Escuela
Politécnica
Superior

Arquitectura de Alto Rendimiento para el Cálculo de la DCT

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Lázaro Corral Sánchez

Tutor/es:
Higinio Mora Mora

Universitat d'Alacant
Universidad de Alicante

Julio 2015

Arquitectura de Alto Rendimiento para el Cálculo de la DCT

Lázaro Corral Sánchez
lcs21@alu.ua.es

Sinopsis. En este trabajo se han revisado los principales métodos de cálculo de la Transformada Discreta del Coseno y se han analizado sus implementaciones. A partir de esta información se ha propuesto una arquitectura de cálculo de alto rendimiento que pone en práctica técnicas de aritmética de computadores en el desarrollo de operadores para crear una estructura compacta que calcula la transformada a partir de su formulación directa. Se ha implementado y simulado el funcionamiento de la arquitectura propuesta en tarjetas reconfigurables para el Procesamiento de señales digitales, para evaluar su rendimiento en términos de área, retardo y potencia consumida. Además, se ha calculado su rendimiento con un modelo homogéneo e independiente de la tecnología de implementación con el propósito de comparar sus prestaciones con las de otras técnicas conocidas.

Palabras clave: Aritmética Digital, Transformada Discreta Coseno, Sistemas y Circuitos, Aritmética Distribuida.

1 Introducción

La Transformada Discreta del Coseno (DCT) es una de las transformadas más utilizadas en el procesamiento digital de imágenes debido a su capacidad de compactación de energía y, por tanto, su eficacia para los procesos de compresión de imágenes digitales [1].

La utilización de estos algoritmos en los nuevos dispositivos de consumo, como reproductores, teléfonos móviles o tablets requieren de una capacidad de proceso que

proporcione la productividad (MB/s) adecuada a las demandas de calidad de servicio deseado. En estos aparatos es necesario un alto volumen de transferencia de datos, y consecuentemente compresión de estos.

Las necesidades de cálculo aritmético intensivo para el procesamiento de los métodos de compresión mediante la transformada del coseno justifican el desarrollo de arquitecturas y hardware especializado que puedan cumplir con las restricciones temporales de esas aplicaciones al tiempo que cumplen con las condiciones de miniaturización necesarias para ser embebidos en esos dispositivos. Todo avance en su desarrollo y construcción que mejore la capacidad de procesamiento de estos aparatos será bienvenida en una industria muy competitiva para aumentar las prestaciones y la calidad de servicio al usuario.

En este trabajo se describe una implementación que pone en práctica técnicas de desarrollo de operadores aritméticos al método de cálculo DCT en su conjunto para mejorar el rendimiento de las operaciones. Básicamente, estas técnicas buscan reorganizar los procesos de cálculo de la transformada para tratar su computación de forma integrada. Se parte de la descripción básica del método de cálculo DCT por tener la formulación más simple y se compara con otras técnicas aritméticas de cálculo de la transformada como la aritmética distribuida.

En el cálculo de la DCT nos centramos en funciones discretas 2D de 8x8 muestras por ser el tamaño más aceptado en los algoritmos de compresión de imagen [1]. De este modo el método de implementación propuesto está optimizado para funciones de entrada de ese tamaño.

A modo de resumen este artículo está reoganizado de la siguiente manera: las expresiones 2-D de la DCT y sus propiedades más relevantes son descritas brevemente en la Sección II. En la sección III se revisa las principales técnicas de diseño e implementaciones hardware para él algoritmo. La Sección IV expone la arquitectura propuesta y la sección V evalúa su rendimiento en cuestiones de área,

retardo y potencia consumida. Por último en la Sección VI se hace una comparativa con las técnicas actuales, y las conclusiones se realizan en la Sección VII.

2 Descripción del Algoritmo DCT

La 2-D DCT para un bloque de muestra 8x8 {f(x, y), x = 0, 1, …, 7, y = 0, 1, …, 7} se expresa generalmente como:

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

Donde:

$$(u), C(v) = \begin{cases} \frac{1}{\sqrt{2}}, & if\ u,v = 0 \\ 1, & en\ otro\ caso \end{cases} \quad (2)$$

El cálculo de la transformada se realiza a través de sus propiedades independientes. Esto es, la 2-D DCT es organizada en dos cálculos consecutivos de 1-D DCT. Primero se calcula la transformada por filas y luego por columnas.

Para 1-D DCT de 8 muestras tenemos que {f(x), x = 0, 1, …, 7} puede ser expresado como:

$$\mathrm{F(u)} = \frac{C(u)}{2}\sum_{x=0}^{7} f(x)\cos\left[\frac{(2x+1)u\pi}{16}\right] \quad (3)$$

Donde C(u) es definida con la misma ecuación que en (2).

El factor de escala (1/2) puede ser eliminado sin la pérdida de sus propiedades generales para facilitar el cálculo. De manera que, de una forma desplegada, la fórmula anterior puede escribirse como:

$$F(u) = Cuof(0) + Cu1f(1) + \ldots + Cu7f(7) \qquad (4)$$

Donde:

$$Cuj \in C \,/\, C \equiv$$
$$\equiv \{ cos\left[\frac{\pi}{16}\right], cos\left[\frac{\pi}{8}\right], cos\left[\frac{3\pi}{16}\right], cos\left[\frac{\pi}{4}\right], \qquad (5)$$
$$, cos\left[\frac{5\pi}{16}\right], cos\left[\frac{3\pi}{8}\right], cos\left[\frac{7\pi}{16}\right]\}$$

La expresión (3) anterior puede ser escrita en notación vectorial de la siguiente manera:

$$F = \frac{1}{2} T(N)_8 \, f \qquad (6)$$

Donde $F = [F(0), \ldots, F(7)]^T$, $f = [f(0), \ldots, f(7)]^T$ y $T(N)_8$ es una matriz de 8 x 8 donde su (u, j)esimo componente es:

$$T(u,j)_8 = C(u)\ cos\left[\frac{(2j+1)u\pi}{16}\right] \qquad (7)$$

Esta expresión ha facilitado la computación mediante métodos de cálculo de productos parciales. Más aun, la expresión

(6) puede ser dividida a su vez, en partes pares e impares para conseguir subexpresiones matriciales más simples.

3 Técnicas de diseño

El propósito de este apartado es el de ordenar las aportaciones, presentar los avances más relevantes y mostrar las principales líneas de investigación en las que se encuadra el trabajo presentado en este artículo.

El cálculo directo de la 2-D DCT requiere de 4096 multiplicaciones and 4032 sumas, y el cálculo separado por filas y columnas de 1-D DCT necesita únicamente 1024 multiplicaciones and 896 sumas. No obstante, el coste computacional sigue siendo elevado si queremos alcanzar los tiempos proporcionados por las aplicaciones actuales.

Existen numerosos métodos y diseños de arquitecturas orientadas a reducir este coste computacional. Además, la proliferación en los últimos años de dispositivos móviles ha creado la necesidad de embeber estas arquitecturas en máquinas de pequeño tamaño pero que necesitan un potente procesamiento en este aspecto.

A. Métodos de Cálculo

En general, los métodos propuestos para implementar el algoritmo DCT se pueden clasificar según los procedimientos de cálculo empleados en su procesamiento.

En primer lugar, un importante conjunto de métodos se basa en la expresión vectorial del cálculo de la DCT descrita en las expresiones (6) y (7). Como se ha mencionado en el apartado anterior, la ecuación (6) puede ser descompuesta de forma recursiva para obtener una expresión final basada en el cálculo de productos parciales en el que el vector formado por los coeficientes [9], [10], [45], [46]. De este modo se obtienen patrones de computación regulares que facilitan su implementación posterior para aplicaciones de procesamiento de señal [11].

Otros métodos aprovechan las simetrías de las operaciones de cálculo derivadas de la ecuación (3) para reducir el número de operaciones necesarias. En este conjunto se han propuesto numerosos diseños con el objetivo de minimizar las multiplicaciones necesarias. Algunas de las propuestas más populares are Chen [4], Wang [5], Lee [6], Vetterli [7] o Loeffer [8] methods. Este último método consigue realizar el cálculo con tan sólo 11 operaciones de multiplicación. Los métodos anteriores se caracterizan por el diseño de su Cálculo del Grafo de Flujo y las Configuraciones de Mariposa.

B. Implementaciones Hardware

Teniendo en cuenta que el coste del procesamiento es un aspecto muy relevante en los procesos de compresión de imágenes en aplicaciones o dispositivos electrónicos con restricciones temporales, se han desarrollado numerosas arquitecturas hardware que realizan la implementación VLSI de los métodos descritos en el apartado anterior.

La implementación con Aritmética Distribuida (DA) a partir de la formulación vectorial de la DCT supone una técnica eficiente para procesar estas operaciones cuando uno de los vectores es fijo debido a la regularidad de su implementación. En los últimos tiempos se ha desarrollado abundante investigación para este método de cálculo de la DCT. Existen principalmente dos aproximaciones para implementar una unidad de cálculo DA: basadas ROM y basadas en sumadores. Las DA basadas en ROM o aproximación convencional acelera las operaciones de multiplicación precalculando los productos para cada posible valor de los datos de entrada y obtiene el resultado mediante algoritmos de desplazamiento y suma. Este modo requiere de una memoria ROM o Tablas de consulta (LUT) en las que almacenar estos productos precalculados [11], [20], [21], [22]. El segundo tipo, basado en sumadores, no requiere del uso de Tablas LUT y puede aprovechar la distribución de los patrones de operaciones binarias [23], [24], [25], [26] pero necesita más operaciones de suma y son más lentas que las anteriores.

En cuanto a las implementaciones de los algoritmos basados en grafos de flujo (FGA) para el cálculo de la DCT hay que tener en cuenta los siguientes factores. Algunas de estas arquitecturas tratan de minimizar el impacto del procesamiento de las multiplicaciones en coma flotante implicadas, sustituyéndolas directamente por sumas y desplazamientos [12]. Otras propuestas calculan la transformada sin necesidad de multiplicaciones mediante la aproximación de estas operaciones por fracciones enteras con denominador potencia de dos. Con esta técnica, cada multiplicación se sustituye por una serie de sumas enteras y un desplazamiento posterior [14], [15]. Aunque con este método se evitan las multiplicaciones se obtiene una calidad ligeramente inferior de la imagen comprimida. En esta misma línea, otra técnica consiste en utilizar el método CORDIC para sustituir las costosas multiplicaciones y obtiene el cálculo de las configuraciones de mariposa mediante rotaciones de ángulos [16], [17], [44]. En estos métodos, el compensado final necesario del método CORDIC se incluye en la etapa posterior de escalado en el proceso de compresión de la imagen. Existen otros trabajos que describen variantes de los métodos anteriores con el propósito de minimizar las operaciones implicadas en el cálculo del algoritmo. En [13] se presenta un método que mejora el rendimiento ofrecido por las variantes CORDIC a partir del método de Loeffer colocando las multiplicaciones en la última etapa del grafo de flujo.

Finalmente, existen propuestas que presentan variantes del método para adaptarlos a determinadas aplicaciones, por ejemplo, DCT tamaño principal [27] o adaptación de estados DCT [28] y otras implementaciones VLSI personalizadas que tratan de explotar diferentes aspectos de la transformada para construir diseños específicos, como por ejemplo [18], [19], [29], [30], [31].

Como hemos comprobado existe gran cantidad de trabajo desarrollado en este campo con multitud de propuestas y arquitecturas para el cálculo e implementación de la DCT lo que demuestra el interés del tema objeto de estudio incentivado también por el desarrollo de los nuevos aparatos móviles que requieren de sistemas de codificación de media data eficientes y veloces. Aunque las sucesivas propuestas que hemos presentado van avanzando en esa línea, pensamos que existe margen para aplicar técnicas de procesamiento aritmético que de forma integrada mejoren el rendimiento, la satisfacción al

usuario y permitan alcanzar mayores ámbitos de aplicación.

4 Arquitectura Propuesta

En este artículo, nuestro objetivo es presentar una arquitectura hardware más eficiente para el cálculo de la DCT usando técnicas de computación aritmética para procesar los esquemas de sumas y multiplicaciones implicadas.

Usaremos el cálculo directo de la DCT que expresa la formulación original de la ecuación (4), ya que es la expresión más simple de su cálculo. Según esta fórmula, el cálculo de cada F(u) está formado por 8 multiplicaciones y 7 sumas. Cada una de estas multiplicaciones se realiza entre un valor de entrada f(x) y una de las constantes Cuj del conjunto definido en (5).

Las características que definen la arquitectura propuesta son dos: cómo se integran las operaciones de multiplicación y suma necesarias para obtener cada F(u) y de qué manera intervienen las constantes del conjunto definido en (5) en el cálculo de cada producto.

Con respecto a la primera cuestión relativa a la cadena de multiplicaciones y sumas, el procedimiento estándar de cálculo de cada operación de multiplicación se compone a su vez de las bien conocidas fases siguientes [32]: generación de productos parciales, reducción de productos parciales y suma final. La idea que se propone es simple: consiste en implementar de forma combinada las 8 multiplicaciones y las 7 sumas de cada F(u), de modo que se integren las reducciones de productos parciales con las sumas necesarias en cada cálculo. De este modo se puede construir un árbol de reducción conjunto que obtiene el resultado final con una implementación VLSI más eficiente. En términos generales, la arquitectura propuesta considera un diseño para el cálculo de cada F(u) formado por las tres etapas anteriores de cálculo de las multiplicaciones. La figura siguiente muestra un esquema del proceso completo de cálculo de cada F(u) de la transformada.

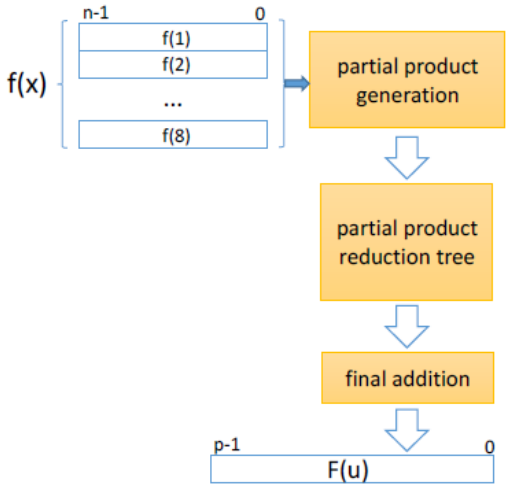

Figura 1: Esquema general de la arquitectura.

En cuanto al tratamiento de las constantes, las diferentes propuestas de implementación que se han repasado en el apartado anterior se han aprovechado de que uno de los factores de estas operaciones tiene siempre un valor conocido para diseñar arquitecturas que eviten su cálculo reiterado. Las propuestas de implementación aritmética distribuidas basadas en ROM construyen una o varias memorias con los cálculos de estos productos para obtener directamente los resultados e ir acumulando los sumandos hasta el resultado final, mientras que las propuestas basadas en FGA tan sólo almacenan el valor de las constantes para utilizarlas en los cálculos necesarios del procesamiento.

Hay varias soluciones para obtener los productos parciales de la multiplicación por las constantes. En esta investigación se analizan dos alternativas: *Basada en la Generación de Registros de productos parciales y Generación Directa de productos parciales.*

A. Generación de Registros de productos parciales

En esta alternativa de diseño no se necesita una ROM propiamente dicha para almacenar los resultados de las multiplicaciones por las constantes, sino tan sólo una colección de registros con la multiplicación de las constantes por algunos números base. De este modo, se podrá dirigir directamente la salida de los registros hacia las entradas de todos los operadores donde intervengan de forma simultánea mediante un bus compartido sin necesidad de direccionamiento ni retardos de espera en los accesos a memorias embebidas [33].

Los números base quedarán definidos por el tamaño de los operadores y la cantidad de productos parciales a generar. Si se considera una longitud k para los operadores, el conjunto de valores que será necesario almacenar para implementar este diseño estará formado por la multiplicación de las 7 constantes con signo de (5) por los números impares entre 0 y 2^k-1. Esto es, será necesario precalcular $7 \cdot 2^k/2$ valores y alojarlos en sus respectivos registros para ser usados en el procesamiento.

Por ejemplo, para k = 4, será necesario precalcular el siguiente conjunto extendido C' de constantes para cada elemento del conjunto C definido en (5):

$$C' \equiv \{\forall ci \in C, RCi = \{ci, 3ci, 5ci, 7ci, 9ci, 11ci, 13ci, 15ci\}\} \qquad (8)$$

Como decisión de implementación se pueden colocar todos los múltiplos de cada constante en un único registro de longitud $m \cdot 2^k/2$, donde m es la precisión de cada valor. Por ejemplo, para k = 4 y m = 12, harían falta sólo 7 registros de 96 bits para almacenar esos datos.

Mediante el conjunto de registros anterior, el procesamiento de cada multiplicación se realiza fragmentando los multiplicandos (f(x)) en trozos de longitud k y obteniendo los productos parciales conectando el conjunto de registros extendido C'. De este modo se generan n/k productos parciales de cada multiplicación, donde n es la longitud de las entradas f(x). La figura siguiente ilustra este esquema.

El error que se comete en cada producto debido a esa forma de compensación del complemento estará, por tanto, dentro de los bits menos significativos del resultado, de modo que, debido a la precisión final necesaria para la función DCT para una longitud n y m apropiada, el error anterior quedará dentro de los bits desechados.

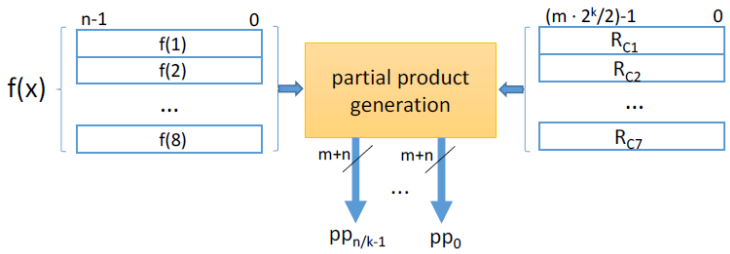

Figura 2: Esquema Generación de productos de registros parciales.

El direccionamiento en las conexiones de estos registros se realiza mediante elementos multiplexores con k entradas de control. Por ejemplo, en la siguiente figura se muestra el detalle de la generación de los productos parciales de una multiplicación para k = 4 bits de longitud.

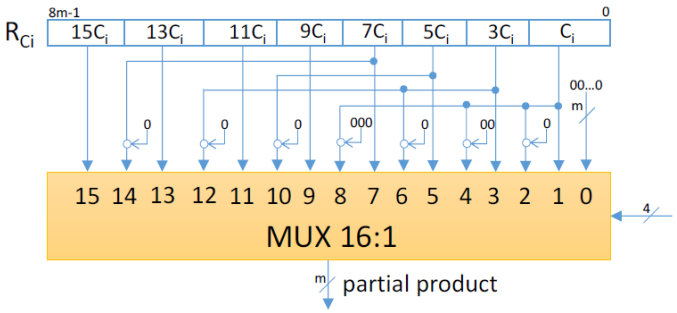

Figura 3: Multiplexor Selección Productos Parciales.

Se utilizará el formato de complemento a 2 para codificar el signo de los productos parciales generados. En este caso, la suma de un uno adicional en la parte menos significativa para realizar el complemento se compensa en el último producto extendiendo los bits de signo también hacia la derecha hasta completar toda la longitud. La siguiente figura describe con detalle el esquema para producir los productos parciales de una multiplicación considerando la obtención de 3 productos parciales en cada multiplicación. El signo de la operación (sgn.) vendrá determinado por el signo de la multiplicación en el cálculo del término correspondiente.

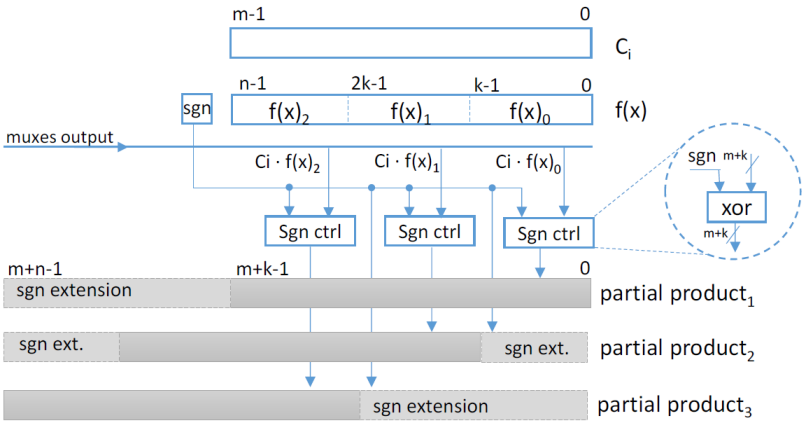

Figura 4: Generación Productos Parciales con compensación de error.

B. Generación Directa de productos parciales

Otra alternativa de diseño de este módulo pretende evitar el empleo de multiplexores y el coste de área que conllevan. Sin embargo, la técnica propuesta no emplea la generación simple directa por el elevado número de productos que se obtienen ni tampoco el método de Booth debido a los retardos adicionales que incorpora [34]. El método propuesto consiste en realizar una generación directa tan sólo de los dígitos distintos de cero. Para ello, se analiza la distribución de bits de cada factor fijo y se crea el circuito personalizado para cada caso. La siguiente tabla muestra las

constantes Ci del conjunto (5) con su valor decimal y binario.

Tabla 1

Cálculo de Constantes DCT

Ci	Valor Decimal	Valor Binario
cos[π/16]	0.9807852	0.111110110001
cos[π/8]	0.9238795	0.111011001000
cos[3π/16]	0.8314696	0.110101001101
cos[π/4]	0.7071067	0.101101010000
cos[5π/16]	0.5555702	0.100011100011
cos[3π/8]	0.3828125	0.011000100000
cos[7π/16]	0.1953125	0.001100100000

Con el propósito de mejorar la generación de productos parciales, se utilizará la representación de las constantes anteriores con cifras con signo que minimice la cantidad de dígitos distintos de cero manteniendo al mismo tiempo el error cometido dentro de los límites representables con 12 bits. La representación con signo utilizada es la canónica con signos en los dígitos (CSD) ya que el número de dígitos que no son cero es mínimo [40], [41]. La tabla siguiente muestra la representación con dígitos con signo y la cantidad de productos parciales resultantes en cada caso.

Tabla 2

CSD Cálculo de Constantes DCT

Ci	Valor binario	Valor CSD binario	Productos Parciales
cos[π/16]	0.111110110001	$1.0000\bar{1}0110001$	5
cos[π/8]	0.111011001000	$1.00\bar{1}011001000$	5
cos[3π/16]	0.110101001101	$0.110101010\bar{1}0$	
cos[π/4]	0.101101010000	0.101101010000	6
cos[5π/16]	0.100011100011	$0.10010\bar{0}100100$	5
cos[3π/8]	0.011000100000	0.011000100000	
cos[7π/16]	0.001100100000	0.001100100000	4
cos[3π/8]	0.011000100000	0.011000100000	3
cos[7π/16]	0.001100100000	0.001100100000	3

Los circuitos generadores producirán directamente los productos parciales directos o inversos según el signo de cada dígito. En la siguiente figura se ilustra el esquema de producción con este sistema para una de las constantes de la tabla anterior, en donde los bits oscuros corresponden a los bits distintos de cero. La forma de compensar el error proveniente de la complementación se realiza de la manera descrita anteriormente.

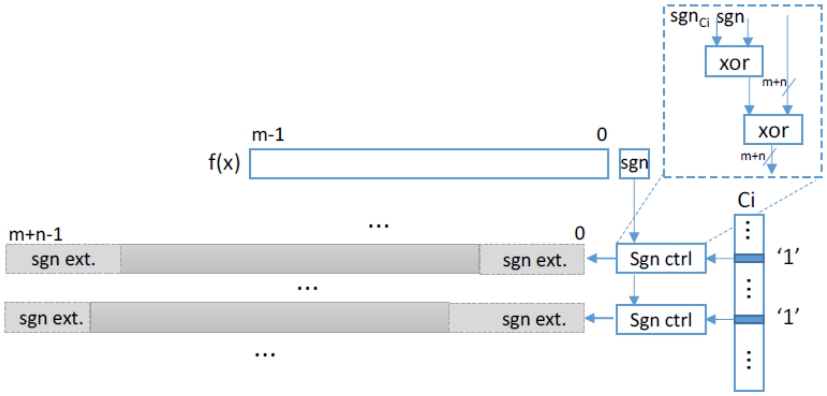

Figura5: Generación Productos Parciales con compensación de error.

La siguiente figura muestra el esquema de este método de cálculo con los generadores de productos parciales de un valor de la función.

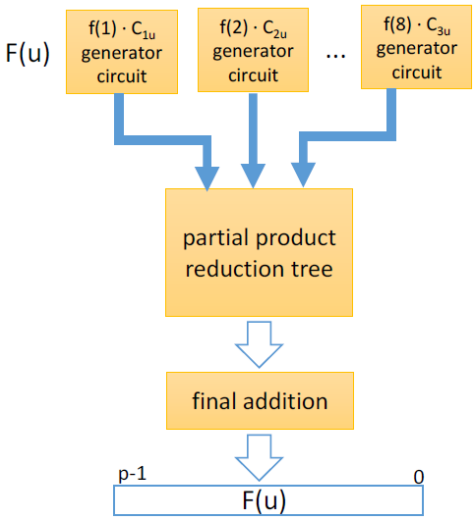

Figura 6: Diagrama de bloque de la propuesta DCT final.

A partir de aquí, los productos parciales generados se reducirán de forma combinada mediante una celda de reducción formado por etapas de contadores y compresores de diferente escala.

Con respecto al primer método de generación, la cantidad de productos generados para cada F(u) será de 8 * ⌈n/k⌉. La configuración concreta de este árbol se podrá determinar según los valores de precisión de n y k. Por ejemplo, para n =12 y k = 4, se tendrá en total 24 productos parciales a reducir que podrán encaminarse según el esquema que describe la siguiente figura.

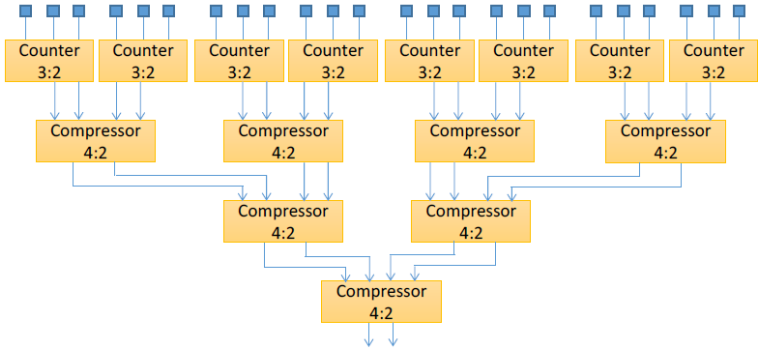

Figura 7: Árbol Reducción de 24 productos parciales.

Con el método de generación directa, la cantidad de productos parciales producidos en cada F(u) oscilan entre 32 y 40, por tanto, es posible construir a partir de aquí un árbol de reducción común para todos ellos que irá alimentándose de la salida de los circuitos generados de cada Fi. Igual que en el caso anterior, al tratar con números con signo codificados en complemento a 2, se utilizan 24 bits para la codificación de cada producto parcial. El árbol de reducción resultante en este caso, se representa en la siguiente figura.

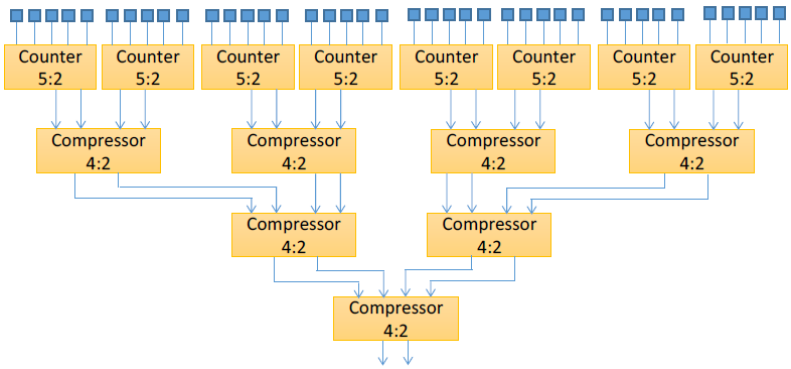

Figura 8: Árbol Reducción de 40 productos parciales.

La suma final generada podrá realizarse según cualquier método conocido de suma, por ejemplo mediante un sumador CLA.

5 Evaluación del Rendimiento de la Arquitectura

La arquitectura especificada en el apartado anterior permite calcular de una forma integrada toda la colección de productos y sumas de cada F(u). La composición de todas las sumas implicadas en un árbol de reducción y compresión crea una estructura compacta que favorece su implementación VLSI.

Para obtener la mayor precisión recomendada por el IEEE 1180-1990 estándar [35] y otros trabajos recientes [12], [13] hemos establecido unas señales de entrada de 12 bits y una separación entre estados de 16 bits. Estas características de precisión establecen unas longitudes de n y m de 12 bits. Se añadirá un bit más en el multiplicando para codificar el signo de las términos con una codificación en complemento a dos de los datos y de los resultados. Los resultados parciales de las sumas y las reducciones se establecerán a los 16 bits más significativos, por lo que la pérdida de precisión será despreciable en los resultados. Se establece un tamaño de bloque de k = 4 para facilitar el direccionamiento de los trozos de cada entrada.

Para mejorar en lo posible el rendimiento, se considerará que las constantes precalculadas están alojadas en Tablas de Consulta (LUT) cerca del operador DCT dentro de la unidad aritmética y por tanto, no es necesario, perder ciclos de ejecución para cargarlas de memoria.

El aspecto más relevante de estudio en este trabajo es el tiempo de retardo en el cálculo de la DCT. No obstante, somos conscientes de la importancia de otras características, como por ejemplo el área ocupada y potencia consumida, los cuales son muy importantes en aplicaciones embebidas. El rendimiento de la técnica propuesta para el cálculo de la DCT está por tanto analizado en términos de área, retardo y potencia consumida para establecer las bases de comparación con otras propuestas.

A. Estimación de Área

Es frecuente el expresar el área en términos de puertas lógicas para obtener una comparación homogénea con otros diseños. Sea τ_a el área ocupada por una puerta lógica, por ejemplo una puerta *XOR*. Con este criterio es sencillo trasladar los resultados a otra escala de comparación como por ejemplo número de transistores o área ocupada en μm^2, ya que existe una amplia literatura con estos datos para una gran cantidad técnicas de implementación [36]. Para facilitar los cálculos, en la tabla siguiente resumimos la estimación de área ocupada de los principales elementos de la arquitectura en términos homogéneos. Estos datos han sido obtenidos por métodos de cálculo de reconocido prestigio en la literatura de computación aritmética [33], [38], [39], [42], [43].

Tabla 3
Coste Estimado de Área

Dispositivo	Área (τa)
3:2 contador	3
4:2 compresor	6
5:2 contador	8
16-bit sumador (CLA)	48
12-bit multiplicador	109
up to 8-bit entrada LUT	30 Γa /Kbit
16:1 multiplexor	16
k-bit RAC (ROM acumulador)	$2^k \cdot$ precision
k-bit registro	k
12-bit acarreo	48

Las mayores aportaciones en el área requerida para la implementación del cálculo F(u) vienen representadas por los valores de entrada en la fig. 1: banco de registros, generación de productos parciales, reducción de productos parciales y su suma final.

La tabla LUT contiene la codificación de las 7 constantes del cálculo DCT. Además, en el caso de implementar una generación basada en registro será necesario almacenar

también los múltiplos impares de estas constantes. Esto nos da un total de área ocupada para almacenar esa información de 7·12 bits = 72 bits para las constantes y 7·7·12 = 588 bits para sus múltiplos. Es necesaria una puerta compleja para almacenar un bit en un biestable, por tanto la codificación de esta información requiere 72 τ_a para las constantes y 588 τ_a para sus múltiplos.

Para proceso de generación de productos parciales tenemos dos opciones de implementación: según el diseño basado en registros el área necesaria está formada por la selección del valor correcto de cada registro mediante un multiplexor, donde las señales de control corresponden con el valor de cada bloque k del multiplicador. La cantidad de puertas complejas que requiere cada uno de estos multiplexores 16:1 es de 16 τ_a [37], por tanto, con la fragmentación de los multiplicadores en 3 trozos, cada multiplicación requiere $3 \cdot 12 \cdot 16 = 576$ τ_a. La realización de todas las multiplicaciones de cada F(u) en paralelo necesita por tanto de 4608 τ_a.

En el caso de implementar una generación directa, la estimación de área de esta etapa vendrá dada solo por el circuito combinacional descrito por la fig. 5 que incluye la colocación del signo correcto. En ese caso es necesario 24 τ_a para establecer cada producto parcial y por tanto 120 τ_a para ejecutar cada multiplicación, por lo que la generación de cada F(u) requiere un área total de 960 τ_a.

En cuanto al árbol de reducción de productos parciales, su coste está ligado a la cantidad de sumandos a reducir. Con total precisión, el resultado de cada multiplicación es de 24 bits, donde cada producto parcial con extensión de signo al principio y/o al final también tiene una longitud de 24.

Según los datos individualizados sobre área ocupada que ilustra la tabla III, el árbol de reducción de productos parciales que ilustran las fig. 7 y 8 requieren un área total de 1536 τ_a y 2544 τ_a para tratar simultáneamente los 24 bits de longitud.

La última etapa de suma final requiere de una operación de suma. Aunque se haya realizado la reducción para sumandos de 24 bits de longitud, la suma se realizará para los 16 bits más significativos de acuerdo a las recomendaciones de precisión que marca el estándar. Utilizando un método conocido de suma, por ejemplo CLA, se requiere un área de 48 τ_a para realizar una suma de 16 bit.

Por tanto, de los resultados anteriores se obtiene que el área total requerida para construir la arquitectura que calcule cada F(u) es de 6732 τ_a en el caso de implementar la generación mediante un diseño basado en registros y de 3504 τ_a para la generación directa.

Para validar el diseño y las estimaciones de su coste se han implementado las técnicas propuestas en una plataforma de síntesis FPGA (ver Apéndice A). La siguiente tabla muestra el coste de área para una implementación FPGA de la función F(u). Los datos no incluyen el coste de almacenamiento de las constantes porque éstas se alojan en la memoria de la tarjeta y no utilizan la lógica programable.

Tabla 4
Coste de Área para Implementación FPGA

Circuito	Celdas	4entrada LUT
4:2 compresor	3	5
3:2 contador	1	2
5:2 contador	3	7
12-bit 16:1 multiplexor	7	12
16-bit sumador (CLA)	12	16
12-bit multiplicador	148	292
Productos parciales basados en registros	1512	2520
Reducción de productos parciales 24:2 (Fig. 7)	504	888
Generación productos parciales directa	560	968
Reducción productos parciales 40:2 (Fig. 9)	1016	1824
Total cálculo basado en registros	2028	3424
Total cálculo directo	1588	2808

B. Estimación de los retardos

Al igual que en subapartado anterior, se fija un valor homogéneo para el cálculo de los tiempos de respuesta de los elementos implicados a fin de facilitar su comparación con otros diseños o propuestas. Se establece en este caso como unidad de medida τ_t el retardo producido por una puerta compleja. La tabla IV resume los retardos de cada componente.

El retardo producido en el procesamiento de cada F(u) es el punto fuerte de nuestra propuesta. Siguiendo el mismo esquema que en el apartado anterior, a continuación, desglosamos los retardos producidos en cada etapa del cálculo.

Las constantes del banco de registros y su direccionamiento hacia el multiplexor de generación de productos parciales no tienen retardo, ya que no están almacenados en

memoria que haya que leer. Su enrutamiento hacia las entradas es directo y los multiplexores se encargarán de seleccionar el dato correcto.

Tabla 5
Estimación Retardo

Dispositivo	Retardo (τt)
3:2 contador	2
4:2 compresor	3
5:2 countador	4
16-bit sumador (CLA)	8
12-bit multiplicador	26
Acceso hasta 8-bit LUT	3.5
16:1 multiplexor	2
k-bit RAC (ROM acumulador)	
k-bit registro	-
12-bit acarreo	4

El retardo del proceso de generación de productos parciales también es una etapa rápida. En el diseño basado en multiplexores, el retardo que introducen para seleccionar el valor correcto de sus entradas es de 4 τ_t. En el caso de su generación directa, es de tan sólo 2 τ_t.

En el árbol de reducción de los productos parciales, se obtiene un coste de 11 τ_t y en la reducción de los 24 sumandos y 13 τ_t para los 40 sumandos que produce el método directo. Finalmente, la suma CLA para sumandos de 16 bits conlleva un retardo de 8 τ_t.

Los resultados anteriores arrojan un coste total para el cálculo de cada F(u) de 23 τ_t.

Al igual que en la sección anterior, se incluyen los resultados de la implementación FPGA del diseño propuesto con el fin de validar los resultados obtenidos.

Tabla 6

Retardo en la Implementación FPGA

Circuito	Retardo (ns)
4:2 compresor	1.344
3:2 contador	0.882
5:2 contador	1.848
12-bit 16:1 multiplexor	1.068
16-bit sumador (CLA)	9.068
12-bit multiplicador	46.89
Acceso a constantes tablas consulta LUT	4.520
Productos parciales basados en registros	9.551
Reducción de productos parciales 24:2 (Fig. 7)	5.556
Generación productos parciales directa	1.068
Reducción productos parciales 40:2 (Fig. 9)	7.154
Total cálculo basado en registros	24.175
Total cálculo directo	17.291

Según los datos de nuestra simulación en la tarjeta programable, el método directo es más rápido debido a la menor complejidad que conlleva su implementación.

C. Potencia consumida

Los datos de potencia consumida de la implementación del método se muestran en la tabla VII.

Además de lo anterior, la simulación realizada arroja un consumo estático de 118.08 mW para alimentación de la tarjeta FPGA.

A continuación, en la próxima sección se comparan las estimaciones realizadas con las prestaciones de otros diseños conocidos con el fin de evaluar la bondad del diseño propuesto.

Tabla 7

Consumo Potencia Implementación FPGA

Circuito	Potencia(mW)
Productos parciales basados en registros	46
Reducción de productos parciales 24:2 (Fig. 7)	16
Generación productos parciales directa	17
Reducción productos parciales 40:2 (Fig. 9)	32
Total cálculo basado en registros	62
Total cálculo directo	50

6 Discusión y Comparativas

Debido a la naturaleza de las aplicaciones a las puede ir dirigido este algoritmo, es esencial evaluar y comparar las prestaciones con otros métodos. En este apartado se comparan las prestaciones del método propuesto con otros diseños revisados en el apartado III con el fin de evaluar la mejor manera de diseñar un método de alto rendimento para el cálculo de la DCT.

Para estimar los costes de otras propuestas se utiliza la información que muestra las tablas III and V sobre los costes temporales y el área necesaria para implementar los principales componentes que intervienen en los diseños. La forma en la que esos elementos se combinan para cada método DCT se obtiene del detalle explicado en las referencias correspondientes. La comparación entre circuitos ha sido tomada por los mejores diseños mencionados en publicaciones en el apartado de técnicas de diseño. Los resultados obtenidos se resumen en las tablas VIII and IX.

En este apartado no se ha realizado la comparación de las diferentes implementaciones en la tarjeta FPGA debido a que se ha preferido tomar los datos directamente de las referencias para no interferir con la implementación realizada y evitar distorsiones introducidas por las técnicas de optimización del compilador

VHDL.

A. Comparaciones de Área

Como primera arquitectura para la comparativa consideramos un diseño básico basado en la implementación directa del algoritmo DCT a partir de la ecuación (3). Según este esquema, los elementos necesarios para calcular cada F(u) consisten en una tabla LUT para almacenar las 7 constantes del conjunto C, 1 operador multiplicador y un sumador. Estos componentes utilizan un área conjunta de 237 τ_a.

Los diseños según la técnica de aritmética distribuida basada en ROM se componen principalmente de los bucles de acumulación de resultados (RAC – Acumulador ROM). La principal contribución al hardware requerido para la computación de la DCT mediante estos métodos reside en este componente. A su coste se debe añadir el hardware necesario para los registros de acumulación y los sumadores paralelos de la primera etapa necesitando el diseño un área entre los 25 $K\tau_a$ [11] de los métodos convencionales y los 2.56 $K\tau_a$ de las implementaciones hardware eficientes [20].

Las implementaciones basadas en sumadores DA no necesitan del componente ROM y por tanto se pueden implementar con menos silicio. Sin embargo necesitan más operaciones de suma para calcular los productos parciales de la formula [25]. Reutilizando los operadores de suma según su especificación se puede estimar un coste en torno a los $1.5K\tau_a$ [26]. Por otra parte, las implementaciones basadas en el algoritmo FGA minimizan la cantidad de operaciones de sumas y productos necesarios para el cálculo de la DCT, de este modo, las implementaciones basadas en el método de Loeffer requieren sólo 11 multiplicaciones y 29 sumas [8]. Teniendo en cuenta sólo los recursos necesarios para implementar estos elementos se obtiene una estimación del área de 237 τ_a para calcular la 1D DCT. En caso de una ejecución concurrente, serían necesarias 8 unidades de suma y 6 unidades de multiplicación para

realizar en paralelo todas las operaciones implicadas. Esta operatoria aritmética conlleva un coste de 1118 τ_a más la circuitería de control necesaria.

Para las propuestas FGA que sustituyen las multiplicaciones por rotaciones CORDIC, tenemos en cuenta sólo el menor número de iteraciones para obtener la precisión requerida. Además, en estas propuestas, se pospone el compensado final a la etapa de cuantización posterior al cálculo de la DCT. Sin embargo, para realizar la comparación en términos homogéneos, incluimos este cálculo al final de la operatoria.

Según los diseños de la bibliografía reseñados, se requieren entre 48 y 56 sumas de la precisión señalada para realizar los cálculos [13], [16], [44] con los desplazamientos correspondientes. Si la ejecución se realiza en serie será necesario un sumador, un procesador CORDIC y un multiplicador para el compensado final, lo que requiere de 349 τ_a de para el cálculo de toda la DCT. De ejecutarse en paralelo las operaciones, se necesitarán 8 sumas, 3 procesadores CORDIC y 6 multiplicadores, por lo que habría que disponer de 1614 τ_a.

Tabla 8
Comparación Coste en Área

Método	Área (τa)
Diseño básico directo	237
DA basada en ROM	2.5K – 25K
DA basada en sumadores	1.5K
Loeffer FGA*	237 – 1.1K
Loeffer CORDIC FGA*	349 – 1.6K
Propuesto basado registros	6.7 K
Propuesto cálculo directo	3.5 K

* 1D completa DCT

Los datos de la tabla anterior ponen de manifiesto que nuestra propuesta de cálculo directo requiere de una cantidad de recursos hardware comparable a las propuestas basadas en DA pero superior a los diseños FGA.

B. Comparaciones retardo

Se han normalizado e igualado los valores usando los retardos de los componentes homogéneos e independientes de la tecnología de medida. Los retardos corresponden a la parte crítica de los circuitos, y han sido determinados por la parte más lenta de estos.

El esquema básico necesita realizar 8 accesos a la tabla LUT, 8 multiplicaciones y 7 sumas para calcular cada F(u) de la DCT, estas operaciones dan un coste combinado de 1208 τ_t. Este dato justifica que es necesario dedicar esfuerzos a reducir su coste, especialmente en aquellas aplicaciones sujetas a restricciones temporales.

El retardo del camino de los diseños basados en aritmética distribuida reside en el bucle de acumulación de resultados (RAC – ROM Accumulador) obtenidos de las tablas LUT [18], [20]. La cantidad de iteraciones es lineal con la precisión de los operandos (12 bits), y en cada una de sus iteraciones es necesario realizar un acceso a datos y una suma. Según los costes asociados a estos componentes y, sin tener en cuenta retados asociados a circuitería de control de iteración ni a los desplazamientos necesarios, el coste para calcular cada F(u) de la DCT es de 94 τ_t.

En el caso de las arquitecturas DA basadas en sumadores, se requieren casi tres veces más operaciones de suma consecutivas que con el método convencional. Siendo éste el elemento que más aporta retardo final, se puede estimar un coste temporal de 282 τ_t para calcular cada F(u) de la DCT.

Para las implementaciones basadas en el algoritmo de Loeffer FGA se requieren 11 multiplicaciones y 29 sumas. En el caso de ejecutar estas operaciones en serie, se requiere un coste total de 518 τ_t. En caso de ejecutar en paralelo las operaciones según los grafos de flujo, aun acosta de emplear más hardware para duplicar las operadores suma y

multiplicación, es posible ejecutar el cálculo completo con tan sólo 2 multiplicaciones y 4 sumas [8], lo que tiene una estimación de tiempo de 84 τ_t.

En caso de usar implementaciones CORDIC para la ejecución de las multiplicaciones e incluyendo los compensados de la etapa final se requiere un tiempo de ejecución de 620 τ_t (464 τ_t sin la compensación de las multiplicaciones). Con los recursos hardware necesarios para una ejecución concurrente sólo serían necesarios 146 τ_t (120 τ_t sin la compensación de las multiplicaciones).

Tabla 9
Comparación Coste en Retardo Estimado

Método	Retardo (τt)
Diseño básico directo	1208
DA basada en ROM	94
DA basada en sumadores	282
Loeffer FGA*	84 – 518
Loeffer CORDIC FGA*	146 (120) – 620 (464)
Propuesto basado registros	23
Propuesto cálculo directo	23

* 1D completa DCT

Los resultados de las estimaciones realizadas muestran que la propuesta descrita en este trabajo reduce de manera significativa el tiempo de ejecución empleado por los diseños basados en DA y se mantiene en unos niveles comparables a los alcanzados por los métodos FGA.

Además de los anterior, nuestra propuesta, al igual que las basadas en DA, permite una implementación más regular al calcular todos los componentes de la 1D DCT de manera homogénea y permite por tanto, concebir diseños segmentados que podrían reducir el cálculo combinado.

7 Conclusiones

En este trabajo se ha propuesto un método de cálculo para la Transformada Discreta del Coseno a partir de su formulación directa. La arquitectura dispone una estructura compacta para realizar las operaciones que, a diferencia de los métodos basados en DA, no requiere de tablas ROM para las multiplicaciones.

Hemos comprobado mediante la implementación VLSI en simulaciones de circuitos FPGA podemos fácilmente cumplir los requisitos aplicaciones de alto rendimiento. En comparación con el diseño existente, nuestra aproximación ofrece ventajas que pueden ser explotadas en cálculos de alto rendimiento. Los resultados de las pruebas y test realizados demuestran que sus prestaciones son comparables a las de los mejores métodos conocidos tanto los basados en DA como en FGA.

Como líneas de trabajo futuro, se está trabajando en extender la investigación a otros algoritmos como el DFT. De este modo sería posible construir unidades aritméticas para procesamiento de señales con un núcleo de primitivas con las mismas técnicas de cálculo.

Referencias

[1] K.R. Rao, J.J. Hwang, Techniques and Standards for Image, Video and Audio Coding. Englenwood Cliffs, NJ: Prentice –Hall, 1996.

[2] Netflix, Inc, URL: http://www.netflix.com, 2014.

[3] Skype, Microsoft, URL: http://www.skype.com/, 2014.

[4] W. H. Chen, C. Smith, S. Fralick, A fast computational algorithm for the discrete cosine transform, IEEE Trans. Commun., vol. 25, pp. 1004-1009, 1977.

[5]Z. Wang, Fast algorithms for the discrete W transform and for the discrete fourier transform, IEEE Trans. Acoust. Speech. Signal Processing, vol. 32, pp. 803-816, 1984.

[6]B. Lee, A new Algorithm to Compute the Discrete Cosine Transform, IEEE Transaction on Acoustic, Speech and Signal Processing, Vol. 32, (6), pp. 1243-1245, 1984.

[7]M. Vetterli, J. Kovacevic, Wavelets and subband coding. Englewood Cliffs, NJ: Prentice-Hall, 1995.

[8] C. Loeffler, A. Lightenberg, G. S. Moschytz, Practical fast 1-D DCT algorithms with 11-multiplications,

Proc. of ICASSP, Glagow, vol. 2, pp. 988-991, 1989.
[9] B.G. Lee, A New Algorithm to Compute the Discrete Cosine Transform, IEEE Trans. On Acoustics, Speech, and Signal Processing, vol. 32, pp. 1243-1245, 1984.
[10] H.S. Hou, A Fast Recursive Algorithm for Computing the Discrete Cosine Transform," IEEE Trans. On Acoustics, Speech, and Signal Processing, vol. 35, pp. 1455-1461, 1987.
[11] S.A. White, Applications of distributed arithmetic to digital signal processing: a tutorial review, IEEE ASSP Magazine, Vol. 6, nº 3, pp. 4-19, 1989.
[12] El Aakif, M.; Belkouch, S.; Chabini, N.; Hassani, M.M. Low power and fast DCT architecture using multiplier-less method, Faible Tension Faible Consommation (FTFC), pp. 63 – 66, 2011.
[13] Z. Wu, J. Sha, Z. Wang, L. Li, M. Gao, An improved scaled DCT architecture, IEEE Transactions on Consumer Electronics, vol.55-2, pp. 685-689, 2009.
[14] T.D. Tran, The binDCT: fast multiplierless approximation of the DCT, IEEE Signal Process. Letters, vol. 7, pp. 141–144, 2000.
[15] J. Liang, T.D. Tran,: Fast multiplierless approximations of the DCT with the lifting scheme, IEEE Trans. Signal Process., vol. 49, nº 12, pp. 3032– 3044, 2001.
[16] C.-C. Sun, S.-J. Ruan, B. Heyne and J. Goetze, Low-power and high quality CORDIC-based Loeffler DCT for signal processing, IET Proc. Circuits, Devices & Systems, vol. 1, no. 6, pp. 453–461, 2007.
[17] H. Huang; L. Xiao, CORDIC Based Fast Radix-2 DCT Algorithm, IEEE Signal Processing Letters, vol. 20, nº. 5, pp. 483 - 486, 2013.
[18] Madisetti, A. N. Willson, A 100 MHz 2-D 8 x 8 DCT/IDCT Processor for HDTV Applications, IEEE Trans. On Circuits and Systems for Video Technology. Vol. 5, No. 4, pp. 158- 165. 1995.
[19] S.-F. Hsiao, W-R. Shiue, New Hardware-Efficient Algorithm and Architecture for the Computation of 2-D DCT on a Linear Systolic Array, IEEE International Conference on Acoustics, Speech, and Signal Processing, 1999.
[20] Y. Sungwook; E.E Swartziander, DCT implementation with distributed arithmetic, IEEE Transactions on Computers, Vol. 50, nº 9, pp. 985-991, 2001.
[21] H.-C. Chen, J.-I. Guo, T.-S. Chang, C.-W. Jen, Memory-Efficient Realization of Cyclic Convolution and its Application to Discrete Cosine Transform, IEEE Trans. Circuits Syst. Video Technol., vol. 15, no. 3, pp. 445–453. 2005.
[22] H. Mora Mora, J. Mora Pascual, JL Sánchez Romero, F. Pujol López, Partial product reduction based on look-up tables, VLSI Design Conference, 2006.
[23] M. Alam, W. Badawy, G. Jullien, New Time Distributed DCT Architecture for MPEG-4 Hardware Reference Model, IEEE Trans. Circuits Syst. Video Technol., vol. 15, no. 5, pp. 726–730, 2005.
[24] T-S Chang, C-S. Kung, C-W Jen, A simple processor core design for DCT/IDCT, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 10, No. 3, pp. 439-447, 2000.
[25] Shams, A.M., Chidanandan, A., Pan, W., Bayoumi, M.A., NEDA: a low-power high-performance DCT architecture, IEEE Transactions on Signal Processing, Vol. 54, No 3,pp. 955-964, 2006.
[26] Sharma, V.K.; Mahapatra, K.K.; Pati, U.C. An Efficient Distributed Arithmetic Based VLSI Architecture for DCT", International Conference on Devices and Communications, pp. 1-5, 2011.

[27] J. Xie, P. K. Meher, J. He, Hardware-Efficient Realization of Prime-Length DCT Based on Distributed Arithmetic, IEEE Transactions on Computers, vol. 62, no. 6, pp. 1170-1178, 2013.

[28] Kinane, N. O'Connor, Energy-efficient Hardware Accelerators for the SA-DCT and Its Inverse, Journal of VLSI Signal Processing, Vol. 47, pp. 127-152, 2007.

[29] V. Srinivasan, K.J.R. Liu, VLSI design of high-speed time-recursive 2-D DCT/IDCT processor for video applications, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 6 (1), pp. 87-96, 1996.

[30] D. Slawecki and W. Li, DCT/IDCT processor design for high-data rate image coding, IEEE Trans. Circuits Syst. Video Technol., Vol. 2, pp.135-146 1992.

[31] P. Subramanian, A.S.C. Reddy, VLSI Implementation of fully pipelined multiplierless 2D DCT/IDCT architecture for JPEG, IEEE International Conference on Signal Processing (ICSP), pp. 401-404, 2010.

[32] V. G. Oklobdzija, D. Villeger, Improving multiplier design by using improved column compression tree and optimized final adder in CMOS technology, IEEE Transactions on Very Large Scale Integration Systems, vol. 3, no. 2, pp. 292–301, 1995.

[33] R. Rajsuman, Design and test of large embedded memories: an overview, IEEE Design and Test of Computers 18 (3), pp.16-27, 2001.

[34] W. Yeh, C. Jen, High speed booth encoded parallel multiplier design, IEEE Trans. Comput. 49 (7), pp. 692–700, 2000.

[35] IEEE Std. 1180-1990, IEEE standard specification for the implementation of 8x8 inverse cosine transform. Institute of Electrical and Electronics Engineers, New York USA, International Standard 1990.

[36] S. S. Mishra, A. K. Agrawal, R.K. Nagaria, A comparative performance analysis of various CMOS design techniques for XOR and XNOR circuits, International Journal on Emerging Technologies, vol. 1, nº 1, 2010.

[37] P. Balasubramanian, D.A. Edwards, Power, delay and area efficient self-timed multiplexer and demultiplexer designs, International Conference on Design & Technology of Integrated Systems in Nanoscal Era, pp. 173-178, 2009.

[38] R. Menon, D. Radhakrishnan, High performance 5 : 2 compressor architectures, IEE Proc. Circuits Devices Syst., Vol. 153, No. 5, 2006.

[39] M.A Song et al., A low-error and area-time efficient fixed-width booth multiplier, IEEE Midwest Symposium on Circuits and Systems, pp.590-593, 2003.

[40] B. Phillips, N. Burgess, Minimal weight digit set conversions, IEEE Transactions on Computers 53 (6), pp. 666–677, 2004.

[41] G. A. Ruiz, M. Granda, Efficient canonic signed digit recoding, Microelectronics Journal, no. 42, pp. 1090-1097, 2011.

[42] R. Zimmermann, Binary adder architectures for cell-based VLSI and their synthesis, Ph.D. Thesis, Swiss Federal Institute of Technology, 1997.

[43] M. R. Pillmeier, M. J.Schulte; E.G. Walters, Design alternatives for barrel shifters, Advanced Signal Processing Algorithms, Architectures, and Implementations Proceedings of the SPIE, Volume 4791, pp. 436-447, 2002.

[44] Y. Sungwook, E.E. Swartzlander, Jr. A scaled DCT architecture with the CORDIC algorithm, IEEE Transactions on Signal Processing, vol. 50 (1), pp. 160-167, 2002.
[45] MT Signes, JM García, H Mora, Improvement of the Discrete Cosine Transform calculation by means of a recursive method, Mathematical and Computer Modelling 50 (5), 750-764, 2009.
[46] FA Pujol, H Mora, JL Sánchez, A Jimeno, EZW-based image compression with omission and restoration of wavelet subbands, Progress in Pattern Recognition, Image Analysis and Applications, 134-141, 2007.

Apéndice A: Entorno experimental

El diseño hardware, técnicas de simulación y su implementación se ha realizado en sistemas de tarjetas reconfigurables FPGAs, los cuales proporcionan diseños válidos y una alta productividad en el desarrollo de sistemas hardware. Este tipo de dispositivos tienen una amplia aceptación y son frecuentemente usados por la comunidad científica. Los resultados obtenidos permiten comprobar las estimaciones teóricas y la corrección de los diseños.

En este trabajo, el dispositivo FPGA utilizado es el *xc3sd1800a* perteneciente a la familia *Xilinx Spartan Spartan3adsp* que ha sido optimizado para el procesamiento de señales digitales por el sistema de integración (http://www.xilinx.com/products/silicon-devices/fpga/xa-spartan-3a-dsp/). La herramienta de síntesis utilizada es *Xilinks: ISE Design Suite 14.7 (Webpack license)* y el lenguaje de programación empleado VHDL. La información sobre potencia consumida se ha obtenido de la herramienta Xilink XPower Estimator (XPE) (http://www.xilinx.com/support/ documentation/user_guides/ug440.pdf).

Las implementaciones de los circuitos se han basado en diseños conocidos referenciados en la literatura, y se ha utilizado para los circuitos estándar códigos VHDL estándar del Dept. of Computer Science & Engineering, College of Engineering, University of California, Riverside. (http://esd.cs.ucr.edu/labs/tutorial/).