Diseño e implementación de un sistema de adquisición, compresión y almacenamiento de imágenes empleando VHDL y FPGAs

Pablo X. Jiménez e Iván M. Bernal
Escuela Politécnica Nacional

***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, como lo es el 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 usuaria, desarrollada con el entorno de programación gráfica GUIDE de Matlab, para la interacción con el sistema diseñado y la visualización de resultados.**

***Términos para indexación*—Codificación JPEG Baseline, Estilos descriptivos, FPGA, GUIDE, JPEG, VHDL.**

I. INTRODUCCIÓN A LOS FPGAS

LOS FPGAs hacen su aparición a mediados de la década de los años ochenta 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]. En la actualidad, estos dispositivos han llegado a un grado muy alto de sofisticación, pudiendo 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) [2], ampliando los campos de aplicación de estos dispositivos, como por ejemplo en sistemas de comunicaciones, procesamiento de imágenes y video, redes neuronales artificiales, procesamiento digital de señales, por mencionar unos pocos.

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 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 (bloques de interconexión y líneas de interconexión).* La Fig. 1 indica la arquitectura básica de un FPGA.

Los bloques lógicos configurables (CLB) están constituidos por *Look-Up Tables* (LUTs), por multiplexores y por elementos de almacenamiento tipo flip-flops que permiten sincronizar la salida del CLB con una señal de reloj.

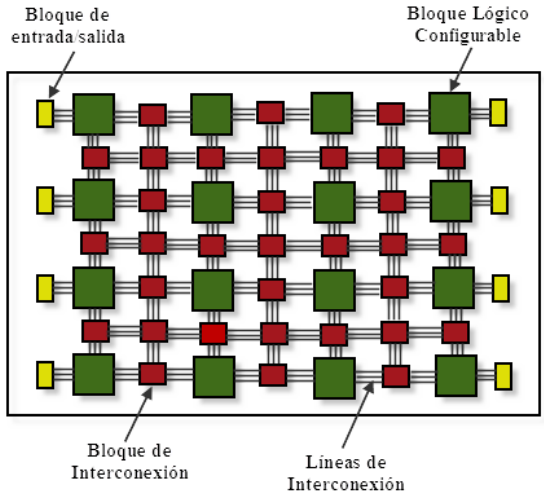


Fig. 1. Arquitectura básica de un FPGA

Este proyecto se realizó en la Escuela Politécnica Nacional (EPN), en el Departamento de Electrónica, Telecomunicaciones y Redes de Información.

P. X. Jiménez participó en el proyecto por la Escuela Politécnica Nacional (e-mail: pablo_pxje85@hotmail.com).

I. M. Bernal trabaja en la Escuela Politécnica Nacional en el Departamento de Electrónica, Telecomunicaciones y Redes de Información, Ladrón de Guevara E11-253, Quito-Ecuador (teléfono: 5932-2507-144; fax: 5932-2547-175; e-mail: imbernal@mailfie.epn.edu.ec)

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.

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. 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.

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.

Las nuevas generaciones de dispositivos FPGAs presentan, además de los bloques tradicionales expuestos anteriormente, bloques embebidos que realizan una función específica. Entre los bloques embebidos más comunes, se pueden mencionar: RAM embebida, Multiplicadores, Bloques DSPs, Administradores de Reloj (Digital Clock Managers – DCM), Microprocesadores Embebidos y Transceivers de alta Velocidad.

Para el caso particular de los FPGAs de la Familia Spartan-3A de Xilinx, la arquitectura de estos dispositivos está constituida por 6 bloques funcionales, a saber:

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

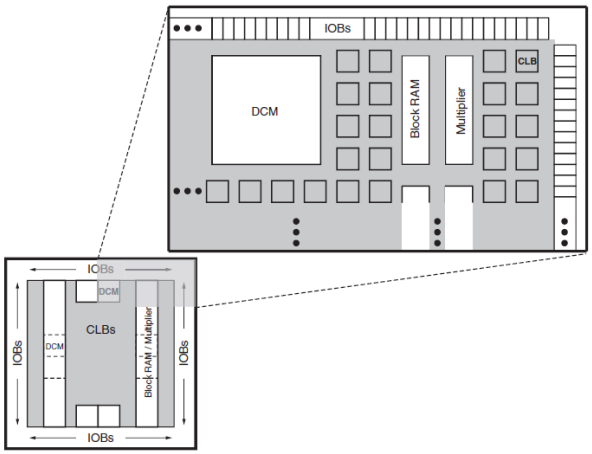
En la Fig. 2 se puede observar la arquitectura de los FPGAs de la Familia Spartan-3A.

Los CLBs de los FPGAs de la Familia Spartan-3A, están constituidos por cuatros *slices*, y cada *slice* está constituido por dos LUTs, dos flip-flop tipo D y recursos lógicos dedicados (compuertas, multiplexores e interconexiones) para optimizar la implementación de operaciones aritméticas. En un CLB, cuatro de los ocho LUTs disponibles en los cuatro *slices* pueden implementar funciones de memoria RAM distribuida. Los bloques de entrada/salida de los FPGAs de esta familia, pueden soportar lógicas de diferentes estándares eléctricos, así como también, soportar transmisión de doble tasa de datos (DDR).

La familia de FPGAs Spartan-3A de Xilinx, posee tres tipos de bloques embebidos:

Bloques de RAM embebida. Estos bloques se encuentran distribuidos en columnas en medio del arreglo de CLBs, y físicamente son memorias *dual-port* de 18Kbits de capacidad y su relación de aspecto puede ser variable.

Multiplicadores embebidos. Estos bloques son encargados de generar el producto de dos números binarios de 18 bits en complemento a 2, en un ciclo de reloj.



Referencia [3]

Fig. 2. Arquitectura de un FPGA de la Familia Spartan-3A de Xilinx

Administradores Digitales del Reloj (DCMs). Mediante estos bloques se puede sintetizar nuevas señales de reloj con diferentes frecuencias, así como variar la fase de la señal de reloj de entrada en pasos fijos y variables.

La Matriz de Interconexión de los FPGAs de la Familia Spartan-3A, posee cuatro tipos de líneas de interconexión que dan la flexibilidad necesaria para la interconexión de todos los bloques funcionales de la arquitectura, en términos de distancia entre bloques y en función de la frecuencia de la señal transmitida.

En la Tabla 1 se resumen los recursos lógicos que posee el FPGA XC3S700A de la Familia Spartan-3A de Xilinx

TABLA 1
RECURSOS LÓGICOS DEL FPGA XC3S700A

Recurso lógico	Capacidad
Sistema de Compuertas	700K
Bloques Lógicos Configurables	1472
Slices	5888
Celdas Lógicas	13248
RAM distribuida	92Kb
RAM embebida	360Kb
Multiplicadores	20
DCMs	8
Bloques I/O	372

II. EL LENGUAJES VHDL Y SUS ESTILOS DESCRIPTIVOS

Para proporcionar una noción de las construcciones del lenguaje VHDL que se pueden emplear para describir el hardware, se presenta una breve introducción al lenguaje.

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 2.

TABLA 2
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: Refleja el comportamiento del sistema mediante procesos concurrentes que contienen sentencias secuenciales [4].

Flujo de datos: La descripción se basa en un conjunto de ecuaciones concurrentes, existe una correspondencia directa entre el código y el hardware.

Estructural: La descripción se basa en conexiones de componentes.

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 [5].

La entidad define la interfaz del sistema electrónico con su entorno. Especifica los pines (puertos) de entradas y salida de un 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.

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” [4]. Su sintaxis es la siguiente:

```
ARCHITECTURE architecture_name OF entity_name IS
    [declarations]
BEGIN
    (code)
END architecture_name;
```

Una arquitectura tiene dos partes: una declarativa, donde señales, componentes y constantes (entre otros) son declarados y la otra de código propiamente dicha (desde *begin* hacia abajo). Además de dar un nombre a la arquitectura, debe indicarse el nombre de la entidad a la que pertenece.

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.
- Operadores Lógicos (NOT, AND, OR, NAND, NOR, XOR, XNOR)
- Operadores Aritméticos (+, -, *, /, **)
- Operadores de Comparación (=, /=, <, >, <=, >=)
- Operadores de Desplazamiento (sll, srl)
- Operador de Concatenación (&)

En el estilo de descripción algorítmico, el lenguaje VHDL puede tener dos tipos de código: código concurrente y código secuencial.

El 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,
...;
```

El código secuencial consiste de 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.

La principal sección de código secuencial son los procesos. Un proceso es una sección secuencial de código VHDL que se caracteriza por la presencia de 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, una entidad o una arquitectura, mientras una variable solo 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 CASE. Esta instrucción es muy similar a la instrucción concurrente WHEN. Todas las permutaciones deben ser probadas, esto se logra con la palabra clave OTHERS. Otra palabra clave importante es NULL, utilizada cuando acción alguna debe ejecutarse. La instrucción CASE permite múltiples asignaciones por cada condición probada, al contrario de su contraparte concurrente (WHEN) que solo admite una. Su sintaxis es la siguiente:

```
CASE identifier IS
    WHEN value => assignments;
    WHEN value => assignments;
    ...
END CASE;
```

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];
```

Una característica importante que posee el lenguaje de descripción de hardware VHDL, es que en una descripción pueden convivir los tres estilos descriptivos antes mencionados. Las secciones del sistema de adquisición, compresión y almacenamiento de imágenes, realizadas en VHDL, convergen en una entidad de más alto nivel, como componentes de la misma, haciendo que la descripción de esta entidad sea netamente estructural. El resto de componentes de la entidad de más alto nivel, son en su mayoría descritos combinando los estilos descriptivos algorítmico y estructural.

III. SISTEMA DE ADQUISICIÓN, COMPRESIÓN Y ALMACENAMIENTO DE IMÁGENES

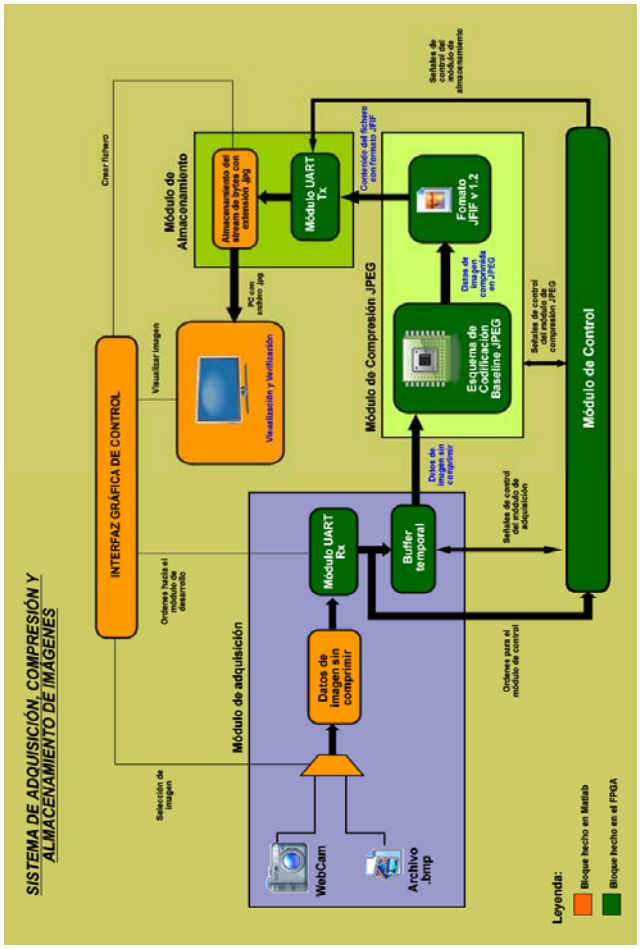


Fig. 3. Arquitectura de un FPGA de la Familia Spartan-3A de Xilinx

El diagrama de bloques del sistema de adquisición, compresión y almacenamiento de imágenes se muestra en la Fig.3.

Algunas funciones en los módulos que se presentan en la Fig.3, 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, 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).

El sistema de adquisición, compresión y almacenamiento de imágenes consta de cinco módulos:

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 y se disponen los datos comprimidos para que cumplan con el formato de intercambio de archivos JFIF. 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.

A. *Módulo de Adquisición*

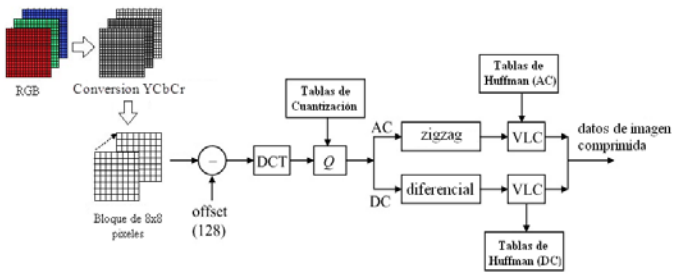
En el módulo de adquisición se obtienen las imágenes sin compresión de dos maneras: desde un archivo de imagen mediante la instrucción *imread* de Matlab o capturando una trama de imagen a través de una webcam mediante la instrucción *getsnapshot* de Matlab. En ambas opciones, la imagen original es convertida a escala de grises (componente de luminancia) y redimensionada a 160x120 pixeles; esto se logra con las instrucciones *rgb2gray* e *imresize* de Matlab.

Los datos de la imagen redimensionada y convertida a escala de grises, son divididos en bloques de 8x8 pixeles, mediante un arreglo de lazos *for* anidados. La imagen dividida en bloques de 8x8 pixeles, es enviada a través del puerto serial al módulo de desarrollo Spartan-3A Starter Kit.

En el módulo de desarrollo, los datos son recibidos por un sistema de recepción UART, y después almacenados en una memoria RAM de 19200 bytes (160x120 bytes) de capacidad. Mientras los datos son almacenados en la memoria RAM, el módulo de control monitorea el puerto de dirección de escritura de la memoria RAM, para determinar el momento en que se tienen todos los datos de la imagen sin comprimir almacenados en la memoria.

B. *Módulo de Compresión*

El módulo de compresión sintetiza a nivel de hardware el esquema de codificación Baseline del estándar JPEG mostrado en la Fig.4.



Referencia [6]

Fig. 4. Diagrama de Bloques del esquema de codificación Baseline del estándar JPEG

Los bloques de 8x8 pixeles, almacenados en la memoria RAM del módulo de adquisición, son recuperados fila por fila. Los elementos de cada fila son almacenados en 8 registros substrayendo previamente 128 a su valor, para desplazar el coeficiente DC al mismo rango de variación de los coeficientes AC (-1023 a +1023) generados en la DCT-2D [7].

El algoritmo utilizado para implementar la transformada discreta del coseno en dos dimensiones, 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. Este algoritmo puede representarse de forma matricial como se indica a continuación:

$$Y = CXC^T$$

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

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

Si $Z = XC^T$, mediante agrupaciones de términos se puede llegar a obtener las expresiones para los elementos de cualquier fila de la matriz Z, como se indica a continuación:

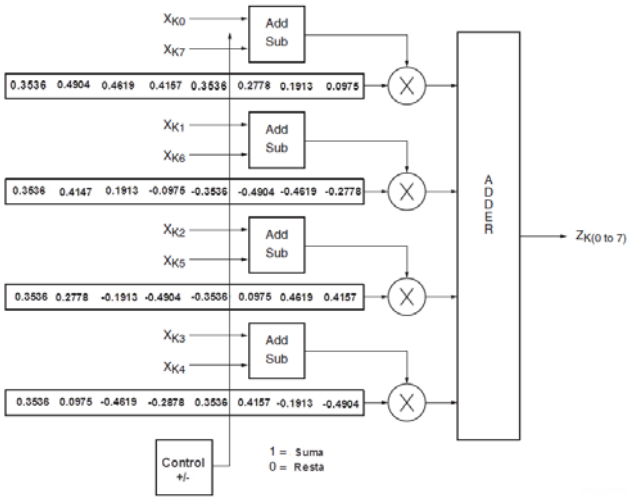
$$
\begin{aligned}
Z_{k0} &= 0.3536(X_{k0} + X_{k1} + X_{k2} + X_{k3} + X_{k4} + X_{k5} + X_{k6} + X_{k7}) \\
Z_{k1} &= 0.4904(X_{k0} - X_{k7}) + 0.4157(X_{k1} - X_{k6}) + 0.2778(X_{k2} - X_{k5}) + 0.0975(X_{k3} - X_{k4}) \\
Z_{k2} &= 0.4619(X_{k0} + X_{k7}) + 0.1913(X_{k1} + X_{k6}) - 0.1913(X_{k2}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

La matriz Z representa la transformada discreta del coseno unidimensional aplicada a las filas del bloque de 8x8 pixeles.

Mediante un procedimiento de agrupaciones parecido al anterior, se puede obtener los elementos de cualquier columna de la matriz Y= C.Z, que representa los coeficientes DCT-2D, como se indica a continuación:

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

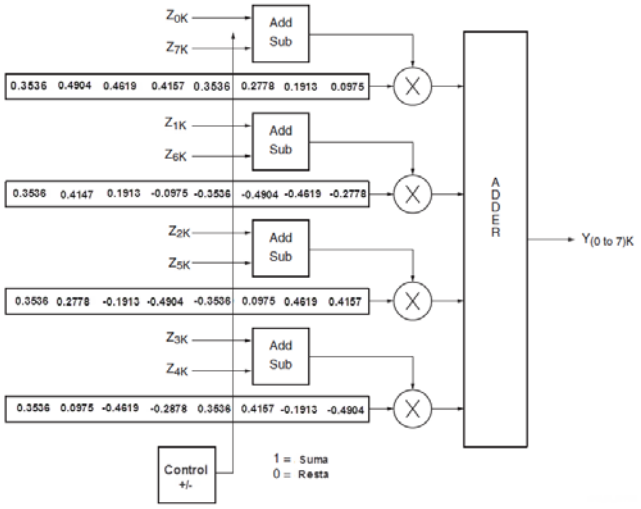
Mediante los dos conjuntos de expresiones indicados anteriormente, se puede inferir la estructura de los sistemas digitales que implementarán las dos transformadas discretas del coseno unidimensionales. La Fig.5 muestra el diagrama de bloques del sistema digital, para implementar la primera transformada discreta del coseno unidimensional, aplicada a las filas del bloque de 8x8 pixeles.



Referencia [8]

Fig. 5. Diagrama de bloques de la implementación de la primera DCT-1D

La Fig.6 muestra el diagrama de bloques del sistema digital, para implementar la segunda transformada discreta del coseno unidimensional, aplicada a las columnas de la matriz Z.



Referencia [8]

Fig. 6. Diagrama de bloques de la implementación de la segunda DCT-1D

Como se observa en la Fig.5 y en la Fig.6, en ambos casos, la implementación de las transformadas discretas del coseno unidimensionales, utilizan el mismo conjunto de constantes provenientes de la matriz C. Esta característica permite obtener un sistema de control que asigna los valores de las constantes dependiendo del número de fila (o columna) que se esté procesando, por ejemplo con un contador módulo 8, que en cada cuenta, asigna un conjunto de constantes específico. También es posible obtener un sistema de control que indique que operación aritmética debe aplicarse a los pares de elementos de cada fila (o columna), por ejemplo una señal basculante que con un 1L indique que se debe sumar los pares de elementos, y con un 0L se deban restar los mismos. Para poder implementar el sistema completo que realiza la transformada discreta del coseno en dos dimensiones, se requiere de una memoria RAM de transposición, en la cual los resultados de la primera DCT-1D, se almacenan fila por fila, y la lectura de los mismos se realiza columna por columna. De esta manera se puede resumir la implementación de la DCT-2D en el diagrama de bloques que se indica en la Fig.7.

Fig. 7. Diagrama de bloques simplificado de la implementación de la DCT-2D

El bloque de la DCT-2D se habilita cuando el módulo de control advierte que se han almacenado en la memoria del módulo de adquisición los 19200 bytes de la imagen sin comprimir. En la descripción VHDL de este bloque, se incluye una bandera que indica el momento en que se comienzan a obtener los coeficientes DCT-2D en el puerto de salida de esta entidad. Esta misma bandera, sirve también para desactivar la señal de *reset* del bloque de cuantización.

El bloque de cuantización consta de dos componentes: una memoria ROM, que almacena las tablas de cuantización de la componente de luminancia, y un divisor para obtener los coeficientes DCT-2D cuantizados. La memoria ROM alberga cuatro tablas de cuantización para factores de calidad de imagen de 25, 35, 50 y 75; el esquema de direccionamiento utilizado para adquirir los elementos de una tabla de cuantización consta de dos bits (los más significativos) para la selección de la tabla y seis bits para apuntar al elemento dentro de la tabla. Mediante dos switches de deslizamiento del módulo de desarrollo Spartan-3A Starter Kit, el usuario del sistema puede seleccionar la tabla de cuantización utilizada y por ende la calidad de la imagen comprimida. Los seis bits restantes de la dirección de memoria, son generados por un contador módulo 64, que se activa con la bandera que indica la validez de los coeficientes DCT-2D. En la Fig.8 se ilustra un diagrama de bloques de la implementación del bloque de cuantización.

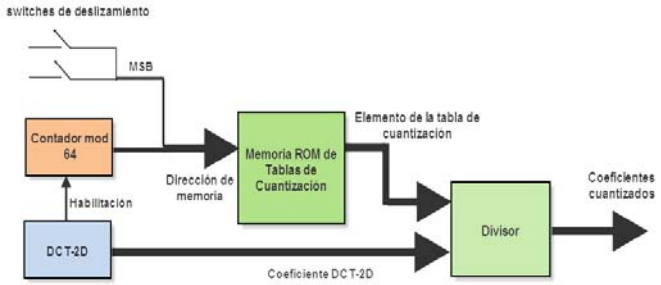


Fig. 8. Diagrama de bloques simplificado de la implementación del bloque de cuantización

El divisor, que se encarga de la cuantización propiamente dicha, divide el coeficiente DCT-2D, generado en el bloque previo, para su correspondiente elemento de la tabla de cuantización y es implementado mediante un IP Core[1]. El IP Core generado, es capaz de realizar una división en cada ciclo de reloj, con una latencia inicial de 25 ciclos de reloj. Después de la latencia inicial, una bandera indica el momento en que los datos a la salida de la entidad de cuantización son válidos. Esa misma bandera se encarga de desactivar la señal de *reset* del bloque de exploración en zig-zag.

El bloque de exploración en zig-zag consiste de una memoria ROM que almacena el orden de exploración de los coeficientes cuantizados, mostrado en la Fig.9, y una memoria RAM de almacenamiento temporal. Cuando la señal de *reset* de este bloque se desactiva, se inicia el generador de direcciones de la memoria ROM. Los datos de salida de esta memoria, actúan como la dirección de almacenamiento de los coeficientes cuantizados, en la memoria RAM de almacenamiento temporal.

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

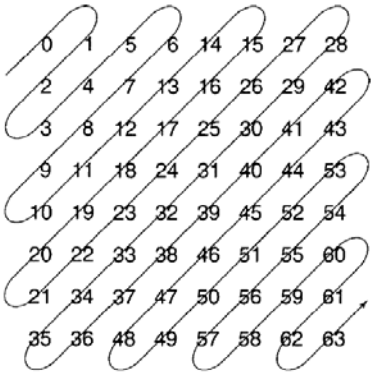


Fig. 9. Orden en zig-zag para los coeficientes DCT-2D cuantizados

En la Fig.10 se muestra un diagrama de bloques simplificado del bloque de exploración en zig-zag implementado. Los datos del primer bloque de 64 coeficientes cuantizados son almacenados, con la disposición en zig-zag, en 66 ciclos de reloj. En la descripción VHDL de este bloque, se ha incluido una bandera que indica el momento en que los 64 coeficientes cuantizados se encuentran almacenados con la disposición en zig-zag. Esta bandera también sirve para desactivar la señal de *reset* del bloque de codificación de Huffman.

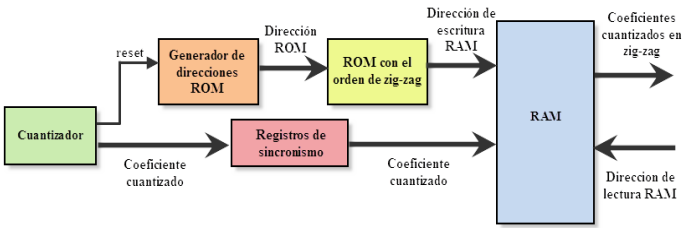


Fig. 10. Diagrama de bloques simplificado del bloque de exploración en zig-zag

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 3. 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. 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 3
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

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 [9].

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.

La codificación de los coeficientes DC es algo parecido a la codificación en AC. De igual 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' 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 4 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. Al igual que en el caso AC, existen dos tablas de códigos, una para cada componente de color, que indican la palabra código del coeficiente y su longitud en bits.

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.

TABLA 4
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

La palabra B' se obtiene de igual forma que para el caso de 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.

Para implementar este esquema de codificación, se almacena en dos memorias ROM, las palabras código A para la componente de luminancia definidas en el estándar JPEG y la longitud en bits de cada palabra, tanto para los coeficientes AC y DC. La discriminación del coeficiente DC se lleva a cabo mediante una bandera. La bandera se activa cada vez que un contador módulo 64 inicia su cuenta (el contador tiene un valor de 1). Cuando se detecta que la bandera de un coeficiente DC está activa, se realiza la resta del coeficiente DC actual y el coeficiente DC previo (almacenado en un registro especial). Esta diferencia pasa a una serie de comparadores para definir su categoría según la Tabla 3. Para los coeficientes AC, existen dos comparadores; en el primer comparador se verifica si el coeficiente es nulo, en caso de serlo, se incrementa un contador, que define la longitud de series de ceros. Cuando el coeficiente no es nulo, un segundo comparador verifica en qué rango de la Tabla 4 está el valor no nulo, para definir la categoría del coeficiente AC. Cabe mencionar que el contador de coeficientes nulos, se habilita sólo cuando la bandera que indica un coeficiente DC no está activada, de esta manera siempre se tendrá para los coeficientes DC, una longitud de series de ceros igual a cero.

Los datos de la bandera de coeficiente DC, la categoría y la longitud de series de ceros, ingresan a una máquina de estados, donde se verifica que la longitud de series de ceros no sobrepase los 15 ceros consecutivos. Si la longitud de series de ceros es menor a 16, la máquina de estados produce una dirección de memoria concatenando los datos de entrada como se indica en la Fig.11.

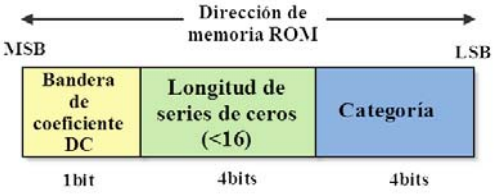


Fig. 11. Esquema de direccionamiento para la memoria ROM con las tablas de Huffman

Si la longitud de series de ceros es igual o mayor a 16, se produce la dirección de memoria correspondiente a la palabra ZRL, al mismo tiempo se resta 16 de la longitud de series de ceros, y se verifica nuevamente si ésta es menor a 16. Este procedimiento se repite hasta obtener una longitud de series de ceros menor a 16, y poder obtener la dirección de memoria como se indica en la Fig.11. Mediante la dirección de memoria establecida, se obtiene de las memorias ROM, la palabra código A y la longitud en bits de la palabra código A para el coeficiente AC no nulo (o el coeficiente DC diferencial). La palabra código B, se encuentra directamente con el valor del coeficientes AC no nulo (o el coeficiente DC diferencial) expresado en complemento a 1 en caso de que éste sea negativo. La longitud en bits de la palabra código B es la categoría del coeficiente no nulo. Una bandera indica cuándo se tienen palabras código válidas.

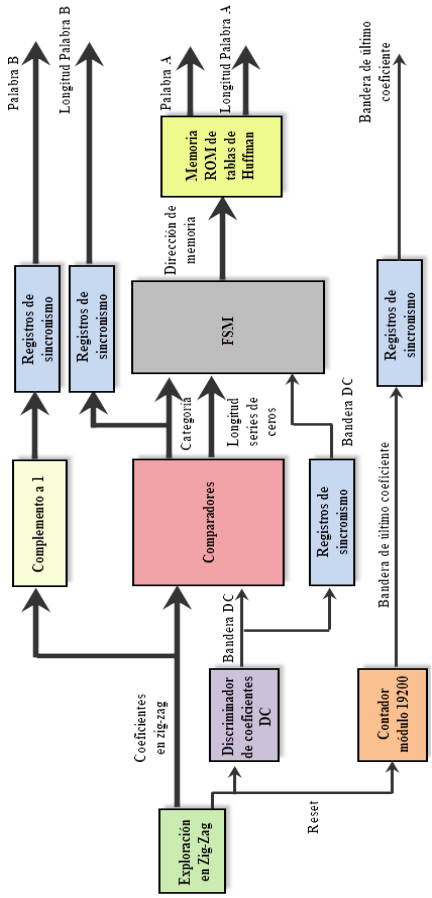


Fig. 12. Diagrama de bloques de la implementación de la codificación de la entropía de Huffman

Con el objeto de identificar el último coeficiente cuantizado de la imagen, cuando la señal de *reset* del bloque de codificación de Huffman se desactiva, se inicia la cuenta en un contador módulo 19200. Cuando el contador alcanza su valor máximo, se activa una bandera para indicar que los datos procesados (categoría y longitud de series de ceros) corresponden al último coeficiente de la imagen. En la Fig.12 se ilustra un diagrama de bloques para la implementación de la codificación de Huffman.

La palabra código A y la palabra código B válidas, ingresan a un bloque de generación de bytes, el cual agrupa los bits de datos de las palabras código y genera los bytes de datos de la imagen comprimida. La generación de bytes depende de la longitud de las palabras código A y B. Los bytes generados son almacenados en un FIFO y después transmitidos hacia una PC con el formato de archivo JFIF. Cuando en el bloque de generación de bytes se detecta que la bandera del último coeficiente de la imagen está activa, otra bandera se activa para indicar que el byte generado corresponde al último byte de datos. Esta bandera indica al módulo de control cuándo puede comenzar a transmitir el *stream* del archivo de imagen generado.

C. Módulo de Transferencia y Almacenamiento de archivos

Este módulo se encarga de transmitir el *stream* de bytes del archivo generado a través de un transmisor UART. Lo primero que se transmite es la cabecera de archivo JFIF, que se encuentra almacenada en una memoria ROM. La cabecera JFIF contiene campos de datos que ayudan en la decodificación de la imagen comprimida. Después de transmitir la cabecera JFIF, se empieza con la transmisión de los bytes de datos de la imagen comprimida, almacenados en el FIFO del módulo de compresión. Al terminar de transmitir los bytes de datos de la imagen comprimida, se transmiten dos bytes que indican el fin del archivo de imagen. El programa de la interfaz gráfica de control, recepta los bytes transmitidos desde el módulo de desarrollo Spartan-3A Starter Kit y los almacena en un fichero con extensión jpg; con esto el sistema operativo del computador puede reconocer al *stream* de bytes recibidos como un archivo de imagen.

D. 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. El módulo de control habilita el almacenamiento de los datos de la imagen sin comprimir. Cuando el módulo de control detecta que se han recibido los 19200 bytes de la imagen sin comprimir, habilita la lectura de la memoria RAM de almacenamiento del módulo de adquisición y desactiva la señal de *reset* del módulo de compresión, los datos de la memoria RAM son transportados ha2cia el módulo de compresión. El módulo de control monitorea la bandera del último byte de datos del módulo de compresión, y cuando ésta está activa, procede a habilitar la transmisión de la cabecera de archivo JFIF y después los bytes de datos de la imagen comprimida almacenados en el FIFO

del módulo de compresión. Después habilita la trasmisión de dos bytes que señalan el fin del *stream* del archivo de imagen.

IV. RESULTADOS OBTENIDOS

Los resultados de la síntesis en términos de recursos utilizados, de las secciones de hardware del sistema de adquisición, compresión y almacenamiento de imágenes, utilizando el FPGA XC3S700A de la Familia Spartan-3A de Xilinx y el lenguaje de descripción de hardware VHDL, se muestran en la Fig.13.

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%

Fig. 13. Recursos lógicos utilizados en la implementación del sistema de adquisición compresión y almacenamiento de imágenes

Con el nivel de utilización del dispositivo mostrado en la Fig.13, la frecuencia máxima a la que puede operar el hardware del sistema es de 82.7 MHz.

Los resultados de la compresión de imágenes se ilustran en la Tabla 5. El escenario de pruebas consistió en la adquisición de los datos de la imagen sin comprimir desde un archivo bmp. Como se esperaba la calidad de imagen mejora conforme se selecciona una factor de calidad de imagen superior.

Mediante la herramienta de software *JPEGsnoop*, se pudo verificar los parámetros de compresión establecidos, y obtener las tasas de compresión de las imágenes comprimidas. En la Tabla 6, se indica la tasa de compresión para cada imagen mostrada en la Tabla 5.

V. 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 una misma descripción, facilitando el proceso de descripción y síntesis de algoritmos de gran nivel de abstracción, como es el esquema de codificación *Baseline* del estándar JPEG, implementado en este proyecto.

TABLA 5. RESULTADOS DE LA COMPRESIÓN DE IMÁGENES CON EL SISTEMA

FACTOR DE CALIDAD	**IMAGEN JPEG**
25	
35	
50	
75	

TABLA 6. TASAS DE COMPRESIÓN OBTENIDAS EN LAS PRUEBAS DEL SISTEMA

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

Es necesario conocer la arquitectura del FPGA utilizado en un diseño, para poder, mediante código de descripción de hardware y desde la descripción del sistema diseñado, deducir en la síntesis, con simplicidad, los bloques funcionales y evitar 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.

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, constituyen un importante apoyo en el diseño de sistemas digitales en FPGAs de Xilinx, ya que implementan funciones específicas, optimizando recursos del FPGA y evitan al diseñador su descripción, enfocándolo sólo a configurar parámetros en los asistentes de los IP Cores, y a desarrollar aspectos más relevantes del diseño, reduciendo el tiempo de diseño y facilitando la verificación en el campo del sistema diseñado.

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 similares a las versiones originales no comprimidas; 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.

REFERENCIAS

[1] R. Woods, J. McAllister, G. Lightbod, Y. Yi. *FPGA-based Implementation of Signal Processing Systems*. 1 Ed. Editorial John Wiley & Sons Ltd, 2008.

[2] C. Maxfield. *The Design Warrior's Guide to FPGA*. 1 Ed. Burlington, Editorial Newnes, 2004.

[3] Xilinx, Inc. *Spartan-3 Generation FPGA User Guide*, Enero 2009, versión 1.5. Disponible en: http://www.xilinx.com/support/documentation/user_guides/ug331.pdf.

[4] S. Olloz, E. Villar, Y. Torroja, L. Teres. *VHDL Lenguaje estándar de diseño electrónico*. Editorial McGraw-Hill, 1998.

[5] V. Pedroni. *Circuit Design with VHDL*. Editorial MIT Press, 2004.

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

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

[8] L. Pillai. *Video Compression using DCT*. Xilinx Application Note, 2002 Disponible en: http://www.cs.york.ac.uk/rts/docs/Xilinx-datasource-2003-q1/appnotes/xapp610.pdf.

[9] W. Pennebaker, J. Mitchell. *JPEG still image data compression standard*. 3 Ed. Editorial Van Nostrand Reinhold, 1993.

Pablo X. Jiménez, nació en Quito-Ecuador el 21 de marzo de 1985. Realizó sus estudios secundarios en la Unidad Educativa La Salle de Conocoto, donde obtuvo el título de Bachiller en Ciencias Experimentales. En el año 2003 Ingresó a la Escuela Politécnica Nacional y en el año 2008 egresó de la carrera de Ingeniería en Electrónica y Telecomunicaciones.

Iván M. Bernal, graduado del Instituto Nacional Mejía. Obtuvo el título de Ingeniero en Electrónica y Telecomunicaciones en la Escuela Politécnica Nacional en 1992. Obtuvo los títulos de M.Sc. (1997) y Ph.D. (2002) en *Computer Engineering* en *Syracuse University*, NY, USA. Ha realizado cursos especializados en varios países europeos, latinoamericanos, Estados Unidos y en Corea del Sur. Actualmente trabaja en la EPN, en el Departamento de Electrónica, Telecomunicaciones y Redes de Información (DETRI).