

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA



**DISEÑO DE UNA ARQUITECTURA HARDWARE DEL
ALGORITMO BLOCK MERGING SEGÚN EL ESTÁNDAR HEVC
DE TRANSMISIÓN DE VIDEO 4K EN TIEMPO REAL**

Tesis para obtener el título profesional de Ingeniero Electrónico

AUTOR

Salvador Caceres Palacios

ASESORES

MSc. Ing. Ernesto Cristopher Villegas Castillo

MSc. Ing. Mario Andrés Raffo Jara

Lima, Junio, 2023

Informe de Similitud

Yo, Ernesto Cristopher Villegas Castillo,

docente de la Facultad de Ciencias e Ingeniería de la Pontificia

Universidad Católica del Perú, asesor(a) de la tesis/el trabajo de investigación titulado

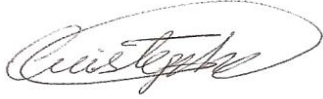
Diseño de una arquitectura hardware del algoritmo Block Merging según el estándar HEVC de transmisión de video 4k en tiempo real,

del autor Salvador Cáceres Palacios,

dejo constancia de lo siguiente:

- El mencionado documento tiene un índice de puntuación de similitud de 15%. Así lo consigna el reporte de similitud emitido por el software *Turnitin* el 21/06/2023.
- He revisado con detalle dicho reporte y la Tesis o Trabajo de Suficiencia Profesional, y no se advierte indicios de plagio.
- Las citas a otros autores y sus respectivas referencias cumplen con las pautas académicas.

Lugar y fecha: Lima, 21 de junio del 2023

Apellidos y nombres del asesor / de la asesora: <u>Villegas Castillo, Ernesto Cristopher</u>	
DNI: 45484048	
ORCID: 0009-0005-8586-512X	

Resumen

Las exigencias actuales en cuanto a calidad del contenido de video, a la par con el incremento del ancho de banda destinado a la transmisión de video, impulsan la necesidad de desarrollar estándares de codificación que sean más eficientes en cuanto a la calidad de reconstrucción de las imágenes decodificadas y a la cantidad de bits usados para la codificación. En este contexto, surge el nuevo estándar HEVC (*High Efficiency Video Coding*) denominado H.265 [1].

Si bien el estándar HEVC ha superado enormemente a sus predecesores con la inclusión de nuevas herramientas tales como el particionamiento en bloques de dimensión variable de los cuadros de la secuencia de video, así como nuevas técnicas de predicción de vectores de movimiento como el AMVP (*Advanced Motion Vector Prediction*), la etapa de codificación no es del todo eficiente.

Con el objetivo de mejorar la eficiencia de codificación del HEVC se incorpora la técnica de *Block Merging*. Esta técnica busca aprovechar las redundancias temporales y espaciales entre bloques de predicción vecinos, de manera que se puedan codificar en conjunto aquellos bloques que cumplan con ciertos criterios de selección [2]. Por lo tanto, el algoritmo de *Block Merging* tiene como finalidad encontrar, a partir de una lista de posibles candidatos, el bloque de predicción (bloque de píxeles de 8x8, 16x16, 32x32 o 64x64) cuyo vector de movimiento tenga asociado el mejor costo RD (*Rate-Distortion*). Al codificar en conjunto bloques de predicción, se evita enviar parámetros (como los vectores de movimiento) repetidas veces, logrando así una importante reducción en la cantidad de bytes a emplear durante la codificación de cuadros de video, considerando que resoluciones de 4k u 8k son cada vez más comunes en los contenidos audiovisuales [2].

El presente trabajo se enfoca en implementar la etapa de elección del mejor candidato usando el algoritmo de SATD (*Sum of Absolute Transformed Differences*) como parte del algoritmo de *Block Merging*. La arquitectura fue descrita usando Verilog-HDL y sintetizada en dispositivos FPGA (*Field Programmable Gate Array*) de la familia Kintex 7 de Xilinx. Por otra parte, se verificó el funcionamiento de la arquitectura mediante el

uso conjunto del simulador RTL (*Register Transfer Level*) de la herramienta Vivado Design Suite y un software de referencia en MATLAB (*Matrix Laboratory*), logrando una frecuencia de operación de 263.158 MHz. Tomando en cuenta una lista de tres candidatos y un particionamiento variable con un Parámetro de Cuantización QP (*Quantization Parameter*) en el rango de 22 hasta 36 considerado en trabajos pasados, la tasa de procesamiento de secuencias de video 4k (3840x2160) de la presente arquitectura se encuentra en el rango de 77.30 fps hasta 145.93 fps, lo cual se ajusta a los requerimientos por parte del estándar HEVC para transmitir video en tiempo real.

Palabras clave — HEVC, *Block Merging*, H.265, *Three Step Search*, Particionamiento Árbol Cuádruple, Lista de candidatos, Compensación de movimiento, Elección del mejor candidato, SATD, FPGA, MATLAB, Implementación en hardware.

Dedicado a mis queridos padres, María y Eusebio, quienes me han apoyado incondicionalmente desde la primera vez que abrí los ojos.

A mis queridos hermanos, Estrella y David, por la confianza y amistad que desde pequeños hemos tenido y seguiremos teniendo hasta los últimos días.

A mi abuelo Andrés, por cuidarme desde el cielo.

Muchas gracias, amada familia, por siempre estar conmigo y ser mi soporte día a día.

Agradezco también a mis asesores, Christopher y Mario, por compartirme sus conocimientos y experiencias a lo largo del desarrollo de esta tesis.

Al profesor Carlos Silva y al Grupo de Microelectrónica, por haberme permitido ser parte de un entorno donde pude experimentar y aprender nuevas tecnologías.

A mis colegas y amigos del Grupo de Microelectrónica, por los momentos de confraternidad y enseñanzas brindadas.

A la música electrónica, por acompañarme durante mi etapa escolar, universitaria y laboral.

Y a mi amigo Garrett, quien durante la pandemia me enseñó que los malos momentos pasan y que siempre podemos volver a levantarnos.

Índice General

Índice de figuras	i
Índice de tablas	v
Índice de abreviaturas	vii
Introducción	1
1 Desafíos actuales de los sistemas de compresión de video	3
1.1 Declaración de la problemática	3
1.2 Declaración del Marco Problemático	5
1.3 Objetivos	6
1.3.1 Objetivo Principal	6
1.3.2 Objetivos Secundarios	6
2 HEVC: Esquema Híbrido, Particionamiento y Etapa de Inter-Predicción con enfoque en la técnica de Block Merging	7
2.1 Esquema de Codificación Híbrido	7
2.2 Particionamiento de Árbol Cuádruple	8
2.2.1 Bloque del Árbol de Codificación	9
2.2.2 Bloque de Codificación	9
2.2.3 Bloque de Predicción	10
2.2.4 Bloque de Transformación	10
2.3 Etapa de Inter-Predicción	11
2.4 Problemática del Particionamiento de Árbol Cuádruple	12
2.5 Algoritmo de <i>Block Merging</i>	13
2.5.1 Construcción de la Lista de Candidatos	13
2.5.2 Compensación de Movimiento	15

2.5.3	Elección del mejor candidato	16
3	Consideraciones iniciales, requerimientos y diseño de la arquitectura	18
3.1	Propuesta de diseño	18
3.2	Arquitectura en Hardware vs. Software de Referencia	19
3.3	GPU, FPGA y ASIC como dispositivos de implementación	19
3.3.1	Unidad de Procesamiento de Gráficos	19
3.3.2	Circuito Integrado de Aplicación Específica	20
3.3.3	Matriz de Puertas Programables	20
3.4	Metodología planteada	22
3.4.1	Desarrollo del Software de Referencia	22
3.4.2	Diseño de la Arquitectura en Hardware	22
3.5	Consideraciones iniciales	23
3.5.1	Particionamiento de bloques de dimensión variable	23
3.5.2	Estimación de Movimiento	23
3.5.3	Uso de estructuras en MATLAB	25
3.6	Diseño de arquitectura hardware para la técnica <i>Block Merging</i>	26
3.6.1	Información obtenida a partir del Software de Referencia	26
3.7	Arquitectura en hardware	28
3.7.1	Cálculo del costo SATD	28
3.7.2	Bloque de cálculo de Distorsión	29
3.7.3	Bloque de cálculo de Transformada de Hadamard para bloques de dimensión variable	30
3.7.3.1	Transformada de Hadamard en una dimensión para bloques de 8x8 píxeles	30
3.7.3.2	Transformada de Hadamard en una dimensión para bloques de 16x16 píxeles	32
3.7.3.3	Transformada de Hadamard en una dimensión para bloques de 32x32 píxeles	33
3.7.3.4	Transformada de Hadamard en una dimensión para bloques de 64x64 píxeles	35
3.7.4	Bloque de cálculo de Transformada de Hadamard para filas	37
3.7.5	Buffer de Transposición	37
3.7.6	Bloque de cálculo de Transformada de Hadamard para columnas	39

3.7.7	Bloque de Suma de Valores Absolutos	39
3.7.8	Bloque de Escalamiento	41
3.7.9	Contador de carrera libre	42
3.7.9.1	Contador de carrera libre de 5 bits	43
3.7.9.2	Contador de carrera libre de 7 bits	43
3.7.10	Arreglo de Comparadores	44
3.7.11	Máquina de Estados	45
3.7.12	Elección del mejor candidato	48
4	Verificación y análisis de resultados	49
4.1	Almacenamiento de Bloques de Predicción y Vectores de Movimiento Enteros . .	49
4.2	Simulación del Bloque de Cálculo del costo SATD	52
4.3	Simulación del Bloque de Elección del mejor candidato	57
4.4	Síntesis de la arquitectura	58
	Conclusiones	62
	Recomendaciones	64
	Bibliografía	65

Índice de figuras

Figura 1:	Incremento en la calidad de video para los años 2018-2023 según el Reporte de Internet Anual de Cisco [3].	4
Figura 2:	Línea de tiempo de estándares de codificación de video publicados por el ITU-T (<i>International Telecommunication Union - Telecommunication Standardization Sector</i>) y el ISO (<i>International Organization for Standardization</i>)/IEC (<i>International Electrotechnical Commission</i>) [4].	4
Figura 3:	Esquema de Codificación Híbrido [4].	8
Figura 4:	<i>Coding Tree Blocks</i> para un cuadro de video [5].	9
Figura 5:	División de un CTB (<i>Coding Tree Blocks</i>) en <i>Coding Blocks</i> de dimensión variable [5].	10
Figura 6:	División de un CB (<i>Coding Blocks</i>) en <i>Prediction Blocks</i> de dimensión variable [5].	10
Figura 7:	División de un CB (<i>Coding Blocks</i>) en <i>Transform Blocks</i> de dimensión variable [5].	11
Figura 8:	Análisis del desplazamiento de bloques en la imagen actual respecto a una imagen de referencia previamente decodificada. A la izquierda, la imagen de referencia ya decodificada. A la derecha, la imagen actual en proceso de codificación [6].	11
Figura 9:	Comparación entre la cantidad de bloques particionados antes y después de aplicar <i>Block Merging</i> . (a) Cuadro de video a codificar. (b) Particionamiento <i>Quadtree</i> inicial con una alta cantidad de subdivisiones. (c) Subdivisión en bloques con parámetros de movimiento iguales, luego de aplicar <i>Block Merging</i> [2].	12
Figura 10:	Diagrama de flujo del algoritmo de Block Merging [7].	13
Figura 11:	Vecinos espaciales a considerar para la construcción de la lista de candidatos [2].	14

Figura 12:	(a) Candidatos espaciales para el bloque X. (b) Chequeo de redundancia de los vecinos espaciales [2].	14
Figura 13:	Vecinos temporales a considerar para la construcción de la lista de candidatos [2].	15
Figura 14:	(a) Bloque de píxeles de referencia. (b) Vectores de movimiento de componente entera y (c) fraccional [8].	16
Figura 15:	Diagrama Mariposa de la Transformada Rápida de Walsh-Hadamard aplicada a las filas. [9].	17
Figura 16:	NVIDIA V100 Tensor Core GPU (<i>Graphics Processing Unit</i>) [10].	20
Figura 17:	Estructura interna genérica de un FPGA [11].	21
Figura 18:	Porcentaje del Área de la imagen vs. Parámetro de Cuantización QP [12].	23
Figura 19:	Patrón de búsqueda del Algoritmo TSS (<i>Three Step Search</i>) [13].	25
Figura 20:	Ejemplo práctico de un Arreglo de Estructuras en MATLAB [14].	25
Figura 21:	Particionamiento para el frame 150 de la secuencia <i>Akiyo</i>	26
Figura 22:	Particionamiento para el frame 167 de la secuencia <i>Foreman</i>	26
Figura 23:	(a) Estructura obtenida para la secuencia de video <i>Foreman</i> . (b) Estructura que contiene los bloques de predicción del cuadro 167 de la secuencia <i>Foreman</i>	26
Figura 24:	Estimación de Movimiento Entera del cuadro 255 con respecto al cuadro 254 y el cuadro 255 compensado para la secuencia <i>Akiyo</i>	27
Figura 25:	Estimación de Movimiento Entera del cuadro 46 con respecto al cuadro 45 y el cuadro 46 compensado para la secuencia <i>Foreman</i>	27
Figura 26:	Estimación de Movimiento Entera del cuadro 15 con respecto al cuadro 14 y el cuadro 15 compensado para la secuencia <i>Bus</i>	27
Figura 27:	Estimación de Movimiento Entera del cuadro 31 con respecto al cuadro 30 y el cuadro 31 compensado para la secuencia <i>Ice</i>	27
Figura 28:	Diagrama de bloques de la arquitectura en hardware diseñada para el cálculo del costo SATD.	28
Figura 29:	Diagrama esquemático del bloque de cálculo de Distorsión.	29
Figura 30:	Diagrama esquemático del bloque de cálculo de Transformada de Hadamard en una dimensión para bloques de 8x8 píxeles.	31
Figura 31:	Diagrama esquemático del bloque de cálculo de Transformada de Hadamard en una dimensión para bloques de 16x16 píxeles.	32

Figura 32:	Diagrama esquemático del bloque de cálculo de Transformada de Hadamard en una dimensión para bloques de 32x32 píxeles.	34
Figura 33:	Diagrama esquemático del bloque de cálculo de Transformada de Hadamard en una dimensión para bloques de 64x64 píxeles.	36
Figura 34:	Diagrama esquemático base del Buffer de Transposición.	38
Figura 35:	Diagrama esquemático base para el bloque de Suma de Valores Absolutos.	40
Figura 36:	Diagrama esquemático del bloque de Escalamiento.	41
Figura 37:	Diagrama esquemático base para los contadores de carrera libre.	42
Figura 38:	Diagrama esquemático para los arreglos de comparadores.	44
Figura 39:	Diagrama esquemático para la Máquina de Estados Moore.	46
Figura 40:	Diagrama esquemático para el bloque de elección del mejor candidato. . .	48
Figura 41:	Particionamiento del cuadro 167 de la secuencia de video <i>Foreman</i>	50
Figura 42:	Bloques de Predicción obtenidos a partir del particionamiento del cuadro 167 de la secuencia de video <i>Foreman</i>	50
Figura 43:	Estructura de los vectores de Movimiento obtenidos para el cuadro 46 hallados con respecto al cuadro 45 de la secuencia de video <i>Foreman</i> . . .	51
Figura 44:	Bloque de 8x8 píxeles para el bloque actual de evaluación (bloque de referencia).	52
Figura 45:	Bloques de 8x8 píxeles para cada candidato en la lista. (a) Candidato de índice 0. (b) Candidato de índice 1. (c) Candidato de índice 2.	53
Figura 46:	Valores de costo SATD calculados a partir de MATLAB para cada candidato.	53
Figura 47:	Resultados obtenidos en la etapa de Distorsión. (a) Resultados obtenidos de MATLAB. (b) Resultados obtenidos del Testbench en Vivado.	53
Figura 48:	Resultados obtenidos en la etapa de Transformada de Hadamard para filas. (a) Resultados obtenidos de MATLAB. (b) Resultados obtenidos del Testbench en Vivado.	54
Figura 49:	Resultados obtenidos luego de aplicar la transpuesta en el Buffer de Transposición. (a) Resultados obtenidos de MATLAB. (b) Resultados obtenidos del Testbench en Vivado.	55
Figura 50:	Resultados obtenidos en la etapa de Transformada de Hadamard para columnas. (a) Resultados obtenidos de MATLAB. (b) Resultados obtenidos del Testbench en Vivado.	55

Figura 51:	Resultados obtenidos en la etapa de SAV (<i>Sum of Absolute Values</i>). (a) Resultados obtenidos de MATLAB. (b) Resultados obtenidos del Testbench en Vivado.	56
Figura 52:	Resultados obtenidos en la etapa de Escalamiento. (a) Resultados obtenidos de MATLAB. (b) Resultados obtenidos del Testbench en Vivado.	56
Figura 53:	Costo SATD final para el segundo candidato.	57
Figura 54:	Costo SATD final para el tercer candidato.	57
Figura 55:	Resultados de la elección del mejor candidato.	57
Figura 56:	(a) Bloque de referencia. (b) Mejor candidato (índice 2).	57
Figura 57:	Resumen de tiempos de diseño para la arquitectura propuesta.	58
Figura 58:	Frecuencia de operación para la arquitectura propuesta.	58
Figura 59:	Impacto de la cantidad de candidatos en la lista sobre la (a) <i>performance</i> y (b) complejidad [2].	60
Figura 60:	Porcentaje de bloques en el cuadro vs. Parámetro de Cuantización QP [12].	60

Índice de tablas

Tabla I:	COMPARACIÓN ENTRE GPU, FPGA y ASIC	19
Tabla II:	COMPARACIÓN DE PSNR (<i>Peak Signal-to-Noise Ratio</i>) Y NÚMERO DE PUNTOS DE BÚSQUEDA ENTRE ALGORITMOS DE ESTIMACIÓN DE MOVIMIENTO	24
Tabla III:	SEÑALES PARA EL BLOQUE DE CÁLCULO DE DISTORSIÓN . . .	30
Tabla IV:	SEÑALES PARA EL BLOQUE DE CÁLCULO DE TRANSFORMADA DE HADAMARD EN UNA DIMENSIÓN PARA BLOQUES DE 8x8 PÍXELES	31
Tabla V:	SEÑALES PARA EL BLOQUE DE CÁLCULO DE TRANSFORMADA DE HADAMARD EN UNA DIMENSIÓN PARA BLOQUES DE 16x16 PÍXELES	33
Tabla VI:	SEÑALES PARA EL BLOQUE DE CÁLCULO DE TRANSFORMADA DE HADAMARD EN UNA DIMENSIÓN PARA BLOQUES DE 32x32 PÍXELES	33
Tabla VII:	SEÑALES PARA EL BLOQUE DE CÁLCULO DE TRANSFORMADA DE HADAMARD EN UNA DIMENSIÓN PARA BLOQUES DE 64x64 PÍXELES	35
Tabla VIII:	SEÑALES PARA EL BLOQUE DE CÁLCULO DE TRANSFORMADA DE HADAMARD PARA FILAS	37
Tabla IX:	SEÑALES PARA EL BUFFER DE TRANSPOSICIÓN	38
Tabla X:	SEÑALES PARA EL BLOQUE DE CÁLCULO DE TRANSFORMADA DE HADAMARD PARA COLUMNAS	39
Tabla XI:	SEÑALES PARA EL BLOQUE DE SUMA DE VALORES ABSOLUTOS	40
Tabla XII:	DIMENSIÓN DEL BLOQUE VS. FACTOR DE ESCALAMIENTO . .	41
Tabla XIII:	SEÑALES PARA EL BLOQUE DE ESCALAMIENTO	42
Tabla XIV:	SEÑALES PARA EL CONTADOR DE CARRERA LIBRE DE 5 BITS .	43

Tabla XV:	SEÑALES PARA EL CONTADOR DE CARRERA LIBRE DE 7 BITS .	43
Tabla XVI:	SEÑALES PARA LOS ARREGLOS DE COMPARADORES	44
Tabla XVII:	CORRELACIÓN ENTRE LAS SEÑALES DEL DIAGRAMA FSM (<i>Finite State Machine</i>) Y LAS USADAS EN EL DISEÑO FINAL	45
Tabla XVIII:	SEÑALES PARA LA MÁQUINA DE ESTADOS MOORE	47
Tabla XIX:	SEÑALES PARA EL BLOQUE DE ELECCIÓN DEL MEJOR CANDIDATO	48
Tabla XX:	CICLOS DE RELOJ EMPLEADOS PARA ELEGIR AL MEJOR CANDIDATO	59
Tabla XXI:	CANTIDAD DE BLOQUES EN UN CUADRO DE VIDEO 4K (PARTICIONAMIENTO UNIFORME)	59
Tabla XXII:	PORCENTAJE DE BLOQUES EN UN CUADRO DE VIDEO	61
Tabla XXIII:	PERFORMANCE EN CUADROS POR SEGUNDO DE LA ARQUITECTURA PROPUESTA	61
Tabla XXIV:	COMPARACIÓN DEL PRESENTE DISEÑO CON ARQUITECTURAS DE TRABAJOS PREVIOS	61

Índice de abreviaturas

AMVP Advanced Motion Vector Prediction

ASIC Application-Specific Integrated Circuit

AVC Advanced Video Coding

CB Coding Blocks

CMOS Complementary Metal-Oxide Semiconductor

CPU Central Processing Unit

CTB Coding Tree Blocks

FME Fractional Motion Estimation

FPGA Field Programmable Gate Array

FS Full Search

FSM Finite State Machine

FWHT Fast Walsh-Hadamard Transform

GPU Graphics Processing Unit

HD High Definition

HDL Hardware Description Language

HEVC High Efficiency Video Coding

IEC International Electrotechnical Commission

ISO International Organization for Standardization

ITU-T International Telecommunication Union - Telecommunication Standardization Sector

MATLAB Matrix Laboratory

PB Prediction Blocks

PSNR Peak Signal-to-Noise Ratio

QP Quantization Parameter

RD Rate-Distortion

RTL Register Transfer Level

SAD Sum of Absolute Differences

SATD Sum of Absolute Transformed Differences

SAV Sum of Absolute Values

SSE Sum of Squared Errors

TB Transform Blocks

THS Total Hold Slack

TNS Total Negative Slack

TSS Three Step Search

UHD Ultra High Definition

VoIP Voice over Internet Protocol

WHS Worst Hold Slack

WNS Worst Negative Slack

Introducción

Con el empleo cada vez más común de resoluciones de video 4k y una creciente demanda de contenido de audiovisual por internet [3], surge la necesidad de incrementar la tasa de compresión de video. En este contexto, el estándar HEVC creado conjuntamente por el ITU-T *Video Coding Experts Group* y el ISO/IEC *Moving Picture Experts Group* tiene como principal objetivo reducir en un 50% la tasa de datos empleados para la codificación respecto a estándares pasados, manteniendo visualmente la misma calidad [1].

Una de las características resaltantes que ofrece el HEVC es la incorporación de un particionamiento de cuadros de video variable, el cual consta básicamente de una cuadrícula de píxeles que pueden ser de dimensión simétrica (8x8, 16x16, 32x32 y 64x64) o asimétrica (8x8, 8x4, 4x4, 4x8, 8x2 entre otros). Esta nueva forma de subdivisión de cuadros de video, a pesar de ser más eficiente respecto a los macrobloques usados en estándares pasados como el H.264, presentan ciertas redundancias entre bloques vecinos. En efecto, se observó una subdivisión excesiva de bloques que compartían los mismos parámetros de movimiento (vectores de movimiento) y que eran codificados por separado [2].

De esta forma, surge la técnica de *Block Merging*, la cual busca aprovechar las redundancias tanto temporales como espaciales, producto del particionamiento variable, entre bloques de píxeles vecinos en un cuadro de video. Es necesario precisar que uno de los procesos más importantes y demandantes a nivel de tiempo de procesamiento y recursos computacionales es la Estimación de Movimiento [1], por lo que la adición de una nueva técnica que complemente esta etapa no debe incrementar demasiado el tiempo de procesamiento y debe ser lo menos compleja posible para no usar demasiados recursos hardware.

El presente trabajo se enfoca en la implementación en hardware de la etapa de elección del mejor candidato como parte del algoritmo de *Block Merging*. La razón de esto es el aprovechamiento del paralelismo de las operaciones inherentes en el algoritmo, lo cual nos permite paralelizar tareas en una implementación en hardware a diferencia de una implementación en software que es ejecutada instrucción por instrucción a través de un CPU (*Central Processing Unit*). El diseño de la arquitectura de la etapa de elección del mejor candidato se realizó mediante el uso del lenguaje de descripción de hardware Verilog-HDL, y fue sintetizada en dispositivos FPGA de la familia Kintex 7 de Xilinx. Por otro lado, mediante el uso de un Software de Referencia en MATLAB basado en la técnica de *Block Merging* y el simulador RTL de la herramienta Vivado Design Suite de Xilinx, se pudo verificar la funcionalidad de la arquitectura diseñada.

Por último, el objetivo de este trabajo es lograr una tasa de procesamiento del bloque mayor a 30 cuadros por segundo, requerida por el estándar HEVC para garantizar una codificación de video en tiempo real de cuadros con una resolución de hasta 4k. El texto de la presente tesis está organizado de la siguiente manera: el capítulo 1 presenta los desafíos actuales de los sistemas de compresión de video y el marco problemático. En el capítulo 2 se presenta la teoría necesaria para entender la forma de operación de la técnica de *Block Merging*. El capítulo 3 se enfoca en presentar la arquitectura en hardware diseñada para implementar la etapa de elección del mejor candidato como parte de la técnica de *Block Merging*. Por último, el capítulo 4 presentará y analizará los resultados obtenidos del software y hardware del algoritmo, así como las conclusiones y recomendaciones del presente trabajo.

Capítulo 1

Desafíos actuales de los sistemas de compresión de video

1.1 Declaración de la problemática

De acuerdo al Reporte de Internet Anual de Cisco (del inglés *Cisco Annual Internet Report*) [3], el cual presenta las proyecciones y pronósticos del crecimiento de usuarios en internet, el uso de dispositivos, rendimiento de la red, tráfico de datos global, entre otros, para los años 2018-2023, se producirá un aumento considerable en la demanda anual de contenido de video a través de dispositivos móviles, de tal forma que para el 2023 el tráfico de datos móviles del mundo será ocupado mayoritariamente por video, superando a servicios tales como transferencia de archivos, reproducción de audio, navegación web, telefonía VoIP (*Voice over Internet Protocol*), entre otros.

Además, Cisco también ha estimado que la calidad de video que consuman los usuarios será cada vez más exigente, dando paso definitivamente al formato de video UHD (*Ultra High Definition*), con una resolución de hasta 7680 x 4320 píxeles (8K) a una transferencia de 30 o 60 cuadros por segundo. Además, este informe también indica que para el 2023, el 66% de televisores soportarán UHD en 4k, tal como se puede observar en la Fig. 1.

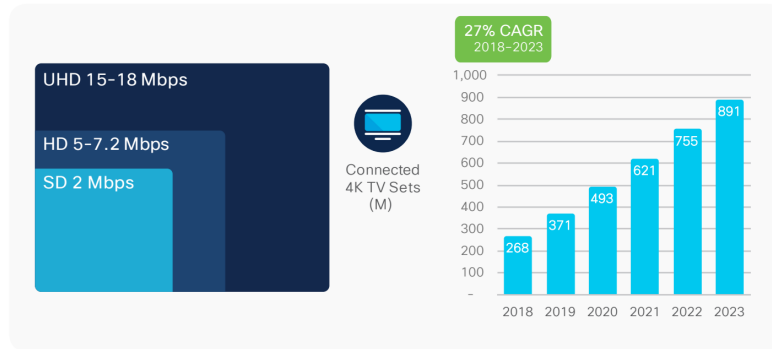


Fig. 1: Incremento en la calidad de video para los años 2018-2023 según el Reporte de Internet Anual de Cisco [3].

Con el objetivo de poder hacer frente a las exigencias del usuario, tanto a corto como a largo plazo, surge un nuevo estándar de codificación de video denominado HEVC [1]. El estándar HEVC fue desarrollado a través de una colaboración entre las 2 organizaciones responsables de la estandarización de la codificación de video: ITU-T *Video Coding Experts Group* y el ISO/IEC *Moving Picture Experts Group* [1]. Estas dos organizaciones han trabajado anteriormente en estándares tales como MPEG-1, MPEG-4 Visual, H.262/MPEG-2 Video y el H.264/MPEG-4 AVC (*Advanced Video Coding*) [4], cuya importancia aún prevalece en aplicaciones tales como la emisión de señales de televisión en HD (*High Definition*) a través de sistemas de transmisión satelital (por cable o terrestre), Blu-Ray, seguridad, grabación de video y conversación en tiempo real (videoconferencias, video chat, entre otros). Como se indica en [1], el estándar HEVC, también denominado estándar H.265, tiene como objetivo principal lograr un ahorro de hasta 50% en la tasa de datos usados para la codificación con respecto a estándares pasados, percibiendo la misma calidad visual.

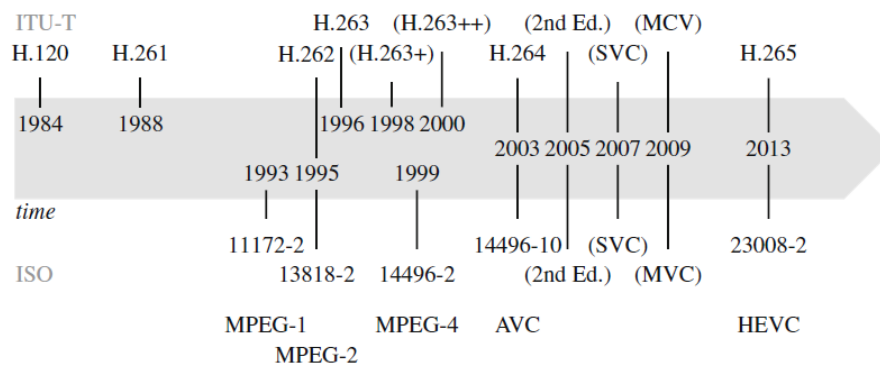


Fig. 2: Línea de tiempo de estándares de codificación de video publicados por el ITU-T y el ISO/IEC [4].

Una de las etapas más complejas y computacionalmente demandantes a nivel de recursos es la Estimación de Movimiento (del inglés *Motion Estimation*) [1]. Por ende, constantemente se realizan mejoras en los algoritmos de búsqueda con el objetivo de reducir la complejidad de este proceso manteniendo la eficiencia en la codificación, es decir, sin sacrificar la calidad de la imagen reconstruida [13], [15], [16]. Sin embargo, si bien se reduce el tiempo que tarda en realizarse la codificación, la cantidad de datos empleados para codificar cada cuadro de video se mantiene. Particularmente para el caso de los dispositivos móviles esto resulta crítico. Como ejemplo podría considerarse el hecho de que algunas aplicaciones mundialmente usadas, como es el caso de WhatsApp, requieren almacenar los videos en el móvil antes de poder verlos, utilizando de esta forma la limitada capacidad de almacenamiento interno que poseen muchos de estos dispositivos. Por otro lado, aplicaciones como la transmisión en vivo (del inglés *live broadcasting*) pueden llegar a agotar una elevada cantidad de datos móviles dependiendo de la calidad en la que se consuman estos contenidos audiovisuales. Por lo tanto, se puede concluir que existe una clara necesidad de reducir la cantidad de datos requeridos en la codificación de video.

Con el objetivo de solucionar esta problemática, nace la técnica llamada *Block Merging*, la cual busca aprovechar las redundancias tanto espaciales como temporales entre bloques de dimensión variable, resultado de una subdivisión excesiva por parte del Particionamiento de Árbol Cuádruple (del inglés *Quadtree Partitioning*) presente en el HEVC [2]. Así, el *Block Merging* permite reducir la cantidad de parámetros de movimiento a ser transmitidos mediante la fusión de bloques de predicción que comparten la misma información de movimiento, cumpliendo con las exigencias planteadas anteriormente.

1.2 Declaración del Marco Problemático

De la misma forma que con otras técnicas de codificación de video de la ITU-T e ISO/IEC, el HEVC solo estandariza la estructura, sintaxis y restricciones del flujo de datos, así como su mapeo para la obtención de secuencias de video decodificadas. Esto quiere decir que se permite realizar sobre el estándar tantas modificaciones como se deseen para lograr un equilibrio entre calidad de compresión, coste de implementación, complejidad, robustez entre otras consideraciones, teniendo siempre presente la normalización antes indicada.

En este sentido, la flexibilidad que ofrece el estándar ha contribuido a que se realicen diversas investigaciones sobre el algoritmo y posibles mejoras de la técnica Block Merging. Por lo tanto,

se puede afirmar que esta es un área de constante investigación ([7], [17], [18], [19]) en la que existe la posibilidad de brindar una mejora en cuanto a *performance* RD, complejidad, robustez, resistencia a errores, frecuencia de operación, costo computacional y paralelismo de operaciones que permitan implementaciones en dispositivos FPGA [2].

1.3 Objetivos

1.3.1 Objetivo Principal

El objetivo general de esta tesis es realizar el diseño de una arquitectura en hardware de la etapa de elección del mejor candidato como parte de la técnica de Block Merging aplicada a bloques de dimensión variable de 8x8, 16x16, 32x32 y 64x64 píxeles, según el estándar HEVC para la transmisión de video con resoluciones de hasta 4k en tiempo real con una tasa mayor a 30 cuadros por segundo.

1.3.2 Objetivos Secundarios

- Replicar el funcionamiento del algoritmo de Block Merging de la etapa de Inter Predicción para bloques de 8x8, 16x16, 32x32 y 64x64 píxeles.
- Replicar el funcionamiento del algoritmo de Estimación de Movimiento Entera.
- Verificar la funcionalidad del hardware en base a tiempo de operación y desempeño.
- Aplicar conceptos de HDL (*Hardware Description Language*) usando el lenguaje Verilog-HDL.

Capítulo 2

HEVC: Esquema Híbrido, Particionamiento y Etapa de Inter-Predicción con enfoque en la técnica de Block Merging

2.1 Esquema de Codificación Híbrido

Si bien cada uno de los estándares de codificación implementa su propio algoritmo y configuración, todos estos comparten una estructura básica esencial denominada esquema de codificación de video híbrido (del inglés *Hybrid Video Coding Scheme*). Se le llama híbrido debido a que mezcla tanto técnicas de predicción temporal entre cuadros de video como técnicas de codificación de la transformación del error de la predicción [4]. En la Fig. 3 se puede apreciar cómo el codificador (del inglés *encoder*) consiste del decodificador (del inglés *decoder*) y algunos bloques adicionales. Por otro lado, es necesario precisar que este esquema de codificación será aplicado a cada bloque de la imagen luego de realizarse el particionamiento en bloques de cada cuadro de la secuencia de video.

Este esquema busca generar una trama de bits (del inglés *bitstream*) que incluya toda la información necesaria para realizar tanto la transmisión como reconstrucción de los cuadros de la secuencia de video sin problemas. Esta información puede comprender el error de predicción, parámetros de la predicción (tipo de predicción y herramientas que se usaron), parámetros de filtrado, parámetros de cuantización y configuraciones adicionales [4].

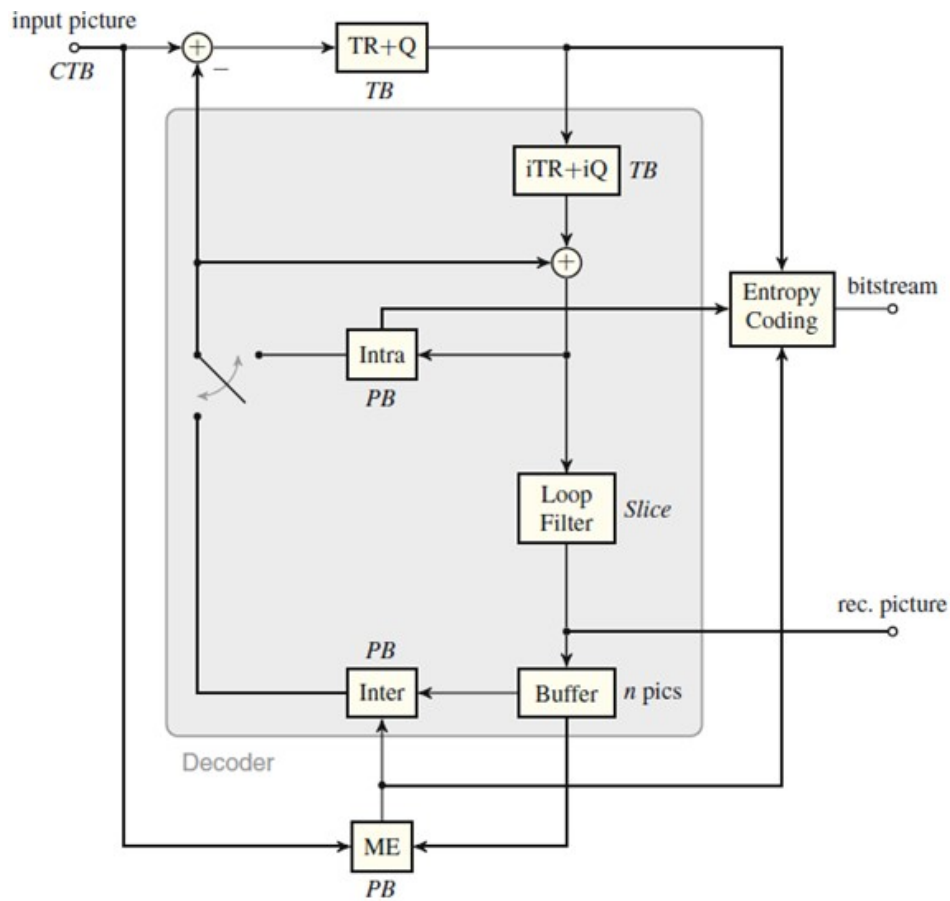


Fig. 3: Esquema de Codificación Híbrido [4].

Uno de los primeros pasos para comenzar con la codificación de video es hallar el error de predicción, el cual consiste en sustraer la señal de predicción de la señal de entrada, Para obtener la señal de predicción existen dos métodos: Intra-Predicción e Inter-Predicción. La decisión sobre cual tipo de predicción optar se basa en criterios como el de costo, optimización de la tasa de distorsión, disponibilidad de imágenes decodificadas, entre otros [4], [6].

2.2 Particionamiento de Árbol Cuádruple

A diferencia de estándares de codificación pasados, el HEVC incorpora un nuevo enfoque de particionamiento de cuadros de video llamado Particionamiento de Árbol Cuádruple y que responde a las siguientes problemáticas. En primer lugar, las resoluciones incrementan cada vez más, y por ende, el tamaño de los cuadros de video. De aquí se puede afirmar que para realizar una codificación eficiente se necesitan bloques incluso más grandes que los ampliamente usados macrobloques (del inglés *Macroblocs*) en el estándar H.264. Por otro lado, los contenidos de

video actual presentan una mayor cantidad de detalles que necesitarían ser abordados por un nuevo particionamiento capaz de manejar bloques de predicción de dimensiones más pequeñas que un macrobloque [1], [12]. La estructura del Particionamiento de Árbol Cuádruple es explicada en las siguientes secciones.

2.2.1 Bloque del Árbol de Codificación

De las siglas en inglés CTB (*Coding Tree Blocks*), son aquellos bloques o porciones de un cuadro de video producto de realizar un primer particionamiento, cuyas resoluciones pueden llegar a ser de 16x16, 32x32 o 64x64 píxeles. Cada uno de estos bloques se divide a su vez en CB (*Coding Blocks*) [5].

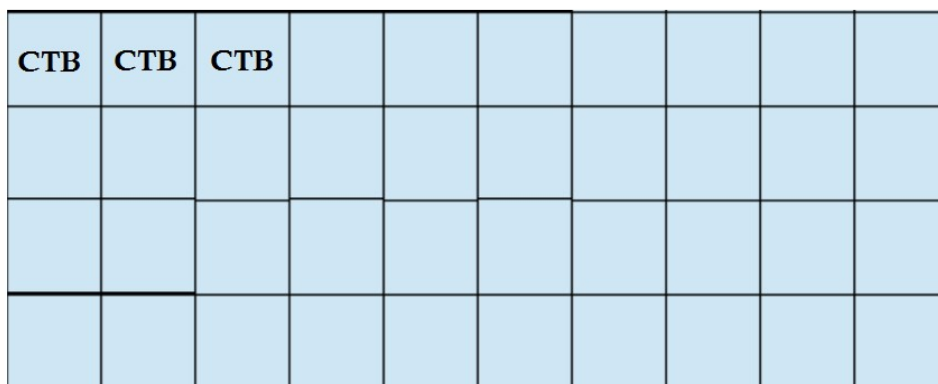


Fig. 4: *Coding Tree Blocks* para un cuadro de video [5].

2.2.2 Bloque de Codificación

De las siglas en inglés CB (*Coding Blocks*), son aquellos bloques cuya resolución se encuentra en un rango de 8x8 píxeles hasta la propia resolución del CTB del cual se originó. A partir de este bloque es posible saber el tipo de predicción realizar; sin embargo, a pesar de que la resolución mínima de este bloque es relativamente pequeña, todavía puede ser considerada muy grande para aplicar los métodos de predicción. Por lo tanto, se realiza una nueva división de este bloque en unos más pequeños llamados PB (*Prediction Blocks*) y TB (*Transform Blocks*) [5].

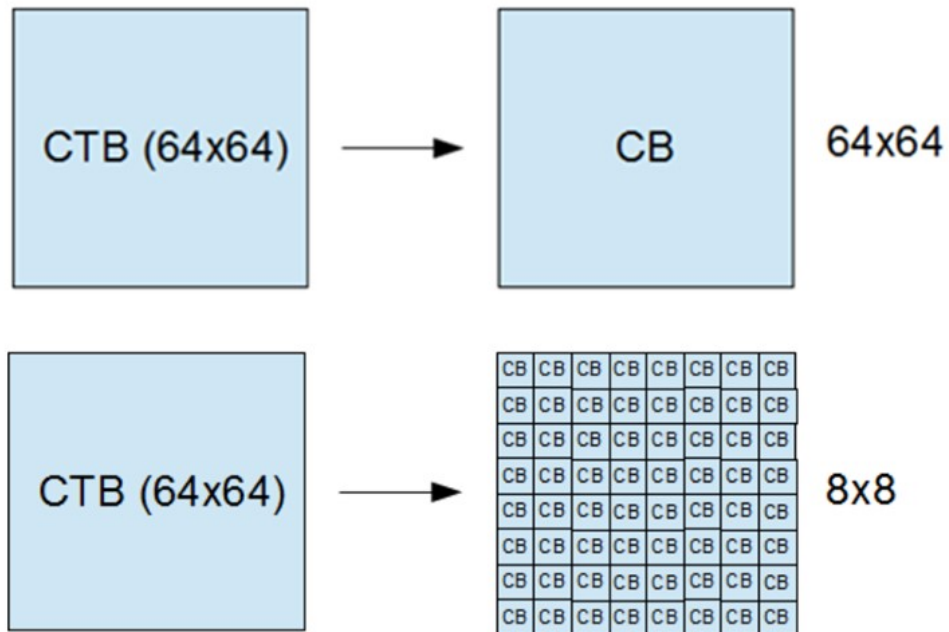


Fig. 5: División de un CTB en *Coding Blocks* de dimensión variable [5].

2.2.3 Bloque de Predicción

De las siglas en inglés PB (*Prediction Blocks*), este bloque es la unidad básica sobre la cual se ejecuta el Esquema Híbrido de Codificación de Video y, específicamente, es en donde se decide si aplicar la Intra Predicción o Inter Predicción. Además, dependiendo del tipo de predicción, cada Bloque de Codificación se dividirá en Bloques de Predicción de diferentes resoluciones que pueden ser tanto simétricas como asimétricas [5].

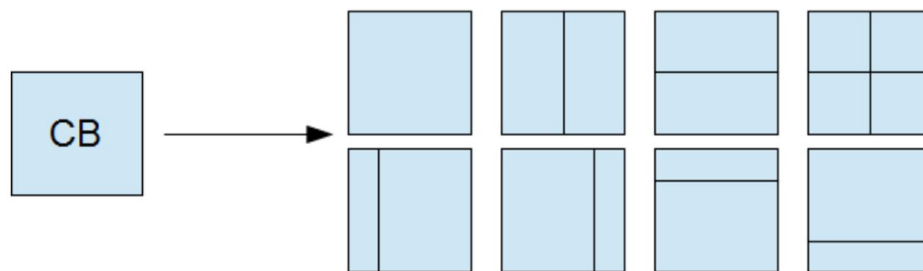


Fig. 6: División de un CB en *Prediction Blocks* de dimensión variable [5].

2.2.4 Bloque de Transformación

De las siglas en inglés TB (*Transform Blocks*), luego de realizarse la predicción se procede a codificar la distorsión (diferencia entre la imagen predecida y la imagen real) por medio de la

transformación. De acuerdo al estándar HEVC, el CB puede resultar ser de una dimensión muy grande para esto, por lo que se procede a realizar una subdivisión adicional en *Transform Blocks* de resoluciones simétricas [5].

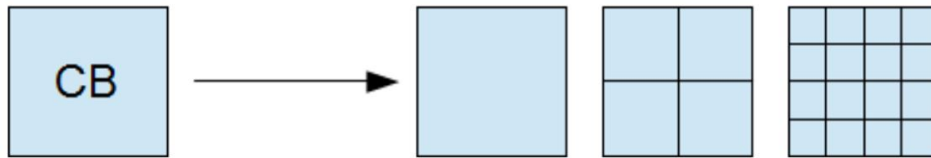


Fig. 7: División de un CB en *Transform Blocks* de dimensión variable [5].

2.3 Etapa de Inter-Predicción

El método de Inter-Predicción busca aprovechar la correlación temporal entre el bloque de análisis y los bloques de las imágenes previamente ya decodificadas. Para este caso, luego de realizarse el particionamiento en bloques de cada imagen de la secuencia de video, se halla la señal de predicción a partir de un vector de desplazamiento, llamado Vector de Movimiento (del inglés *Motion Vector* producto de la etapa de Estimación de Movimiento (del inglés *Motion Estimation*)).

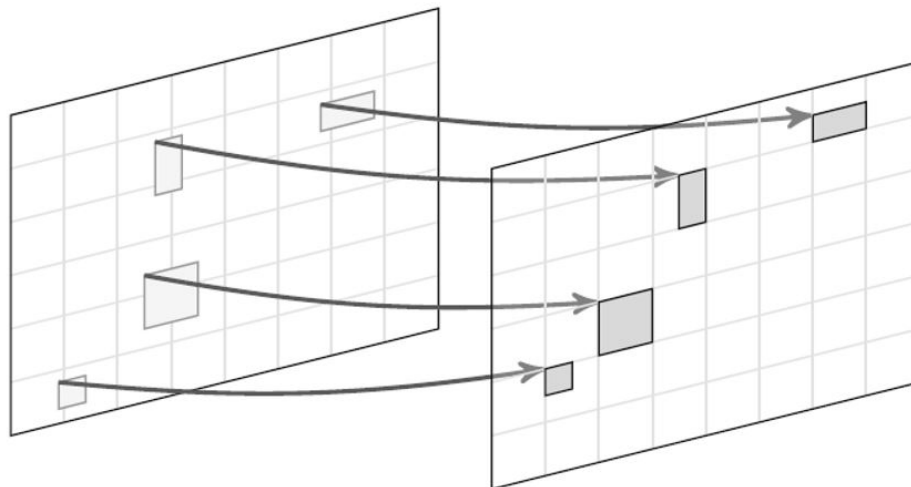


Fig. 8: Análisis del desplazamiento de bloques en la imagen actual respecto a una imagen de referencia previamente decodificada. A la izquierda, la imagen de referencia ya decodificada. A la derecha, la imagen actual en proceso de codificación [6].

2.4 Problemática del Particionamiento de Árbol Cuádruple

Como ya se había indicado anteriormente, el particionamiento *Quadtree* responde a un incremento de las resoluciones de video y a la alta cantidad de detalles presentes en los contenidos audiovisuales que no podrían ser abordadas por los macrobloques. Además, el hecho de poder subdividir una imagen en bloques de dimensión variable brinda una forma de optimización del costo RD [1], [2], [12].

Sin embargo, como se puede observar en la Fig. 9, dicho esquema de particionamiento puede llegar a realizar segmentaciones excesivas sobre la imagen que perjudican su eficiencia de codificación, ya que se estarían codificando por separado bloques cuya información de movimiento la comparten con bloques vecinos, información que comprende vectores de movimiento, cantidad de listas de imágenes de referencia usadas e índices de referencia. Por lo tanto, lo que se busca con la técnica de *Block Merging* es aprovechar las redundancias espaciales y temporales, de tal forma que se puedan codificar en conjunto bloques que compartan la misma información de movimiento. Por ende, durante la codificación no habría la necesidad de transmitir los mismos parámetros de movimiento, reduciendo de esta forma los datos empleados en el proceso de codificación.

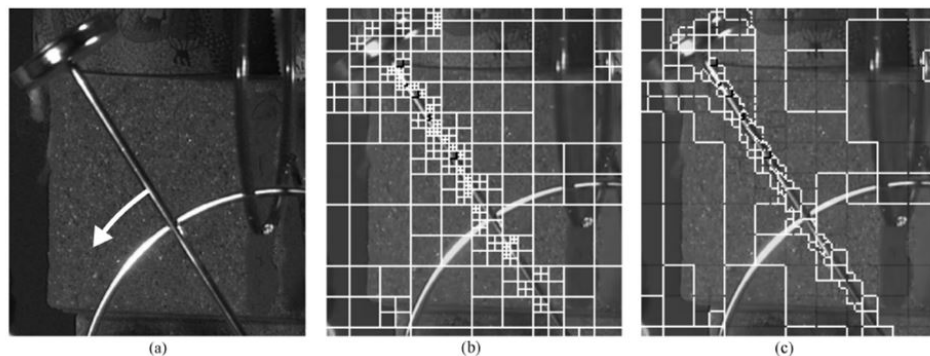


Fig. 9: Comparación entre la cantidad de bloques particionados antes y después de aplicar *Block Merging*. (a) Cuadro de video a codificar. (b) Particionamiento *Quadtree* inicial con una alta cantidad de subdivisiones. (c) Subdivisión en bloques con parámetros de movimiento iguales, luego de aplicar *Block Merging* [2].

2.5 Algoritmo de *Block Merging*

Este algoritmo basa su funcionamiento en tres etapas: construcción de la lista de candidatos, compensación de movimiento y la elección del mejor candidato. El diagrama de flujo presentado en la Fig. 10 resume el proceso de ejecución del algoritmo [2], [7].

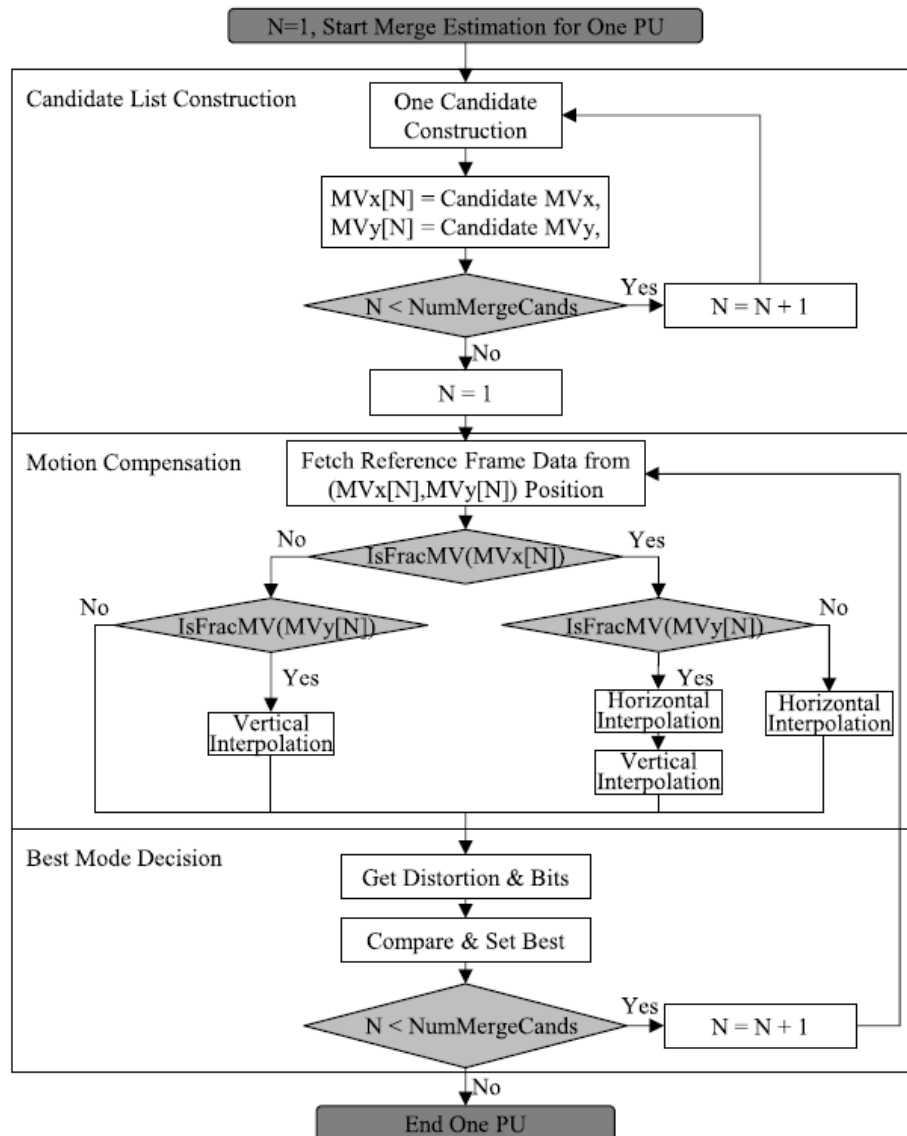


Fig. 10: Diagrama de flujo del algoritmo de Block Merging [7].

2.5.1 Construcción de la Lista de Candidatos

Del inglés *Candidate List Construction*, esta etapa consiste en evaluar una serie de posiciones predeterminadas, tanto espaciales como temporales, respecto al bloque de análisis actual. Cada posición está relacionada con un bloque de predicción, el cual tiene asociado un vector de

movimiento. Los candidatos espaciales son aquellos bloques ubicados alrededor del bloque de predicción actual en las posiciones mostradas en la Fig. 11. Según el estándar HEVC, el orden en el que se evalúan dichos candidatos es A_1 , B_1 , A_0 , B_0 y por último B_2 , formando así una lista de candidatos que podrá contener como máximo 4 candidatos espaciales. Es importante tener en cuenta que durante la evaluación de candidatos se consideran solo aquellos bloques cuyo método de predicción haya sido la Inter-Predicción. Con el objetivo de aumentar la eficiencia de codificación, se realiza un proceso de verificación de redundancia antes de agregar los candidatos a la lista, en donde estos son comparados en grupos también ya establecidos estadísticamente por el estándar, tal como se muestra en la Fig. 12. En caso alguno de ellos sea redundante, no será agregado a dicha lista. Es oportuno recalcar que si bien esta última operación de análisis de redundancia debería realizarse uno a uno entre cada candidato, analizar grupos de vectores de movimiento es suficiente para lograr un equilibrio entre complejidad y eficiencia en la codificación [2].

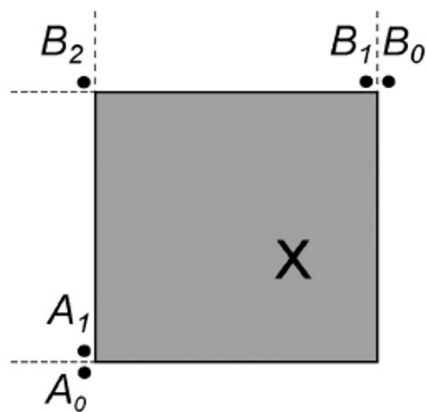


Fig. 11: Vecinos espaciales a considerar para la construcción de la lista de candidatos [2].

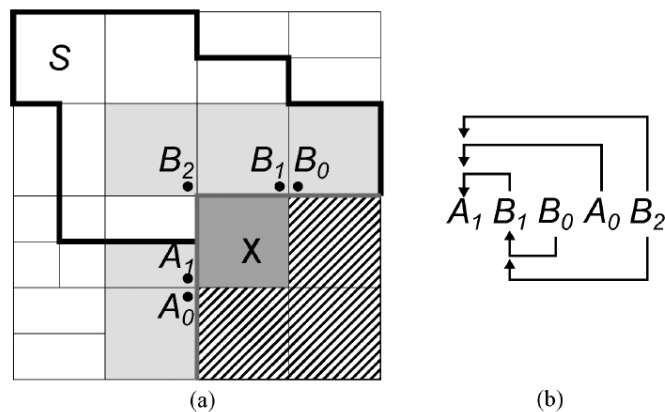


Fig. 12: (a) Candidatos espaciales para el bloque X. (b) Chequeo de redundancia de los vecinos espaciales [2].

De la misma manera, los candidatos temporales reúnen aquella información de movimiento perteneciente a cada uno de los bloques de predicción ubicados en posiciones específicas en el bloque colocado en el cuadro de referencia correspondiente. El orden de prioridad a considerar para este tipo de candidatos es C_0 , C_1 tal como se observa en la Fig. 13. El estándar HEVC indica que la lista de candidatos tendrá como máximo un solo candidato temporal. Por esa razón, se excluye el proceso de redundancia para este tipo de candidatos [2].

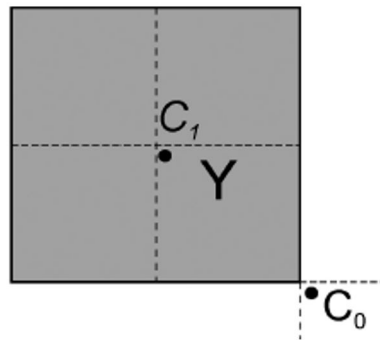


Fig. 13: Vecinos temporales a considerar para la construcción de la lista de candidatos [2].

Por último, si los candidatos espaciales y temporales no logran cubrir el número de candidatos que debería tener una lista, se procede a completar los espacios faltantes con candidatos adicionales, tales como los candidatos bi-predictivos y los candidatos de vector de movimiento cero. Por simplicidad, el presente trabajo está considerando un proceso con Uni-Predicción, por ende los candidatos adicionales a considerar serán únicamente los de vector de movimiento cero [2].

2.5.2 Compensación de Movimiento

Del inglés *Motion Compensation*, durante esta etapa lo que se hace es evaluar los componentes vectoriales de cada vector de movimiento pertenecientes a cada candidato y en caso estas componentes sean fraccionales, es decir tengan longitudes de mitad de píxel o un cuarto de píxel, se procederá a realizar la interpolación en la componente determinada, pudiendo ser esta horizontal o vertical. En caso los vectores de movimiento sean de naturaleza entera, se procederá a hallar el bloque de predicción compensado mediante el uso del vector de movimiento del candidato a ser evaluado [2], [7]. La Fig. 14 ejemplifica visualmente la diferencia entre un vector de movimiento entero y fraccional.

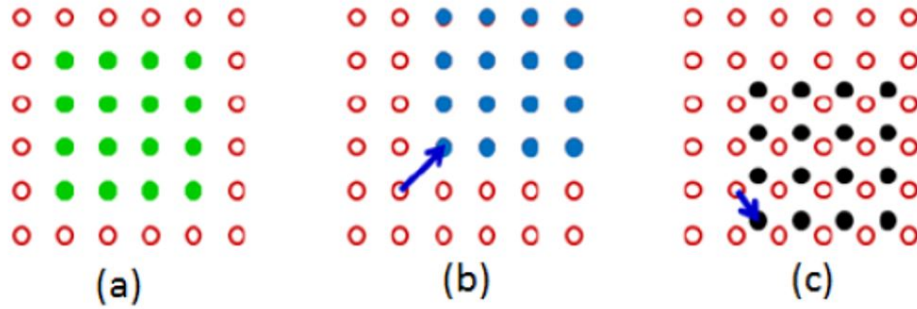


Fig. 14: (a) Bloque de píxeles de referencia. (b) Vectores de movimiento de componente entera y (c) fraccional [8].

2.5.3 Elección del mejor candidato

Del inglés *Best Mode Decision*, durante esta etapa lo que se busca es escoger aquel bloque de predicción cuyo costo RD sea el mínimo de entre todos los candidatos de la lista previamente construida [2]. Existen diversas formas de poder obtener el costo RD, en donde las más conocidas son la Suma de Errores Cuadrados, de las siglas en inglés SSE (*Sum of Squared Errors*), y la Suma de Diferencias Absolutas, de las siglas en inglés SAD (*Sum of Absolute Differences*). Sin embargo, dada la complejidad de la primera el costo computacional puede llegar a ser bastante alto. Por el contrario, la segunda técnica es relativamente más directa pero a cambio de su eficiencia [20]. El método que equilibra tanto complejidad como eficiencia es la Suma de Diferencias Transformadas Absolutas, de las siglas en inglés SATD (*Sum of Absolute Transformed Differences*) [21]. La ecuación genérica está indicada en la ecuación 2.1. Nótese el valor absoluto aplicado a la expresión $HT(i,j)$.

$$SATD = \sum_{i,j} |HT(i,j)| \quad (2.1)$$

La expresión HT hace referencia a la Transformada de Walsh-Hadamard, también llamada Transformada de Hadamard, aplicada a la distorsión entre bloques. Por ejemplo, para bloques de predicción de 8x8 píxeles, la Transformada de Hadamard se calcularía resolviendo la ecuación 2.2. La H hace referencia a la Matriz de Hadamard (del inglés *Hadamard Matrix*). La matriz a usar dependerá de la dimensión del bloque de predicción siendo evaluado. Considerando un bloque de 8x8 píxeles, se tendría una Matriz de Hadamard como la expresión 2.3. Si los bloques de predicción tuvieran una dimensión de 64x64 píxeles, la Matriz de Hadamard a usar sería 8 veces más grande que la usada anteriormente [9].

$$HT_{8x8} = H.Distortion.H^T \quad (2.2)$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix} \quad (2.3)$$

De igual forma que con la Transformada de Fourier, la Transformada de Walsh-Hadamard también posee un algoritmo computacionalmente más eficiente llamado la Transformada Rápida de Walsh-Hadamard, de las siglas en inglés FWHT (*Fast Walsh-Hadamard Transform*). Este método consta de aplicar una serie de operaciones matemáticas simples basadas en sumas y restas a la distorsión entre el bloque de predicción actual y el candidato de la lista de candidatos. Una característica de la Transformada de Hadamard es que es una matriz separable, permitiendo dividir la transformada en 2 etapas: transformación a las filas y transformación a las columnas. La operación sobre cada eje o dimensión de la imagen puede ser representada con un diagrama mariposa (del inglés *Butterfly Diagram*). Suponiendo que necesitamos aplicar la Transformada Rápida de Hadamard a las filas de Distorsión de bloques de 4x4 píxeles, el conjunto de operaciones a realizar estaría representado por la Fig. 15.

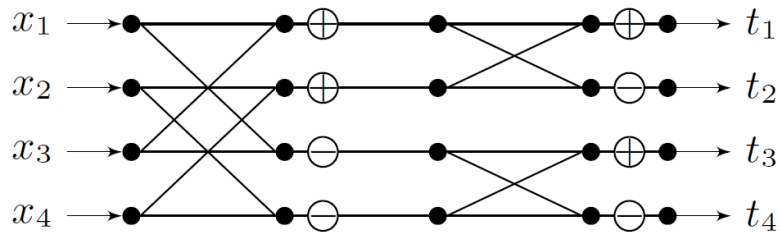


Fig. 15: Diagrama Mariposa de la Transformada Rápida de Walsh-Hadamard aplicada a las filas. [9].

Capítulo 3

Consideraciones iniciales, requerimientos y diseño de la arquitectura

3.1 Propuesta de diseño

Como resultado del particionamiento excesivo en bloques de los cuadros de video, se estarían codificando parámetros de movimiento (vectores de movimiento) en muchos casos redundantes, tanto a nivel espacial como temporal. La técnica del Block Merging aprovecha estas redundancias y permite fusionar y codificar de manera conjunta dichos bloques, minimizando así la cantidad de datos empleados al codificar y decodificar cuadros de video [2].

En este contexto, conviene saber en qué casos fusionar o no los bloques de predicción, así como conocer cuáles agrupar. Por ende, con el objetivo de lograr la mejor calidad de video posible, es de vital importancia emplear un algoritmo capaz de determinar el mejor candidato. Como se mencionó antes, tenemos varias formas de obtener el costo RD como la SSE, SAD y SATD. El algoritmo que logra un equilibrio entre complejidad y eficiencia es el SATD [20]. Por lo tanto, el presente trabajo se enfocará en el diseño e implementación en FPGA de una arquitectura en hardware de la etapa de elección del mejor candidato como parte de la técnica de *Block Merging* usando el algoritmo de SATD.

3.2 Arquitectura en Hardware vs. Software de Referencia

La razón de por qué optar por el diseño de una arquitectura en hardware o el desarrollo de un software de referencia va a depender principalmente de la aplicación a la cual esté dirigida la solución propuesta. Dado que en un estándar de codificación de video lo que se prioriza es poder transmitir en tiempo real los cuadros de la secuencia de video, entonces la solución que se proponga debe considerar implementarse de tal forma que sus operaciones se realicen tan pronto como sea posible e incluso con cierta independencia entre ellas.

A partir de este enfoque, si bien el desarrollo de una solución en software que implemente la elección del mejor candidato llega a ser totalmente funcional, no se aprovecha el paralelismo de operaciones ya que las sentencias en software son puramente secuenciales, es decir, cada operación necesita haber finalizado para empezar la siguiente. Por el contrario, el diseño de una arquitectura en hardware tiene la posibilidad de explotar el paralelismo de las operaciones del algoritmo, lo cual permite ejecutar operación tras operación sin necesidad de esperar a que alguna de ellas finalice (*Pipeline* de operaciones).

3.3 GPU, FPGA y ASIC como dispositivos de implementación

Para poder elegir el dispositivo a usar en la implementación de la arquitectura en hardware, se tomó como referencia la Tabla I, la cual detalla las ventajas y desventajas de cada dispositivo.

TABLA I
COMPARACIÓN ENTRE GPU, FPGA y ASIC

	GPU	FPGA	ASIC
Capacidad de Procesamiento	+++	++	+++
Paralelismo de operaciones	+	+++	+++
Consumo de potencia	+++	++	+

3.3.1 Unidad de Procesamiento de Gráficos

La Unidad de Procesamiento de Gráficos, de las siglas en inglés GPU (*Graphics Processing Unit*), es un coprocesador encargado principalmente del procesamiento de gráficos u operaciones de coma flotante, de manera que aligera la carga de trabajo del procesador central. Si bien este dispositivo permite emplear una gran cantidad de núcleos de procesamiento útiles para el cálculo

de operaciones complejas, falla en que posee un alto consumo de potencia. Además, para poder aprovechar la característica de paralelismo, es necesario que los algoritmos a implementar sean reformulados, lo cual aumenta la complejidad y tiempo del diseño [22], [23].



Fig. 16: NVIDIA V100 Tensor Core GPU [10].

3.3.2 Circuito Integrado de Aplicación Específica

En cuanto al Circuito Integrado de Aplicación Específica, de las siglas en inglés ASIC (*Application-Specific Integrated Circuit*), es un chip diseñado para realizar una función en particular, en comparación a otros dispositivos como los FPGA que son de propósito general. Dada la naturaleza bastante específica de un ASIC, durante la etapa de desarrollo es continuamente optimizado en cuanto a eficiencia de operación y consumo de potencia, logrando alcanzar el rendimiento esperado. Por ende, esto lo convierte en un dispositivo bastante más rápido en comparación con el GPU y FPGA. Sin embargo, dado que se trata de un diseño estático y que luego de la fabricación del chip ya no será posible adicionarle mejoras, el tiempo de desarrollo del diseño final optimizado es mucho más largo, convirtiéndolo en la alternativa más lenta. Además, posee un alto costo de implementación lo cual limita su uso a un sector industrial [22], [23].

3.3.3 Matriz de Puertas Programables

Por otro lado, se tiene a la Matriz de Puertas Programables, de las siglas en inglés FPGA (*Field Programmable Gate Array*), el cual es un tipo de circuito integrado compuesto por compuertas lógicas que serán interconectadas de acuerdo a la lógica definida por el diseñador. La estructura interna de un FPGA puede ser observada en la Fig. 17. Por medio del uso de lenguajes de descripción de hardware tales como VHDL y Verilog, los FPGA pueden reprogramar sus conexiones conforme las tareas que realice convirtiéndolo en un dispositivo de propósito general.

Así, se puede concluir que las ventajas más resaltantes son su flexibilidad en el desarrollo y la alta capacidad de aprovechar el paralelismo de operaciones; mientras que algunas de sus desventajas son su relativamente alto consumo de potencia y la dificultad del paradigma de diseño de un HDL, bastante diferente al de un lenguaje de programación convencional (C, Python, Java, Apex) [22], [23].

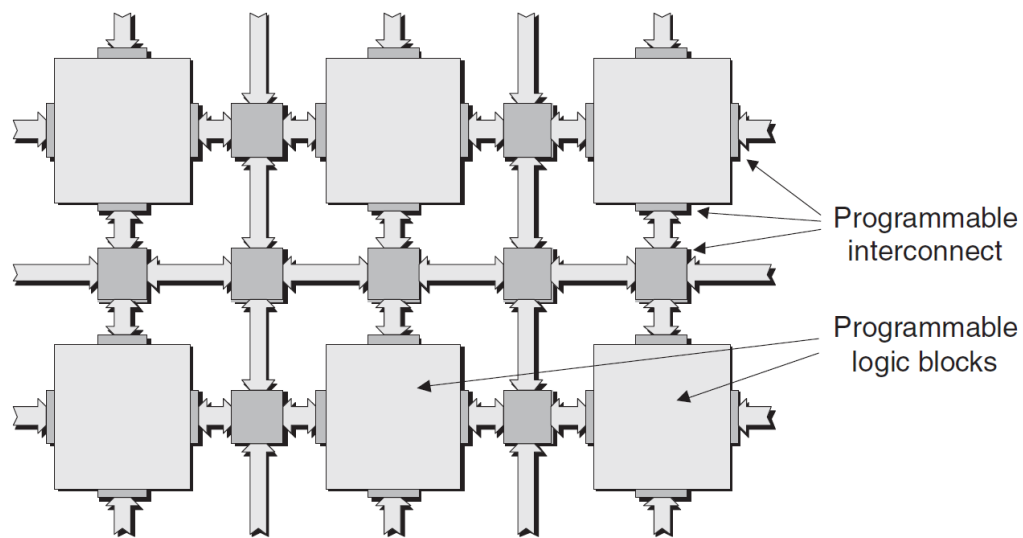


Fig. 17: Estructura interna genérica de un FPGA [11].

Por lo tanto, a partir de las comparaciones antes explicadas, se puede concluir que el FPGA se presenta como la alternativa más viable para implementar la arquitectura en hardware de la elección del mejor candidato, principalmente por presentar las características de paralelismo de operaciones, moderado consumo de potencia y alta flexibilidad en el diseño.

3.4 Metodología planteada

3.4.1 Desarrollo del Software de Referencia

Se partió con el desarrollo de un software de referencia que sea capaz de replicar la funcionalidad de la técnica de *Block Merging*. La herramienta que se empleó fue el software matemático MATLAB, el cual ofrece un entorno de desarrollo integrado con un lenguaje de programación propio. Las razones por la que se optó por desarrollar un software de referencia son principalmente las siguientes:

- Dado que el bloque de *Block Merging* forma parte de un conjunto de bloques más complejo (codificador), muchos de los datos de entrada a emplearse en este bloque eran resultados de bloques previos, ajenos al tema principal de esta tesis (los cuales se detallarán más adelante). Por esta razón fue necesario realizar un pre procesamiento de la información de los cuadros de la secuencia de video.
- Crear un conjunto de datos que puedan ser usados como estímulos por el Testbench de la arquitectura diseñada.
- Los resultados finales del software desarrollado sirven como herramienta para comprobar el correcto funcionamiento de la arquitectura.

3.4.2 Diseño de la Arquitectura en Hardware

A partir del flujo de operación de la técnica de *Block Merging*, con énfasis en la etapa de elección del mejor candidato, se procedió a diseñar en hardware los diversos circuitos digitales que permitan replicar esta operación. Con el objetivo de aprovechar al máximo las características del dispositivo en cuestión, FPGA, se buscó explotar el paralelismo de operaciones usando etapas de pipeline, de manera que la frecuencia de operación de la arquitectura diseñada no se vea comprometida por la alta complejidad del algoritmo SATD y se cumpla con los objetivos antes explicados. Asimismo, los resultados obtenidos a partir del diseño en hardware deben de guardar relación con los resultados arrojados por el software de referencia.

3.5 Consideraciones iniciales

3.5.1 Particionamiento de bloques de dimensión variable

Como se señaló antes, una de las características más resaltantes e innovadoras que propone el estándar HEVC es el particionamiento de la imagen en bloques de dimensión variable [1]. Uno de los principales beneficios de esta herramienta es que incrementa la eficiencia de codificación enormemente, a diferencia del caso en donde se consideraba una subdivisión estática de la imagen en macrobloques, sin importar la naturaleza de las imágenes en la secuencia de video [12].

A partir de la Fig. 18, la cual relaciona el porcentaje del área del cuadro de video ocupada por bloques de cierta dimensión vs. el valor del parámetro de cuantización, se puede notar que las dimensiones mayormente empleadas para los bloques de predicción son las de 64x64, 32x32, 16x16 y 8x8 píxeles. Por ende, con base en este resultado, el software de referencia programado implementará el particionamiento variable con dimensiones que abarquen los valores anteriormente indicados.

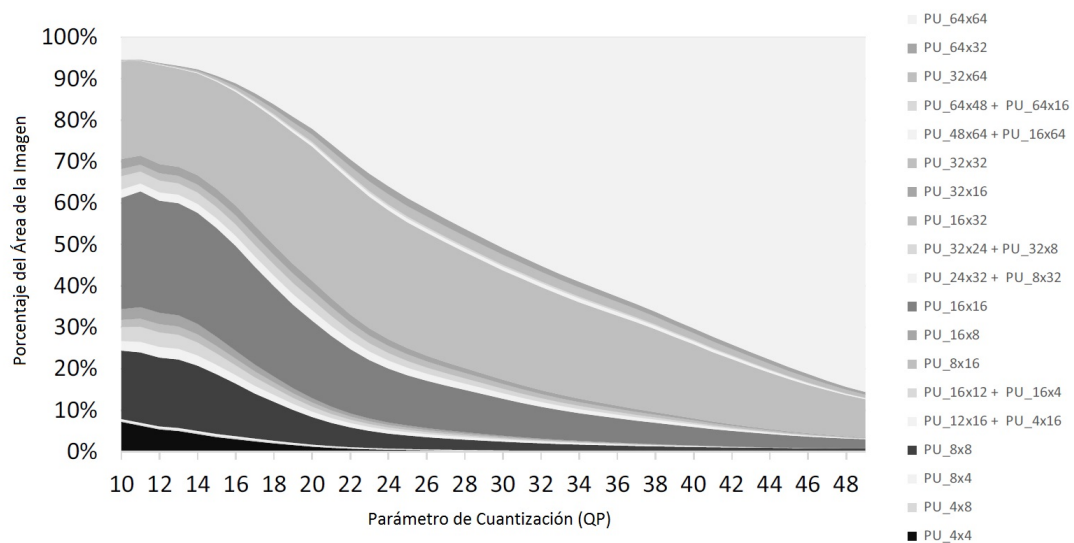


Fig. 18: Porcentaje del Área de la imagen vs. Parámetro de Cuantización QP [12].

3.5.2 Estimación de Movimiento

Dado que el procedimiento de la técnica de *Block Merging* realiza diversas operaciones con los vectores de movimiento, es necesario contar con esta información de movimiento previamente. Por lo tanto, dado que el diseño en hardware de una arquitectura de Estimación de Movimiento no está comprendido dentro de los objetivos principales del presente trabajo, será suficiente con replicar el funcionamiento del proceso de estimación de movimiento entera y usar los datos

obtenidos como datos de entrada del proceso de *Block Merging*. De igual forma, se considera que a cada bloque de predicción se le asocia un único vector de movimiento, cuyas componentes se encuentran tanto en el eje X como en el eje Y; en otras palabras, se tomará en cuenta la codificación de tipo Uni-Predicción.

Existen diferentes métodos para estimar el movimiento de los bloques de predicción, los cuales varían en cuanto a dificultad de implementación, tiempo de procesamiento, calidad de la imagen reconstruida, entre otros aspectos [16]. La Tabla II compara cada algoritmo de estimación de movimiento analizando el parámetro PSNR (*Peak Signal-to-Noise Ratio*), el cual cuantifica la calidad de reconstrucción de las imágenes, y la cantidad de puntos de búsqueda evaluados para cada caso. El algoritmo de Búsqueda Completa, de las siglas en inglés FS (*Full Search*), posee la mayor cantidad de puntos evaluados, lo cual resulta bastante coherente dado que este método realiza una búsqueda intensiva considerando una ventana de búsqueda amplia [24]. Por lo tanto, luego de un análisis de estos datos, se puede apreciar que el algoritmo TSS (*Three Step Search*) es el que mejor equilibra tanto la calidad en la reconstrucción de la imagen como la cantidad de puntos a evaluar durante la Estimación de Movimiento. Por ende, el algoritmo TSS es el que se usará para obtener los Vectores de Movimiento en el software de referencia programado en MATLAB.

TABLA II
COMPARACIÓN DE PSNR Y NÚMERO DE PUNTOS DE BÚSQUEDA ENTRE
ALGORITMOS DE ESTIMACIÓN DE MOVIMIENTO

Secuencias de video	Foreman QCIF		Akaio QCIF		Stefan QCIF	
	PSNR	Puntos de búsqueda	PSNR	Puntos de búsqueda	PSNR	Puntos de búsqueda
FS	26.0698	886.0101	38.6632	886.0101	25.5755	984.9192
TSS	24.1439	29.3131	38.6632	28.3131	25.2177	30.9293
NTSS	24.1157	34.5152	38.6632	14.8687	25.0231	35.3333
4SS	20.0052	22.3232	38.6632	14.6832	20.6187	20.158
DS	22.6815	37.8283	38.6632	11.5354	20.158	23.0631
ARPS	24.4228	17.0707	38.6632	5.1616	24.2563	10.5227

Fuente: Adaptado de [24].

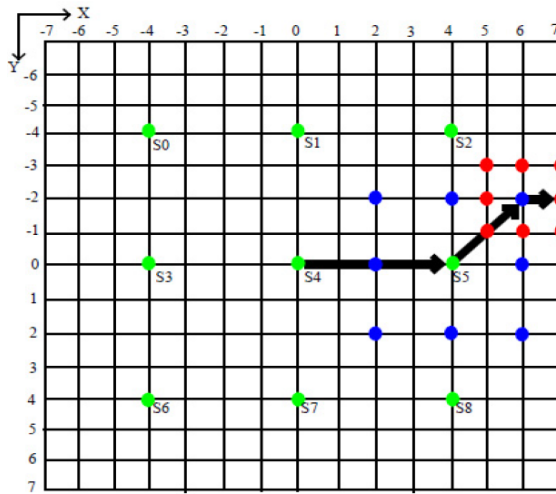


Fig. 19: Patrón de búsqueda del Algoritmo TSS [13].

3.5.3 Uso de estructuras en MATLAB

Debido a que se manipulará una gran cantidad de datos en las etapas de Particionamiento de Árbol Cuádruple, Estimación de Movimiento y *Block Merging* es necesario encontrar un tipo de dato en el entorno de MATLAB que ofrezca grandes prestaciones y flexibilidad en cuanto al manejo intensivo de datos. En este contexto, para el almacenamiento de las imágenes de la secuencia de video, vectores de movimiento, candidatos, lista de candidatos, nodos, entre otros datos necesarios para la ejecución del *Block Merging* se consideraron Arreglos de Estructuras (del inglés *Structure Array*), los cuales son un tipo de dato que agrupa información mediante contenedores denominados campos, que a su vez pueden ser de cualquier tipo de dato tales como números, caracteres, entre otros [14].

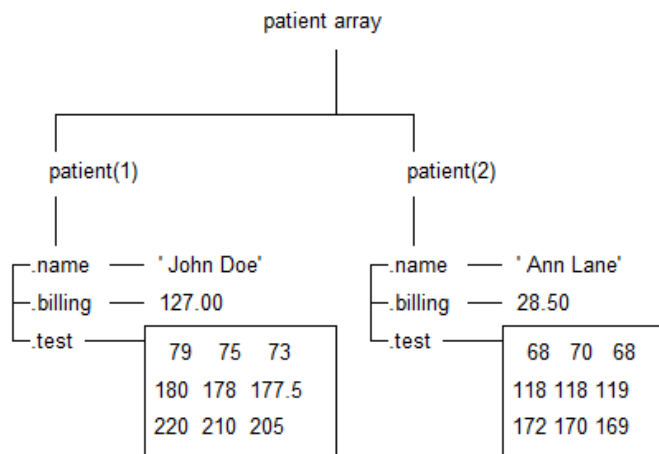


Fig. 20: Ejemplo práctico de un Arreglo de Estructuras en MATLAB [14].

3.6 Diseño de arquitectura hardware para la técnica *Block Merging*

3.6.1 Información obtenida a partir del Software de Referencia

Como se indicó previamente, es necesario de disponer de cierta información de cada cuadro de la secuencia de video, la cual servirá como dato de entrada para la arquitectura de la etapa de elección del mejor candidato del *Block Merging*. Esta información comprende principalmente los bloques de predicción de dimensión variable obtenidos a partir del Particionamiento de Árbol Cuádruple y los vectores de movimiento calculados a partir del algoritmo TSS, explicado en la sección 3.5.2. En cuanto al particionamiento variable, se realizaron pruebas con diversas secuencias de video, algunas de las cuales se detallan a continuación:



Fig. 21: Particionamiento para el frame 150 de la secuencia *Akiyo*.



Fig. 22: Particionamiento para el frame 167 de la secuencia *Foreman*.

Fields	idx	idx_ref	luma	quad	Fields	dim	nodo_row	nodo_col
1	1		144x176 uint8	1x264 struct	1	32	1	65
2	2		144x176 uint8	1x258 struct	2	32	65	65
3	3		144x176 uint8	1x258 struct	3	32	65	97
4	4		144x176 uint8	1x252 struct	4	16	97	17
5	5		144x176 uint8	1x255 struct	5	16	113	17
6	6		144x176 uint8	1x255 struct	6	16	1	33
7	7		144x176 uint8	1x267 struct	7	16	17	33
8	8		144x176 uint8	1x267 struct	8	16	49	33
9	9		144x176 uint8	1x270 struct	9	16	113	33
10	10		144x176 uint8	1x270 struct	10	16	1	49

(a)

(b)

Fig. 23: (a) Estructura obtenida para la secuencia de video *Foreman*. (b) Estructura que contiene los bloques de predicción del cuadro 167 de la secuencia *Foreman*.

Respecto al proceso de estimación de movimiento entera, se aplicó el algoritmo TSS para hallar los vectores de movimiento de diferentes secuencias de video, cuyos resultados se muestran a continuación:



Fig. 24: Estimación de Movimiento Entera del cuadro 255 con respecto al cuadro 254 y el cuadro 255 compensado para la secuencia *Akiyo*.



Fig. 25: Estimación de Movimiento Entera del cuadro 46 con respecto al cuadro 45 y el cuadro 46 compensado para la secuencia *Foreman*.



Fig. 26: Estimación de Movimiento Entera del cuadro 15 con respecto al cuadro 14 y el cuadro 15 compensado para la secuencia *Bus*.



Fig. 27: Estimación de Movimiento Entera del cuadro 31 con respecto al cuadro 30 y el cuadro 31 compensado para la secuencia *Ice*.

3.7 Arquitectura en hardware

A continuación se presentará en detalle los bloques que conforman la arquitectura en hardware del proceso de elección del mejor candidato, el cual es usado por el algoritmo de *Block Merging*.

3.7.1 Cálculo del costo SATD

Para procesar el cálculo del costo SATD, se asumirá que tanto los píxeles de bloques candidatos como los del bloque actual y compensado ya han sido obtenidos durante las etapas de creación de la Lista de Candidatos y Compensación de Movimiento respectivamente, como parte de otros bloques funcionales dentro del codificador. De esta forma, el flujo de operación del cálculo del costo SATD se detalla a continuación:

1. Se reciben filas de píxeles de los bloques actual y compensado, los cuales se usarán para hallar la distorsión entre ellos.
2. Para cada fila de distorsión calculada, se empezará a procesar la Transformada de Hadamard aplicada a cada una de estas filas.
3. A medida que se obtengan las filas coeficientes, se irán guardando progresivamente en el Buffer de Transposición (modo de escritura).
4. Cuando se tengan listas todas las filas de coeficientes, se comenzará a procesar la Transformada de Hadamard aplicada a las columnas. Para ello, el Buffer de Transposición cambiará a modo lectura.
5. A medida que se obtengan los columnas de coeficientes, se calculará el valor absoluto de cada uno de ellos, para luego sumarlos y almacenar este resultado en un acumulador.
6. Luego de procesar todas las columnas de coeficientes y aplicar un factor de escalamiento, el valor resultante será el costo SATD final correspondiente al candidato evaluado.

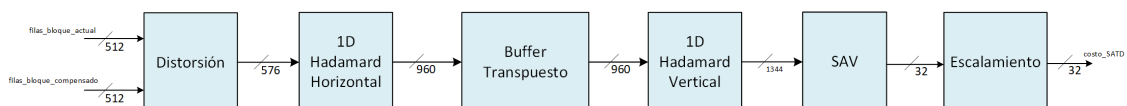


Fig. 28: Diagrama de bloques de la arquitectura en hardware diseñada para el cálculo del costo SATD.

3.7.2 Bloque de cálculo de Distorsión

El presente bloque tiene por función calcular la distorsión entre las filas de píxeles de los bloques actual y compensado. Dado que cada píxel abarca un ancho de 8 bits (rango de valores de 0 a 255) y que la dimensión más grande de bloques es de 64x64 píxeles, por cada fila de píxeles se tendría un ancho máximo de 512 bits. Asimismo, la diferencia obtenida entre cada píxel abarca un ancho de 9 bits, por lo que para una fila de 64 píxeles, la salida del bloque de Distorsión sería de 576 bits. Las señales de entrada y salida a considerar para el bloque de Distorsión se encuentran detalladas en la Tabla III.

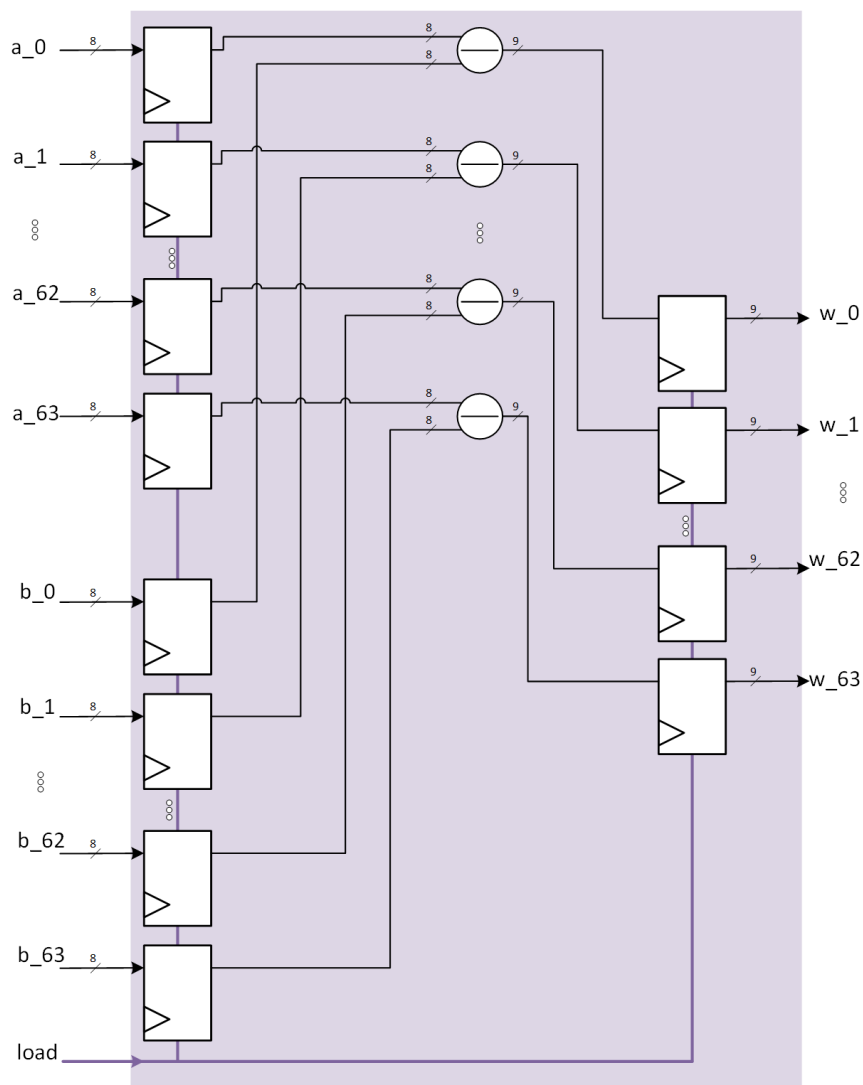


Fig. 29: Diagrama esquemático del bloque de cálculo de Distorsión.

TABLA III
SEÑALES PARA EL BLOQUE DE CÁLCULO DE DISTORSIÓN

Señales			
Nombre	Tipo	# Bits	Descripción
clock	Entrada	1	Señal de clock
reset_n	Entrada	1	Señal de reset asíncrona
clear_sync	Entrada	1	Señal de clear síncrona
load	Entrada	1	Señal de carga síncrona
a	Entrada	512	Fila de 64 píxeles del bloque actual
b	Entrada	512	Fila de 64 píxeles del bloque compensado
w	Salida	576	Distorsión obtenida para una fila 64 píxeles

3.7.3 Bloque de cálculo de Transformada de Hadamard para bloques de dimensión variable

El objetivo de este bloque es calcular la Transformada de Hadamard ya sea para filas como para columnas. Dada la alta complejidad de esta operación y teniendo en cuenta la alta variabilidad del particionamiento, se decidió modularizar y desarrollar bloques más pequeños que se detallarán a continuación.

3.7.3.1 Transformada de Hadamard en una dimensión para bloques de 8x8 píxeles

Haciendo uso de la Transformada Rápida de Walsh-Hadamard FWHT explicada en la sección 2.5.3, el módulo de operación base que procesará la Transformada Rápida de Hadamard en una dimensión para bloques de predicción de 8x8 píxeles es presentado en la Fig. 30. Las señales de entrada y salida a considerar para este bloque se encuentran detalladas en la Tabla IV. Nótese que la cantidad de bits para la entrada w y la salida s está representada por una variable k , lo que significa que el bloque en cuestión es parametrizable y será reutilizado en la arquitectura final.

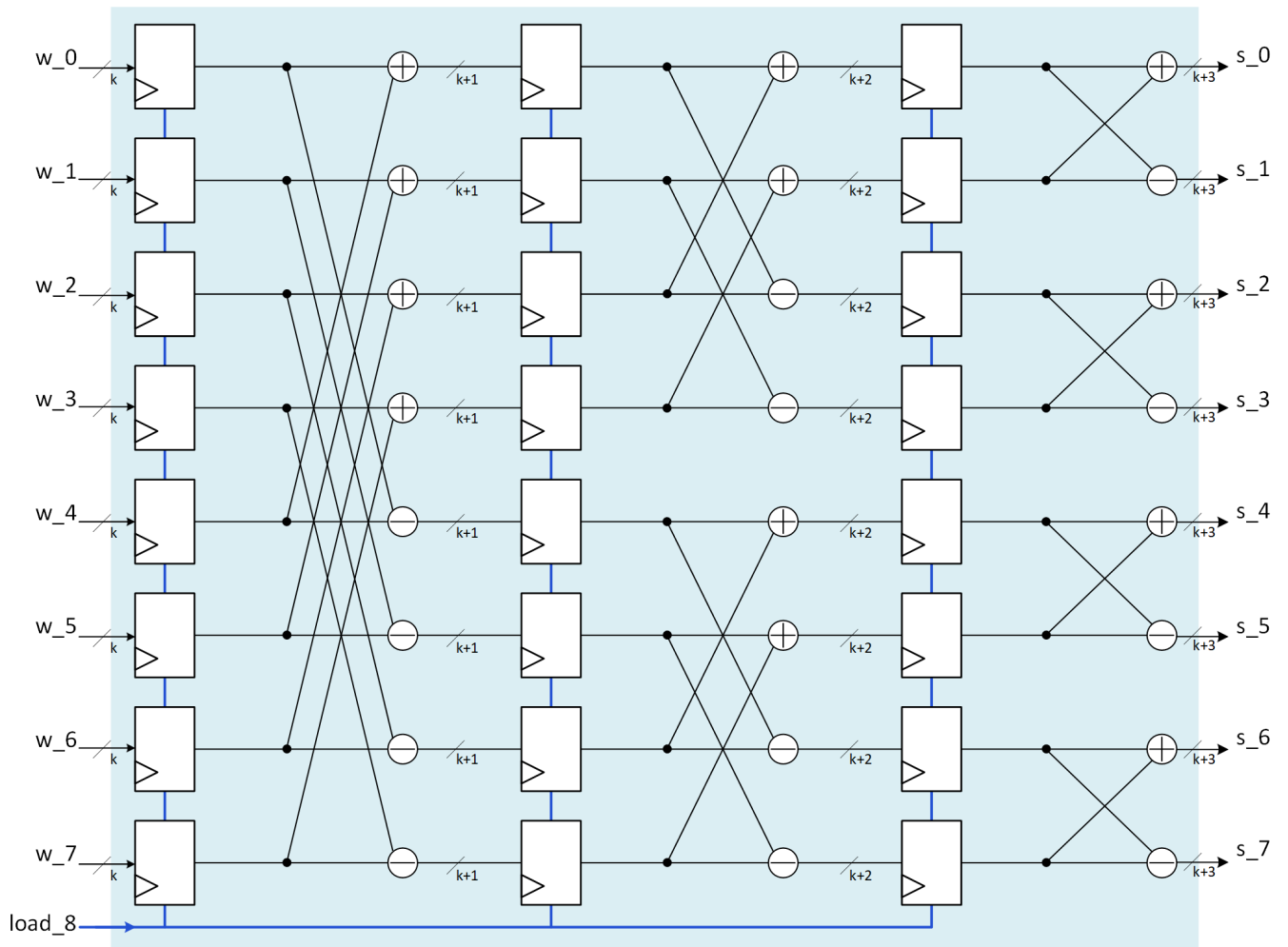


Fig. 30: Diagrama esquemático del bloque de cálculo de Transformada de Hadamard en una dimensión para bloques de 8x8 píxeles.

TABLA IV

SEÑALES PARA EL BLOQUE DE CÁLCULO DE TRANSFORMADA DE HADAMARD EN UNA DIMENSIÓN PARA BLOQUES DE 8x8 PÍXELES

Señales			
Nombre	Tipo	# Bits	Descripción
clock	Entrada	1	Señal de clock
reset_n	Entrada	1	Señal de reset asíncrona
clear_sync	Entrada	1	Señal de clear síncrona
load_8	Entrada	1	Señal de carga síncrona para el bloque hadamard 8x8
w	Entrada	8*k	Filas/Columnas de 8 coeficientes parciales
s	Salida	8*(k + 3)	Filas/Columnas de 8 coeficientes parciales

3.7.3.2 Transformada de Hadamard en una dimensión para bloques de 16x16 píxeles

Este bloque reutiliza el módulo de la sección 3.7.3.1 y añade una etapa más de procesamiento, tal como se observa en la Fig. 31. Las señales de entrada y salida a considerar para este bloque se encuentran detalladas en la Tabla V. Nótese que la cantidad de bits para la entrada w y la salida s está representada por una variable k , lo que significa que el bloque en cuestión es parametrizable y será reutilizado en la arquitectura final.

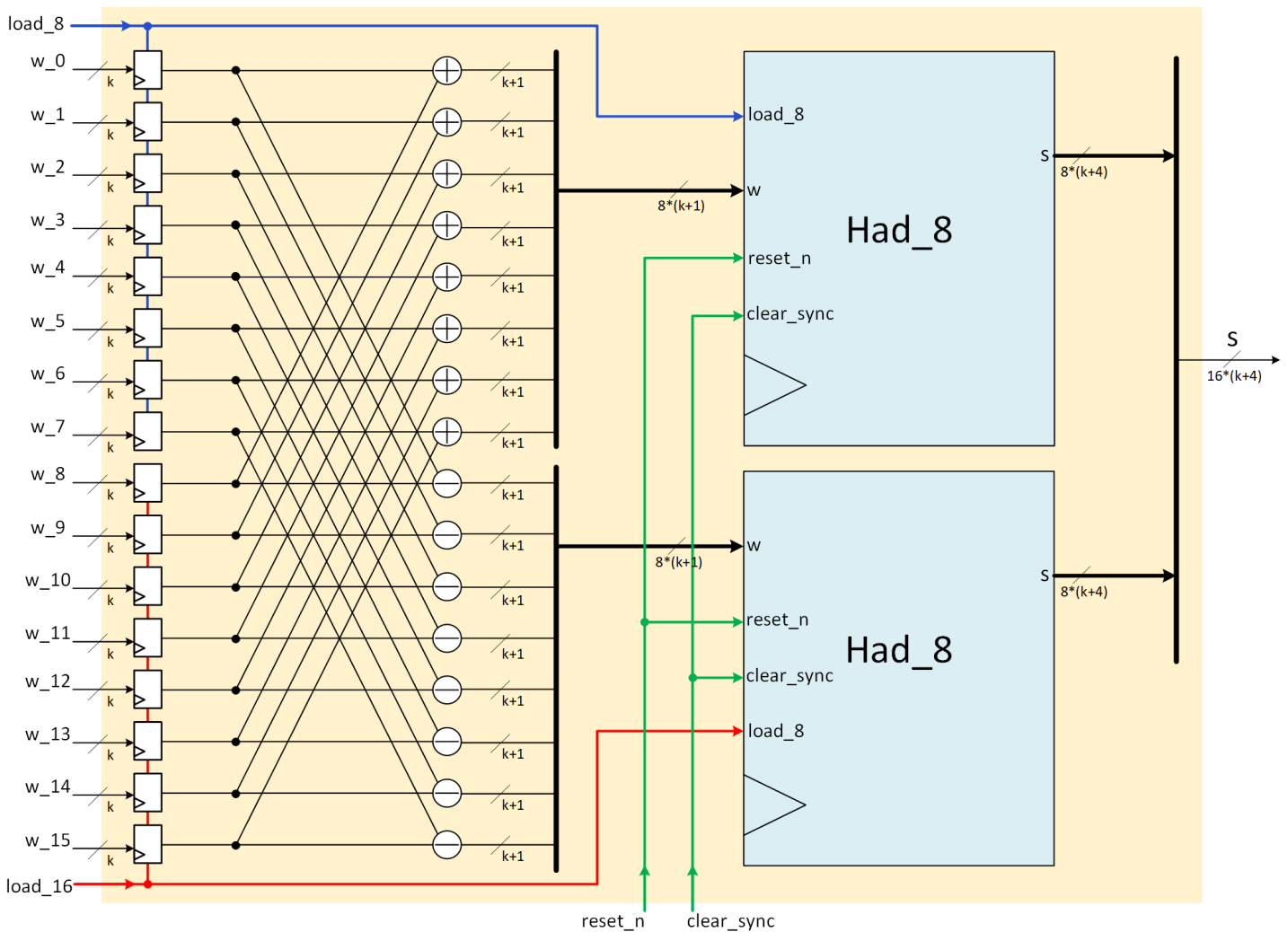


Fig. 31: Diagrama esquemático del bloque de cálculo de Transformada de Hadamard en una dimensión para bloques de 16x16 píxeles.

TABLA V

SEÑALES PARA EL BLOQUE DE CÁLCULO DE TRANSFORMADA DE HADAMARD EN UNA DIMENSIÓN PARA BLOQUES DE 16x16 PÍXELES

Señales			
Nombre	Tipo	# Bits	Descripción
clock	Entrada	1	Señal de clock
reset_n	Entrada	1	Señal de reset asíncrona
clear_sync	Entrada	1	Señal de clear síncrona
load_8	Entrada	1	Señal de carga síncrona para el bloque hadamard 8x8
load_16	Entrada	1	Señal de carga síncrona para el bloque hadamard 16x16
w	Entrada	16*k	Filas/Columnas de 16 coeficientes parciales
s	Salida	16*(k + 4)	Filas/Columnas de 16 coeficientes parciales

3.7.3.3 Transformada de Hadamard en una dimensión para bloques de 32x32 píxeles

Este bloque reutiliza el módulo de la sección 3.7.3.2 y añade una etapa más de procesamiento, tal como se observa en la Fig. 32. Las señales de entrada y salida a considerar para este bloque se encuentran detalladas en la Tabla VI. Nótese que la cantidad de bits para la entrada w y la salida s está representada nuevamente por una variable k, lo que significa que el bloque en cuestión es parametrizable y será reutilizado en la arquitectura final.

TABLA VI

SEÑALES PARA EL BLOQUE DE CÁLCULO DE TRANSFORMADA DE HADAMARD EN UNA DIMENSIÓN PARA BLOQUES DE 32x32 PÍXELES

Señales			
Nombre	Tipo	# Bits	Descripción
clock	Entrada	1	Señal de clock
reset_n	Entrada	1	Señal de reset asíncrona
clear_sync	Entrada	1	Señal de clear síncrona
load_8	Entrada	1	Señal de carga síncrona para el bloque hadamard 8x8
load_16	Entrada	1	Señal de carga síncrona para el bloque hadamard 16x16
load_32	Entrada	1	Señal de carga síncrona para el bloque hadamard 32x32
w	Entrada	32*k	Filas/Columnas de 32 coeficientes parciales
s	Salida	32*(k + 5)	Filas/Columnas de 32 coeficientes parciales

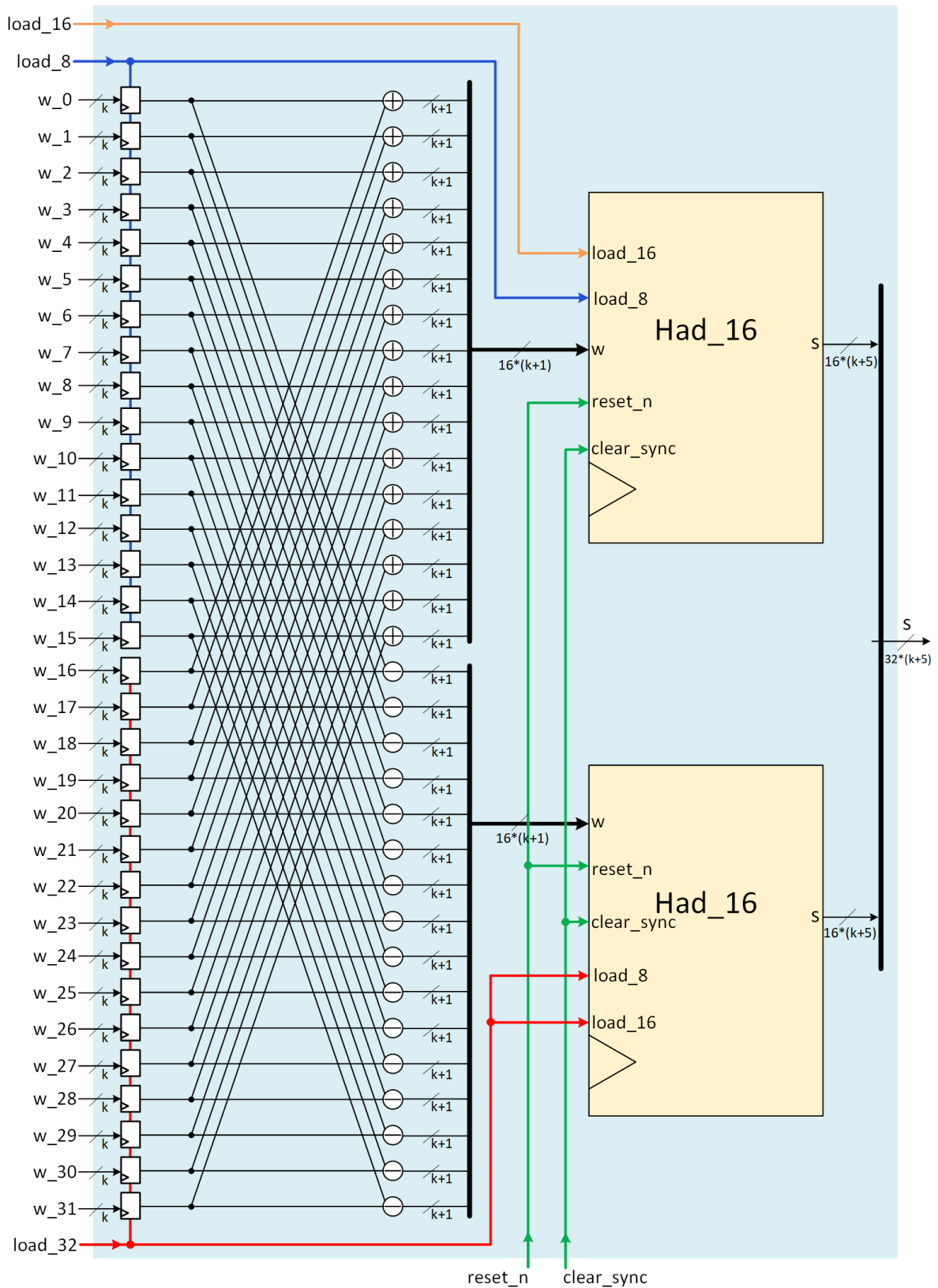


Fig. 32: Diagrama esquemático del bloque de cálculo de Transformada de Hadamard en una dimensión para bloques de 32x32 píxeles.

3.7.3.4 Transformada de Hadamard en una dimensión para bloques de 64x64 píxeles

Finalmente se tiene el diseño de la Fig. 33 que será utilizado para calcular la Transformada de Hadamard para filas y columnas, ambos procesos explicados en detalle más adelante en las secciones 3.7.4 y 3.7.6. Las señales de entrada y salida a considerar para este bloque se encuentran detalladas en la Tabla VII. Nótese que la cantidad de bits para la entrada w y la salida s está representada por una variable k , lo que significa que el bloque en cuestión es parametrizable y será reutilizado en la arquitectura final.

TABLA VII

SEÑALES PARA EL BLOQUE DE CÁLCULO DE TRANSFORMADA DE HADAMARD EN UNA DIMENSIÓN PARA BLOQUES DE 64x64 PÍXELES

Señales			
Nombre	Tipo	# Bits	Descripción
clock	Entrada	1	Señal de clock
reset_n	Entrada	1	Señal de reset asíncrona
clear_sync	Entrada	1	Señal de clear síncrona
load_8	Entrada	1	Señal de carga síncrona para el bloque hadamard 8x8
load_16	Entrada	1	Señal de carga síncrona para el bloque hadamard 16x16
load_32	Entrada	1	Señal de carga síncrona para el bloque hadamard 32x32
load_64	Entrada	1	Señal de carga síncrona para el bloque hadamard 64x64
w	Entrada	$64*k$	Filas/Columnas de 64 coeficientes parciales
s	Salida	$64*(k + 6)$	Filas/Columnas de 64 coeficientes parciales

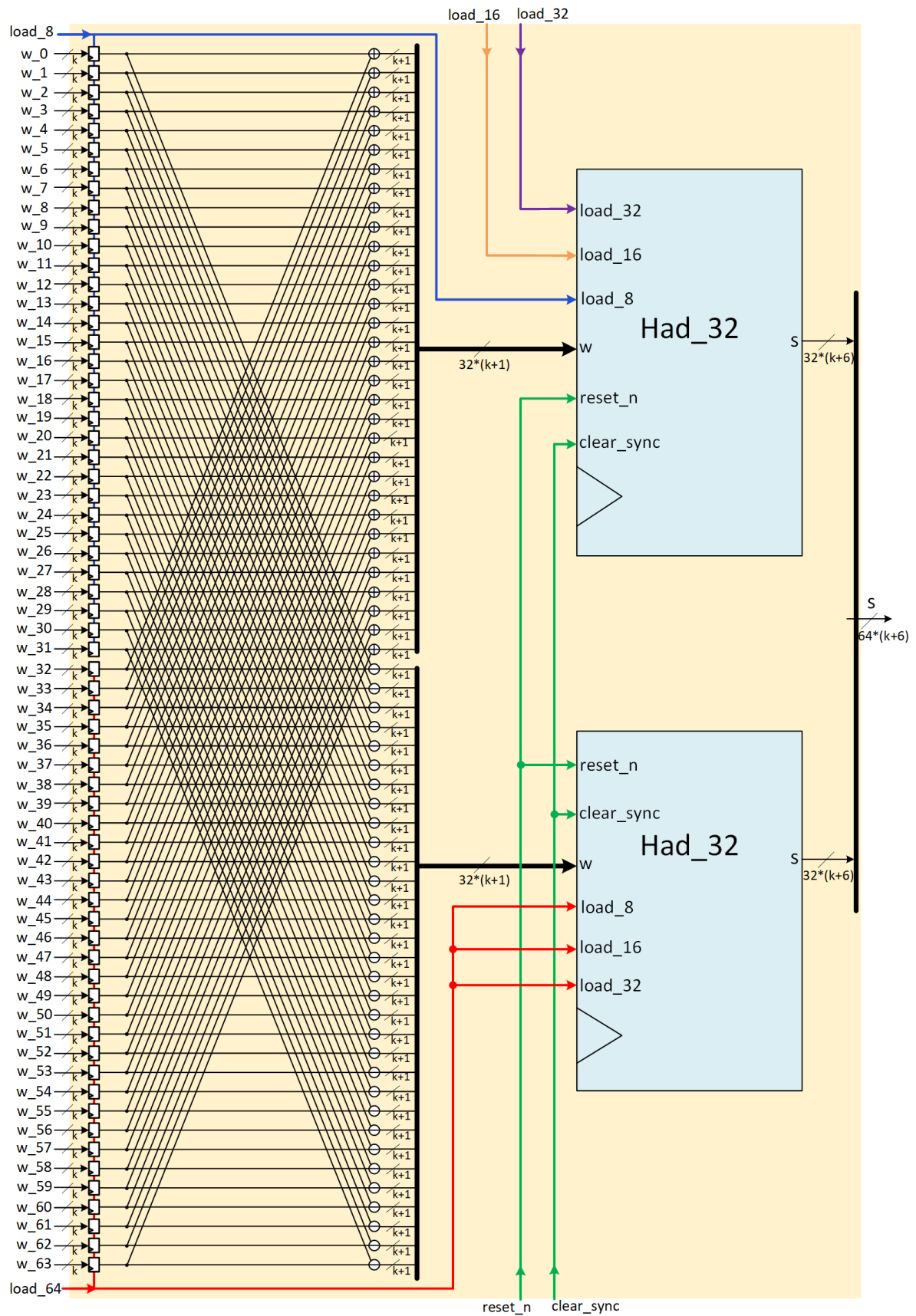


Fig. 33: Diagrama esquemático del bloque de cálculo de Transformada de Hadamard en una dimensión para bloques de 64x64 píxeles.

3.7.4 Bloque de cálculo de Transformada de Hadamard para filas

La arquitectura de este bloque está basado en el módulo de la sección 3.7.3.4. Durante esta etapa, el objetivo es aplicar la Transformada de Hadamard a cada una de las filas de distorsión obtenidas anteriormente. Como resultado de esta operación, se obtendrán filas de como máximo 64 coeficientes y que pueden ser valores tanto positivos como negativos que abarcan un ancho de 15 bits cada uno. Por ende, la salida del presente bloque tendrá un ancho de 960 bits. Las señales de entrada y salida a considerar para el bloque de cálculo de Transformada de Hadamard para filas se encuentran detalladas en la Tabla VIII.

TABLA VIII
SEÑALES PARA EL BLOQUE DE CÁLCULO DE TRANSFORMADA DE HADAMARD
PARA FILAS

Señales			
Nombre	Tipo	# Bits	Descripción
clock	Entrada	1	Señal de clock
reset_n	Entrada	1	Señal de reset asíncrona
clear_sync	Entrada	1	Señal de clear síncrona
load_8	Entrada	1	Señal de carga síncrona para el bloque hadamard 8x8
load_16	Entrada	1	Señal de carga síncrona para el bloque hadamard 16x16
load_32	Entrada	1	Señal de carga síncrona para el bloque hadamard 32x32
load_64	Entrada	1	Señal de carga síncrona para el bloque hadamard 64x64
w	Entrada	576	Distorsión para 64 píxeles
s	Salida	960	Filas de 64 coeficientes parciales

3.7.5 Buffer de Transposición

Conforme se obtengan las filas de coeficientes parciales, estas deberán ir almacenándose en el Buffer de Transposición. Para ello, el modo de operación de la matriz será el de escritura, por ende, la señal *write_read* estará seteada a '0' lógico. Luego de almacenarse por completo las 64 filas de 64 coeficientes parciales en el Buffer de Transposición, se procederá a cambiar el modo de operación a lectura seteando la señal *write_read* a '1' lógico. En resumen, el Buffer de Transposición almacenará filas y entregará columnas de coeficientes en cada ciclo de reloj. El diseño del presente bloque se basó en el diagrama de la Fig. 34, basado en otros trabajos como [25], [26]. *Zerro_Array* hace referencia a un arreglo de ceros de la forma "000" cuya longitud estará determinada por la cantidad de bits que pueda almacenar cada Flip Flop. Para este caso tenemos 64 coeficientes que abarcan un total de 960 bits y que serán almacenados en 64 Flip

Flops, por ende la longitud del Zero_Array será de 15 bits. Las señales de entrada y salida a considerar para el Buffer de Transposición se encuentran detalladas en la Tabla IX.

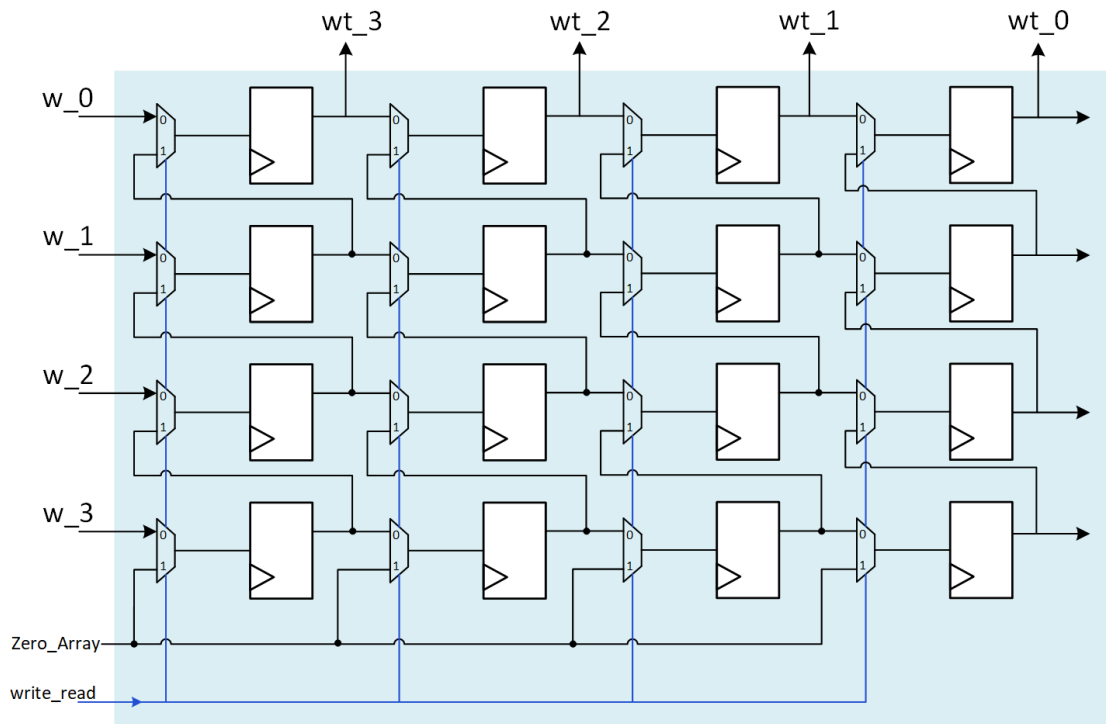


Fig. 34: Diagrama esquemático base del Buffer de Transposición.

TABLA IX

SEÑALES PARA EL BUFFER DE TRANSPOSICIÓN

Señales			
Nombre	Tipo	# Bits	Descripción
clock	Entrada	1	Señal de clock
reset_n	Entrada	1	Señal de reset asíncrona
clear_sync	Entrada	1	Señal de clear síncrona
load	Entrada	1	Señal de carga síncrona
write_read	Entrada	1	Señal que determina el modo de operación del buffer
w	Entrada	960	Filas de 64 coeficientes parciales
wt	Salida	960	Columnas de 64 coeficientes parciales

3.7.6 Bloque de cálculo de Transformada de Hadamard para columnas

La arquitectura de este bloque está basado en el módulo de la sección 3.7.3.4. A medida que el Buffer de Transposición entregue cada una de las columnas de coeficientes parciales almacenados, estas serán procesadas una a una por el presente bloque. Como resultado de esta operación, se obtendrán columnas de como máximo 64 coeficientes finales y que pueden ser valores tanto positivos como negativos que abarcan un ancho de 21 bits cada uno. Por lo tanto, la salida del presente bloque tendrá un ancho de 1344 bits. Las señales de entrada y salida a considerar para el bloque de cálculo de Transformada de Hadamard para columnas se encuentran detalladas en la Tabla X.

TABLA X
SEÑALES PARA EL BLOQUE DE CÁLCULO DE TRANSFORMADA DE HADAMARD
PARA COLUMNAS

Señales			
Nombre	Tipo	# Bits	Descripción
clock	Entrada	1	Señal de clock
reset_n	Entrada	1	Señal de reset asíncrona
clear_sync	Entrada	1	Señal de clear síncrona
load_8	Entrada	1	Señal de carga síncrona para el bloque hadamard 8x8
load_16	Entrada	1	Señal de carga síncrona para el bloque hadamard 16x16
load_32	Entrada	1	Señal de carga síncrona para el bloque hadamard 32x32
load_64	Entrada	1	Señal de carga síncrona para el bloque hadamard 64x64
w	Entrada	960	Columnas de 64 coeficientes parciales
s	Salida	1344	Columnas de 64 coeficientes finales

3.7.7 Bloque de Suma de Valores Absolutos

El presente bloque tiene como función sumar progresivamente los valores absolutos de cada uno de los coeficientes finales de las columnas obtenidas en la etapa anterior. A medida que se reciban y procesen las columnas, el valor de las sumas parciales se irá almacenando en un acumulador de 32 bits. El diseño del bloque de Suma de Valores Absolutos se basó en el diagrama de la Fig. 35. Las señales de entrada y salida a considerar se encuentran detalladas en la Tabla XI.

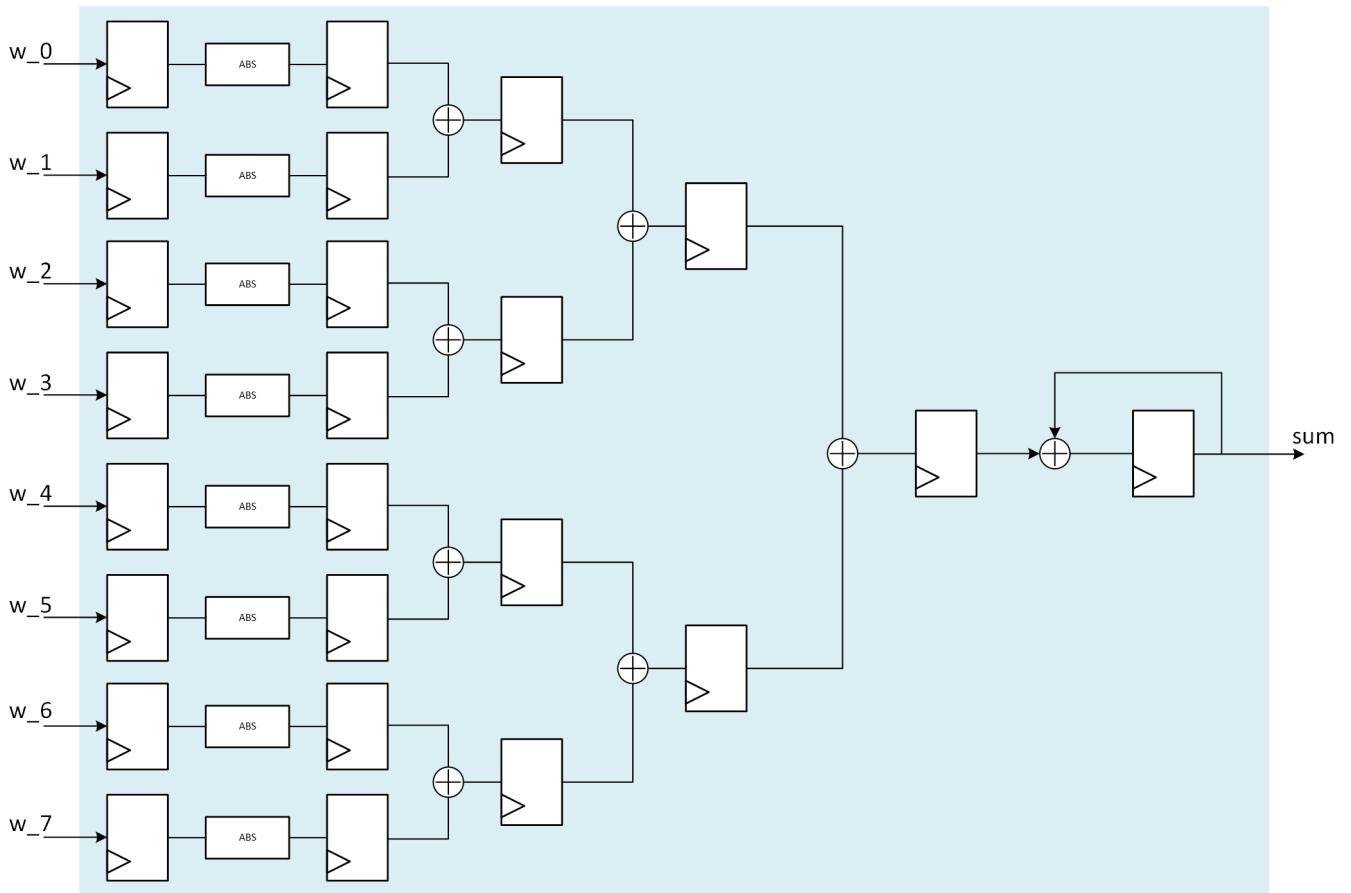


Fig. 35: Diagrama esquemático base para el bloque de Suma de Valores Absolutos.

TABLA XI

SEÑALES PARA EL BLOQUE DE SUMA DE VALORES ABSOLUTOS

Señales			
Nombre	Tipo	# Bits	Descripción
clock	Entrada	1	Señal de clock
reset.n	Entrada	1	Señal de reset asíncrona
clear_sync	Entrada	1	Señal de clear síncrona
load	Entrada	1	Señal de carga síncrona
w	Entrada	1344	Columnas de 64 coeficientes finales
sum	Salida	32	Suma acumulada

3.7.8 Bloque de Escalamiento

Finalmente, para poder obtener el costo SATD final se necesita escalar el valor de la suma acumulado en la etapa anterior. El factor de escalamiento dependerá de la dimensión de los bloques procesados. La Tabla XII muestran los factores considerados para el diseño de la Fig. 36. Las señales de entrada y salida a considerar para el bloque de Escalamiento se encuentran detalladas en la Tabla XIII.

TABLA XII
DIMENSIÓN DEL BLOQUE VS. FACTOR DE ESCALAMIENTO

Dimensión del Bloque	Factor de Escalamiento
8 x 8	$1/2^6$
16 x 16	$1/2^8$
32 x 32	$1/2^{10}$
64 x 64	$1/2^{12}$

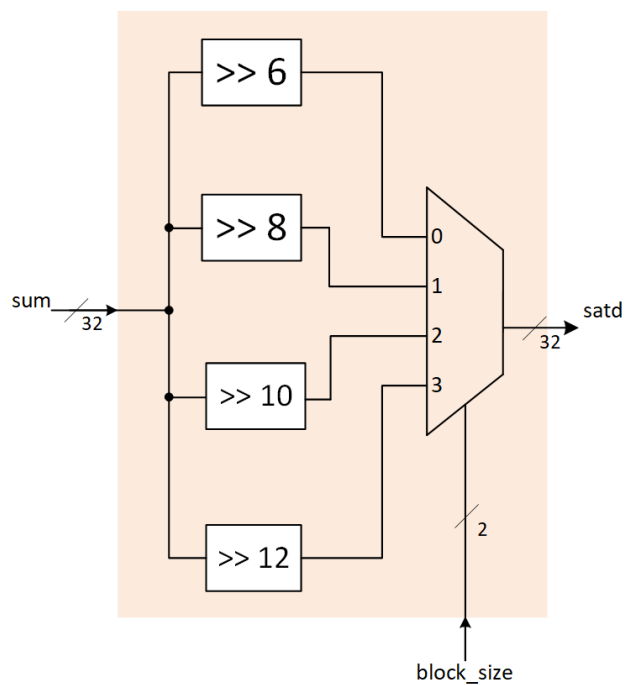


Fig. 36: Diagrama esquemático del bloque de Escalamiento.

TABLA XIII

SEÑALES PARA EL BLOQUE DE ESCALAMIENTO

Señales			
Nombre	Tipo	# Bits	Descripción
clock	Entrada	1	Señal de clock
reset_n	Entrada	1	Señal de reset asíncrona
clear_sync	Entrada	1	Señal de clear síncrona
load	Entrada	1	Señal de carga síncrona
block_size	Entrada	2	Dimensión del bloque
sum	Entrada	32	Suma acumulada
satd	Salida	32	Costo SATD

3.7.9 Contador de carrera libre

Con el objetivo de generar señales de control para la Máquina de Estados, se consideró pertinente usar 2 contadores de carrera libre de 5 y 7 bits cada uno. La señal Zero_Array hace referencia a un arreglo de ceros de longitud “n” bits. La Fig. 37 ilustra el diagrama esquemático para un contador genérico de “n” bits.

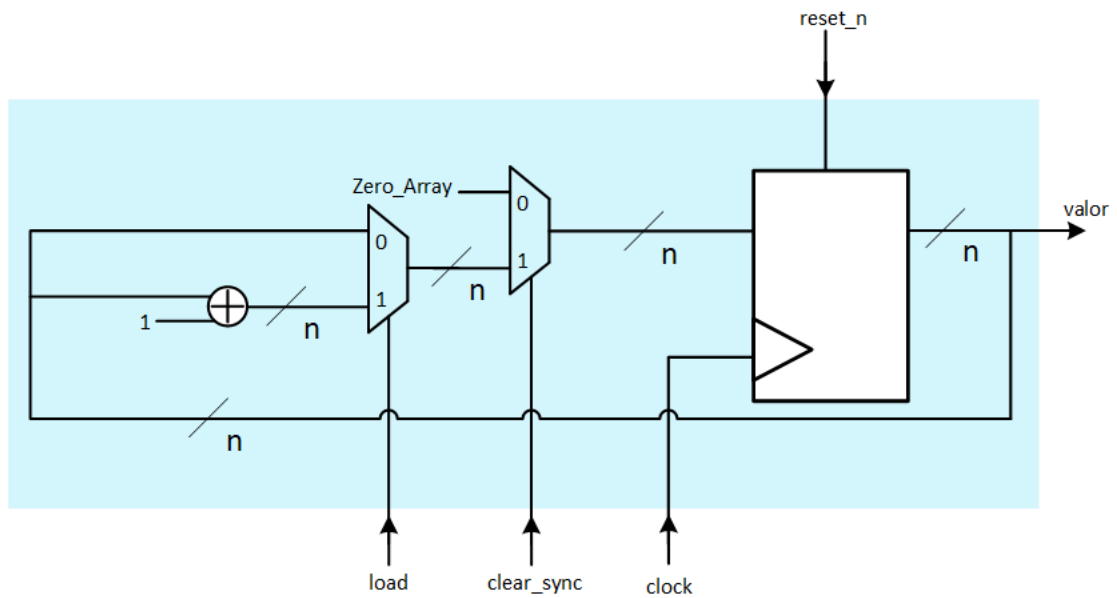


Fig. 37: Diagrama esquemático base para los contadores de carrera libre.

3.7.9.1 Contador de carrera libre de 5 bits

Contador cuyo diagrama esquemático está indicado en la Fig. 37 y cuyas señales de entrada y salida se encuentran detalladas en la Tabla XIV.

TABLA XIV
SEÑALES PARA EL CONTADOR DE CARRERA LIBRE DE 5 BITS

Señales			
Nombre	Tipo	# Bits	Descripción
clock	Entrada	1	Señal de clock
reset_n	Entrada	1	Señal de reset asíncrona
clear_sync	Entrada	1	Señal de clear síncrona
load	Entrada	1	Señal de carga síncrona
valor	Salida	5	Valor de la cuenta

3.7.9.2 Contador de carrera libre de 7 bits

Contador cuyo diagrama esquemático está indicado en la Fig. 37 y cuyas señales de entrada y salida se encuentran detalladas en la Tabla XV.

TABLA XV
SEÑALES PARA EL CONTADOR DE CARRERA LIBRE DE 7 BITS

Señales			
Nombre	Tipo	# Bits	Descripción
clock	Entrada	1	Señal de clock
reset_n	Entrada	1	Señal de reset asíncrona
clear_sync	Entrada	1	Señal de clear síncrona
load	Entrada	1	Señal de carga síncrona
valor	Salida	7	Valor de la cuenta

3.7.10 Arreglo de Comparadores

Conjunto de comparadores cuyas salidas permitirán controlar la Máquina de Estados. Las señales de entrada y salida a considerar para el arreglo de comparadores se encuentran detalladas en la Tabla XVI.

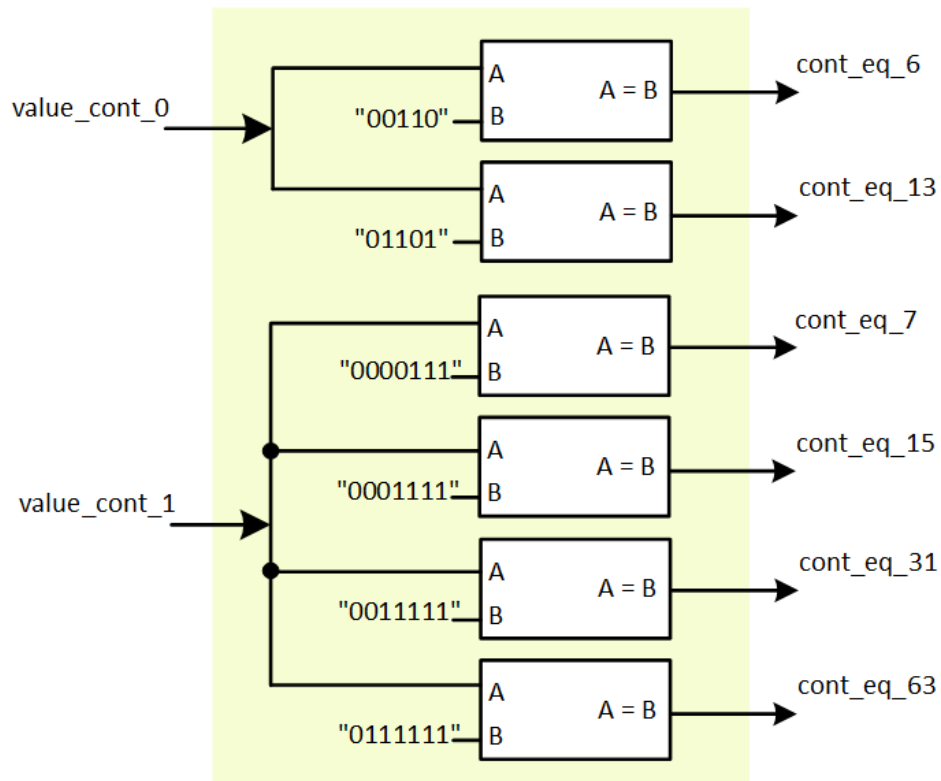


Fig. 38: Diagrama esquemático para los arreglos de comparadores.

TABLA XVI

SEÑALES PARA LOS ARREGLOS DE COMPARADORES

Señales			
Nombre	Tipo	# Bits	Descripción
value_cont_0	Entrada	5	Valor actual de la cuenta del contador de 5 bits
value_cont_1	Entrada	7	Valor actual de la cuenta del contador de 7 bits
cont_eq_6	Salida	1	Señal que indica si value_cont_0 es igual a 6
cont_eq_13	Salida	1	Señal que indica si value_cont_0 es igual a 13
cont_eq_7	Salida	1	Señal que indica si value_cont_1 es igual a 7
cont_eq_15	Salida	1	Señal que indica si value_cont_1 es igual a 15
cont_eq_31	Salida	1	Señal que indica si value_cont_1 es igual a 31
cont_eq_63	Salida	1	Señal que indica si value_cont_1 es igual a 63

3.7.11 Máquina de Estados

El flujo de operación del cálculo del costo SATD explicado en la sección 3.7.1 es automatizado por medio de la presente Máquina de Estados tipo *Moore*. Con el objetivo de que las salidas de esta FSM (*Finite State Machine*) estén libre de errores o fallas (en inglés *glitches*), se procedió a registrar dichas salidas. Para compensar el retardo de 1 ciclo de reloj, se optó por implementar una FSM *Moore* de salida adelantada (del inglés *Look-ahead output circuit for Moore output*) [27]. Por otro lado, como uno de los objetivos del presente trabajo es lograr una transmisión de cuadros de video a tiempo real, requerimos que el circuito sea lo más rápido posible. Por esa razón, para el diseño de la FSM se usó el método de codificación One-Hot (del inglés *One-Hot Encoding*) que demuestra ser más rápido que la codificación Binaria (del inglés *Binary Encoding*) y la codificación Gray (del inglés *Gray Encoding*) a cambio del uso de más registros [28]. El diagrama esquemático de la FSM está detallado en la Fig. 39 y las señales de entrada y salida en la Tabla XVIII. Con el objetivo de facilitar la comprensión del diagrama de estados, la señales provenientes de los comparadores se renombraron por una nomenclatura más simple. La Tabla XVII indica la correlación entre los nombres de las señales en el diagrama y las usadas en el diseño final.

TABLA XVII
CORRELACIÓN ENTRE LAS SEÑALES DEL DIAGRAMA FSM Y LAS USADAS EN EL
DISEÑO FINAL

Señal en Diagrama FSM	Señal usada en diseño
N	block_size
W6	cont_eq_6
W7	cont_eq_7
W13	cont_eq_13
W15	cont_eq_15
W31	cont_eq_31
W63	cont_eq_63

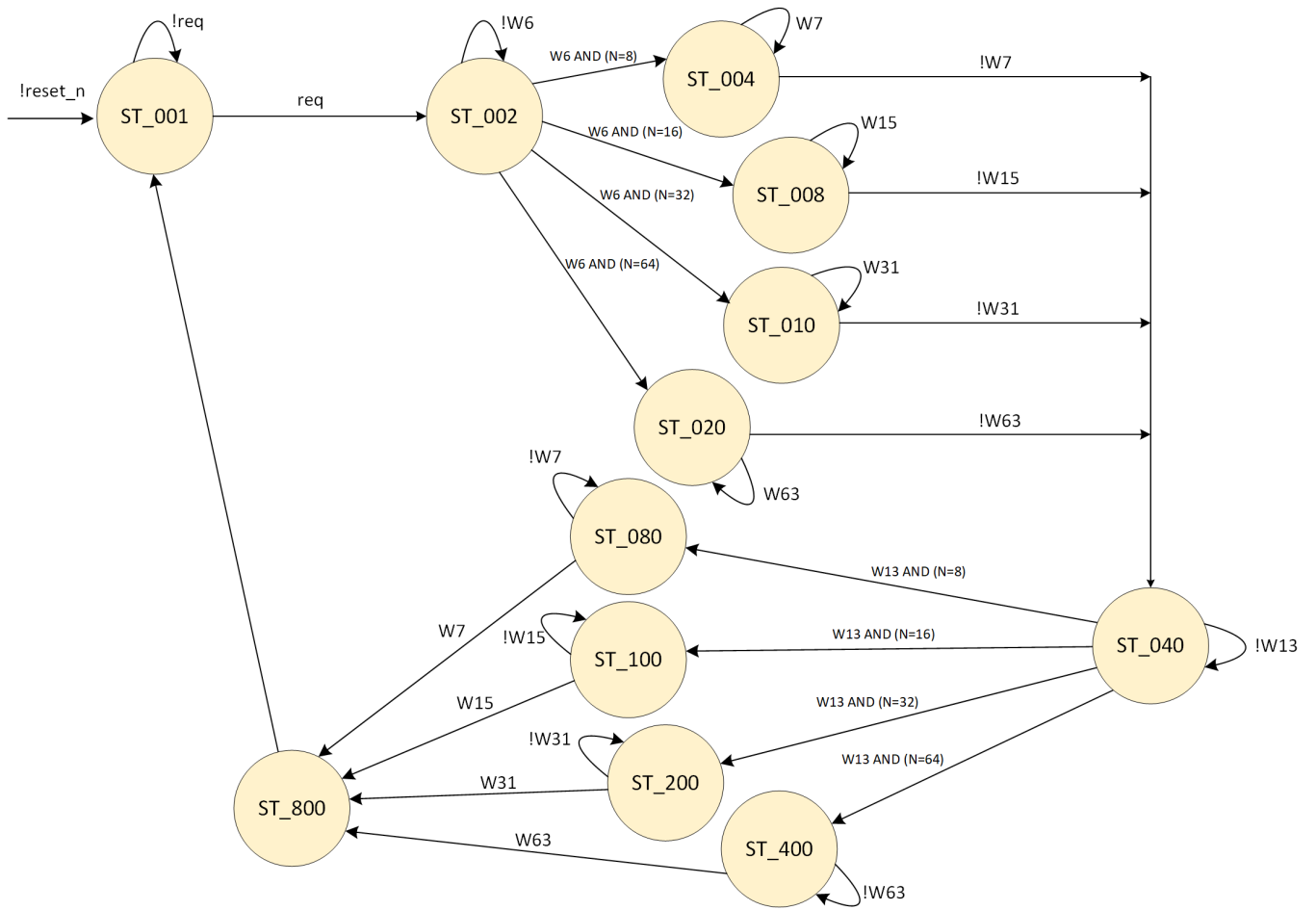


Fig. 39: Diagrama esquemático para la Máquina de Estados Moore.

TABLA XVIII

SEÑALES PARA LA MÁQUINA DE ESTADOS MOORE

Señales			
Nombre	Tipo	# Bits	Descripción
clock	Entrada	1	Señal de clock
reset_t	Entrada	1	Señal de reset asíncrona
req	Entrada	1	Señal de request
block_size	Entrada	2	Indica la dimensión del bloque a procesar
cont_eq_6	Entrada	1	Señal que indica si la cuenta es igual a 6
cont_eq_13	Entrada	1	Señal que indica si la cuenta es igual a 13
cont_eq_7	Entrada	1	Señal que indica si la cuenta es igual a 7
cont_eq_15	Entrada	1	Señal que indica si la cuenta es igual a 15
cont_eq_31	Entrada	1	Señal que indica si la cuenta es igual a 31
cont_eq_63	Entrada	1	Señal que indica si la cuenta es igual a 63
load_distorsion	Salida	1	Señal de carga para el bloque de Distorsión
clear_sync_distorsion	Salida	1	Señal de clear síncrona para el bloque de Distorsión
load_had_h	Salida	1	Señal de carga para el bloque de Hadamard aplicado a filas
clear_sync_had_h	Salida	1	Señal de clear síncrona para el bloque de Hadamard aplicado a filas
load_bt	Salida	1	Señal de carga para el Búfer Transpuesto
clear_sync_bt	Salida	1	Señal de clear síncrona para el Búfer Transpuesto
write_read	Salida	1	Modo de operación del Búfer Transpuesto
load_had_v	Salida	1	Señal de carga para el bloque de Hadamard aplicado a columnas
clear_sync_had_v	Salida	1	Señal de clear síncrona para el bloque de Hadamard aplicado a columnas
load_sav	Salida	1	Señal de carga para el bloque de Suma de Valores Absolutos
clear_sync_sav	Salida	1	Señal de clear síncrona para el bloque de Suma de Valores Absolutos
load_cont_1	Salida	1	Señal de carga para el contador de carrera libre de 5 bits
clear_sync_cont_1	Salida	1	Señal de clear síncrona para el contador de carrera libre de 5 bits
load_cont_2	Salida	1	Señal de carga para el contador de carrera libre de 7 bits
clear_sync_cont_2	Salida	1	Señal de clear síncrona para el contador de carrera libre de 7 bits
ack	Salida	1	Señal que indica la finalización del cálculo del costo SATD

3.7.12 Elección del mejor candidato

Este bloque permitirá elegir al candidato cuyo costo SATD calculado previamente sea el menor de toda la lista de candidatos. Dado que la máxima cantidad de candidatos en la lista es 5, se está considerando la entrada `index` y salida `merge_index` de 3 bits (índices desde 0 hasta 4 en la lista de candidatos). Las señales de entrada y salida a considerar para el diagrama esquemático de la Fig. 40 se encuentran detalladas en la Tabla XIX.

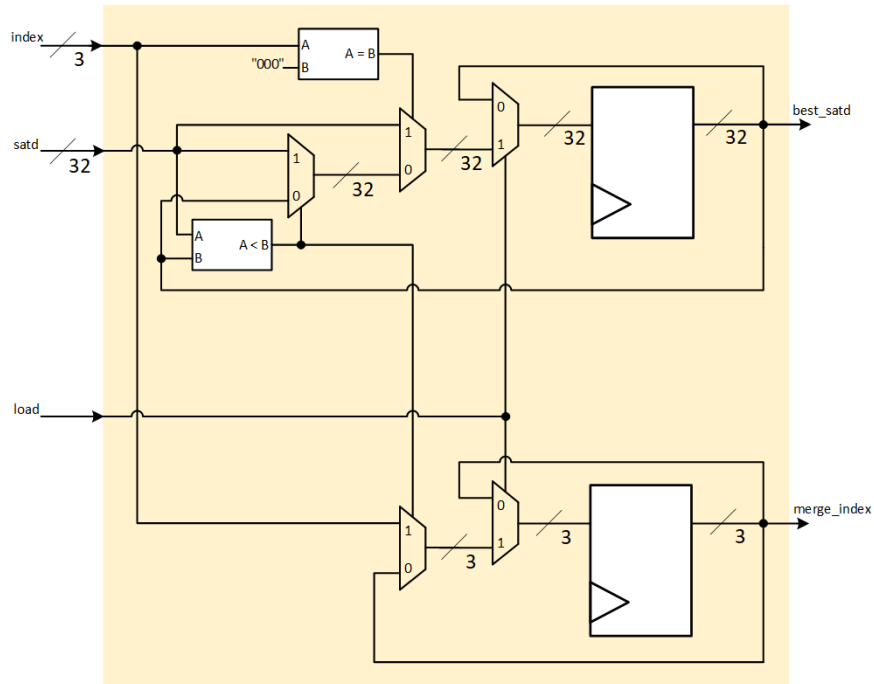


Fig. 40: Diagrama esquemático para el bloque de elección del mejor candidato.

TABLA XIX

SEÑALES PARA EL BLOQUE DE ELECCIÓN DEL MEJOR CANDIDATO

Señales			
Nombre	Tipo	# Bits	Descripción
satd	Entrada	32	Costo SATD del candidato
index	Entrada	3	Índice del candidato en la lista
clock	Entrada	1	Señal de clock
reset_n	Entrada	1	Señal de reset asíncrona
clear_sync	Entrada	1	Señal de clear síncrona
load	Entrada	1	Señal de carga síncrona
best_satd	Salida	32	Menor costo SATD
merge_index	Salida	3	Índice del candidato en la lista con menor costo SATD

Capítulo 4

Verificación y análisis de resultados

En el presente capítulo, se mostrarán los resultados obtenidos a partir del diseño en hardware del algoritmo de *Block Merging* explicado en la sección 2.5. Con el objetivo de comparar los resultados y verificar el correcto funcionamiento de la etapa de elección del mejor candidato del algoritmo de *Block Merging*, se usó el Software de Referencia programado en MATLAB. Además, se crearon algoritmos para recrear el proceso de Particionamiento de Árbol Cuádruple y Estimación de Movimiento Entera también en MATLAB, de manera que sirvieron como fuente de estímulos para la verificación de la arquitectura propuesta.

4.1 Almacenamiento de Bloques de Predicción y Vectores de Movimiento Enteros

En primer lugar, se aplicó el particionamiento variable a cada cuadro de una secuencia de video, tal como se puede observar en la Fig. 41. Toda la información de los bloques de píxeles subdivididos, posición relativa dentro del cuadro de video y dimensión del bloque fue almacenada en estructuras en MATLAB, también explicadas en la sección 3.5.3. La Fig. 42 ejemplifica la información del particionamiento almacenada en estructuras.

Por otro lado, también se almacenaron los vectores de movimiento obtenidos luego de haber aplicado el algoritmo TSS explicado en la sección 3.5.2, Estos vectores de movimiento presentan valores dentro del rango de -7 hasta +7 para ambas componentes X e Y, tal como se indica en la Fig. 43.



Fig. 41: Particionamiento del cuadro 167 de la secuencia de video *Foreman*.




Fields	 dim	 nodo_row	 nodo_col
1	16	81	17
2	16	97	17
3	16	113	17
4	16	97	33
5	16	113	33
6	16	33	49
7	16	81	49
8	16	113	49
9	16	1	65
10	16	17	65
11	16	33	65
12	16	65	65
13	16	81	65
14	16	97	65
15	16	113	65
16	16	17	81
17	16	33	81
18	16	65	81
19	16	81	81
20	16	97	81
21	16	33	97
22	16	65	97

Fig. 42: Bloques de Predicción obtenidos a partir del particionamiento del cuadro 167 de la secuencia de video *Foreman*.

Fields	dim	nodo_row	nodo_col	me_row	me_col
1	16	81	17	0	0
2	16	97	17	0	0
3	16	113	17	0	0
4	16	97	33	5	-5
5	16	113	33	0	0
6	16	33	49	1	7
7	16	81	49	0	0
8	16	113	49	3	-7
9	16	1	65	0	0
10	16	17	65	-6	5
11	16	33	65	-1	5
12	16	65	65	1	-1
13	16	81	65	1	-2
14	16	97	65	0	-1
15	16	113	65	0	-1
16	16	17	81	-6	-4
17	16	33	81	0	2
18	16	65	81	1	-1
19	16	81	81	0	-1
20	16	97	81	1	-1

Fig. 43: Estructura de los vectores de Movimiento obtenidos para el cuadro 46 hallados con respecto al cuadro 45 de la secuencia de video *Foreman*.

4.2 Simulación del Bloque de Cálculo del costo SATD

En esta sección se mostrarán los resultados obtenidos en el Testbench usando la herramienta Vivado Design Suite de Xilinx. Además, para cada una de las etapas explicadas en el capítulo 3, se verificará que los resultados parciales y finales obtenidos en MATLAB coinciden con lo obtenido en el Testbench.

Como punto de partida, se trabajará con el cuadro de video 80 de la secuencia de video akiyo_qcif.yuv. Las dimensiones de este cuadro son de 176x144 píxeles. Además, se tienen un conjunto de bloques de 8x8 píxeles que comprenden el bloque actual a analizar (bloque de referencia) y los bloques candidatos (obtenidos por medio de la compensación de movimiento usando los vectores de movimiento en la lista de candidatos antes creada, para un total de 3 candidatos). Entonces, la arquitectura en hardware irá calculando el costo SATD entre el bloque de referencia y cada candidato de la lista. Los valores de costo SATD obtenidos usando MATLAB son mostrados en la Fig. 46.



Fig. 44: Bloque de 8x8 píxeles para el bloque actual de evaluación (bloque de referencia).

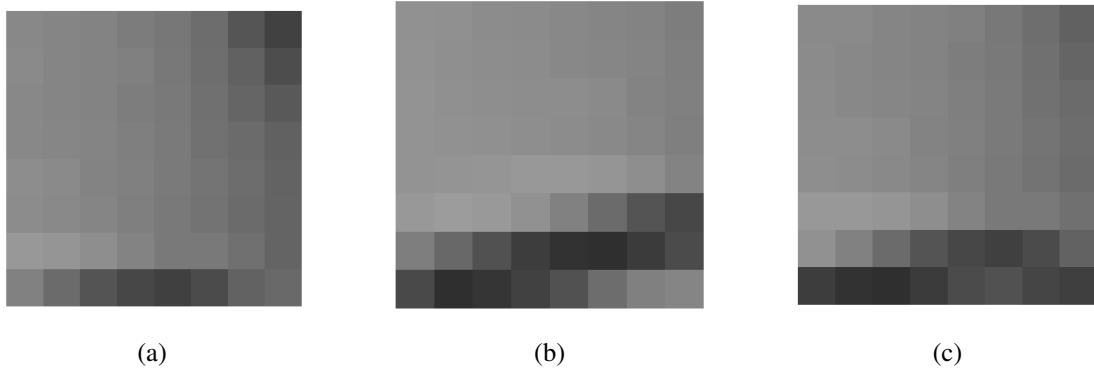


Fig. 45: Bloques de 8x8 píxeles para cada candidato en la lista. (a) Candidato de índice 0. (b) Candidato de índice 1. (c) Candidato de índice 2.

actual	candidato	index	costoSATD
8x8 uint8	8x8 uint8	0	78
8x8 uint8	8x8 uint8	1	111
8x8 uint8	8x8 uint8	2	38

Fig. 46: Valores de costo SATD calculados a partir de MATLAB para cada candidato.

La Fig. 47 muestra el cálculo de la distorsión por filas entre el bloque candidato (act_8) y el de referencia (ref_8). Respecto a los bloques, cada número representa un píxel que abarca 8 bits (valores desde 0 hasta 255).

-2	-4	-1	-6	-8	-9	-24	-27
-1	-5	-2	-3	-6	-9	-13	-21
-2	-2	-4	-6	-4	-8	-11	-16
-5	-6	-5	-4	-4	-7	-8	-13
-1	-2	-5	-3	-3	-4	-6	-8
-8	-11	-12	-16	-13	-11	-14	-9
-1	10	20	30	35	41	19	-7
50	47	30	15	1	11	38	37

(a)

0	0	-2	-1	-2	-5	-1	-8	-1	50
0	0	-4	-5	-2	-6	-2	-11	10	47
0	0	-1	-2	-4	-5	-12	20	30	30
0	0	-6	-3	-6	-4	-3	-16	30	15
0	0	-8	-6	-4	-3	-13	35	1	1
0	0	-9	-8	-7	-4	-11	41	11	11
0	0	-24	13	-11	-8	-6	-14	19	38
0	0	-27	21	-16	-13	-8	-9	-7	37

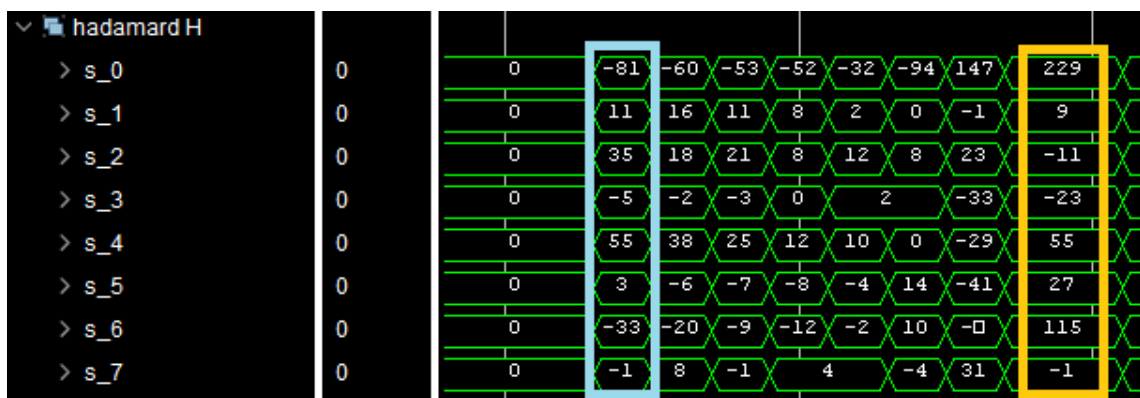
(b)

Fig. 47: Resultados obtenidos en la etapa de Distorsión. (a) Resultados obtenidos de MATLAB. (b) Resultados obtenidos del Testbench en Vivado.

Cada una de estas filas de distorsión obtenidas se irán procesando en la siguiente etapa, la cual calcula la Transformada Rápida de Hadamard en el eje de las filas. La Fig. 48 evidencia lo antes mencionado.

-81	11	35	-5	55	3	-33	-1
-60	16	18	-2	38	-6	-20	8
-53	11	21	-3	25	-7	-9	-1
-52	8	8	0	12	-8	-12	4
-32	2	12	2	10	-4	-2	4
-94	0	8	2	0	14	10	-4
147	-1	23	-33	-29	-41	-105	31
229	9	-11	-23	55	27	115	-1

(a)



(b)

Fig. 48: Resultados obtenidos en la etapa de Transformada de Hadamard para filas. (a) Resultados obtenidos de MATLAB. (b) Resultados obtenidos del Testbench en Vivado.

Las filas obtenidas de la etapa anterior se irán almacenando una a una en el Buffer de Transposición. Luego de almacenarse completamente estos 64 valores (se consideraron bloques de 8x8 píxeles), el buffer comenzará a entregar columnas una a una a la siguiente etapa, tal como se puede ver en la Fig. 49.

-81	11	35	-5	55	3	-33	-1
-60	16	18	-2	38	-6	-20	8
-53	11	21	-3	25	-7	-9	-1
-52	8	8	0	12	-8	-12	4
-32	2	12	2	10	-4	-2	4
-94	0	8	2	0	14	10	-4
147	-1	23	-33	-29	-41	-105	31
229	9	-11	-23	55	27	115	-1

(a)

(b)

Fig. 49: Resultados obtenidos luego de aplicar la transpuesta en el Buffer de Transposición. (a) Resultados obtenidos de MATLAB. (b) Resultados obtenidos del Testbench en Vivado.

Cada columna se seguirá procesando en la etapa de Transformada de Hadamard para columnas, cuyos resultados se pueden observar en la Fig. 48.

4	56	114	-62	166	-22	-56	40
-42	-10	68	-16	-44	-76	-242	26
-538	2	32	56	40	36	-34	-26
124	4	-26	10	98	58	192	-28
-496	36	50	42	94	-14	-92	-20
-2	6	-8	4	104	96	222	-54
466	14	16	-64	72	-12	-30	34
-164	-20	34	-10	-90	-42	-224	20

(a)

(b)

Fig. 50: Resultados obtenidos en la etapa de Transformada de Hadamard para columnas. (a) Resultados obtenidos de MATLAB. (b) Resultados obtenidos del Testbench en Vivado.

La etapa anterior genera progresivamente columnas de coeficientes que corresponden a una matriz final, resultado de aplicar la Transformada de Hadamard a la distorsión entre los bloques de referencia y candidato. Cada columna seguirá siendo procesada en el bloque de SAV (*Sum of Absolute Values*), cuya suma se irá almacenando en un acumulador, tal como se observa en la Fig. 51.

```

Command Window
>> sum(sum(abs(hadamard_v)))

ans =

    5000

fx >> |
  
```

(a)

> satd_sin_escalar	0	0	1836	1984	2332	2596	3304	3660	4752	5000
--------------------	---	---	------	------	------	------	------	------	------	------

(b)

Fig. 51: Resultados obtenidos en la etapa de SAV. (a) Resultados obtenidos de MATLAB. (b) Resultados obtenidos del Testbench en Vivado.

El costo SATD final se obtiene luego de escalar la suma acumulada en la etapa anterior. El factor de escalamiento depende de la dimensión de los bloques a evaluar, tal como lo indica la table XII. La Fig. 52 muestra el costo SATD final obtenido para un candidato.

```

Command Window
>> floor(sum(sum(abs(hadamard_v)))/8^2)

ans =

    78

fx >> |
  
```

(a)

> satd_sin_escalar	0	0	1836	1984	2332	2596	3304	3660	4752	5000
> costo_satd	0	0	28	31	36	40	51	57	74	78

(b)

Fig. 52: Resultados obtenidos en la etapa de Escalamiento. (a) Resultados obtenidos de MATLAB. (b) Resultados obtenidos del Testbench en Vivado.

Este proceso se repetirá para los 2 candidatos restantes, obteniendo así 3 valores de costo SATD que serán analizados más adelante para determinar el mejor candidato. Las Fig. 53 y 54 evidencian dichos costos obtenidos por la arquitectura, los cuales son iguales a los costos hallados por MATLAB en la Fig. 46.

> satd_sin_escalar	0	0	3034	3318	4172	4306	6000	6214	6730	7124
> costo_satd	0	0	47	51	65	67	93	97	105	111

Fig. 53: Costo SATD final para el segundo candidato.

> satd_sin_escalar	0	0	864	950	1032	1168	1688	1904	2296	2466
> costo_satd	0	0	13	14	16	18	26	29	35	38

Fig. 54: Costo SATD final para el tercer candidato.

4.3 Simulación del Bloque de Elección del mejor candidato

Cada costo SATD final obtenido anteriormente será analizado para obtener el menor valor posible. El índice del candidato con menor costo así como el valor del costo SATD serán las salidas del presente bloque, tal como indica la Fig. 55. A partir de los costos obtenidos en la Fig. 46, se verifica que el menor costo SATD es de 38, correspondiente a un candidato de índice 2.

SALIDAS					
> best_satd[31:0]	0			0	78
> merge_index[2:0]	0			0	2
ENTRADAS					
> satd[31:0]	0	0	78	0	111
> index[2:0]	0	0	1	0	2

Fig. 55: Resultados de la elección del mejor candidato.



(a)



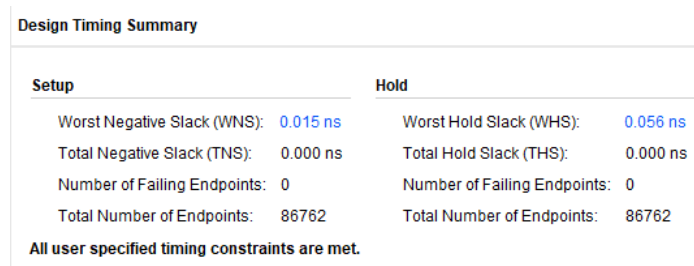
(b)

Fig. 56: (a) Bloque de referencia. (b) Mejor candidato (índice 2).

4.4 Síntesis de la arquitectura

Para entender el análisis de tiempos realizado por la herramienta Vivado, es necesario definir primero los conceptos de *Setup time* y *Hold requirement*. El *Setup time* es el tiempo antes de la ocurrencia de un flanco de reloj en el que una señal debe mantenerse disponible para poder ser capturada satisfactoriamente; mientras que el *Hold requirement* hace referencia al tiempo luego de la ocurrencia de un flanco de reloj durante el cual una señal debe mantenerse estable para evitar capturar valores no deseados [29]. Además, el término *Setup slack* se define como la diferencia entre el tiempo requerido (*Setup time*) y el tiempo real de llegada de una señal; a su vez, el término *Hold slack* se define como la diferencia entre el tiempo real de llegada de una señal y el tiempo requerido (*Hold requirement*). La transferencia de información es realizada correctamente cuando el *Setup* y *Hold slacks* son positivos. Por el contrario, cuando exista alguna violación en las restricciones para el *Setup time* o *Hold requirement*, el slack será negativo [29]. Por último, el WNS (*Worst Negative Slack*) y WHS (*Worst Hold Slack*) son valores positivos o negativos que corresponden al peor *slack* en el análisis del *Setup* y *Hold* respectivamente. La suma de todas las violaciones WNS es llamada TNS (*Total Negative Slack*), mientras que la suma de todas las violaciones WHS es denominada THS (*Total Hold Slack*) [30]. Cuando el TNS y THS son ambos iguales a cero, se dice que el diseño cumple con los requerimientos de tiempo establecidos (*timing constraints*) [30].

La presente arquitectura se sintetizó en el FPGA Kintex-7 de Xilinx usando la herramienta Vivado Design Suite. La mayor frecuencia de operación que permite tener WNS y WHS positivos es de 263.158 MHz (Periodo de 3.8ns). Las Fig. 57 y 58 verifican estos valores.



Design Timing Summary	
Setup	Hold
Worst Negative Slack (WNS): 0.015 ns	Worst Hold Slack (WHS): 0.056 ns
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 86762	Total Number of Endpoints: 86762

All user specified timing constraints are met.

Fig. 57: Resumen de tiempos de diseño para la arquitectura propuesta.



Name	Waveform	Period (ns)	Frequency (MHz)
clock	{0.000 1.900}	3.800	263.158

Fig. 58: Frecuencia de operación para la arquitectura propuesta.

Para calcular la cantidad de cuadros de video por segundo que la arquitectura diseñada es capaz de procesar, se hizo uso de la Tabla XX, la cual indica cuántos ciclos de reloj se emplean para obtener el mejor candidato para un solo bloque de referencia. Este valor dependerá tanto de la dimensión del bloque a procesar, así como de la cantidad de candidatos que se consideren en la lista.

TABLA XX
CICLOS DE RELOJ EMPLEADOS PARA ELEGIR AL MEJOR CANDIDATO

Dimensión del bloque de referencia	Número de candidatos				
	1	2	3	4	5
8 x 8	39	77	115	153	191
16 x 16	55	109	163	217	271
32 x 32	97	193	289	385	481
64 x 64	151	301	451	601	751

Se tiene también la Tabla XXI que permite tener una idea de la cantidad de bloques que puede contener un cuadro de video de resolución 4k (3840x2160 píxeles) considerando un particionamiento uniforme (cuadrícula de bloques de dimensión constante).

TABLA XXI
CANTIDAD DE BLOQUES EN UN CUADRO DE VIDEO 4K (PARTICIONAMIENTO UNIFORME)

Dimensión de bloques	Total de bloques
8 x 8	129,600
16 x 16	32,400
32 x 32	8,100
64 x 64	2,025

De esta forma, el caso más crítico de procesamiento sería en el empleo de un particionamiento uniforme en bloques de 8x8 píxeles, mientras que el caso con menor procesamiento sería el correspondiente de un particionamiento uniforme en bloques de 64x64 píxeles. Por otro lado, la Fig. 59 permite explicar el impacto que tiene la cantidad de candidatos de la lista como parte del algoritmo de *Block Merging*. En (a) se puede notar como la pérdida de *performance* aumenta conforme se disminuye la cantidad de candidatos. A partir de 2 candidatos ya se comienza a percibir una pérdida moderada, mientras que para 1 candidato el *performance* ya disminuyó significativamente. En (b) la complejidad disminuye aproximadamente de forma

lineal respecto a la cantidad de candidatos. Sin embargo, cuando se tienen 2 candidatos el tiempo de codificación se mantiene casi constante. Por lo tanto, se puede concluir que una lista con 3 candidatos para el *Block Merging* es el mejor enfoque para mantener un balance entre *performance* y complejidad en la codificación [2].

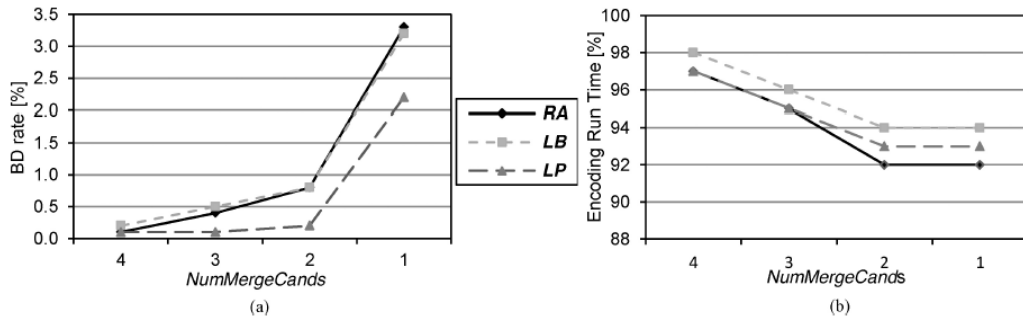


Fig. 59: Impacto de la cantidad de candidatos en la lista sobre la (a) *performance* y (b) complejidad [2].

Dado que la arquitectura diseñada fue pensada para abordar un particionamiento variable, los datos mostrados en la Tabla XXI deben ajustarse a tal contexto. Tomando como referencia la Fig. 60, se puede saber qué porcentaje de la totalidad del *frame* está compuesto por bloques de las dimensiones antes señaladas. Los valores de Parámetro de Cuantización QP empleados por otros trabajos ([18], [21], [31], [32] y [33]) para comparar resultados son usualmente 22, 24, 32 y 36, por lo que el presente trabajo también usará dichos valores. De esta forma, se obtiene la Tabla XXIII, la cual da a conocer el *performance* (cuadros por segundo) de la arquitectura propuesta para procesar secuencias de video en 4k (3840x2160) y 8k (7680x4320).

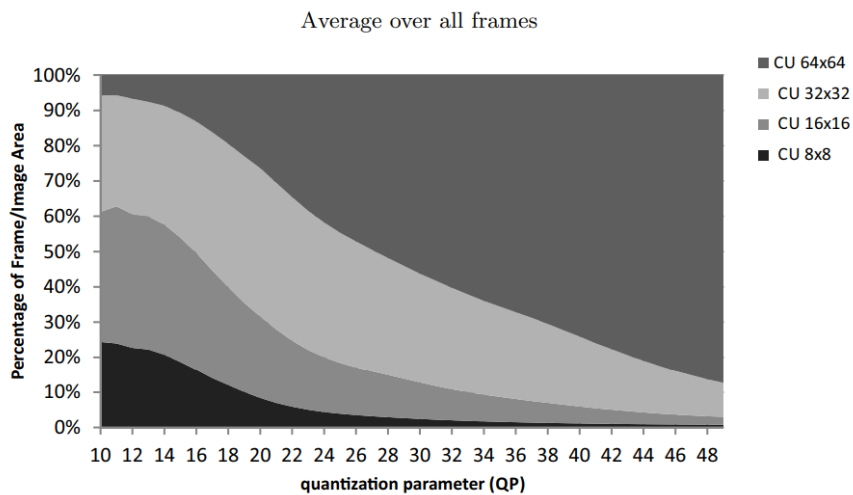


Fig. 60: Porcentaje de bloques en el cuadro vs. Parámetro de Cuantización QP [12].

TABLA XXII

PORCENTAJE DE BLOQUES EN UN CUADRO DE VIDEO

Dimensión del bloque	Parámetro de Cuantización (QP)			
	22	24	32	36
8 x 8	8%	4%	2%	1%
16 x 16	18%	17%	9%	9%
32 x 32	41%	37%	29%	25%
64 x 64	33%	42%	60%	65%

TABLA XXIII

PERFORMANCE EN CUADROS POR SEGUNDO DE LA ARQUITECTURA PROPUESTA

Resolución de cuadros de video	Parámetro de Cuantización (QP)			
	22	24	32	36
3840 x 2160 (4k)	77.3065	95.9145	131.5649	145.9392
7680 x 4320 (8k)	19.3266	23.9786	32.8912	36.4848

Por último, la Tabla XXIV muestra una comparación entre el diseño de la presente tesis y las arquitecturas de trabajos previos usados como referencia. Se puede notar que el presente trabajo supera la frecuencia de operación máxima de los trabajos anteriores, a excepción de la arquitectura sintetizada en CMOS (*Complementary Metal-Oxide Semiconductor*) de 45nm, dado que los ASIC son diseñados y optimizados específicamente para realizar una única tarea, y por ende mucho más rápidos que los FPGA. Por otro lado, la arquitectura diseñada también soporta múltiples tamaños de bloques, así como una mayor cantidad de candidatos en la lista.

TABLA XXIV

COMPARACIÓN DEL PRESENTE DISEÑO CON ARQUITECTURAS DE TRABAJOS PREVIOS

	Arquitectura propuesta	Luciano V. Agostini [20]	Diniz, Cláudio [21]	Diniz, Cláudio [21]
Tecnología	FPGA Kintex 7	FPGA Virtex-II Pro	FPGA Altera Stratix III	CMOS 45 nm
Frecuencia máxima de operación	263.158 MHz	152.685 MHz	188 MHz	592 MHz
Estándar de codificación compatible	H.265	H.264	H.265	H.265
Número de candidatos en la lista	3	1	1	1
Tamaños de bloques soportados	8x8, 16x16, 32x32, 64x64	8x8	8x8	8x8

Conclusiones

Se logró diseñar una arquitectura en hardware que permite calcular el costo SATD y obtener el mejor candidato. Esta arquitectura puede ser usada dentro de la técnica de *Block Merging* según el estándar HEVC para la transmisión de video con resoluciones de hasta 4k en tiempo real con una tasa mayor a 30 cuadros por segundo.

Se logró recrear el proceso de Estimación de Movimiento Entera, para lo cual se empleó el algoritmo de TSS, que permitió hallar vectores de movimiento enteros para cada bloque de predicción, producto del particionamiento inicial, con un rango de desplazamiento de -7 píxeles hasta +7 píxeles en ambos ejes X e Y. Estos vectores de movimiento permitieron brindar los estímulos necesarios al Testbench de la arquitectura en hardware de la etapa de elección del mejor candidato.

Gracias al Software de Referencia en MATLAB y al Testbench de la herramienta Vivado Design Suite, se pudo demostrar que los resultados obtenidos por la arquitectura en hardware son correctos. También se pudo observar detalladamente todos los cálculos internos para obtener el costo SATD correspondiente a cada candidato y su mínimo valor.

Se pudieron comprobar las prestaciones de las herramientas empleadas en el diseño del presente trabajo. Por un lado, la plataforma MATLAB permitió el procesamiento de grandes cantidades de información provenientes de los cuadros de video. Una de las operaciones más usadas fue la manipulación de matrices de dimensión variable: 8x8, 16x16, 32x32 y 64x64 elementos. Por otro lado, el lenguaje Verilog facilitó de manera importante el proceso de diseño de cada componente del circuito principal. A diferencia de VHDL, Verilog tiene una sintaxis mucho más compacta. Para aquellos con experiencia en lenguajes de programación como C, Verilog será relativamente sencillo de aprender.

La arquitectura propuesta logra superar en cuanto a frecuencia de operación a algunos de los trabajos previos usados como referencia, a excepción de aquel que sintetiza el diseño en un ASIC y que, como se explicó anteriormente, será mucho más rápido que una arquitectura sintetizada en FPGA. Otras de las características resaltantes del presente trabajo es que permite procesar bloques de píxeles de dimensión variable tales como 8x8, 16x16, 32x32 y 64x64; a diferencia de los trabajos previos que procesan solamente bloques de 8x8 píxeles.

Recomendaciones

Respecto al algoritmo de *Block Merging*, como parte de futuros trabajos se podría diseñar una arquitectura que se encargue de la compensación de movimiento y la creación de la lista de candidatos, de manera que complemente el trabajo propuesto.

Sería recomendable considerar un particionamiento asimétrico en futuros trabajos, de manera que se pueda abordar un escenario mucho más real acorde a las secuencias de video actuales.

Aplicar la Estimación de Movimiento Fraccional, de las siglas en inglés FME (*Fractional Motion Estimation*), para poder hallar vectores de movimiento que describan el movimiento de bloques de manera más precisa.

Dado que el SATD es una operación que permite obtener el costo RD, el bloque de cálculo de costo SATD diseñado podría ser reutilizado en otras etapas del codificador tales como en la decisión del modo Intra Predicción o la Estimación de Movimiento Fraccional [21].

Bibliografía

- [1] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, “Overview of the high efficiency video coding (hevc) standard,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, 2012.
- [2] P. Helle, S. Oudin, B. Bross, D. Marpe, M. O. Bici, K. Ugur, J. Jung, G. Clare, and T. Wiegand, “Block merging for quadtree-based partitioning in hevc,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1720–1731, 2012.
- [3] Cisco, “Cisco annual internet report (2018-2023).” <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html>, 2020.
- [4] M. Wien, *High Efficiency Video Coding: Coding Tools and Specification*. Springer Publishing Company, Incorporated, 2014.
- [5] MatLab, “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/>, 2022.
- [6] V. Sze, M. Budagavi, and G. J. Sullivan, *High Efficiency Video Coding (HEVC): Algorithms and Architectures*. Springer Publishing Company, Incorporated, 2014.
- [7] T. S. Kim, C. E. Rhee, and H.-J. Lee, “Merge mode estimation for a hardware-based hevc encoder,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, no. 1, pp. 195–209, 2016.
- [8] J. G. M. Soto León, “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] L. H. Cancellier, A. B. Bräscher, I. Seidel, and J. L. Güntzel, “Energy-efficient hadamard-based satd architectures,” in *2014 27th Symposium on Integrated Circuits and Systems Design (SBCCI)*, pp. 1–6, 2014.
- [10] NVIDIA, “Gpu nvidia v100 tensor core.” <https://www.nvidia.com/en-us/data-center/v100/>, 2017.
- [11] C. Maxfield, *The Design Warrior’s Guide to FPGAs: Devices, Tools and Flows*. USA: Newnes, 1st ed., 2004.
- [12] J. Stankowski, T. Grajek, D. Karwowski, K. Klimaszewski, O. Stankiewicz, K. Wegner, and M. Domański, “Analysis of frame partitioning in hevc,” in *Computer Vision and Graphics* (L. J. Chmielewski, R. Kozera, B.-S. Shin, and K. Wojciechowski, eds.), (Cham), pp. 602–609, Springer International Publishing, 2014.
- [13] S. Kulkarni, D. Bormane, and S. Nalbalwar, “Coding of video sequences using three step search algorithm,” *Procedia Computer Science*, vol. 49, pp. 42–49, 12 2015.
- [14] MatLab, “Arreglos de estructuras.” https://la.mathworks.com/help/matlab/matlab_prog/create-a-structure-array.html, 2022.
- [15] H. Chaudhry Mendivil, “Diseño de una arquitectura de predicción de vectores de movimiento y cálculo de rango de búsqueda para el estándar hevc en tiempo real.” Lima, Pontificia Universidad Católica del Perú, Facultad de Ciencias e Ingeniería: Tesis para optar el título de Ingeniero Electrónico, 2018.
- [16] M. Saikia and H. Choudhury, “Comparative study of block matching algorithms for motion estimation,” 11 2013.
- [17] T. S. Kim, H.-J. Lee, and C. E. Rhee, “Highly utilized merge mode estimation for a hardware-based hevc encoder,” in *2015 IEEE Workshop on Signal Processing Systems (SiPS)*, pp. 1–6, 2015.
- [18] J. Wu, B. Guo, J. Hou, Y. Yan, and J. Jiang, “Fast cu encoding schemes based on merge mode and motion estimation for hevc inter prediction,” *KSII Transactions on Internet and Information Systems*, vol. 10, pp. 1195–1211, 03 2016.
- [19] T. S. Kim, C. E. Rhee, and H.-J. Lee, “A highly utilized hardware-based merge mode estimation with candidate level parallel execution for high-efficiency video coding,” *Journal of Signal Processing Systems*, vol. 90, 05 2018.

- [20] J. Jr, V. Possani, D. Silveira, L. Jr, and L. Agostini, “High throughput 4x4 and 8x8 satd similarity criteria architectures for video coding applications,” 04 2011.
- [21] E. Silveira, C. Diniz, M. Beck Fonseca, and E. Costa, “Satd hardware architecture based on 8x8 hadamard transform for hevc encoder,” in *2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, pp. 576–579, 2015.
- [22] B. Niehoff, “Is the cpu, gpu, fpga, or asic better?” <https://blog.samtec.com/post/is-the-cpu-gpu-fpga-or-asic-better/>, 2020.
- [23] B. Gómez, “Qué son las fpga y asic y en qué se diferencian de una cpu o gpu?” <https://www.profesionalreview.com/2020/12/18/fpga-asic-cpu-gpu/>, 2020.
- [24] N. A. Bahran and A. A. Zekry, “Matlab implementation of epzs motion estimation in h.264 avc,” *Open Access journal of science*, vol. Vol. 1, 06 2017.
- [25] M. A. Portocarrero Rodriguez, “Diseño de la arquitectura de transformada discreta directa e inversa del coseno para un decodificador hevc.” Lima, Pontificia Universidad Católica del Perú, Facultad de Ciencias e Ingeniería: Tesis para optar el título de Ingeniero Electrónico, 2018.
- [26] D. Llamocca, “Self-reconfigurable architectures for hevc forward and inverse transform,” *J. Parallel Distrib. Comput.*, vol. 109, pp. 178–192, nov 2017.
- [27] P. P. CHU, *RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability*. Wiley-IEEE Press, 2006.
- [28] M. Ferdjallah, *Introduction to Digital Systems: Modeling, Synthesis, and Simulation Using VHDL*. Wiley, first ed., 2011.
- [29] Xilinx, *UltraFast Design Methodology Guide for Xilinx FPGAs and SoCs*. Xilinx.
- [30] Xilinx, *Vivado Design Suite User Guide: Design Analysis and Closure Techniques*. Xilinx.
- [31] E. Kalali, E. Ozcan, O. M. Yalcinkaya, and I. Hamzaoglu, “A low energy hevc inverse transform hardware,” *IEEE Transactions on Consumer Electronics*, vol. 60, no. 4, pp. 754–761, 2014.
- [32] D. Ruiz, V. Adzic, H. Kalva, and G. Fernández-Escribano, “Análisis estadístico del particionado de bloques en la predicción intra-frame de hevc,” 9 2014.

- [33] T. Biatek, M. Raulet, J.-F. Travers, and O. Deforges, "Efficient quantization parameter estimation in hevc based on ρ -domain," in *2014 22nd European Signal Processing Conference (EUSIPCO)*, pp. 296–300, 2014.
- [34] S. S. U. Qadri, C. F. Azim, D. Hazry, S. F. Ahmed, M. K. Joyo, M. H. Taveer, and F. A. Warsi, "Hardware implementation of fast-sequence ordered complex hadamard transform," in *2014 IEEE 10th International Colloquium on Signal Processing and its Applications*, pp. 106–110, 2014.
- [35] R. A. Conceição, R. Jeske, B. Zatt, M. S. Porto, and L. V. Agostini, "Low-cost and high-throughput hardware design for the hevc 16x16 2-d dct transform," 2014.
- [36] C. Maxfield, *FPGAs: World Class Designs*. Newnes, 2009.
- [37] P. K. Meher, S. Y. Park, B. K. Mohanty, K. S. 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.
- [38] P.-T. Chiang and T. S. Chang, "A reconfigurable inverse transform architecture design for hevc decoder," in *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1006–1009, 2013.
- [39] D. E. Thomas and P. R. Moorby, *The Verilog Hardware Description Language, 5th Edition*. USA: Kluwer Academic Publishers, 5th ed., 2002.