de fuente *Implementation Constraints File*. Al finalizar el asistente, en la *Ventana de Fuentes*, se creará un nuevo archivo fuente, con el cual mediante el editor de

texto se especificará las restricciones para el sistema. Un UCF tiene disponible diversas opciones de restricciones, pero en este ejemplo se utilizará solo dos clases: las restricciones para la asignación y manejo de pines del FPGA y la restricción de tiempo para especificar el período de la señal de reloj. En las restricciones para asignación de pines, es posible especificar el estándar eléctrico de la lógica que maneja el pin, la corriente que maneja el pin, las terminaciones en resistencias de *pull-up* o *pull-down* del pin, entre otras. En la Figura A5-9 se indica el UCF para el contador módulo 16 que se utiliza como ejemplo de diseño.

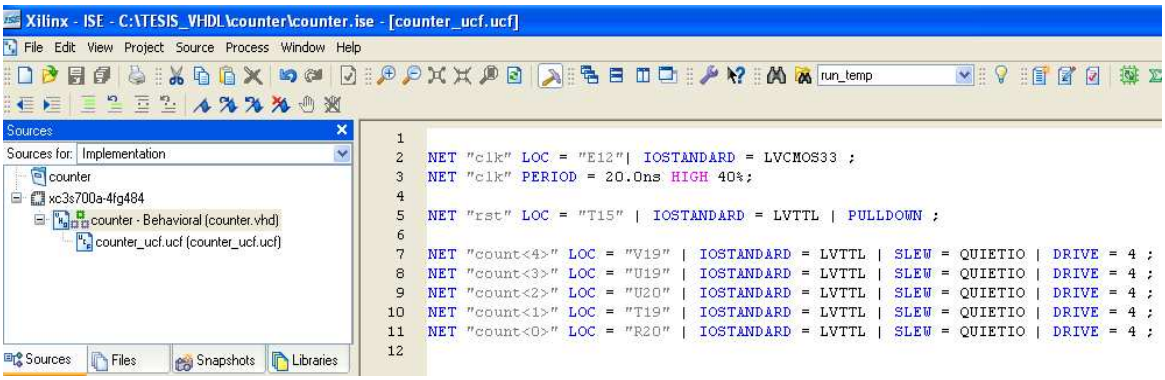


Figura A5- 9. UCF para el contador módulo 16

De la Figura se puede comentar lo siguiente:

- La restricción **NET** hace referencia a un puerto de la entidad de más alta jerarquía en el diseño conectado con un pin del FPGA.
- La restricción **LOC** (para el caso de la implementación física) especifica el pin del FPGA asociado al puerto de la entidad de mayor jerarquía.
- La restricción **IOSTANDARD** especifica el estándar eléctrico de la lógica utilizada por el pin del FPGA.
- La restricción **PULLDOWN** activa la resistencia de pull-down del pin seleccionado en el FPGA. En el caso de requerir una resistencia de pull-up se debe utilizar la restricción **PULLUP**.
- La restricción **SLEW** especifica el comportamiento de la velocidad de transición de voltaje (*slew rate*[29]) del pin y puede ser utilizado en pines de

[29] **Slew Rate:** En sistemas digitales se define como el tiempo de transición de una señal digital entre dos puntos fijos de medida.

entrada, salida o bidireccionales. Las opciones disponibles para esta restricción son *SLOW, FAST y QUIETIO*.

- La restricción **DRIVE** define la corriente que maneja el buffer de entrada/salida del pin del FPGA cuando éste tiene un estándar eléctrico de tipo LVTTL, LVCMOS12, LVCMOS15, LVCMOS18, LVCMOS25, o LVCMOS33 [17].
- La restricción **PERIOD** define la velocidad de la señal de reloj. Puede ser expresada mediante la duración del período (ps, ns, us, ms) o con un valor de frecuencia (GHz, MHz, KHz). También puede definir el ciclo de trabajo mediante la opción HIGH.

Después de tener especificadas las restricciones de usuario en el UCF, se pueden ejecutar los procesos de implementación *Translate*, *Mapping* y *Place & Route* en la *Ventana de Procesos*, como se indica en la Figura A5-10, luego de lo cual, es necesario verificar el diseño mediante una nueva simulación, en la que se compruebe el comportamiento del diseño a la velocidad de funcionamiento especificada en el UCF (Simulación de Tiempos). Esto se hace seleccionando en la *Ventana de Fuentes Post-Route Simulation* y ejecutando el proceso de *Simulate Post-Place&Route Model*. La Simulación de Tiempos utiliza el mismo archivo *Test Bench Waveform* de la simulación funcional, pero el modelo bajo prueba es el que se ha generado en el proceso de *Back Annotation*, que es automáticamente ejecutado en el proceso de simulación mencionado.

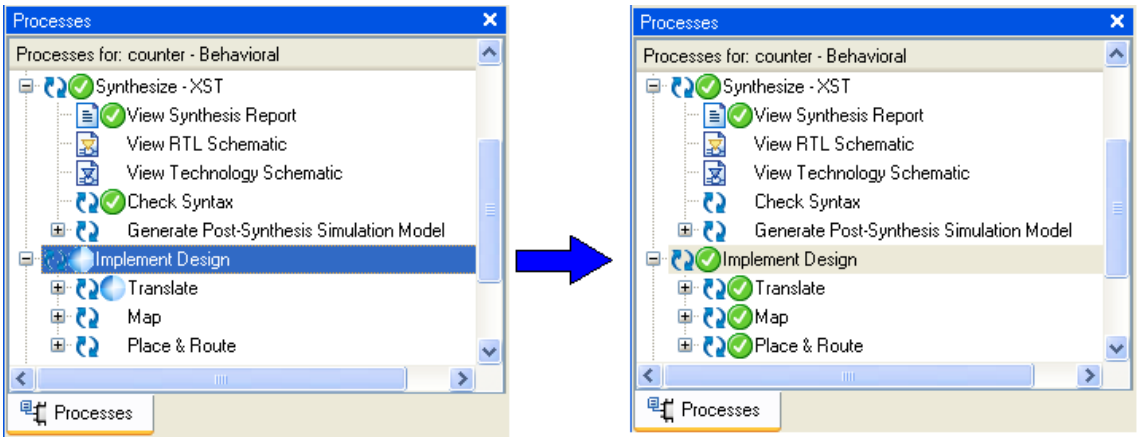


Figura A5- 10. Ejecución de los procesos de implementación para el contador módulo 16

Después de haber ejecutado los procesos antes descritos, se debe generar el *bitstream* que será descargado en el FPGA, esto se logra ejecutando el proceso *Generate Programming File* en la *Ventana de Procesos*. Finalmente, se debe realizar la descarga del *bitstream* de configuración en el FPGA, ejecutando el proceso *Configure Target Device*, el cual utiliza la herramienta *Impact* que se encuentra integrada en el paquete de software ISE Foundation y es la que realiza el proceso de configuración del dispositivo. Antes de iniciar la herramienta, es necesario que el módulo de desarrollo esté conectado al PC; cuando se ejecuta la herramienta *Impact*, se detecta automáticamente el FPGA en el módulo y las memorias Flash PROM de Xilinx. En la Figura A5-11 se muestra un ejemplo de los dispositivos detectados, cuando está conectado el módulo de desarrollo Spartan-3A Starter Kit. Al tener la cadena de los dispositivos detectados, el software automáticamente despliega un cuadro de diálogo sugiriendo el *bitstream* para la configuración del FPGA. Luego de seleccionar el archivo con extensión *.bit* el software busca los archivos de configuración para los otros dispositivos en la cadena, los cuales se pueden omitir seleccionando *Bypass* en el asistente.

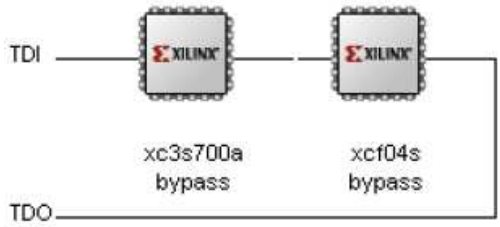


Figura A5- 11. Dispositivos detectados por la herramienta Impact

Luego de que se han especificado los archivos de configuración, se puede programar el dispositivo escogido de la cadena, seleccionando *Operations/Program*. Esta operación despliega un cuadro de diálogo para configurar las propiedades de programación; se puede dejar las propiedades por defecto. Al presionar *Ok*, se iniciará el proceso de programación del dispositivo seleccionado. Cuando el proceso de programación ha finalizado correctamente, aparecerá un indicador como el que se muestra en la siguiente figura.

Figura A5- 12. Programación exitosa del dispositivo

La información generada en cada proceso, pueden ser accedida a manera de resumen seleccionando *View Design Summary* en la *Ventana de Procesos*, lo cual despliega en el Espacio de Trabajo, un informe debidamente tabulado con información sobre parámetros generales del diseño, la utilización de los recursos del dispositivo seleccionado, el cumplimiento de las restricciones de usuario y el reporte detallado de cada proceso ejecutado en el proyecto. La Figura A5-13 indica el resumen del diseño para el contador módulo 16 utilizado como ejemplo.

counter Project Status (01/23/2010 - 21:19:47)			
Project File:	counter.ise	**Current State:**	Programming File Generated
Module Name:	counter	**• Errors:**	No Errors
Target Device:	xc3s700a-4fg484	**• Warnings:**	No Warnings
Product Version:	ISE 10.1.03 - Foundation	**• Routing Results:**	All Signals Completely Routed
Design Goal:	Balanced	**• Timing Constraints:**	All Constraints Met
Design Strategy:	Xilinx Default (unlocked)	**• Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	4	11,776	1%	
Number of 4 input LUTs	3	11,776	1%	
Logic Distribution				
Number of occupied Slices	2	5,888	1%	
Number of Slices containing only related logic	2	2	100%	
Number of Slices containing unrelated logic	0	2	0%	
Total Number of 4 input LUTs	3	11,776	1%	
Number of bonded IOBs				
Number of bonded	6	372	1%	
Number of BUFGMUXs	1	24	4%	

Performance Summary			[-]
Final Timing Score:	0	**Pinout Data:**	Pinout Report
Routing Results:	All Signals Completely Routed	**Clock Data:**	Clock Report
Timing Constraints:	All Constraints Met		

Detailed Reports					[-]
Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Synthesis Report	Current	jue 21. ene 21:28:20 2010	0	0	0
Translation Report	Current	jue 21. ene 21:28:26 2010	0	0	0
Map Report	Current	jue 21. ene 21:28:33 2010	0	0	3 Infos
Place and Route Report	Current	jue 21. ene 21:28:44 2010	0	0	0
Static Timing Report	Current	jue 21. ene 21:28:47 2010	0	0	2 Infos
Bitgen Report	Current	sáb 23. ene 21:19:46 2010	0	0	0

Figura A5- 13. Resumen del Diseño del contador módulo 16

ANEXO 6

TABLAS DE CÓDIGOS DE HUFFMAN RECOMENDADAS EN EL ESTÁNDAR JPEG PARA LA COMPONENTE DE LUMINANCIA

Tablas de Huffman para los coeficientes DC diferencial de la componente de Luminancia

Categoría	Longitud del código	Palabra Código
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110

Tablas de Huffman para los coeficientes AC de la componente de Luminancia

Tabla 1 de 3

Relación R/C	Longitud del Código	Palabra Código	Relación R/C	Longitud del Código	Palabra Código
0/0 (EOB)	4	1010	1/7	16	1111111110000101
0/1	2	00	1/8	16	1111111110000110
0/2	2	01	1/9	16	1111111110000111
0/3	3	100	1/10	16	1111111110001000
0/4	4	1011	2/1	5	11100
0/5	5	11010	2/2	8	11111001
0/6	7	1111000	2/3	10	1111110111
0/7	8	11111000	2/4	12	111111110100
0/8	10	1111110110	2/5	16	1111111110001001
0/9	16	1111111110000010	2/6	16	1111111110001010
0/10	16	1111111110000011	2/7	16	1111111110001011
1/1	4	1100	2/8	16	1111111110001100
1/2	5	11011	2/9	16	1111111110001101
1/3	7	1111001	2/10	16	1111111110001110
1/4	9	111110110	3/1	6	111010
1/5	11	11111110110	3/2	9	111110111
1/6	16	1111111110000100	3/3	12	111111110101

Tablas de Huffman para los coeficientes AC de la componente de Luminancia

Tabla 2 de 3

Relación R/C	Longitud del Código	Palabra Código	Relación R/C	Longitud del Código	Palabra Código
3/4	16	1111111110001111	7/1	8	11111010
3/5	16	1111111110010000	7/2	12	111111110111
3/6	16	1111111110010001	7/3	16	1111111110101110
3/7	16	1111111110010010	7/4	16	1111111110101111
3/8	16	1111111110010011	7/5	16	1111111110110000
3/9	16	1111111110010100	7/6	16	1111111110110001
3/10	16	1111111110010101	7/7	16	1111111110110010
4/1	6	111011	7/8	16	1111111110110011
4/2	10	1111111000	7/9	16	1111111110110100
4/3	16	1111111110010110	7/10	16	1111111110110101
4/4	16	1111111110010111	8/1	9	111111000
4/5	16	1111111110011000	8/2	15	111111111000000
4/6	16	1111111110011001	8/3	16	1111111110110110
4/7	16	1111111110011010	8/4	16	1111111110110111
4/8	16	1111111110011011	8/5	16	1111111110111000
4/9	16	1111111110011100	8/6	16	1111111110111001
4/10	16	1111111110011101	8/7	16	1111111110111010
5/1	7	1111010	8/8	16	1111111110111011
5/2	11	11111110111	8/9	16	1111111110111100
5/3	16	1111111110011110	8/10	16	1111111110111101
5/4	16	1111111110011111	9/1	9	111111001
5/5	16	1111111110100000	9/2	16	1111111110111110
5/6	16	1111111110100001	9/3	16	1111111110111111
5/7	16	1111111110100010	9/4	16	1111111111000000
5/8	16	1111111110100011	9/5	16	1111111111000001
5/9	16	1111111110100100	9/6	16	1111111111000010
5/10	16	1111111110100101	9/7	16	1111111111000011
6/1	7	1111011	9/8	16	1111111111000100
6/2	12	111111110110	9/9	16	1111111111000101
6/3	16	1111111110100110	9/10	16	1111111111000110
6/4	16	1111111110100111	10/1	9	111111010
6/5	16	1111111110101000	10/2	16	1111111111000111
6/6	16	1111111110101001	10/3	16	1111111111001000
6/7	16	1111111110101010	10/4	16	1111111111001001
6/8	16	1111111110101011	10/5	16	1111111111001010
6/9	16	1111111110101100	10/6	16	1111111111001011
6/10	16	1111111110101101	10/7	16	1111111111001100

Tablas de Huffman para los coeficientes AC de la componente de Luminancia

Tabla 3 de 3

Relación R/C	Longitud del Código	Palabra Código	Relación R/C	Longitud del Código	Palabra Código
10/8	16	1111111111001101	14/7	16	1111111111110001
10/9	16	1111111111001110	14/8	16	1111111111110010
10/10	16	1111111111001111	14/9	16	1111111111110011
11/1	10	1111111001	14/10	16	1111111111110100
11/2	16	1111111111010000	15/0 (ZRL)	11	11111111001
11/3	16	1111111111010001	15/1	16	1111111111110101
11/4	16	1111111111010010	15/2	16	1111111111110110
11/5	16	1111111111010011	15/3	16	1111111111110111
11/6	16	1111111111010100	15/4	16	1111111111111000
11/7	16	1111111111010101	15/5	16	1111111111111001
11/8	16	1111111111010110	15/6	16	1111111111111010
11/9	16	1111111111010111	15/7	16	1111111111111011
11/10	16	1111111111011000	15/8	16	1111111111111100
12/1	10	1111111010	15/9	16	1111111111111101
12/2	16	1111111111011001	15/10	16	1111111111111110
12/3	16	1111111111011010			
12/4	16	1111111111011011			
12/5	16	1111111111011100			
12/6	16	1111111111011101			
12/7	16	1111111111011110			
12/8	16	1111111111011111			
12/9	16	1111111111100000			
12/10	16	1111111111100001			
13/1	11	11111111000			
13/2	16	1111111111100010			
13/3	16	1111111111100011			
13/4	16	1111111111100100			
13/5	16	1111111111100101			
13/6	16	1111111111100110			
13/7	16	1111111111100111			
13/8	16	1111111111101000			
13/9	16	1111111111101001			
13/10	16	1111111111101010			
14/1	16	1111111111101011			
14/2	16	1111111111101100			
14/3	16	1111111111101101			
14/4	16	1111111111101110			
14/5	16	1111111111101111			
14/6	16	1111111111110000			

ANEXO 7

MARCADORES DEL FORMATO DE ARCHIVO JFIF

En la Tabla A7-1 se muestran los marcadores definidos en el formato de archivo JFIF

Tabla A7- 1. Marcadores del formato de archivo JFIF

MARCADORES AUTÓNOMOS		
Valor	**Símbolo usado en el estándar JPEG**	**Descripción**
FF01	TEM	Temporal para codificación aritmética
FFD0-FFD7	RST_0-RST_7	Marcado de restablecimiento
FFD8	SOI	Inicio de Imagen
FFD9	EOI	Final de Imagen
MARCADORES NO AUTÓNOMOS		
Valor	**Símbolo usado en el estándar JPEG**	**Descripción**
FFC0	SOF_0	Inicio de trama, Baseline
FFC1	SOF_1	Inicio de trama, secuencial extendido
FFC2	SOF_2	Inicio de trama, progresivo
FFC3	SOF_3	Inicio de trama, codificación sin perdidas
FFC4	DHT	Definición de Tablas de Huffman
FFC5	SOF_5	Inicio de trama, secuencial diferencial
FFC6	SOF_6	Inicio de trama, progresivo diferencial
FFC7	SOF_7	Inicio de trama, sin perdidas diferencial
FFC8	JPG	Reservado
FFC9	SOF_9	Inicio de trama, secuencial extendido, codificación aritmética
FFCA	SOF_{10}	Inicio de trama, progresivo, codificación aritmética
FFCB	SOF_{11}	Inicio de trama, sin perdidas, codificación aritmética
FFCC	DAC	Definición de condiciones de codificación aritmética
FFCD	SOF_{13}	Inicio de trama, secuencial diferencial, codificación aritmética
FFCE	SOF_{14}	Inicio de trama, progresivo diferencial, codificación aritmética
FFCF	SOF_{15}	Inicio de trama, sin pérdidas diferencial, codificación aritmética
FFDA	SOS	Inicio de exploración
FFDB	DQT	Definición de tablas de cuantización
FFDC	DNL	Definición de número de líneas
FFDD	DRI	Definición de intervalo de restablecimiento
FFDE	DHP	Definición de progresión jerárquica
FFDF	EXP	Expandir componentes de referencia
FFE0-FFEF	APP_0-APP_{15}	Datos de aplicación específica
FFFE	COM	Comentario
FFF0-FFFD	JPG_0-JPG_{13}	Reservado
FF02-FFBF	RES	Reservado

Como se observa en la Tabla A7-1, el formato de JFIF proporciona varios tipos de marcadores, dependiendo del estilo de codificación JPEG que se utilice.

ANEXO 8

EL LENGUAJE DE DESCRIPCIÓN DE HARDWARE VHDL

ESTILOS DE DESCRIPCIÓN DEL LENGUAJE VHDL

Dependiendo del nivel de abstracción utilizado en la descripción de un sistema digital, VHDL tiene tres estilos de descripción los cuales se muestran en la Tabla A8-1.

Tabla A8- 1. Niveles de abstracción y estilos descriptivos en VHDL

Nivel de abstracción	Estilo descriptivo
Funcional o comportamental	Algorítmico
Transferencia de registros	Flujo de datos
Lógico o de compuertas	Estructural

- **Algorítmico:** Hace referencia a descripciones similares a los programas software, que reflejan la funcionalidad del módulo, componente o circuito en forma de uno o más procesos concurrentes que contienen descripciones secuenciales del algoritmo correspondiente.
- **Flujo de datos**: Descripciones basadas en ecuaciones y expresiones que reflejan el flujo de información y las dependencias entre datos y operaciones. En este estilo ya existe una correspondencia directa entre el código y su implementación en hardware.
- **Estructural:** Se reflejan directamente componentes por referencia y conexiones entre ellos a través de sus puertos de entrada/salida.

ESTRUCTURA DEL CÓDIGO

Una descripción en VHDL está formada por tres secciones fundamentales: declaración de librerías, entidad y arquitectura.

Una librería es una colección de piezas de código usadas frecuentemente. Los códigos son usualmente escritos en forma de funciones, procedimientos o componentes, los cuales son puestos dentro de un paquete (*package*) y son compilados en una librería de destino.

La entidad define la interfaz del sistema electrónico con su entorno. Especifica los pines (puertos) de entradas y salida de nuestro circuito. Su sintaxis es la siguiente:

```
ENTITY entity_name IS
   PORT (
      port_name : signal_mode signal_type;
      port_name : signal_mode signal_type;
      ...);
END entity_name;
```

Donde:

signal_mode: define la dirección de los pines del circuito definido por la entidad y pueden ser: IN (entrada), OUT (salida), INOUT (bidireccional), BUFFER (utilizado cuando la señal de salida debe ser utilizada internamente).

signal_type: Puede ser BIT, STD_LOGIC, INTEGER, etc.

El nombre de la entidad puede ser cualquier palabra excepto las palabras reservadas del VHDL.

La arquitectura es la descripción de cómo el circuito o sistema debe comportarse. Describe un conjunto de operaciones sobre las entradas de la entidad, que determinan el valor de la salida en cada momento. Su sintaxis es la siguiente:

```
ARCHITECTURE architecture_name OF entity_name IS
    [declarations]
BEGIN
    (code)
END architecture_name;
```

Una arquitectura tiene dos partes: una parte declarativa, donde señales, componentes y constantes (entre otros) son declarados, y la parte de código

propiamente dicha (desde *begin* hacia abajo). Al igual que en el caso de una entidad, el nombre de la arquitectura puede ser cualquier palabra, excepto las palabras reservadas del VHDL.

Además de dar un nombre a la arquitectura, debe indicarse el nombre de la entidad a la que pertenece.

TIPOS DE DATOS DEL LENGUAJE VHDL

- **BIT (y BIT_VECTOR)**: dos niveles lógicos (‘0’, ‘1’).
- **STD_LOGIC (y STD_LOGIC_VECTOR):** introduce un sistema lógico de 8 valores.
- **STD_ULOGIC (y STD_ULOGIC_VECTOR):** define un sistema lógico de 9 niveles, añadiendo el valor lógico ‘U’ a los 8 valores de STD_LOGIC.
- **BOOLEAN:** True, False.
- **INTEGER:** Enteros de 32 bits (de -2147483647 a + 2147483647).
- **NATURAL:** Enteros no negativos (0 a + 2147483647).
- **SIGNED y UNSIGNED:** tienen la apariencia de un STD_LOGIC_VECTOR pero aceptan operaciones aritméticas.
- **ARRAYS:** Son colecciones de datos de un mismo tipo. Pueden ser de una dimensión (1D), una dimensión por una dimensión (1Dx1D) o dos dimensiones (2D). Se pueden tener arreglos de más de dos dimensiones pero, por lo general, estos no son sintetizables.

Existen además otros tipos de datos que no son sintetizables como: REAL, MAGNITUDES FISICAS, CADENAS DE CARACTERES. También, el usuario puede definir sus propios datos, los cuales pueden ser un subconjunto de los anteriores, o pueden ser de tipo enumerados (como los estados de una FSM).

TIPOS DE OPERADORES

VHDL proporciona varios tipos de operadores predefinidos:

- Operadores de asignación
  <= usado para asignar un valor a una señal.
  := usado para asignar un valor a una variable, constante o genérico, también es usado para asignar valores iniciales.
  => usado para asignar valores a elementos individuales de un vector o con OTHERS.
- Operadores Lógicos (NOT, AND, OR, NAND, NOR, XOR, XNOR)
- Operadores Aritméticos (+, -, *, /, **)
- Operadores de Comparación (=, /=, <, >, <=, >=)
- Operadores de Desplazamiento (sll, srl)
- Operador de Concatenación (&)

TIPOS DE CODIGO VHDL

En el estilo de descripción algorítmico, dos tipos de código VHDL pueden ser utilizados: código concurrente y código secuencial.

Código Concurrente

Es un conjunto de instrucciones que se ejecutan de forma concurrente (paralela). El código concurrente tiene tres instrucciones: WHEN, GENERATE y BLOCK.

- **Instrucción WHEN.** Se presenta en dos formas: WHEN/ELSE y WITH/SELECT/WHEN. Su sintaxis es la siguiente:

WHEN / ELSE:

```
assignment WHEN condition ELSE
assignment WHEN condition ELSE
...;
```

WITH / SELECT / WHEN:

```
WITH identifier SELECT
assignment WHEN value,
assignment WHEN value,
...;
```

- **Instrucción GENERATE**. Permite a una sección de código repetirse un número de veces específico. Su sintaxis es la siguiente:

FOR / GENERATE:

```
label: FOR identifier IN range GENERATE
  (concurrent assignments)
END GENERATE;
```

- **Instrucción BLOCK.** Esta instrucción sirve como una simple forma de dividir localmente el código. Su sintaxis es la siguiente:

```
label: BLOCK
  [declarative part]
BEGIN
  (concurrent statements)
END BLOCK label;
```

Código Secuencial

Son secciones de código que se ejecutan de manera secuencial, estas secciones se encuentran dentro de procesos (PROCESS), procedimientos (PROCEDURE) y funciones (FUNCTION). Las instrucciones que pueden encontrarse dentro de un proceso (PROCESS), procedimiento (PROCEDURE) o función (FUNCTION) son: IF, WAIT, CASE y LOOP.

- **Procesos**. Un proceso es una sección secuencial de código VHDL, se caracteriza por la presencia de instrucciones IF, WAIT, CASE, o LOOP, y una lista de sensibilidad. Un proceso es ejecutado cada vez que una señal en la lista de sensibilidad cambie. Su sintaxis es la siguiente:

```
[label:] PROCESS (sensitivity list)
  [VARIABLE name type [range] [:= initial_value;]]
BEGIN
  (sequential code)
END PROCESS [label];
```

- **Señales y variables**. VHDL tiene dos maneras de “transportar” valores no estáticos: mediante señales y mediante variables. Una señal puede ser declarada en un paquete, en una entidad o en una arquitectura, mientras una variable sólo puede ser declarada dentro de una sección de código secuencial (por ejemplo un proceso). El valor de una señal puede ser utilizado en todo el diseño (es global), al contrario de una variable que es local. El valor de una variable nunca puede ser pasado fuera de un proceso directamente, es necesario primero asignarlo a una señal antes de salir del proceso.

- **Instrucción IF.** Su sintaxis es la siguiente:

```
IF conditions THEN assignments;
ELSIF conditions THEN assignments;
...
ELSE assignments;
END IF;
```

- **Instrucción WAIT.** Cuando WAIT es utilizado el proceso no puede tener una lista de sensibilidad. Existen tres formas de la instrucción WAIT:

```
WAIT UNTIL signal_condition;
```

```
WAIT ON signal1 [, signal2, ... ];
```

```
WAIT FOR time;
```

- ✓ **Instrucción WAIT UNTIL.** Acepta sólo una señal, por lo que es apropiado para código sincrónico y asincrónico. Ya que el proceso no puede tener una lista de sensibilidad la instrucción WAIT UNTIL debe ser la primera dentro del proceso. El proceso será ejecutado cada vez que la condición sea verdadera.
- ✓ **Instrucción WAIT ON.** Acepta múltiples señales. El proceso es puesto en espera hasta que cualquiera de las señales cambie.
- ✓ **Instrucción WAIT FOR.** Esta instrucción es pensada sólo para simulación (por ejemplo generación de formas de onda en *testbenches*).

- **Instrucción CASE.** Esta instrucción es muy similar a la instrucción concurrente WHEN. Todas las permutaciones deben ser probadas por lo que la palabra clave OTHERS será utilizada. Otra palabra clave importante es NULL, utilizada cuando ninguna acción se debe tomar. La instrucción CASE permite múltiples asignaciones por cada condición probada, al contrario de su contraparte concurrente (WHEN) que solamente admite una. Su sintaxis es la siguiente:

```
CASE identifier IS
    WHEN value => assignments;
    WHEN value => assignments;
    ...
END CASE;
```

- **Instrucción LOOP.** Esta instrucción es útil cuando se necesita repetir una pieza de código varias veces. Al igual que IF, WAIT, y CASE, LOOP es utilizado sólo para código secuencial. Hay varias maneras de utilizar LOOP como se muestra en la siguiente sintaxis:

  ✓ **FOR/LOOP.** El lazo es repetido un número de veces fijo

```
[label:] FOR identifier IN range LOOP
  (sequential statements)
END LOOP [label];
```

  ✓ **WHILE/LOOP.** El lazo se repite hasta cumplir una condición

```
[label:] WHILE condition LOOP
  (sequential statements)
END LOOP [label];
```

  ✓ **EXIT.** Usado para terminar el lazo

```
[label:] EXIT [label] [WHEN condition];
```

  ✓ **NEXT.** Usado para saltar pasos en el lazo

```
[label:] NEXT [loop_label] [WHEN condition];
```

ANEXO 9

CONTENIDO DE LOS MARCADORES PARA LA CABECERA DEL ARCHIVO JFIF

En las siguientes tablas, se indica el valor de los datos que poseen los marcadores de un archivo con formato JFIF, para una imagen monocromática comprimida con el esquema de codificación Baseline del estándar JPEG. Estos valores son los que se encuentran almacenados en la memoria ROM de la cabecera de archivo JFIF en el módulo de compresión del sistema de adquisición, compresión y almacenamiento de imágenes.

Tabla A9- 1. Marcador de Inicio de imagen y de aplicación

Nombre del campo	Tamaño del campo	Valores (HEX)
Marcador de inicio de Imagen (SOI)	2 bytes	FF D8
Marcador de aplicación (APP0)	2 bytes	FF E0
Longitud de los datos del marcador APP0	2 bytes	00 10
Identificador	5 bytes	4A 46 49 46 00
Identificador principal de versión	1 byte	01
Identificador secundario de versión	1 byte	01
Unidades	1 byte	00
Xdensity	2 bytes	00 01
Ydensity	2 bytes	00 01
Xthumbnail	1 byte	00
Ythumbnail	1 byte	00

Tabla A9- 2. Marcador de inicio de Trama

Nombre del Campo	Tamaño del Campo	Valor (HEX)
Marcador de Inicio de Trama (SOF0)	2 bytes	FF C0
Longitud de los datos del marcador SOF0	2 bytes	00 0B
Precisión de muestreo en bits	1 byte	08
Alto de la imagen en pixeles	2 bytes	00 78
Ancho de la imagen en pixeles	2 bytes	00 A0
Número de componentes de la imagen	1 byte	01
Identificador de componente de luminancia	1 byte	01
Frecuencia de muestreo horizontal (4LSb) y vertical (4MSb) de la componente de luminancia	1 byte	11
Identificador de tabla de cuantización	1 byte	00

Tabla A9- 3. Marcador de definición de Tablas de Cuantización

Nombre del campo	**Tamaño del campo**	**Valor (HEX)**	
Marcador de definición de Tabla de cuantización (DQT)	2 bytes	FF DB	
Longitud de los datos del marcador DQT	2 bytes	00 43	
Identificados de la tabla (4MSb) y tamaño de los valores de la tabla (4LSb)	1 bytes	00	
Valores sin signo de 1 byte de la Tabla de Cuantización.	64 bytes	**Para factor de calidad de 25**	20 16 18 1C 18 14 20 1C 1A 1C 24 22 20 26 30 50 34 30 2C 2C 30 62 46 4A 3A 50 74 66 7A 78 72 66 70 6E 80 90 B8 9C 80 88 AE 8A 6E 70 A0 DA A2 AE BE C4 CE D0 CE 7C 9A E2 F2 E0 C8 F0 B8 CA CE C6
		Para factor de calidad de 35	17 10 11 14 11 0E 17 14 13 14 1A 18 17 1B 22 39 25 22 1F 1F 22 46 32 35 29 39 53 49 57 56 51 49 50 4F 5B 67 83 6F 5B 61 7C 63 4F 50 72 9C 74 7C 88 8C 93 95 93 59 6E A1 AD A0 8F AB 83 90 93 8D
		Para factor de calidad de 50	10 0B 0C 0E 0C 0A 10 0E 0D 0E 12 11 10 13 18 28 1A 18 16 16 18 31 23 25 1D 28 3A 33 3D 3C 39 33 38 37 40 48 5C 4E 40 44 57 45 37 38 50 6D 51 57 5F 62 67 68 67 3E 4D 71 79 70 64 78 5C 65 67 63
		Para factor de calidad de 75	08 06 06 07 06 05 08 07 07 07 09 09 08 0A 0C 14 0D 0C 0B 0B 0C 19 12 13 0F 14 1D 1A 1F 1E 1D 1A 1C 1C 20 24 2E 27 20 22 2C 23 1C 1C 28 37 29 2C 30 31 34 34 34 1F 27 39 3D 38 32 3C 2E 33 34 32

Tabla A9- 4. Marcadores de definición de Tablas de Huffman

Nombre del campo	Tamaño del Campo	Valor (HEX)
Marcador de definición de Tablas de Huffman (DHT)	2 bytes	FF C4
Longitud de los datos del marcador DHT	2 bytes	00 1F
Clase de tabla (4MSb) e identificador de la tabla (4LSb)	1 byte	00
Enumeración de códigos de Huffman según su longitud	16 bytes	00 01 05 01 01 01 01 01 01 00 00 00 00 00 00 00
Símbolos abreviados de los códigos de Huffman	12 bytes	00 01 02 03 04 05 06 07 08 09 0A 0B
Marcador de definición de Tablas de Huffman (DHT)	2 bytes	FF C4
Longitud de los datos del marcador DHT	2 bytes	00 B5
Clase de tabla (4MSb) e identificador de la tabla (4LSb)	1 byte	10
Enumeración de códigos de Huffman según su longitud	16 bytes	00 02 01 03 03 02 04 03 05 05 04 04 00 00 01 7D
Símbolos abreviados de los códigos de Huffman	162 bytes	01 02 03 00 04 11 05 12 21 31 41 06 13 51 61 07 22 71 14 32 81 91 A1 08 23 42 B1 C1 15 52 D1 F0 24 33 62 72 82 09 0A 16 17 18 19 1A 25 26 27 28 29 2A 34 35 36 37 38 39 3A 43 44 45 46 47 48 49 4A 53 54 55 56 57 58 59 5A 63 64 65 66 67 68 69 6A 73 74 75 76 77 78 79 7A 83 84 85 86 87 88 89 8A 92 93 94 95 96 97 98 99 9A A2 A3 A4 A5 A6 A7 A8 A9 AA B2 B3 B4 B5 B6 B7 B8 B9 BA C2 C3 C4 C5 C6 C7 C8 C9 CA D2 D3 D4 D5 D6 D7 D8 D9 DA E1 E2 E3 E4 E5 E6 E7 E8 E9 EA F1 F2 F3 F4 F5 F6 F7 F8 F9 FA

Tabla A9- 5. Marcador de inicio de exploración (SOS)

Nombre del Campo	Tamaño del campo	Valor (HEX)
Marcador de inicio de exploración (SOS)	2 bytes	FF DA
Longitud de los datos del marcador SOS	2 bytes	00 08
Contador de componente	1 byte	01
Descriptor de componente	2 bytes	01 00
Inicio de la selección espectral	1 byte	00
Fin de la selección espectral	1 byte	3F
Aproximación sucesiva	1 byte	00

ANEXO 10

CÓDIGO FUENTE DE LA INTERFAZ GRÁFICA DE CONTROL DEL SISTEMA DE ADQUISICIÓN, COMPRESIÓN Y ALMACENAMIENTO DE IMÁGENES

```
%Código fuentes de la Interfaz Gráfica de Control del Sistema de
%Adquisición, Compresión y Almacenamiento de imágenes

function varargout = tesis(varargin)
    clc;
    gui_Singleton = 1;
    gui_State = struct('gui_Name',       mfilename, ...
                       'gui_Singleton',  gui_Singleton, ...
                       'gui_OpeningFcn', @tesis_OpeningFcn, ...
                       'gui_OutputFcn',  @tesis_OutputFcn, ...
                       'gui_LayoutFcn',  [] , ...
                       'gui_Callback',   @gui_callback);

    if nargin && ischar(varargin{1})
        gui_State.gui_Callback = str2func(varargin{1});
    end

    if nargout
        [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
    else
        gui_mainfcn(gui_State, varargin{:});
    end
return;

function tesis_OpeningFcn(hObject, eventdata, handles, varargin)
    %se centra el formulario
    scrsw=get(0,'Screensize');
    pos_act=get(gcf,'Position');
    xr=scrsw(3)-pos_act(3);
    xp=round(xr/2);
    yr=scrsw(4)-pos_act(4);
    yp=round(yr/2);
    set(gcf,'Position',[xp yp pos_act(3) pos_act(4)]);

    handles.output = hObject;
    guidata(hObject, handles);

    %imágenes del formulario
    axes(handles.epn_logo)
    background = imread('logo_epn.jpg');
    axis off;
    imshow(background);
    axes(handles.buho)
    background = imread('buhoEPN.jpg');
    axis off;
    imshow(background);
return;

function varargout = tesis_OutputFcn(hObject, eventdata, handles)
    varargout{1} = handles.output;
return;

function puerto_CreateFcn(hObject, eventdata, handles)
    if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
    end
return
```

```
function port_default_Callback(hObject, eventdata, handles)
    set(handles.puerto,'Value',1);
return

function capturar_Callback(hObject, eventdata, handles)
    %sección de estado borrada
    set(handles.display,'string', '');
    set(handles.rate_jpeg,'string', '');
    pause(0.001)
    %adquisición de imagen a través de la webcam
    try
    %se crea objeto de video
    vid = videoinput('winvideo',1,'RGB24_352x288');
    %adquisición de una trama
    frame = getsnapshot(vid);
    %redimensionamiento y conversión monocromática
    image = rgb2gray(imresize(frame, [120, 160]));
    handles.metricdata.image=image;
    guidata(hObject,handles);
    %información para la seccion de estado
    set(handles.display,'string', 'Imagen Capturada');
   end
return

function comprimir_Callback(hObject, eventdata, handles)
    %sección de estado borrada
    set(handles.display,'string', '');
    set(handles.rate_jpeg,'string', '');
    pause(0.001)
    %división de la imagen en bloques de 8x8 pixeles
    try
    image=handles.metricdata.image;
    a=1;
    b=1;
    c=1;
        for i=1:15
            for j=1:20
                block8x8=image(a:a+7,b:b+7);
                for k=1:8
                    for l=1:8
                       image_vector(c)=block8x8(k,l);
                       c=c+1;
                    end
                end
                b= b+8;
            end
            b=1;
            a=a+8;
        end

    %configuración del objeto serial
    load configuracion_de_puerto s
    handles.metricdata.s=s;
    guidata(hObject,handles)
    fopen(s);

    %barra de espera (wait bar)
    h = waitbar(0,'Please wait...');
```

```
    steps = 19200;
    for step = 1:10:steps
    waitbar(step / steps)
    end
    pause(0.001)
    close(h)
    %envío del caracter "J"
    fwrite(s,'J','uint8');
    %envío de la imagen sin comprimir
    fwrite(s,image_vector,'uint8');

    total_bytes = fread(s,2,'uint8');
    total_double = double(total_bytes);
    MSB_total = dec2binvec(total_double(1));
    LSB_total = dec2binvec(total_double(2),8);
    total_binvec=[LSB_total MSB_total];
    total = binvec2dec(total_binvec);
    %envío del caracter "R"
    fwrite(s,'R','uint8');
    %recepción de la imagen comprimida
    image_jpeg = fread(s,total,'uint8');
    fclose(s);
    delete(s);
    clear s;

    %cálculo de la tasa de compresión
    factor_jpeg=roundn(19200/total,0);
    factor=num2str(factor_jpeg);
    str_factor=[factor ':1'];
    set(handles.rate_jpeg,'string', str_factor);
    handles.metricdata.str_factor=str_factor;
    guidata(hObject,handles);

    %almacenamiento temporal de la imagen comprimida
    %con formato .jpg
    image_trans = (image_jpeg)';
    image_txt=fopen('image_jpeg.jpg','w');
    fwrite(image_txt, image_trans);
    fclose(image_txt);
    %información para la sección de estado
    set(handles.display,'string', 'Compresión finalizada');
    end
return

%Se visualiza imagen original
function disp_orig_Callback(hObject, eventdata, handles)
clc
try
    image = handles.metricdata.image;
    save image_disp image;
    display_image;
   end
return

%Se visualiza imagen comprimida
function disp_jpeg_Callback(hObject, eventdata, handles)
    clc
    try
    imjpeg=imread('image_jpeg.jpg');
```

```
    clear imjpeg;
    str_factor = handles.metricdata.str_factor;
    save str_factor_jpeg str_factor;
    display_jpeg;
    end
return

%Se guarda la imagen comprimida
function save_Callback(hObject, eventdata, handles)
    try
    set(handles.display,'string', '');
    set(handles.rate_jpeg,'string', '');
    pause(0.001)
    t=fopen('image_jpeg.jpg','r');
    image_guardar=fread(t);
    [filename, pathname] = uiputfile('image_jpeg.jpg','Directorio de
destino');
    direccion=cat(2,pathname,filename);
    image_txt=fopen(direccion,'w');
    fwrite(image_txt, image_guardar);
    fclose(image_txt);
    clear t;
    clear image_guardar;
    clear image_txt;
    %información para la sección de estado
    set(handles.display,'string', 'Imagen Guardada');
    end
return

%Se carga la imagen original desde un archivo .bmp
function load_Callback(hObject, eventdata, handles)
    set(handles.display,'string', '');
    set(handles.rate_jpeg,'string', '');
    pause(0.001)
    [filename1, pathname1] = uigetfile('imagenes\*.bmp','Seleccione un
archivo *.bmp');
    if filename1~0
    direccion1=cat(2,pathname1,filename1);
    try
    %se lee los datos de la imagen
    a=imread(direccion1);
    image = rgb2gray(imresize(a, [120, 160]));
    %redimensionamiento y conversión monocromática
    handles.metricdata.image=image;
    guidata(hObject,handles);
    %información para la seccion de estado
    set(handles.display,'string', 'Imagen Cargada');
    catch
    errordlg('No se puede abrir el archivo seleccionado','ERROR...');
    end
    end
return

%función para salir de la interfaz gráfica de control
function Salir(fig_handle, handles, isreset)
    if isfield(handles, 'metricdata') && ~isreset
    return;
    end
```

```
    respuesta=questdlg('Desea salir del programa
?...','SALIR','Si','No','No');
    if strcmp(respuesta,'No')
    return;
    else
    try
    %se borran los archivos generados
    delete('image_disp.mat');
    t=fopen('image_jpeg.jpg','r')
    fwrite(t,0);
    fclose(t)
    delete('image_jpeg.jpg')
    end
    clear;
    closereq;
    close all;
    end
return

%se ejecuta la aplicación JPEGsnoop
function verificar_jpeg_Callback(hObject, eventdata, handles)
    !JPEGsnoop
return

%abre el formulario para configuración del puerto serial
function conf_port_Callback(hObject, eventdata, handles)
    sel_puerto;
return

%función para probar conectividad con el módulo de desarrollo
function test_port_Callback(hObject, eventdata, handles)
    set(handles.display,'string', '');
    set(handles.rate_jpeg,'string', '');
    pause(0.001)
    load configuracion_de_puerto s
    handles.metricdata.s=s;
    guidata(hObject,handles)
    fopen(s);
    %se envía el caracter "T"
    fwrite(s,'T','uint8');
    %se recibe un byte desde el módulo
    test=(fread(s,1,'uint8'))
    fclose(s);
    delete(s);
    clear s;
    a=uint8(test)
    size(test)
    %comparación del byte recibido
    if test==85
        set(handles.display,'string', 'Conexion establecida');
    else
        set(handles.display,'string', 'Conexion fallida');
    end
return

%%Sección de menús

%función para salvar el archivo .jpg generado
function save_im_Callback(hObject, eventdata, handles)
```

```
    try
    t=fopen('image_jpeg.jpg','r');
    image_guardar=fread(t);
    [filename, pathname] = uiputfile('image_jpeg.jpg','Directorio de
destino');
    direccion=cat(2,pathname,filename);
    image_txt=fopen(direccion,'w');
    fwrite(image_txt, image_guardar);
    fclose(image_txt);
    end
return

%función para salir de la interfaz gráfica de control desde el menú
function exit_Callback(hObject, eventdata, handles)
    if isfield(handles, 'metricdata') && ~isreset
    return;
    end
    respuesta=questdlg('Desea salir del programa
?...','SALIR','Si','No','No');
    if strcmp(respuesta,'No')
    return;
    else
    try
    %se borran los archivos generados
    delete('image_disp.mat');
    t=fopen('image_jpeg.jpg','r')
    fwrite(t,0);
    fclose(t)
    delete('image_jpeg.jpg')
    end
    clear;
    closereq;
    close all;
    end
return

%abre el formulario para configuración del puerto serial
function config_puerto_Callback(hObject, eventdata, handles)
    sel_puerto;
return

%Se carga la imagen original desde un archivo .bmp
function load_image_Callback(hObject, eventdata, handles)
    set(handles.display,'string', '');
    set(handles.rate_jpeg,'string', '');
    pause(0.001)
    [filename1, pathname1] = uigetfile('imagenes\*.bmp','Seleccione un
archivo *.bmp');
    if filename1~0
        direccion1=cat(2,pathname1,filename1);
    try
        a=imread(direccion1);
        image = rgb2gray(imresize(a, [120, 160]));
        handles.metricdata.image=image;
        guidata(hObject,handles);
        set(handles.display,'string', 'Imagen Cargada');
    catch
        errordlg('No se puede abrir el archivo seleccionado','ERROR...');
    end
```

```
    end
return

 %adquisición de imagen a través de la webcam
function cap_image_Callback(hObject, eventdata, handles)
    set(handles.display,'string', '');
    set(handles.rate_jpeg,'string', '');
    pause(0.001)
    try
    vid = videoinput('winvideo',1,'RGB24_352x288');
    frame = getsnapshot(vid);
    image = rgb2gray(imresize(frame, [120, 160]));
    handles.metricdata.image=image;
    guidata(hObject,handles);
    set(handles.display,'string', 'Imagen Capturada');
    end
 return

function com_image_Callback(hObject, eventdata, handles)
    %sección de estado borrada
    set(handles.display,'string', '');
    set(handles.rate_jpeg,'string', '');
    pause(0.001)
    try
    image=handles.metricdata.image;
    %división de la imagen en bloques de 8x8 pixeles
    a=1;
    b=1;
    c=1;
        for i=1:15
            for j=1:20
                block8x8=image(a:a+7,b:b+7);
                for k=1:8
                    for l=1:8
                       image_vector(c)=block8x8(k,l);
                       c=c+1;
                    end
                end
                b= b+8;
            end
            b=1;
            a=a+8;
        end

    %configuración del objeto serial
    load configuracion_de_puerto s
    handles.metricdata.s=s;
    guidata(hObject,handles)
    fopen(s);

    h = waitbar(0,'Please wait...');
    steps = 19200;
    for step = 1:10:steps
        waitbar(step / steps)
    end
    pause(0.001)
    close(h)
    %envio del caracter "J"
    fwrite(s,'J','uint8');
```

```
    %envio de la imagen sin comprimir
    fwrite(s,image_vector,'uint8');

    total_bytes = fread(s,2,'uint8')
    total_double = double(total_bytes);
    MSB_total = dec2binvec(total_double(1));
    LSB_total = dec2binvec(total_double(2),8);
    total_binvec=[LSB_total MSB_total];
    total = binvec2dec(total_binvec);
    %envío del caracter "R"
    fwrite(s,'R','uint8');
    %recepción de la imagen comprimida
    image_jpeg = fread(s,total,'uint8')
    fclose(s);
    delete(s);
    clear s;

    %calculo de la tasa de compresión
    factor_jpeg=roundn(19200/total,0);
    factor=num2str(factor_jpeg);
    str_factor=[factor ':1'];
    set(handles.rate_jpeg,'string', str_factor);
    handles.metricdata.str_factor=str_factor;
    guidata(hObject,handles);

    %almacenamiento temporal de la imagen comprimida
    %con formato .jpg
    image_trans = (image_jpeg)';
    image_txt=fopen('image_jpeg.jpg','w');
    fwrite(image_txt, image_trans);
    fclose(image_txt);
    %información para la seccion de estado
    set(handles.display,'string', 'Compresión finalizada');
    end
return

%se ejecuta la aplicación JPEGsnoop
function ver_jpeg_Callback(hObject, eventdata, handles)
   !JPEGsnoop
return

%se visualiza la imagen original
function vis_original_Callback(hObject, eventdata, handles)
    clc
    try
    image = handles.metricdata.image;
    save image_disp image;
    display_image;
   end
return

%se visualiza la imagen comprimida
function vis_comprimida_Callback(hObject, eventdata, handles)
    clc
    try
    imjpeg=imread('image_jpeg.jpg');
    clear imjpeg;
    str_factor = handles.metricdata.str_factor;
    save str_factor_jpeg str_factor;
```

```
    display_jpeg;
    end
return

%se despliega información del proyecto de tesis
function help_Callback(hObject, eventdata, handles)
    help_sys
return

%función para probar conectividad con el módulo de desarrollo
function test_port_tag_Callback(hObject, eventdata, handles)
set(handles.display,'string', '');
    set(handles.rate_jpeg,'string', '');
    pause(0.001)
    load configuracion_de_puerto s
    handles.metricdata.s=s;
    guidata(hObject,handles)
    fopen(s);
    fwrite(s,'T','uint8');
    test=(fread(s,1,'uint8'))
    fclose(s);
    delete(s);
    clear s;
    a=uint8(test)
    size(test)
    if test==85
        set(handles.display,'string', 'Conexion establecida');
    else
        set(handles.display,'string', 'Conexion fallida');
    end
return
```

ANEXO 11

CÓDIGO VHDL DE LAS SECCIONES DE HARDWARE DEL SISTEMA DE ADQUISICIÓN, COMPRESIÓN Y ALMACENAMIENTO DE IMÁGENES