PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

DISEÑO DE LA ARQUITECTURA DE TRANSFORMADA DISCRETA DIRECTA E INVERSA DEL COSENO PARA UN DECODIFICADOR HEVC

Tesis para optar el Título de Ingeniero Electrónico, que presenta el bachiller:

Marco Antonio Portocarrero Rodriguez

ASESORES: MSc. Ing. Ernesto Cristopher Villegas Castillo, MSc. Ing. Mario Andrés Raffo Jara

Lima, 2018

A mis padres, Marco y Marina, por apoyarme en todas mis decisiones ydarme todo su cariño en todo momento.

A mi hermano César, que me motiva a mejorar y también a disfrutar más de la vida.

A los miembros del laboratorio de microelectrónica, que inspiran a uno a ser mejor cada día, pero también con los que uno puede reírse de vez en cuando.

Al MSc. Ing. Cristopher y MSc. Ing. Mario Raffo por su gran ayuda y disposición para enseñarme en cualquier momento

A cada uno de ellos, Muchas Gracias

Resumen

El empleo de video de alta resolución es una actividad muy común en la actualidad, debido a la existencia de dispositivos portátiles capaces de reproducir y crear secuencias de video, ya sea en HD o en resoluciones mayores, como 4k u 8k. Sin embargo, debido a que las secuencias de video de mayor resolución pueden llegar a ocupar grandes espacios de memoria, estas no pueden ser almacenadas sin antes realizar un proceso de compresión.

Organizaciones especializadas como ITU-T Coding Experts Group e ISO/IEC Moving Picture Experts Group, han sido responsables del desarrollo de estándares de codificación de video. De esta manera, para mejorar la transmisión de video y poder obtener resoluciones cada vez mayores, se llevó a cabo el desarrollo del estándar de codificación HEVC o H.265, el cual es el sucesor al estándar H.264/AVC.

El presente trabajo de tesis está centrado en el módulo de Transformada Discreta e Inversa del Coseno (DCT e IDCT), el cual forma parte del estándar HEVC y su función es hallar los coeficientes en el dominio de la frecuencia de muestras, para poder cuantificarlas y reducir su número.

Se realizó el diseño la arquitectura, tomando en consideración la capacidad de procesamiento de pixeles requerida por el estándar, la frecuencia de operación del circuito y la cantidad de recursos lógicos usados. La arquitectura fue descrita en el lenguaje Verilog HDL y fue sintetizada para dispositivos Zynq – 7000 de la empresa Xilinx. La verificación funcional del circuito fue realizada mediante el uso de Testbenchs en el software ModelSim.

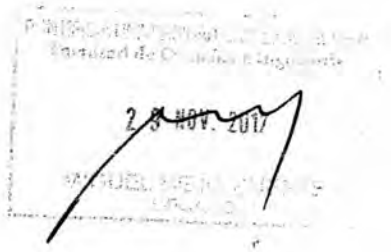
Para verificar el funcionamiento de la arquitectura diseñada, se utilizó el software MATLAB para obtener los resultados esperados y se compararon con los obtenidos en la simulación funcional del circuito. La frecuencia máxima de operación fue hallada mediante la síntesis de la arquitectura, la cual llegó a ser de 135 MHz, que es equivalente al procesamiento de secuencias de video de resolución 4k o 3840x2160 pixeles a 65 fps.

FACULTAD DE
CIENCIAS E INGENIERÍA

PONTIFICIA **UNIVERSIDAD CATÓLICA** DEL PERÚ

TEMA DE TESIS PARA OPTAR EL TÍTULO DE INGENIERO ELECTRÓNICO

Título	: Diseño de la arquitectura de Transformada Discreta Directa e Inversa del coseno para un decodificador HEVC
Área	: Circuitos y Sistemas #1410
Asesor	: MSc Ing. Ernesto Cristopher Villegas Castillo
	MSc. Ing. Mario Andrés Raffo Jara
Alumno	: Marco Antonio Portocarrero Rodriguez
Código	: 20130424
Fecha	: 23/11/17



Descripción y Objetivos

El empleo de video digital en alta resolución es en la actualidad una característica muy común en dispositivos portátiles, especialmente en teléfonos celulares. Las resoluciones de video más utilizadas son HD (1280x720 píxeles), Full HD (1920x1080), y últimamente las resoluciones en Ultra Alta Definición como 4k (3840x2160) u 8k (7680x4320). Sin embargo, debido a que los videos en estas resoluciones poseen una cantidad de datos en el orden de gigabytes, no existen medios de almacenamiento portátiles de suficiente capacidad para estos, asimismo su transmisión a través de canales digitales como 4G o Wi-fi está limitada por el ancho de banda. Para poder reducir esta carga existen diversos formatos de compresión que se vienen desarrollando continuamente para mejorar la tasa de compresión; una de las últimas versiones es el estándar HEVC (High Efficiency Video Coding) conocido también como H.265.

El objetivo principal del presente trabajo de tesis es el diseño de las arquitecturas de Transformada Discreta del Coseno (DCT) y de Transformada Inversa del Coseno (IDCT) para el estándar HEVC. Los tamaños de las Unidades de Transformada (TU) que debe procesar son: 4x4, 8x8, 16x16 y 32x32 píxeles.

Los objetivos específicos son los siguientes:

1) Emplear el lenguaje Verilog HDL para la descripción de la arquitectura conjunta de DCT e IDCT para los tamaños de TU de 4x4, 8x8, 16x16 y 32x32.

2) Verificar funcionalmente la arquitectura diseñada

3) Sintetizar la arquitectura y optimizar la arquitectura para obtener la máxima frecuencia de operación para procesamiento de secuencias 4k en tiempo real.

MÁXIMO 50 PÁGINAS

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

M. Sc. Ing. WILLY CARRERA SORIA
Coordinador de la Especialidad de Ingeniería Electrónica

FACULTAD DE
CIENCIAS E INGENIERÍA

PONTIFICIA **UNIVERSIDAD CATÓLICA** DEL PERÚ

TEMA DE TESIS PARA OPTAR EL TÍTULO DE INGENIERO ELECTRÓNICO

Título : Diseño de la arquitectura de Transformada Discreta Directa e Inversa del coseno para un decodificador HEVC

Índice

Introducción

1. Estándar de compresión HEVC

2. Fundamentos teóricos de las transformadas DCT e IDCT para el estándar HEVC

3. Diseño de la arquitectura de hardware del módulo de DCT e IDCT

4. Resultados

Conclusiones

Recomendaciones

Bibliografía

Anexos

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

M. Sc. Ing. WILLY CARRERA SORIA
Coordinador de la Especialidad de Ingeniería Electrónica

Índice

Índice de figuras

Índice de tablas

Introducción

El empleo de video digital en alta resolución (en inglés, “High Definition” o “HD”) es una característica muy común en dispositivos portátiles actuales, especialmente en teléfonos celulares. Las resoluciones de video más utilizadas son HD (1280x720 píxeles), Full HD (1920x1080) y últimamente las resoluciones en Ultra Alta Definición (en inglés, “Ultra High Definition” o “UHD”) como 4k (3840x2160) o 8k (7680x4320). Sin embargo, debido a que las secuencias de video en estas resoluciones poseen una alta cantidad de datos, requieren grandes espacios de memoria que pocos dispositivos portátiles poseen. Asimismo, su transmisión a través de canales digitales como 4G o Wi-fi es limitada por el ancho de banda, por lo que será necesario comprimir su información. Además, en la actualidad, existe demanda por soluciones que requieren video digital en tiempo real, como sistemas de vigilancia o servicios de conversación online. Por estas razones, la transmisión de video es la carga más crítica en redes de comunicación actuales [1].

Para poder reducir esta carga, el Grupo de Expertos en Codificación de Video de la Unión Internacional de Telecomunicaciones (“Internacional Telecomunications Unit Video Coding Experts Group, ITU-T VCEG”) y el Grupo de Expertos en Imágenes en Movimiento de la Organización Internacional de Estandarización (“International Standarization Organization for Standarization Moving Picture Experts Group – ISO/IEC MPEG”) formaron el Equipo de Colaboración Conjunta en Codificación de Video (en inglés, “Joint Colaborative Team on Video Coding” – JCT-VC) para desarrollar los estándares de codificación como el HEVC (High Efficiency Video Coding).

El propósito de la codificación HEVC es resolver el problema de la cantidad de información necesaria para representar una secuencia de imágenes digitales. Desde un

punto de vista matemático, este proceso se refiere a transformar la matriz 2D de pixeles en un grupo de datos totalmente no correlacionados. La transformación es aplicada antes del almacenamiento o transmisión de la secuencia de imágenes. En algún momento siguiente, las imágenes comprimidas son descomprimidas para reconstruir la secuencia original o una aproximación.

Capítulo 1

Estándar de compresión HEVC

1.1 Presentación de la problemática

Durante los últimos años, el acceso a señales de video ha cambiado cada vez más a medios digitales y los medios analógicos están retirandose del mercado poco a poco, lo que es conocido como el ”apagón analógico”. En Perú, la implementación Televisión Digital Terrestre (TDT) se está realizando progresivamente y ,según el Informe Anual de Implementación de la TDT del año 2017 [2], se planea que a partir del año 2020 se dará el fin de transmisiones análogicas de televisión en Lima y Callao. Este cambio indica que, a nivel local, la necesidad de tecnologías de video digital va a aumentar significativamente y pasar a ser el nuevo estándar para todos los sectores de la población.

Por otro lado, el acceso a video de cada vez más calidad ha causado que se necesite reducir la información que se transmite en cada video para poder proveer la mejor calidad, en especial en aplicaciones en tiempo real (grabaciones de seguridad y transmisión de eventos en vivo, por ejemplo). Algunos formatos de compresión de video que se han usado son AVC o MPEG-2[1]. Uno de los últimos formatos de compresión es el HEVC (en inglés “High Efficiency Video Coding”) o H.265. En las siguientes líneas se describirá sus principales características y su funcionamiento de acuerdo a la referencia [1].

1.2 Conceptos fundamentales sobre codificación de video

Una secuencia de video está formada por una serie de imágenes que se reproducen a una alta velocidad para dar la ilusión de movimiento al ojo humano. Estas imágenes son formadas a partir de las señales obtenidas de los sensores de una cámara. Para poder representar los colores de un cuadro se pueden utilizar diferentes métodos, de los cuales los más usados son la representación RGB y YCbCr. La representación RGB representa los colores mediante una combinación de intensidades de los colores rojo, verde y azul ("Red", "Green" y "Blue" en inglés) como se puede ver en la Figura 1.1. Por otro lado, el espacio de color YCbCr permite representar cada cuadro mediante sus componentes de luminancia (Y) y cromancia (Cb y Cr), como se puede ver en la Figura 1.2. La codificación HEVC utiliza la representación YCbCr debido a que los componentes Cb y Cr se pueden representar con resolución menor que el componente Y, ya que la visión humana es menos sensible a cambios de color que a cambios de luminancia. Esto permite reducir la cantidad de información requerida para representar la cromancia de un cuadro [3].

Figura 1.1: Imagen a color separada en sus componentes R (rojo), G (verde) y B (azul) [4]

Figura 1.2: Imagen a color separada en sus componentes de luminancia (Y) y cromancia (Cb y Cr) [4]

Existen diferentes tipos de muestreo para conseguir los componentes Y, Cb y Cr. Tres de ellos son 4:4:4, 4:2:2 y 4:2:0. Para los dos primeros, los valores indican el número de muestras relativas de cada componente. Para el muestreo 4:4:4, cada 4 muestras del componente Y se tienen 4 muestras de Cb y 4 muestras de Cr, lo cual indica que se tiene la resolución completa de cromancia. Para el muestro 4:2:2, cada 4 muestras Y se tiene 2 muestras de Cb y 2 muestras Cr. Por otro lado, el muestreo 4:2:0 significa que los componentes Cb y Cr tienen la mitad de muestras horizontales y verticales que Y. Los valores del muestreo 4:2:0 no tienen significado lógico y su nombre se eligió históricamente para diferenciarse de 4:4:4 y 4:2:2.[5]

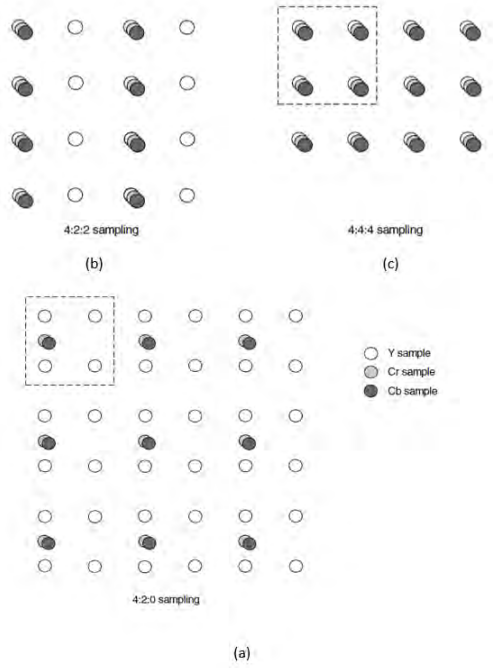


Figura 1.3: Diferentes tipos de muestreo. (a) representa el muestreo 4:2:0; (b), el muestreo 4:2:2 ;y (c), el muestreo 4:4:4 [5]

1.3 Funcionamiento del estándar HEVC

El estándar HEVC está basado en la partición de cada cuadro en bloques de menor tamaño. Estos bloques son los que son procesados y no los cuadros enteros, lo que permite reducir la complejidad computacional del proceso.

Cada imagen es dividida en unidades llamadas ”Coding Tree Units” (CTU). Cada CTU está a su vez formada por bloques CTB (”Coding Tree Block”), uno por cada componente de color asociada a cada CTU (Figura 1.4). A su vez, un CTB puede ser dividido en CB (“Coding Blocks”) porque los CTB pueden ser muy grandes para poder realizar las tareas de predicción inter-imagen o intra-imagen. Estos procesos se encargan de generar datos referentes al movimiento relativo entre cuadro, comprimir un cuadro de imagen usando la información del mismo cuadro (intra-predicción) y cuadros de imágenes posteriores (inter-predicción).

Un CB puede ser aún demasiado grande para guardar la información de vectores de movimiento, por lo que estos se pueden dividir en bloques más pequeños llamados PB (“Prediction Block”). Finalmente, una vez que la predicción se ha realizado, se necesita codificar el residuo entre la imagen predicha y la imagen original mediante una tranformación. Para lograr esto, se divide un CB en bloques llamados TB (”Transform Block”) [6]. Estas últimas dos divisiones se pueden ver en la Figura 1.5.

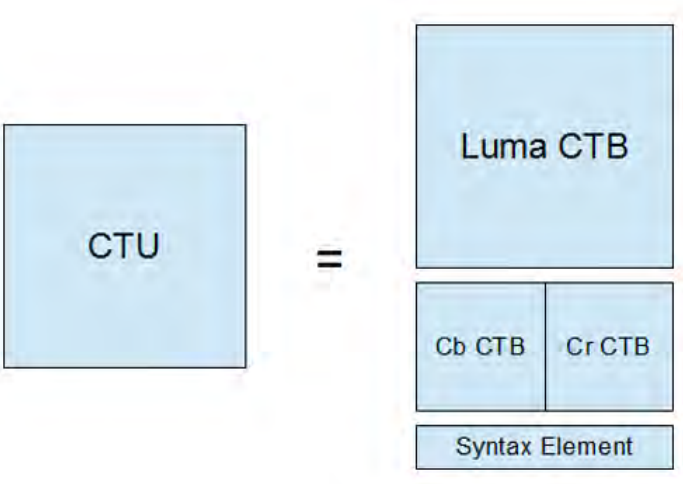


Figura 1.4: División de un ”Coding Tree Unit” en ”Coding Tree Block” para sus componentes de luminancia y cromancia [6]

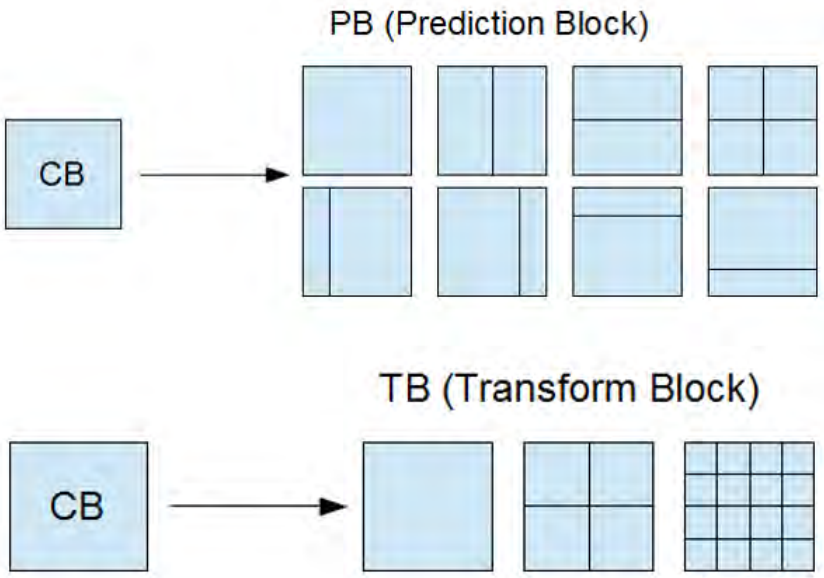


Figura 1.5: Divisiones posibles de un Coding block [6]

La división mencionada anteriormente es necesaria para el procesamiento que se necesita la compresión y descompresión de video. En la Figura 1.6 se tiene un esquema general de todas las partes del codificador del HEVC. Las cuatro principales partes del codificador han sido resaltadas a continuación se describirá su función. En el bloque de Predicción Intra-imagen e Inter-imagen (en inglés “Intra-picture prediction” e “Inter-picture prediction”) se utiliza la información de una misma imagen para predecir los diferentes valores de ella misma (Intra-imagen). Además, se utiliza información de imágenes previas para predecir la información de las siguientes (Inter-imagen). El objetivo es reducir la cantidad de datos para necesaria para representar cada cuadro ya que solo sería necesario almacenar y transmitir la diferencia entre entre cuadros diferentes junto con información adicional para reconstruir una secuencia codificada.

Luego de la etapa de prediccion, la información puede ser comprimida aún más en el bloque de Transformación, escalamiento y cuantización (en inglés “Transform, Scaling & Quantization”). En este bloque toma el residuo de la imagen de la parte de predicción y la imagen real y se le aplica la transformación discreta del coseno (DCT), con lo cual se puede representar los valores mediante sus coeficientes en frecuencia. Se pueden establecer niveles de cuantización para eliminar los coeficientes de menor impacto en la imagen, con lo cual se reduciría aún más la información de una secuencia

de video.

Para poder eliminar posibles artefactos que se pueden generar debido al efecto de la codificación de bloques se utiliza el Control de filtros (en inglés “Filter Control Analysis”). En esta parte se utilizan dos filtros: “deblocking filter” y “Sample adaptive offset” (SAO). Estos filtros son usados para suavizar las divisiones de los bloques y codificación entrópica (en inglés ”Header formatting & CABAC”). En esta parte se junta toda la información de las previas etapas, se agreans las cabeceras necesarias y se codifica de manera entrópica utilizando el algoritmo CABAC (en inglés “Context Adaptive Binary Arithmetic Coder”).

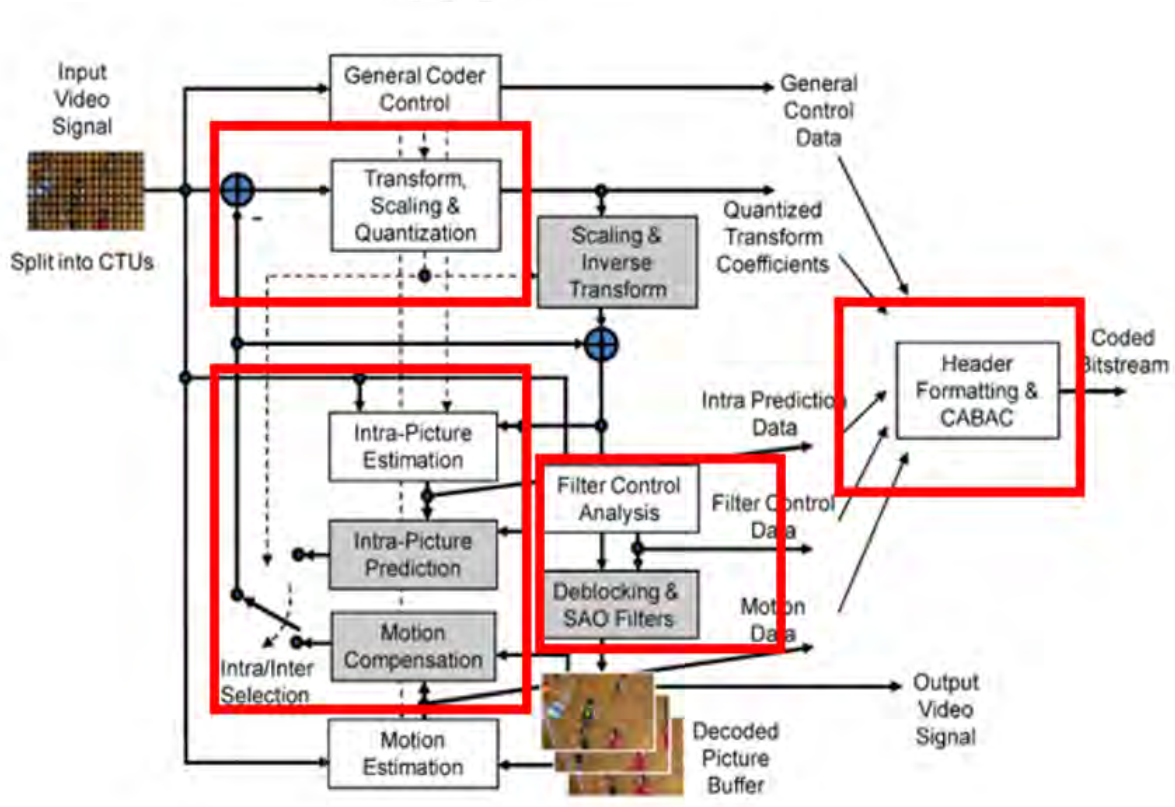


Figura 1.6: Diagrama de flujo del proceso de codificación HEVC [1]

Por otro lado, el decodificador HEVC tiene gran parecido con el codificador, salvo que este tiene sus partes en el orden necesario para obtener la información codificada. El decodificador se puede ver en la Figura 1.7, donde sus partes importantes han sido resaltadas.

Se puede ver que el primer bloque es para decodificar la parte entrópica (en inglés

"Entropy Decoder"), el cual es la parte final de la codificación. Luego se tiene el bloque de transformación inversa (en inglés "Inverse Transform") el cual realiza la transformación inversa de la DCT de la codificación. Finalmente, se tiene lo bloque de predicción y de filtrado (en inglés "Prediction" e "In-loop filters") los cuales invierten el proceso de predicción para obtener las imágenes completas y filtran cada imagen para reducir el efecto de los bloques anteriores, respectivamente.

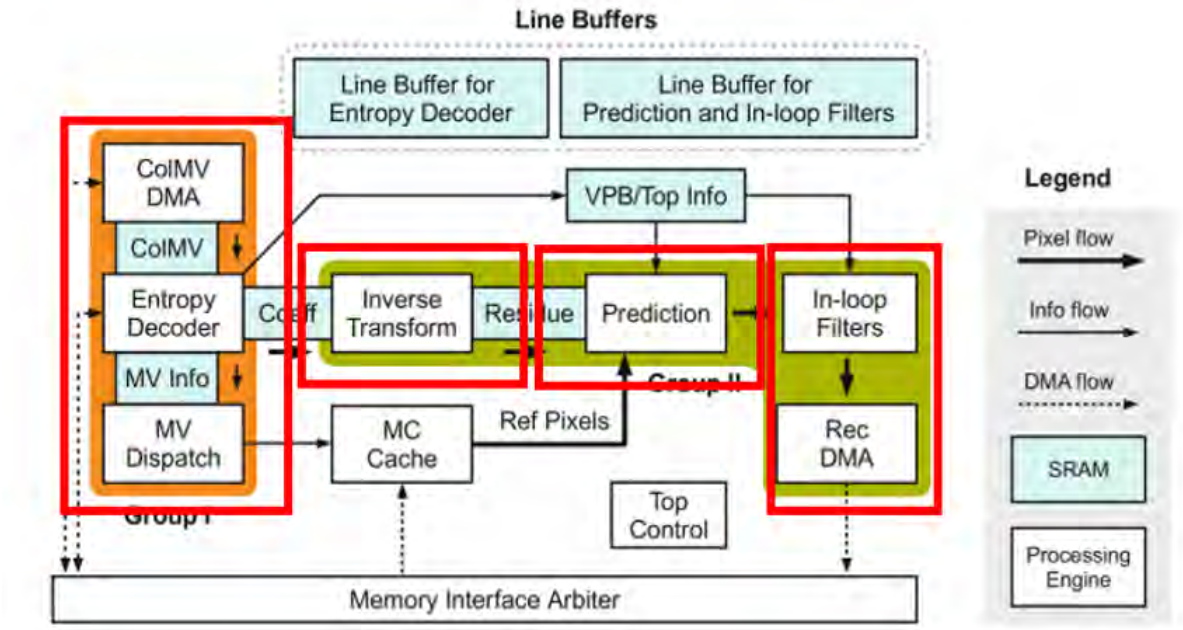


Figura 1.7: Diagrama de bloque de decodificación HEVC [1]

En síntesis, el proceso de compresión consiste en generar un bitstream en el cual cada imagen es dividida en Coding Tree Units (CTU), las cuales son procesadas por bloques de inter-predicción e intra-predicción. Estos módulos de procesamiento se encargan de generar datos referentes al movimiento relativo entre cuadro, lo que permitirá comprimir un cuadro de imagen usando la información del mismo cuadro (intra-predicción) y cuadros de imágenes posteriores (inter-predicción).

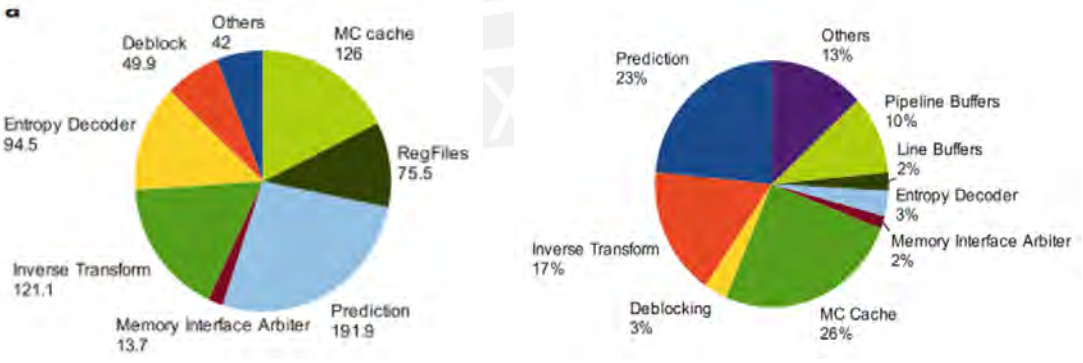
Luego de este proceso, la diferencia entre el bloque de píxeles original y el predicho es procesada por el módulo de transformación lineal conocida como Transformada Discreta Coseno (DCT, del inglés Discrete Cosine Transform).

La DCT consiste en procesar un bloque de Transformación o TB derivado del CU residual transformándolos en coeficientes [1]. Finalmente, los coeficientes de la

transformación son escalados (se multiplican por constantes), cuantizados (se aproximan a valores predefinidos separados por una distancia fija) y codificados entrópicamente (se usa la codificación CABAC para codificar los datos en base a su probabilidad de ocurrencia) para ser transmitidos junto con la información de predicción [1].

1.4 Resultados de implementación de decodificador HEVC

En [1], Sze V. et. al. se muestran los resultados de la implementación en ASIC de un decodificador HEVC en términos de área de silicio utilizada (kgates) y potencia consumida por módulo de procesamiento, tal como presenta la Figura 1.8. Se puede observar que los módulos que ocupan mayor espacio de hardware son: la transformada inversa, el módulo de predicción y el módulo de decodificación entrópica, mientras que los módulos que consumen más potencia son: el módulo de predicción y la transformada inversa, ya que consumen el 40% de la potencia total (17% transformada inversa, y 23% predicción). Debido a estos resultados, los puntos clave para mejorar la eficiencia del decodificador son: la transformada inversa, predicción y decodificación entrópica.



(a) (b)

Figura 1.8: Consumo de lógica (a) y de potencia (b) del decodificador HEVC [1]

En el grupo de Microelectrónica de la Pontificia Universidad Católica del Perú (PUCP) se han realizado trabajos pasados relacionados a la compresión de video. En

el estándar de compresión H.264, se tiene el trabajo de Villegas [7] en el que se utilizó el algoritmo de Estimación de Movimiento Fraccional con precisión Quarter-Pixel para el módulo de Predicción de Movimiento. Por otro lado, en Soto [8] también se trabajó en el mismo módulo que Villegas, pero esta vez fue en el estándar H.265/HEVC, lo cual implicó mayor complejidad en el diseño. Estos trabajos muestran que existe una línea de trabajo relacionada al diseño de módulos dentro de los estándares de compresión de video. El trabajo actual también pertenece a esta línea de trabajo y se concentra en el diseño de una arquitectura para la parte de transformación discreta del coseno. El estándar HEVC no especifica un diseño específico de codificador o decodificador, solo describe el comportamiento que estos tienen que tener y las entradas y salidas de cada bloque que los conforma. Debido a esto, existe la libertad para innovar en el diseño de cada parte del codificador y del decodificador manteniendo la compatibilidad con las demás partes. Asimismo, existen diversas propuestas de arquitecturas para la Transformada Discreta Coseno y su inversa (DCT e IDCT), para una gran variedad de aplicaciones y entornos, siempre que estos puedan cumplir con las especificaciones del estándar.

1.5 Etapa DCT del estándar HEVC

En [3] se explica que la etapa de transformación de un sistema de compresión de imágenes consiste en mapear una imagen en un grupo de coeficientes de transformación, los cuales luego son cuantizados y codificados. Para la mayoría de imágenes, un gran número de coeficientes tendrán magnitudes bajas, los cuales pueden ser cuantizados o simplemente ser descartados con una distorsión mínima de la imagen original. Un ejemplo de esto se puede ver en la Figura 1.9. Para el estándar HEVC se seleccionó la transformada discreta del coseno (DCT) porque, tal como se explica en [9], no necesita números complejos para su cálculo y además necesita menos coeficientes que la transformada discreta de Fourier (DFT) para obtener una buena aproximación de una señal, lo cual la DCT está mejor equipada para modelar funciones de longitud limitada.

La transformada discreta del coseno (DCT) para una secuencia de N muestras u[j]

está dada por

$$F[i] = \sum_{j=1}^{N-1} f[j] * \frac{P}{\sqrt{N}} * cos(\frac{\pi}{N} * (j + \frac{1}{2}) * i) \tag{1.1}$$

Donde $P = 1$ dado $i = 0$ y $P = \sqrt{2}$ dado $i > 0$.

El principal uso de la DCT es poder compactar la energía de una señal en un número limitado de coeficientes La transformación inversa del coseno permite restaurar la señal original usando los coeficientes. La transformada inversa de para una secuencia de N muestras F[i] está dada por

$$f[j] = \sum_{i=1}^{N-1} F[i] * \frac{C}{\sqrt{N}} * cos(\frac{\pi}{N} * (j + \frac{1}{2}) * i) \tag{1.2}$$

Donde $C = \frac{1}{\sqrt{2}}$ dado $j = 0$ y $C = 1$ dados otros valores de j.

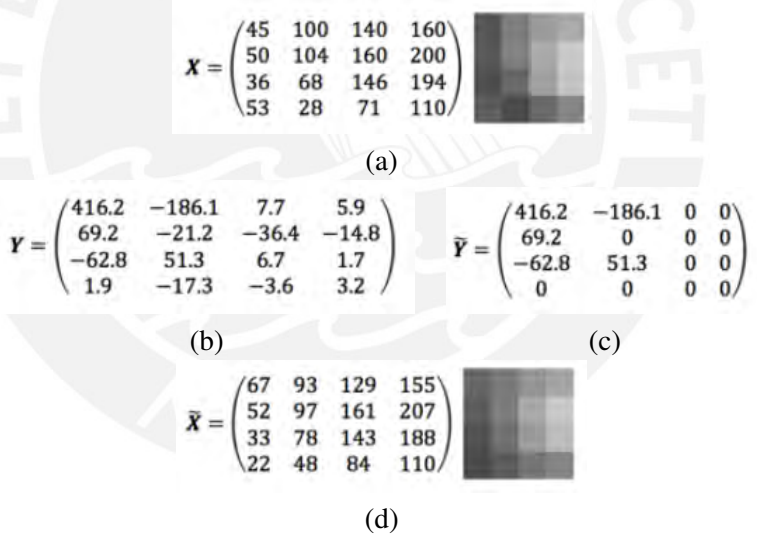


Figura 1.9: En (a) se tiene un ejemplo de una matriz 4x4 de entrada para un sistema de DCT. En (b) se tiene los coeficientes de (a) luego de realizar la transformada. En (c) se han eliminado los coeficientes de menor valor de (b). Finalmente, en (d) se ha formado una nueva imagen ven base a los coeficientes en (c). Se puede ver que no hay mucha diferencia entre (a) y (d) luego de la eliminación de los coeficientes de menor valor.

Existen ciertas propiedades de la DCT que la hacen útil para la compresión de imágenes y para su implementación, las cuales son:

1. Ortogonalidad: Los vectores base c son ortogonales. Por ejemplo, dado

$$c_{ij} = \frac{P}{\sqrt{N}} * cos(\frac{\pi}{N} * (j + \frac{1}{2}) * i) \tag{1.3}$$

   Se cumple que

$$c_i^T c_j = 0, i \neq j \tag{1.4}$$

   Lo que permite la representación de imágenes usando la suma de los productos con c_{ij}

2. Compactación de energía: Los vectores base de la DCT proveen una buena compactación de energía [7].

3. Reutilización de matriz de coeficientes: Los elementos de la matriz DCT pueden ser derivados de una matriz más grande, lo que puede utilizarse para reducir costos de implementación.

Las matrices de transformación de la codificación HEVC son aproximaciones finitas de la matriz original DCT. La matriz original c de tamaño 4x4

$$Real(c) = \begin{bmatrix} 0.5000 & 0.5000 & 0.5000 & 0.5000 \\ 0.6533 & 0.2705 & -0.2706 & -0.6533 \\ 0.5000 & -0.5000 & -0.5000 & 0.5000 \\ 0.2706 & -0.6533 & 0.6533 & -0.2706 \end{bmatrix} \tag{1.5}$$

Y una versión entera aproximada es

$$Approx(c) = \begin{bmatrix} 64 & 64 & 64 & 64 \\ 83 & 36 & -36 & -83 \\ 64 & -64 & -64 & 64 \\ 36 & -83 & 83 & -36 \end{bmatrix} \tag{1.6}$$

Los valores de la matriz aproximada fueron multiplicados por 128 para poder tener una versión entera, con la cual se facilita los cálculos. Además, algunos valores fueron modificados manualmente para cumplir mejor las propiedades de la DCT. Los valores finales son especificados en el estándar, por lo que no dependen de cada

implementación. Sin embargo, la aproximación causa que algunas propiedades orginales de la DCT no se cumplan por completo. Por lo tanto, existe un compromiso entre usar elementos de la matriz de transformación con alta cantidad de bits y el grado con el que se cumplen las propiedades mencionadas anteriormente en [1].

El valor de cada elemento de la matriz de transformación está representado con 8 bits (incluido el bit de signo) y se puede ver el total del lado izquierdo de la matriz de transformación de 32x32 en el Anexo 1. Los tamaños posibles de TU ("transform unit") son 4x4, 8x8, 16x16 y 32x32. Estos diferentes tamaños dependen de la división de cada cuadro, lo cual depende de la composición de cada uno: si un cuadro tiene áreas muy uniformes (como un cielo azul grande) se pueden usar divisiones grandes porque no hay mucha variación, pero si un cuadro tiene gran variación (muchas personas y colores) se necesita divisiones de menor tamaño para captar la mayor cantidad de detalles posicles. Los valores de las matrices de transformación de tamaños inferiores a 32x32 se puede extraer a partir de la matriz mostrada en el Anexo 1.

Como se explica en [10], con la proliferación de productos como cámaras, teléfonos y servicios como videoconferencia, es necesario que exista captura de video además de reproducción de video en un mismo dispositivo. Debido a esto, la transformada y su inversa necesitan ser implementado en un mismo dispositivo. En [10] se muestra un ejemplo de diseño en hardware que implementa la DCT y la IDCT que se describe en la Figura 1.10.

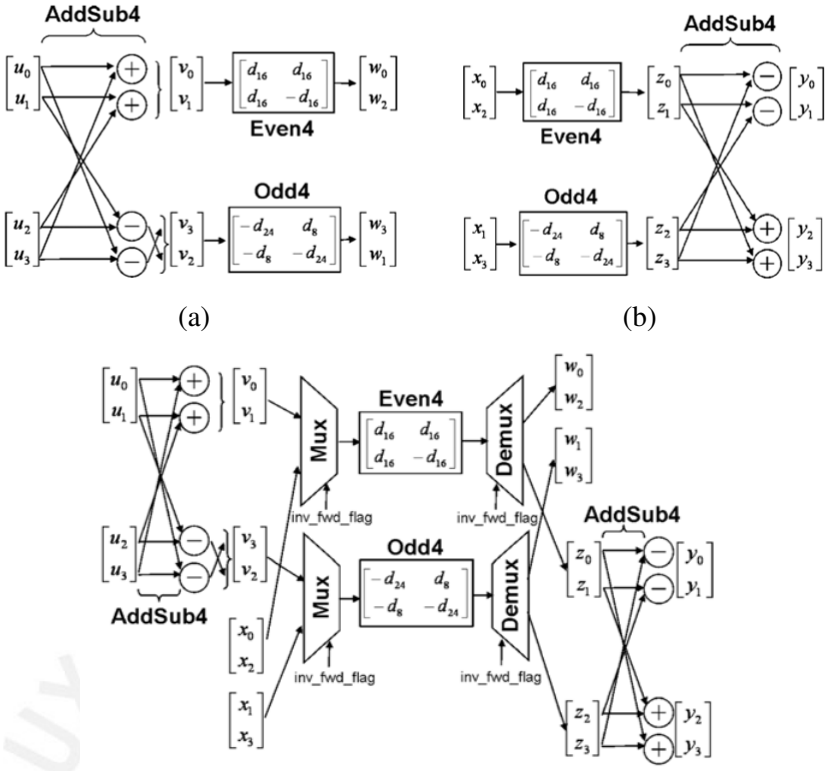


Figura 1.10: Ejemplo de diseño de DCT (a) e IDCT (b) dividiendo el proceso entre los datos pares e impares. En (c) se muestra la combinación de (a) y (b) donde se reutiliza la mayoría de sus elementos para ambas operaciones[10]

Para el cálculo de DCT e IDCT, [1] propone separar cada proceso en dos etapas: una primera para realizar la multiplicación de columnas y una segunda para realizar la multiplicación de filas. Además, para poder mantener un tamaño de 16 bits a la salida de cada etapa, se incluyen factores de escalamiento para la DCT que se dan por

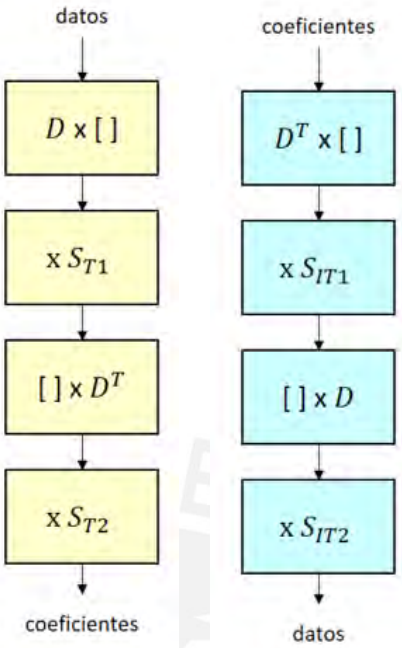
$$S_{T1} = 2^{-(B+M-9)}, S_{T2} = 2^{-(M+6)} \tag{1.7}$$

y para la IDCT

$$S_{IT1} = 2^{-6}, S_{IT2} = 2^{-(21-B)} \tag{1.8}$$

Donde B es el número de bits para representar los bits de profundidad usados para representar el video y $M = log_2(N)$, donde N es el tamaño de la transformada (4,8,16 o 32). Un resumen de las etapas de la DCT e IDCT con sus factores de escalamiento

se encuentra en la Figura 5.



(a) (b)

Figura 1.11: Ejemplo de diseño de DCT (a) e IDCT (b) dividiendo el proceso entre los datos pares e impares. En (c) se muestra la combinación de (a) y (b) donde se reutiliza la mayoría de sus elementos para ambas operaciones [15]

En el siguiente capítulo se muestran ejemplos de arquitecturas que explotan las propiedades del estándar para mejorar la implementación en diversas maneras.

Capítulo 2

Fundamentos teóricos de las transformadas DCT e IDCT para el estándar HEVC

2.1 Arquitecturas propuestas

En [11], se muestran las arquitecturas de un módulo de transformada inversa para los posibles tamaño de TU ("transform unit"), cuyos tamaños pueden ser: 4x4, 8x8, 16x16, 32x32. Se describe que la transformada para una matriz de MxN píxeles puede ser implementadas como una transformada de M puntos de 1 dimensión y otra transformada de N puntos de 1 dimensión. Además, se muestran tres propiedades de la DCT que pueden ser usadas para reducir el costo de implementación:

1. Simetría entre columnas pares e impares de la matriz de coeficientes.
2. Flexibilidad y Escalabilidad permitiendo que las matrices de transformación para 16 puntos, 8 puntos y 4 puntos sean derivadas de la matriz de transformación de 32 puntos, (por lo que no se necesita implementaciones para cada caso).
3. Simetría entre la transformada directa e inversa del coseno.

El diseño para la arquitectura de la transformada directa e inversa se muestra en la Figura 7 [11]. En esta, se multiplica los valores de las muestras (en la imagen, estas son

M0, M1, M2, M3) por los coeficientes definidos por el estándar HEVC. Para acelerar el proceso, se separó las columnas pares e impares de la matriz de coeficientes para poder realizar las multiplicaciones en paralelo.

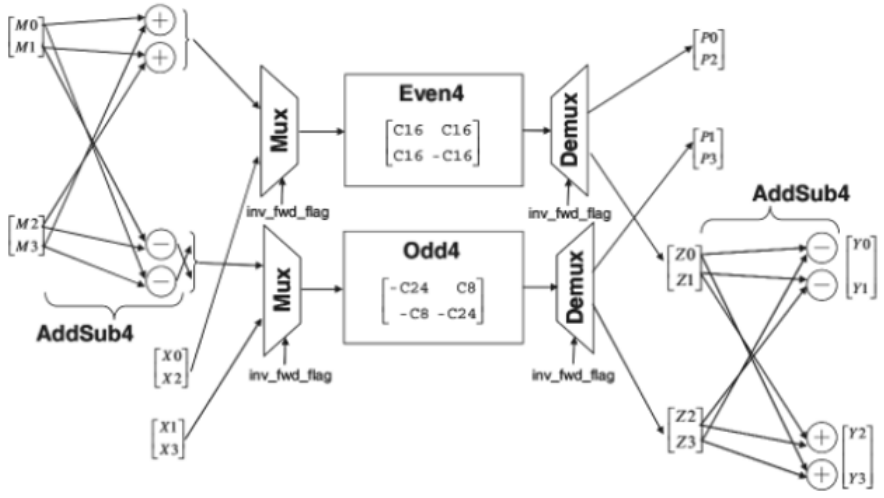


Figura 2.1: Arquitectura de transformada directa e inversa de 4 puntos[11]

En [12] se presenta una arquitectura que tiene un consumo menor de energía, según el autor. La técnica que presenta consiste en evitar procesar ciertas multiplicaciones entre la matriz de transformación y los datos si estos últimos no son mayores que un valor mínimo.

En el hardware propuesto en [12] también se utiliza el algoritmo MCM (en inglés, Multiplierless Multiple Constant Multiplication) para reducir el número de iteraciones al momento de realizar las multiplicacione[13]. También se usa “clock gating” (uso de compuertas lógicas en las señales de reloj) para reducir la potencia consumida por los registros (Flip Flops) de la arquitectura; y una estructura “mariposa” para realizar las multiplicaciones entre columnas. El hardware resultante se muestra en la Figura 2.2.

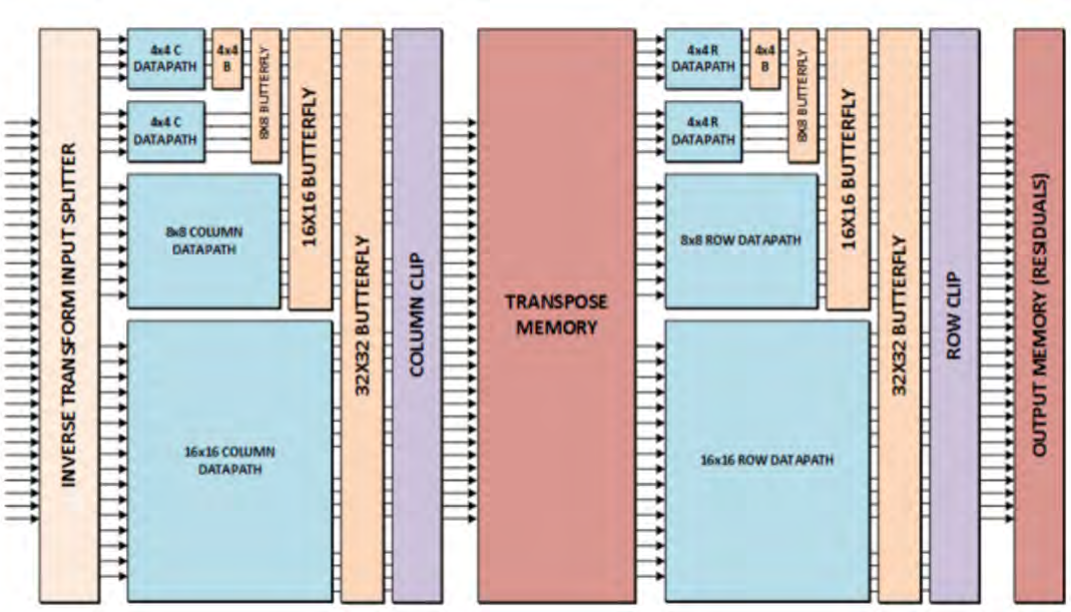


Figura 2.2: Hardware descrito con ahorro de energía[12]

Al igual que en los anteriores ejemplos, en [14] se descompone la transformada 2D en columnas para que se pueda procesar las operaciones a nivel de una dimensión. Además, cada transformación de una dimensión se subdivide aún más usando una separando las columnas pares e impares. Para poder realizar transformaciones de matrices grandes (por ejemplo, 32x32) se sub-divide recursivamente. La Figura 2.3 muestra un ejemplo de esta sub-división.

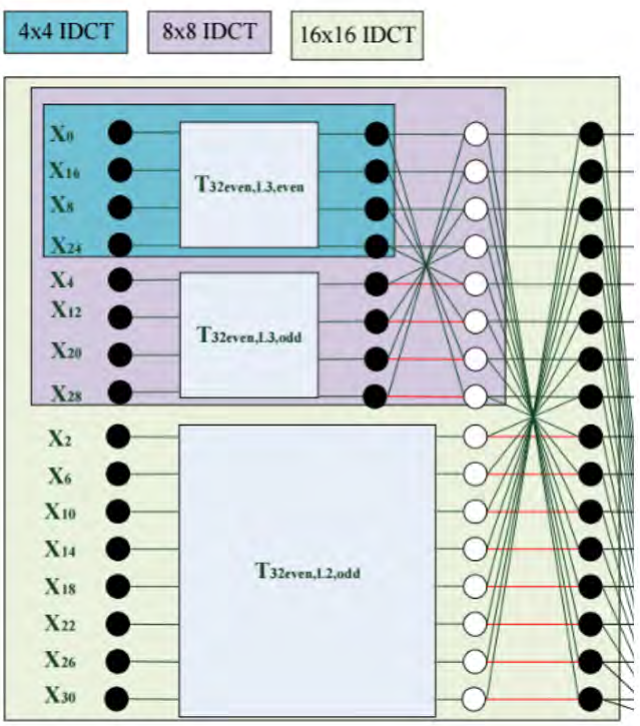


Figura 2.3: División recursiva de una matriz de 32x32 en unidades menores de 16x16, y luego estas se dividen a su vez en matrices 8x8 y luego en 4x4 [14]

2.2 Algoritmos para el cálculo de DCT e IDCT

De la misma manera que se puede acelerar el cálculo de la transformada discreta de Fourier (DFT), se puede realizar el mismo procedimiento para calcular rápidamente de la transformada discreta del coseno (DCT) y su inversa. La Figura 2.4 muestra el procedimiento que se sigue para obtener la DFT usando bloques básicos llamados "mariposa" que permite hallar la transformada de 2 puntos. Luego, con los resultados obtenidos, se pude volver a realizar el procedimiento, pero ahora utilizando 4 puntos. Finalmente, con los resultados anteriores se pude obtener la transformada de los 8 puntos originales.

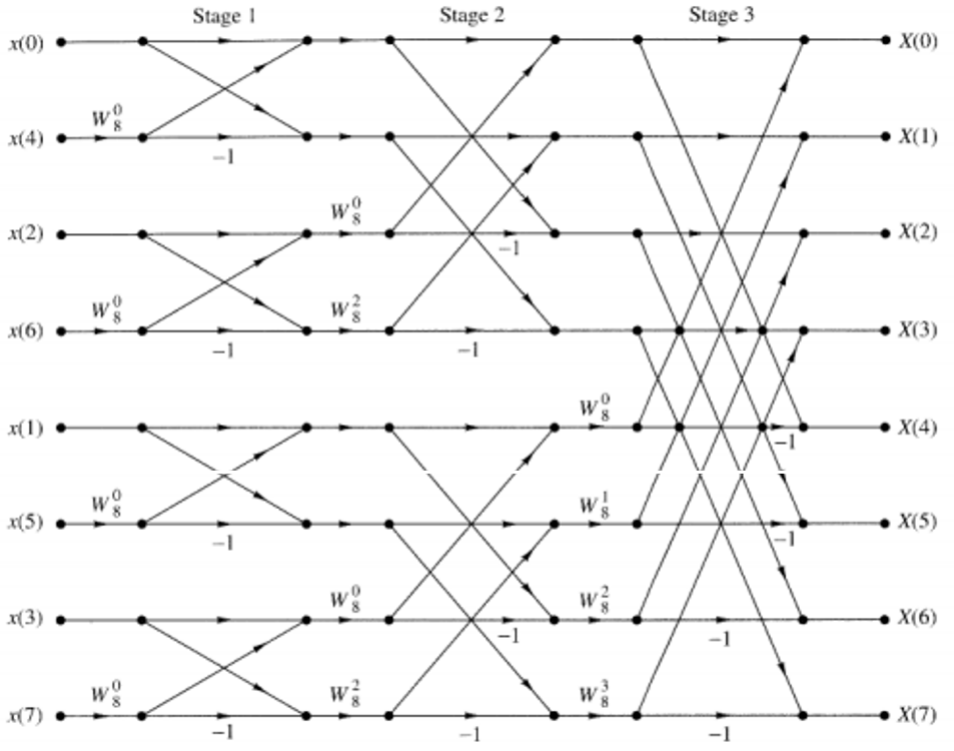


Figura 2.4: Cálculo de DFT usando bloques mariposa de creciente tamaño.

Como la diferencia entre la DFT y la DCT está en los valores de coeficientes que se multiplican (en la DFT, estos se basan en exponenciales y en la DCT estos se basan en cosenos), se pude aplicar el mismo procedimiento descrito anteriormente para obtener rápidamente la DCT y la IDCT dividiendo el procedimiento en bloque más pequeños.

Se puede calcular los valores pares e impares de la DCT e IDCT por separado usando lo que se llama ”descomposición par-impar” que se explica en [15]. Esta consiste en usar la simetría de los coeficientes para separar el cálculo. La operación normal sería.

$$y = C_4 * x \tag{2.1}$$

Sin embargo, se puede dividir el proceso de la siguiente manera utilizando la siguiente descomposición par-impar, donde x representa una entrada de 4 puntos, z son valores intermedios y y es la salida del proceso de descomposición DCT de tamaño 4.

$$\begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} x_0 + x_3 \\ x_1 + x_2 \\ x_2 - x_1 \\ x_3 - x_0 \end{bmatrix} \tag{2.2}$$

$$\begin{bmatrix} y_0 \\ y_2 \end{bmatrix} = \begin{bmatrix} 64 & 64 \\ 64 & -64 \end{bmatrix} \begin{bmatrix} z_0 \\ z_1 \end{bmatrix} \tag{2.3}$$

$$\begin{bmatrix} y_1 \\ y_3 \end{bmatrix} = \begin{bmatrix} -36 & -83 \\ 83 & -36 \end{bmatrix} \begin{bmatrix} z_2 \\ z_3 \end{bmatrix} \tag{2.4}$$

Un procedimiento parecido se sigue para calcular la IDCT de 4 puntos.

$$\begin{bmatrix} z_0 \\ z_1 \end{bmatrix} = \begin{bmatrix} 64 & 64 \\ 64 & -64 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \tag{2.5}$$

$$\begin{bmatrix} z_1 \\ z_3 \end{bmatrix} = \begin{bmatrix} -36 & -83 \\ 83 & -36 \end{bmatrix} \begin{bmatrix} x_2 \\ x_3 \end{bmatrix} \tag{2.6}$$

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} z_0 + z_3 \\ z_1 + z_2 \\ z_2 - z_1 \\ z_3 - z_0 \end{bmatrix} \tag{2.7}$$

Este procedimiento también reduce el numero de operaciones necesarias para calcular la DCT e IDCT.

Para hallar la DCT o su inversa (IDCT) es necesario realizar muchas operaciones de multiplicación entre los valores de la imagen que se quiere operar y los coeficientes de la transformada. Debido a esto, para poder realizar una implementación en hardware que pueda procesar un video en tiempo real es necesario que exista la menor cantidad de compuertas lógicas para reducir el tiempo de procesamiento. El Algoritmo MCM permite implementar multiplicación de números con un 20% menos operaciones de suma y resta, lo cual se traduce en que se necesita menos lógica.

El algoritmo MCM permite implementar cualquier multiplicación usando sólo multiplicaciones por potencias de 2 y sumas de estas. Como las multiplicaciones por potencias de 2 se traducen en desplazamientos en los registros que contienen los números, es mucho más fácil de implementar que un método clásico de sumas y a la vez es mucho más rápido. En ejemplo de cómo puede multiplicarse un número utilizando este algoritmo se puede observar en la Figura 11.

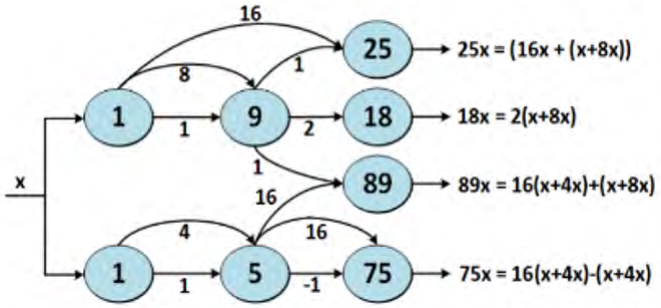


Figura 2.5: Cálculo de multiplicaciones variadas usando el algoritmo MCM

Por ejemplo, la multiplicación de '25' y 'x' se puede dividir en 16x + x + 8x, lo cual se implementa con la suma del desplazamiento de cuatro bits del número x, el desplazamiento de tres bits del número x y el mismo número x. Como los coeficientes de la codificación HEVC son valores fijos y limitados en número, se puede implementar un arreglo de desplazamientos y sumas para cada multiplicación por cada coeficiente., además, el bloque permite implementar etapas de pipeline para cada multiplicación, lo cual acelera el proceso aún más.y

2.3 Trabajos relacionados

En [15], Meher et al. proponen una arquitectura para el cálculo de DCT que puede ser reutilizada para todos los tamaños de DCT: 4, 8 16 y 32. En este trabajo se utiliza la reutilización de bloque de menor tamaño para el cálculo, además de bloques de suma y desplazamiento para realizar las operaciones necesarias. Además, en el artículo se muestra el diseño de arquitecturas en paralelo y reutilizando el bloque de cálculo de DCT de una dimensión. La diferencia entre estas dos arquitecturas puede verse en la Figura 2.6.

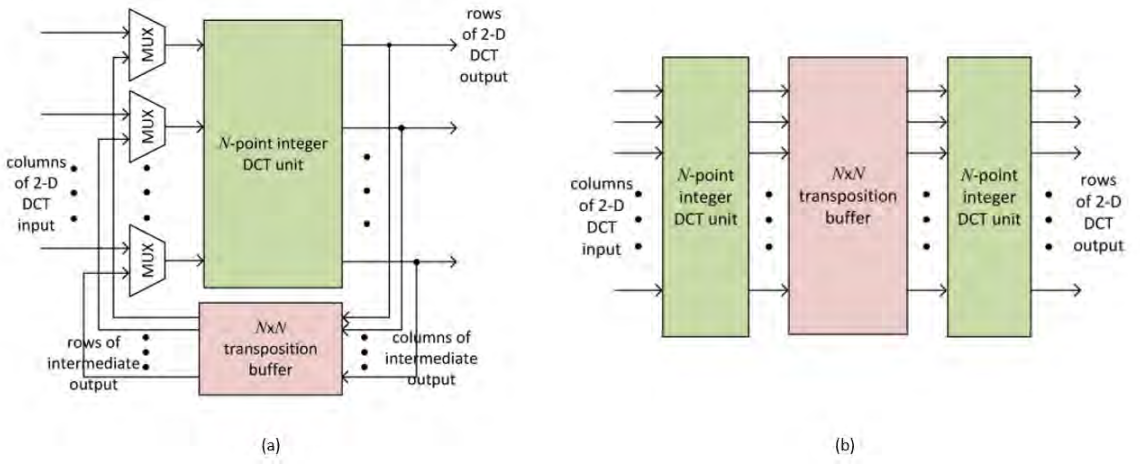


Figura 2.6: Arquitecturas diseñadas en [15] para el cálculo de DCT: (a) muestra la reutilización de su unidad de cálculo de DCT de 1D que (b) muestra un diseño completamente paralelo que usa dos unidades de cálculo de DCT de 1D.

En la Figura 2.7 se muestra el diagrama interno del bloque de cálculo de DCT en 1D para N valores. El bloque de color verde muestra que se utiliza la misma arquitectura pero de menor tamaño (N/2) para calcualar la DCT de tamaño N. Sin embargo, esto solo sirve para la mitad de coeficientes, así que la otra mitad es calculada de manera directa utilizando SAU (shift-add units) cuya función es realizar las multiplicaciones necesarias mediante desplazamientos y sumas.

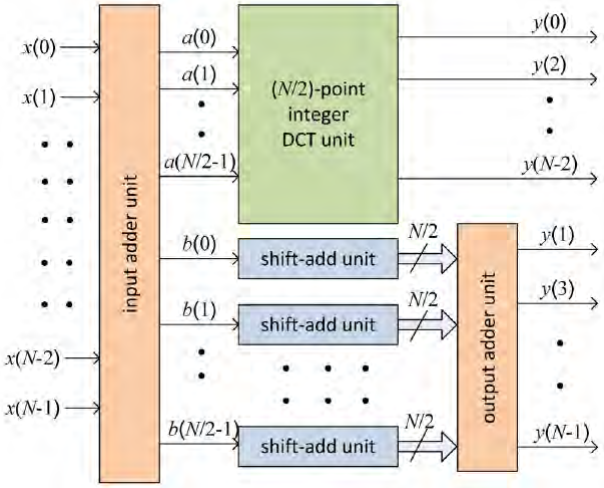


Figura 2.7: Unidad de cálculo de DCT en 1D para N valores [15]

Por otro lado, en [16], Chen et al. proponen una arquitectura similar a la anterior pero para calcular el IDCT. En la Figura 2.8 se puede ver el bloque de cálculo de IDCT

en 2D propuesto en este artículo. Se tiene dos bloques principales: DAE (”Distributed Arithmetic Even”) y DAO (”Distributed Arithmetic Odd”) para realizar los cálculos solo con valores con indice par o impar, respectivamente.

Luego del DAE y DAO, hay un par de registors y un bloque sumador/restador encargados de almacenar los datos para la salida y para el siguiente ciclo de cálculo. También se tiene una memoria intermedia para realizar la operación de transposición.

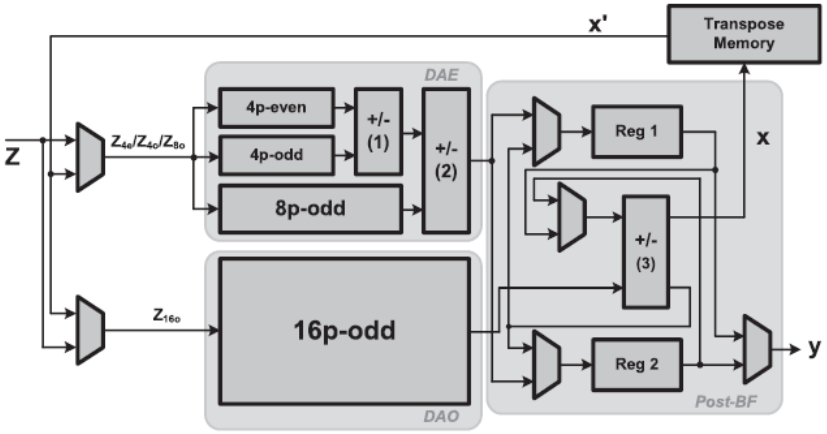


Figura 2.8: Unidad de cálculo de IDCT en 2D para 32x32 valores [16]

Finalmente, en [17] se presenta el diseño e implementación de arquitecturas reconfigurables para DCT e IDCT. La principal diferencia entre lo propuesto por Llamocca y [16] y [15] es que el sistema propuesto puede cambiar su estructura dados requerimientos variantes en el tiempo de recursos, throughtput y eficiencia.

Capítulo 3

Diseño de la arquitectura de hardware del módulo DCT e IDCT

De acuerdo a lo expuesto en los capítulos anteriores, la implementación en hardware de la DCT e IDCT permite aprovechar las características propuestas en el estándar HEVC y también permite implementaciones capaces de realizar ambas operaciones reutilizando los mismos bloques. Debido a esto, se proponen los siguientes objetivos para este trabajo.

3.1 Objetivos generales y específicos

Objetivos generales:

- Diseñar un módulo de cálculo de la Transformada Discreta Coseno (DCT) para el estándar HEVC.
- Diseñar un módulo de cálculo de la Transformada Discreta Coseno Inversa (IDCT) para el estándar HEVC.

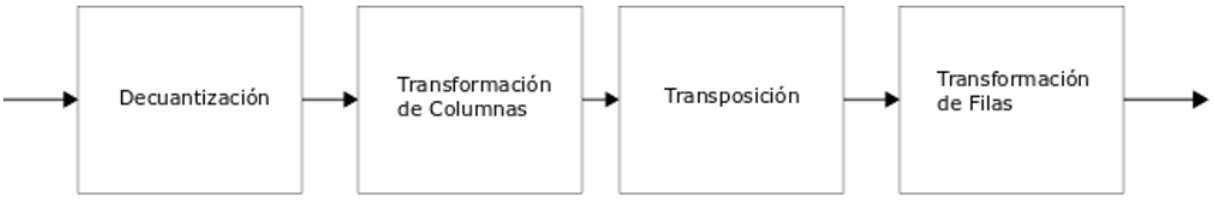
Objetivos específicos:

- Emplear el lenguaje Verilog HDL para la descripción de la arquitectura conjunta de DCT e IDCT para los tamaños de TU de 4x4, 8x8, 16x16 y 32x32. Se eligió este lenguaje porque es más usado en la industria del diseño digital.

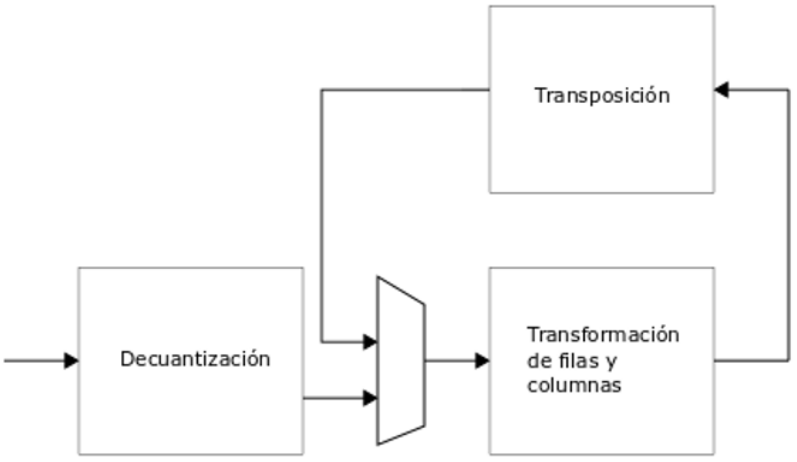
- Verificar funcionalmente la arquitectura diseñada.
- Sintetizar la arquitectura y optimizar la arquitectura para obtener la máxima frecuencia de operación para procesamiento de secuencias 4k en tiempo real.

3.2 Diseño de la arquitectura

La arquitectura para el diseño de los módulos DCT e IDCT de un decodificador HEVC fue realizado en el lenguaje Verilog HDL. En la bibliografía revisada, se propone dos opciones de alto nivel para los módulos DCT e IDCT que se resumen en [18]. La primera se ve en la Figura 3.1(a) donde se utilizan dos bloques de transformación de 1D (uno para las filas y otro para las columnas) y un bloque de transposición en entre ambos. La segunda opción en la Figura 3.1(b) se reutiliza un mismo bloque de 1D para las columnas y filas.



(a)



(b)

Figura 3.1: Estructura paralela utilizando dos bloques de transformación 1D (a) y estructura que reutiliza el mismo bloque 1D para las filas y columnas (b) [18]

Para este trabajo se eligió el uso de dos bloques de 1D, junto con una matriz de transposición, para poder dividir todo el circuito en diferentes etapas de pipeline, El gráfico de alto nivel de la arquitectura diseñada se puede ver en la Figura 3.2. El primer bloque 1D procesa las columnas y el segundo bloque procesa las filas. El detalle de la matriz de transposición 4x4 se puede observar en la Figura 3.6.

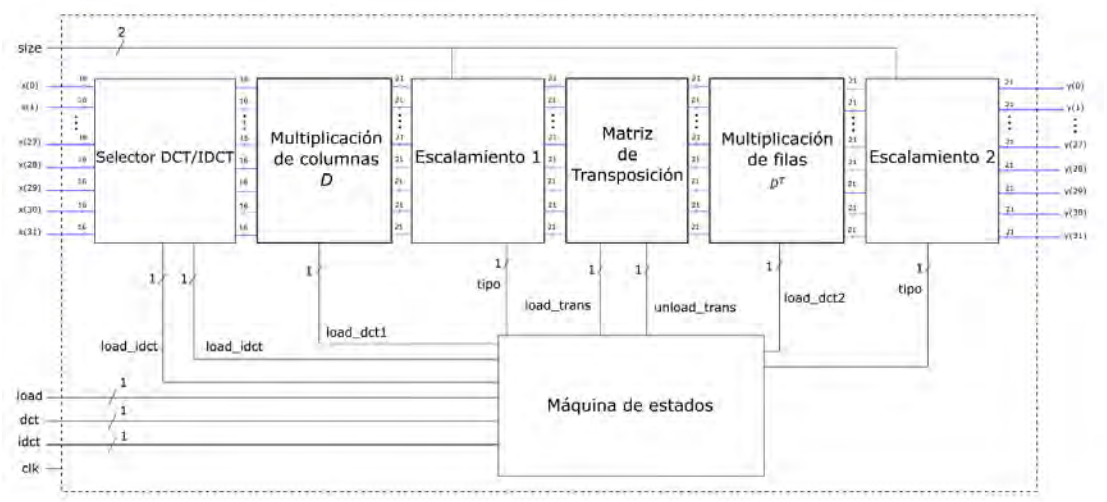


Figura 3.2: Diagrama de alto nivel de la arquitectura propuesta

En la siguientes líneas se describe las diferentes partes de la arquitectura: el bloque de DCT de una dimensión (1-D DCT), la matriz de transposición, el bloque de escalamiento, máquina de estados que controla el funcionamiento del circuito.

3.3 Bloque de DCT de una dimensión (1-D DCT)

Los dos bloques de 1D son iguales, salvo que procesan diferentes datos. En cada bloque 1D, se utiliza la arquitectura mariposa para poder calcular la DCT o IDCT. Cada bloque puede procesar 4, 8, 16 o 32 datos. Cada bloque está basado en el conjunto de dos bloques de menor tamaño: el de 32 puntos está formado por dos de 16 puntos, el de 16 puntos está formado por dos de 8 puntos y el de 8 puntos está formado por dos de 4 puntos. En la Figura 3.3 se puede ver el detalle de la arquitectura mariposa utilizada para el bloque de n datos. En la figura se ve que en la entrada y salida existe una etapa de sumas y al medio se tiene 2 etapas: una etapa DCT de tamaño N/2 y una

etapa de desplazamientos y sumas. La separación de las dos etapas permite reducir la complejidad computacional del cálculo.

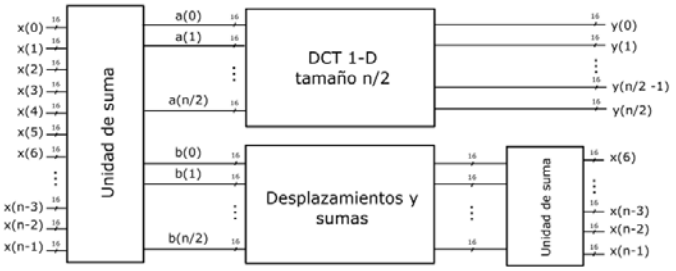


Figura 3.3: Arquitectura mariposa para el procesamiento de una columna de n elementos

Como se necesitan realizar DCT de diferentes tamaños para el estándar HEVC, es conveniente que los bloques de DCT de menor tamaño puedan ser reutilizados para calcular los bloques de mayor tamaño. De esta manera, para calcular una DCT de menor tamaño solo se utiliza una parte del circuito, mientras que para calcular una DCT del mayor tamaño posible se estaría utilizando todos los recursos descritos, lo cual es posible con la arquitectura propuesta.

La unidad de sumas realiza la siguiente operación:

$$a(i) = x(i) + (N - i - 1) \tag{3.1}$$

$$b(i) = x(i) - (N - i - 1) \tag{3.2}$$

Los valores a(i) son los se vuelve las entradas para la DCT de tamaño N/2 y los vales b(i) son los que son multiplicados directamente. Este proceso se puede llamar mariposa parcial como se hace en [12]. En la Figura 3.4 se puede ver en detalle la unidad de suma de tamaño 8 para el cálculo de DCT.

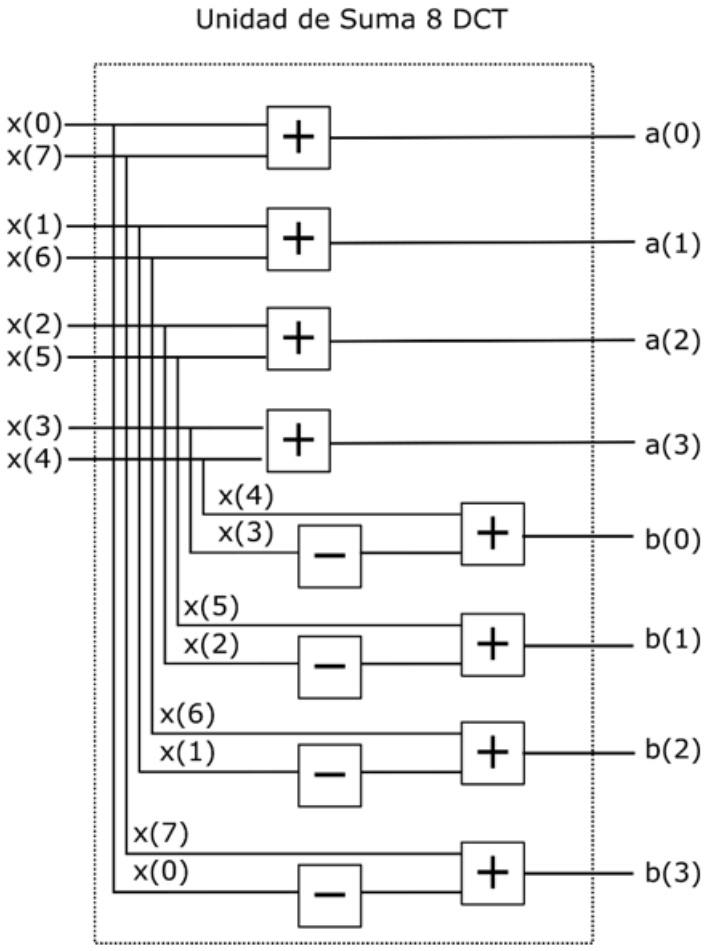


Figura 3.4: Detalle de Unidad de Sumas para DCT de tamaño 8

Como se proponen en [15], esta arquitectura permite la reutilización de bloque de menor tamaño para DCT de mayor tamaño de manera que no es necesario hacer un bloque DCT 1-D para cada tamaño de DCT, sino solo utilizar una parte del circuito para cada tamaño dependiendo de los datos de entrada. Sin embargo, a diferencia de [15], es necesario que la arquitectura propuesta pueda procesar todos los tipos de DCT e IDCT, no solo un tamaño a la vez. Para solucionar esto, se utiliza una señal de control para indicar que tamaño de DCT o IDCT se está procesando para que se evite el cálculo innecesario si se tiene un tamaño menor a 32x32. En la Figura 3.5 se puede observar una representación de la reutilización de bloques de menor tamaño para la arquitectura.

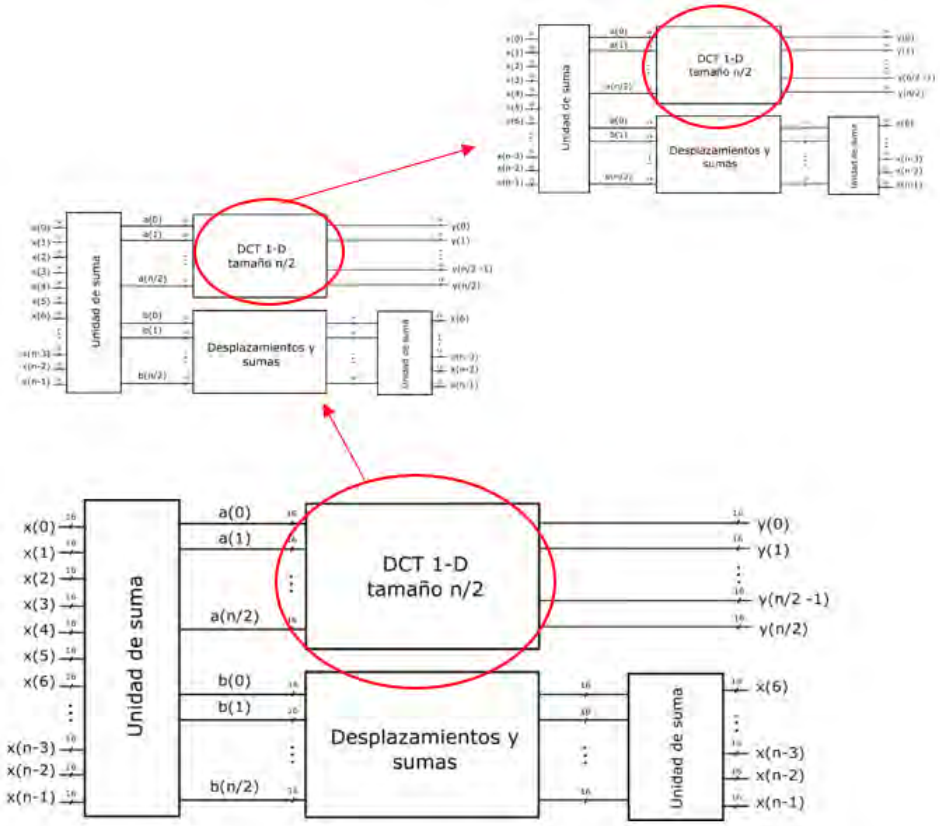


Figura 3.5: Reutilización de bloques de cálculo de menor tamaño para cálculo de DCT

Tal como se explicó en el capítulo anterior, la matriz de transformación D que se utiliza en el HEVC solo está formada por números enteros. Se divide cada número en sus potencias de 2 para evitar la implementación de bloques de multiplicación. Los coeficientes que son directas potencias de 2 se pueden implementar como desplazamientos de bits a la izquierda, mientras de los demás valores necesitan múltiples desplazamientos y sumas. En la Tabla 1 se puede observar cómo se realiza la multiplicación cada valor único de la matriz D. Un ejemplo se observa en la Figura 16 para la multiplicación por 90.

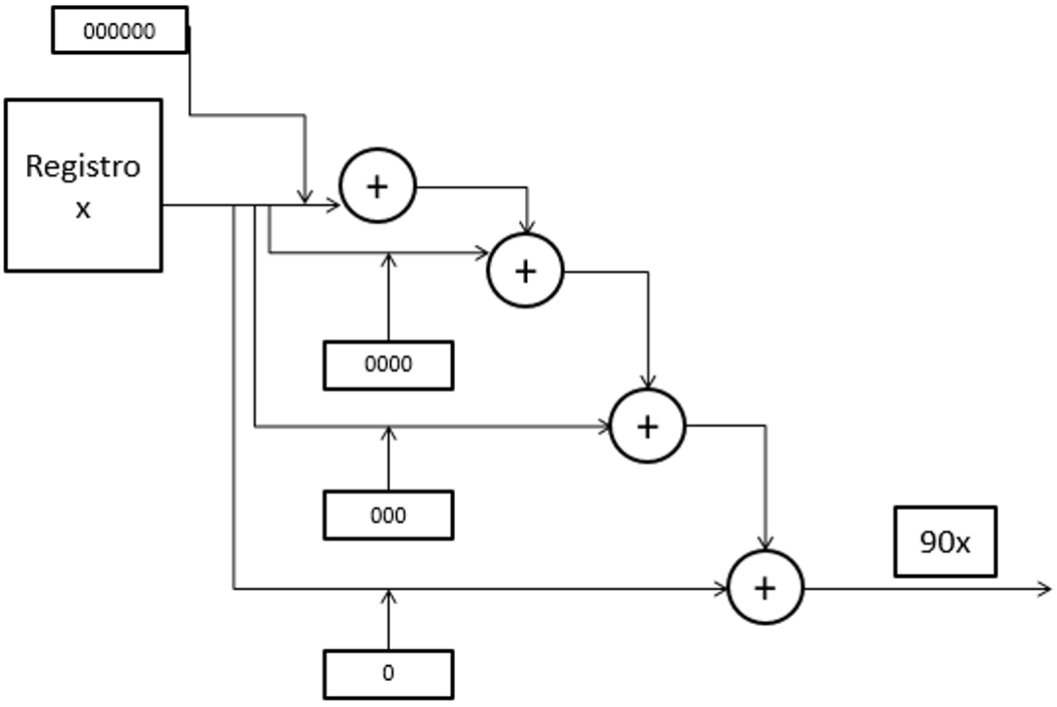


Figura 3.6: Multiplicación por 90 usando desplazamientos y sumas. Se concatena el valor "x" con cadenas de ceros para luego sumarlas y obtener la multiplicación deseada

Una lista completa de los desplazamientos y sumas necesarios para el cálculo se puede encontrar en la Tabla 3.1

Tabla 3.1: Multiplicación por 90 usando desplazamientos y sumas. Se concatena el valor "x" con cadenas de ceros para luego sumarlas y obtener la multiplicación deseada

Operación a realizar	**Método de operación**
90x	(x<<6) + (x<<4) + (x<<3) + (x<<1)
89x	(x<<6) + (x<<4) + (x<<3) + x
88x	(x<<6) + (x<<4) + (x<<3)
87x	(x<<6) + (x<<4) + (x<<2) + (x<<1) + x
85x	(x<<6) + (x<<4) + (x<<2) + x
83x	(x<<6) + (x<<4) + (x<<1) + x
82x	(x<<6) + (x<<4) + (x<<1)
80x	(x<<6) + (x<<4)
78x	(x<<6) + (x<<3) + (x<<2) + (x<<1)
75x	(x<<6) + (x<<3) + (x<<1) + x
73x	(x<<6) + (x<<3) + x
70x	(x<<6) + (x<<2) + (x<<1)
67x	(x<<6) + (x<<1) + x
64x	x<<6
61x	(x<<5) + (x<<4) + (x<<3) + (x<<2) + x
57x	(x<<5) + (x<<4) + (x<<3) + x
54x	(x<<5) + (x<<4) + (x<<2) + (x<<1)
50x	(x<<5) + (x<<4) + (x<<1)
46x	(x<<5) + (x<<3) + (x<<2) + (x<<1)
43x	(x<<5) + (x<<3) + (x<<1) + x
38x	(x<<5) + (x<<2) + (x<<1)
36x	(x<<5) + (x<<2)
31x	(x<<4) + (x<<3) + (x<<2) + (x<<1) + x
25x	(x<<4) + (x<<3) + x
22x	(x<<4) + (x<<2) + (x<<1)
18x	(x<<4) + (x<<1)
13x	(x<<3) + (x<<2) + x
9x	(x<<3) + x
4x	x<<2

3.4 Matriz de transposición

Para poder realizar la DCT, es necesario hacer la transposición de los datos que se obtienen del primer bloque de DCT 1-D. Como todos los datos de la operación anterior, estos se obtienen columna por columna y es necesario almacenarlos para que estos puedan ser leídos luego. La arquitectura de la matriz de transposición permite almacenar cada columna de los diferentes tamaños de DCT e IDCT en cada ciclo de reloj y luego permite pasar los datos de las filas a la siguiente etapa. En la Figura 3.6 se puede ver el diagrama usado en [17], en el cual se basó el usado en el trabajo actual.

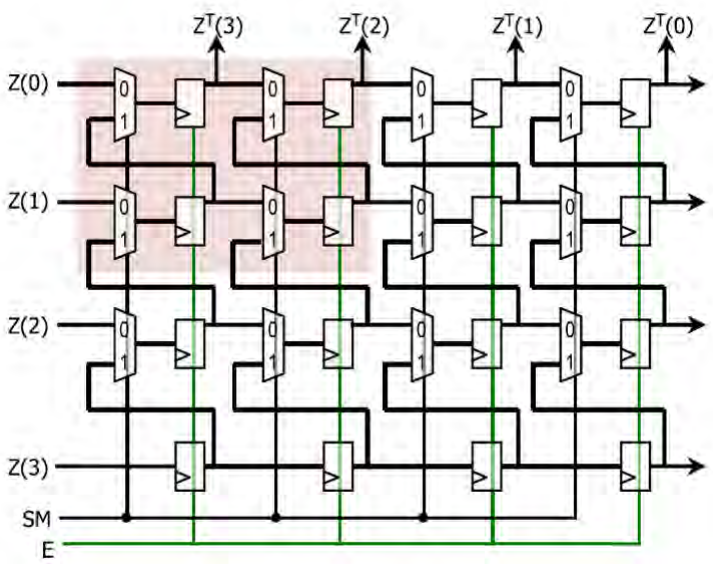


Figura 3.7: Matriz de transposición como se muestra en [17]

3.5 Bloque de Escalamiento

Los valores de las matrices HEVC han sido escalados en comparación a los de una DCT ortonormal, por lo que para preservar la norma del bloque residual que se ingresa al módulo de DCT e IDCT se necesita utilizar factores de escalamiento a la salida de cada bloque 1D de DCT e IDCT.

En la Tabla 3.2 se muestran los valores que tienen que tener cada factor de escalamiento según las fórmulas en 1.7 y 1.8. Los valores de escalamientos y el uso de la matriz D o DT dependen de si se está efectuando la operación de DCT o IDCT y

del tamaño de la operación. Estos valores están determinados por el bloque de control del circuito de 1D.

Tabla 3.2: Factores de escalamiento para cada tamaño de entrada

Escalamiento\Tamaño	N = 4	N = 8	N = 16	N = 32
ST1	2^{-1}	2^{-2}	2^{-3}	2^{-4}
ST2	2^{-8}	2^{-9}	2^{-10}	2^{-11}
SIT1	2^{-7}	2^{-7}	2^{-7}	2^{-7}
SIT2	2^{-12}	2^{-12}	2^{-12}	2^{-12}

3.6 Máquina de estados

Para la máquina de estados, se definieron seis estados los cuales son:

- IDLE: Estado inicial donde no se procesa ningún dato.
- DC1: Estado donde se procesan las columnas ingresadas.
- IDCT_TRANS: Estado donde se realiza la transposición de datos en caso de que se quiera hacer la operación de IDCT.
- IDCT_UNLOAD: Estado donde se vacía los registros de transposición del estado IDCT_TRANS.
- LOAD_TRANS: Estado donde se realiza la transposición de datos.
- UNLOAD_TRANS: Estado donde se vacía los registros de transposición del estado LOAD_TRANS.
- DCT2: Estado donde se procesan las filas del dato luego del procesamiento de columnas.

En la Figura 3.9 se puede ver el diagrama de estados y en la Tabla 3.3 se pueden ver las señales generadas dependiendo del estado.

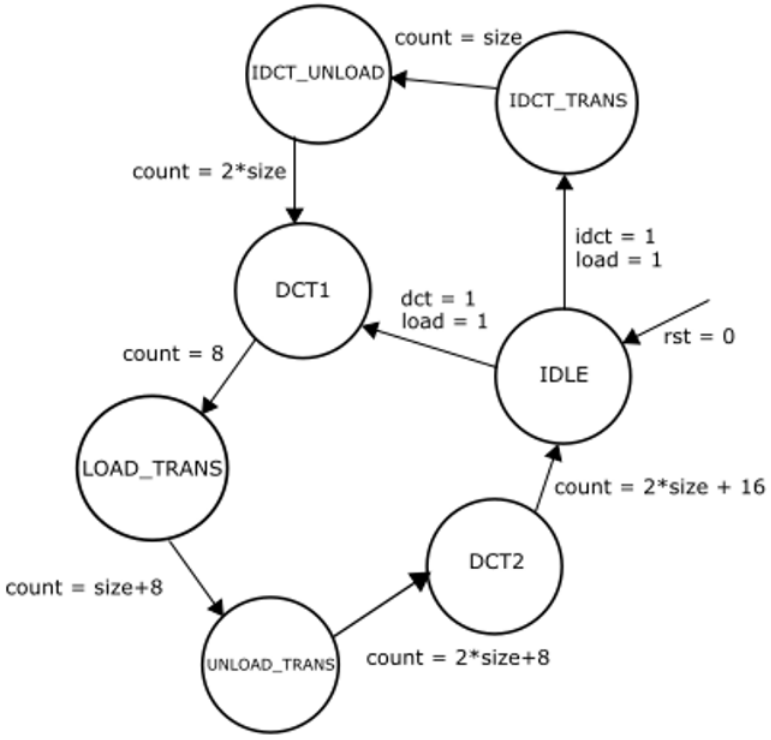


Figura 3.8: Diagrama de estados

Tabla 3.3: Señales de control según estado

Estado	load_1dct	load_trans	unload_trans	load_2dct	load_idct	unload_idct
IDLE	0	0	0	0	0	0
DCT1	1	0	0	0	0	0
LOAD_TRANS	1	1	0	0	0	0
UNLOAD_TRANS	0	0	1	1	0	0
DCT2	0	0	0	1	0	0
IDCT_TRANS	0	0	0	0	1	0
IDCT_UNLOAD	1	0	0	0	0	1

En la Figura 3.9 se puede ver un diagrama completo de la arquitectura diseñada.

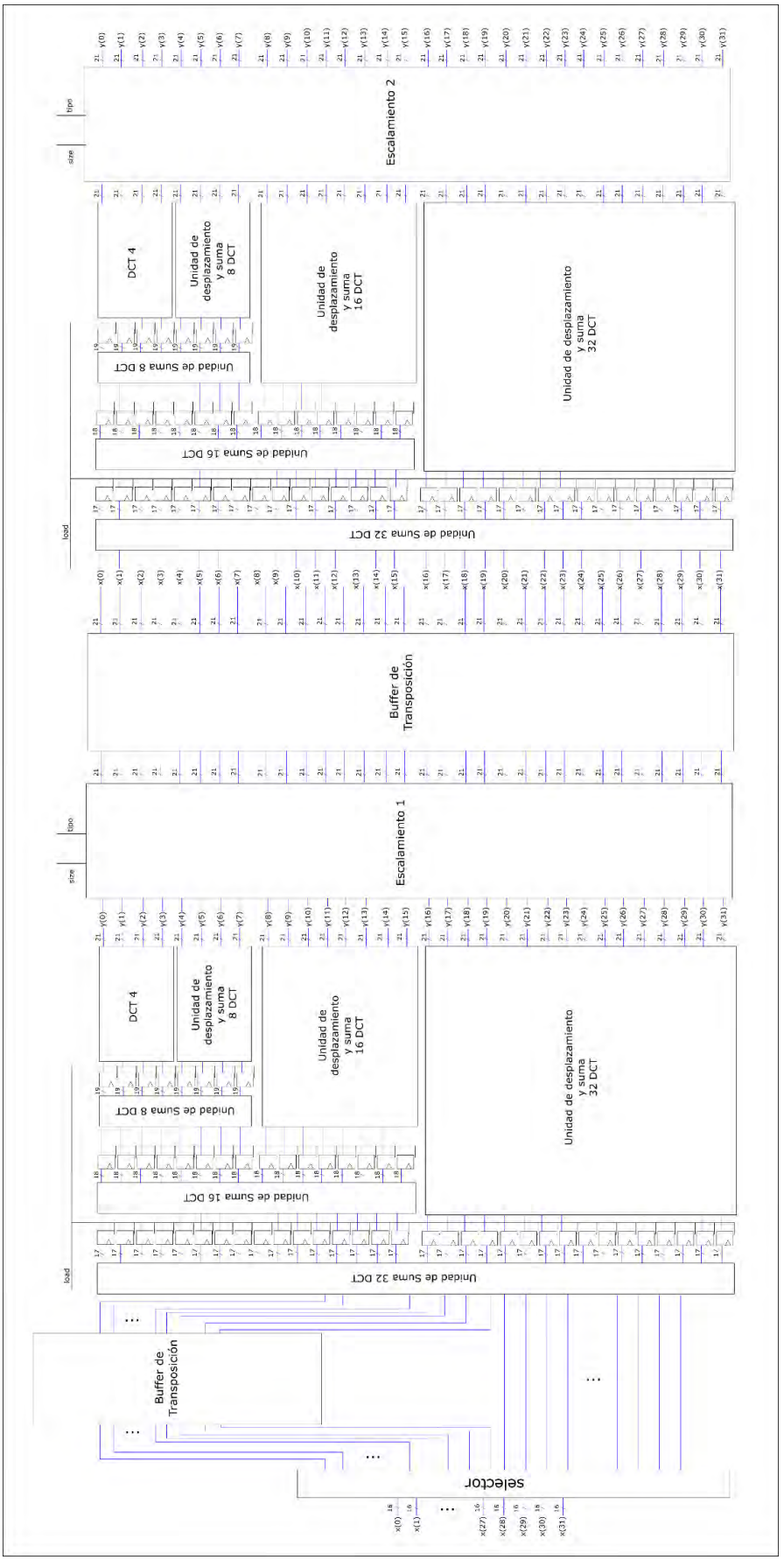


Figura 3.9: Diagrama de la arquitectura propuesta

Capítulo 4

Resultados

4.1 Simulación comportamental

El circuito de DCT e IDCT tiene datos de 9 bits de entrada dado que se asume el uso de pixeles codificados en 8 bits y su diferencia puede estar dentro del rango [-255, 255]. Se realizaron dos simulaciones para comprobar el funcionamiento del circuito: la primera simulación fue con la Figura 4.1 (a) y la segunda con la Figura 4.1 (b).

En la primera simulación se utilizó una imagen de tamaño 32x32 pixeles para facilitar la corrección de errores. Una vez que se obtuvieron los resultados esperados, se procedió a realizar una segunda simulación con una imagen 4K (2160 x 3840 pixeles).Las dos imagenes se pueden ver en la Figuras 4.1 y Figuras 4.2.

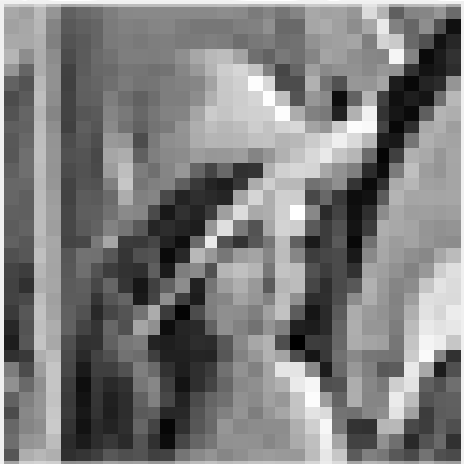


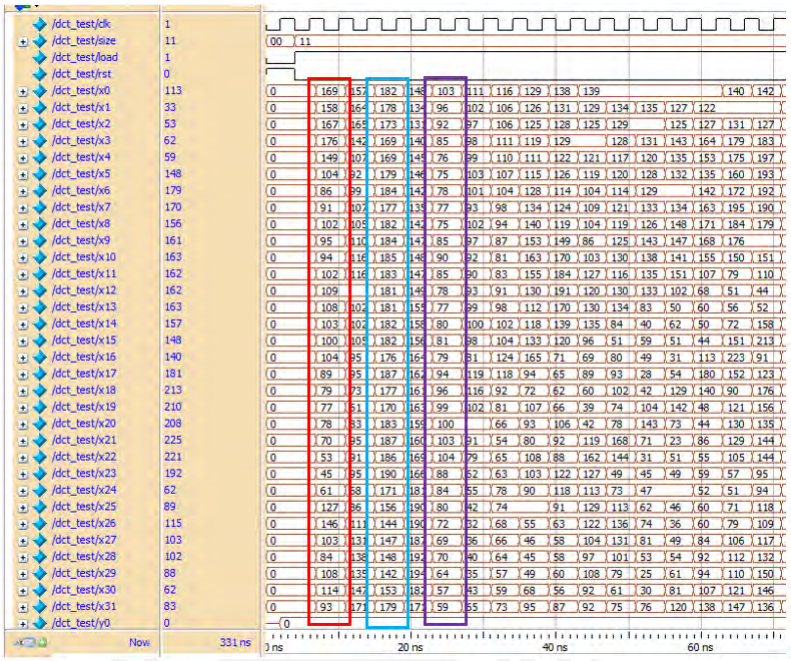
Figura 4.1: Imagen de prueba de tamaño 32x32

Figura 4.2: Imagen de prueba de tamaño 4k

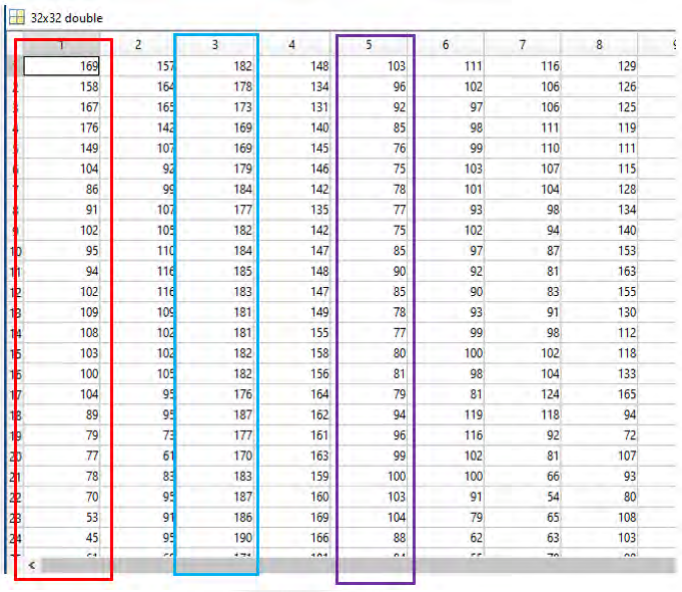
4.1.1 Simulación con imagen 32x32

Para mostrar el funcionamiento del circuito se eligió mostrar la simulación del proceso de DCT paso por paso. La operación de IDCT sigue el mismo número de pasos solo que las operaciones se realizan con diferentes parámetros, tal como se explicó en el capítulo 3.

La lectura de datos de la imagen se muestra en la Figura 4.3. Se utilizó el software Matlab para obtener el archivo de datos necesarios para el Testbench del circuito. En la Figura 4.4 se muestra la salida de la primera etapa de DCT junto con los resultados de Matlab. En la Figura 4.5 se muestra el funcionamiento del bloque de transposición de la arquitectura: primero como se llena la matriz de registros y luego como estos mueven los datos hacia la salida. En la Figura 4.6 se muestra la salida de la operación de DCT y se comprueba que los valores corresponden con los obtenidos de manera teórica en Matlab.

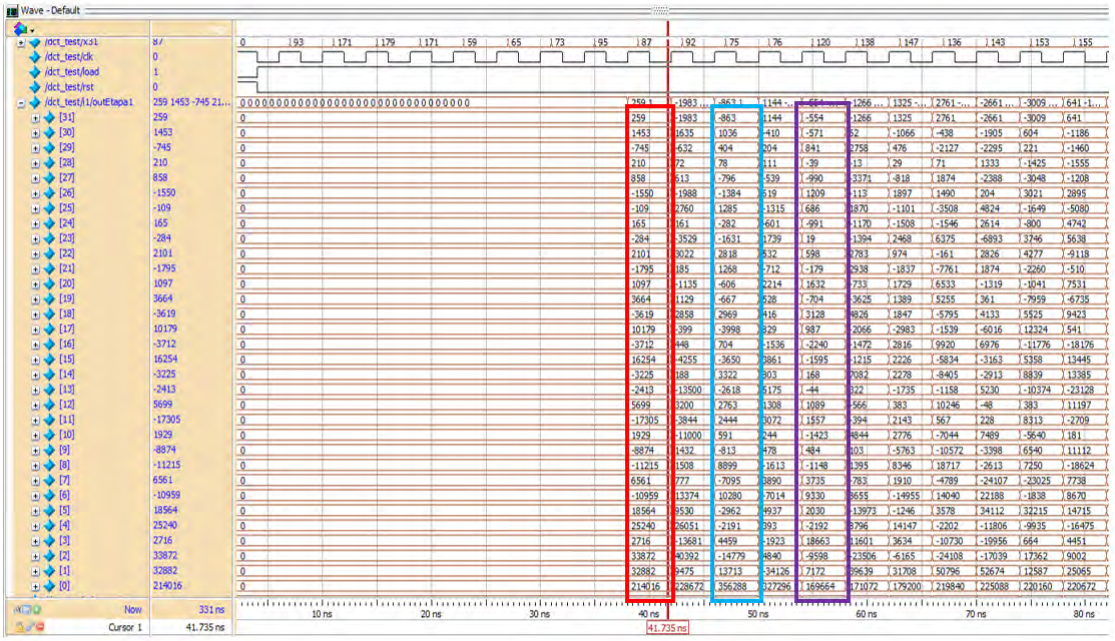


(a)

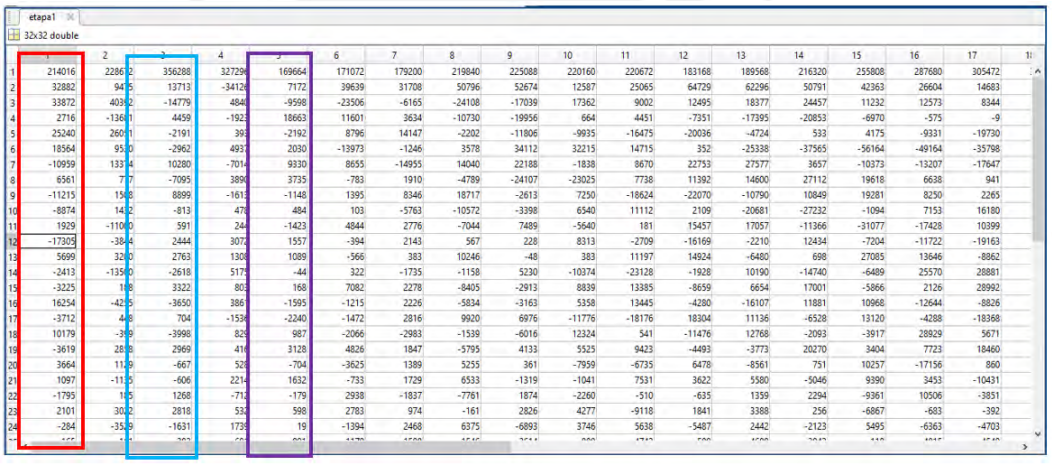


(b)

Figura 4.3: Ingreso de datos. En (a) se muestran los valores leídos por el circuito y en (b) se muestras los valores de la imagen de prueba que se están utilizando. Se está ingresando un dato en cada ciclo



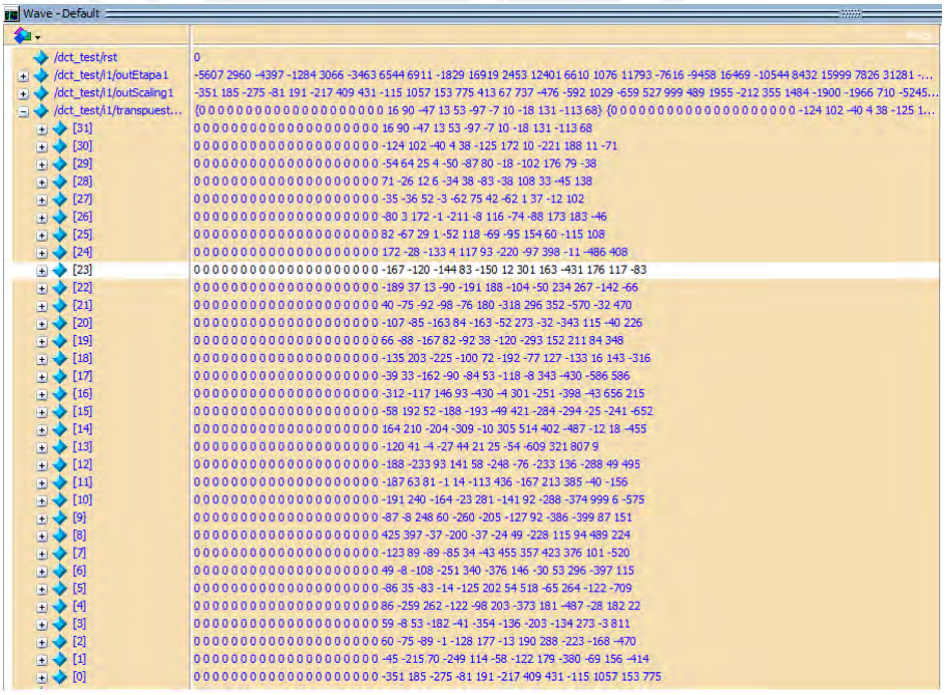
(a)



(b)

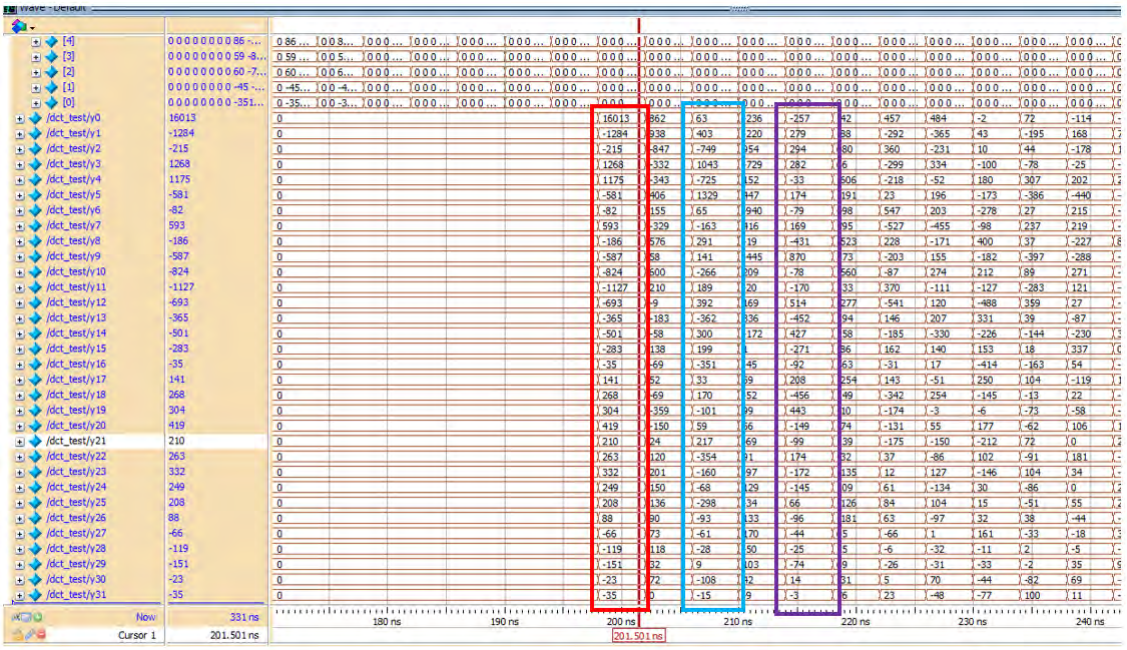
Figura 4.4: Salida de datos de la primera etapa de DCT. En (a) se observan los resultados de la primera etapa de la DCT, mientras que en (b) se muestran los resultados obtenidos en Matlab.

(a)

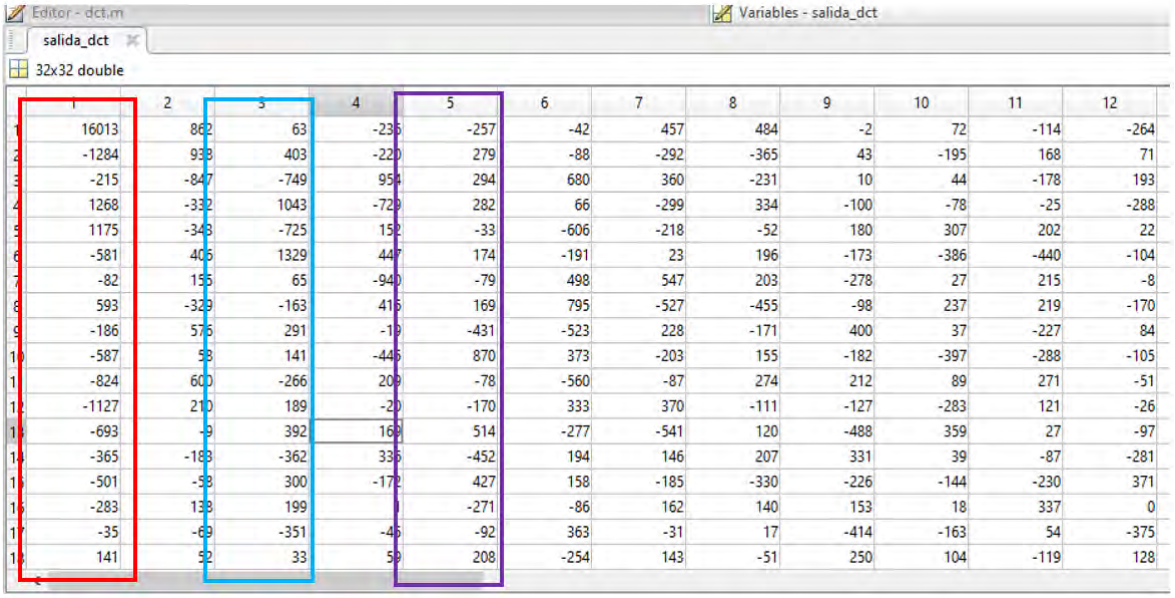


(b)

Figura 4.5: Comportamiento de la matriz de transposición. En (a) se puede ver que la matriz 32x32 está completamente llena, mientras que en (b) se muestra que la matriz mueve los datos de manera lateral para realizar la transposición.



(a)



	1	2	3	4	5	6	7	8	9	10	11	12
1	16013	862	63	-236	-257	-42	457	484	-2	72	-114	-264
2	-1284	938	403	-220	279	-88	-292	-365	43	-195	168	71
3	-215	-847	-749	954	294	680	360	-231	10	44	-178	193
4	1268	-332	1043	-729	282	66	-299	334	-100	-78	-25	-288
5	1175	-343	-725	152	-33	-606	-218	-52	180	307	202	22
6	-581	406	1329	447	174	-191	23	196	-173	-386	-440	-104
7	-82	155	65	-940	-79	498	547	203	-278	27	215	-8
8	593	-329	-163	416	169	795	-527	-455	-98	237	219	-170
9	-186	576	291	-19	-431	-523	228	-171	400	37	-227	84
10	-587	58	141	-446	870	373	-203	155	-182	-397	-288	-105
11	-824	600	-266	209	-78	-560	-87	274	212	89	271	-51
12	-1127	210	189	-20	-170	333	370	-111	-127	-283	121	-26
13	-693	-9	392	169	514	-277	-541	120	-488	359	27	-97
14	-365	-183	-362	336	-452	194	146	207	331	39	-87	-281
15	-501	-58	300	-172	427	158	-185	-330	-226	-144	-230	371
16	-283	138	199	0	-271	-86	162	140	153	18	337	0
17	-35	-69	-351	-46	-92	363	-31	17	-414	-163	54	-375
18	141	52	33	59	208	-254	143	-51	250	104	-119	128

(b)

Figura 4.6: Datos de salida luego de realizar la DCT. Se puede observar que los resultados obtenidos en (a) son iguales a los obtenidos usando Matlab en (b).

4.1.2 Simulación con imagen 4K

Para la simulación con la imagen 4K, se tuvo que dividir la imagen en partes de 32x32 y 16x16 pixeles. Estas divisiones se procesan por separado en el circuito propuesto. El detalle de la división se puede observar en la Figura 4.7.

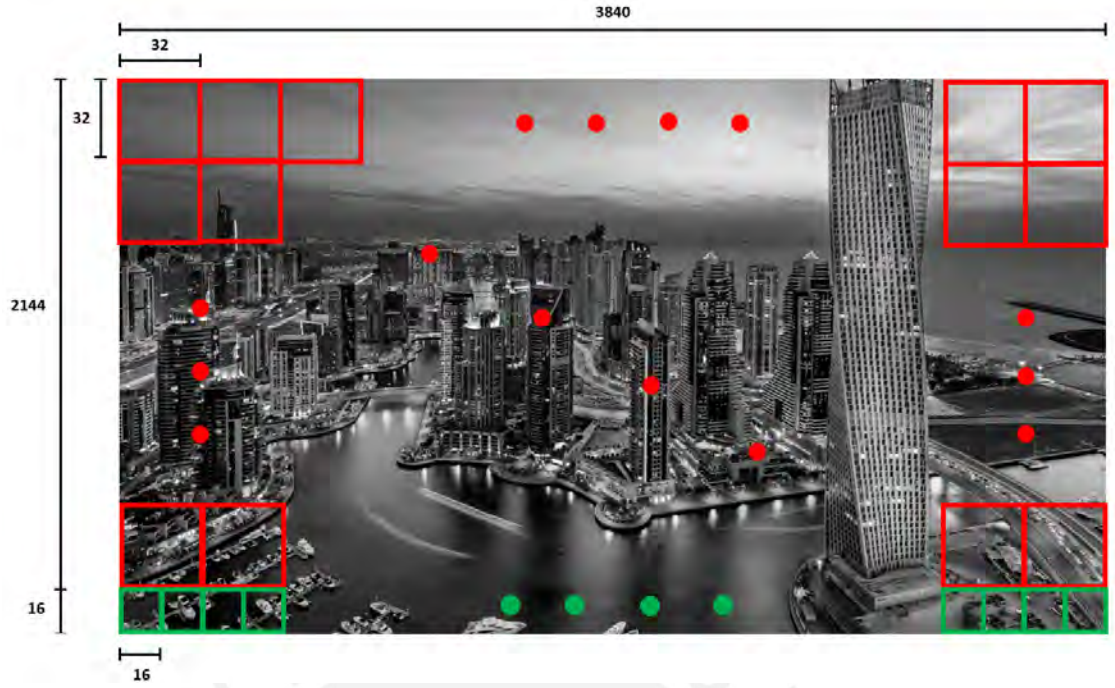


Figura 4.7: División de imagen 4K. La imagen se puede dividir en bloques de 32x32 pixeles (rojo) y una fila de bloques de 16x16 pixeles (verde).

Para la DCT se usaron los bloques de la imagen directamente como entrada y para la IDCT se utilizaron como entrada los coeficientes obtenidos en Matlab luego de procesar cada bloque por separado.

Luego del procesamiento de cada bloque, se comprobó el resultado de la misma manera que en la primera simulación tanto para DCT e IDCT. Una parte del procesamiento para obtener la DCT de la imagen 4K se muestra en la Figura 4.8. Se puede ver que el proceso consiste en ingresar los datos a las entradas X del circuito, esperar que salgan por las salidas Y, para luego ser comprobadas por el Testbench con los resultados obtenidos en Matlab. Este proceso se repite con todas los bloques de la imagen de prueba hasta que todas las operaciones hayan sido verificadas.

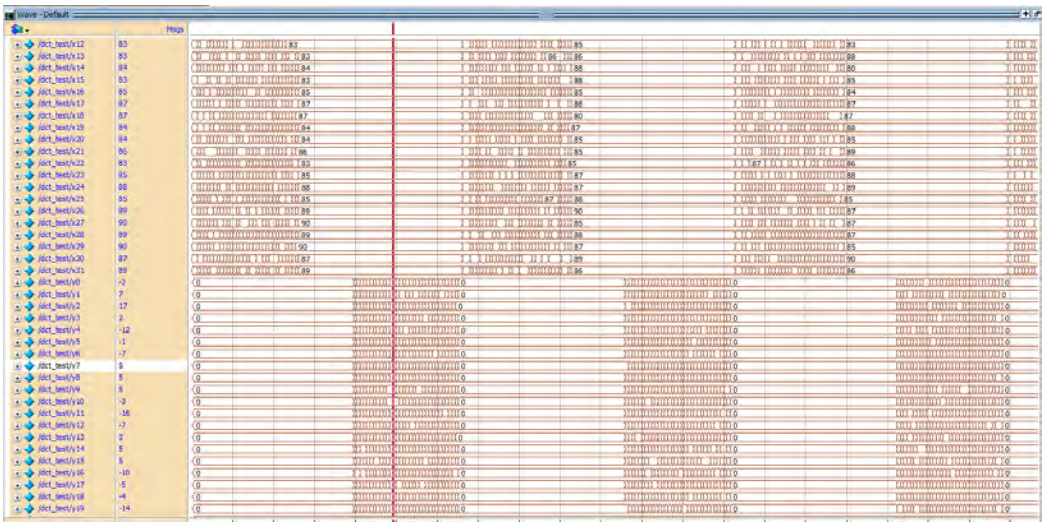


Figura 4.8: Parte del procesamiento de imagen 4K. Se puede como se ingresan los datos de un bloque de 32x32, se espera a tener la salida completa y se verifica los valores antes de ingresar las siguientes entradas.

4.2 Resultados de la síntesis de la arquitectura

La síntesis de la arquitectura diseñada se realizó para el dispositivo Zynq-7000 utilizando el software Vivado v2017.3 de la compañía Xilinx. En la Figura 4.6 se observa los resultados de la síntesis.

Para entender los resultados de la síntesis es necesario definir los siguientes términos según [19] y [20] y [21]:

- Tiempo de Setup (”Setup Time”): Es el tiempo necesario que una señal tiene que mantenerse estable antes de una señal de reloj para que esta sea correctamente reconocida.

- Tiempo de Hold (”Hold Time”): Es el tiempo necesario que una señal tiene que mantenerse estable despúes de una señal de reloj para que esta sea correctamente reconocida

- Setup Slack: Es la diferencia entre el tiempo requerido y el tiempo de llegada real. Si es positivo, se cumple con el tiempo requerido. Si es negativo, no se cumple con el requerimiento.

- Hold Slack: Es la diferencia entre el tiempo de llegada real y el tiempo requerido. Si es positivo, se cumple con el tiempo requerido. Si es negativo, no se cumple con el requerimiento.

- Worst Negative Slack: es el peor valor de Setup Slack dentro de todo el diseño (mientras más negativo sea un Slack, menos se cumple con el requerimiento de tiempo).

- Worst Hold Slack: el el peor valor de Hold Slack dentro de todo el diseño.

En la Figura 4.9 se puede ver los tiempos de Setup y Hold en un Flip-Flop.

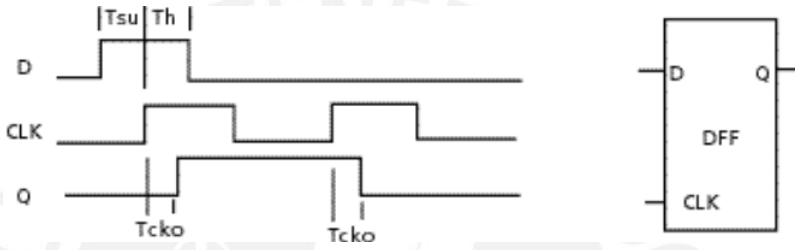


Figura 4.9: Ejemplo de tiempo de Setup (Tsu) y tiempo de Hold (Th). La señal D se tiene que mantener estable un tiempo Tsu antes del flanco de subida de CLK y luego tiene que mantenerse estable un tiempo Th después del flanco de subida [21].

Si los valores de Slack son positivos, esto significa que se cumplen los requerimientos de tiempo determinados. Además, en [22] se definen dos violaciones de tiempo posibles:

- Violación de “setup”: ocurre cuando el tiempo de propagación de un camino lógico es mayor que el tiempo requerido.

- Violación de “hold”: ocurre cuando una señal cambia demasiado rápido y los elementos de memoria no son capaces de guardar su valor.

En la Figura 4.10 se muestran los resultados de la síntesis para el periodo objetivo de 16.075 ns. El periodo de 16,075 ns se halló en base al objetivo de resolución 4k o 3840x2160 pixeles a 30fps. Asumiendo que los cuadros se dividen solo en bloques de 4x4 (es el peor caso porque es el que implica más procesamiento posible).

$$3840 * 2160@30fps = 518400bloques@30fps = 15552000bloques/s = 62208000columnas/s = 16.075ns/columna \tag{4.1}$$

(a)

Recursos	Consumo
LUT	41825
FF	20907
IO	805
BUFG	1

(b)

Figura 4.10: Resultados de la síntesis del circuito con periodo objetivo de 16.075 ns

Como se puede ver en el resultado, se tiene un slack positivo, así que es posible aumentar la frecuencia de operación del circuito. Finalmente se llegó la frecuencia de 135 MHz (periodo 7.4 ns) que es equivalente a procesar secuencias de resolución 3840x2160 @ 65.169 fps. En la Figura 4.11 se puede ver los resultados luego del cambio de periodo.

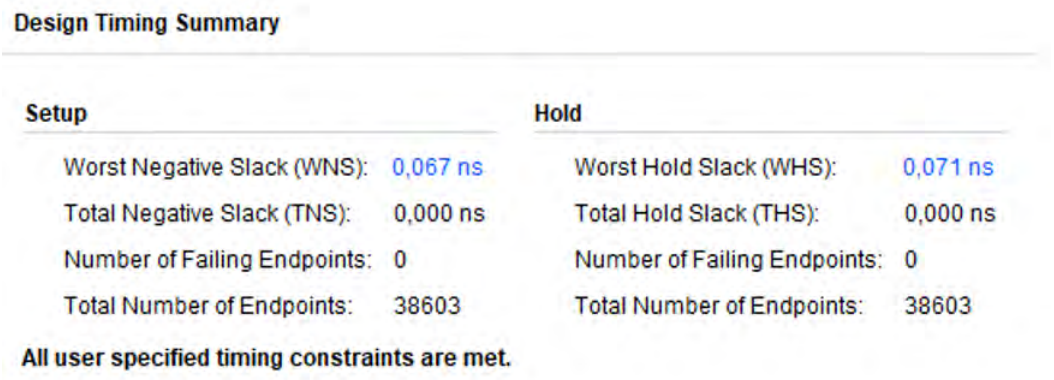


Figura 4.11: Resultados de la síntesis del circuito con periodo objetivo de 7.4 ns

En la Tabla 2 se puede observar una comparación de los resultados obtenidos con trabajos similares.

Tabla 4.1: Comparación de resultados con trabajos previos

	Este trabajo	Llamocca [13]	Meher et al. [11]	Chen et al. [12]	Tikekar et al. [14]
Tecnología	Zynq-7000	Zynq-7000	TSMC 90nm	180 nm	CMOS 40 nm
Pixeles por ciclo	32 (máximo)	32 (máximo)	16	2	2
Secuencias de video soportadas	3840x2160 @65fps (~7680x4320 @16fps)	Desde 3840x2160 @25 fps hasta 7860x4320 @193fps	7860x3420 @60fps	3840x2160 @30 fps	4096x3072 @30 fps
Frecuencia máxima de operación	135 MHz	200MHz	187MHz	125 MHz	200 MHz

En [17], se muestran diferentes variantes de arquitecturas, es por eso que puede soportar diferentes secuencias de video. Por otro lado, los resultados obtenidos

muestran que la arquitectura diseñada tiene mayor capacidad de procesamiento por ciclo, en comparación a tres de los trabajos anteriores, pero solo supera a uno de ellos en frecuencia de operación. Esto se debe al compromiso que hay entre los recursos que se deben utilizar para procesar más datos por cada ciclo de reloj y la frecuencia de operación de cada circuito.

Conclusiones

1. Se logró el diseñó la arquitectura conjunta de DCT e IDCT para los tamaños de TU de 4x4, 8x8, 16x16 y 32x32 en el lenguaje Verilog HDL que es capaz de procesar como máximo 32 datos por cada ciclo de reloj y que puede procesar secuencias de video de resolución 4k o 3840x2160 pixeles a 65 fps como máximo, o a resolución 8k igual a 7680x4320 pixeles a 16fps.

2. Los resultados obtenidos luego de la descripción de la arquitectura fueron comparados con los resultados de software en Matlab, el cual ha permitido demostrar que los coeficientes obtenidos eran correctos y que los datos de cada etapa intermedia estaban siendo calculados correctamente.

3. La arquitectura diseñada procesa la máxima capacidad de datos establecida por el estándar HEVC para los bloques de DCT e IDCT. Además, permite utilizar la misma arquitectura vara varios tamaños de transformada sin detener su procesamiento debido a las etapas de pipeline presentes en el circuito.

4. En comparación a trabajos anteriores, se obtuvo una mejor capacidad de procesamiento por ciclo de reloj, salvo trabajos que utilizan reconfiguración dinámica para variar la cantidad de recursos que se utilizan en base al tamaño de TU. Además, se puede procesar diferentes tamaños de TU sin demora en el circuito debido a las etapas de pipeline internas que permiten procesar diferentes datos al mismo tiempo. Sin embargo, esto causa que la frecuencia de operación sea menor a la de los trabajos pasado, lo cual es una consecuencia del compromiso entre la mayor cantidad de recursos utilizado en la arquitectura que soporte todos los tamaños de TU y la frecuencia de operación del circuito.

Recomendaciones

- En estudios futuros se debería tomar en consideración en maneras de reducir el buffer de transposición intermedia, puesto que ocupa gran espacio. Además, se podría cambiar el diseño de buffer para evitar la que las etapas pipeline se detengan en el momento que vacía sus resultados.
- Realizar pruebas con secuencias de videos reales y mostrarlas en tiempo real en una pantalla para verificar su funcionamiento.
- Modificar el software de referencia HM para añadir la arquitectura diseñada y probarlo en conjunto con todo el software del decodificador.

Bibliografía

[1] V. Sze, M. Budagavi, and G. J. Sullivan, *High efficiency video coding (HEVC)*. Springer, 2014.

[2] M. de Transportes y Comunicaciones del Perú, "Informe anual de evaluación del proceso de implementación de la televisión digital terrestre (TDT) en el perú." http://www.mtc.gob.pe/comunicaciones/autorizaciones/radiodifusion/documentos/TelevisionDigitalTerrestre/borrador%20de%20extracto%202017%20TDT.pdf.

[3] R. W. R. Gonzales, *Digital Image Processing*. Prentice-Hall, 2002.

[4] Microsoft, "Jpeg ycbcr support." https://msdn.microsoft.com/en-us/library/windows/desktop/dn424131(v=vs.85).aspx.

[5] I. Richardson, *H.264 and MPEG-4 Video Compressión: Video Coding for Next Generation*. John Wiley & Sons Inc, 2003.

[6] C. Sequoia, "HEVC – what are ctu, cu, ctb, cb, pb and tb." https://codesequoia.wordpress.com/2012/10/28/hevc-ctu-cu-ctb-cb-pb-and-tb/, 2015.

[7] E. Villegas, *Diseño de una arquitectura para la interpolación de quarter-pixel para estimación de movimiento según el formato H.264/AVC empleado en el estándar SBTVD de televisión digital terrestre*. Lima, Pontificia Universidad Católica del Perú, Facultad de Ciencias e Ingeniería: Tesis para optar el título de Ingeniero Electrónico, 2011.

[8] J. Soto, *Diseño de una arquitectura para estimación de movimiento fraccional según el estándar de codificación HEVC para video de alta resolución en tiempo real*. Lima, Pontificia Universidad Católica del Perú, Facultad de Ciencias e Ingeniería: Tesis para optar el título de Ingeniero Electrónico, 2016.

[9] J. F. Blinn, "What's that deal with the dct?," *IEEE Computer Graphics and Applications*, vol. 13, no. 4, pp. 78–83, 1993.

[10] M. Budagavi, A. Fuldseth, G. Bjontegaard, V. Sze, and M. Sadafale, "Core transform design in the high efficiency video coding (hevc) standard," *IEEE Journal of Selected Topics in Signal Processing*, pp. 1029–1041, 2013.

[11] M. Budagavi and V. Sze, "Unified forward + inverse transform architecture for hevc," *19th IEEE International Conference on Image Processing*, pp. 209–212, 2012.

[12] K. et al., "A low energy hevc inverse transform hardware," *IEEE Transactions on Consumer Electronics*, vol. 60, no. 4, pp. 754–761, 2014.

[13] Y. P. M. Voronenko, "Multiplierless multiple constant multiplication," *ACM Transactions on Algorithms*, vol. 3, no. 2, p. 11, 2007.

[14] P. T. and C. T. S. Chiang, "A reconfigurable inverse transform architecture design for hevc decoder," *2013 IEEE International Symposium on Circuits and Systems (ISCAS2013)*, pp. 1006–1009, 2013.

[15] P. Meher, S. Park, B. Mohanty, K. Lim, and C. Yeo, "Efficient integer dct architectures for hevc," *IEEE Transactions on Circuits and systems for Video Technology*, vol. 24, no. 1, pp. 168–178, 2014.

[16] Y. H. Chen and Y. F. Ko, "High-throughput idct architecture for high-efficiency video coding (hevc)," *International Journal of Circuit Theory and Applications*, 2017.

[17] D. Llamocca, "Self-reconfigurable architectures for hevc forward and inverse transform," *Journal of Parallel and Distributed Computing*, 2017.

[18] T. M., C. T. Huang, V. Sze, and Chandrakasan, "A. energy and area-efficient hardware implementation of hevc inverse transform and dequantization," *2014 IEEE International Conference on Image Processing (ICIP)*, pp. 2100–2104, 2014.

[19] Xilinx, "Ultrafast design methodology guide for the vivado design suite," 2017.

[20] Xilinx, "Vivado design suite user guide," 2017.

[21] C. Gaschet, "Understanding setup and hold times one key to successful designs,"

[22] W. F. Lee, "Vhdl coding and logic synthesis with synopsis," 2000.