Departamento de Tecnología Electrónica

PROYECTO FIN DE CARRERA

ACELERACIÓN HARDWARE CON FPGA DE ALGORITMO PARA ESTEGOANÁLISIS

Autor: Eric Gutiérrez Fernández

Tutor: Marta Portela García

Leganés, septiembre de 2012

Título: Aceleración Hardware con FPGA de algoritmo para estegoanálisis
Autor: Eric Gutiérrez Fernández
Director: Marta Portela García

EL TRIBUNAL

Presidente: Celia López Ongil

Vocal: Juan Manuel Estévez Tapiador

Secretario: Enrique San Millán Heredia

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 3 de Octubre de 2012 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

En general, siempre me ha costado agradecer a la gente todo lo que hace por mí, gran defecto, y no estoy acostumbrado a expresar este tipo de emociones. Por ello voy a ser breve y conciso, intentando no olvidarme de nadie.

Dar las gracias a Marta Portela, por ser mi tutora en este Proyecto y enseñarme todo lo que me ha podido enseñar a lo largo de la Carrera, que es muchísimo. Gracias por ayudarme en todo lo que te he pedido y por ser tan atenta.

Dar las gracias a mis padres, por recordarme continuamente las páginas del Proyecto que llevaba escritas…y las que me faltaban por escribir. Gracias por interesaros por todo lo que hago y por apoyarme en aquello que consideráis adecuado.

Dar las gracias a mi hermano Edgar, el físico, por ayudarme a pensar y reflexionar sobre aquellas cosas en las que no hubiera caído en toda mi vida. Sigue así y no cambies nunca.

Dar las gracias a Martuca, por estar conmigo durante tanto tiempo, especialmente el último verano. Sin tí no hubiera conseguido nada.

Finalmente me gustaría destacar la labor importante y agradecer el apoyo de otras personas sin las cuales mi vida sería mucho más aburrida y no habría alcanzado lo que he llegado a alcanzar actualmente: gracias a mis abuelos, a mis tíos y tías, a mis amigos; y en general a toda persona que se sienta aludida si algún día se le ocurre leer estas líneas. Muchas gracias.

Resumen

En este Proyecto se plantea el desarrollo de un sistema de estegoanálisis sobre hardware para imágenes en formato JPEG con el objetivo de ser implementado sobre una FPGA y conseguir tiempos de ejecución menores que el mismo sistema desarrollado sobre software.

Se pretende conseguir la llamada aceleración hardware, aprovechando la capacidad de una FPGA para realizar operaciones simultáneas y ejecutar procesos concurrentes para conseguir reducciones de tiempo muy significativas que, al final, acaban significando un mayor número de imágenes analizadas por unidad de tiempo.

El sistema ha sido desarrollado utilizando el lenguaje de diseño hardware VHDL, quedando conformado como un conjunto de módulos, sincronizados entre sí, que implementan cada una de las etapas necesarias para el análisis de una imagen, según el algoritmo de estegoanálisis usado.

En este Proyecto el sistema no se llega finalmente a implementar sobre una FPGA, sino que su utilización se queda a nivel de simulación. A pesar de ello, sí que se ha llegado a sintetizar y mapear sobre una FPGA para comprobar la posibilidad de usar el sistema en la práctica.

Palabras clave: esteganografía, estegoanálisis, estegoimagen, FPGA, VHDL, JPEG, MATLAB, DCT, IDCT, matriz de cuantificación, porcentaje de imagen incompatible.

Abstract

This Project proposes the development of a JPEG steganalysis system on an FPGA. The objective of this system is getting smaller runtimes than the same system developed on software.

Hardware acceleration is the main goal. The capacity of an FPGA is used for doing simultaneous actions and running concurrent processes in order to achieve very important time reductions. This means a greater number of images analyzed per time unit.

The system has been developed using the VHDL hardware design language, being formed as a set of modules, synchronized with each other. It implements each of the necessary steps for the analysis of an image, according to the used steganalysis algorithm.

In this Project, the system has not been finally prototyped in an FPGA. Simulations have been used for testing it. However, its design has been synthesized and mapped on an FPGA to verify the possibility of using the system in practice.

Keywords: steganography, steganalysis, stegoimage, FPGA, VHDL, JPEG, MATLAB, DCT, IDCT, quantization matrix, incompatible image percentage.

Índice general

Índice de Definiciones

Índice de Ecuaciones

Índice de Ejemplos

Índice de Figuras

Índice de Pruebas

Capítulo 1

Introducción y objetivos

1.1 Introducción

La esteganografía es el arte de ocultar información u objetos sobre otros objetos, llamados portadores, de modo que no se perciba su existencia. En el ámbito que nos interesa en este Proyecto se trata de la ocultación de cualquier tipo de dato sobre imágenes.

Como contraposición a la esteganografía, aparece el estegoanálisis, que consiste en el arte de analizar cualquier objeto y poder detectar cuándo ese objeto oculta algún tipo de información y, en algunos casos, poder obtener esa información.

En la sociedad actual el desarrollo de las redes de comunicaciones, y sobre todo de Internet, ha abierto una nueva vía que permite la transmisión de datos y que puede ser utilizada con fines maliciosos. Además el desarrollo de la tecnología multimedia ha permitido la aparición de nuevos elementos que pueden ser utilizados para la ocultación de información, tales como imágenes o vídeos. La misión de los sistemas actuales de seguridad es el análisis de este tipo de elementos con el objetivo de detectar información que puede ser peligrosa para la seguridad de una sociedad, por ejemplo, información procedente de organizaciones terroristas. La enorme cantidad de información que circula por las redes de comunicaciones hace necesario el desarrollo de sistemas que ejecuten sus métodos de análisis de la forma más rápida posible, con el objetivo de poder controlar una mayor cantidad de información en un mismo intervalo de tiempo. Una de las formas

de hacer que este tipo de sistemas se ejecuten a mayor velocidad se basa en la aceleración hardware, es decir, el desarrollo de los mismos algoritmos de análisis, que antes se ejecutaban sobre software, en hardware. La ventaja que ofrecen este tipo de sistemas se basa en la mayor velocidad presentada por sistemas electrónicos como FPGA's o CPLD's, que aportan la ventaja de poder ejecutarse con varios procesos simultáneos, lo que reduce significativamente los tiempos de ejecución de los diferentes programas si se comparan con los tiempos correspondientes a sus ejecuciones en software sobre microprocesadores.

El desarrollo que se presenta como Proyecto Fin de Carrera consiste en la elaboración de un sistema de estegoanálisis sobre hardware, concretamente sobre una FPGA, utilizando para ello VHDL, un lenguaje de diseño de hardware.

En nuestro caso el desarrollo consiste en un sistema de estegoanálisis que tiene como principal objetivo el análisis de imágenes en formato JPEG y la detección de aquellas imágenes que han sido modificadas de tal forma que puede que transporten algún tipo de información que en un principio no llevaban consigo. La selección de un algoritmo que trabaje con imágenes en formato JPEG se debe a la extensión que actualmente recae sobre este formato, siendo el formato más utilizado en Internet para la presentación de imágenes en las distintas webs y por ser el formato estándar del que se valen las cámaras digitales actuales.

Una vez diseñado un sistema como el que se presenta en el actual Proyecto es posible extender su uso a ámbitos en los que se lleven a cabo operaciones de estegoanálisis, observando una mejora del rendimiento en cuanto a número de imágenes analizadas por unidad de tiempo.

1.2 Objetivos

El objetivo fundamental del Proyecto consiste en conseguir aceleración hardware en un sistema de estegoanálisis en cuanto a tiempo de ejecución, comparándolo con el mismo algoritmo ejecutado sobre una plataforma software. Considerando la finalidad principal, se proponen los siguientes objetivos:

- Conseguir que, en un mismo intervalo de tiempo, el número de imágenes analizadas con el sistema hardware sea mayor que el mismo sistema ejecutado en software.
- Alcanzar un rendimiento en el algoritmo hardware, en cuanto a resultados obtenidos (adecuación en el análisis de las imágenes), que sean equivalentes a los obtenidos con el algoritmo software y que permita, por lo tanto, considerar la posible sustitución de la ejecución software por la implementación hardware en ámbitos en donde se utilicen este tipo de sistemas.

- Desarrollar simplificaciones en ciertas operaciones aritméticas a la hora de implementarlas sobre el lenguaje VHDL que puedan ser generalizadas al diseño de otros sistemas.
- Ampliar los conocimientos en el ámbito de la esteganografía, el estegoanálisis y el desarrollo de algoritmos sobre dispositivos hardware y, en concreto, sobre FPGAs.

1.3 Fases del desarrollo

Durante la elaboración del Proyecto se pueden distinguir las siguientes etapas, que se desarrollaron de forma secuencial:

1) En primer lugar, aparece una fase de documentación que implicó la lectura de una gran cantidad de artículos, sitios web y capítulos de libros dedicados a la esteganografía, el estegoanálisis y, sobre todo, relativos al algoritmo de estegoanálisis que se pretendía implementar sobre hardware. También fue necesaria la consulta de documentación relativa a métodos matemáticos, simplificaciones de operaciones y base de programación en lenguaje de diseño hardware VHDL y lenguaje software MATLAB.
2) A continuación, con los fundamentos básicos asimilados, se programó una versión del algoritmo sobre la plataforma software MATLAB. Con ello se pretendía comprobar el correcto funcionamiento del algoritmo y que su función estegoanalítica se realizaba de forma adecuada. Una vez desarrollado el algoritmo software se comprobó su correcta ejecución con imágenes no modificadas y estegoimágenes, tanto propias como obtenidas vía Internet.
3) Una vez validada la versión software del sistema, se pasó al diseño de la versión hardware, implementada mediante el lenguaje de modelado hardware VHDL. Esta etapa del desarrollo fue la que mayores esfuerzos y tiempo supuso debido a la gran cantidad de consideraciones que es necesario tener en cuenta: simplificaciones a realizar, funcionamiento de cada uno de los bloques y sincronización de todos los módulos para generar, al final, un sistema robusto que funcionara de forma correcta y similar a la versión software. De igual forma que en la etapa 2), una vez implementado el sistema, se comprobó su correcto funcionamiento con distintas imágenes y estegoimágenes.
4) Finalmente, la última y más importante etapa, fue la comparación de tiempos de ejecución entre la versión software y la versión hardware para comprobar la diferencia de dichos tiempos. En este punto es donde se comprueba que el objetivo fundamental del Proyecto, la aceleración hardware del sistema de estegoanálisis, es una realidad y que se podría extender su uso a ámbitos en donde sea conveniente o necesaria dicha aceleración.

1.4 Medios empleados

Para el desarrollo del Proyecto ha sido necesario el uso de dos tipos de software fundamentales: por un lado, una plataforma software para el desarrollo y ejecución del código software: *MATLAB 7.5.0*; y por otro lado, otro software equivalente que permitiera el desarrollo y la simulación del código hardware: *ModelSim SE PLUS 6.5c*. Además, a pesar de que en el Proyecto la implementación hardware se queda en la simulación (el sistema no se ha llegado a implementar sobre una FPGA), se ha comprobado su síntesis, traducción y mapeo sobre una FPGA que tenga los recursos lógicos necesarios para albergar el sistema desarrollado. Para la realización de esta comprobación se ha utilizado el software *Xilinx ISE Design Suite 12.1*, que permite la implementación del sistema sobre FPGAs de la compañía Xilinx.

1.5 Estructura de la memoria

La memoria del Proyecto se divide en siete capítulos:

- Capítulo 1: Introducción y objetivos: el capítulo actual. Se realiza una introducción al Proyecto, se describen los objetivos fundamentales que se pretenden alcanzar, las etapas de desarrollo del sistema a implementar, las herramientas empleadas en el diseño y la estructura del presente documento.
- Capítulo 2: Conceptos básicos: en este capítulo se describe la base teórica necesaria para comprender el funcionamiento del sistema que se ha desarrollado y los objetivos que se pretenden alcanzar con dicho sistema.
- Capítulo 3: Implementación software del algoritmo: cómo se ha desarrollado la versión software del algoritmo, funcionamiento sobre la plataforma MATLAB y aplicación a imágenes reales.
- Capítulo 4: Implementación hardware del algoritmo: diseño del código hardware en VHDL que implementa el algoritmo de estegoanálisis y explicación de las simplificaciones a tener en cuenta.
- Capítulo 5: Resultados: descripción de la herramienta de esteganografía utilizada y resultados obtenidos del algoritmo a partir de las pruebas realizadas con imágenes reales: rendimiento, recursos utilizados y tiempos de ejecución.
- Capítulo 6: Conclusiones y trabajos futuros: en función de los resultados obtenidos se exponen una serie de conclusiones relativas al sistema de estegoanálisis desarrollado y se describen posibles mejoras y líneas de desarrollo a tener en cuenta en futuros trabajos.
- Capítulo 7: Presupuesto: presupuesto del Proyecto.

- Anexo A: Uso de números decimales en VHDL: breve explicación sobre cómo se pueden usar los números decimales en VHDL.
- Anexo B: Códigos MATLAB: códigos ejecutados en MATLAB. El algoritmo principal del sistema de estegoanálisis y un código auxiliar.

Capítulo 2

Conceptos básicos

Para comprender los pilares sobre los que se sustenta el Proyecto desarrollado es necesario, en primer lugar, entender y conocer una serie de conceptos básicos del mundo en el que se mueve el Proyecto desarrollado. Para ello se ha elaborado el presente capítulo en donde se explica lo fundamental del mundo de la esteganografía y del estegoanálisis. Este capítulo se divide en tres secciones. La primera de ellas, el apartado 2.1, habla acerca de la esteganografía, del arte de ocultar información sobre un elemento portador: qué es la esteganografía, cuáles son los elementos típicos de un sistema esteganográfico, qué características definen un sistema esteganográfico y qué tipos de sistemas se pueden encontrar. Se verá como, si se atiende al modo de introducir la información "secreta" sobre el portador de la misma, se pueden distinguir dos tipos fundamentales de sistemas esteganográficos. A continuación, en el apartado 2.2, se describe el otro lado de la moneda en el mundo de la esteganografía: el estegoanálisis, es decir, el mundo que trata de volver ineficaz cualquier intento de ocultar información sobre un elemento portador. Se describirán los tipos de análisis que se pueden aplicar para detectar información sobre una imagen y cuáles son los puntos fuertes y los puntos débiles de cada uno de ellos. Finalmente, en el apartado 2.3, se explica en detalle el sistema de estegoanálisis que se pretente desarrollar en el presente Proyecto: algoritmo a desarrollar, métodos de esteganografía a los que se va aplicar, base matemática sobre la que se sustenta y operaciones aritméticas necesarias para su ejecución.

2.1 La esteganografía

La esteganografía es el arte de escribir mensajes ocultos en un medio cualquiera de tal forma que nadie, excepto el emisor y el receptor del mensaje, sospeche de la existencia de dicho mensaje.

Históricamente, el primer uso de técnicas de esteganografía se registra en el año 440 a.C., cuando Herodoto menciona dos ejemplos de esteganografía en su obra *Historias*. Mucho más adelante, en el siglo XV, Johannes Trithemius, en su trabajo *Polygraphiae*, desarrolló una técnica a través de la cual se podía esconder información en una oración en latín dedicada a Dios.

Un ejemplo de uso reciente de la esteganografía se puede encontrar en el siguiente párrafo [1]:

Ejemplo 1. Uso reciente de la esteganografía

Salutations, Mr.Robertson of CIS 5371. The

Florida Society of Math and Cryptography

is proud to present you with an small exam

for qualification into our society. The key

for passing is studying. Cryptography is

rigorous and only those with patience in

themselves pass. We have an exam PO Box

in Tallahasee. But please submit by 12/12.

En el cual, si se toman las últimas palabras de cada párrafo, se puede leer:

The Cryptography exam key is in PO Box 12/12.

La esteganografía difiere de la criptografía en que ésta última no pretende ocultar el hecho de que se ha escondido información, sino que suele ser evidente que un mensaje transmite información oculta, a pesar de que en principio únicamente el emisor y el receptor del mensaje poseen la clave para descodificar la información. Por el contrario la esteganografía sí que realiza esfuerzos para evitar que se descubra que un portador cualquiera oculta información; de hecho, en el momento en que, por ejemplo, se detecta que una imagen ha sido modificada para esconder información, se considera que el sistema esteganográfico ha fallado; a pesar de que todavía no se sepa cuáles son los datos ocultos.

Actualmente una de las aplicaciones más importantes de la esteganografía se aplica al mundo digital, con el objetivo de la ocultación de datos sobre archivos de texto,

imágenes, audios o contenidos multimedia. Con el desarrollo de Internet aparecen múltiples opciones para enviar mensajes sin que nadie sospeche que esos mensajes existen. Por ejemplo, se han detectado en la red ciertas imágenes que contenían información de organizaciones terroristas.

Los elementos fundamentales en todo proceso esteganográfico son los siguientes:

- Mensaje: datos a ocultar.
- Portador: elemento sobre el cual se oculta el mensaje.
- Estegoobjeto: unión del portador junto al mensaje.
- Estegoclave: clave con la que se codifica el proceso de ocultación y que, posteriormente, será necesaria para recuperar el mensaje oculto.
- Estegofunción: forma en que se va a ocultar el mensaje sobre el portador (por ejemplo: un algoritmo matemático).

En términos de desarrollo la esteganografía se compone de dos algoritmos: uno para introducir el mensaje y otro para extraerlo. El primero de ellos consiste en introducir el mensaje sobre el portador utilizando herramientas de distinta naturaleza (estegofunción) y utilizando una estegoclave para generar el estegoobjeto. Se trata del proceso más delicado de ambos, ya que se pretende que, en caso de intercepción del estegoobjeto, éste no dé muestras de llevar consigo información escondida. Por otro lado el proceso de extracción es tradicionalmente más simple y se trata simplemente del primer proceso pero invertido, para al final acabar obteniendo el mensaje separado de su portador.

El proceso completo se muestra en la Figura 1 [2].

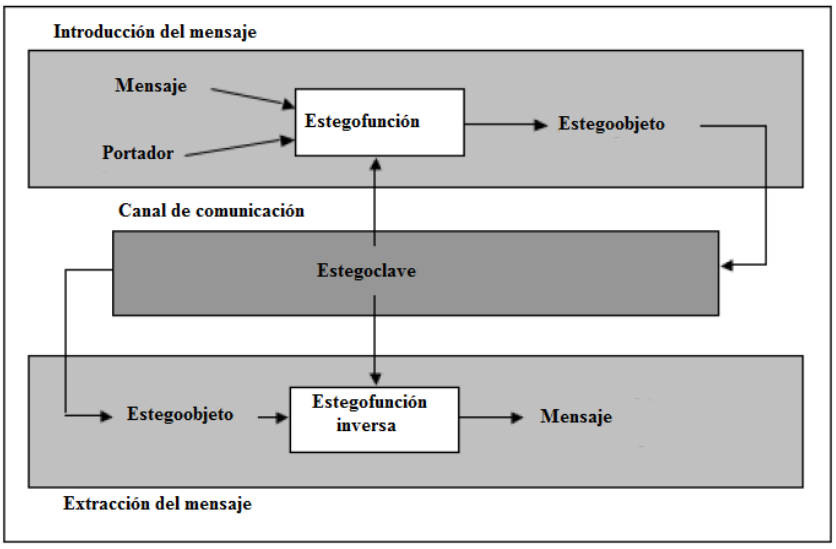


Figura 1. Proceso de esteganografía

Al igual que en sistemas criptográficos, un sistema esteganográfico debería cumplir con los llamados principios de Kerckhoffs, según los cuales todo sistema debe cumplir con los siguientes requisitos [3]:

- Si no es teóricamente irrompible, al menos debe serlo en la práctica. Un sistema esteganográfico siempre debe encontrarse un paso por delante de los métodos de estegoanálisis, de forma que, si bien no es imposible descifrar la información oculta sí que es imposible hacerlo con los sistemas de detección actuales.
- La efectividad del sistema no debe depender de que su diseño (la estegofunción, el modo en que se introduce la información) permanezca en secreto, sino únicamente de su clave. Al final, formas distintas de embeber información son escasas y acaban siendo descubiertas y resueltas, de ahí la importancia de la clave empleada.
- La clave debe ser fácilmente memorizable, con el objetivo de eliminar la necesidad de almacenarla en algún dispositivo escrito o de otro tipo que pueda ser interceptado.
- Los criptogramas deberán dar resultados alfanuméricos, que puedan ser comprendidos fácilmente por el ser humano.
- El sistema debe ser operable por una única persona. Sistemas que necesiten de más de un operario para funcionar acaban fracasando debido al mayor número de personas inmiscuidas en aspectos teóricamente "secretos".
- El sistema debe ser fácil de utilizar. En la misma línea que el punto anterior, sistemas demasiado complejos fracasan debido a la necesidad de una mayor cantidad de personas, instrucciones del sistema o simplificaciones que se realizan para hacer más sencillo su funcionamiento y que, al final, acaban siendo evidentes para la mayor parte del mundo.

Generalmente, los sistemas de ocultación de información se suelen clasificar atendiendo a tres características:

- Capacidad: cantidad de información que puede ser ocultada.
- Seguridad: dificultad para un tercero de detectar información oculta.
- Robustez: cantidad de modificaciones que el medio puede soportar antes de que se pierda la información oculta.

En este ámbito los sistemas esteganográficos optan sobre todo por alcanzar elevada capacidad y elevada seguridad, lo que conlleva que, a menudo, a información sea frágil. Incluso modificaciones triviales del portador pueden destruir la información oculta.

El presente trabajo se centra en la esteganografía aplicada a la ocultación de información sobre portadores en formato imagen. Existen dos formas de ocultar información sobre imágenes que se explican en los siguientes subapartados [2].

2.1.1 Sustitución en los bits menos significativos

Las imágenes se pueden entender como matrices de números en donde cada número especifica el color de un píxel. Es muy habitual encontrarse con imágenes representadas en el entorno RGB, en el cual se tienen tres matrices del tamaño de la imagen cada una de ellas representando cada uno de los entornos siguientes: R (rojo), G (verde) y B (azul). Estas tres matrices actuando de forma simultánea darían la imagen final. Cada una de las matrices está representada por píxeles donde cada uno de ellos posee en valor perteneciente al intervalo [0,255], lo que se conoce como nivel de gris. Para representar los valores de ese intervalo son necesarios 8 bits, de los cuales el último de la secuencia de 8 es el que se conoce como bit menos significativo. Por ejemplo, si un píxel viene dado por el valor 16 se representaría de la siguiente forma en el sistema binario:

Ejemplo 2. Byte de un píxel

0 0 0 1 0 0 0 **0**

Donde el bit menos significativo (LSB, Least Significant Bit) es el 0 (en negrita y subrayado).

Este valor determina si el valor del píxel es un valor par o impar, en el caso en que sea cero se trata de un valor par (en el Ejemplo 2 es 16); por el contrario, al modificarlo, sería un valor impar (en el Ejemplo 2 pasaría a ser 17). Por lo tanto la modificación de este bit, al final, determina una variación de ±1; variación inapreciable por el ojo humano si se extrapola a tonos de colores en las imágenes. Es por ello, que estos bits ofrecen la posibilidad de introducir mensajes a través de su modificación sin que la imagen, a simple vista, sufra modificaciones apreciables.

A continuación se muestra un ejemplo de lo expuesto en el párrafo anterior [1]:

Ejemplo 3. Introducción de información oculta sobre los píxeles de una imagen

Píxeles originales (9 bytes):

11011011 00100100 10100011
00011111 00101101 11101111
00001111 00100111 10000011

Mensaje a insertar (8 bits): ‘A’ (10010111)

Nuevos píxeles (‘A’ se introduce de izquierda a derecha y de arriba hacia abajo):

11011011 00100100 10100010
00011111 00101100 11101111
00001111 00100111 10000011

Además, cuando se habla de métodos LSB, se incluyen dos métodos distintos de introducción de la información: secuencial y aleatoria. La introducción secuencial

(Sequential Hide & Seek) consiste en comenzar con la modificación del primer píxel, a continuación el segundo y así sucesivamente hasta finalizar con la introducción del mensaje (el Ejemplo 3 constituye una introducción secuencial). Sin embargo, la introducción aleatoria (Randomised Hide & Seek) consiste en modificar los píxeles de forma aleatoria con el objetivo de que, posibles sistemas de estegoanálisis (que posteriormente se explicarán), tengan mayor dificultad a la hora de detectar imágenes que ocultan información.

Este tipo de técnicas fueron las primeras que se desarrollaron al creer que los bits menos significativos de una imagen se asignaban "al azar" y que su modificación no afectaría a las propiedades de la imagen y ofrecería la posibilidad de embeber información indetectable. En cierta manera esta creencia es coherente si se observan los planos de bits (BP) que se muestran en la Figura 2. Cada plano muestra la figura considerando únicamente uno de los 8 bits de cada byte, asignando color blanco al valor 0 y color negro al valor 1, desde el bit más significativo en la Figura 2(b) hasta el menos significativo en Figura 2(i) [2].

Se puede observar cómo los planos obtenidos en los bits menos significativos son, aparentemente, aleatorios (por ejemplo en (i)); y que su modificación no generaría variaciones importantes en la imagen. Aún así existen estudios que demuestran cómo estos planos de bits no son aleatorios y que su modificación es detectable, lo que supone por tanto que este tipo de técnicas pueden ser neutralizadas por sistemas de estegoanálisis. Estos métodos de esteganografía también reciben el nombre de técnicas en el dominio del espacio, ya que actúan directamente sobre los valores de los píxeles de las imágenes.

2.1.2 Transformación de dominio

Una vez demostrado que las técnicas esteganográficas basadas en el dominio del espacio eran vulnerables a ciertos métodos de estegoanálisis, se desarrollaron nuevos métodos de introducción de información que trabajaban en otros dominios. Tomando como caso concreto las imágenes JPEG, cuando éstas son comprimidas, en primer lugar se aplica a cada valor de píxel la transformada discreta del coseno (DCT) lo que supone la conversión de la imagen desde el dominio del espacio al dominio de la frecuencia, estableciendo las zonas de la imagen que poseen altas frecuencias (aquellas en donde existen altas variaciones, generalmente zonas de gran detalle) y las zonas de bajas frecuencias (poco detalle). A continuación la compresión que se produce viene a través de la eliminación de aquellas zonas de muy elevada frecuencia, zonas de tanto detalle que el ojo humano no es capaz de percibirlas y que se podrían eliminar sin que ello afectara a la percepción humana de la imagen. Al final, por lo tanto, lo que se produce es una variación en los coeficientes DCT de la imagen. De la misma forma que en el dominio del espacio, es posible controlar las modificaciones que se realizan y conseguir introducir el mensaje secreto entre los coeficientes DCT de la imagen.

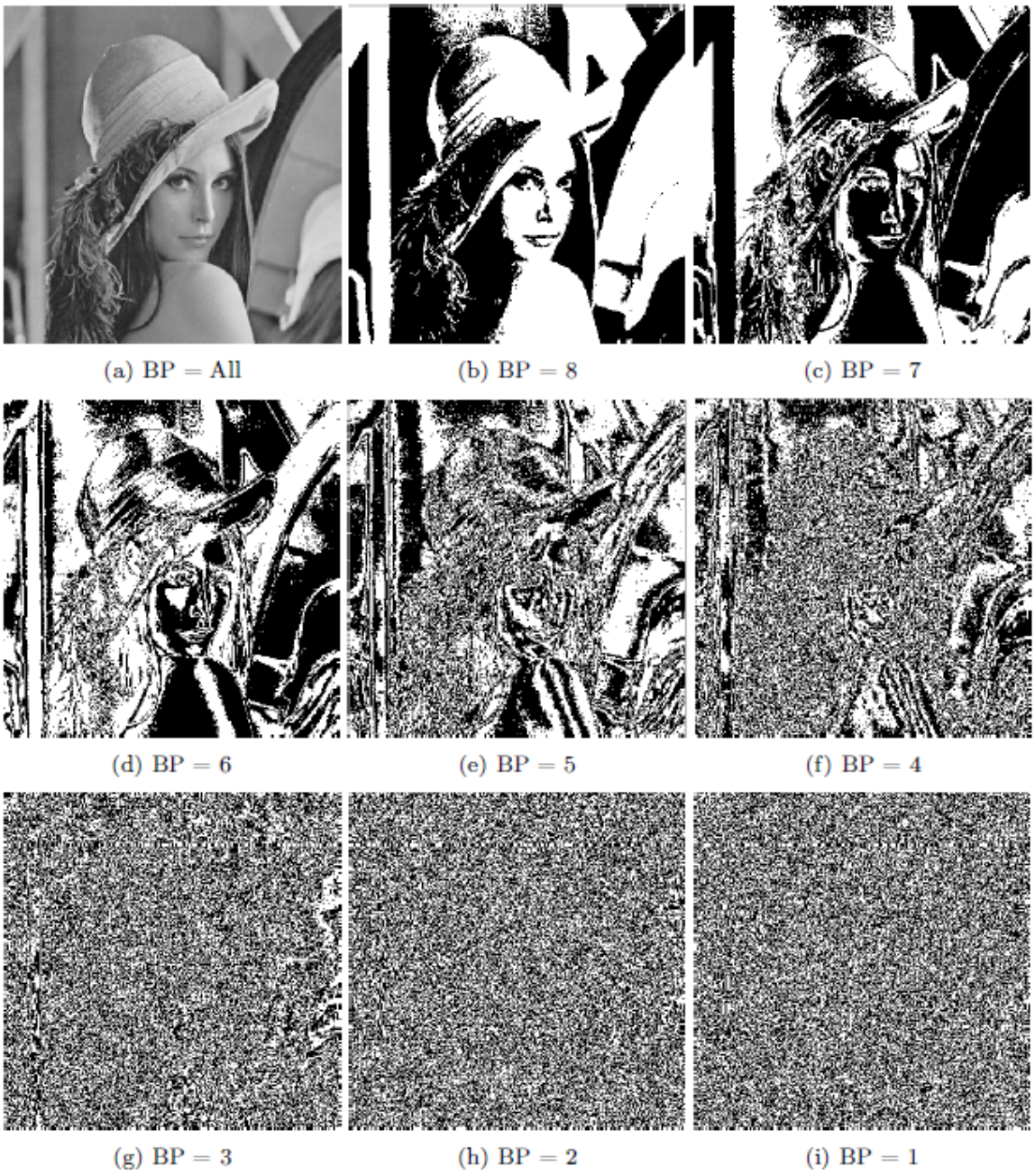


Figura 2. Planos de los 8 bits de cada byte de la imagen

La introducción del mensaje de esta forma es más efectiva a la hora de protegerlo frente a posibles acciones de sistemas de estegoanálisis, ya que éstos tendrán que profundizar más en el estudio de la imagen y no solo centrarse en un estudio en el dominio del espacio.

Hay muchos tipos de transformaciones que pueden ser utilizadas para la introducción de mensajes secretos en imágenes: DCT (*Discrete Cosene Transformation*), DWT (*Discrete Wavelet Transform*) ó FFT (*Fast Fourier Transformation*); siendo la DCT la transformada que se emplea para las imágenes en formato JPEG.

A la hora de aplicar la DCT a una imagen en el dominio del espacio, ésta se divide en bloques de 64 píxeles en forma de matrices de 8x8. A continuación se muestra un ejemplo de matriz original y su transformada:

Ejemplo 4. Cálculo de la DCT de un bloque de imagen

Matriz 1. Bloque 8x8 que representa parte de una imagen

35	34	33	32	35	29	35	34
34	34	34	33	35	28	35	34
34	34	35	33	35	29	35	34
34	35	35	33	34	29	35	34
36	36	34	31	33	29	36	34
35	35	33	29	33	29	35	31
33	33	31	29	33	30	33	27
31	31	30	28	34	30	32	24

Matriz 2. DCT de la Matriz 1

261	5	4	0	0	0	-8	10
8	-2	2	-6	4	0	0	0
-6	0	-4	4	0	0	0	0
2	0	4	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	-1	0	0	0	0	-1
0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0

En cada bloque de 8x8 se pueden encontrar dos tipos de coeficientes: AC y DC. El valor (1,1) de la matriz es el coeficiente DC, mientras que el resto de los valores son los coeficientes AC. El coeficiente DC ofrece una estimación del nivel de detalle en cada bloque, y contiene un valor promedio del resto de los valores AC del bloque. De esta forma, cambiar el valor del coeficiente DC supondrá la modificación de varios de los coeficientes AC del bloque, lo que provocará discrepancias visuales a la hora de volver a visualizar la imagen en el dominio del espacio.

Algunos algoritmos concretos empleados son los siguientes:

I) JSteg: similar al Sequential Hide & Seek visto en el dominio del espacio con la diferencia de que el mensaje a ocultar se introduce, en este caso, sobre los bits menos significativos de los coeficientes DCT cuantificados de la imagen (tras la obtención de los bloques DCT cada uno de los bloques se divide elemento a elemento por un bloque de cuantificación para obtener los coeficientes DCT cuantificados).

II) OutGuess: de la misma forma que JSteg, este algoritmo modifica los bits menos significativos de los coeficientes DCT cuantificados de la imagen. Sin embargo dicha modificación no se realiza de forma secuencial, sino que aquellos datos que van a ser modificados se seleccionan a partir de una serie aleatoria generada mediante un algoritmo a partir de un valor que realiza la

función de "semilla" del algoritmo. El principal inconveniente de este método es la modificación de los parámetros estadísticos de la imagen una vez introducido el mensaje, lo que puede ser utilizado para detectar el mensaje. Por ello, tras introducir los datos en la imagen, se suelen aplicar algoritmos de limpieza cuyo objetivo es eliminar cualquier indicio estadístico de que la imagen haya sido modificada.

Es necesario especificar que tanto JSteg como OutGuess, no modifican ni los coeficientes DC ni los coeficientes AC con valor 0 ó 1, por la relación existente entre los coeficientes DC y AC explicada anteriormente.

III) F5: se diferencia de los anteriores algoritmos en que, en este caso, sí se utilizan los coeficientes AC con valor igual a 1 para embeber el mensaje secreto; continuando sin usar el coeficiente DC y los coeficientes AC con valor 0. El mensaje se introduce de forma secuencial. Además, otro cambio que aparece en este algoritmo, es que los datos no se introducen directamente en los bits menos significativos de los coeficientes DCT, sino que se lleva a cabo una comparación entre el valor absoluto de los coeficientes y el valor absoluto de los bits del mensaje a ocultar. En el caso en que ambos sean el mismo no se realiza ningún cambio, por el contrario en caso en que sean diferentes se reduce el valor del coeficiente DCT en 1. Esta reducción genera a menudo elementos con valor 0, de los cuales no se puede extraer información. Por ello, en estos casos, el dato embebido se vuelve a introducir a través del siguiente coeficiente DCT. Una vez realizada la introducción se aplican tratamientos con el objetivo de "limpiar" la imagen y conseguir eliminar posibles parámetros estadísticos que evidencien la presencia de información oculta. Finalmente, el algoritmo F5 hace uso de una matriz de codificación gracias a la cual consigue reducir la distorsión sobre la imagen a la hora de introducir el mensaje, disminuyendo el número de cambios necesarios sobre la imagen.

Para finalizar, cabe destacar la importancia de la naturaleza de la propia imagen. El éxito al crear una estegoimagen depende tanto del algoritmo seleccionado para almacenar los datos como de la imagen a modificar, es decir, se puede utilizar un algoritmo de introducción muy bueno, muy difícil de detectar; pero si la imagen tiene malas características para transferir información oculta los resultados pueden llegar a ser desastrosos. En general se puede decir que buenas candidatas a la introducción de información son aquellas imágenes con mucho contenido en altas frecuencias, es decir, imágenes con muchos detalles y mucha variación de tonos y colores; las cuales poseerán información suficiente como para "enmascarar" entre tanta información el mensaje que se quiere esconder [1].

2.2 El estegoanálisis

De la misma forma que existen sistemas esteganográficos a través de los cuales se introducen mensajes sobre portadores, existen sistemas de estegoanálisis cuya misión es neutralizar a los anteriores y detectar todos aquellos elementos que lleven consigo información oculta.

La detección de la esteganografía, la estimación del tamaño del mensaje y su extracción son los campos en donde se mueve el estegoanálisis.

Existen tantos métodos de estegoanálisis como maneras de introducir la información mediante esteganografía. A continuación se pasa a enumerar alguno de ellos [2]:

I) Análisis visual: en algunas ocasiones es posible que, bien por utilizar un algoritmo no adecuado, bien por usar una imagen con malas características esteganográficas; se produzcan distorsiones en la imagen que hagan ver a simple vista que esa imagen ha sido modificada. Otras veces no basta con observar simplemente la imagen, sino que las evidencias de información oculta pueden ser observadas, por ejemplo, a través de los planos de los bits menos significativos mediante la aparición de patrones no aleatorios en las mismas. De cualquier manera, aunque estos análisis son sencillos de realizar, su validez tiene un elevado grado de subjetividad.

II) Análisis estructurales: estos métodos se basan en el análisis de la estructura de la imagen centrando la atención en aquellos puntos de los que se sabe que son afectados cuando se utilizan los algoritmos esteganográficos. Por ejemplo, es posible analizar el tamaño de la imagen, ya que generalmente las imágenes grandes poseen más información redundante para la incorporación de mensajes ocultos. En este ámbito existen complejos métodos de estegoanálisis aplicados a cada tipo de formato en concreto, tales como PNG o GIF.

III) Análisis estadísticos: un estegoobjeto puede ser dividido en dos partes: los datos relativos a la imagen original y los datos relativos al mensaje embebido. Se puede afirmar con bastante seguridad que los datos relativos al mensaje están distribuidos de forma más aleatoria que los datos relativos a la imagen, es por ello que análisis estadísticos de la imagen pueden detectar los porcentajes de aleatoriedad introducidos por los mensajes ocultos. Hay varios métodos conocidos que prueban la existencia de datos ocultos en imágenes mediante procesos estadísticos, de entre los cuales se podría destacar el test Chi-Cuadrado, los métodos duales o los análisis del histograma.

Sin embargo, para incrementar la efectividad de estos métodos, al final, es necesario algún tipo de dato acerca de la imagen original: ¿qué tipo de imagen es la portadora?, ¿qué método esteganográfico se ha usado? o ¿qué formato se ha utilizado? Con ello se selecciona el método de estegoanálisis más adecuado a cada caso. Sin embargo, en

muchas ocasiones, no se tiene ningún dato acerca de la imagen que se desea analizar, de ahí que se hayan desarrollado los métodos "blind" (métodos a ciegas). Estos métodos se han desarrollado para alcanzar la máxima eficiencia posible a pesar de que no se tenga ninguna información acerca de la imagen. La clave consiste en buscar un conjunto apropiado de parámetros estadísticos (vector de características) con valores distintivos que hagan sospechar de la manipulación de la imagen. Redes neuronales, algoritmos "clustering" (de racimo) y otras herramientas de computación pueden ser usados para encontrar los umbrales correctos y construir el modelo de detección.

2.3 Estegoanálisis basado en la compatibilidad JPEG

El sistema de estegoanálisis que se estudia e implementa en hardware en este Proyecto Fin de Carrera es el método basado en la compatibilidad JPEG (método de análisis estructural). A grandes rasgos el método consiste en comprobar si una imagen, que previamente ha sido almacenada en formato JPEG, ha perdido los parámetros típicos de dicho formato tras haber introducido una imagen oculta en ella [4] [5].

El método generalmente se aplica a imágenes en escala de grises (8 bits/píxel). Para comprenderlo, en primer lugar es necesario conocer el proceso de compresión que sufren las imágenes en formato JPEG.

Con estas imágenes se parte de su definición en el dominio del espacio. A continuación se divide la imagen en bloques B_{orig} de 8x8 y a cada bloque se aplica la transformada discreta del coseno (DCT) para obtener los coeficientes DCT:

Ecuación 1. Valores DCT sin cuantificar

$$d_k(i), 0 \leq i \leq 64, k = 1, \dots, T$$

Donde T es el número de bloques en que se ha dividido la imagen.

Para ajustar la imagen y resaltar aquellas zonas a las cuales el ojo humano es más sensible, cada uno de los bloques 8x8 transformados se divide entre una matriz de cuantificación Q. Existen matrices estándar Q que se suelen usar, sin embargo no tiene por qué usarse siempre la misma, de ahí que en el sistema a desarrollar sea necesario deducir dicha matriz a través del análisis de la imagen (posteriormente se explicará el proceso).

La matriz estándar Q definida en el formato JPEG es la siguiente:

Matriz 3. Matriz estándar Q definida en el formato JPEG

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

De esta forma se tiene:

Ecuación 2. Valores DCT cuantificados

$$D_k(i) = \text{redondeo\_entero}\left(\frac{d_k(i)}{Q(i)}\right)$$

Estos coeficientes D_k son ordenados en forma de zig-zag [6]:

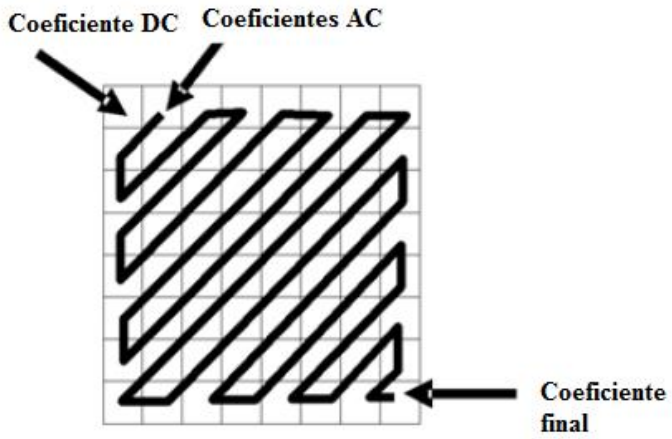


Figura 3. Ordenación zig-zag empleada en la compresión JPEG

Finalmente son comprimidos a través de la codificación Huffman (para más información de dicha codificación consultar [7]) generando el fichero JPEG final comprimido.

La descompresión se basa en el mismo proceso pero realizado de forma inversa. Primeramente se aplica el algoritmo inverso de Huffman y ordenando los valores obtenidos se consiguen los coeficientes D_k, que posteriormente serán multiplicados por la matriz Q para obtener los valores d_k. Para terminar se aplica la transformada inversa del coseno y se redondean los valores al entero más próximo para que se encuentren entre 0 y 255 (en caso de valores por debajo de 0 se les asigna 0 y en caso de valores mayores que 255 se les asigna 255). Con ello, al final, se alcanzan los bloques B 8x8 de los que se partieron:

Ecuación 3. Obtención de B' mediante la IDCT

$$B' = DCT^{-1}(QD)$$

Ecuación 4. Obtención de B a partir de B'

$$B = \text{redondeo\_entero}(B')$$

Es muy importante comprender que la compresión JPEG supone pérdida de información (se trata de un formato "lossy"); por lo que, en general, los bloques B_{orig} de los que se partía en un principio no tienen por qué ser similares a los bloques B obtenidos tras la descompresión.

Teniendo eso en cuenta, aquellos bloques no saturados (es decir, aquellos bloques que no posean píxeles con valor 0 ni 255) cumplen con la Ecuación 5.

Ecuación 5. Expresión usando la norma L cuadrado

$$\|B' - B\|^2 \leq 16$$

La Ecuación 5 se basa en que la diferencia elemento a elemento entre B' y B es como máximo 0.5 debido a que B es el redondeo al entero más próximo de B'.

Ecuación 6. Diferencia entre elementos de B y B'

$$|B'(i) - B(i)| \leq 0.5 \qquad \text{para cualquier píxel i}$$

Llegado a este punto la pregunta clave en la que se va a basar el método de estegoanálisis va a ser la siguiente:

¿Dado B, es posible que este bloque proceda de la descompresión de una imagen previamente almacenada en formato JPEG a través de una matriz de cuantificación Q determinada?

Como respuesta a esta cuestión puede haber dos opciones: por un lado se da el caso en que la imagen no haya sido modificada para albergar información, por lo que su estructura tampoco se vea variada y, como consecuencia, ésta sigue siendo perfectamente compatible con el formato JPEG. Sin embargo, por el contrario, es posible que la imagen lleve consigo datos ocultos, lo que implica de forma prácticamente segura, que la estructura de la imagen haya sido modificada y presente un estado de incompatibilidad con los parámetros típicos del formato JPEG. El objetivo del método es determinar qué imágenes son incompatibles con el formato JPEG, ya que encontrar imágenes previamente almacenadas como JPEG pero que sean incompatibles con los parámetros de dicho formato es altamente sospechoso, y evidencia, con un elevado grado de probabilidad, que dicha imagen ha sido modificada mediante técnicas de esteganografía.

Siguiendo con el desarrollo matemático, se desarrolla la Ecuación 5 aplicando la identidad de Parseval [4]:

Ecuación 7. Desarrollo de la Ecuación 5

$$\|B' - B\|^2 = \|B - B'\|^2 = \|DCT(B) - DCT(B')\|^2 \leq 16$$

En la práctica, se parte de la imagen descomprimida, por lo que el dato de entrada es B. En principio B' es desconocido. Por ello, se utiliza una estimación por defecto de la expresión para alcanzar otra en la que todos los parámetros sean conocidos, a la que llamamos S:

Ecuación 8. Definición del parámetro S

$$S = \sum_{i=1}^{64} \left| DCT(B)(i) - Q(i) \cdot round(\frac{DCT(B)(i)}{Q(i)}) \right| \leq 16$$

El valor S se calcula para todos los bloques 8x8 de la imagen y se analiza su valor:

- Si S>16 el bloque B es incompatible con la compresión JPEG usando la matriz de cuantificación Q en cuestión. Esto es así debido a que, como consecuencia de la información oculta que potencialmente la imagen alberga, la matriz Q empleada ha dejado de ser compatible con los valores DCT de la imagen (incompatibilidad con el formato JPEG), lo que implica que la diferencia expresada en la Ecuación 6 es mayor de 0.5 para algunos elementos, lo que supone que el valor de S puede llegar a ser mayor que 16. Ahora es posible comprender por qué no se pueden emplear bloques de la imagen saturados: la diferencia de valores entre B y B' para estos bloques puede ser mayor que 0.5, lo que podría suponer valores de S mayores que 16 y la detección de bloques incompatibles que, en realidad, no han sido modificados.
- Si S≤16 el bloque B puede ser incompatible o compatible con la compresión JPEG y la matriz de cuantificación Q empleada. Es posible que se dé el caso en que, dentro de un mismo bloque, existan elementos incompatibles en los cuales la diferencia entre B y B' sea mayor que 0.5, pero también bloques compatibles para los cuales dicha diferencia sea 0 o próxima a 0. En estos supuesto cabe la posibilidad de que unos elementos se compensen con otros de forma que al final se tenga un valor S menor que 16 en un bloque incompatible, lo que implica la necesidad de desarrollar otro método de análisis para llegar a una conclusión correcta.

Por lo tanto para S>16 el análisis finaliza, llegando a la conclusión de que existen evidencias de que el bloque en cuestión ha podido ser modificado por procesos esteganográficos. Sin embargo en el caso S≤16 el análisis debe continuar para poder llegar a una respuesta concreta. En este caso, se deben calcular una serie de matrices cuyos elementos son múltiplos de la matriz de cuantificación Q y presentan valores lo más cercanos posibles a la DCT del bloque que se está analizando. Por ejemplo, suponiendo que se analiza el bloque B de la Matriz 4.

Ejemplo 5. Análisis de un bloque de imagen

Matriz 4: Bloque B, 8x8

28	36	34	37	30	31	32	33
31	37	34	36	30	32	34	34
34	38	32	35	30	33	35	35
36	38	31	34	31	34	35	34
35	36	29	34	32	34	33	32
34	34	28	34	32	33	31	30
34	34	26	33	30	31	30	30
34	33	26	32	29	30	30	30

Cuyo valor DCT es (restando a todos los elementos 128), DCT(B):

Matriz 5. DCT (B)

261	5	4	0	0	0	-8	-10
8	-2	-2	-9	-5	0	0	0
-6	0	-2	-4	0	0	0	0
-2	0	-4	0	1	-1	0	0
0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	-1	0	0	0	0	0
0	0	0	0	0	0	0	-1

La matriz de cuantificación Q es:

Matriz 6. Matriz de cuantificación Q de B

3	1	1	3	4	6	8
1	1	1	3	4	9	10
1	1	3	4	6	9	11
1	3	4	5	8	14	13
3	4	6	9	11	18	17
4	6	9	10	13	17	18
8	10	13	14	17	20	19
12	15	15	16	18	16	17

Las matrices que se deben calcular son aquellas cuyos elementos son múltiplos de Q más cercanos a DCT(B). Denotando a estas matrices como P_p, la primera matriz P_1 es aquella cuyos elementos son múltiplos de Q y se encuentran lo más cerca posible de los correspondientes elementos de la matriz DCT(B), es decir, P_1 es:

Matriz 7. Matriz P_1

261	5	4	0	0	0	-8	-10
8	-2	-2	-9	-4	0	0	0
-6	0	-3	-4	0	0	0	0
-2	0	-4	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Donde:

Ecuación 9. Obtención de los elementos de P_1

$$P_1(i) = Q(i) \cdot \text{round}(\frac{DCT(B)(i)}{Q(i)})$$

De forma similar P_2 es la matriz de múltiplos de Q cuyos elementos son los segundos más cercanos a los elementos de DCT(B), P_3 para los terceros más cercanos y así sucesivamente. En el ejemplo propuesto P_2 es:

Matriz 8. Matriz P_2

261	5	4	0	0	0	-8	-10
8	-2	-2	-9	-8	0	0	0
-6	0	0	-4	0	0	0	0
-2	0	-4	0	8	-14	0	0
0	0	0	0	0	0	0	0
0	0	9	0	0	0	0	0
0	0	-13	0	0	0	0	0
0	0	0	0	0	0	0	-16

Así, se obtendrían las matrices de forma sucesiva comenzando por P_1 hasta alcanzar una matriz P_i para la cual ya no se cumpliera la condición presente en la Ecuación 10, matriz que ya no se consideraría para el análisis.

Ecuación 10. Condición para el cálculo de matrices P_p

$$S = \sum_{i=1}^{64} \left| DCT(B)(i) - P_p(i) \right| \leq 16$$

En el ejemplo:

$$S = 7, \text{para } P_1$$

$$S = 60, \text{para } P_2$$

Por lo tanto, supuesto el caso del Ejemplo 5, únicamente es necesario analizar con el tratamiento que se pasa a describir a continuación la matriz P_1.

Para las matrices P_p que cumplen la Ecuación 10 se debe verificar que:

Ecuación 11. Condición de compatibilidad JPEG

$$B = DCT^{-1}(P_p)$$

La compatibilidad JPEG exige que alguna de las matrices P_p cumplan la Ecuación 11, por lo que en este punto existen dos posibilidades:

Si la expresión anterior se cumple para alguna matriz P_p entonces el bloque B en estudio es compatible con el formato JPEG, por lo que no existirían evidencias de esteganografía.

Por el contrario, si para ninguna de las matrices P_p se cumple la expresión anterior, entonces se deduce que el bloque es incompatible con el formato JPEG.

En ambos casos, el estudio del bloque B finaliza.

Terminando con el ejemplo propuesto se parte de la Matriz 7 y se calcula su IDCT.

Matriz 9. IDCT (P_1)

28	36	34	37	30	31	32	33
31	37	34	36	30	32	34	34
34	38	32	35	30	33	35	35
36	38	31	34	31	34	35	34
35	36	29	34	32	34	33	32
34	35	28	34	32	33	31	30
34	34	27	33	30	31	30	29
34	34	26	32	29	30	30	30

Si se comparan la Matriz 9 con la Matriz 4 se puede observar cómo son aproximadamente similares (salvo cuatro elementos que se diferencian en una unidad, diferencia que se desprecia), por lo que, en este caso, se concluye que el bloque es compatible y no ha sido modificado.

Paso a paso el algoritmo se resume a través de las siguientes etapas [5]:

1. Se divide la imagen en bloques de 8x8. Si el tamaño de la imagen no es múltiplo de 8 los píxeles sobrantes no se tienen en cuenta.

2. Analizar los bloques y aquellos con algún píxel saturado (con valor 0 ó 255) se eliminan del análisis. El número de bloques finales a analizar se denota con T.

3. Se obtiene la matriz de cuantificación Q (el proceso se explica más adelante). En caso de que la matriz de cuantificación tenga todos sus elementos con valor 1, la imagen no ha sido previamente almacenada como JPEG, por lo que el análisis no se puede realizar.

4. Para cada bloque se calcula el valor S.

5. Si S>16, el bloque es incompatible con la compresión JPEG. Si S≤16, el bloque puede ser o no ser incompatible con la compresión JPEG, en este caso es necesario calcular las matrices P_p, múltiplos de Q más cercanas al bloque B; aplicar la DCT^{-1} y comprobar si alguna de las matrices resultante es igual B. En caso de que exista tal igualdad el bloque es compatible, mientras que, en caso de no existir la igualdad, el bloque es incompatible.

6. Tras haber analizado todos los bloques T, si éstos se clasifican como compatibles, la conclusión es que el estegoanálisis no ha encontrado ninguna evidencia de presencia de mensajes secretos. Si, por otro lado, se encuentran algunos bloques incompatibles, es posible intentar determinar el tamaño del mensaje escondido e, incluso, intentar obtener la imagen original antes de que el mensaje fuese embebido.

7. Finalmente, ante resultados dudosos o extraños, es recomendable volver a repetir el algoritmo trasladando la división de los bloques: bien en la dirección x, bien en la dirección y.

2.3.1 Cálculo de la matriz de cuantificación Q

Para finalizar es necesario explicar el método a usar a la hora de calcular la matriz de cuantificación de una determinada imagen en formato JPEG. Partiendo de la imagen ya transformada al dominio de la frecuencia a través de la DCT, se identifica a cada coeficiente con la siguiente nomenclatura:

$$d_k(i), \qquad 1 \leq i \leq 64, 1 \leq k \leq T$$

Donde T representa el número total de bloques no saturados con los que se va a trabajar.

Por ejemplo: $d_k(1)$, representaría el coeficiente DC de todos los bloques que se van a estudiar.

Con esto, para los 64 valores de cada bloque se calcula la Ecuación 12.

Ecuación 12. Distribución $E_i(q)$

$$E_i(q) = \frac{1}{T}\sum_{k=1}^{T}\left|d_k(i) - q \cdot \text{round}\left(\frac{d_k(i)}{q}\right)\right|$$

Esta distribución se calcula para valores de q comprendidos entre 1 y 100 aproximadamente y se representa, obteniendo gráficas del estilo a la de la Figura 4 [4].

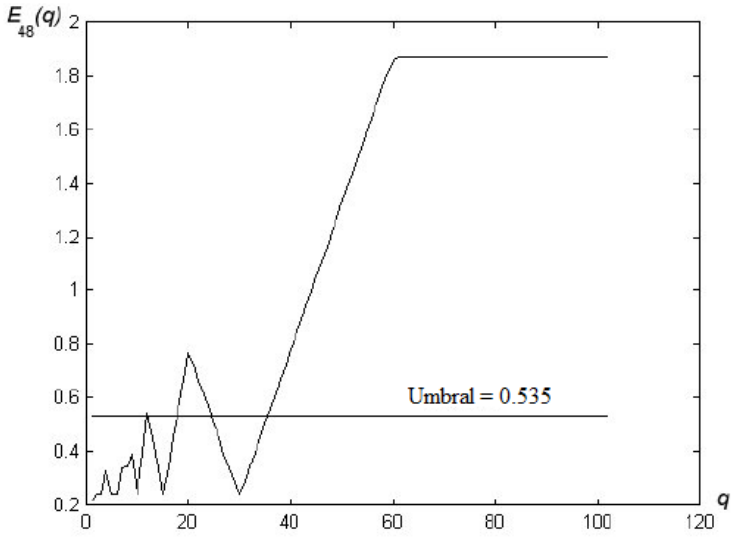


Figura 4. Distribución $E_{48}(q)$

En esta gráfica se observa cómo existen ciertos valores de *q* para los cuales la función presenta mínimos locales. Estos mínimos son los valores para los cuales la imagen presenta la mayor compatibilidad posible con el proceso de compresión al que ha sido sometida la imagen, y por lo tanto son los candidatos a ser el valor de cuantificación

deseado. De entre todos ellos el valor correcto es el mayor; ya que, si se observa, el resto de mínimos no son más que divisores del mayor mínimo.

En el ejemplo propuesto en la Figura 4 se afirmaría que el valor de cuantificación Q(48) es 30, pudiéndose comprobar como el resto de mínimos son divisores de 30: 15, 10, 6, 5, 3 y 2.

Con este método en algunas ocasiones es posible que aparezcan algunos "falsos" mínimos por encima del valor correcto de *q*. Para solucionar este fenómeno se utiliza un umbral de forma que aquellos mínimos locales que aparezcan por encima del valor umbral no se consideran. Teniendo en cuenta los experimentos documentados en [4] [5], este valor umbral se calcula como $\mu_i+3\sigma_i$, donde μ_i y σ_i representan, respectivamente, la media y la desviación típica del vector formado por el valor $E_i(q)$ de los mínimos locales.

Finalmente si la distribución no presenta ningún mínimo, el valor correcto será 1 si $E_i(1)< 0.6*E_i(2)$; en caso contrario el valor correcto es 2.

2.3.2 La Transformada Discreta del Coseno (DCT, Discrete Cosine Transformation)

El algoritmo que se va a implementar en hardware presenta una base matemática bastante sencilla de comprender en donde la única operación destacable que se debe explicar es la transformada discreta del coseno (DCT).

Desde un punto de vista puramente matemático, la DCT expresa una secuencia finita de varios puntos como resultado de la suma de distintas señales sinusoidales. Tiene un signficado físico parecido al de la transformada discreta de Fourier (DFT), pero mientras la DCT trabaja únicamente con cosenos (lo que genera resultados en el dominio de los números reales), la DFT lo hace con exponenciales complejas.

Aplicado a las imágenes, la DCT lleva desde el dominio del espacio al dominio de la frecuencia (similar a lo que hace la DFT).

Un ejemplo de aplicar la DCT sobre una imagen se observa en la Figura 5.

En la Figura 5 se puede observar cómo el paso de una imagen desde el dominio del espacio al dominio de la frecuencia tiene la consecuencia de mostrar los contornos más importantes de la imagen, siendo estos puntos de contorno las zonas de alta frecuencia.

La transformada discreta del coseno (DCT) presenta dos variantes: unidimensional y bidimensional. No obstante, la transformada bidimensional (que es la que en tratamiento de imágenes interesa) acaba derivando de la unidimensional, por lo que la explicación se va a centrar en la transformada unidimensional.

Ejemplo 6. Aplicación de la DCT sobre una imagen

Figura 5. Imagen original (izquierda) y su DCT (derecha)

La transformada unidimensional se define de la siguiente manera [7]:

Definición 1. Definición de la DCT unidimensional

$$\mathrm{DCT}\big(\mathrm{x(n)}\big) = \mathrm{C(u)} = \alpha(\mathrm{u}) \cdot \sum_{\mathrm{n=0}}^{\mathrm{N-1}} \mathrm{x(n)} \cdot \cos\frac{(2\mathrm{n}+1)\cdot\pi\cdot\mathrm{u}}{2\cdot\mathrm{N}}, \qquad u = 0,1,\dots,N-1$$

donde $\mathrm{x(n)}$ es el vector de entrada y $\mathrm{C(u)}$ es el vector de salida

tanto $\mathrm{x(n)}$ como $\mathrm{C(u)}$ tienen la misma dimensión: N

El parámetro α(u) tiene dos posible valores:

$$\alpha(\mathrm{u}) = \frac{1}{\sqrt{\mathrm{N}}}, \qquad \text{si u} = 0$$

$$\alpha(\mathrm{u}) = \frac{\sqrt{2}}{\sqrt{\mathrm{N}}}, \qquad \text{si u} \neq 0$$

De modo inverso la transformada inversa se define como:

Definición 2. Definición de la IDCT unidimensional

$$\mathrm{DCT}^{-1}(\mathrm{C(u)}) = \mathrm{x(n)} = \sum_{\mathrm{n=0}}^{\mathrm{N-1}} \alpha(\mathrm{u}) \cdot \mathrm{C(u)} \cdot \cos\frac{(2\mathrm{n}+1)\cdot\pi\cdot\mathrm{u}}{2\cdot\mathrm{N}}, \qquad \mathrm{n} = 0,1,\dots,\mathrm{N}-1$$

La nomenclatura típica se vale del término de la Definición 3 y se añade a la Definición 1 y Definición 2:

Definición 3. Definición del parámetro K(u)

$$\mathrm{K(u)} = \frac{\sqrt{\mathrm{N}}}{\sqrt{2}} \cdot \alpha(\mathrm{u})$$

Con ello la definición de la transformada directa e inversa quedan de la siguiente forma:

Definición 4. Definición de la DCT unidimensional con parámetro K(u)

$$\mathrm{DCT}\big(x(n)\big) = C(u) = \frac{\sqrt{2}}{\sqrt{N}} \cdot K(u) \cdot \sum_{n=0}^{N-1} x(n) \cdot \cos\frac{(2n+1)\cdot\pi\cdot u}{2\cdot N}, \qquad u = 0,1,\dots,N-1$$

Definición 5. Definición de la IDCT unidimensional con parámetro K(u)

$$\mathrm{DCT}^{-1}\big(C(u)\big) = x(n) = \frac{\sqrt{2}}{\sqrt{N}} \cdot \sum_{n=0}^{N-1} K(u) \cdot C(u) \cdot \cos\frac{(2n+1)\cdot\pi\cdot u}{2\cdot N},$$
$$n = 0,1,\dots,N-1$$

Tomando K(u) dos posibles valores:

$$K(u) = \frac{1}{\sqrt{2}}, \qquad \text{si } u = 0$$

$$K(u) = 1, \qquad \text{si } u \neq 0$$

En imágenes las operaciones se aplican a bloques de 8x8 que de forma unidimensional constituyen 8 vectores de dimensión 8, por lo que N=8. Particularizando para este caso se tienen las ecuaciones finales siguientes:

Definición 6. Definición de la DCT unidimensional aplicada a imágenes

$$\mathrm{DCT}\big(x(n)\big) = C(u) = \frac{1}{2} \cdot K(u) \cdot \sum_{n=0}^{7} x(n) \cdot \cos\frac{(2n+1)\cdot\pi\cdot u}{16}, \qquad u = 0,1,\dots,7$$

Definición 7. Definición de la IDCT unidimensional aplicada a imágenes

$$\mathrm{DCT}^{-1}(C(u)) = x(n) = \frac{1}{2} \cdot \sum_{n=0}^{7} K(u) \cdot C(u) \cdot \cos\frac{(2n+1)\cdot\pi\cdot u}{16}, \qquad n = 0,1,\dots,7$$

Una propiedad muy importante que presenta la DCT es su condición de separabilidad o la posibilidad de expresar una DCT bidimensional como un conjunto de varias DCT unidimensionales. De forma general la expresión sería la siguiente:

Definición 8. Definición de la DCT bidimensional

$$C(u,v) = \frac{2}{\sqrt{M\cdot N}} \cdot K(u) \cdot K(v) \cdot \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} x(m,n) \cdot \cos\frac{(2m+1)\cdot\pi\cdot u}{2\cdot M} \cdot \cos\frac{(2n+1)\cdot\pi\cdot v}{2\cdot N},$$

$$u = 0,1,\dots,M-1 \quad v = 0,1,\dots,N-1$$

Definición 9. Definición de la IDCT bidimensional

$$x(m,n) = \frac{2}{\sqrt{M\cdot N}} \cdot \sum_{u=0}^{M-1} K(u) \sum_{v=0}^{N-1} K(v) \cdot C(u,v) \cdot \cos\frac{(2m+1)\cdot\pi\cdot u}{2\cdot M} \cdot \cos\frac{(2n+1)\cdot\pi\cdot v}{2\cdot N},$$

$$m = 0,1,\dots,M-1 \quad n = 0,1,\dots,N-1$$

$$K(u) = \frac{1}{\sqrt{2}}, \quad \text{si } u = 0, \quad K(u) = 1, \quad \text{si } u \neq 0$$

$$K(v) = \frac{1}{\sqrt{2}}, \quad \text{si } v = 0, \quad K(v) = 1, \quad \text{si } v \neq 0$$

Así una DCT que se emplee para bloques de imágenes expresadas como matrices de 8x8 se ajustaría a las siguientes definiciones:

Definición 10. Definición de la DCT bidimensional aplicada a imágenes

$$C(u,v) = \frac{1}{4} \cdot K(u) \cdot K(v) \cdot \sum_{m=0}^{7} \sum_{n=0}^{7} x(m,n) \cdot \cos \frac{(2m+1) \cdot \pi \cdot u}{16} \cdot \cos \frac{(2n+1) \cdot \pi \cdot v}{16},$$
$$u = 1, \dots, 7 \quad v = 1, \dots, 7$$

Definición 11. Definición de la IDCT bidimensional aplicada a imágenes

$$x(m,n) = \frac{1}{4} \cdot \sum_{u=0}^{7} \sum_{v=0}^{7} K(u) \cdot K(v) \cdot C(u,v) \cdot \cos \frac{(2m+1) \cdot \pi \cdot u}{16} \cdot \cos \frac{(2n+1) \cdot \pi \cdot v}{16},$$
$$m = 1, \dots, 7 \quad n = 1, \dots, 7$$

$$K(u) = \frac{1}{\sqrt{2}}, \quad \text{si } u = 0, \quad K(u) = 1, \quad \text{si } u \neq 0$$

$$K(v) = \frac{1}{\sqrt{2}}, \quad \text{si } v = 0, \quad K(v) = 1, \quad \text{si } v \neq 0$$

Al final esta condición acaba resultando en una aplicación práctica mucho más sencilla de aplicar que las ecuaciones anteriores, y es que una DCT bidimensional de una matriz se puede dividir en dos grandes etapas de cálculo: DCT unidimensional sobre las filas y DCT unidimensional sobre las columnas. De tal forma que la DCT bidimensional de los bloques 8x8 se va a calcular obteniendo en primer lugar la DCT unidimensional de las 8 filas del bloque; y en segundo lugar, con la matriz resultante de esta primera etapa, aplicando de nuevo la DCT unidimensional, pero en este caso sobre las 8 columnas para obtener la DCT bidimensional final.

A modo de ejemplo se supone la Matriz 10 de la cual se quiere calcular su DCT bidimensional.

Ejemplo 7. Propiedad de separabilidad de una DCT bidimensional

Matriz 10. Matriz de inicio A

```
A =

   -88   -86   -78   -59   -47   -26   -83   -58
   -76   -80   -59   -59   -80  -114   -54   -92
   -39   -80   -88   -63   -80  -114    24   -92
   -76  -116   -59   -59   -85   -30   -83  -105
    -5    -4    19   -63   -85   -28   -83  -105
   -76  -116   -59   -59   -85   -30   -83  -105
    81   -63   -59   -59   -55   -30  -116  -105
   -76  -116  -105   -63   -96   -30    26  -105
```

En primer lugar se realiza la DCT unidimensional a todas las filas de la matriz, obteniéndose:

Matriz 11. DCT unidimensional aplicada a las filas de la Matriz 10

```
A =

 -185.6155  -31.5744  -30.9148   16.6544    7.4246  -16.9235   22.3724  -17.8265
 -217.0818   14.3638   -5.9339  -23.6172         0   31.2902  -23.5646   21.3503
 -188.0904   -8.3649   33.4792   14.7061   -5.6569   75.3271  -65.1471   36.5320
 -216.7282   -5.0175  -38.1394   22.2743  -13.0815   32.2190   43.7337  -12.8108
 -125.1579   97.0842    2.6291    4.7077  -57.2756    2.7683   43.3023   -3.4396
 -216.7282   -5.0175  -38.1394   22.2743  -13.0815   32.2190   43.7337  -12.8108
 -143.5427  104.8010   24.3538   87.4893   45.9619   21.1855   58.7953   -6.6740
 -199.7577  -62.4278   -1.5523   53.5203  -40.6586   84.0949  -24.9968   -5.0888
```

A continuación se realiza la misma operación, pero aplicando la DCT sobre las columnas:

Matriz 12. DCT unidimensional aplicada a las columnas de la Matriz 11

```
A =

 -527.7500   36.7155  -19.1689   70.0068  -27.0000   92.6948   34.7292   -0.2716
  -24.6146  -33.3570  -11.0730  -64.6583   10.8451  -30.4902  -41.2137   18.1967
  -11.6322  -60.5908    5.8220   25.0980   29.5285    4.3298  -30.5793   -4.8094
   24.4459   51.6525  -33.0489   -5.6569    8.5558  -72.3034   81.0014  -29.6233
   13.5000  -38.0840  -28.8984   -1.3067  -46.2500  -20.4577   24.9588  -27.4227
    4.7160   10.1466   -3.2664   50.8098   54.9935  -16.5680   43.1057  -16.3640
  -28.7359  -96.8316  -10.0793   -4.1578  -22.7844   31.5974  -43.3220    2.9029
   78.6187   76.8100   55.3162   15.5068   -1.1280   -9.1817  -17.9781   16.0818
```

Finalmente, como se trata de una imagen, se redondean los valores obtenidos al entero más próximo, obteniéndose la DCT bidimensional que se buscaba:

Matriz 13. DCT bidimensional de la Matriz 10

```
A =

  -528    37   -19    70   -27    93    35     0
   -25   -33   -11   -65    11   -30   -41    18
   -12   -61     6    25    30     4   -31    -5
    24    52   -33    -6     9   -72    81   -30
    13   -38   -29    -1   -46   -20    25   -27
     5    10    -3    51    55   -17    43   -16
   -29   -97   -10    -4   -23    32   -43     3
    79    77    55    16    -1    -9   -18    16
```

Se puede comprobar cómo el resultado es correcto. MATLAB posee una función interna que calcula DCT bidimensionales: dct2. Aplicando:

$$A = round(dct2(A))$$

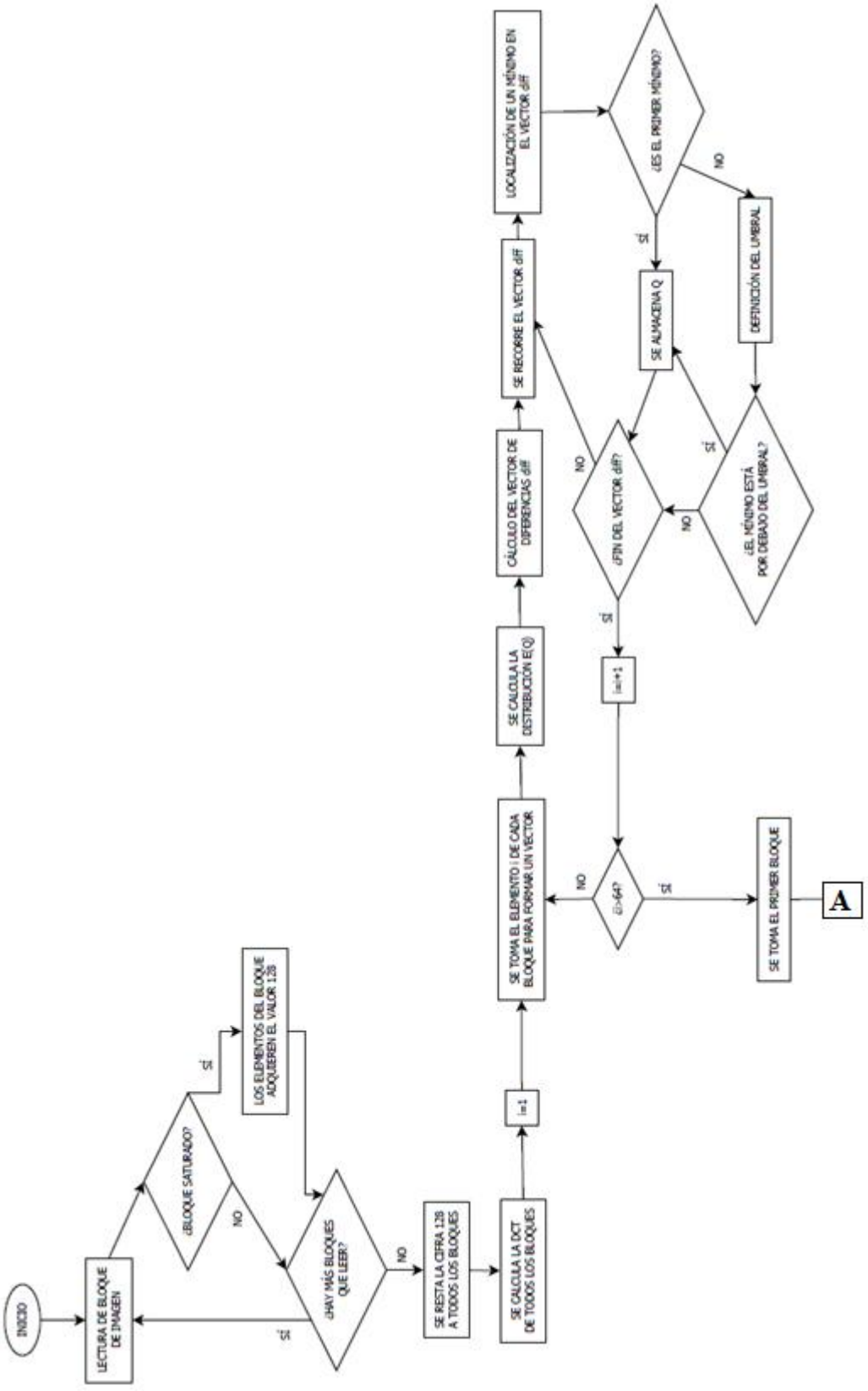
Se obtiene un resultado idéntico al anterior.

Capítulo 3

Implementación software del algoritmo

El sistema de estegoanálisis que se plantea desarrollar sobre hardware es el presentado y explicado en el apartado 2.3 de este documento. Sin embargo, antes de su implementación en una FPGA, se planteó su desarrollo en software para comprobar su correcto funcionamiento. Por ello, se programó una versión del algoritmo en el entorno MATLAB. En este capítulo se explica y se detalla todo lo relacionado con el algoritmo que se desarrolló para realizar las pruebas en MATLAB: diagrama de flujo, funcionamiento y ejemplos de uso. Además se expone un segundo código desarrollado para funciones de visualización y comprobación a la hora de calcular los elementos de la matriz de cuantificación Q de una imagen.

3.1 Diagrama de flujo del algoritmo



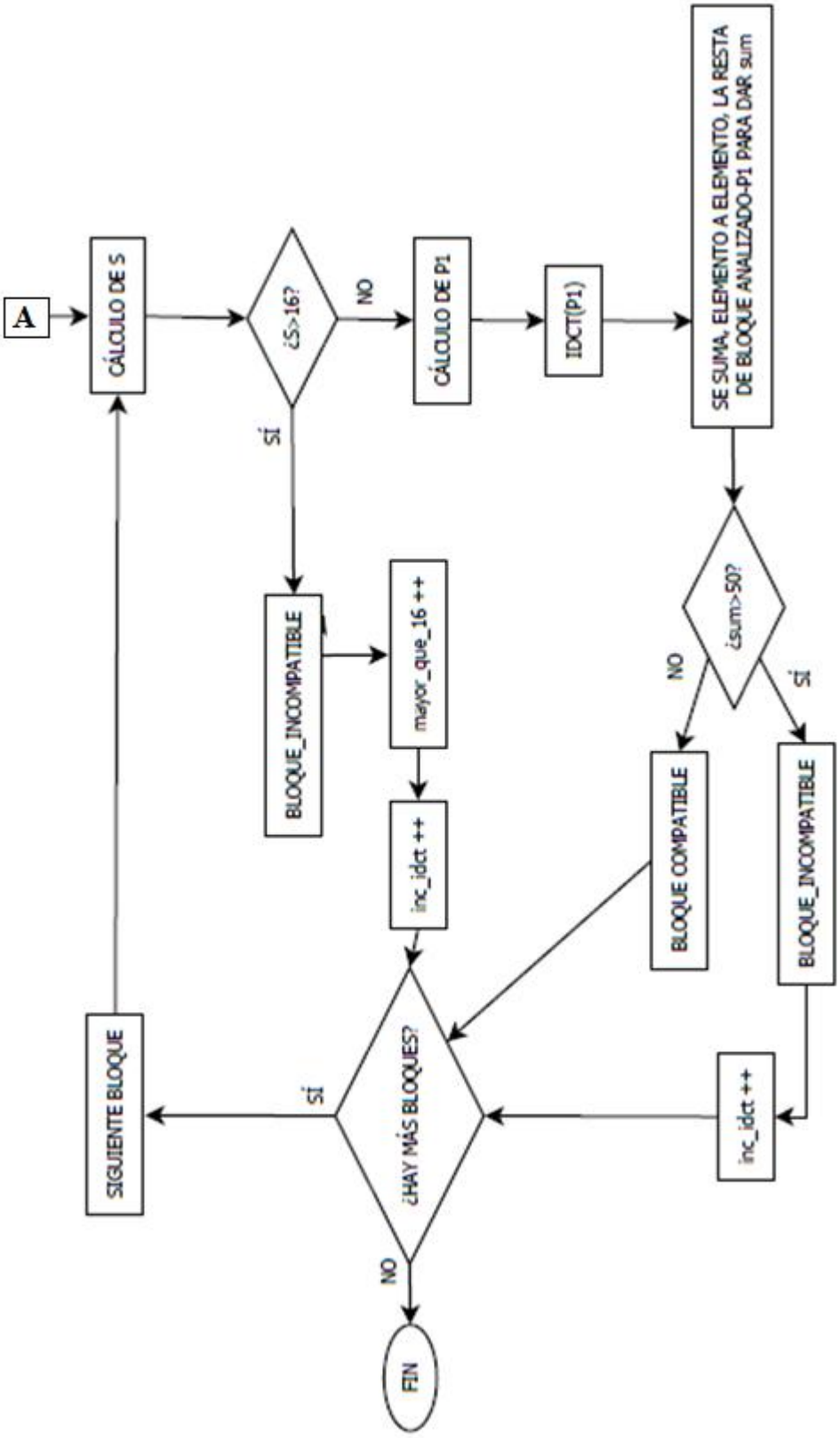


Figura 6. Diagrama de flujo del algoritmo de MATLAB

3.2 Funcionamiento del algoritmo

El algoritmo desarrollado en MATLAB describe exactamente las etapas propuestas en [4] y [5], sin realizar ninguna simplificación como se verá más tarde que es necesario realizar en la implementación hardware.

El código escrito tiene la funcionalidad, por un lado de comprobar el correcto funcionamiento del algoritmo; y por otro lado comprobar que la implementación sobre una FPGA responde de forma adecuada y muestra los resultados que debería dar. Por ello, debido al objetivo de comprobación que desarrolla el código software, se ha ideado un programa con más salidas de las necesarias para facilitar las tareas de comprobación y de depuración, tanto del código software como del código hardware.

La función desarrollada se observa en Definición 12:

Definición 12. Función compatibilidadJPEG

[mayorque16 inc_idct s Q DCT]=compatibilidadJPEG(im)

- Entradas:
  - *im*: representa la matriz de la imagen que se desea analizar expresada en el dominio del espacio y en forma de única matriz en niveles de gris.
- Salidas:
  - *mayorque16*: indica el número de bloques clasificados como incompatibles debido a que su parámetro S es mayor que 16.
  - *inc_idct*: indica el número total de bloques incompatibles detectados, de forma que el número de bloques incompatibles como consecuencia de que la IDCT de P_1 es distinta del bloque original se calcula restando *inc_idct* menos *mayorque16*.
  - *Q*: matriz de cuantificación de la imagen.
  - *DCT*: DCT de la imagen completa.

Para ilustrar su funcionamiento se tiene el Ejemplo 8.

En primer lugar es necesario almacenar la imagen que se desea leer en la variable *im*. Supongamos que la imagen que se desea leer es la que aparece en la Figura 7.

Ejemplo 8. Funcionamiento del algoritmo en software

Figura 7. Imagen a leer, lena.jpg

Se trata de una imagen representada en el dominio RGB, lo que implica que el color de cada uno de sus píxeles viene definido a través de tres componentes numéricas: componente R (*red*), componente G (*green*) y componente B (*blue*) [8]. Para tratar con una única matriz, se convierte la imagen al dominio de los niveles de gris mediante la función de MATLAB: *rgb2gray* [9].

Teniendo en cuenta esta consideración la imagen se lee de la siguiente manera:

Definición 13. Lectura de imagen

im=double(rgb2gray(imread('lena.jpg')));

- *imread*: function de MATLAB para la lectura de imágenes [9].
- *rgb2gray*: función para cambiar del dominio RGB al domino de los niveles de gris.
- *double*: la función *imread* lee la imagen en formato *uint8*, esto quiere decir que sus píxeles únicamente pueden presentar valores enteros entre 0 y 255 lo que a la hora de operar con la imagen resulta un problema debido a que resultados por encima de 255 se truncan a 255, resultados por debajo de 0 se truncan a 0 y resultados decimales se redondean al entero más próximo. Para el algoritmo desarrollado esto no es interesante, por lo que mediante la función *double* se transforma el formato de *uint8* a un formato decimal

que sí acepta números decimales y valores por encima de 255 y por debajo de 0.

Una vez almacenada la imagen en la variable *im* se aplica el algoritmo (Definición 12) y se obtienen los resultados, que se pueden observar en el *Workspace* de MATLAB:

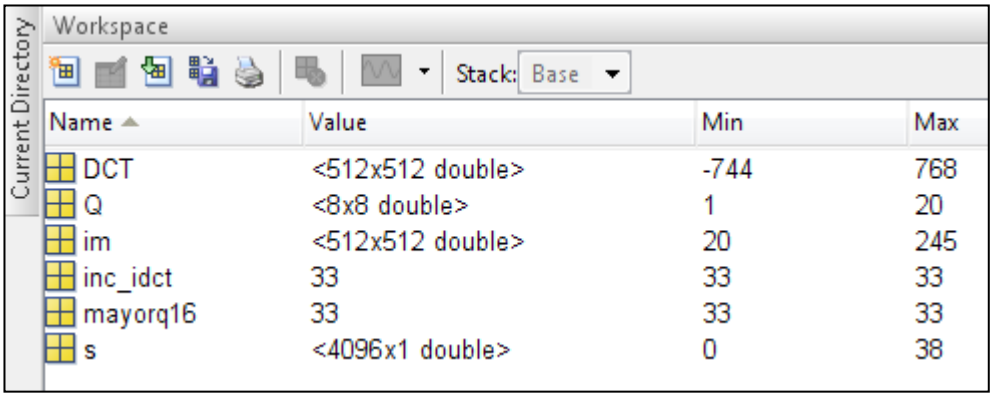


Figura 8. *Workspace* de MATLAB con los resultados obtenidos

La matriz DCT muestra la transformada discreta del coseno de la imagen bloque a bloque, por ejemplo si se desea observar la DCT del primer bloque 8x8:

DCT(1:8,1:8)

Matriz 14. DCT del primer bloque de la imagen

261	5	4	0	0	0	-8	10
8	-2	2	-6	4	0	0	0
-6	0	-4	4	0	0	0	0
2	0	4	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	-1	0	0	0	0	-1
0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0

La matriz Q es la matriz de cuantificación:

Matriz 15. Matriz de cuantificación Q

3	1	1	3	4	6	8	10
1	1	1	3	4	9	10	9
1	1	3	4	6	9	11	9
1	3	4	5	8	14	13	10
3	4	6	9	11	18	17	12
4	6	9	10	13	17	18	15
8	10	13	14	17	20	19	16
12	15	15	16	18	16	17	16

La obtención de la matriz de cuantificación es el punto del algoritmo más complejo y que puede estar sometido a una mayor cantidad de errores. El resto de operaciones acaban siendo operaciones triviales salvo la DCT. No obstante, sobre el cálculo de la DCT existe una gran cantidad de información que puede ser contrastada y verificada hasta alcanzar los resultados correctos. No es así con la matriz de cuantificación, sobre la

cual existe existe muy poca información disponible y en los documentos consultados [4] [5] aparece el método de obtención pero no aparece ningún ejemplo concreto. De ahí que sea posible el planteamiento de la siguiente pregunta:

¿La matriz Q obtenida es correcta?

Para responder a esta cuestión se encontró en [10] una serie de funciones de MATLAB elaboradas para la realización de estegoanálisis, las cuales ofrecían la posibilidad de obtener la matriz de cuantificación. Para la utilización de las funciones fue necesario la compilación de una serie de librerías de C mediante el comando *mex*, que posteriormente se utilizaban desde MATLAB (la información necesaria para el uso de estas funciones se puede encontrar en [10] y [11]).

Con estas funciones se realizaron comprobaciones para verificar que, efectivamente, la matriz de cuantificación obtenida en el algoritmo era correcta.

Siguiendo con los resultados obtenidos aparecen las variables *mayorque16* y *inc_idct* que expresan lo que ya se afirmó en la Definición 13. En este caso se deduce que 33 bloques son incompatibles por poseer su valor S mayor que 16 y ninguno de ellos es incompatible por el método de comparación de la IDCT.

También se ofrece como salida el vector *S*, que muestra el parámetro S de todos los bloques analizados en la imagen y que puede ser utilizado a título informativo.

Además, para terminar, durante la ejecución del programa en MATLAB se muestra por pantalla la variable T, que indica el número de bloques de la imagen que se van a analizar, es decir, los bloques totales de la imagen menos aquellos que están saturados.

En el ejemplo propuesto la imagen tenía unas dimensiones de 512 x 512, lo que supone 4096 bloques totales de 8x8. Se obtiene:

```
T =

        4096
```

Lo que supone que no existe ningún bloque saturado en la imagen.

Si, por ejemplo, se tomara la Figura 9 se obtendría:

```
T =

       12492
```

La Figura 9 tiene unas dimensiones de 780 x 1040, lo implica 12610 bloques totales. Por lo tanto el número de bloque saturados es 12610 – 12492 = 118.

Figura 9. mar.jpg

3.2.1 Código cuantificacionQ.m

Además del algoritmo principal (Definición 12) se ha programado también un código auxiliar que permite visualizar de forma gráfica el proceso realizado para el cálculo de los elementos de la matriz de cuantificación Q.

Esta función presenta la forma de la Definición 14:

Definición 14. Función cuantificacionQ.m

Q=cuantificacionQ(A,num_sat)

- Entradas:
  - *A*: matriz con los valores DCT que se usan en la Ecuación 12. Por ejemplo, para calcular Q_1, A posee los valores DCT(1) de todos los bloques analizados.
  - *num_sat*: número de bloques saturados que presentaba la imagen.
- Salidas:
  - *Q*: elemento Q_i de la matriz de cuantificación.

Este código está pensado para ser usado tras haber ejecutado el programa principal (Definición 12) ya que sus entradas pueden ser obtenidas fácilmente a partir de las salidas de dicho programa.

Se supone que, tras haber ejecutado el programa principal con la Definición 12 y obtener los resultados, se desea conocer cúal ha sido el proceso de cálculo de la matriz de

cuantificación Q (Matriz 15). Por ejemplo, se desea saber por qué el elemento Q(**7**,**7**) es 19.

Ejemplo 9. Uso del programa cuantificacionQ.m

Se sabe que el número de bloques saturados de la imagen es 0, lo que implica que *num_sat* = 0; y la entrada A se obtiene a partir de la salida DCT:

dim=size(DCT); para calcular las dimensiones de DCT

*A=DCT(**7**:8:dim(1),**7**:8:dim(2));* para obtener las entradas DCT deseadas

Q=cuantificacionQ(A,num_sat);

Se obtiene la Figura 10:

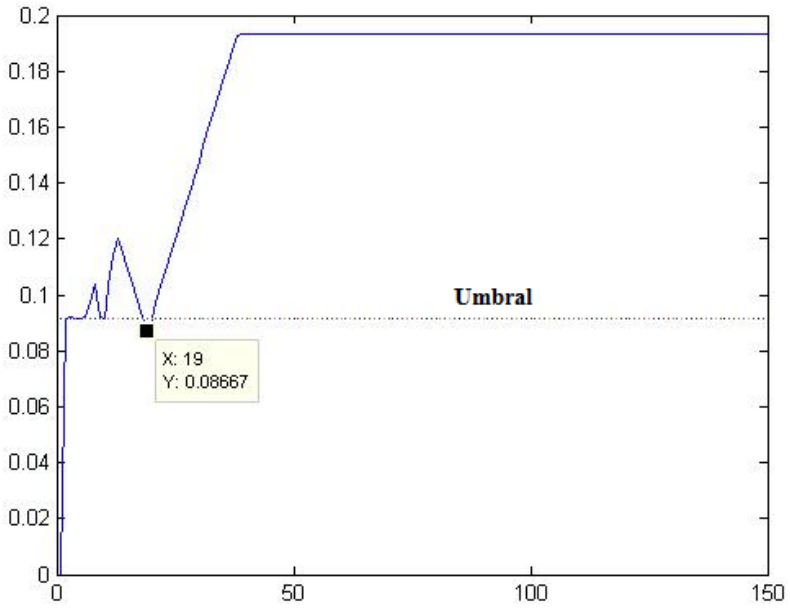


Figura 10. Distribución $E_{(7,7)}$(q) de la Figura 7

Se puede observar que el mayor mínimo se sitúa, efectivamente, en 19; que no existen "falsos mínimos" y que el umbral es mayor que el mínimo tomado.

Capítulo 4

Implementación hardware del algoritmo

Una vez comprobado el funcionamiento del algoritmo sobre MATLAB se pasó a su implementación sobre el lenguaje de diseño hardware VHDL. El algoritmo hardware constituye una versión modificada y simplificada del algoritmo software por razones que posteriormente se verán. Aun así su comportamiento es similar. Este capítulo se divide en dos grandes bloques: por un lado, en el apartado 4.1 se explican las simplificaciones del algoritmo que se han tomado para generar un sistema que pueda ser implementado sobre una FPGA sin demasiada complejidad y sin usar recursos lógicos excesivos. Estas simplificaciones se basan sobre todo en conseguir que, operaciones aritméticas que a priori no son triviales de implementar en VHDL, se puedan obtener mediante operaciones más sencillas, siempre y cuando la precisión no se vea mermada en exceso y, lógicamente, el resultado de la operación sea correcta. Por otro lado, el apartado 4.2, constituye el grueso del Proyecto ya que en él se describe el sistema desarrollado en su forma genérica y especificando y detallando el funcionamiento de cada uno de los bloques que lo conforman. Además, para ilustrar mejor dicho comportamiento, la descripción de alguno de los bloques va acompañada de ejemplos.

4.1 Simplificaciones realizadas

A la hora de traducir el algoritmo software presentado en el Capítulo 3 a un lenguaje de descripción hardware es necesario reflexionar acerca de las posibilidades que ofrece una FPGA. Estos dispositivos pueden realizar cualquier operación, sin embargo unas operaciones suponen una mayor complejidad en el diseño y recursos necesarios que otras, por lo que para alcanzar un diseño final que haga un uso coherente de los recursos es necesario estudiar qué operación puede ser simplificada y ejecutada de forma que no sea difícil de diseñar y que su funcionamiento sea el correcto. Por ello, tras el estudio del algoritmo, se ha concluido que es posible llevar a cabo tres simplificaciones sin que ello suponga que el algoritmo deje de funcionar de forma correcta. De hecho, en el Capítulo 5 se compararán los resultados ofrecidos tanto por el algoritmo software como por la implementación hardware y se observará que son idénticos.

Las simplicaciones realizadas se describen a continuación:

4.1.1 Simplificación en el cálculo de P_p

Situando el punto de estudio en la parte final del algoritmo, al calcular el parámetro S de un bloque, si éste era menor que 16 el bloque podía ser compatible o incompatible. Para salir de dudas era necesario calcular el conjunto de matrices P_p que cumplieran con la condición expresada en la Ecuación 10. La simplificación que se plantea en este caso es el uso, únicamente, de una matriz: P_1, ignorando el resto de posibles matrices.

Únicamente se va a utilizar una matriz P, aquella que se encuentra más cercana (P_1) y que se calcula a través de la Ecuación 9.

Realizando pruebas con diferentes imágenes se ha comprobado cómo, con la mayor parte de los bloques a los que es necesario realizar la prueba de igualdad anterior, únicamente se obtienen bloques incompatibles con la matriz más cercana, consiguiendo un porcentaje prácticamente nulo de bloques incompatibles para otras matrices. Es por ello que los resultados realizando esta simplificación son similares a los que se obtendrían considerando todas las matrices que cumplieran la condición definida en Ecuación 10.

Como argumento a favor de este razonamiento se puede observar el ejemplo propuesto en la Matriz 8, en donde la matriz P_2 ni siquiera cumplía con la condición de la Ecuación 10.

La razón de no calcular el resto de matrices radica en que el número de matrices a analizar varía en función de la imagen con la que se esté trabajando y la velocidad de análisis se vería ralentizada lo que supondría el empeoramiento de uno de los principales objetivos del Proyecto para, finalmente, obtener resultados similares.

4.1.2 Simplificación en el cálculo de la matriz de cuantificación Q

En el momento de calcular la matriz de cuantificación Q, la distribución que se debe estudiar viene definida por la Ecuación 12. En ella, se puede observar como la función está dividida entre el valor T, sin embargo lo interesante es calcular los mínimos locales, por lo que dicha división acaba siendo innecesaria. Los resultados a obtener sin realizar esa división acaban siendo los mismos que si se considera la división, con la ventaja de no tener que implementar en VHDL un algoritmo que realice una división en función de un término de entrada que actúe como coeficiente.

Para ilustrar esta simplificación, se supone la Figura 11.

Ejemplo 10. Simplificación en el cálculo de Q

Figura 11. lena.jpg, en niveles de gris

La matriz de cuantificación obtenida sin realizar la simplificación es:

Matriz 16. Matriz de cuantificación Q de la Figura 11 sin simplificación

Q =

3	1	1	3	4	6	8	10
1	1	1	3	4	9	10	9
1	1	3	4	6	9	11	9
1	3	4	5	8	14	13	10
3	4	6	9	11	18	17	12
4	6	9	10	13	17	18	15
8	10	13	14	17	20	19	16
12	15	15	16	18	16	17	16

Mientras que la matriz obtenida aplicando la simplificación es:

Matriz 17. Matriz de cuantificación Q de la Figura 11 con simplificación

Q =

3	1	1	3	4	6	8	10
1	1	1	3	4	9	10	9
1	1	3	4	6	9	11	9
1	3	4	5	8	14	13	10
3	4	6	9	11	18	17	12
4	6	9	10	13	17	18	15
8	10	13	14	17	20	19	16
12	15	15	16	18	16	17	16

Se observa que son exactamente iguales.

Por ejemplo para el elemento (8,8), cuyo valor es 16, se obtiene, aplicando el método sin simplificar la Figura 12, mientras que la distribución con el método simplificado es la de la Figura 13. En la Figura 13 se puede observar cómo el valor en el eje x es el mismo mientras que en el eje y los valores están relacionados a través de la multiplicación con el valor T (que en el caso del ejemplo es 4096, el número de bloques que se van a analizar):

$$0.08203 \cdot \mathrm{T} = 0.08203 \cdot 4096 = 336$$

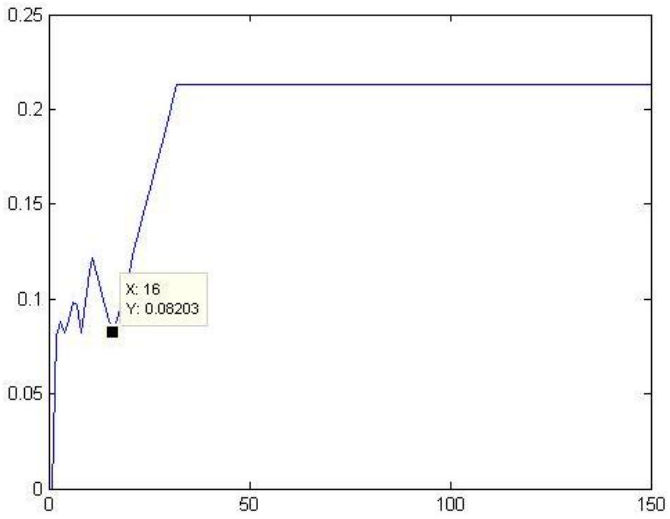


Figura 12. Distribución $E_{64}(q)$ para el método sin simplificar

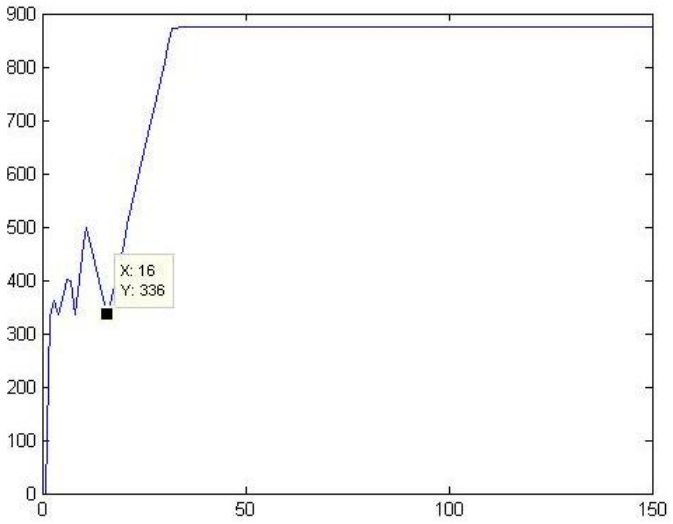


Figura 13. Distribución $E_{64}(q)$ para el método simplificado

4.1.3 Simplificación en la obtención del mínimo de la distribución $E_i(q)$

Finalmente, en el apartado 2.3, cuando se explicaba el método para calcular la matriz de cuantificación Q, se afirmó la posibilidad de que aparecieran "falsos" mínimos por encima de los valores correctos [4]. Para solucionar este fenómeno en [4] se definió un umbral por encima del cual los posibles mínimos que aparecieran no se iban a considerar.

En el método implementado este umbral no se va a considerar. Durante las pruebas realizadas sobre una colección de imágenes, los mayores valores en la matriz de cuantificación encontrados aparecían en torno a 50, no apareciendo ningún "falsos" mínimo. De forma alternativa, para casos en los que sí puedan aparecer "falsos" mínimos, se ha limitado el mayor valor que pueda tener un elemento de la matriz de cuantificación a 75 (valor que es un 50% superior al mayor valor encontrado en Q, con el objetivo de ofrecer al algoritmo cierto margen de cálculo). De esta forma que nunca se van a considerar valores de *q* mayores que 75, limitando la aparición de estos mínimos mayores que el valor correcto.

La razón de no usar este umbral se encuentra en su dificultad de cálculo sobre hardware. Este umbral supone el cálculo de una media y de una desviación típica [4], como se especificó anteriormente. En el caso de la media, su cálculo se podría atajar con pocas dificultades; sin embargo el cálculo de la desviación típica implica una raíz cuadrada de difícil implementación en hardware y gran consumo de recursos. Finalmente, la solución implementada en [4] es empírica, no estando demostrado su perfecto funcionamiento para todas las imágenes.

Por otro lado, el algoritmo expuesto en el apartado 2.3 afirmaba que en caso de no existir ningún mínimo se debía decidir entre 1 ó 2. Se comparaban $E_i(1)$ con $E_i(2)$ y si $E_i(1)<0.6* E_i(2)$ el valor correcto es 1, en caso contrario el valor correcto es 2. En el caso desarrollado siempre se toma 1. La razón de hacerlo así se debe a que la relación

expresada en el algoritmo es empírica y no garantiza un correcto funcionamiento. Se han realizado diversas pruebas con los dos casos: usando la relación y no usándola; obteniendo resultados similares e, incluso, resultados mejores en los casos en los que no se usaba tal relación. De ahí que no se tenga en cuenta y, en caso de que no aparezca ningún mínimo, se tome el valor de *q* como 1.

4.2 Funcionamiento y Esquema General

En este apartado se describen las etapas que presenta el sistema desarrollado a la hora de realizar el análisis de las distintas imágenes. Además también se muestra el esquema general del sistema, presentado las entradas y las salidas correspondientes.

Una nota importante a tener en cuenta en este apartado es la siguiente: durante todos los bloques desarrollados las señales se consideran activas cuando se encuentran a nivel alto, es decir, cuando adoptan el valor '1'; mientras que se consideran desactivadas a nivel bajo, es decir, cuando toman el valor '0'.

El sistema global se basa en una máquina de estados que realiza de forma secuencial los distintos pasos necesarios para el análisis de cada imagen. Esta máquina de estados presenta el diagrama de flujo de la Figura 14.

Se proponen cinco etapas:

1) Lectura de la imagen: se leerá la imagen desde un dispositivo externo y se almacenará de la manera más adecuada para ser procesada.
2) Transformada discreta del coseno: a la imagen previamente almacenada en memoria se le aplica la transformada discreta del coseno (DCT) y se almacena el resultado.
3) Cálculo de la matriz de cuantificación Q: a partir de la DCT, se calcula la matriz de cuantificación Q.
4) Análisis de bloques: en este punto ya se tiene todo lo necesario para aplicar el análisis de los bloques. En primer lugar, para cada bloque, se calcula el valor S. En función del valor de S (si es mayor o menor que 16) el análisis finaliza o es necesario calcular la transformada inversa del coseno (IDCT). Al final se este paso se conocen el número de bloques incompatibles que aparecen en la imagen.
5) Análisis de resultados: una vez el algoritmo ha recorrido todos y cada uno de los bloques, ha mostrado sus resultados; y en función de ellos se deberá determinar si la imagen presenta evidencias de esteganografía o no.

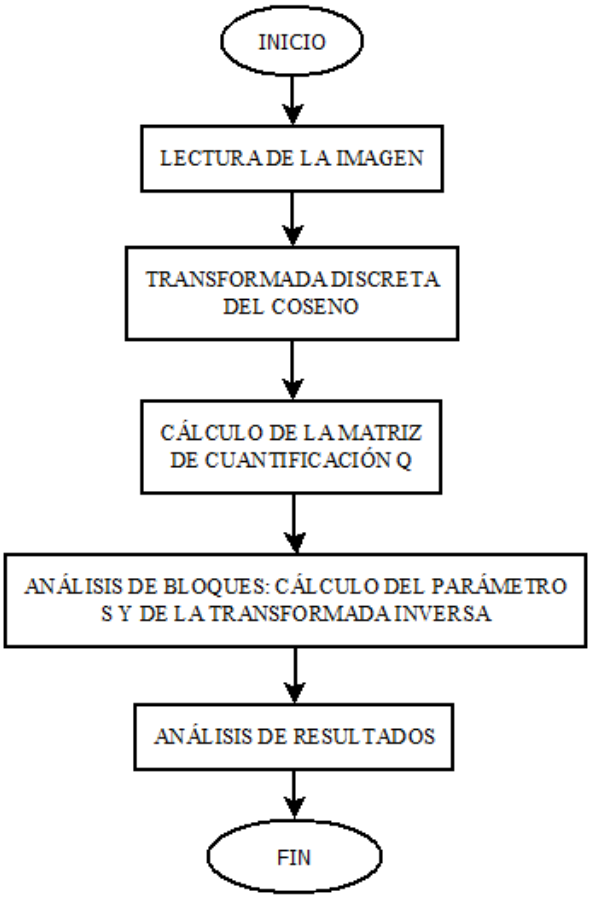


Figura 14. Diagrama de flujo del algoritmo implementado

Las entradas y salidas del bloque general del sistema que se ha diseñado son las siguientes:

- Entradas:
  - *dk0, dk1, dk2, dk3, dk4, dk5, dk6, dk7 (11 bits/señal)*: coeficientes DCT, calculados previamente y almacenados en memoria, que se utilizan en las etapas de cálculo de Q y de análisis de los bloques.
  - *im0, im1, im2, im3, im4, im5, im6, im7 (8 bits/señal)*: píxeles de las imágenes, introducidas de 8 valores en 8 valores.
  - *q_in1, q_in2, q_in3, q_in4, q_in5, q_in6, q_in7, q_in8 (7 bits/señal)*: valores de la matriz de cuantificación Q almacenados en memoria externa.
  - *T (18 bits/señal)*: número de bloques 8x8 que presenta la imagen.
  - *x_comp0, x_comp1, x_comp2, x_comp3, x_comp4, x_comp5, x_comp6, x_comp7 (8 bits/señal)*: valores de la imagen almacenados en memoria y que se utilizan durante la fase de análisis para ser comparados con los resultados del módulo de cálculo IDCT.
  - *clear_global (1 bit)*: reset síncrono.
  - *clk (1 bit)*: señal de reloj.

- *enable_global (1 bit)*: señal de habilitación del sistema.
- *reset (1 bit)*: reset asíncrono.

- Salidas:
  - *min0, min1, min2, min3, min4, min5, min6, min7 (7 bits/señal)*: elementos de la matriz de cuantificación Q hacia memoria externa.
  - *num_dct_compatible (1 bit)*: número de bloques compatibles detectados.
  - *num_dct_incompatible (1 bit)*: número de bloques incompatibles detectados.
  - *x0, x1, x2, x3, x4, x5, x6, x7 (8 bits/señal)*: valores de la imagen menos 128 hacia memoria externa (únicamente aquellos bloques no saturados).
  - *y0, y1, y2, y3, y4, y5, y6, y7 (11 bits/señal)*: coeficientes DCT calculados y que se almacenan en memoria externa para usarlos en posteriores operaciones.
  - *datos (1 bit):* señal que se activa a nivel alto cuando se requieren datos relativos a un nuevo bloque.
  - *dct_realizada (1 bit)*: señal que se activa a nivel alto mientras el sistema muestra por salida los resultados de la DCT.
  - *dct_transmitida (1 bit)*: señal que se activa a nivel alto cuando el sistema muestra por salida el último grupo de valores DCT de cada bloque. Se activa durante un ciclo de reloj.
  - *estado_bloque_saturado (1 bit)*: señal que se activa a nivel alto cuando se ha encontrado un bloque saturado.
  - *fin (1 bit)*: señal que se activa a nivel alto cuando el análisis de la imagen finaliza.
  - *fin_lectura (1 bit)*: señal que se activa a nivel alto cuando la lectura finaliza.
  - *min_calculado (1 bit)*: señal que se activa a nivel alto cuando los elementos de la matriz de cuantificación han sido calculados (se calculan por grupos).

NOTA: estas siete últimas salidas son meramente informativas, pudiendo ser utilizadas para la comunicación con la memoria externa. Aún así su función principal es interna al bloque.

El sistema en detalle se muestra en Figura 15.

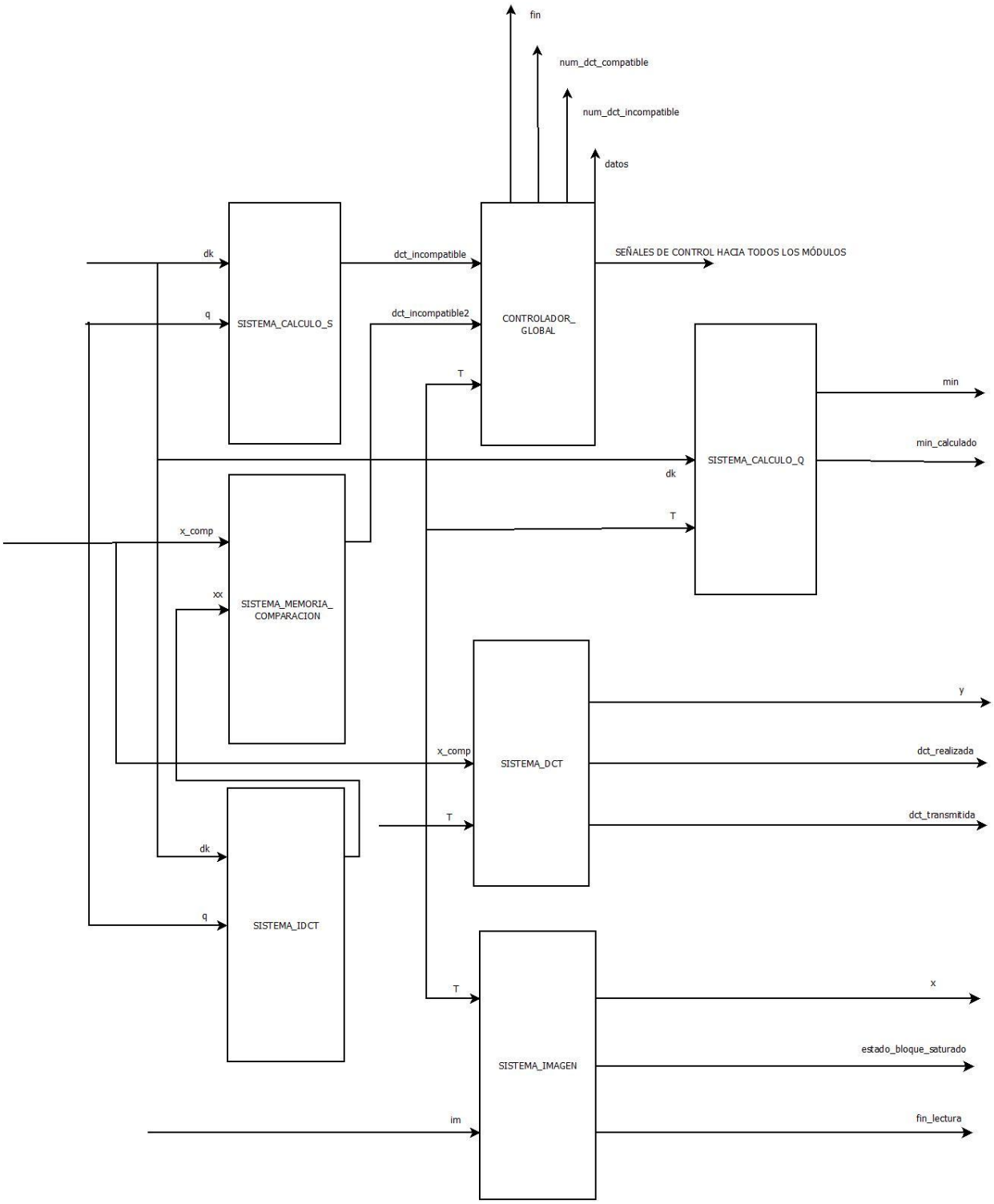


Figura 15. Mapa detallado del sistema global

4.3 Descripción de los bloques

Las principales dificultades que se han encontrado a la hora de programar el algoritmo sobre una FPGA es la complejidad para la realización de ciertas operaciones, tales como multiplicaciones o divisiones genéricas, en función de un término de entrada y sin el uso de excesivos recursos lógicos que supondrían el uso de FPGA's más grandes y caras. A lo largo de la descripción de los diferentes bloques que conforman el sistema se irán describiendo estas dificultades y se explicarán las soluciones adaptadas y como éstas ofrecen un comportamiento paralelo al que se observa en el algoritmo software.

4.3.1 Controlador global (controlador_global.vhd)

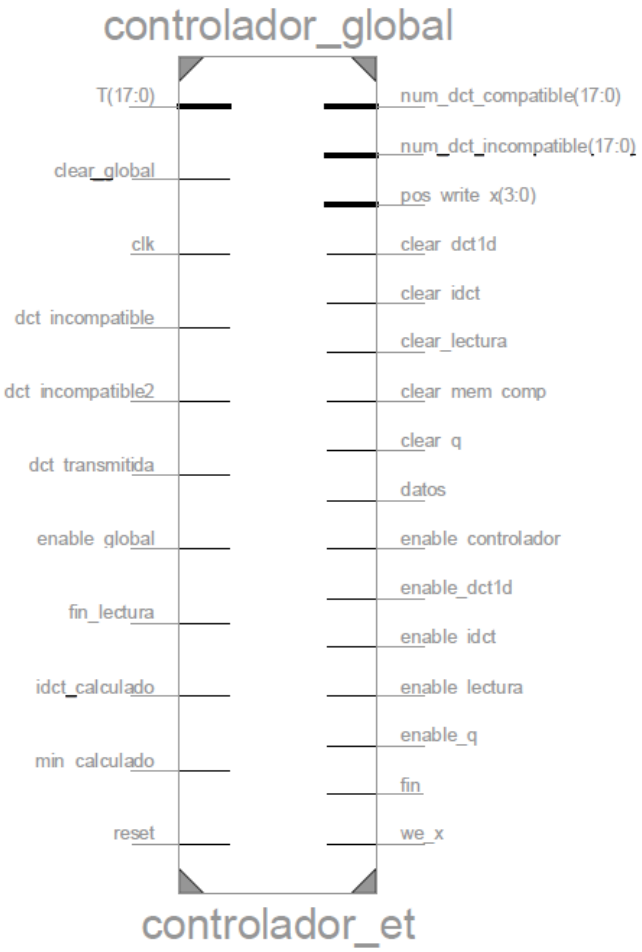


Figura 16. Controlador global

- Entradas:
  - *T*: número de bloques totales que se deben analizar.
  - *clear_global*: reset síncrono.
  - *clk*: señal de reloj.

- *dct_incompatible*: señal que indica que el bloque analizado es incompatible debido a que el valor correspondiente S es mayor que 16.
- *dct_incompatible2*: señal que indica que el bloque analizado es incompatible debido a que el bloque original es muy diferente respecto del bloque obtenido a través de la transformada inversa.
- *dct_transmitida*: señal que avisa que la DCT del bloque ya ha sido calculada.
- *enable_global*: habilita el funcionamiento del bloque.
- *fin_lectura*: señal que se activa cuando la imagen ha sido leída completamente.
- *idct_calculado*: señal que indica que la IDCT del bloque ha sido calculada.
- *min_calculado*: señal que informa del momento en que la matriz de cuantificación Q ha sido calculada.
- *reset*: reset asíncrono.

- Salidas:
  - *num_dct_compatible*: número de bloques compatibles.
  - *num_dct_incompatible*: número de bloques incompatibles.
  - *pos_write_x*: indica la posición en memoria en la que deben ser almacenados cada uno de los valores calculados de la IDCT.
  - *clear_dct1d*: reset síncrono dirigido hacia el bloque que calcula la DCT de cada bloque.
  - *clear_idct*: reset síncrono dirigido hacia el bloque que calcula la IDCT de cada bloque.
  - *clear_lectura*: reset síncrono dirigido hacia el bloque que lee la imagen.
  - *clear_mem_comp*: reset síncrono dirigido hacia el bloque que almacena y compara el bloque original de la imagen con el bloque obtenido a través de la IDCT.
  - *clear_q*: reset síncrono dirigido hacia el bloque que calcula la matriz de cuantificación Q.
  - *datos*: señal que se activa en el traspaso entre un bloque y el siguiente.
  - *enable_controlador*: señal que habilita el bloque que calcula el valor S de un bloque.
  - *enable_dct1d*: señal que habilita el bloque que calcula la DCT de cada bloque.
  - *enable_idct*: señal que habilita el bloque que calcula la IDCT de cada bloque.

- *enable_lectura*: señal que habilita el bloque que lee la imagen.
- *enable_q*: señal que habilita el bloque que calcula la matriz de cuantificación Q.
- *fin*: señal que se activa a nivel alto cuando el análisis de la imagen ha finalizado.
- *we_x*: señal que habilita el almacenamiento en memoria de los valores de la IDCT obtenidos.

Este bloque se encarga de controlar el funcionamiento global del sistema mediante la generación de una máquina de estados que implementa un esquema similar al diagrama presentado en el apartado 4.2. El bloque va recorriendo todos y cada uno de los estados para realizar todas las funciones necesarias para el análisis de la imagen.

La máquina de estados propuesta se muestra en la Figura 17.

Descripción de los estados:

- *espera*: estado de espera del sistema del que sale en el momento en que se habilita *enable_global*.
- *lectura_imagen_fsm*: se realiza la lectura de la imagen.
- *dct*: una vez leída la imagen, se calcula la DCT de la imagen que se vaya a analizar.
- *q*: se calcula la matriz de cuantificación Q.
- *espera_q*: estado de transición entre las diferentes etapas de cálculo de la matriz de cuantificación Q.
- *s_idct*: con el bloque que se analiza, se calcula el valor S. Si éste es mayor que 16, el bloque es incompatible y se pasa al estado *nuevos_datos*. Por el contrario si S es menor que 16, es necesario calcular la IDCT del bloque, una vez calculado el siguiente estado es *comparación*.
- *comparación*: se compara el bloque original de la imagen con el bloque obtenido de la IDCT, en función de su similitud el bloque es compatible o incompatible. A continuación se toman nuevos datos de bloque.
- *nuevos_datos*: si quedan bloques por analizar se toman nuevos valores. Si ya han analizado todos, el siguiente estado es *fin_calculo*.
- *fin_calculo*: el análisis de la imagen finaliza.

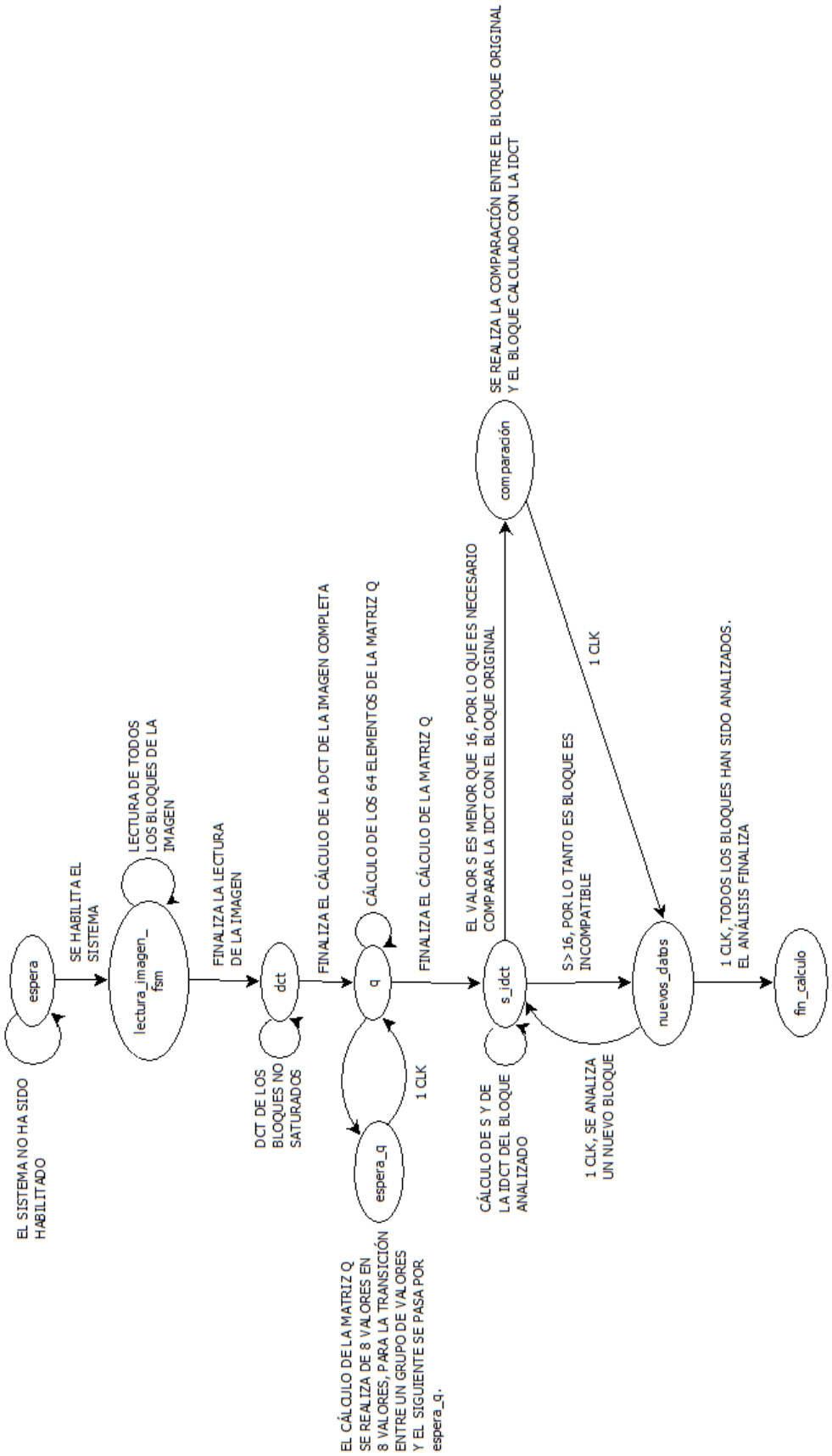


Figura 17. Máquina de estados del sistema global

4.3.2 Sistema de lectura de la imagen (sistema_imagen.vhd)

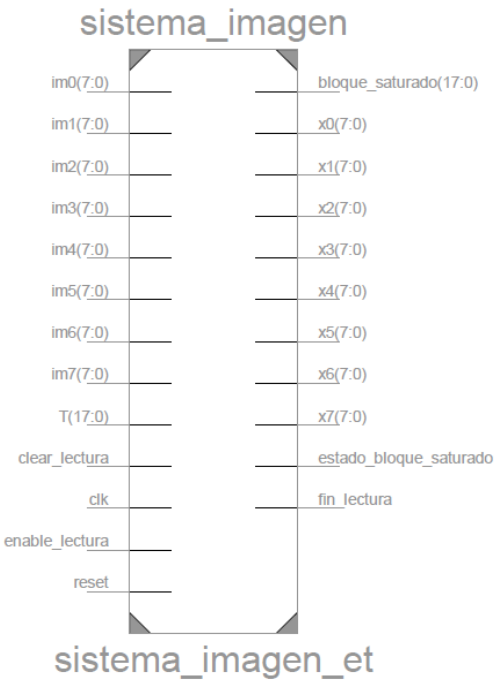


Figura 18. Sistema de lectura de la imagen

- Entradas:
  - *im0*, *im1*, *im2*, *im3*, *im4*, *im5*, *im6*, *im7*: los valores de la imagen se leerán de 8 en 8 a través de estas entradas.
  - *T*: indica el número de bloques 8x8 que presenta la imagen.
  - *clear_lectura*: reset síncrono del bloque.
  - *clk*: señal de reloj.
  - *enable_lectura*: señal de habilitación del bloque.
  - *reset*: reset asíncrono.
- Salidas:
  - *bloque_saturado*: número de bloques saturados encontrados al procesar la imagen.
  - *x0*, *x1*, *x2*, *x3*, *x4*, *x5*, *x6*, *x7*: salidas a través de las cuales se almacenarán los bloques no saturados en memoria externa.
  - *estado_bloque_saturado*: señal que se activa cuando el bloque se encuentra en el estado *bloque_saturado_fsm*.
  - *fin_lectura*: señal que se activa cuando el bloque se encuentra en el estado *fin_lectura_fsm*.

La lectura de la imagen se realiza gracias a una máquina de estados que recorre cada uno de las etapas necesarias para llevar a cabo dicha lectura. Básicamente el objetivo de este bloque es:

1) Realizar la lectura de la imagen analizando sus valores para obviar aquellos bloques que están saturados y a los que no es posible aplicar el algoritmo.
2) Aquellos bloques no saturados almacenarlos en memoria externa. Esto es así debido a que, en general, las imágenes tienen tamaños lo suficientemente grandes como para no poder ser almacenadas dentro de la FPGA, de ahí que sea necesario emplear memorias externas (posteriormente se comentará este aspecto de forma más detallada).

El aspecto que presenta la máquina de estados es el de la Figura 19.

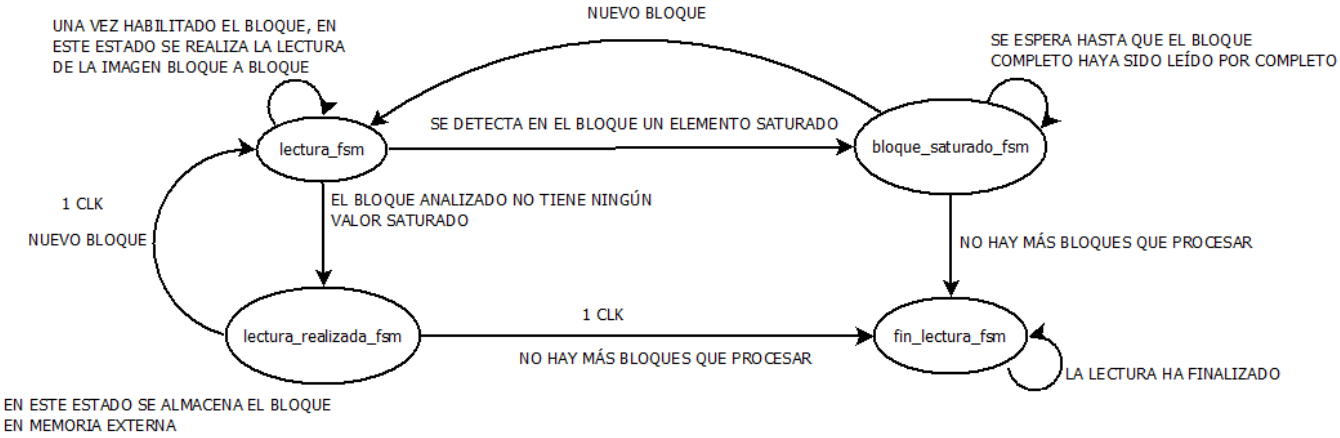


Figura 19. Máquina de estados del sistema_imagen

Descripción de los estados:

- *lectura_fsm*: por defecto el sistema comienza en este punto. Una vez habilitado, comienza con la lectura del primer bloque leyendo los valores de entrada de 8 en 8. Si se encuentra algún valor saturado se pasa al estado *bloque_saturado_fsm* y se incrementa un contador para el número de bloques saturados. Por el contrario, si se lee el bloque completo y no se encuentra ningún valor saturado se pasa al estado *lectura_realizada_fsm*, donde el bloque leído (y almacenado en una memoria temporal) se carga en la memoria externa. Es importante destacar que estos valores almacenados en memoria externa se utilizarán en la siguiente etapa del algoritmo para calcular la DCT, por lo que no se almacenan como tal sino su valor menos 128, según lo establecido para el formato JPEG por el cual los valores previos a la DCT deben encontrarse en el intervalo [-128,127] [7].
- *lectura_realizada_fsm*: en este estado se realiza el volcado del bloque leído no saturado en memoria externa.
- *bloque_saturado_fsm*: si se alcanza este estado es porque alguno de los valores leídos de la imagen se encuentra saturado y, por lo tanto, no sirve para el análisis.

La única diferencia entre este estado y el de *lectura_realizada_fsm* es que, en este caso, no se realiza el volcado del bloque en memoria externa.

- *fin_lectura_fsm*: la lectura de la imagen ha finalizado.

4.3.3 Sistema de cálculo de la transformada discreta del coseno, DCT (sistema_dct.vhd)

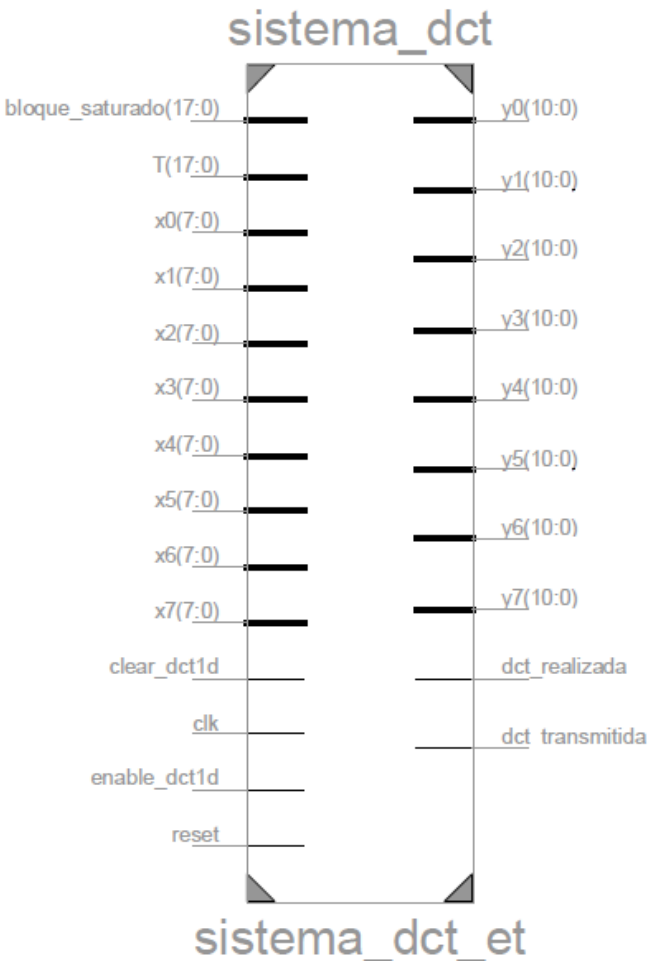


Figura 20. Sistema de cálculo DCT

- Entradas:
  - *bloque_saturado*: número de bloques saturados que se detectaron en la fase de lectura de la imagen.
  - *T*: número de bloques totales (saturados y no saturados).
  - *x0*, *x1*, *x2*, *x3*, *x4*, *x5*, *x6*, *x7*: entradas para el cálculo de la DCT (los valores leídos de la imagen menos 128).
  - *clear_dct1d*: reset síncrono.
  - *clk*: señal de reloj.
  - *enable_dct1d*: señal de habilitación del módulo.

  - *reset*: reset asíncrono.

- Salidas:
  - *y0, y1, y2, y3, y4, y5, y6, y7*: valores DCT calculados.
  - *dct_realizada*: se activa cuando en la salida aparecen los valores finales DCT.
  - *dct_transmitida*: se activa en el último grupo de valores finales DCT (en el octavo grupo).

La implementación del módulo que se encarga de calcular la DCT de la imagen ha sido uno de los pasos más complejos y que presentaba mayores dificultades del proyecto completo. Para su desarrollo ha sido necesaria la consulta de una gran cantidad de documentación: [12], [13], [6] y [14]; para, al final, desarrollar el método más común que se suele repetir en gran parte de la documentación. A la hora de calcular la DCT existen dos caminos posibles: por un lado es posible seguir la nomenclatura matemática y desarrollar el método más estricto posible (este método se puede consultar en el apartado 2.3.2); sin embargo ello implica operaciones aritméticas demasiado complicadas como para implementarlas en VHDL y alcanzar un diseño más o menos sencillo y que emplee la menor lógica posible. De ahí, que se hayan desarrollado métodos aproximados, más sencillos, pero no por ello más imprecisos. En el proyecto presente se utiliza uno de éstos métodos aproximados, método muy contrastado y del cual se ha verificado su correcto funcionamiento.

La DCT posee una propiedad de mucha utilidad según la cual la DCT bidimensional de una matriz se puede dividir en dos DCT unidimensionales: una por filas y otra por columnas (propiedad ya explicada en el apartado 2.3.2). Por ello, el cálculo se reduce a ejecutar la DCT unidimensional en dos ocasiones. Partiendo de esta propiedad ya no es necesario preocuparse por calcular la DCT bidimensional, sino que es necesario buscar la forma de calcular la DCT unidimensional y, a partir de la propiedad anterior, calcular la DCT bidimensional. Esto es lo que se va a realizar en el sistema a desarrollar.

Dentro de la variedad de métodos aproximados de cálculo de la DCT, uno muy aceptado y empleado por su sencillez de operaciones y precisión es el presentado en los documentos [14] y [6], que es el que se va a implementar en el proyecto.

Con este método la DCT unidimensional se calcula a través de una serie de pasos en los que, en cada paso, se van obteniendo los valores que se emplearán en el siguiente paso. Los pasos son los siguientes:

1) Se parte del vector de dimensión 8 denotando sus elementos como a_0, a_1, …, a_7 para obtener los valores b_0, b_1, …, b_7:

$$b_0 = a_0 + a_7;\ \ b_1 = a_1 + a_6;\ \ b_2 = a_3 - a_4;\ \ b_3 = a_1 - a_6;$$
$$b_4 = a_2 + a_5;\ \ b_5 = a_3 + a_4;\ \ b_6 = a_2 - a_5;\ \ b_7 = a_0 - a_7;$$

2) Se obtiene $c_0, c_1, \ldots, c_7$ a partir de $b_0, b_1, \ldots, b_7$:

$$c_0 = b_0 + b_5;\ \ c_1 = b_1 - b_4;\ \ c_2 = b_2 + b_6;\ \ c_3 = b_1 + b_4;$$
$$c_4 = b_0 - b_5;\ \ c_5 = b_3 + b_7;\ \ c_6 = b_3 + b_6;\ \ \ c_7 = b_7;$$

3) Se obtiene $d_0, d_1, \ldots, d_8$ a partir de $c_0, c_1, \ldots, c_7$:

$$d_0 = c_0 + c_3;\ \ d_1 = c_0 - c_3;\ \ d_2 = c_2;\ \ d_3 = c_1 + c_4;\ \ d_8 = c_7;$$
$$d_4 = c_2 - c_5;\ \ \ d_5 = c_4;\ \ \ d_6 = c_5;\ \ \ d_7 = c_6;$$

4) Se obtiene $e_0, e_1, \ldots, e_8$ a partir de $d_0, d_1, \ldots, d_8$:

$$e_0 = d_0;\ \ \ e_1 = d_1;\ \ e_2 = m_3 \times d_2;\ \ e_3 = m_1 \times d_7;\ \ e_8 = d_8;$$
$$e_4 = m_4 \times d_6;\ \ e_5 = d_5;\ \ e_6 = m_1 \times d_3;\ \ e_7 = m_2 \times d_4;$$

$$m_1 = 0.7071; m_2 = 0.3827;\ \ m_3 = 0.5412; m_4 = 1.3066$$

5) Se obtiene $f_0, f_1, \ldots, f_7$ a partir de $e_0, e_1, \ldots, e_8$:

$$f_0 = e_0;\ \ \ f_1 = e_1;\ \ \ f_2 = e_5 + e_6;\ \ f_3 = e_5 - e_6;$$
$$f_4 = e_3 + e_8;\ \ f_5 = e_8 - e_3;\ \ f_6 = e_2 + e_7;\ \ f_7 = e_4 + e_7;$$

6) Se obtiene $s_0, s_1, \ldots, s_7$ a partir de $f_0, f_1, \ldots, f_7$:

$$S_0 = f_0;\ \ S_1 = f_4 + f_7;\ \ S_2 = f_2;\ \ S_3 = f_5 - f_6;$$
$$S_4 = f_1;\ \ S_5 = f_5 + f_6;\ \ S_6 = f_3;\ \ S_7 = f_4 - f_7;$$

En este punto se obtendría, con $s_0, s_1, \ldots, s_7$, los valores de la DCT sin escalar, sin embargo en el formato JPEG se emplean valores escalados, por lo que estos valores $s_0, s_1, \ldots, s_7$ es necesario multiplicarlos por cifras escaladas:

7) Se obtiene $t_0, t_1, \ldots, t_7$ a partir de $s_0, s_1, \ldots, s_7$:

$$t_0 = m_6\ x\ S_0;\ \ \ t_1 = m_7\ x\ S_1;\ \ t_2 = m_8\ x\ S_2;\ \ t_3 = m_9\ x\ S_3$$

$$t_4 = m_6\ x\ S_4;\ \ \ t_5 = m_{11}\ x\ S_5;\ \ t_6 = m_{12}\ x\ S_6;\ \ t_7 = m_{13}\ x\ S_7$$

$$m_6 = 0.3535; m_7 = 0.2549; m_8 = 0.2706; m_9 = 0.3;$$

$$m_{11} = 0.45; m_{12} = 0.6533; m_{13} = 1.2815;$$

8) Finalmente los valores obtenidos se dejarían con representación decimal en la primera DCT calculada (la de filas o columnas) y se redondearían al entero más próximo en la segunda DCT.

Al final se consigue alcanzar los valores DCT con 29 operaciones de sumas/restas y 13 operaciones de multiplicación, un balance mucho menor que el que se obtendría si se aplicaran las operaciones matemáticas estrictas.

Una vez comprendido el algoritmo que se debe desarrollar para calcular la DCT unidimensional, es necesario implementar el sistema que generará la DCT bidimensional final. Este sistema debiera presentar tres módulos básicos: un módulo que implementara la DCT unidimensional, otro módulo que actuara como memoria intermedia para almacenar los resultados de la DCT realizada por filas o columnas, y que éstos puedan ser empleados para la siguiente etapa de cálculo; y otro módulo que actuara de controlador. Para ello se pensó en dos formas distintas de organizar estos módulos:

1) La primera arquitectura presentaría la forma de la Figura 21.

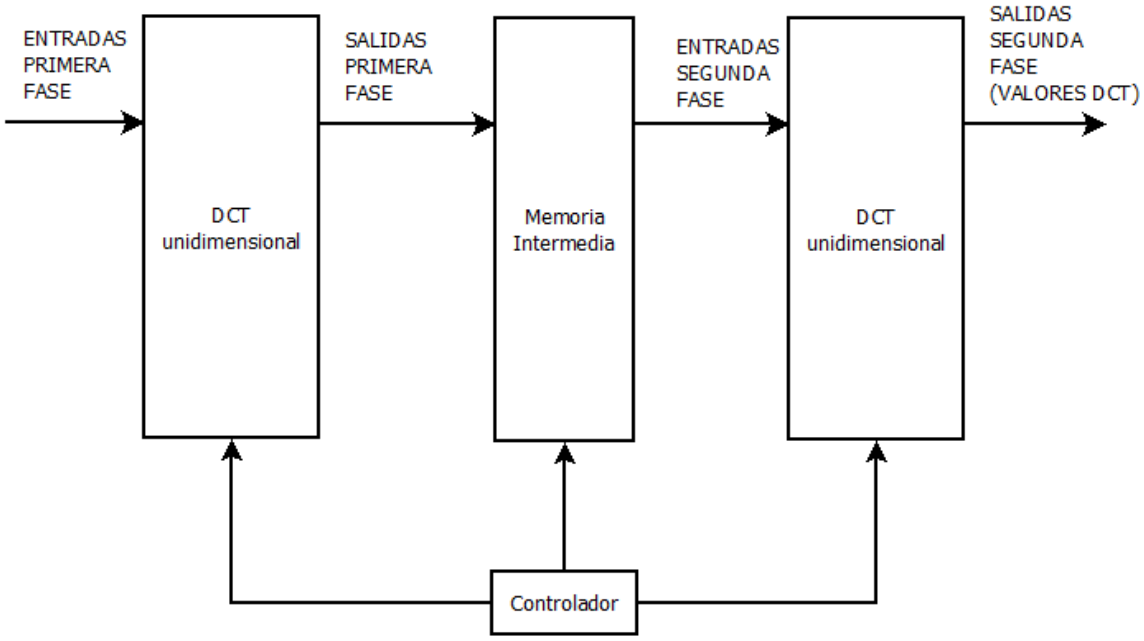


Figura 21. Sistema DCT (I)

Este esquema de funcionamiento tomaría valores para el cálculo en la primera fase y los resultados se irían almacenando en la memoria intermedia. Una vez almacenados los resultados de la primera DCT unidimensional, la segunda DCT unidimensional leería estos valores y, en función de ellos, calcularía los valores DCT finales. Este funcionamiento se dice que es en cascada, porque, si bien para los primeros ocho valores se tarda una serie de ciclos de reloj en mostrarlos, a partir de estos ocho primeros valores el sistema tiene la capacidad de ofrecer ocho nuevos valores en cada ciclo de reloj. Por ello, se trata de una versión del sistema muy rápida pero que utiliza más lógica debido a que, por un lado se requieren dos tablas de memoria de 64 posiciones cada una (para leer y escribir al mismo

tiempo); y por otro lado se requieren dos módulos que implementen la DCT unidimensional.

2) La segunda arquitectura posible sería la que aparece en la Figura 22

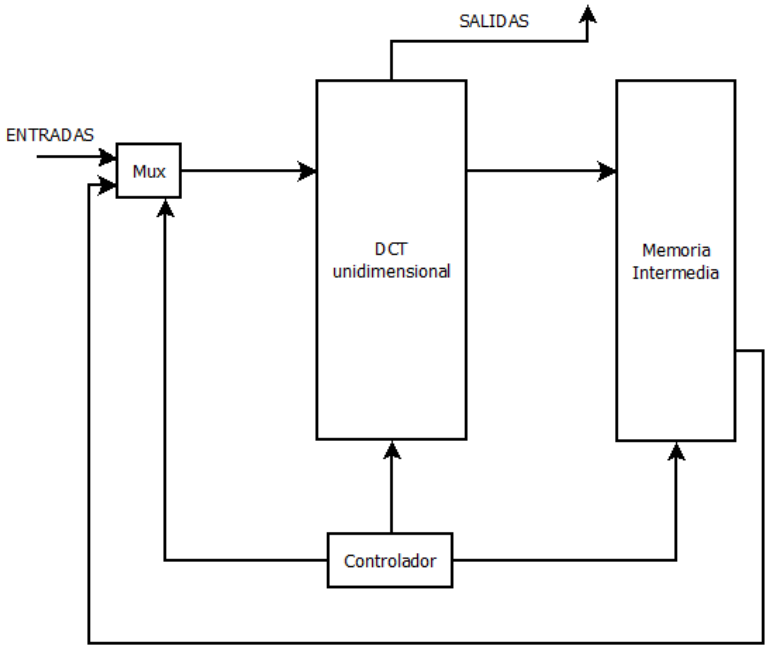


Figura 22. Sistema DCT (II)

En este caso únicamente se utiliza un módulo DCT unidimensional de tal forma que este módulo es el que calcula tanto la DCT por filas como por columnas. En primer lugar se leen las entradas y se calcula la primera DCT, almacenando los resultados en la memoria intermedia. Una vez almacenados los valores, éstos se derivan de nuevo al mismo módulo para realizar la segunda DCT, que, al calcularlos, los sitúa en la salida. La diferencia entre este esquema y el anterior es, por un lado, que ahora no se pueden realizar la primera y la segunda DCT de forma simultánea, y por otro lado, la memoria únicamente necesita una tabla de 64 posiciones y no dos como antes, ya que en este caso no se van a realizar operaciones simultáneas de lectura y de escritura. El resultado de ello es una lógica menor pero también una mayor lentitud a la hora de calcular los valores finales.

Estudiando ambos tipos de estructura, se ha preferido diseñar la segunda debido a la enorme carga lógica que supone la implementación de la primera estructura, prácticamente el doble de la segunda. En cuanto a los tiempos de ejecución, es verdad que el resultado se obtiene más rápidamente en la primera estructura, sin embargo los tiempos obtenidos para el segundo esquema siguen siendo bastante correctos y mucho más rápidos que los obtenidos en software (el análisis de tiempos se lleva a cabo con más detalle en apartados posteriores).

Por tanto, el esquema a seguir es el que aparece en la Figura 22 , donde los módulos que lo conforman son los siguientes:

- DCT unidimensional (dct1d.vhd):

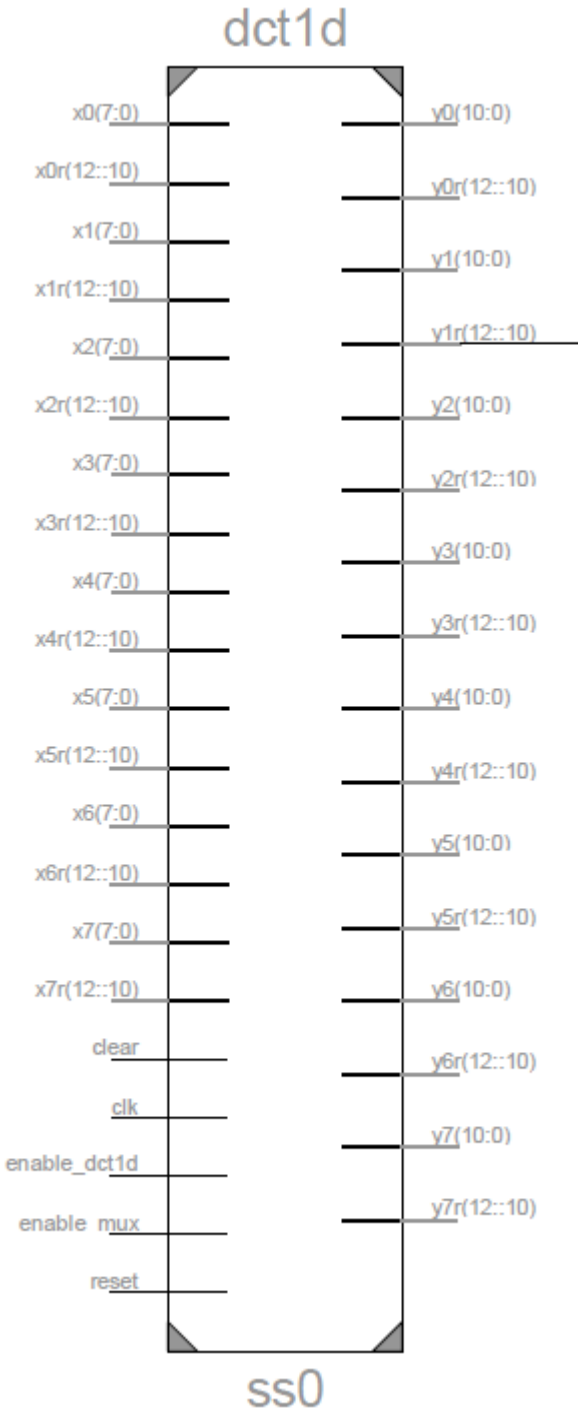


Figura 23. Módulo de cálculo DCT unidimensional

- Entradas:
  - *x0*, *x1*, *x2*, *x3*, *x4*, *x5*, *x6*, *x7*: entradas para la primera DCT (por filas).
  - *x0r*, *x1r*, *x2r*, *x3r*, *x4r*, *x5r*, *x6r*, *x7r*: entradas para la segunda DCT (por columnas).

- *clear*: reset síncrono.
- *clk*: señal de reloj.
- *enable_dct1d*: señal que habilita el funcionamiento del módulo.
- *enable_mux*: señal que controla el multiplexor para modificar las entradas al módulo de cálculo. Adopta nivel bajo para la primera DCT y nivel alto para la segunda.
- *reset*: reset asíncrono.

- Salidas:
  - *y0*, *y1*, *y2*, *y3*, *y4*, *y5*, *y6*, *y7*: salidas de la segunda DCT (por filas). Los valores finales.
  - *y0r*, *y1r*, *y2r*, *y3r*, *y4r*, *y5r*, *y6r*, *y7r*: salidas de la primera DCT (por columnas).

Este módulo realiza las operaciones presentadas en el algoritmo a desarrollar. Para ello utiliza señales y variables dentro de procesos tardando 19 ciclos de reloj en mostrar los resultados finales. Además, debido a la necesidad del uso de números decimales, se ha utilizado una librería VHDL que facilita el empleo de este tipo de numeración. La documentación de esta librería se encuentra en [15] y los aspectos básicos se explican en el Anexo A.

- Memoria intermedia (memoria1_dct.vhd):

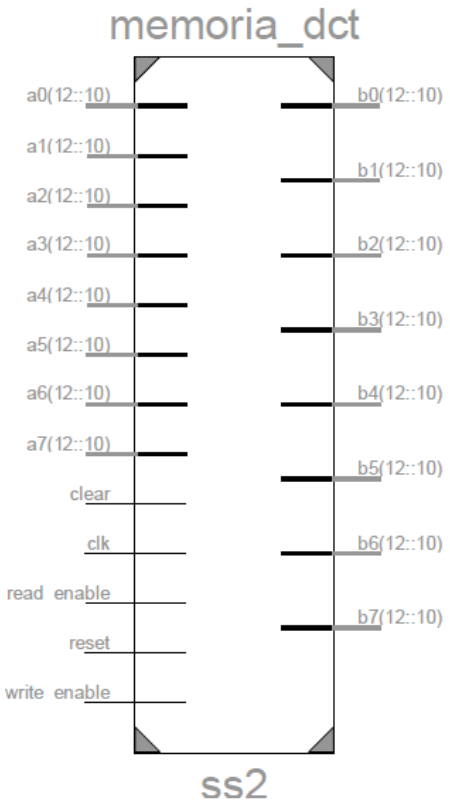


Figura 24. Memoria DCT

- Entradas:
  - *a0*, *a1*, *a2*, *a3*, *a4*, *a5*, *a6*, *a7*: entradas numéricas a memoria.
  - *clear*: reset síncrono.
  - *clk*: señal de reloj.
  - *read_enable*: señal de habilitación para la lectura de la memoria.
  - *reset*: reset asíncrono.
  - *write_enable*: señal de habilitación para la escritura de la memoria.
- Salidas:
  - *b0*, *b1*, *b2*, *b3*, *b4*, *b5*, *b6*, *b7*: salidas de memoria.

La memoria consiste en una tabla de 64 posiciones a las que se accede de 8 valores en 8 valores, tanto para la escritura como para la lectura, con señales de habilitación y de inicialización.

- Controlador (controlador_dct.vhd):

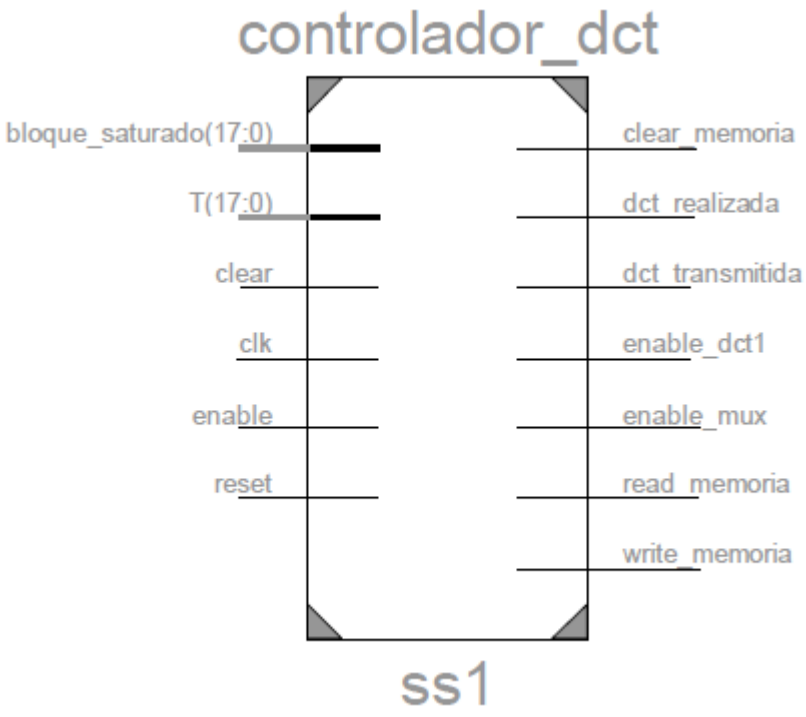


Figura 25. Controlador DCT

- Entradas:
  - *bloque_saturado*: número de bloques saturados detectados durante la lectura de la imagen.
  - *T*: número de bloques totales de la imagen.
  - *clear*: reset síncrono.
  - *clk*: señal de reloj.

- *enable*: señal de habilitación del módulo.
- *reset*: reset asíncrono.

- Salidas:
  - *clear_memoria*: señal de inicialización de la memoria intermedia.
  - *dct_realizada*: señal que se activa durante los ciclos de reloj en los cuales las salidas muestran los resultados finales.
  - *dct_transmitida*: se activa durante el ciclo de reloj en el cual aparece el último grupo de resultados finales.
  - *enable_dct1*: señal de habilitación del módulo que DCT unidimensional.
  - *enable_mux*: señal de control para el multiplexor que selecciona la entrada al módulo unidimensional.
  - *read_memoria*: señal de habilitación para la lectura de la memoria intermedia.
  - *write_memoria*: señal de habilitación para la escritura de la memoria intermedia.

Este bloque realiza las funciones de control del sistema de cálculo DCT. Su funcionamiento es de la siguiente forma: en primer lugar habilita el módulo DCT unidimensional con los valores de la imagen como entradas y se realiza la DCT unidimensional de las 8 filas del bloque a analizar. A continuación habilita la memoria para almacenar los resultados. Una vez guardados los 64 valores los realimenta de nuevo al módulo DCT unidimensional pero esta vez organizados por columnas. El módulo realiza de nuevo la DCT y muestra los resultados finales. Durante estos instantes en los que aparecen los valores finales, la señal *dct_realizada* se activa para informar al resto del sistema de que la DCT ha sido calculada.

Durante el tiempo en que el módulo completo se encuentra realizando cálculos, el controlador se encuentra en un estado llamado *calculo_dct* para pasar a otro llamado *fin_dct* cuando el cálculo de la DCT de todos los bloques finaliza.

Finalmente, al igual que durante la lectura de la imagen, en la FPGA no existe memoria suficiente como para almacenar la DCT de una imagen completa. De ahí que en este caso, al igual que en el anterior, se emplee una memoria externa almacenar estos valores DCT finales. Al mismo tiempo que dichos resultados aparecen en las salidas del sistema DCT se van almacenando en memoria externa.

4.3.4 Sistema de cálculo de la matriz de cuantificación Q (sistema_q_por8.vhd)

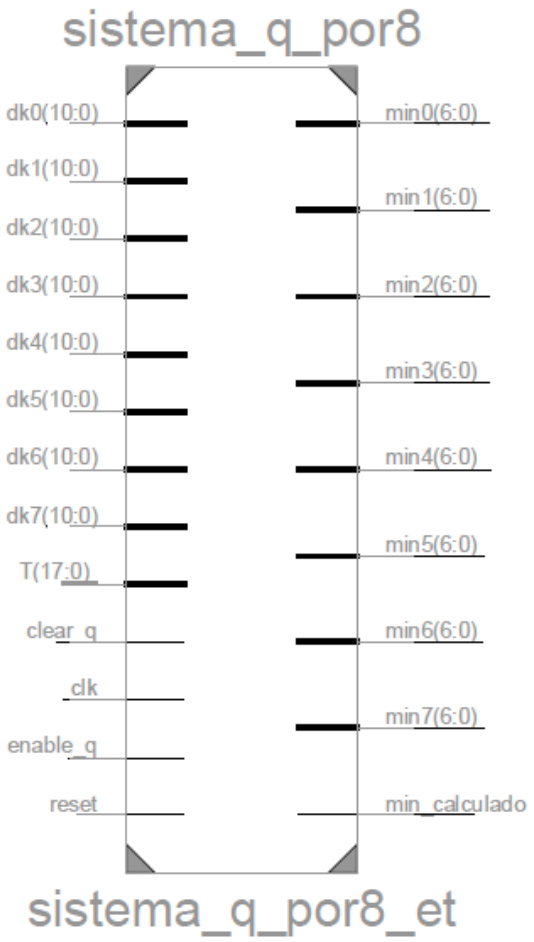


Figura 26. Sistema de cálculo de la matriz Q

- Entradas:
  - *dk0*, *dk1*, *dk2*, *dk3*, *dk4*, *dk5*, *dk6*, *dk7*: entradas para el cálculo de los elementos *q* de la matriz de cuantificación. Son los valores DCT que se calcularon con el sistema DCT.
  - *T*: número de bloques que se van a analizar. Es igual al número de bloques totales de la imagen menos los saturados.
  - *clear_q*: reset síncrono.
  - *clk*: señal de reloj.
  - *enable_q*: señal que habilita el funcionamiento del sistema.
  - *reset*: reset asíncrono.
- Salidas:
  - *min0*, *min1*, *min2*, *min3*, *min4*, *min5*, *min6*, *min7*: valores de *q* calculados.

- *min_calculado*: señal que se activa en el momento en que los valores correctos de *q* han sido calculados.

En primer lugar es necesario decir que el sistema presentado en la Figura 26 engloba en su interior 8 módulos de cálculo de elementos *q* de la matriz de cuantificación, lo que permite calcular de forma simultánea 8 elementos. Cada uno de estos módulos presenta la forma de la Figura 27.

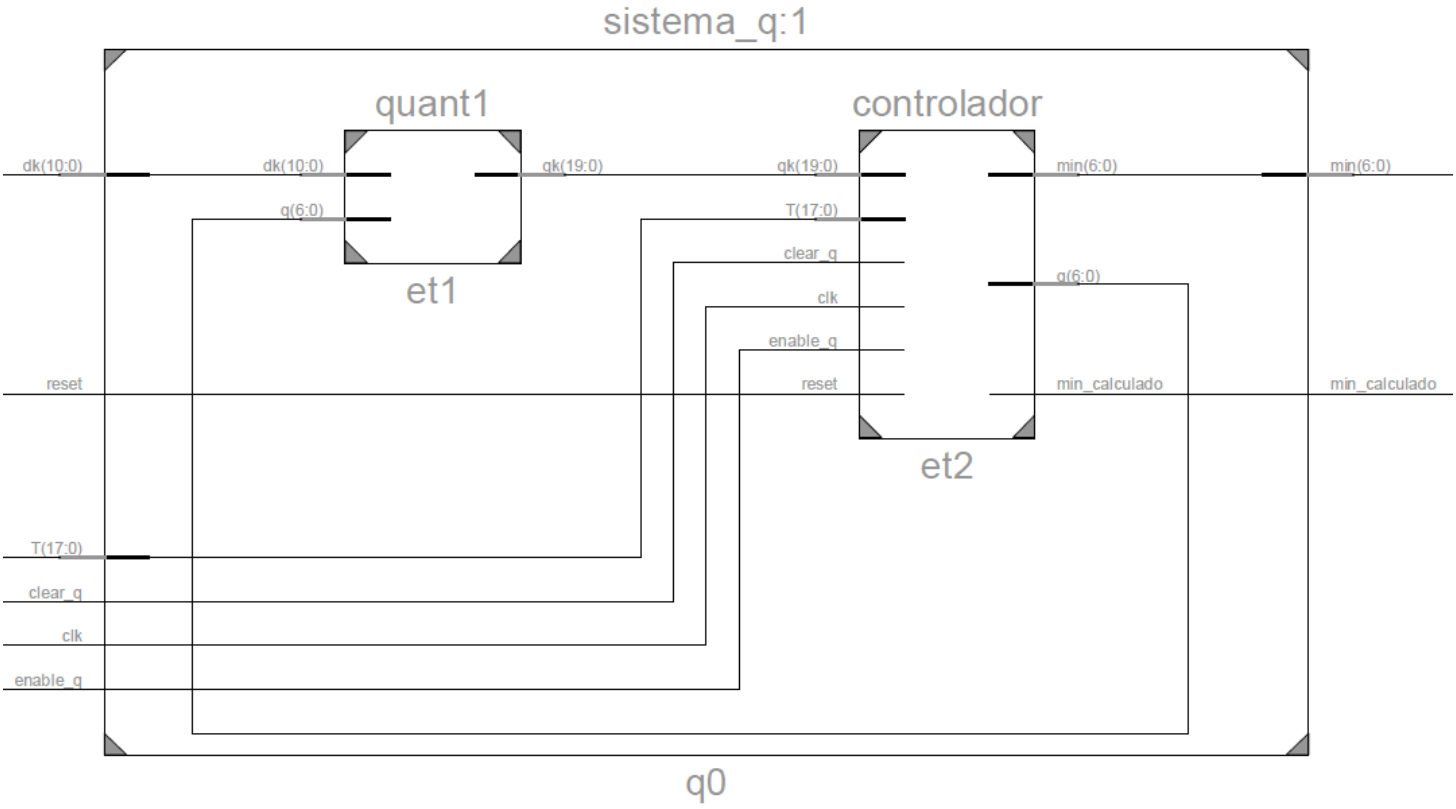


Figura 27. Esquema detallado del módulo de cálculo de los elementos *q*

- Entradas:
  - *dk*: valor DCT necesario para el cálculo de *q*.
  - *reset*: reset asíncrono.
  - *T*: número de bloques que se van a analizar. Es igual al número de bloques totales de la imagen menos los saturados
  - *clear_q*: reset síncrono.
  - *clk*: señal de reloj.
  - *enable_q*: señal que habilita el funcionamiento del sistema.
  - *reset*: reset asíncrono.
- Salidas:
  - *min*: valor de *q* calculado.

- *min_calculado*: señal que se activa en el momento en que el valor correcto de *q* ha sido calculado.

Para comprender mejor el funcionamiento de estos módulos de cálculo, se va a repasar rápidamente el proceso de cálculo de los elementos de la matriz de cuantificación Q. Para el cálculo de un elemento *q* cualquiera de la matriz era necesaria la obtención de la gráfica que ya aparecía en la Figura 4.

Cuya función matemática venía definida por Ecuación 12.

Una vez obtenida, el valor correcto de *q* era el mayor de los mínimos que se encontraban por debajo del umbral. Estos mínimos, que en su conjunto conforman la matriz de cuantificación Q, se almacenan posteriormente en memoria externa.

El funcionamiento del módulo implementado presenta como entrada el valor d_k de la Ecuación 12. A partir de estos valores se irán calculando el valor $E_i(q)$ para $1 \leq q \leq 75$ (el módulo que lo calcula es *quant1*) y existirá un controlador que, en función de los valores E_i, obtendrá el mayor mínimo de la serie.

El módulo *quant1* presenta el esquema de la Figura 28.

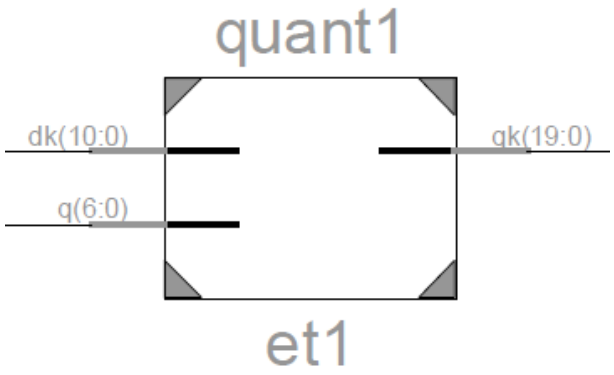


Figura 28. Cuantificación de $E_i(q)$

- Entradas:
  - *dk*: valores DCT.
  - *q*: posibles valores del elemento *q* correcto, $1 \leq q \leq 75$.
- Salidas:
  - *qk*: valor $E_i(q)$.

La misión de este bloque es el cálculo de la fórmula dada en la Ecuación 13.

Ecuación 13. Distribución $E_i(q)$ (sin el parámetro T)

$$E_i(q) = \sum_{k=1}^{T} \left| d_k(i) - q \cdot \text{round}\left(\frac{d_k(i)}{q}\right) \right|$$

NOTA: obsérvese que no presenta el elemento 1/T. Los motivos por los que no se utilizaba dicho elemento se explicaron en el apartado 4.1.2.

Como se puede apreciar la fórmula presenta una serie de operaciones aritméticas que no son triviales de implementar en VHDL, fundamentalmente la división genérica de un valor entre otro que va a ir variando en función del momento del cálculo en que nos encontremos; y el redondeo al entero más próximo de un valor decimal. Sin embargo, aprovechando el resto de las operaciones que aparecen dentro de los símbolos de valor absoluto se ha ideado un método de cálculo mucho más sencillo que ofrece el mismo resultado. El método en cuestión es el siguiente:

La primera operación que se realiza es una división $\frac{d_k(i)}{q}$. Como toda división, ésta se puede expresar mediante la Definición 15.

Definición 15. Definición de una división

$$d_k(i) = q \cdot C + R, \qquad \text{donde C es el cociente y R el resto}$$

La expresión se divide entre *q* para obtener la expresión deseada:

$$\frac{d_k(i)}{q} = C + \frac{R}{q}$$

Esta división hay que redondearla al entero más próximo y multiplicarla por *q*, por lo que:

$$\text{round}\left(\frac{d_k(i)}{q}\right) = C, \qquad \text{si}\,\frac{R}{q} < 0.5$$

$$\text{round}\left(\frac{d_k(i)}{q}\right) = C + 1, \qquad \text{si}\,\frac{R}{q} > 0.5$$

Despejando C se tiene:

$$C = \frac{d_k(i)}{q} - \frac{R}{q}$$

A continuación, si se sustituye esta expresión en la fórmula completa se alcanzan las fórmulas finales para el cálculo de la distribución $E_i(q)$:

Ecuación 14. Ecuación para el cálculo de $E_i(q)$ en caso de R/q<0.5

$$d_k(i) - q \cdot \text{round}\left(\frac{d_k(i)}{q}\right) = d_k(i) - q \cdot \left(\frac{d_k(i)}{q} - \frac{R}{q}\right) = R, \qquad \text{si}\,\frac{R}{q} < 0.5$$

Ecuación 15. Ecuación para el cálculo de $E_i(q)$ en caso de R/q>0.5

$$d_k(i) - q \cdot \text{round}\left(\frac{d_k(i)}{q}\right) = d_k(i) - q \cdot \left(\frac{d_k(i)}{q} - \frac{R}{q} + \frac{q}{q}\right) = R - q, \qquad \text{si}\,\frac{R}{q} > 0.5$$

Para terminar se aplica el valor absoluto, que resulta trivial.

Se puede observar como las expresiones alcanzadas son mucho más sencillas que las que se imponían de inicio, ya que la única dificultad que se presenta es el cálculo de *R*: el resto de la división $\frac{d_k(i)}{q}$. Para la realización de este cálculo se ha adaptado un algoritmo

genérico en VHDL que se ha encontrado en [16] y que se puede consultar en dicha dirección.

Para comprobar que el desarrollo anterior es correcto, se utiliza el Ejemplo 11 y el Ejemplo 12.

Ejemplo 11. Comprobación de Ecuación 14

Se supone:

$$d_k = 4;\ \ q = 3 \ \Rightarrow\ \ C = 1;\ \ R = 1$$

$$d_k(i) - q \cdot \text{round}\left(\frac{d_k(i)}{q}\right) = 4 - 3 \cdot \text{round}(1.33) = 4 - 3 \cdot 1 = 1$$

$$\text{Como } \frac{R}{q} = \frac{1}{3} = 0.33 < 0.5 \ \Rightarrow d_k(i) - q \cdot \text{round}\left(\frac{d_k(i)}{q}\right) = R = 1$$

Ambos resultado son iguales

Ejemplo 12. Comprobación de Ecuación 15

Se supone:

$$d_k = 11;\ \ q = 3 \ \Rightarrow\ \ C = 3;\ \ R = 2$$

$$d_k(i) - q \cdot \text{round}\left(\frac{d_k(i)}{q}\right) = 11 - 3 \cdot \text{round}(3.67) = 11 - 3 \cdot 4 = -1$$

$$\text{Como } \frac{R}{q} = \frac{2}{3} = 0.67 > 0.5 \ \Rightarrow d_k(i) - q \cdot \text{round}\left(\frac{d_k(i)}{q}\right) = R - q = 2 - 3 = -1$$

Ambos resultado son iguales

El otro módulo del sistema es el controlador, con el esquema que aparece en la Figura 29.

- Entradas:
  - *qk*: valor $E_i(q)$ procedente del módulo *quant1*.
  - *T*: número de bloques que se van a analizar. Es igual al número de bloques totales de la imagen menos los saturados
  - *clear_q*: reset síncrono.
  - *clk*: señal de reloj.
  - *enable_q*: señal que habilita el funcionamiento del sistema.
  - *reset*: reset asíncrono.

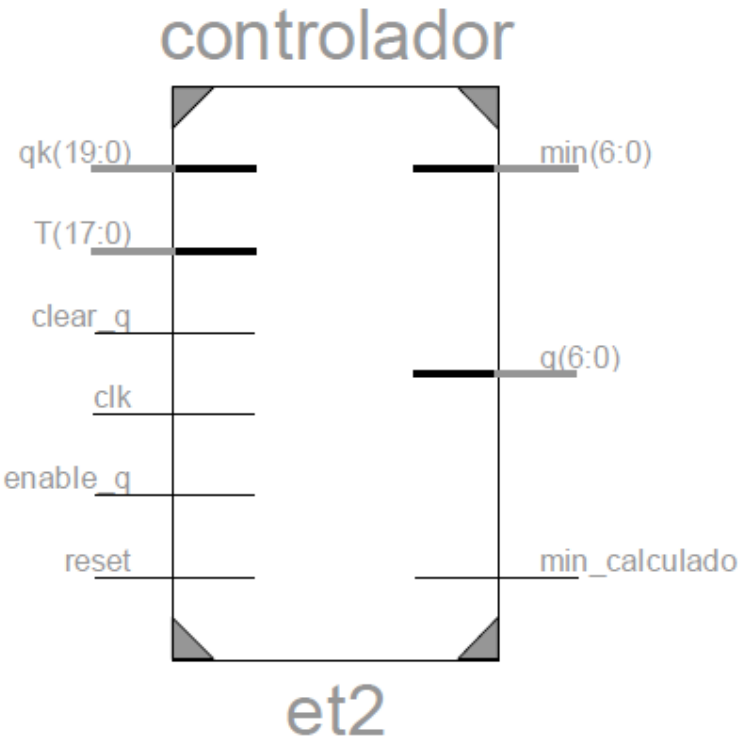


Figura 29. Controlador del cálculo de Q

- Salidas:
  - *min*: valor de *q* calculado.
  - *q*: posibles valores del elemento *q* correcto, $1 \leq q \leq 75$.
  - *min_calculado*: señal que se activa en el momento en que el valor correcto de *q* ha sido calculado.

Este controlador se encarga de buscar los mínimos dentro de la función que se tiene que representar y aportar los valores correctos de los elementos *q* de la matriz de cuantificación. Este bloque va recibiendo progresivamente en el tiempo los valores de $E_i(q)$ procedentes del módulo *quant1* correspondiente y los va almacenando estudiando cuál es su progresión, es decir, si se encuentra en una fase creciente de la serie o decreciente. Para ello compara el último valor recibido con el anterior, determinando si es mayor o menor. Se emplea una máquina de estados sencilla con únicamente dos estados: *espera1* y *espera2*; donde *espera1* representa los momentos en que la función se encuentra creciendo y *espera2* representa el tiempo en que la función se encuentra decreciendo. Con ello, una transición *espera1- espera2* supone que se ha alcanzado un máximo (que en este caso no interesa), mientras que una transición *espera2- espera1* supone un mínimo (que son los que se deben detectar). De esta forma detectando las transiciones *espera2- espera1* se puede determinar cuando se produce un mínimo. Después, no hay más que el sistema se quede con el mayor mínimo (en el eje de la *q*) para conseguir el valor correcto.

Para mayor ilustración, en la Figura 30 se indican los estados del controlador a lo largo del cálculo.

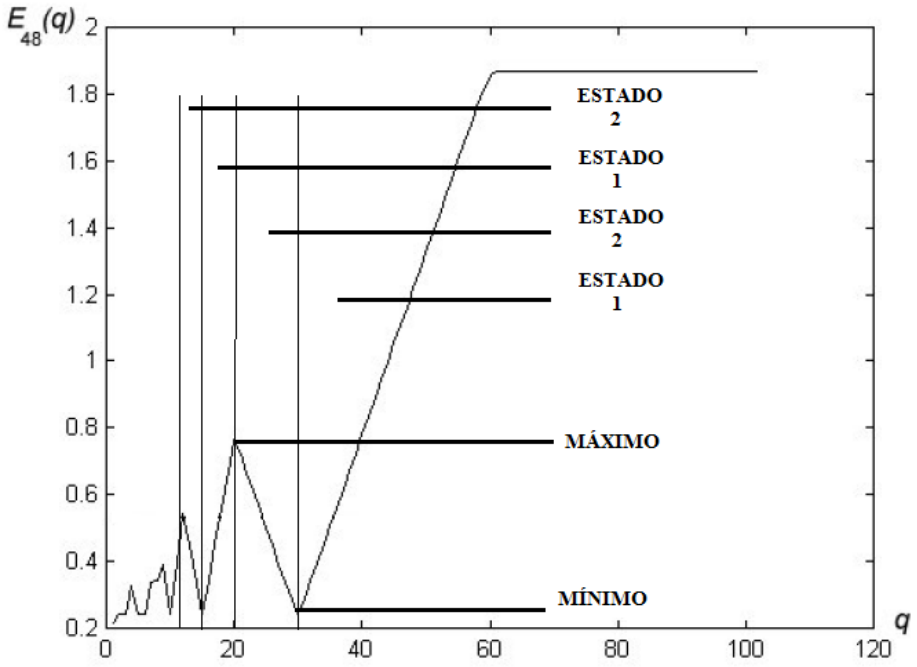


Figura 30. Estados en el cálculo de *q*

La teoría establece la necesidad de emplear un umbral por encima del cual no se deben considerar los posibles mínimos que aparezcan, por ser éstos incorrectos. En el apartado 4.1.3 se dieron las razones por las cuales este umbral no se iba a utilizar, por ello se ha implementado otro método que evite la aparición de estos "falsos" mínimos. A lo largo de las pruebas realizadas se ha apreciado como los mínimos correctos siempre se asocian a un valor de E_i muy parecido (en la figura anterior se puede apreciar como el valor de todos estos mínimos en ordenadas se encuentra siempre en torno a 0.2-0.4) y que los mínimos incorrectos suelen presentar valores de E_i mucho mayores a los valores correctos. De ahí que se haya impuesto que todos los mínimos considerados, y en concreto el mayor de ellos por ser el valor del elemento *q*, se deben encontrar siempre en un intervalo de valores de E_i parecido, descartando todos aquellos que se encuentren por encima de dicho intervalo. La selección de la amplitud del intervalo ha sido completamente empírica empleando uno del que se han comprobado buenos resultados. Este umbral se corresponde con el valor de E_i del anterior mínimo correcto más 50, de forma que si se encuentra un mínimo cuyo valor de $E_i(q)$ sea, por ejemplo, $E_i(q-1) + 75$; se considera que el valor es incorrecto y no se tendrá en cuenta. En el caso del primer mínimo encontrado, éste siempre se va a considerar como correcto y va a servir de referencia para el segundo mínimo.

A modo de ilustración aparece la Figura 31.

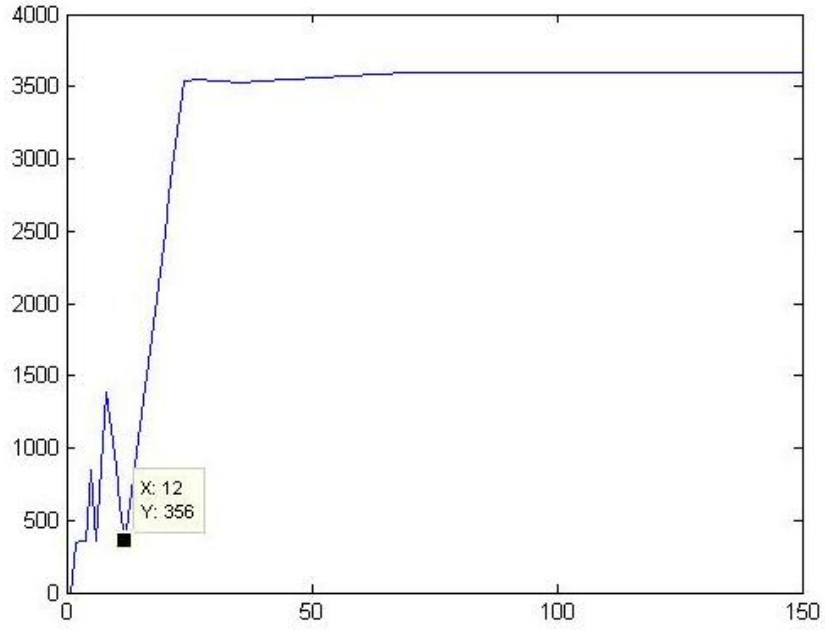


Figura 31. Proceso de cálculo de *q* (I)

En la Figura 31 se observa como el valor correcto de q es 12. Sin embargo, si se presta atención, en la Figura 32, se puede observar otro mínimo a valores mayores de q.

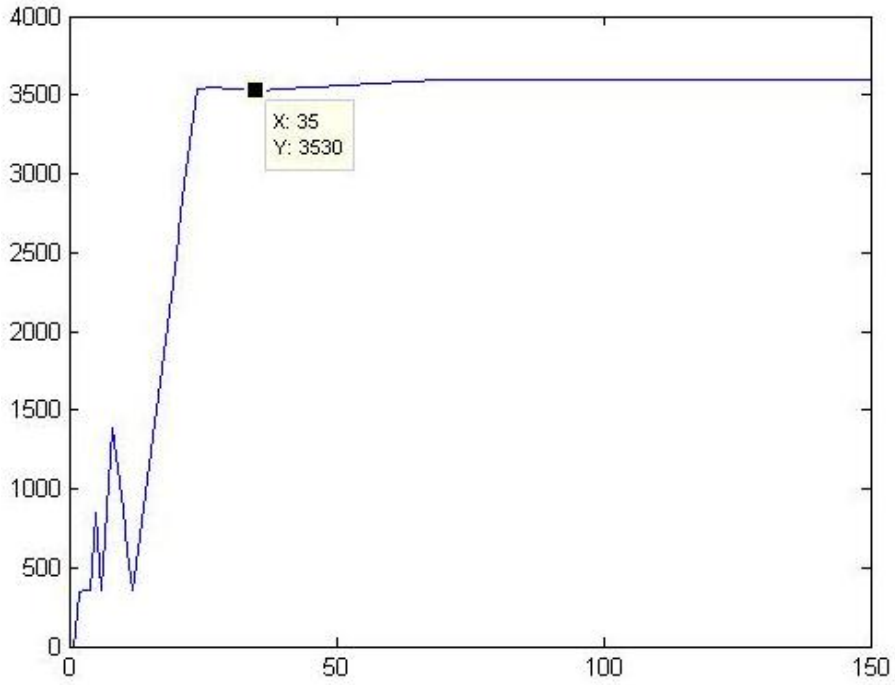


Figura 32. Proceso de cálculo de *q* (II)

Este falso mínimo sería detectado por el sistema desarrollado si no se implementara ningún método de control. En el momento en que el sistema detecta este mínimo compara su valor de $E_i(35)$ con el de $E_i(12)$ + 50:

$$E_i(12) = 356;\ \ E_i(35) = 3530$$

$3530 > 356 + 50 = 406$; por lo que el mínimo no se tiene en cuenta

4.3.5 Sistema de cálculo de S (sistema_calculo_S.vhd)

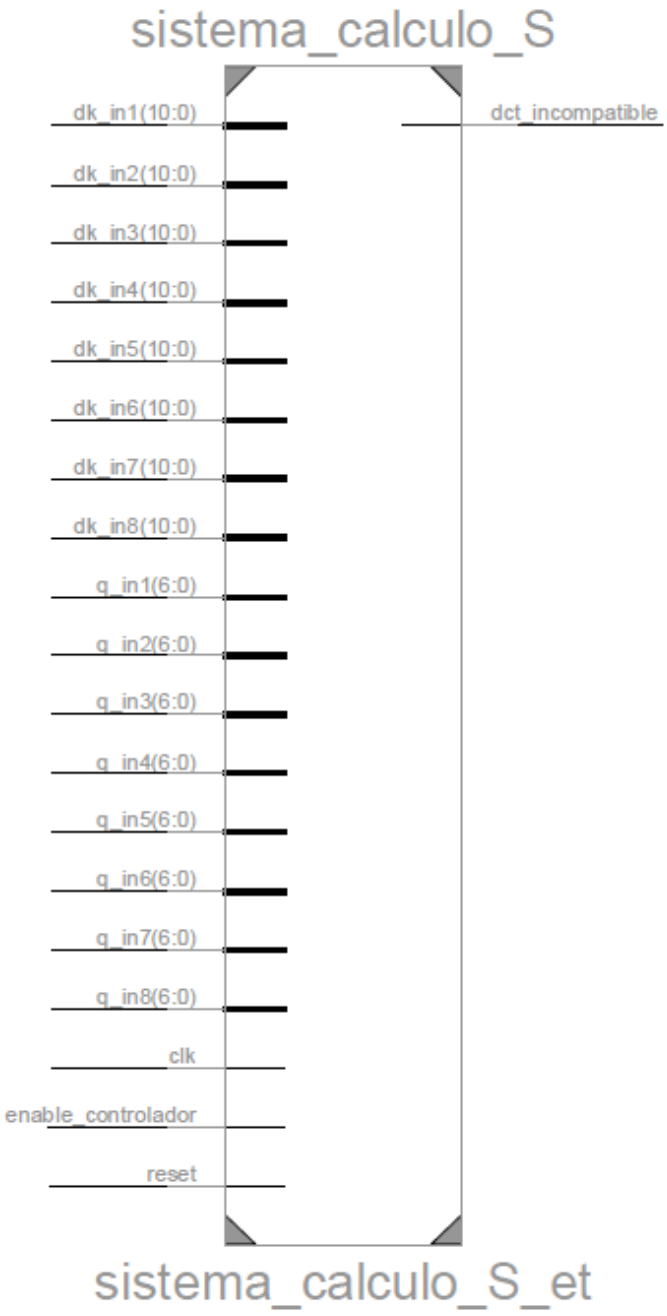


Figura 33. Sistema de cálculo del parámetro S

- Entradas:
  - *dk_in1*, *dk_in2*, *dk_in3*, *dk_in4*, *dk_in5*, *dk_in6*, *dk_in7*, *dk_in8*: entradas de los valores DCT.
  - *q_in1*, *q_in2*, *q_in3*, *q_in4*, *q_in5*, *q_in6*, *q_in7*, *q_in8*: entradas de los valores *q* de la matriz de cuantificación Q.

- *clk*: señal de reloj.
- *enable_controlador*: señal de habilitación del módulo.
- *reset*: reset asíncrono.

- Salidas:
  - *dct_incompatible*: señal que se activa cuando el bloque de la imagen analizado presenta un valor S mayor que 16 y, por tanto, es incompatible.

Este módulo se encarga de calcular, para un bloque de la imagen dado, el parámetro S que venía definido por la Ecuación 8:

$$S = \sum_{i=1}^{64} \left| DCT(B)(i) - Q(i) \cdot \text{round}\left(\frac{DCT(B)(i)}{Q(i)}\right) \right| \leq 16$$

Como se aprecia el parámetro se calcula a partir de un sumatorio de 64 elementos, que se corresponden con cada uno de los elementos de un bloque de imagen analizado. De esta forma, la expresión a calcular se puede entender de la siguiente forma:

$$S = S_1 + S_2 + \cdots + S_{64}$$

El sistema implementado *sistema_calculo_S* incorpora 8 módulos de cálculo de S_i, que actúan de forma simultánea. Esto quiere decir que durante el primer ciclo de cálculo se obtienen los valores S_1, S_2,…, S_8; lo que supone la necesidad de 8 ciclos de cálculo para obtener el valor final de S.

Es importante observar que las operaciones aritméticas para el cálculo los elementos de S son similares a las que ya se emplearon para la obtención de la matriz de cuantificación Q, por lo que los desarrollos y simplificaciones que se emplearon en aquellas operaciones se aplican de la misma forma en este caso.

Cada módulo de cálculo S_i tiene la forma de la Figura 34:

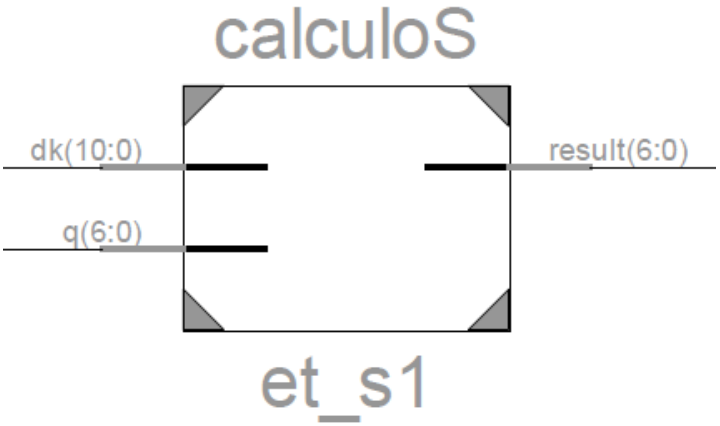


Figura 34. Módulo de cálculo de S

- Entradas:
  - *dk*: valor DCT.
  - *q*: valor *q* de la matriz de cuantificación Q.
- Salidas:
  - *result*: valor S_i.

El funcionamiento de este módulo es similar al que presentaba *quant1* en el apartado 4.3.4, ya que la operación que se desea realizar es la misma. La única diferencia radica en que, en este caso, el módulo actúa de forma combinacional, sin almacenar valores y únicamente realizando la operación correspondiente en función de los valores de entrada.

Para el almacenamiento de los resultados y la realización del sumatorio aparece el módulo de la Figura 35:

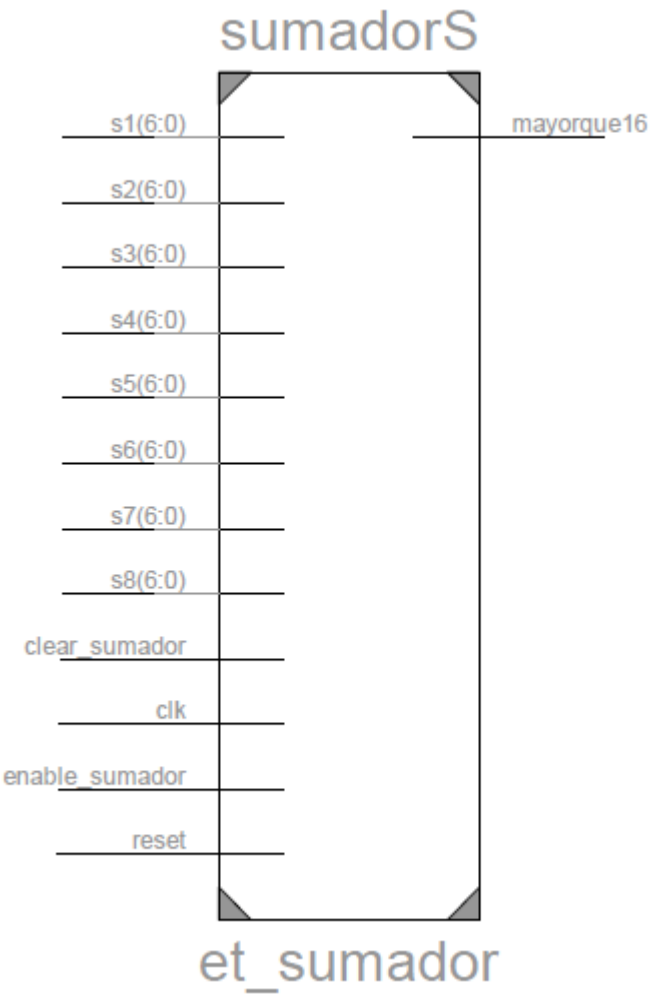


Figura 35. Sumador de términos S_i

- Entradas:
  - *s1*, *s2*, *s3*, *s4*, *s5*, *s6*, *s7*, *s8*: elementos S_i procedentes de los ocho módulos de cálculo descritos anteriormente.

- *clear_sumador*: reset síncrono.
- *clk*: señal de reloj.
- *enable_sumador*: señal que habilita el funcionamiento del bloque.
- *reset*: reset síncrono.

- Salidas:
  - *mayorque16*: señal que se activa cuando el bloque de la imagen analizado presenta un valor S mayor que 16.

El funcionamiento de este bloque es muy sencillo; simplemente recibe los valores S_i procedentes de los 8 módulos de cálculo y los va almacenando, sumándolos, en una señal interna que, una vez recibidos los 64 elementos, comparará con el valor 16 para determinar si el bloque analizado es incompatible o no.

Finalmente, es necesario un controlador para el sistema de cálculo:

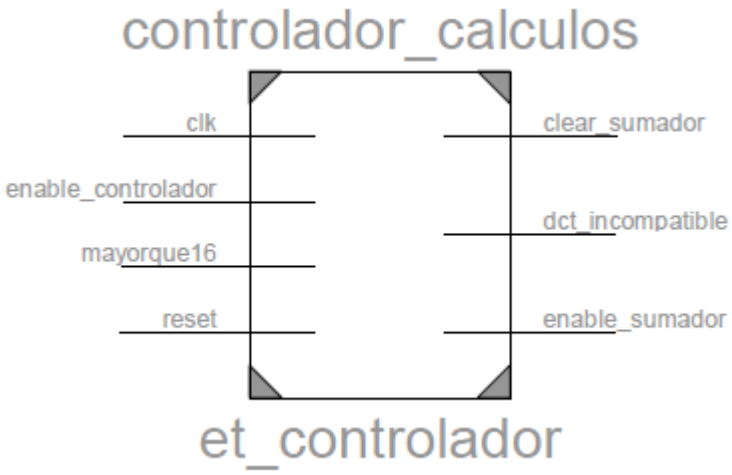


Figura 36. Controlador para el cálculo de S

- Entradas:
  - *clk*: señal de reloj.
  - *enable_controlador*: señal que habilita el funcionamiento del controlador.
  - *mayorque16*: señal procedente del sumador que avisa cuando se detecta un bloque incompatible.
  - *reset*: reset asíncrono.

- Salidas:
  - *clear_sumador*: señal de inicialización del sumador, que implica la puesta a cero de la variable interna que realiza el sumatorio.
  - *dct_incompatible*: señal que se activa cuando el bloque de la imagen analizado presenta un valor S mayor que 16 y, por tanto, es incompatible. Durante el estado *fin_calculo* presenta el mismo valor que la señal *mayorque16*, durante el resto del tiempo se mantiene a nivel bajo.

- *enable_sumador*: señal que habilita el funcionamiento del módulo sumador.

Este bloque presenta un funcionamiento basado en una máquina de estados con tres posibles estados: *espera*, *calculo* y *fin_calculo*. El esquema que se sigue se muestra en la Figura 37.

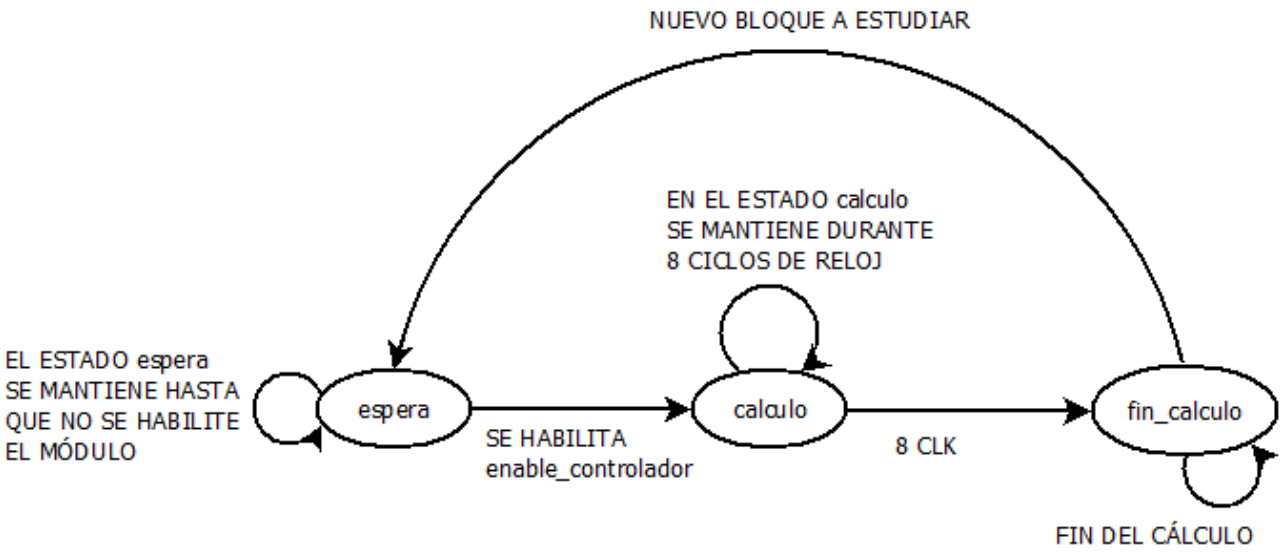


Figura 37. Máquina de estados del módulo de cálculo de S

Inicialmente el controlador se mantiene en el estado *espera* hasta que se habilita, por parte del controlador global del sistema, la señal que lo habilita. En ese momento pasa al estado *calculo*, en donde comenzará a recibir los valores S_i. Como se tienen 8 módulos de cálculo S_i actuando simultáneamente, para calcular los 64 valores del sumatorio S se requieren 8 ciclos de reloj. Una vez transcurridos, se compara el parámetro S obtenido con el valor 16 para determinar si el bloque analizado es incompatible o no. En caso en que sea incompatible se activa la señal *dct_incompatible*, que será empleada por el controlador global del sistema para contabilizar el número de bloques incompatibles. En caso contrario, la señal no se activa y es necesario realizar otro tipo de análisis para determinar la compatibilidad del bloque estudiado. Llegado a este punto se pasaría a analizar un nuevo bloque (en caso en que haya más) y se repetirían todas las operaciones descritas.

4.3.6 Sistema IDCT (sistema_idct.vhd)

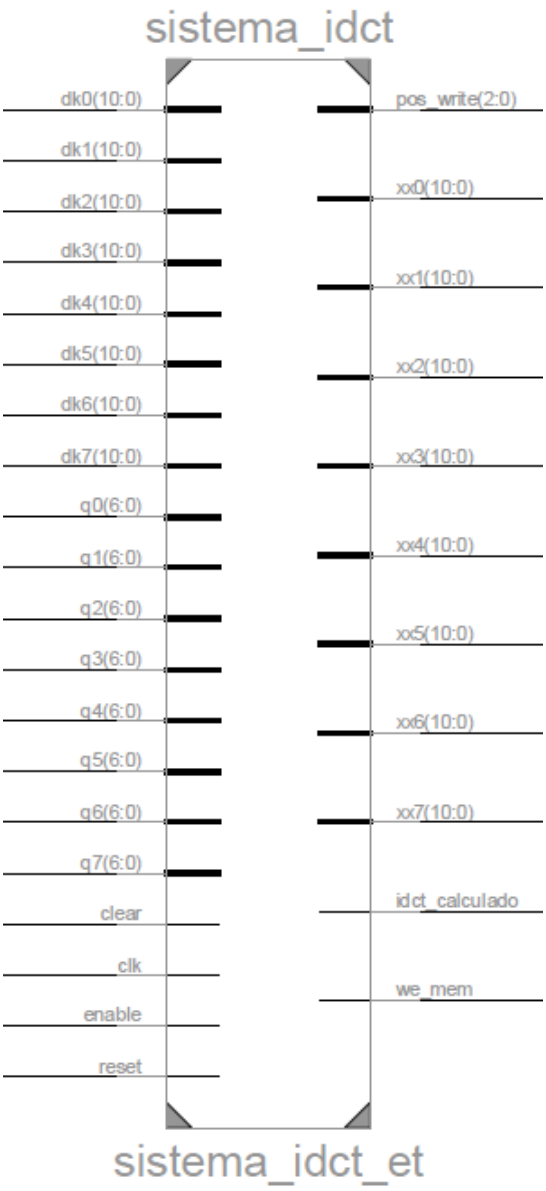


Figura 38. Sistema IDCT

- Entradas:
  - *dk0*, *dk1*, *dk2*, *dk3*, *dk4*, *dk5*, *dk6*, *dk7*: valores DCT.
  - *q0*, *q1*, *q2*, *q3*, *q4*, *q5*, *q6*, *q7*: valores de la matriz de cuantificación Q.
  - *clear*: reset síncrono.
  - *clk*: señal de reloj.
  - *enable*: señal que habilita el funcionamiento del módulo.
  - *reset*: reset asíncrono.

- Salidas:
  - *pos_write*: señal que indica la posición de memoria donde se debe almacenar los valores IDCT calculados.
  - *xx0, xx1, xx2, xx3, xx4, xx5, xx6, xx7*: valores IDCT.
  - *idct_calculado*: señal que se activa cuando la IDCT del bloque analizado ha sido calculada completamente. Como los valores IDCT salen del módulo de 8 en 8, esta señal se activa en el ciclo de reloj correspondiente al último de grupo de 8 valores IDCT.
  - *we_mem*: señal que habilita el almacenamiento de los valores IDCT en la memoria de comparación que se describe en el apartado 4.3.7.

Este módulo presenta varias etapas de funcionamiento que se ajustan a la teoría explicada en el apartado 2.3 del presente documento. Para un bloque determinado, una vez calculado su parámetro S y obtenido que éste es menor que 16, no es posible saber todavía si dicho bloque es compatible o incompatible. Para alcanzar una conclusión se necesita seguir, de forma muy resumida, el siguiente procedimiento:

1) A partir del bloque estudiado, en su forma DCT, se debe calcular el bloque formado por múltiplos de la matriz de cuantificación Q que se encuentre más cercano, elemento a elemento, al bloque DCT.

2) A ese bloque de múltiplos se le aplica la IDCT y se compara el resultado de dicha IDCT con el bloque original de la imagen.

3) Si estos son similares el bloque es compatible, por el contrario no lo es.

El sistema estudiado ahora mismo implementa la lógica necesaria para llegar hasta el cálculo de la IDCT, teniendo como salida, precisamente, esos valores IDCT.

El esquema que presenta el sistema es el mostrado en la Figura 39.

De forma similar al resto del proyecto, las operaciones en este caso también se realizan de 8 en 8; almacenando los valores de 8 en 8 y calculando la IDCT sobre vectores de dimensión 8.

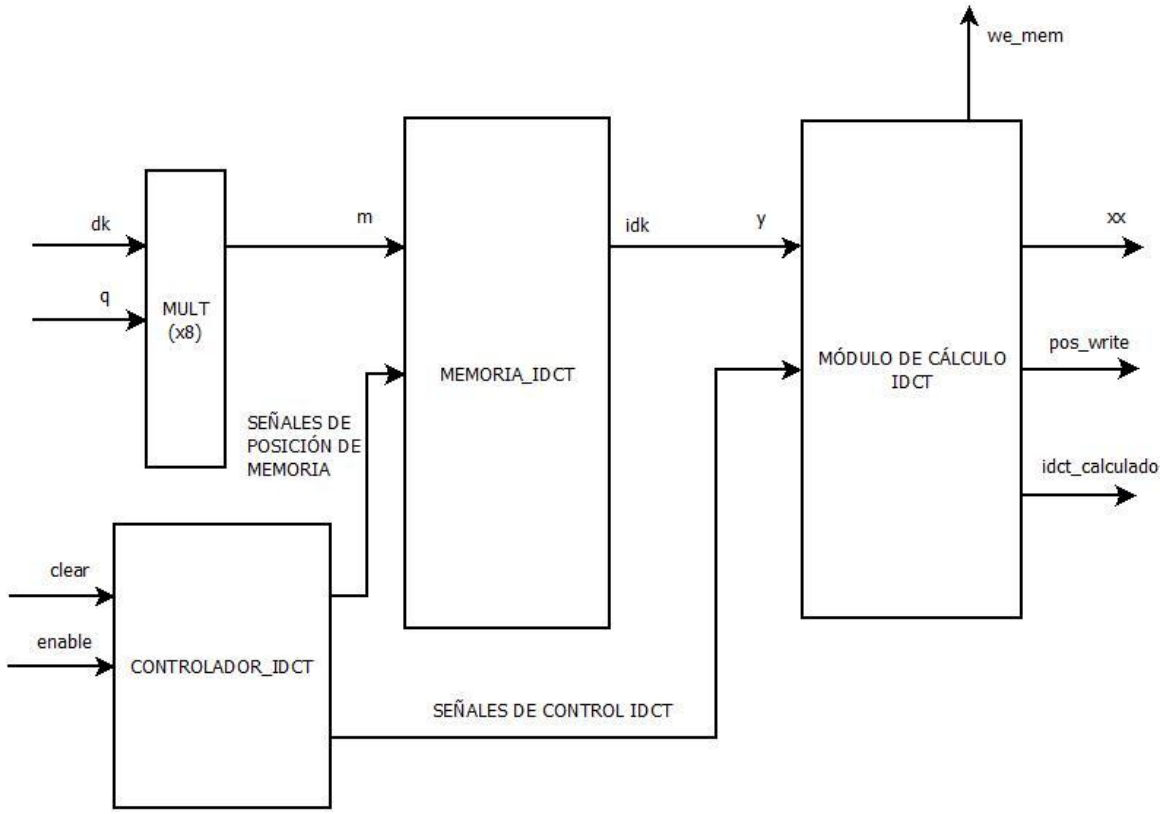


Figura 39. Esquema del módulo de cálculo IDCT

El módulo *multiplo_ent*:

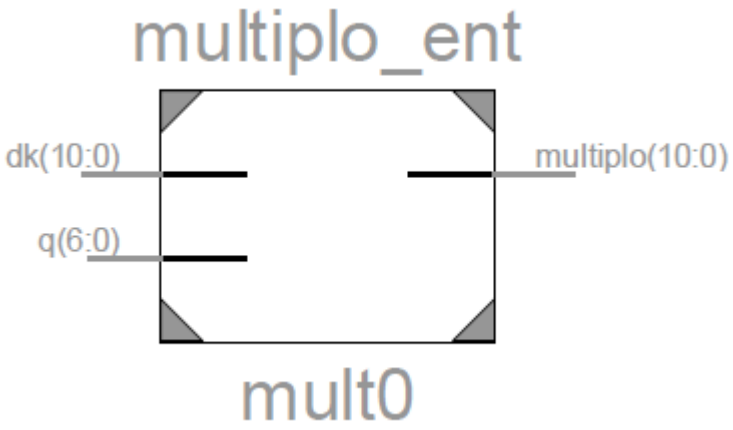


Figura 40. Cálculo de los múltiplos de la matriz Q

- Entradas:
  - *dk*: valor DCT.
  - *q*: valor de la matriz de cuantificación Q.

- Salidas:
  - *multiplo*: múltiplo del elemento *q* más cercano a *dk.*

El funcionamiento de este bloque es combinacional y se basa en la Ecuación 16:

Ecuación 16. Cálculo de múltiplos de Q más cercanos a los valores DCT

$$\text{mult} = q \cdot \text{round}(\frac{dk}{q})$$

Como se puede observar esta operación se vuelve a repetir, siendo la misma que aparecía para el cálculo de la matriz de cuantificación Q en el apartado 4.3.4 y para el cálculo del parámetro S en el apartado 4.3.5; es por ello, que los desarrollos y las operaciones aplicadas para su cálculo siguen siendo las mismas y se pueden consultar en sendos apartados.

Introduciendo, para cada bloque, los valores DCT así como la matriz de cuantificación Q se va obteniendo la matriz P_1 de múltiplos de Q más cercana al bloque DCT.

A continuación, esta matriz resultante se almacena en una memoria interna que ha sido definida como *memoria_idct* en la Figura 41:

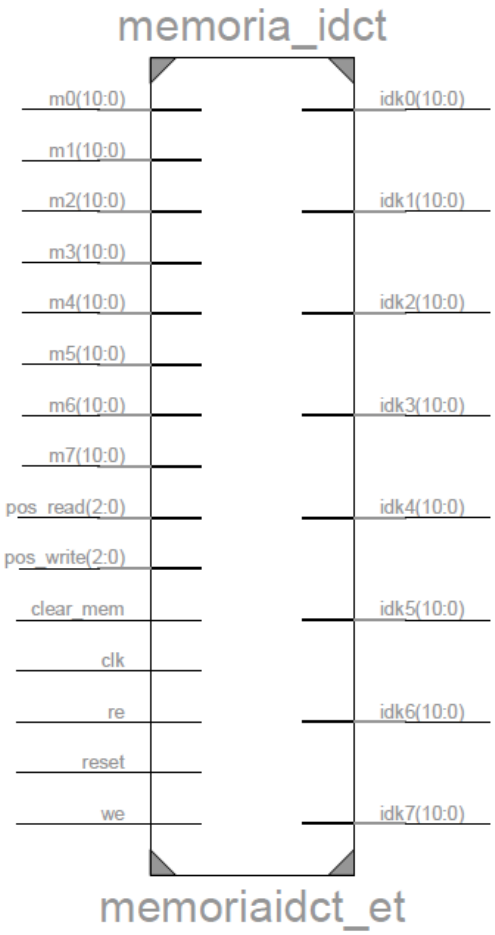


Figura 41. Memoria IDCT

- Entradas:
  - *m0*, *m1*, *m2*, *m3*, *m4*, *m5*, *m6*, *m7*: componentes de la matriz de múltiplos que van a ser almacenados.

- *pos_read*: posición que se desea leer de memoria.
- *pos_write*: posición que se desea escribir en memoria.
- *clear_mem*: reset síncrono.
- *clk*: señal de reloj.
- *re*: señal que habilita la lectura de la memoria.
- *reset*: reset asíncrono.
- *we*: señal que habilita la escritura de la memoria.

- Salidas:
  - *idk0*, *idk1*, *idk2*, *idk3*, *idk4*, *idk5*, *idk6*, *idk7*: valores de memoria que se leen.

Con los valores de los múltiplos almacenados en memoria, el siguiente punto del proceso consiste en realizar la transformada inversa del bloque. El sistema que implementa esta operación aparece en la Figura 42:

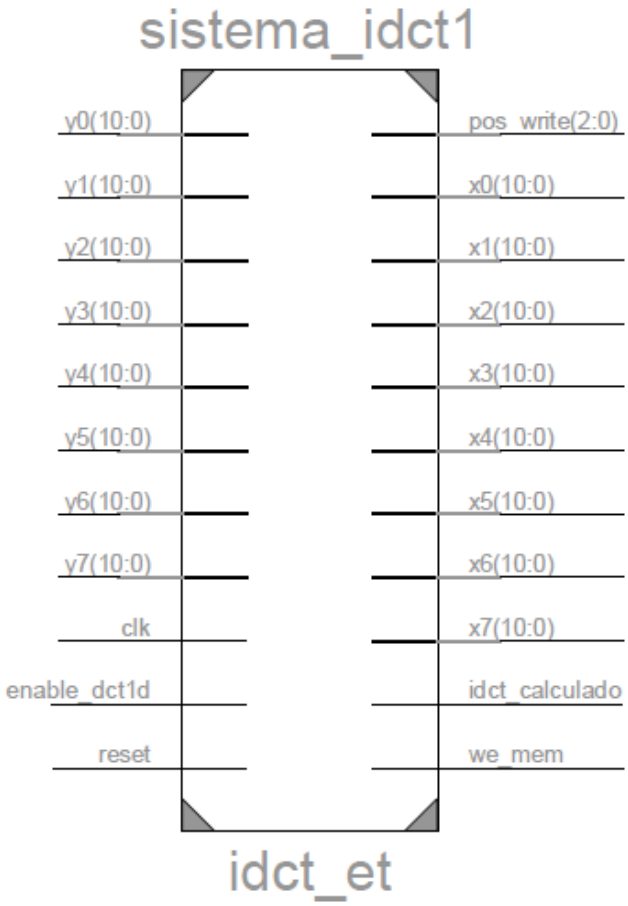


Figura 42. Módulo de cálculo de la IDCT

- Entradas:
  - *y0*, *y1*, *y2*, *y3*, *y4*, *y5*, *y6*, *y7*: valores de la matriz de múltiplos.
  - *clk*: señal de reloj.

- *enable_dct1d*: señal que habilita el funcionamiento del módulo.
- *reset*: reset asíncrono.

- Salidas:
  - *pos_write*: posiciones en las que se van a almacenar los resultados de la IDCT en la memoria de comparación.
  - *x0*, *x1*, *x2*, *x3*, *x4*, *x5*, *x6*, *x7*: resultados IDCT.
  - *idct_calculado*: señal que se activa cuando la IDCT del bloque analizado ha sido calculada completamente. Como los valores IDCT salen del módulo de 8 en 8, esta señal se activa en el ciclo de reloj correspondiente al último de grupo de 8 valores IDCT.
  - *we_mem*: señal que habilita el almacenamiento de los valores IDCT en la memoria de comparación que se describe en el apartado 4.3.7.

A la hora de implementar el cálculo de la IDCT existen las mismas posibilidades de organización que en el caso de la DCT (posibilidades presentadas en el apartado 4.3.3). Además, la IDCT presenta las mismas propiedades de cálculo que la DCT, por lo que una IDCT bidimensional se puede dividir en dos cálculos: IDCT unidimensional por columnas e IDCT unidimensional por filas.

La implementación se ha realizado de forma paralela y similar a la DCT, con la misma organización por la que un mismo módulo de cálculo IDCT unidimensional se utiliza tanto para la IDCT por filas como para la IDCT por filas, con la ayuda de una memoria y el correspondiente controlador.

El algoritmo que se sigue para el cálculo de la IDCT es el inverso al empleado en la DCT, es decir, se ha tomado este último y se ha desarrollado de delante hacia detrás, apareciendo los siguientes pasos de cálculo:

1) Se parte de los valores de la matriz de múltiplos m_0, m_1, …, m_7 y se obtiene S_0, S_1, …, S_7:

$$S_0 = m_1\ x\ y_0;\quad S_1 = m_2\ x\ y_1;\quad S_2 = m_3\ x\ y_2;\quad S_3 = m_4\ x\ y_3$$

$$S_4 = m_1\ x\ y_4;\quad S_5 = m_5\ x\ y_5;\quad S_6 = m_6\ x\ y_6;\quad S_7 = m_7\ x\ y_7$$

$$m_1 = 2.83; m_2 = 3.9231; m_3 = 3.6955; m_4 = 3.3333;$$
$$m_5 = 2.2222; m_6 = 1.5309; m_7 = 0.7803;$$

2) Se obtiene f_0, f_1, …, f_7 a partir de S_0, S_1, …, S_7:

$$f_0 = S_0;\quad f_1 = S_4;\quad f_2 = S_2;\quad f_3 = S_6$$

$$f_4 = 0.5\ x\ (S_1 + S_7); f_5 = 0.5\ x\ (S_3 + S_5); f_6 = 0.5\ x\ (S_5 - S_3); f_7 = 0.5\ x\ (S_1 - S_7)$$

3) Se obtiene e_0, e_1, …, e_7 a partir de f_0, f_1, …, f_7:

$$e_0 = f_0; e_1 = f_1; e_2 = m_8 \text{ x } f_7 + 0.5 \text{ x } f_6; e_3 = 0.5 \text{ x } (f_4 - f_5)$$

$$e_4 = m_9 \text{ x } f_7 - 0.5 \text{ x } f_6; e_5 = 0.5 \text{ x } (f_2 + f_3); e_6 = 0.5 \text{ x } (f_2 - f_3); \; e_7 = 0.5 \text{ x } (f_4 + f_5)$$

$$m_8 = 0.2091; \; m_9 = 1.2050$$

4) Se obtiene d_0, d_1, …, d_7 a partir de e_0, e_1, …, e_7:

$$d_0 = e_0; \; d_1 = e_1; d_2 = m_{10} \text{ x } e_2; d_3 = m_{11} \text{ x } e_6$$

$$d_4 = e_4; d_5 = m_{12} \text{ x } e_4; d_6 = m_{11} \text{ x } e_3; d_7 = e_7$$

$$m_{10} = 1.8477; \; m_{11} = 1.4142; \; m_{12} = 0.7650$$

5) Se obtiene $idct_0$, $idct_1$, …, $idct_7$ a partir de d_0, d_1, …, d_7 y e_0, e_1, …, e_7:

$$idct_0 = 0.125 \text{ x } (f_0 + f_1 + f_2 + f_3) + 0.25 \text{ x } (f_4 + f_5)$$

$$idct_1 = 0.125 \text{ x } (d_0 - d_1) + 0.25 \text{ x } (d_3 - d_4) + 0.5 \text{ x } (d_5 - d_7)$$

$$idct_2 = 0.125 \text{ x } (d_0 - d_1) - 0.25 \text{ x } (d_3 - d_4) + 0.5 \text{ x } (d_6 - d_5 + d_7)$$

$$idct_3 = 0.125 \text{ x } (d_0 + d_1) - 0.25 \text{ x } (d_4) + 0.5 \text{ x } (d_2 - d_6 + d_5 - d_7)$$

$$idct_4 = 0.125 \text{ x } (d_0 + d_1) - 0.25 \text{ x } (d_4) + 0.5 \text{ x } (d_6 - d_2 + d_7 - d_5)$$

$$idct_5 = 0.125 \text{ x } (d_0 - d_1) + 0.25 \text{ x } (d_4 - d_3) + 0.5 \text{ x } (d_5 - d_6 - d_7)$$

$$idct_6 = 0.125 \text{ x } (d_0 - d_1) + 0.25 \text{ x } (d_3 - d_4) + 0.5 \text{ x } (d_7 - d_5)$$

$$idct_7 = 0.125 \text{ x } (d_0 + d_1) + 0.25 \text{ x } (d_4) - 0.5 \text{ x } (d_7)$$

Finalmente, se requiere un controlador para el sistema completo:

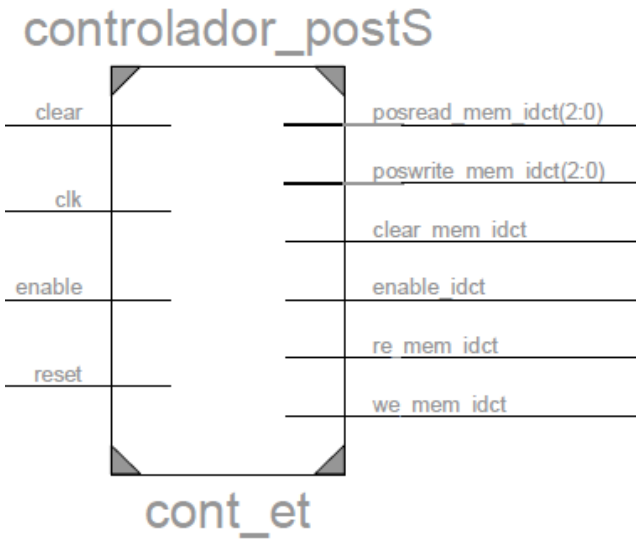


Figura 43. Controlador del sistema IDCT

- Entradas:
  - *clear*: reset síncrono.
  - *clk*: señal de reloj.
  - *enable*: señal que habilita el funcionamiento del módulo.
  - *reset*: reset síncrono.
- Salidas:
  - *posread_mem_idct*: posiciones de memoria en la matriz de múltiplos que se desean leer.
  - *poswrite_mem_idct*: posiciones de memoria en la matriz de múltiplos que se desean escribir.
  - *clear_mem_idct*: señal la inicialización de la memoria de la matriz de múltiplos.
  - *enable_idct*: señal que habilita el módulo de cálculo de la IDCT.
  - *re_mem_idct*: señal que habilita la lectura en la matriz de múltiplos.
  - *we_mem_idct*: señal que habilita la escritura en la matriz de múltiplos.

El funcionamiento de este controlador es idéntico al descrito para el controlador DCT en el apartado 4.3.3.

4.3.7 Memoria de comparación (memoria_comparacion_final.vhd)

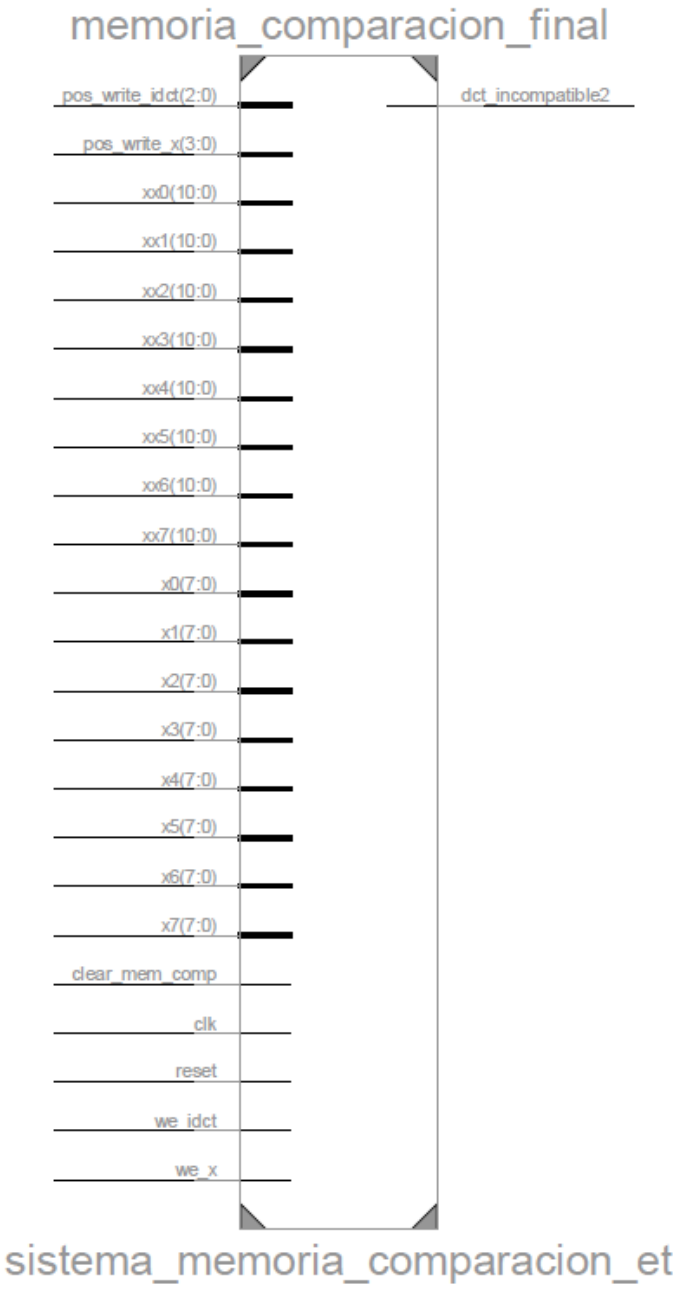


Figura 44. Memoria de comparación

- Entradas:
  - *pos_write_idct*: posición de escritura en memoria de los valores IDCT.
  - *pos_write_x*: posición de escritura en memoria del bloque de imagen original.
  - *xx0*, *xx1*, *xx2*, *xx3*, *xx4*, *xx5*, *xx6*, *xx7*: valores IDCT.
  - *x0*, *x1*, *x2*, *x3*, *x4*, *x5*, *x6*, *x7*: valores de imagen original.

- *clear_mem_comp*: reset síncrono.
- *clk*: señal de reloj.
- *reset*: reset asíncrono.
- *we_idct*: señal que habilita la escritura en memoria de los valores IDCT.
- *we_x*: señal que habilita la escritura en memoria de los valores de imagen original.

- Salidas:
  - *dct_incompatible2*: señal que se activa a nivel alto cuando el módulo detecta un bloque incompatible.

Esta memoria presenta dos tablas de almacenamiento: por un lado, los valores procedentes de la transformada inversa del coseno (IDCT) y, por otro lado; los valores originales del bloque de la imagen que se está estudiando y que fueron almacenados en memoria externa durante la lectura de la imagen (se corresponden con la imagen restando 128 a todos sus valores).

Cuando los 64 elementos correspondientes a cada tipo de bloque han sido almacenados en memoria, éstos se comparan, concluyendo con un bloque compatible en caso en que sean similares y con un bloque incompatible en caso en que sean distintos. Es importante tener en cuenta que la precisión del sistema es finita y que, para los dos bloques, pueden existir diferencias que no se deben tener en cuenta. Se habla de diferencias de unidades (por ejemplo, tener en un elemento el valor 70 y el elemento paralelo el valor 71) que no implican necesariamente la incompatibilidad del bloque. De ahí que se haya dejado un margen de ±3 unidades a la hora de comparar un bloque con otro, de forma que con el ejemplo que se acaba de exponer se supondrían ambos bloques iguales pero si se encontrara un caso, como por ejemplo un bloque con un elemento igual a 70 y su correspondiente elemento en el otro bloque con valor 75; se supondría la incompatibilidad del mismo. Este margen usado es completamente experimental, y se usa por los buenos resultados que ha dado en las distintas pruebas llevadas a cabo.

En el caso en que se detecte la incompatibilidad del bloque se activa la señal *dct_incompatible2* para que el controlador global sepa acerca de esa incompatibilidad y contabilice el bloque incompatible.

Capítulo 5

Resultados

En este capítulo se describen los resultados finales obtenidos a partir de la ejecución del algoritmo en MATLAB y de la ejecución, en simulación, del código hardware en VHDL. En primer lugar, en el apartado 5.1, se describe el funcionamiento de la herramienta empleada para la generación de estegoimágenes: SteganPEG. A continuación, en el apartado 5.2, se explican los resultados obtenidos con imágenes de diferentes tamaños y de diferentes orígenes: Internet e imágenes propias obtenidas con distintos dispositivos. Con ello se pretende investigar la sensibilidad del algoritmo a distintas imágenes y cómo afecta la naturaleza de la propia imagen al rendimiento del algoritmo. A partir de los resultados obtenidos se deducen una serie de conclusiones, de entre las cuales se debería destacar el umbral impuesto que nos permite, en principio, decidir si una imagen ha sido modificada o no. En el apartado 5.3 se sintetiza y se introduce el código VHDL sobre una FPGA, para confirmar que es posible introducir el código en un sistema real y que éste funcione, objetivo que en la práctica es lo que verdaderamente interesa. Para demostrar que el sistema hardware funciona y comprender mejor su funcionamiento, en el apartado 5.4 se muestra la simulación de una de las pruebas realizadas con una imagen real que previamente fue modificada. Aquí se pueden observar los puntos más importantes que aparecen en el análisis de una imagen. Finalmente, en el apartado 5.5, se comparan los tiempos de ejecución del algoritmo sobre software con los que se obtendrían de utilizar una FPGA, generando resultados muy interesantes desde el punto de vista práctico.

5.1 Herramienta de esteganografía: SteganPEG

A la hora de generar estegoimágenes que almacenaran información oculta fue necesario el uso de una herramienta de software que permitiese la generación de tales elementos. Para ello la herramienta que se empleó fue el software SteganPEG [17].

SteganPEG es un software desarrollado con el objetivo de ocultar información sobre imágenes que se encuentran en formato JPEG. Una de las ventajas fundamentales de este software es su capacidad para modificar, lo menos posible, el tamaño de la imagen y su calidad. A partir de las pruebas realizadas (de las cuales se hablará en el apartado 5.2) se puede afirmar que el programa cumple con sus objetivos de forma muy aceptable, ya que consigue introducir la información sin que el tamaño de la imagen sea modificado y sin que la calidad de la imagen se vea modificada en la mayor parte de las ocasiones, aunque en algunos casos sí que se detecte cierta borrosidad en la misma.

El proceso de esteganografía que realiza el programa (la forma de introducir la información en la imagen portadora) se divide en dos etapas fundamentales:

1) Encriptación de los datos usando un nuevo algoritmo de criptografía, "Rotatocrypt", que se describe en la documentación [17].

2) Método de esteganografía usando la técnica "Decodificación parcial de JPEG" sobre la imagen portadora para generar la estegoimagen de salida.

De ambas etapas en este Proyecto interesa conocer la segunda etapa; y dentro de esta segunda etapa, en la que se realiza un análisis de la clave, generación de variables y procesamiento de marcas y segmentos; es interesante conocer el método por el cual se introduce la información oculta a la imagen portadora. Para que el algoritmo desarrollado funcione es necesario que este método afecte directamente a los valores que presentan los píxeles en alguna de las etapas de la compresión JPEG, de forma que esa variación genere estructuras que no sean típicas del formato en cuestión y se puedan detectar tales incompatibilidades.

El método de introducción de la información generado en SteganPEG consiste en la modificación de los bits menos significativos de los bloques DCT en la imagen. Como se mostraba en la Figura 3, la DCT se aplica sobre bloques 8x8 de la imagen, resultando otra matriz 8x8 con un coeficiente DC (elemento(1,1)) y 63 coeficientes AC (el resto). El coeficiente DC no se modifica porque ello afectaría a la calidad de la imagen, por lo que el proceso de embeber la información se centra en los coeficientes AC de cada bloque. Al tratarse de valores DCT, ya no se trata con bytes cuyo valor varía entre 0 y 255; sino que el valor de los coeficientes puede variar mucho más de forma que cuanto mayor sea el valor absoluto del coeficiente más bits se requerirán para almacenarlo. Con el objetivo de no modificar el tamaño de la imagen, los coeficientes resultantes deben ser del mismo tamaño que los coeficientes originales. Por ello, el proceso de esteganografía se centra

únicamente en el bit menos significativo (LSB) de cada coeficiente AC. Así, de forma secuencial, se parte de todos los bits que se deben ocultar (en su conjunto constituyen el mensaje o la imagen que se va a introducir en la imagen portadora) y se recorren los LSB de los coeficientes AC de la imagen portadora: en caso en que bit de la portadora y bit a ocultar coincidan (ambos poseen valor '0' ó valor '1') se mantiene tal cual; en caso en que sean distintos se modifica el bit de la portadora. En el último caso existen dos posibilidades: o bien el bit a introducir es '0' y el bit de la imagen portadora es '1', por lo que el bit de la portadora pasaría a ser '0'; o bien el caso inverso, en el que el bit de la portadora pasaría a ser '1'. Esta forma de introducir información es similar a la que se mostraba en el Ejemplo 3, aunque la introducción se realice sobre los coeficientes DCT y no, directamente, sobre el valor de los píxeles.

Una vez comprendido el método de esteganografía usado por el programa se entiende que éste es compatible con el algoritmo desarrollado, ya que afecta directamente a los valores de los píxeles de la imagen (en su forma DCT) y a su compatibilidad con el formato JPEG.

Finalmente se muestra un ejemplo del uso del software en el que la imagen *gato.jpg* se introduce en la imagen portadora *lena.jpg* para generar la estegoimagen *lena_gato.jpg*:

Ejemplo 13: Uso de SteganPEG

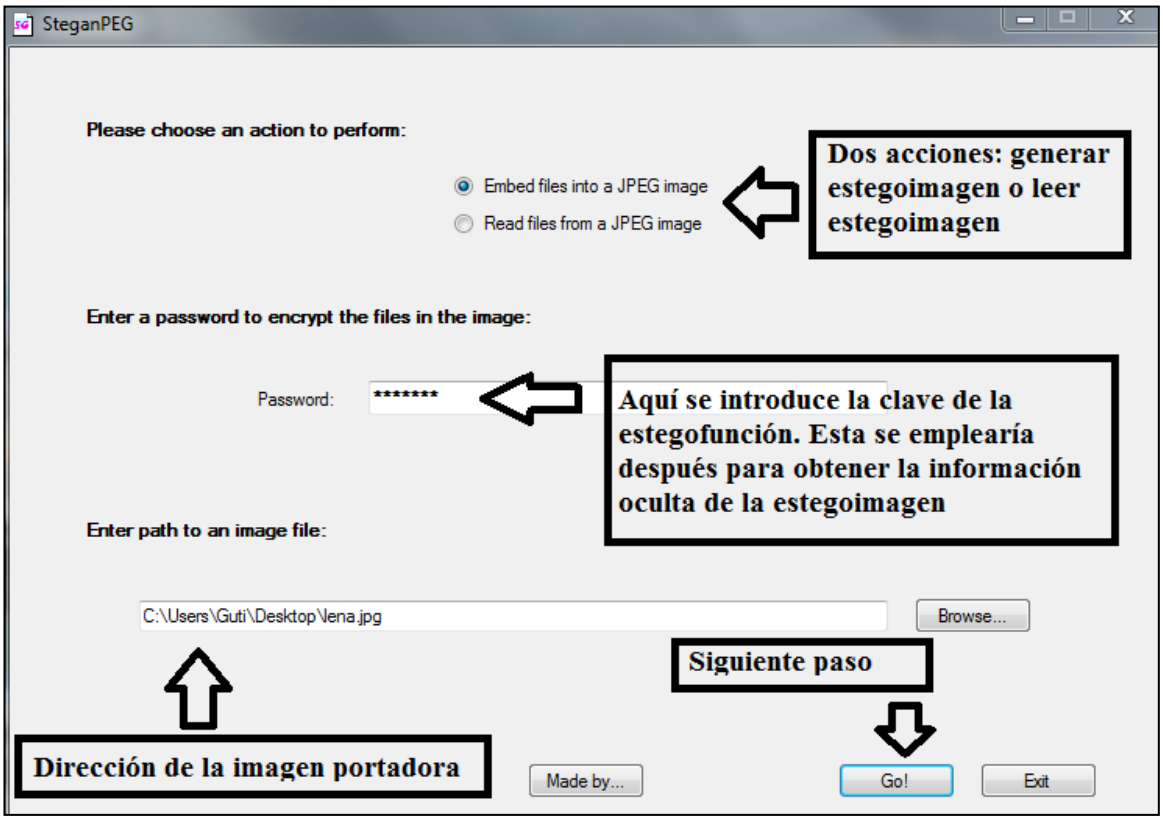


Figura 45. Paso I para generar una estegoimagen

Figura 46. Paso II para generar una estegoimagen

5.2 Funcionamiento

5.2.1 Pruebas realizadas sobre software

Primeramente, se realizaron las pruebas de funcionamiento sobre el algoritmo software que, como se especificó en el Capítulo 3, se trata de un algoritmo sin simplificar con un diagrama de flujo similar al descrito en [5] y [4]. Las pruebas realizadas y sus correpondientes resultados se especifican a continuación:

Prueba 1: Imagen no modificada: lena.jpg

Imagen original: lena.jpg

Tamaño de la imagen: 512 x 512

Número de bloques saturados: 0

Número de bloques analizados: 4096

Número de bloques incompatibles: 33

Porcentaje de imagen incompatible: 0.81%

La imagen no ha sido modificada, aún así se detectan algunos bloques incompatibles. Sin embargo el número de bloques incompatibles es muy pequeño, apareciendo debido a imprecisiones del método sobre todo a la hora de calcular la matriz de cuantificación Q. A lo largo de las pruebas realizadas, y tras comprobar que el algoritmo se ejecuta correctamente de acuerdo a lo especificado en los documentos de la bibliografía [4] y [5], se ha detectado que el método es muy sensible a los valores de la matriz Q, de forma que pequeñas variaciones en la misma cambian los resultados de forma significativa. Esto implica que imprecisiones en dicha matriz, incluso tan pequeñas como las unidades, alteran los resultados. Este fenómeno habrá que tenerlo en cuenta a la hora de estudiar las distintas imágenes estableciendo un umbral empírico por encima del cual se considerará que la imagen ha sido modificada. Este umbral se especificará como un porcentaje de la imagen incompatible con el formato JPEG.

Prueba 2: lena_gato.jpg

Imagen original: lena.jpg

+

Imagen oculta: gato.jpg

=

Imagen resultante: lena_gato.jpg

Tamaño de la imagen: 512 x 512

Número de bloques saturados: 0

Número de bloques analizados: 4096

Número de bloques incompatibles: 71

Porcentaje de imagen incompatible: 1.71%

Se ha tomado la imagen de la Prueba 1 y se ha introducido como mensaje oculto la imagen de menor tamaño *gato.jpg*. Se comprueba como el porcentaje de imagen incompatible ha aumentado debido a la imagen introducida, hasta alcanzar el 1.71%.

Prueba 3: mar.jpg

Imagen original: mar.jpg

Tamaño de la imagen: 780 x 1040

Número de bloques saturados: 118

Número de bloques analizados: 12492

Número de bloques incompatibles: 85

Porcentaje de imagen incompatible: 0.67 %

De la misma forma que en la Prueba 1 la imagen no está modificada, aún así se detectan ciertos bloques incompatibles que, al final, acaban dando un porcentaje de imagen incompatible menor del 1%. Lo que se pretendía con esta prueba era conocer la respuesta del algoritmo frente a diferentes tamaños de la imagen, ya que ésta imagen es aproximadamente el doble de grande que la de la Prueba 1. Se deduce que el comportamiento del algoritmo es independiente del tamaño de la imagen, dando porcentajes prácticamente similares.

Prueba 4: mar_gato.jpg

Imagen original: mar.jpg

Imagen oculta: gato.jpg

Imagen resultante: mar_gato.jpg

Tamaño de la imagen: 780 x 1040

Número de bloques saturados: 111

Número de bloques analizados: 12499

Número de bloques incompatibles: 222

Porcentaje de imagen incompatible: 1.76 %

De nuevo, se vuelve a ocultar la imagen *gato.jpg* sobre una imagen portadora, en este caso de mayor tamaño que la utilizada en la Prueba 2. Los resultados obtenidos son

iguales a los obtenidos anteriormente, generando un porcentaje de imagen incompatible del 1.7%.

Prueba 5: pueblo.jpg

Imagen original: pueblo.jpg

Tamaño de la imagen: 2448 x 3264

Número de bloques saturados: 1902

Número de bloques analizados: 122946

Número de bloques incompatibles: 0

Porcentaje de imagen incompatible: 0 %

Se comprueba el funcionamiento con una imagen de mucho mayor tamaño y en este caso el funcionamiento del algoritmo es perfecto, ya que no se detecta ningún bloque incompatible.

Prueba 6: pueblo_gato.jpg

Imagen original: pueblo.jpg

+

Imagen oculta: gato.jpg

=

Imagen resultante: pueblo_gato.jpg

Tamaño de la imagen: 2448 x 3264

Número de bloques saturados: 1902

Número de bloques analizados: 122946

Número de bloques incompatibles: 0

Porcentaje de imagen incompatible: 0 %

Se introduce una imagen oculta de mucho menor tamaño que la imagen original portadora. En este caso, no se detecta la presencia de la imagen oculta.

Prueba 7: pueblo_edimburgo.jpg

Imagen original: pueblo.jpg

+

Imagen oculta: edimburgo.jpg

=

Imagen resultante: pueblo_edimburgo.jpg

Tamaño de la imagen: 2448 x 3264

Número de bloques saturados: 2166

Número de bloques analizados: 122682

Número de bloques incompatibles: 5445

Porcentaje de imagen incompatible: 4.36 %

Se decide introducir en la imagen de la Prueba 5 otra imagen de mayor tamaño que la empleada en la Prueba 6. En este caso los resultados obtenidos son mejores que los anteriores, ya que se obtiene un elevado porcentaje de imagen modificada.

Teniendo en cuenta los resultados de la Prueba 6 y la Prueba 7, se deduce una relación entre el rendimiento del algoritmo y el tamaño de la imagen a introducir. El rendimiento es elevado cuanto mayor sea el porcentaje imagen modificada así como cuanto más similares sean los tamaños de ambas imágenes, obteniendo rendimientos bajos en caso de introducir pocos datos (imágenes pequeñas) sobre imágenes muy grandes.

Prueba 8: tienda.jpg

Imagen original: tienda.jpg

Tamaño de la imagen: 1920 x 2560

Número de bloques saturados: 805

Número de bloques analizados: 75995

Número de bloques incompatibles: 618

Porcentaje de imagen incompatible: 0.805 %

Con esta imagen no modificada se obtienen resultados muy similares a los de la Prueba 3, con mayor número de bloques incompatibles por ser una imagen de mayor tamaño pero con un porcentaje total de incompatibilidad muy escaso: 0.805%.

Prueba 9: tienda_gato.jpg

Imagen original: tienda.jpg

Imagen oculta: gato.jpg

Imagen resultante: tienda_gato.jpg

Tamaño de la imagen: 1920 x 2560

Número de bloques saturados: 790

Número de bloques analizados: 76010

Número de bloques incompatibles: 724

Porcentaje de imagen incompatible: 0.943 %

De nuevo, al igual que en la Prueba 6, se introduce una imagen muy pequeña (*gato.jpg*) en una imagen de mucho mayor tamaño (*tienda.jpg*). Sin embargo, en este

caso, sí que se detecta la información oculta pasando de un porcentaje de incompatibilidad del 0.805% de la Prueba 8 al 0.943% obtenido en esta Prueba 9.

Prueba 10: tienda_edimburgo.jpg

Imagen original: tienda.jpg

+

Imagen oculta: edimburgo.jpg

=

Imagen resultante: tienda_edimburgo.jpg

Tamaño de la imagen: 1920 x 2560

Número de bloques saturados: 742

Número de bloques analizados: 76058

Número de bloques incompatibles: 4252

Porcentaje de imagen incompatible: 5.53 %

De nuevo, como en la Prueba 7, la información oculta se detecta de forma muy evidente.

Prueba 11: mono.bmp

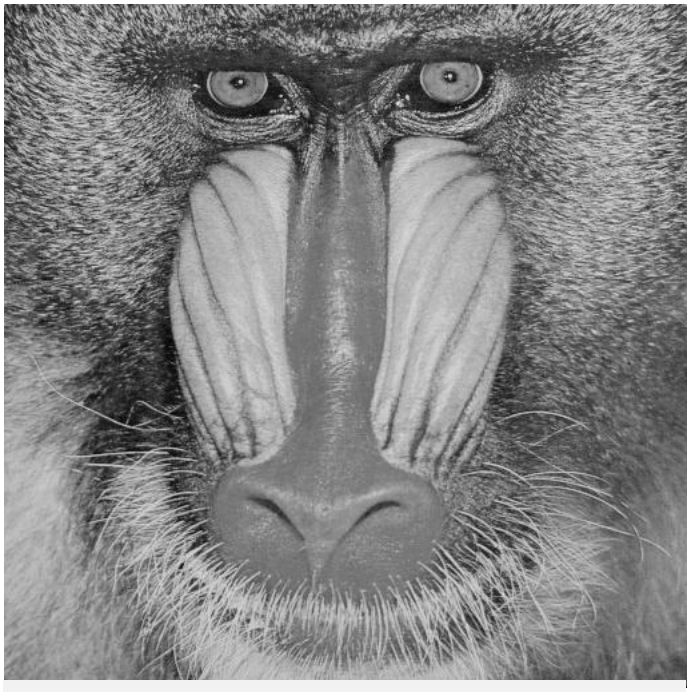


Imagen original en formato BMP

Con esta imagen se aplica el algoritmo y se obtiene la siguiente matriz de cuantificación:

Matriz 18: Matriz de cuantificación para *mono.bmp*

Q =

1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

Esta matriz es unitaria, por lo que según los principios del algoritmo en los que se basa el sistema de estegoanálisis [5] la imagen no procede de un formato JPEG y, por lo tanto, el algoritmo no se puede aplicar para imágenes almacenadas en formato BMP.

Prueba 12: gato_png.png

Imagen en formato PNG

Igual que en la Prueba 11 la matriz de cuantificación Q es unitaria, por lo que el algoritmo no es aplicable a imágenes en formato PNG.

5.2.2 Pruebas desarrolladas sobre hardware

Las pruebas que se describen a continuación se han realizado sobre el código final en VHDL utilizando la herramienta de simulación *ModelSim SE Plus 6.2c*. Para ello se empleó la capacidad de VHDL de trabajar con ficheros para que éstos hicieran las veces de memoria externa. Las pruebas desarrolladas se han realizado con las mismas imágenes empleadas en las pruebas software sobre MATLAB.

Los resultados obtenidos fueron los siguientes:

Prueba 1: Imagen no modificada: lena.jpg

Número de bloques analizados: 4096

Número de bloques incompatibles: 33

Porcentaje de imagen incompatible: 0.81%

Resultado similar al de la prueba en software.

Prueba 2: lena_gato.jpg

Número de bloques analizados: 4096

Número de bloques incompatibles: 72

Porcentaje de imagen incompatible: 1.76%

En este caso, se obtiene un bloque incompatible más que en el caso del software. Este bloque se detecta por cuestiones de precisión, a la hora de analizar el bloque número 1682 el código software genera un valor de S igual a 16, por lo que el bloque se considera como compatible. Sin embargo, el código VHDL, al realizar operaciones con mayores redondeos, genera un valor de 17 en lugar de 16, lo que implica que el bloque es

considerado incompatible. Aún así se puede observar cómo el porcentaje final de incompatibilidad apenas se ve modificado.

Prueba 3: mar.jpg

Número de bloques analizados: 12492

Número de bloques incompatibles: 84

Porcentaje de imagen incompatible: 0.67 %

En este caso ocurre algo similar al caso de la Prueba 2. En el bloque 2570 el código software obtiene un valor de S igual a 17, por lo que considera bloque incompatible; mientras que el código hardware obtiene un valor igual a 16, por lo que no puede considerar la incompatibilidad del bloque. A pesar de ello, el porcentaje no se ve afectado de forma significativa.

Prueba 4: mar_gato.jpg

Número de bloques analizados: 12499

Número de bloques incompatibles: 223

Porcentaje de imagen incompatible: 1.77 %

En las pruebas software se obtenían 222 bloques incompatibles, el bloque de más que aparece en este análisis se debe al bloque número 1149 que en la ejecución software presenta el valor 16 mientras que la ejecución hardware presenta el valor 17.

Prueba 11: mono.bmp y Prueba 12: gato_png.png

Para estos casos, igual que en las pruebas software, se obtiene una matriz de cuantificación Q unitaria, por lo que el algoritmo no es aplicable.

5.3 Área

A la hora de sintetizar el código completo fue necesario buscar FPGAs con dos características importantes:

- Por un lado era conveniente trabajar con FPGAs que poseyeran multiplicadores que permitieran implementar los módulos de cálculo de la DCT (apartado 4.3.3) y de la IDCT (apartado 4.3.6) sin usar lógica tradicional, ya que ello supondrían un consumo de recursos excesivo.
- Por otro lado, el sistema se caracteriza por la gran cantidad de entradas y de salidas necesarias, debido sobretodo a las comunicaciones con las memorias externas de las que hace uso. Se necesitan en total 545 pines de entrada/salida, y no todas las FPGAs poseen tal cantidad de pines. De ahí la necesidad de buscar una que satisfaga esta condición.

Teniendo en cuenta estas restricciones, se estudia, por ejemplo, la posibilidad de usar una FPGA de la familia Virtex 6 de Xilinx. A partir de las condiciones se selecciona una FPGA de dicha familia:

Device	Logic Cells	Configurable Logic Blocks (CLBs)		DSP48E1 Slices[2]	Block RAM Blocks			MMCMs[4]	Interface Blocks for PCI Express	Ethernet MACs[5]	Maximum Transceivers		Total I/O Banks[6]	Max User I/O[7]
		Slices[1]	Max Distributed RAM (Kb)		18 Kb[3]	36 Kb	Max (Kb)				GTX	GTH		
XC6VLX75T	74,496	11,640	1,045	288	312	156	5,616	6	1	4	12	0	9	360
XC6VLX130T	128,000	20,000	1,740	480	528	264	9,504	10	2	4	20	0	15	600
XC6VLX195T	199,680	31,200	3,040	640	688	344	12,384	10	2	4	20	0	15	600
XC6VLX240T	241,152	37,680	3,650	768	832	416	14,976	12	2	4	24	0	18	720
XC6VLX365T	364,032	56,880	4,130	576	832	416	14,976	12	2	4	24	0	18	720
XC6VLX550T	549,888	85,920	6,200	864	1,264	632	22,752	18	2	4	36	0	30	1200
XC6VLX760	758,784	118,560	8,280	864	1,440	720	25,920	18	0	0	0	0	30	1200
XC6VSX315T	314,880	49,200	5,090	1,344	1,408	704	25,344	12	2	4	24	0	18	720
XC6VSX475T	476,160	74,400	7,640	2,016	2,128	1,064	38,304	18	2	4	36	0	21	840
XC6VHX250T	251,904	39,360	3,040	576	1,008	504	18,144	12	4	4	48	0	8	320
XC6VHX255T	253,440	39,600	3,050	576	1,032	516	18,576	12	2	2	24	24	12	480
XC6VHX380T	382,464	59,760	4,570	864	1,536	768	27,648	18	4	4	48	24	18	720
XC6VHX565T	566,784	88,560	6,370	864	1,824	912	32,832	18	4	4	48	24	18	720

Figura 47: FPGAs de la familia Virtex 6

Se selecciona el modelo *Virtex 6 XC6VLX130T, FF1156*. Con esta FPGA se realiza la síntesis y la implementación sobre el dispositivo en el software *Xilinx ISE Design Suite 12.2* atendiendo a dos criterios: por un lado se implementa para optimizar la velocidad del sistema, y por otro lado para optimizar el área ocupada por el sistema.

Para la optimización de la velocidad se obtienen los resultados de lógica usada expresados en la Figura 48:

	Usados	**Disponibles**	**Utilización (%)**
Registros	6360	160000	3
LUTs	14720	80000	18
Pines I/O	545	600	90
DSP	65	480	13

Figura 48: Resultados de implementación optimizados en velocidad

Se puede observar que el diseño se puede implementar en la FPGA seleccionada, ocupando recursos relativamente escasos: 3% de los registros disponibles, 18% de LUT, 13% de los multiplicadores DSP y el 90% de los pines de entrada/salida.

Para este caso la frecuencia máxima de funcionamiento obtenida es de 48.2 MHz.

Para la optimización en área, se obtienen los resultados de la Figura 49:

	Usados	Disponibles	Utilización (%)
Registros	6212	160000	3
LUTs	14901	80000	18
Pines I/O	545	600	90
DSP	65	480	13

Figura 49: Resultados de implementación optimizados en área

De nuevo los recursos empleados son escasos si se comparan con los que ofrece la FPGA y, además, son muy similares a los que se obtuvieron en la optimización en velocidad. El principal objetivo del diseño es la velocidad por lo que, a la hora de implementar el sistema sobre un dispositivo hardware, se utilizará siempre la optimización en velocidad.

La frecuencia de funcionamiento máxima para la optimización en área es de 46.8 MHz, algo menor a la obtenida para la optimización en velocidad.

5.4 Simulaciones

La simulación que se presenta a continuación tiene como entrada la estegoimagen correspondiente a la Prueba 2 del apartado 5.2.2, cuyos resultados sobre hardware fueron:

Tamaño de la imagen: 512 x 512

Número de bloques saturados: 0

Número de bloques analizados: 4096

Número de bloques incompatibles: 72

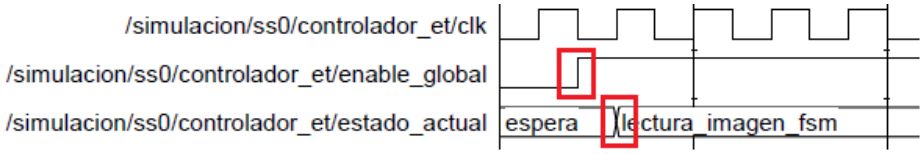
Porcentaje de imagen incompatible: 1.76%

Los puntos que se siguen en simulación son los mismos que ya se describieron en la Figura 14 del apartado 4.2.

A lo largo de la explicación se irán describiendo los puntos más importantes de cada fase, de forma que se observe directamente el funcionamiento del sistema y se comprenda mejor su funcionamiento:

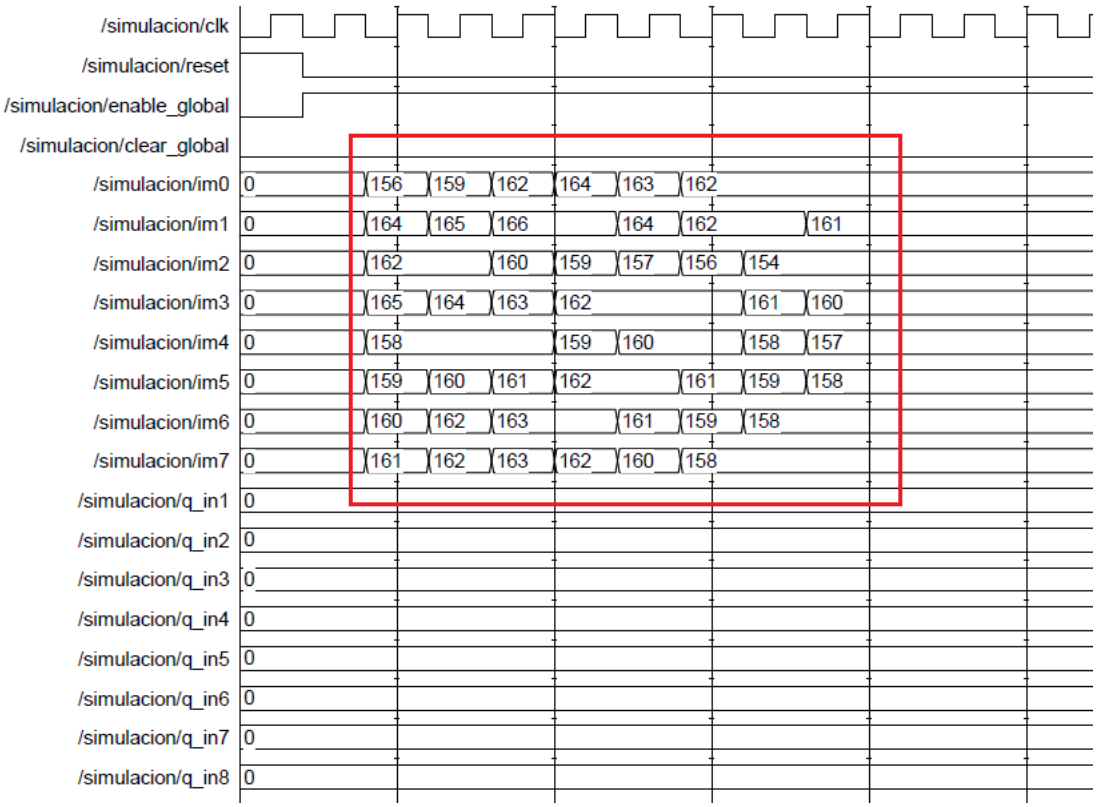
1) El sistema se encuentra en estado de reposo (*espera*) hasta que se activa la señal *enable_global*. En ese momento se pasa a la fase de lectura de la imagen (*lectura_imagen_fsm*).

Simulación 1



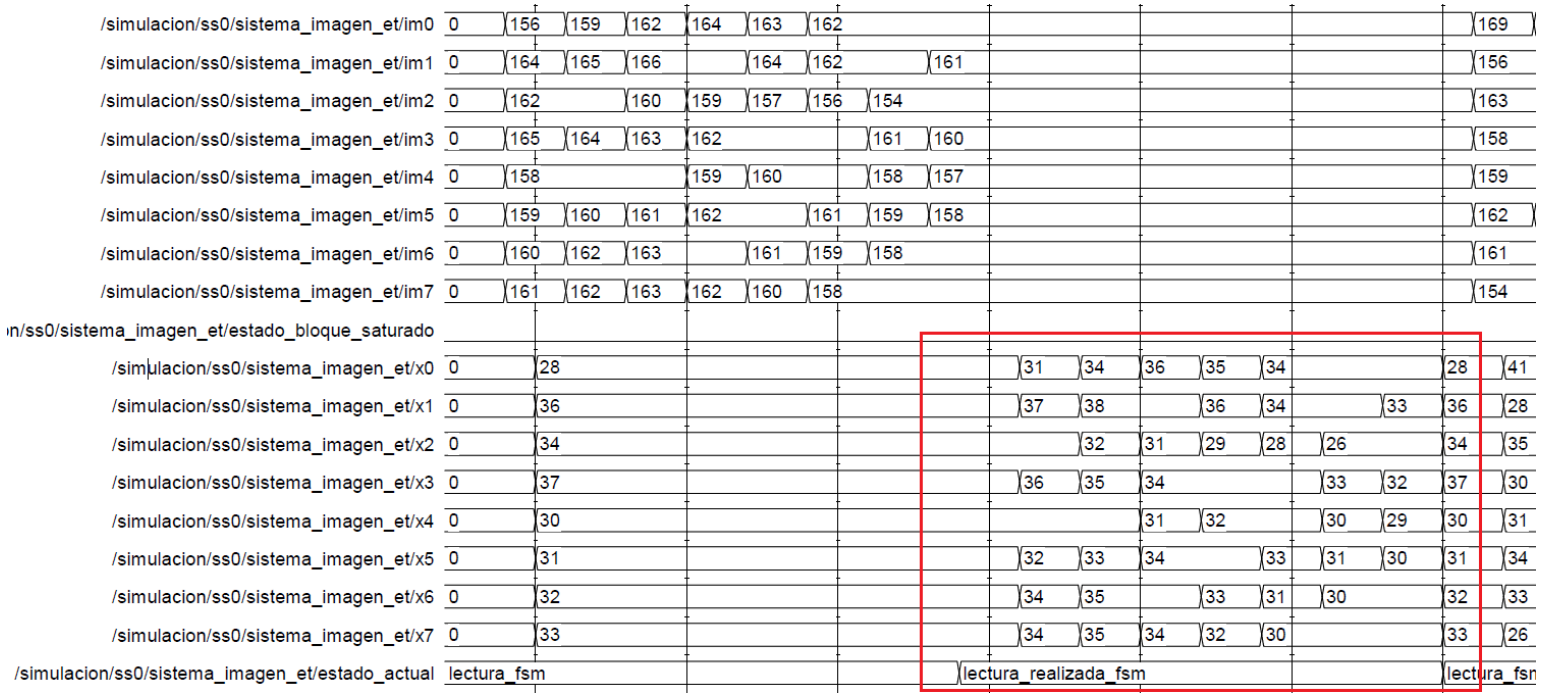
2) Se comienzan a introducir al sistema los valores de los píxeles de la imagen que se desea a analizar. Los valores se introducen de 8 en 8, modificándose a cada ciclo de reloj. En Simulación 2, se puede observar cómo se introduce el primer bloque de la imagen.

Simulación 2



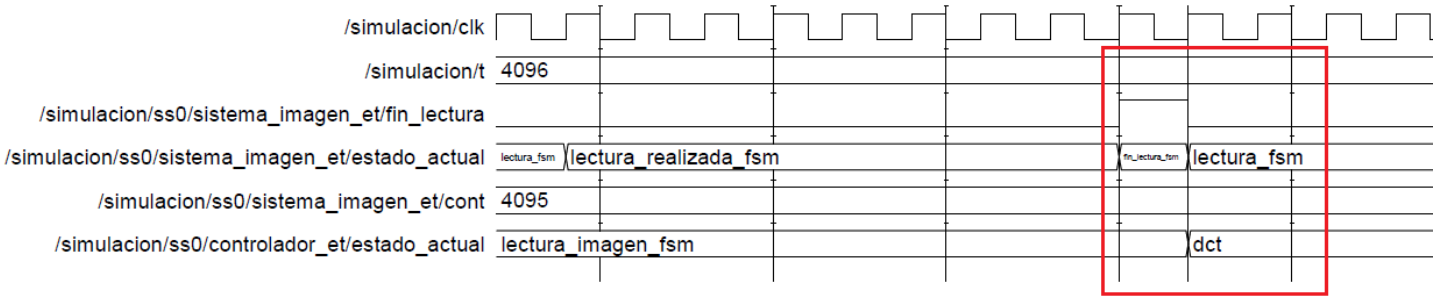
3) Tras haber leído completamente el bloque de 64 elementos, en caso en que no esté saturado, éste se almacena en memoria externa con su valor menos 128. Durante la lectura de la imagen el módulo *sistema_imagen* se encuentra en el estado *lectura_fsm*. En el momento en que el bloque no saturado se almacena en memoria externa el módulo presenta el estado *lectura_realizada_fsm*.

Simulación 3



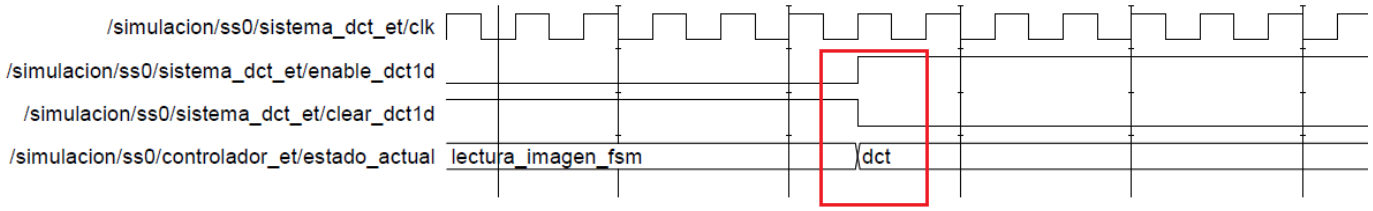
4) Las etapas observadas en 2) y 3) se repiten a lo largo de todos los bloques que presenta la imagen. El sistema conoce el número de bloques que debe analizar gracias a la entrada *T*. A través de un contador interno se numeran el número de bloques analizados, en el momento en que se analiza el último, el módulo *sistema_imagen* pasa al estado *fin_lectura* y el sistema global al estado *dct*.

Simulación 4



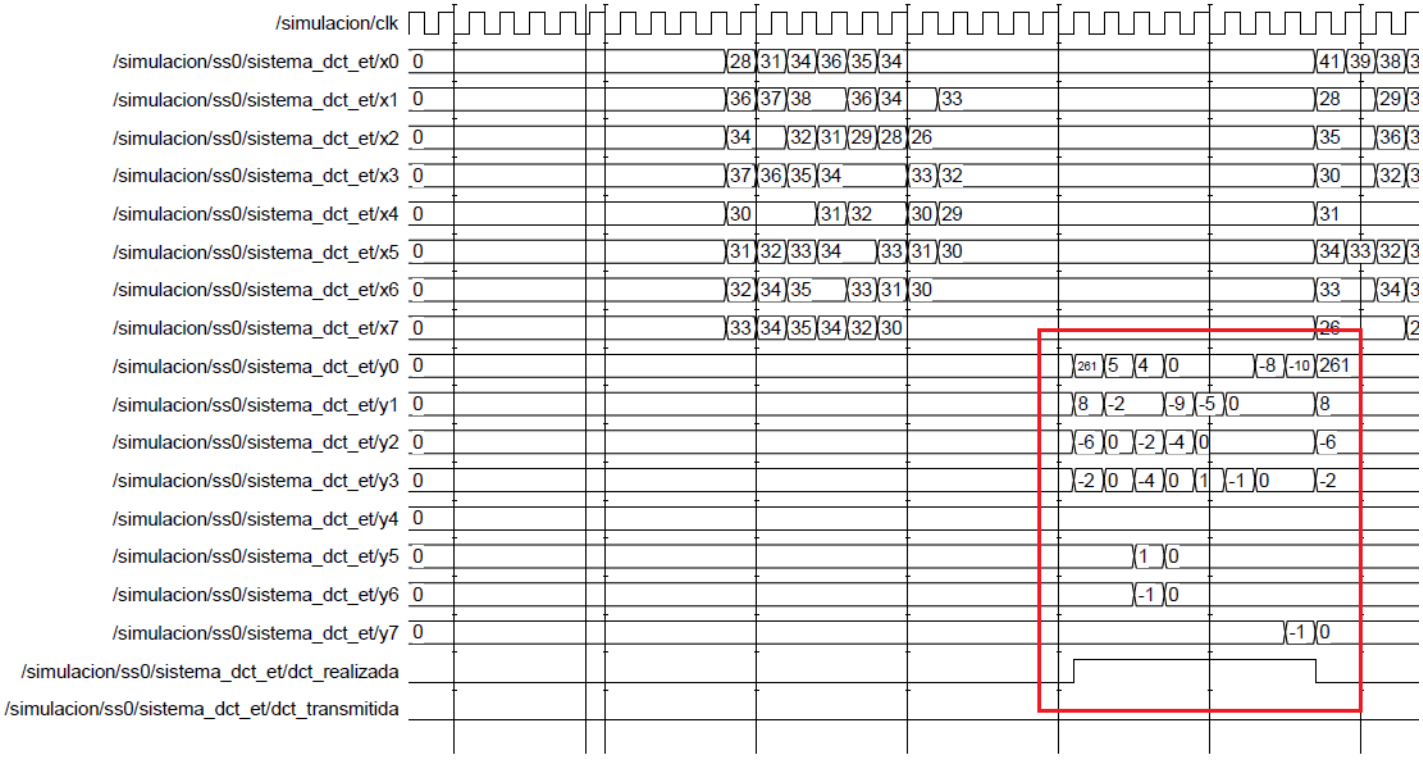
5) A continuación, comienza la fase de cálculo DCT. Se toman los bloques almacenados en memoria y se calcula la DCT de cada uno de estos bloques. En primer lugar se activa la señal de habilitación del bloque *sistema_dct.*

Simulación 5



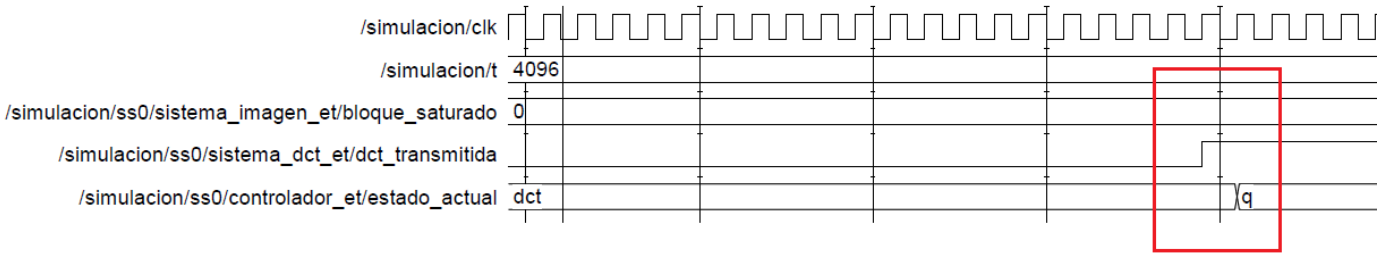
6) Una vez activado el bloque *sistema_dct*, se toman los valores de los bloques y se calcula su DCT. El módulo tarda 19 ciclos de reloj en dar el resultado completo. Este resultado se ofrece de 8 valores en 8 valores, activando la señal *dct_realizada* cuando los valores finales aparecen en salida. Al mismo tiempo los resultados se almacenan en memoria externa.

Simulación 6



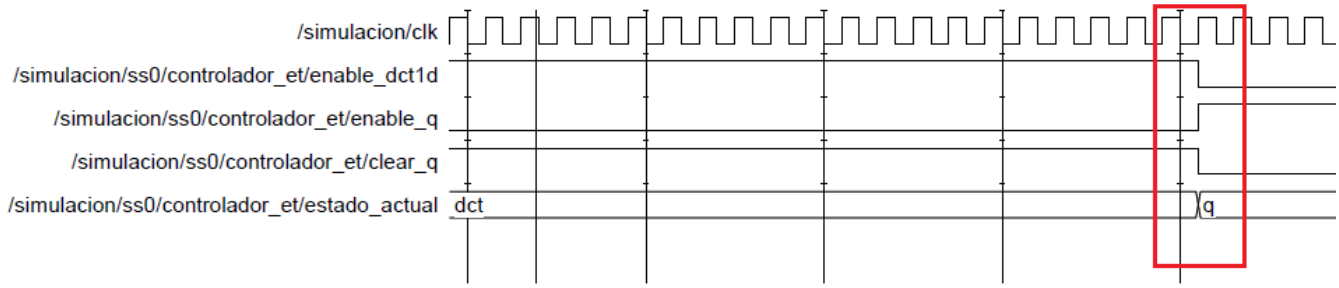
7) Se calcula la DCT de todos y cada uno de los bloques. Cuando se calcula la DCT del último bloque se activa la señal *dct_transmitida* y se pasa a la siguiente fase del cálculo: la matriz de cuantificación Q. La señal *bloque_saturado* almacena el número de bloques saturados, con dicha señal y la señal de entrada *T* el sistema conoce el último bloque del que se debe calcular la DCT.

Simulación 7



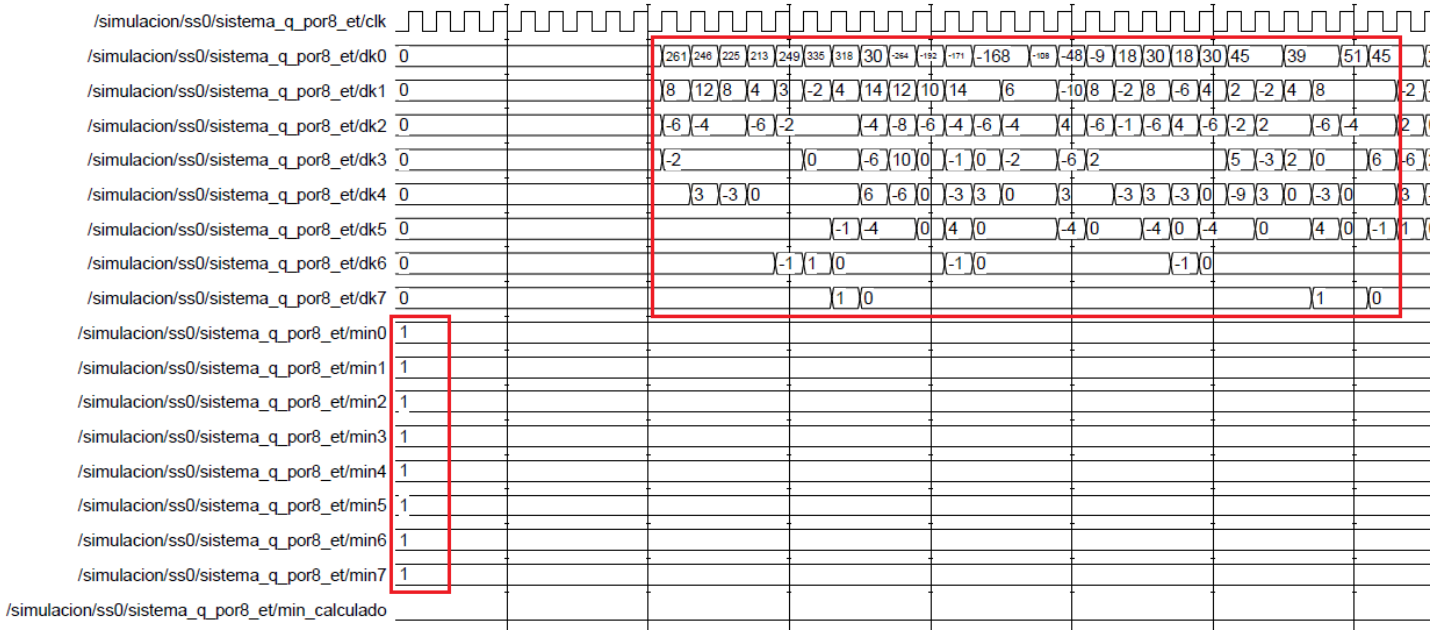
8) De igual manera al resto de los casos, al comienzo de la fase de cálculo de la matriz Q se habilita el módulo correspondiente a través de la señal *enable_q*.

Simulación 8



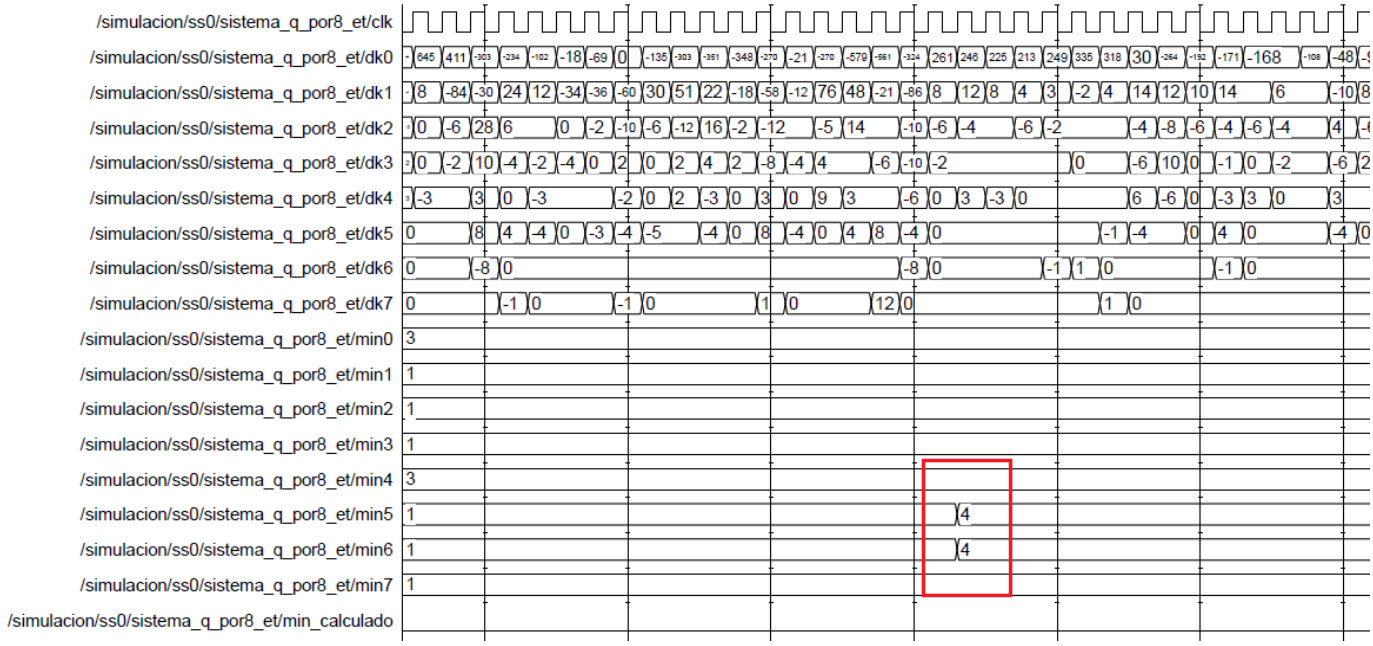
9) El cálculo de la matriz de cuantificación Q se realiza de 8 valores en 8 valores, por lo que son necesarias 8 etapas de cálculo. Los valores necesarios para la obtención de cada uno de los elementos aparecen en las entradas *dk* y el cálculo siempre dura el mismo tiempo ya que se prueban valores de *q* desde 1 hasta 75 (para mayor detalle del proceso de cálculo consultar el apartado 2.3 y 4.3.4). En Simulación 9 se puede observar como en cada ciclo de reloj se introducen valores nuevos para el cálculo y como, al principio del cálculo, el valor de *q* es 1.

Simulación 9



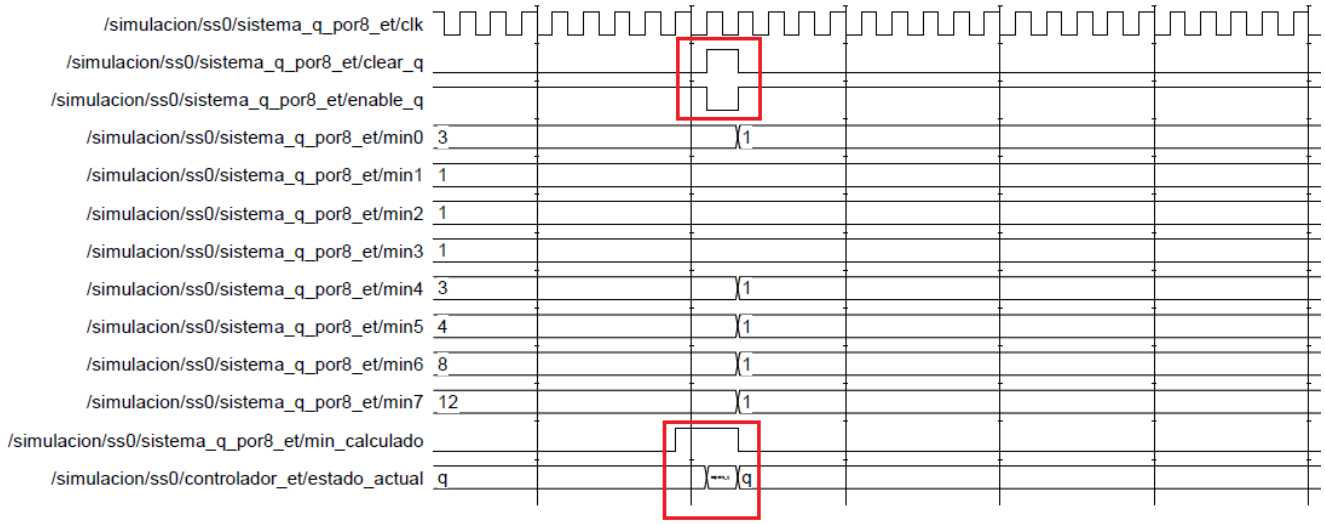
10) En el momento en que se encuentra un mínimo que se considera correcto se actualiza el valor *min*.

Simulación 10



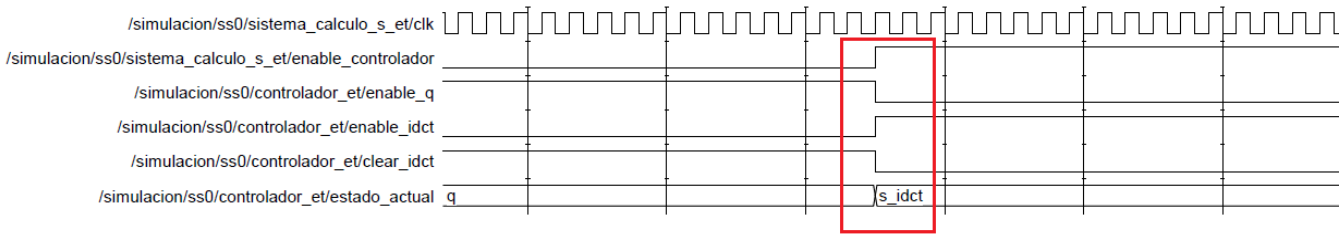
11) Cuando se analizan los posibles 75 valores de *q*, el cálculo finaliza. Para comunicar la finalización se activa la señal *min_calculado*. Con ello el sistema se inicializa a través de la señal *clear_q*, los valores *min* se colocan con valor 1 y el sistema pasa por un estado de transición llamado *espera_q* antes de volver a *q* y seguir con el cálculo. Los valores de *q* son almacenados en memoria externa.

Simulación 11



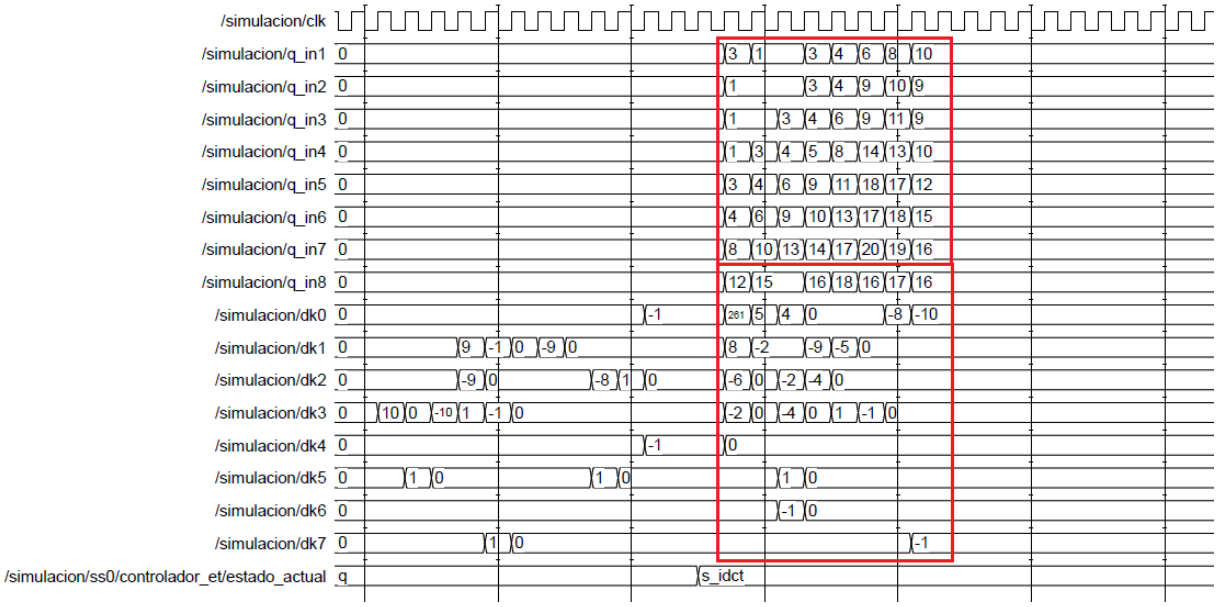
12) Los pasos descritos en 9), 10) y 11) se repiten en 8 ocasiones para calcular los 64 elementos de la matriz de cuantificación. Cuando finaliza el cálculo, se pasa a la fase de análisis donde se calculará el valor S para cada bloque y la IDCT en caso en que sea necesario. En este punto el estado del sistema global pasa de *q* a *s_idct*.

Simulación 12



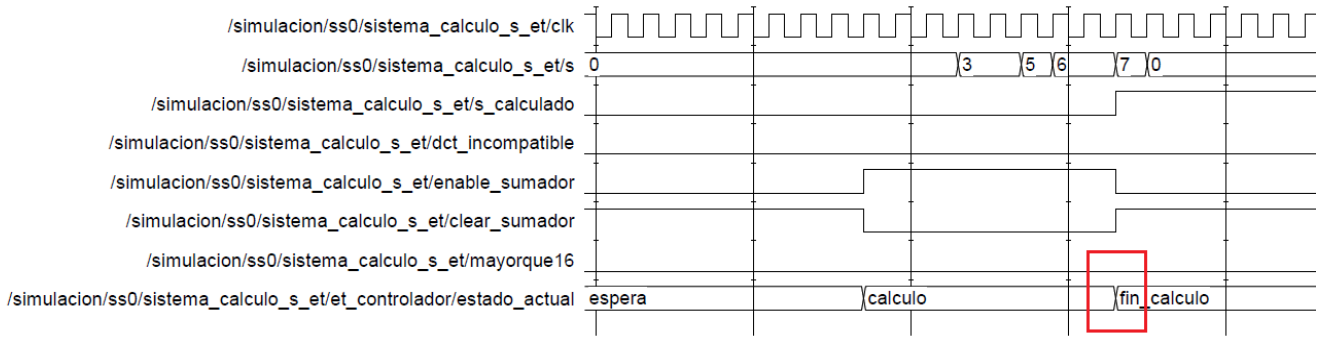
13) Durante esta fase de análisis, para cada bloque, en primer lugar se calcula el valor S. Para ello se introducen al sistema el bloque correspondiente y la matriz Q. Esta introducción se realiza a través de las entradas *dk* y *q_in*.

Simulación 13



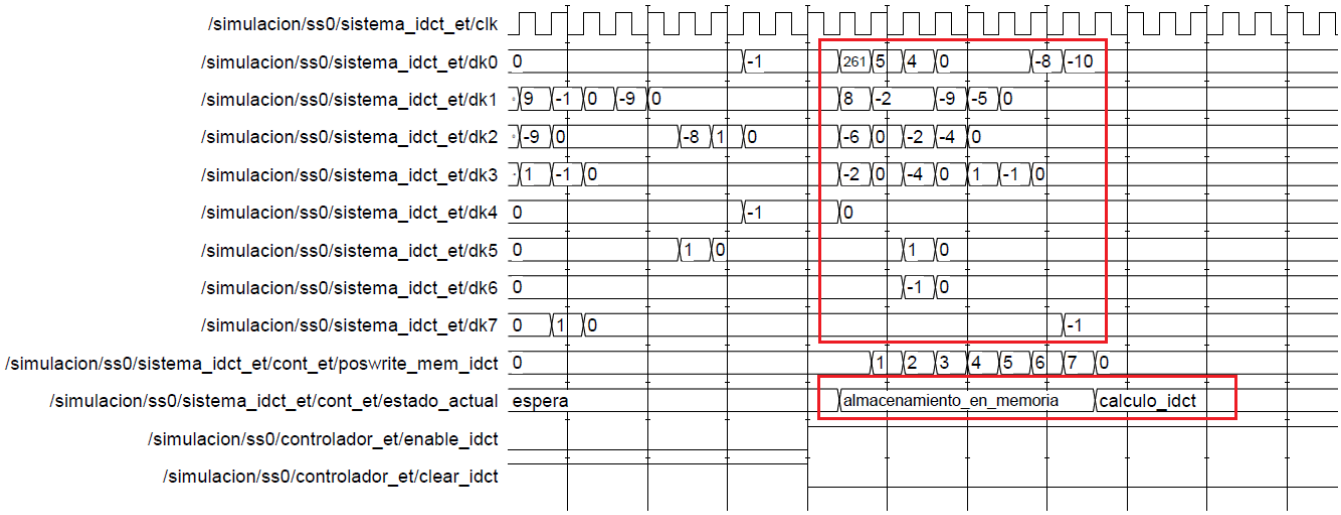
14) Al mismo tiempo de la introducción se calcula el valor S, de forma que al introducir el último grupo de valores *dk* y *q_in* ya se tiene el parámetro S. En ese momento el módulo *sistema_calculo_s* pasa del estado *calculo* al estado *fin_calculo*. En este caso S tiene valor 7 (menor que 16), por lo que todavía no se sabe si el bloque es compatible o incompatible. Para saberlo se necesita realizar la prueba de la IDCT.

Simulación 14



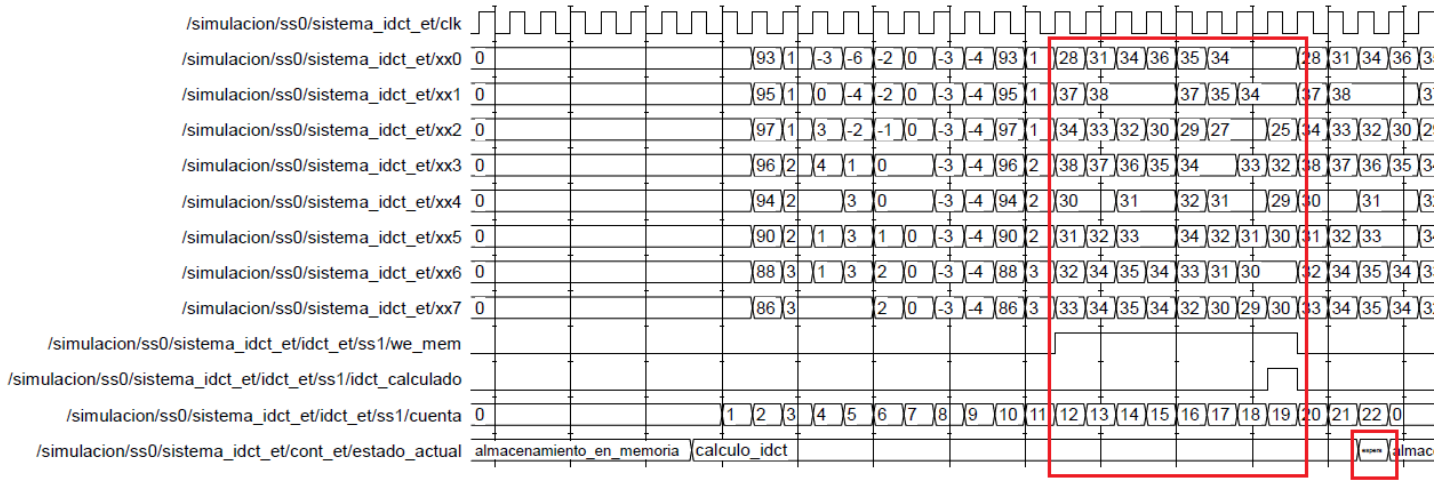
15) Se realiza la IDCT a través del módulo *sistema_idct*, introduciendo los valores a través de las entradas *dk*. Primeramente el sistema almacena los valores correspondientes en memoria interna (estado *almacenamiento*) para posteriormente utilizarlos en el cálculo IDCT (estado *calculo_idct*).

Simulación 15



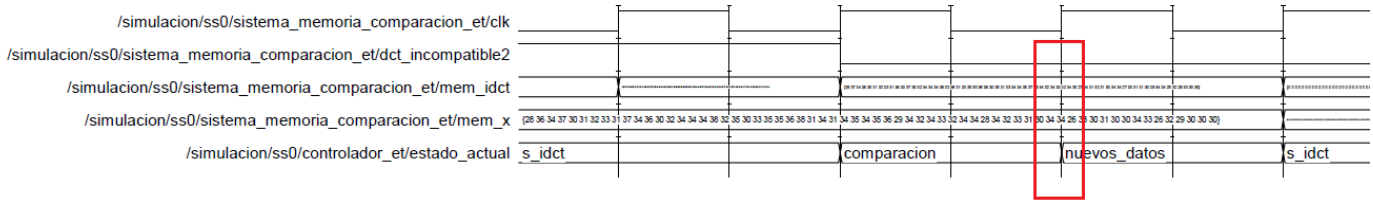
16) El cálculo total de la IDCT dura 19 ciclos de reloj, contabilizados a través de la variable interna *cuenta*. A través de las salidas *xx* se muestran los resultados, siendo los resultados correctos cuando la señal *we_mem* está activada a nivel alto (esta señal también sirve para activar la escritura de los valores IDCT en la memoria que se encarga de realizar la comparación entre la IDCT y el bloque original). Finalmente, cuando el sistema muestra el último grupo de valores correctos, se activa durante un ciclo de reloj la señal *idct_calculado*, el sistema pasa por un estado de transición *espera* para, a continuación, continuar con el cálculo IDCT de un nuevo bloque.

Simulación 16



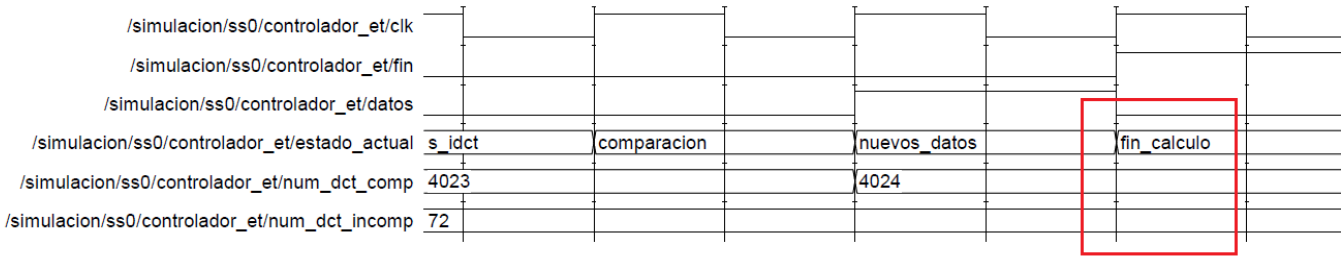
17) El último paso en la fase de análisis consiste en la comparación entre el bloque IDCT obtenido anteriormente y el bloque original de la imagen. En el módulo *memoria_comparacion_final* se almacenan ambos bloques y se comparan: si son aproximadamente similares son compatibles, por el contrario son incompatibles y se activa la señal *dct_incompatible2*. Para realizar la comparación el sistema global pasa del estado *s_idct* al estado *comparacion*, para luego pasar al estado *nuevos_datos*, que indica que el sistema pasa a analizar un nuevo bloque. En la Simulación 17 se han comparado dos bloques similares, por ello la señal *dct_incompatible2* se mantiene a nivel bajo y el bloque analizado se considera compatible.

Simulación 17



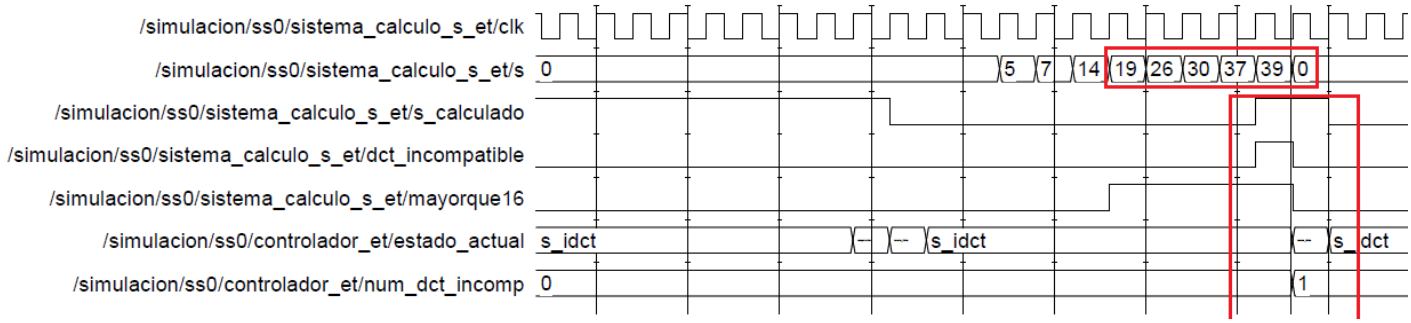
18) Los pasos descritos desde 13) se repetirían para todos los bloques que se deben analizar. En el caso en que se encuentren bloques incompatibles, éstos se contabilizan y al final del análisis se conocería el número de bloques incompatibles que presenta la imagen. El estado que alcanza el sistema al final del análisis es *fin_calculo*. Se puede observar a continuación que el análisis aporta como resultados, 4024 bloques compatibles y 72 bloques incompatibles.

Simulación 18



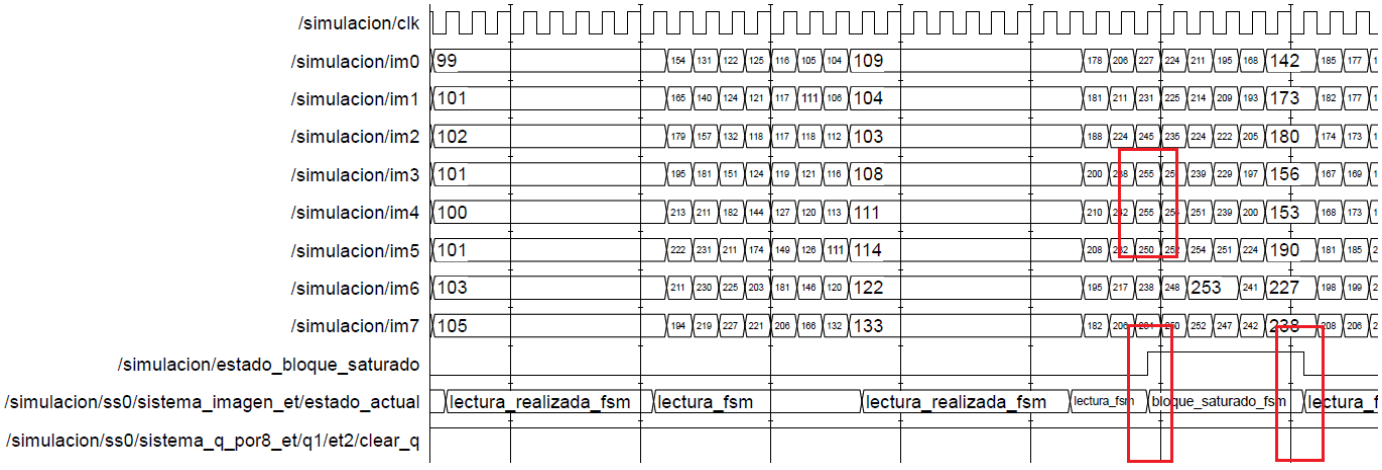
19) Un ejemplo en el que se detecta un bloque incompatible por valor de S se muestra en la siguiente figura. Se observa que en el proceso de cálculo de S llega un momento en que S es mayor que 16, lo que ya implica un bloque incompatible. Se activa la señal *mayorque16*, y cuando finaliza el cálculo de S, se activa la señal *dct_incompatible*; el sistema pasa al estado *nuevos_datos* y se contabiliza el bloque incompatible mediante *num_dct_incomp*.

Simulación 19



20) Para terminar se muestra un ejemplo de detección de un bloque saturado. Cuando se detecta un bloque con alguno de sus elementos saturado el sistema entra en el estado *bloque_saturado_fsm* y no almacena el bloque leído. El sistema espera en este último estado hasta que el bloque sea leído por completo y, a continuación, pasa a la lectura del siguiente bloque. Mientras el sistema se encuentra en el estado *bloque_saturado_fsm*, la salida *estado_bloque_saturado* se mantiene activa a nivel alto.

Simulación 20



5.5 Tiempos

Para el análisis y la comparación de los tiempos de ejecución entre el código software y el código hardware se ha utilizado la imagen *lena_gato.jpg*, la misma utilizada para la Prueba 2 y para la simulación del apartado 5.4

En el caso de la medición del tiempo de ejecución para el código de MATLAB se ha usado la función de dos elementos: *tic toc*, cuyo funcionamiento se puede estudiar en [11]. A grandes rasgos mide el tiempo que transcurre desde que se ejecuta *tic* hasta que se ejecuta *toc*, de forma que, para medir el tiempo completo, es necesario que este par de funciones engloben tanto el proceso de lectura de la imagen como el proceso de análisis, es decir, las mismas acciones que posteriomente ejecutará la versión hardware. Para obtener una estimación aceptable del tiempo de ejecución en software se ejecutará el algoritmo en 50 ocasiones y el tiempo total medido se dividirá entre 50. De esta forma se obtiene un tiempo medio orientativo que se puede utilizar para alcanzar conclusiones, penalizando las ejecuciones más lentas y las más rápidas (hay que tener en cuenta que el ordenador empleado no se dedica íntegramente a ejecutar el algoritmo, sino que en cada instante realiza distintas acciones que pueden afectar al tiempo de ejecución).

Empleando un ordenador con procesador Intel i7 y sistema operativo Windows 7 se obtienen los valores del Resultado 1.

Resultado 1. Tiempo de ejecución de lena_gato.jpg en MATLAB

```
Elapsed time is 162.008761 seconds.
>> 162.008761/50

ans =

    3.2402
```

El tiempo medio de ejecución es de 3.2402 seg.

Al realizar la simulación sobre hardware se obtiene que la imagen es leída y analizada tras 2697845 ciclos de reloj (momento en que se alcanza el estado *fin_calculo*). Tomando la máxima frecuencia de funcionamiento obtenida en el apartado 5.3 es de 48.2 MHz, lo que supone un periodo de ciclo de reloj igual a 20.74 ns. Por tanto el tiempo de ejecución total en hardware es de:

Resultado 2. Tiempo de ejecución de lena_gato.jpg en hardware

$$20.74 \cdot 10^{-9} \left(\frac{\text{seg}}{\text{ciclo}}\right) \cdot 2697845 \text{ ciclos} = 0.05597 \text{ seg} \cong 56 \text{ ms}$$

A partir del Resultado 1 y del Resultado 2 se puede obtener el ratio R.

Resultado 3. Ratio R (tiempo ejecución software/tiempo ejecución hardware) para lena_gato.jpg

$$R = \frac{3.2402}{0.05597} = 57.89$$

Lo que quiere decir que la versión ejecutada en hardware es capaz de analizar 57 imágenes (se toma el valor por defecto) en el mismo tiempo en que tarda la versión de MATLAB en analizar una imagen. Este resultado es teórico y hay que comprender su significado, ya que no se están teniendo en cuenta los tiempos involucrados de los que la FPGA empleada se vale para acceder a las posiciones de la memoria externa usada durante el análisis. Para la obtención de resultados reales sería necesario decidir qué FPGA y qué memoria se va a utilizar y estudiar los tiempos que va a utilizar el controlador de memoria para acceder a dicha memoria. Aún así los resultados obtenidos en este análisis teórico aportan bastante margen de maniobra para alcanzar, en la práctica, velocidades de análisis superiores a las que puede ofrecer la versión software.

Para contrastar resultados se ha realizado una prueba similar con la imagen *mar.jpg* de la Prueba 3, obteniendo un tiempo de ejecución en MATLAB:

Resultado 4. Tiempo de ejecución de mar.jpg en MATLAB

$$\mathrm{t} = 9.488 \text{ seg}$$

Y un tiempo de ejecución en simulación:

Resultado 5. Tiempo de ejecución de mar.jpg en hardware

$$20.74 \cdot 10^{-9} \left(\frac{\text{seg}}{\text{ciclo}}\right) \cdot 8231630 \text{ ciclos} = 0.17072 \text{ seg} \cong 171 \text{ ms}$$

De nuevo se calcula R.

Resultado 6. Ratio R (tiempo ejecución software/tiempo ejecución hardware) para mar.jpg

$$R = \frac{9.488}{0.17072} = 55.57$$

Se puede observar cómo se obtiene un resultado prácticamente similar al Resultado 3.

Capítulo 6

Conclusiones y trabajos futuros

En este capítulo se describen las conclusiones alcanzadas tras el desarrollo, la implementación y las pruebas realizadas sobre el algoritmo tratado en este Proyecto. Se trata de conclusiones relacionadas, por un lado, con el funcionamiento del algoritmo y, por otro lado, relacionadas con los tiempos de ejecución. Se alcanza en primer lugar una conclusión muy importante que determina el porcentaje de bloques incompatibles a partir del cual se considera que una imagen ha sido modificada para ocultar información. Aparte de este resultado se especifican otras conclusiones relacionadas con los parámetros que afectan al rendimiento del algoritmo estudiado: naturaleza, origen o tamaño de la imagen. Para terminar, se describen una serie de líneas futuras de investigación por las cuales deberían dirigirse los próximos estudios relacionados con los temas iniciados en este Proyecto.

6.1 Conclusiones

Las conclusiones que se detallan a continuación se pueden clasificar en tres tipos: en primer lugar, aquellas que derivan de las pruebas hechas directamente con imágenes reales; en segundo lugar, aquellas que derivan de la implementación sobre una FPGA concreta; y en último lugar, aquellas relacionadas con los tiempos medidos de ejecución del algoritmo.

A partir de los resultados obtenidos en el apartado 5.2 se obtienen las siguientes conclusiones relativas al funcionamiento del algoritmo:

- En primer lugar, lo más importante es establecer un umbral de porcentaje de la imagen incompatible por encima del cual se consideraría que la imagen ha sido modificada y que es sospechosa de portar algún tipo de información oculta. Conociendo los resultados del apartado 5.2.1 se puede establecer que un umbral con el que se obtienen conclusiones aceptables es el **0.9%**. Por encima de este umbral de imagen incompatible es probable que la imagen haya sido modificada, aumentando esta probabilidad a medida que aumenta el correspondiente porcentaje, es decir, que la probabilidad de que la imagen de la Prueba 10 haya sido modificada es mucho mayor que la imagen de la Prueba 9.
- Existe una relación entre el tamaño de la imagen portadora y la información oculta con la capacidad del algoritmo para detectar la información escondida. El rendimiento del algoritmo aumenta a medida que el tamaño del mensaje secreto se "adecúa" al tamaño de la imagen. De esta forma un mensaje de pequeño tamaño se va a detectar peor en imagénes muy grandes (Prueba 6), se detectará mejor en imágenes de tamaño medio (Prueba 9); y mucho mejor en imágenes pequeñas (Prueba 2).
- No existe relación entre la procedencia de la imagen y el rendimiento del algoritmo. Las imágenes portadoras de la Prueba 1 y Prueba 3 han sido obtenidas vía Internet, la imagen correspondiente a la Prueba 5 ha sido obtenida con una cámara digital doméstica y la imagen de la Prueba 8 se obtuvo mediante un teléfono móvil; aún así la capacidad de detección de información del algoritmo ha sido idéntica para dichas imágenes.
- A la hora de calcular el valor S, la barrera entre la compatibilidad y la incompatibilidad de un bloque es el valor 16. El hecho de que esa barrera sea un valor y no un intervalo implica que pequeñas imprecisiones a la hora de calcular S, con valores en torno a 16, puede producir que un bloque incompatible sea considerado compatible o viceversa. En desarrollos futuros debe ser considerado este hecho y tomar algún tipo de solución.
- El algoritmo es muy sensible a posibles variaciones o errores que se puedan producir en el cálculo de la matriz de cuantificación Q. El hecho de que un bloque sea tratado como compatible o incompatible depende enormemente de los valores que aparezcan en la matriz Q, pequeñas variaciones en dicha matriz producen resultados fatídicos que acaban produciendo que, por ejemplo, una imagen sea catalogada como portadora de información oculta cuando en realidad no presenta ninguna modificación. Por ello se considera el cálculo de la matriz Q como determinante para el correcto funcionamiento del algoritmo y se advierte de la necesidad de conseguir que esta etapa del algoritmo genere resultados correctos.
- Al final, el rendimiento del algoritmo depende enormemente de la propia imagen portadora. En función del valor de sus píxeles, unas imágenes presentarán una

mayor capacidad para almacenar información oculta sin que ésta sea detectada que otras. Dicho esto se puede observar cómo, por ejemplo, la imagen portadora de la Prueba 7 posee mejor capacidad para ocultar información que la imagen de la Prueba 10; ya que en ésta última el porcentaje de imagen incompatible es mayor.

En cuanto a la implementación sobre FPGA real del apartado 5.3, se deduce que la FPGA a utilizar es de rango medio, no pudiendo utilizarse FPGAs de pequeño tamaño (sobre todo debido a la necesidad de una gran cantidad de pines de entrada/salida) pero tampoco siendo necesarias FPGAs de gran tamaño ya que, como puede observarse en dicho apartado, los recursos empleados no alcanzan el 30% de uso en FPGAs de rango elevado.

Finalmente de los resultados obtenidos en el apartado 5.1 relativo a tiempos se puede concluir que la ejecución hardware es mucho más rápida que la ejecución software (del orden de **55- 57** veces tomando la frecuencia máxima de funcionamiento) a la espera de considerar los tiempos empleados por la FPGA seleccionada para acceder a la memoria externa usada.

6.2 Trabajos futuros

La línea de los principales trabajos futuros que se pueden llevar a cabo a partir de los resultados alcanzados en este Proyecto se ciñen, principalmente, en el uso del código VHDL sobre una FPGA para comprobar su correcto funcionamiento y la mayor velocidad de ejecución que se puede alcanzar sobre un dispositivo hardware. La principal línea de desarrollo es la selección de una FPGA de acuerdo a los criterios del código, la selección de una memoria externa de capacidad suficiente como para almacenar todos los valores previstos y el diseño del controlador que permita la comunicación entre la FPGA y la memoria. Una vez hecho esto, se trata de comprobar la mayor rapidez de análisis del hardware sobre el software para poder extender su uso, en la práctica, a ámbitos en los que se apliquen sistemas de estegoanálisis, tales como el de la seguridad en Internet por ejemplo.

Otras líneas de investigación se pueden encontrar en la mejora del algoritmo mediante nuevas versiones que sean más rápidas. De las simulaciones se obtiene que la etapa que mayor tiempo involucra en el análisis es el cálculo de la matriz de cuantificación Q, por lo que nuevos métodos más rápidos para la obtención de dicha matriz reducirían el tiempo de análisis de forma significativa.

Finalmente, no hay que olvidar que el algoritmo desarrollado se extiende al análisis de imágenes en formato JPEG cuyos valores de píxeles han sido modificados directamente. Aún así existen otras formas de introducción de información sobre una imagen (consultar capítulo 2) que no son detectadas por el algoritmo en cuestión, por lo que también sería necesario extender la implementación hardware sobre otros tipos de

algoritmos de estegoanálisis destinados a la detección de otros métodos de esteganografía.

Capítulo 7

Presupuesto

A la hora de elaborar el presupuesto se contabilizan los costes directos: mano de obra y equipos empleados, directamente a través de sus costes; mientras que para tener en cuenta costes indirectos (luz y transporte) se toma un 20% más de la cifra obtenida por costes directos. El tiempo de dedicación del Proyecto ha sido de 8 meses con 200 horas trabajadas durante esos meses.

Concretamente el presupuesto del Proyecto queda como sigue:

- Personal:

Apellidos y nombre	Categoría	Dedicación (horas)	Coste (euros/hora)	Coste (euros)
Gutiérrez Fernández, Eric	Ingeniero	200	32.6822	6536.44

- Equipos:

Descripción	Coste (euros)	% Uso dedicado al proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable[1] (euros)
Ordenador	600	100	8	60	80.00
Licencia ModelSim (estudiante)	0	100	8	60	00.00
Licencia Xilinx (estudiante)	0	100	8	60	00.00
Licencia MATLAB (Univ. Carlos III de Madrid)	0	100	8	60	00.00

- Total:

Costes directos (euros)	Tasa de coste indirecto (%)	Costes indirectos (euros)	Costes totales (euros)
6616.44	20	1323.29	7939.73

Se obtiene un presupuesto final del Proyecto Fin de Carrera de 7939.73€.

[1] Fórmula de cálculo de la Amortización: (A/B)*C*D
A: nº de meses desde la fecha de facturación en que el equipo es utilizado
B: periodo de depreciación (60 meses)
C: coste del equipo (sin IVA)
D: % del uso que se dedica al proyecto

Glosario

DCT	*Discrete Cosine Transformation*
FPGA	*Field Programme Gate Array*
IDCT	*Inverse Discrete Cosine Transformation*
JPEG	*Joint Photographic Experts Group*
LSB	*Least Significant Bit*
VHDL	*VHSIC Hardware Description Language*

Referencias

[1] A. Ribagorda Garnacho, J. M. Estévez-Tapiador y J. C. Hernández Castro, «Descubriendo el reverso de Internet: web mining, mensajes ocultos y secretos aparentes,» Universidad Carlos III de Madrid, 2007.

[2] P. Bateman, «Image Steganography and Steganalysis,» Tesis Fin de Máster, Universidad de Surrey, 2008.

[3] A. Kerckhoffs, La cryptographie militaire, Journal des sciences militaires, vol. IX, 1883.

[4] J. Fridrich y M. Goljan, «Practical Steganalysis of Digital Images - State of the Art,» SUNY Binghamton, Departamento de Ingeniería Eléctrica, Binghamton, NY 13902-6000, 2002.

[5] J. Fridrich, M. Goljan y R. Du, «Steganalysis Based on JPEG Compatibility,» SUNY Binghamton, Departamento de Ingeniería Eléctrica, Binghamton, NY 13902-6000, 2001.

[6] B. Kaderkulof-Rengifo, J. Velasco-Medina y M. Vera-Lizcano, «Diseño de funciones DSP para compresión de Vídeo usando FPGAs,» Grupo de Bionanoelectrónica, EIEE, Universidad del Valle. A.A. 25360, Cali, Colombia.

[7] A. Martín Marcos, Compresión de Imágenes, Norma JPEG, Ciencia 3, 1999.

[8] A. De la Escalera Hueso, Visión por Computador, Pearson Educación España.

[9] R. C. González, R. E. Woods y S. L. Eddins, Digital Image Processing using MATLAB, Pearson Education, 2004.

[10] J. Fridrich, «http://dde.binghamton.edu/download/» [En línea].

[11] «www.mathworks.com» [En línea].

[12] L. Agostini y S. Bampi, «Integrated Digital Architecture for JPEG Image

Compresion,» de *European Conference on Circuit Theory and Design, Agosto 28-31*, Espoo, Finlandia, 2001.

[13] K. Bukhari, G. Kuzmanov y S. Vassiliadis, «DCT and IDCT Implementations on Different FPGA Technologies,» Laboratorio de Ingeniería de la Computación, Universidad Tecnológica de Delft, P.O. Box 5031, 2600 GA Delft, Países Bajos.

[14] K. Wahid, V. Dimitrov y G. Jullien, «Error-Free Computation of 8x8 2-D DCT and IDCT using Two-Dimensional Algebraic Integer Quantization,» ATIPS Laboratorio, Departamento de Ingeniería Eléctrica e Ingeniería de la Computación, Universidad de Calgary, Calgary, AB, Canada, T2N1N4, 2005.

[15] D. Bishop, «Fixed Point Package User's Guide,» 2008.

[16] «www.vhdlguru.blogspot.com.es» [En línea].

[17] V. Lokeswara Reddy, D. A. Subramanyam y D. P. Chenna Reddy, «SteganPEG, Steganoraphy + JPEG,» Y.S.R. Dist, A.P. India, 2011.

[18] K. Cabeen y P. Gent, «Image Compression and the Discrete Cosine Transform,» Math 45, College of Redwoods.

Anexo A

Uso de números decimales en VHDL

Para el uso de números decimales en el presente proyecto se ha empleado una librería de VHDL cuya documentación se puede encontrar en [15]. Esta librería establece una nomenclatura muy simple a la hora de utilizar números decimales y de realizar operaciones aritméticas con ellos.

El formato decimal que implementa la librería es la de punto fijo, es decir, siempre va a existir un número fijo de bits decimales, número que se va a definir junto con la definición de la propia señal o variable que se desee utilizar. De esta forma toda magnitud estará definida tanto por un número fijo de bits enteros como por un número fijo de bits decimales. Los bits enteros representan potencias de 2 y los bits decimales representan potencias inversas de 2. Por ejemplo, si se desea representar 2.75 con dos bits enteros y dos bits decimales se tiene:

$$2.75 = 1\ 0\ 1\ 1 = 2^1 + 0 + 2^{-1} + 2^{-2}$$

El formato punto fijo es considerado como un paso intermedio entre el formato entero y el formato de coma flotante (empleado en los computadores). Esto presenta la ventaja de ser casi tan rápido en ejecución como el paquete de VHDL para operaciones con enteros, pero con la ventaja de poder representar valores menores a 1.0.

Para ser usado en un documento VHDL, en primer lugar es necesario descargar la librería desde la dirección:

http://www.vhdl.org/fphld/vhdl.html

Una vez obtenida la librería se adjunta al proyecto en el que se desee usar el fichero:

fixed_pkg_c.vhd

Finalmente para poder utilizarla en un archivo del proyecto se necesita añadir, al comienzo del fichero, la instrucción:

use work.fixed_pkg.all;

Esta librería define dos tipos de datos nuevos:

- *ufixed*: números reales sin signo.
- *sfixed*: números reales con signo.

A la hora de definir cualquiera de los dos tipos de datos anteriores se debe especificar el número de bits dedicados a la parte entera y el número de bits dedicados a la parte decimal. Para ello la parte entera se especifica con índices positivos incluyendo el 0 (es decir: 0, 1, 2, 3,…), mientras que la parte decimal se especifica con índices negativos (es decir: -1, -2, -3,…). Por ello, cuando se define, por ejemplo:

signal y: ufixed (4 downto -5)

Se está definiendo una señal con valor decimal sin signo con 5 bits dedicados a la parte entera (desde 4 hasta 0) y 5 bits dedicados a la parte decimal (desde -1 hasta -5). De esta forma al asignar a *y* el valor:

y<=”0011010000”

Se está asignando el valor:

$$y <= 2^2 + 2^1 + 2^{-1} = 6.5$$

En VHDL esta asignación se realiza como sigue:

y<=to_ufixed(6.5,4,-5)

Donde 4 es el índice entero superior y -5 es el índice decimal inferior.

De la misma forma, para decimales con signo, la definición sería:

signal y: sfixed (4 downto -5)

Y la asignación:

y<=to_sfixed(6.5,4,-5)

Finalmente, a la hora de realizar operaciones aritméticas, se debe tener en cuenta una serie de reglas para ajustar el ancho de bit necesario en el resultado a partir de los anchos de bit de los elementos involucrados en la operación y la propia naturaleza de la operación. Estas reglas vienen resumidas en la Figura 50 y su estudio es muy importante

para comprender el funcionamiento de los módulos de cálculo DCT (apartado 4.3.3) y cálculo IDCT(apartado 4.3.6).

Operation	Result Range
A + B	Max(A'left, B'left)+1 downto Min(A'right, B'right)
A - B	Max(A'left, B'left)+1 downto Min(A'right, B'right)
A * B	A'left + B'left+1 downto A'right + B'right
A rem B	Min(A'left, B'left) downto Min(A'right, B'right)
Signed /	A'left - B'right+1 downto A'right - B'left
Signed A mod B	Min(A'left, B'left) downto Min(A'right, B'right)
Signed Reciprocal(A)	-A'right downto -A'left-1
Abs(A)	A'left +1 downto A'right
- A	A'left +1 downto A'right
Unsigned /	A'left - B'right downto A'right - B'left -1
Unsigned A mod B	B'left downto Min(A'right, B'right)
Unsigned Reciprocal(A)	-A'right +1 downto -A'left

Figura 50. Reglas de tamaño de bits en operaciones aritméticas con punto fijo

Ejemplos de uso [15]:

Ejemplo 14. Multiplicación decimal en punto fijo

signal x : ufixed (7 downto –3);
signal y : ufixed (2 downto –9);

*x * y = ufixed (7+2+1 downto –3+(-9)) ó ufixed (10 downto –12);*

Ejemplo 15. División decimal en punto fijo

signal x : sfixed (-1 downto –3);
signal y : sfixed (3 downto 1);

x/y = sfixed (-1-1+1 downto–3-3) ó sfixed (-1 downto –6);

Los conceptos explicados en este Anexo son los necesarios para comprender el funcionamiento de los módulos del sistema desarrollado que hacen uso de nomenclatura decimal. Aún así, el paquete utilizado ofrece más herramientas y aplicaciones que pueden ser consultadas y estudiadas en [15].

Anexo B

Códigos MATLAB

- Algoritmo de estegoanálisis:

```
% Algoritmo de estegoanalisis que implementa el metodo de compatibilidad
de
% imagenes JPEG
% Este programa se diseño para comprobar la factibilidad a la hora de
% implementar el mismo codigo sobre hardware en una FPGA
% La entrada del programa es la imagen que se desea analizar
% Las salidas son :
%   mayorque16: numero de bloques incompatibles
%   inc_idct: numero de bloques con S<16 pero que son incompatibles
debido
%             al analisis de la IDCT
%   s: vector que contiene los valores S de todos los bloques analizados
%   Q: matriz de cuantificacion
%   DCT: imagen transformada al dominio de la frecuencia a traves de la
DCT
% Se puede observar que el programa posee mas salidas de las necesarias,
% esto se debe a que el codigo ha sido utilizado para la realizacion de
% pruebas y la comparacion con el algoritmo implementado en VHDL
% Ejemplo de uso: lectura de la imagen en im y posterior analisis
% im=double(rgb2gray(imread('lena.jpg')));
% [mayorq16 inc_idct s Q DCT]=compatibilidadJPEG(im);
```

```
function [mayorque16 inc_idct s Q DCT]=compatibilidadJPEG(im)
% Se calculan las dimensiones de la imagen
[xtotal ytotal]=size(im);
% Se ajustan las dimensiones a multiplos de 8
xuso=8*floor(xtotal/8);
yuso=8*floor(ytotal/8);
% Se inicializan algunas variables
flag=0;
cont=0;
cont_min=0;
min_umbral=[];
umbral=0;
% En primer lugar se recorre la imagen buscando bloques saturados
% Aquellos bloques saturados no se eliminan pero se les da valor 128 a
los
% 64 elementos que lo forman (posteriormente estos elementos pasana a
ser cero), de esta manera no afectara al calculo y los
% resultados seran los mismos que si se eliminaran de la imagen
for i=1:xuso/8
    for j=1:yuso/8
        for k=0:7
            for l=0:7
                if (im(8*i-7+k,8*j-7+l)==0||im(8*i-7+k,8*j-7+l)==255)
                    im(8*i-7:8*i,8*j-7:8*j)=128;
                    flag=1;
                    cont=cont+1;
                    break;
                end
            end
            if flag==1
                flag=0;
                break;
            end
        end
    end
end
% La variable T indica el numero de bloques a analizar
T=(xuso*yuso/64)-cont;
T
% Calculo de la matriz DCT y la matriz Q
DCT=zeros(xuso,yuso);
% im_red se queda con la imagen ajustada a multiplos de 8 en cuanto a su
% dimension
im_red=im(1:xuso,1:yuso);
% Para realizar la DCT de la imagen, sus pixeles ven reducido su valor
en
% 128 de acuerdo a lo establecido en el formato JPEG
im_red=im_red-128;
% A cada bloque de 8x8 se le aplica la DCT implementada en MATLAB
for i=1:8:xuso
    for j=1:8:yuso
        % int16 redondea los valores decimales a valores enteros
        DCT(i:i+7,j:j+7)=int16(dct2(im_red(i:i+7,j:j+7)));
    end
end
e=zeros(102,1);
Q=ones(8,8);
```

```
% Ahora se calcula la matriz de cuantificacion Q
% Para ello se lleva a cabo el metodo explicado en el documento [4]
% buscando los minimos de la distribucion E(q)
for i=1:8
    for j=1:8
        A=DCT(i:8:xuso,j:8:yuso);
        % Los posibles valores de q van desde 1 hasta 102
        for q=1:102
            for k=1:T+cont
                % Se calcula E(q)
                e(q)=abs(A(k)-q*round(A(k)/q))+e(q);
            end
            e(q)=e(q)/T;
        end
        % Una vez calculada E(q) en forma de vector se restan sus
valores:
        % cada elemento es restado por el siguiente obteniendo f
        % En este vector f los minimos seran aquellos puntos en donde
los
        % elementos pasen de tener un valor positivo a tener un valor
        % negativo
        f=diff(e);
        for l=1:100
            % Se realiza la busqueda de minimos teniendo en cuenta el
            % umbral definido por min_umbral
            if (f(l)<0 && f(l+1)>0)
                if cont_min==0
                    Q(i,j)=l+1;
                    % El umbral se actualiza con cada minimo hallado
                    min_umbral=[min_umbral e(l+1)];
                    cont_min=cont_min+1;
                else
                    umbral=mean(min_umbral)+3*var(min_umbral);
                    % Se ofrece cierto margen al umbral para el calculo
de
                    % los minimos
                    if e(l+1)<=(umbral+0.5)
                        % Si el minimo se encuentra por debajo del
umbral
                        % se contabiliza
                        Q(i,j)=l+1;
                        cont_min=cont_min+1;
                    else
                        % Si se ha hallado un minimo fuera del umbral se
                        % considera que los valores son demasiado
grandes y
                        % no se van a encontrar nuevos minimos correctos
                        % Se sale del bucle for
                        break;
                    end
                    min_umbral=[min_umbral e(l+1)];
                end
            end
        end
        if Q(i,j)==1
            if e(1)>0.6*e(2)
                Q(i,j)=2;
            end
        end
        e=zeros(102,1);
        cont_min=0;
```

```
        min_umbral=[];
        umbral=0;
    end
end
% Se calcula S y se aplica el criterio explicado en el apartado 2.3
s=zeros(xuso*yuso/64,1);index=1;
mayorque16=0;
inc_idct=0;
sum_max=0;
for i=1:8:xuso
    for j=1:8:yuso
        for k=0:7
            for l=0:7
                s(index)=(abs(DCT(i+k,j+l)-
Q(k+1,l+1)*round(DCT(i+k,j+l)./Q(k+1,l+1))))+s(index);
            end
        end
        % Si S>16 el bloque es incompatible
        if s(index)>16
            mayorque16=mayorque16+1;
            inc_idct=inc_idct+1;
       else
         % Analisis de la IDCT
         % Calculo de la matriz P1
           m1=(round((DCT(i:(i+7),j:(j+7)))./Q)).*Q;
         % La matriz B se debe comparar con el bloque original
almacenado en im_red
           B=round(idct2(m1));
           sum=0;
           for a=0:7
               for b=0:7
                   % Para comparar ambas matrices se restan elemento a
                   % elemento y se suma el resultado de todas las restas
                   % Si esta suma es cero es que son completamente
                   % identicas
                   sum=abs(im_red(i+a,j+b)-B(a+1,b+1))+sum;
               end
           end
           if sum>sum_max
               sum_max=sum;
           end
           % Por cuestiones de precision es posible que la diferencia
entre
           % B e im_red sea de alguna unidad, de ahi que se deje cierto
           % margen en cuanto a la suma de las restas
           if sum>50 % 50 es un valor empirico que da buenos resultados
               inc_idct=inc_idct+1;
           end
        end
        index=index+1;
    end
end
end
```

- Código cuantificacionQ.m:

```
% function Q=cuantificacionQ(A,num_sat)
% Esta funcion se ha implementado con el objetivo de visualizar la
% distribucion E(q) que se debe utilizar para calcular la matriz de
% cuantificacion. Permite la vision de los minimos de la funcion
% directamente con el objetivo de comprobar el resultado correcto que se
% debe obtener
% Entradas:
%   - A: elementos dct i de todos los bloques no saturados
%   - num_sat: numero de bloques saturados
% Salidas:
%  - Q: minimo obtenido
% Ejemplo de uso: se supone que se desea calcular el elemento 3x3 de la
% matriz de cuantificacion Q
% Se parte de la salida DCT del algoritmo compatibilidadJPEG y del
numero
% de bloques saturados (num_sat)
% dim=size(DCT);
% A=DCT(3:8:dim(1),3:8:dim(2));
% Q=cuantificacionQ(A,num_sat)

function Q=cuantificacionQ(A,num_sat)
%%hold off;
[x y]=size(A);
e=zeros(150,1);
cont_min=0;
min_umbral=[];
umbral=0;
Q=1;
for i=1:150
    for j=1:x
        for h=1:y
            % En primer lugar se calcula la distribución E(q)
            e(i)=abs(A(j,h)-i*round(A(j,h)/i))+e(i);
        end
    end
    e(i)=(1/(x*y-num_sat))*e(i);
end
% Una vez calculada E(q) en forma de vector se restan sus valores:
% cada elemento es restado por el siguiente obteniendo f
% En este vector f los minimos seran aquellos puntos en donde los
% elementos pasen de tener un valor positivo a tener un valor
% negativo
f=diff(e);
for i=1:148
    % Posteriomente se calcula el minimo de la misma forma que se hizo
en
    % en compatibilidadJPEG.m
    % Se realiza la busqueda de minimos teniendo en cuenta el
    % umbral definido por min_umbral
    if (f(i)<0 && f(i+1)>0)
                if cont_min==0
                    Q=i+1;
                    % El umbral se actualiza con cada minimo hallado
                    min_umbral=[min_umbral e(i+1)];
                    cont_min=cont_min+1;
                else
                    umbral=mean(min_umbral)+3*std(min_umbral);
                    % Se ofrece cierto margen al umbral para el calculo
de
```

```
                    % los minimos
                    if e(i+1)<=(umbral+0.5)
                        % Si el minimo se encuentra por debajo del
umbral
                        % se contabiliza
                        Q=i+1;
                        cont_min=cont_min+1;
                    else
                        % Si se ha hallado un minimo fuera del umbral se
                        % considera que los valores son demasiado
grandes y
                        % no se van a encontrar nuevos minimos correctos
                        % Se sale del bucle for
                        break;
                    end
                     min_umbral=[min_umbral e(i+1)];
                end
    end
end
% Se representa E(q) y el umbral
plot(1:150,e);
hold on;
plot(1:150,umbral);
```