PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

DISEÑO DE UNA ARQUITECTURA DE PREDICCIÓN DE VECTORES DE MOVIMIENTO Y CÁLCULO DE RANGO DE BÚSQUEDA PARA EL ESTÁNDAR HEVC EN TIEMPO REAL

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

Haris Chaudhry Mendívil

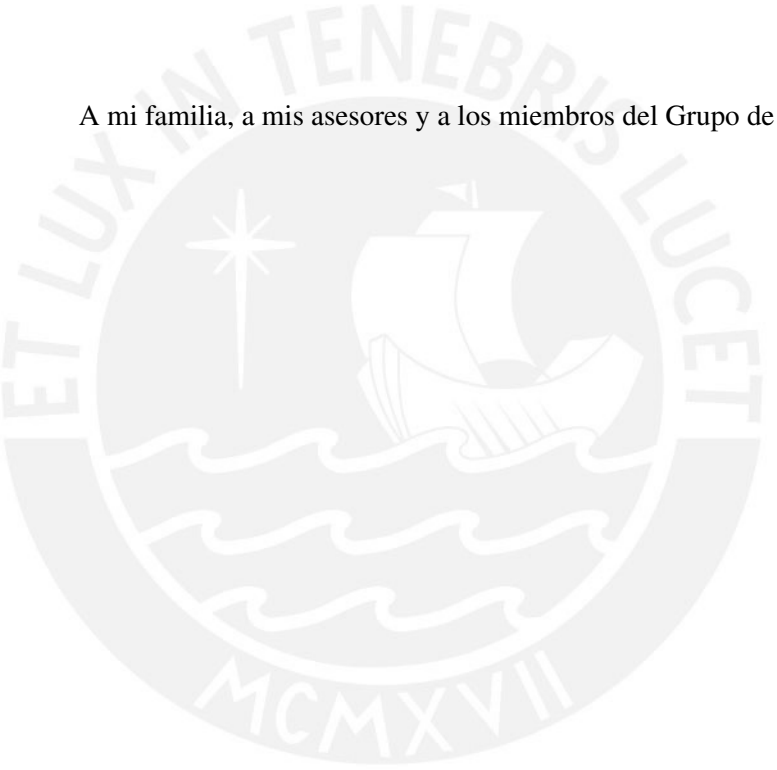
ASESORES:

Ernesto Cristopher Villegas Castillo

Mario Andrés Raffo Jara

Lima, 2018

A mi familia, a mis asesores y a los miembros del Grupo de Microelectrónica.



Resumen

El estándar HEVC (*High Efficiency Video Coding* por sus siglas en inglés) introduce nuevos elementos y técnicas en las diferentes etapas del codificador/decodificador, con el objetivo de conseguir mejoras significativas en la eficiencia de compresión. En relación a la fase de predicción de vectores de movimiento (MV del inglés *Motion Vector*), el estándar ha propuesto una técnica referida como AMVP (*Advanced Motion Vector Prediction* por sus siglas en inglés) que supone una mayor complejidad computacional que la fase de predicción implementada en el estándar previo (H.264/AVC), a costa de un ahorro considerable en términos de *bit-rate* y tiempo de ejecución. Por otro lado, algoritmos y técnicas independientes que consiguen mejoras en el software de referencia del presente estándar se han venido proponiendo en el campo de estudio; siendo uno de estos el algoritmo DSR (del inglés *Dynamic Search Range*) el cual responde a la determinación del rango de búsqueda y consigue una notable reducción en el tiempo de ejecución del proceso de estimación de movimiento (ME del inglés *Motion Estimation*).

Consecuentemente, la presente propuesta plantea el desarrollo de una arquitectura en hardware (HW) de la etapa inicial del proceso ME del codificador HEVC, con la finalidad de reducir la carga computacional del mismo. Este primer paso engloba la determinación de los MVs predictores y el cálculo del rango de búsqueda. En base a ello, se ha conseguido diseñar una arquitectura que atiende a dichos procesos fundado en los algoritmos AMVP y DSR, respectivamente. Asimismo, la arquitectura propuesta resuelve problemas de dependencia presentes en la etapa inicial del ME con la etapa ME propiamente dicha, lo cual permite potenciar el desempeño general.

Los resultados de síntesis demuestran que la arquitectura alcanza procesar secuencias de video con calidad ultra alta definición, referido también como UHD (siglas del término en inglés *Ultra High Definition*) superando los recuadros por segundo requeridos para operar en tiempo real. Específicamente, el diseño logra una tasa de procesamiento de 72 recuadros por segundo para secuencias 8K (7680x4320) con espacio de color YCbCr, en un FPGA de la familia Kintex 7.

Palabras clave — HEVC, Estimación de movimiento, Predicción de vectores de movimiento, Rango de búsqueda, AMVP, DSR, WSAD, Optimización *rate-distorsion*, Arquitectura en HW, FPGA

Índice

Índice de figuras

Índice de tablas

Índice de cuadros

Introducción

La compresión de video digital juega un papel trascendental en nuestra vida moderna; pues permite reducir la información de un contenido de video sin afectar su calidad, visto del ojo humano, lo cual permite su almacenamiento usando menos memoria y su transmisión a través de diferentes canales de comunicación. Asimismo, la compresión de video tiene un uso extendido en diferentes ámbitos como sistemas multimedia (*smartphones*, *tablets*, *laptops*, etc.), y telecomunicaciones (televisión digital, *streaming* por internet, etc.).

El presente trabajo se centra en diseñar una arquitectura en HW de la etapa inicial del proceso de estimación de movimiento del estándar HEVC, el cual es un formato actual de compresión de video digital. Los términos y las siglas mencionadas serán explicados posteriormente en el texto.

El contenido restante del documento es organizado en cinco partes: cuatro capítulos y una sección de conclusiones. El primer capítulo atiende al planteamiento de la problemática, a las plataformas de desarrollo disponibles y aspectos generales de diseño a considerar. En el segundo capítulo se discuten tópicos generales de compresión de video y estimación de movimiento, y generalidades del codificador HEVC que son del interés particular del presente trabajo. Luego, en el capítulo 3 se menciona la propuesta de diseño, los objetivos del presente trabajo, la metodología a utilizar y la arquitectura en HW propuesta. La verificación de la arquitectura propuesta y el análisis de los resultados es tratado en el capítulo 4. Finalmente, se añade una sección adicional para detallar las observaciones y conclusiones, y comentar acerca de las recomendaciones y el trabajo futuro.

Capítulo 1

Problemática, plataformas y desafíos

1.1. Definición de la problemática

1.1.1. Importancia de la compresión de video

La cantidad de datos que ocupa un video digital 'crudo' (sin comprimir) es enorme a comparación de otros tipos de archivos tales como texto, sonido o imagen. Esta enorme cantidad de bits dificulta críticamente tanto el almacenamiento como la transmisión. Por ejemplo, el ancho de banda al que usualmente opera Internet no podría manejar la transmisión en tiempo real de video digital sin comprimir (Richardson 2003: 3). Además, en la actualidad existe un creciente consumo por los contenidos digitales de tipo multimedia, especialmente contenido de video. Ello, por ejemplo, se puede ver reflejado en total del tráfico de video en Internet: se estima que para el 2018 el 79 % de todo el tráfico de internet correspondería solamente al de video (Cisco 2014:2). Por otro lado, existe una tendencia ascendente de la predominancia del contenido de video de alta definición por sobre el estándar, como se muestra en la figura 1.1.

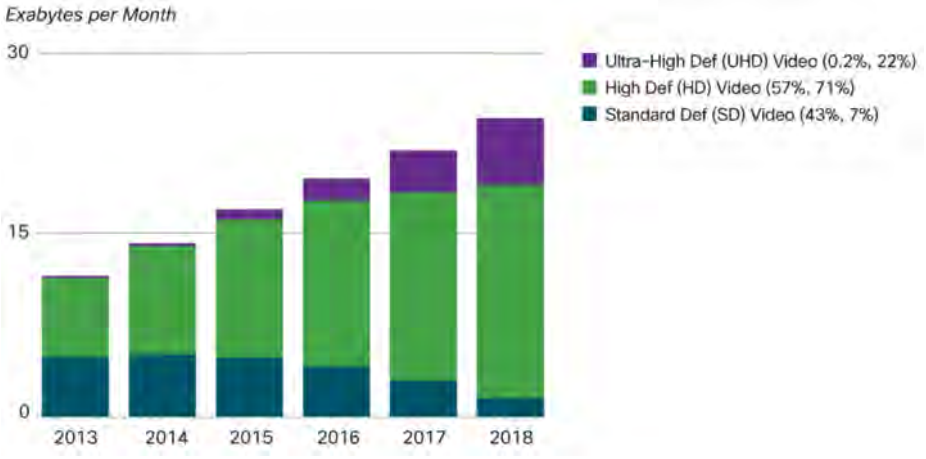


Figura 1.1: Tráfico de video de alta resolución (Cisco 2014: 8)

1.1.2. Estándares de compresión de video

En las últimas décadas, se viene realizando un esfuerzo significativo para ir mejorando las técnicas de compresión. Dichos esfuerzos se vinieron traduciendo en numerosos estándares de compresión tales como H.261, H.262 (MPEG-2), H.263, etc. (Richardson 2003: 95-97). Un estándar relativamente nuevo en compresión de video es el HEVC (H.265), *High Efficiency Video Coding* por sus siglas en inglés, el cual fue lanzado el año 2013. El mencionado estándar fue desarrollado conjuntamente por las organizaciones ITU-T *Video Coding Experts Group* (VCEG) y ISO/IEC *Moving Picture Experts Group* (MPEG) (Sze, Sullivan et al., 2014, p. 2-3). Dichos grupos son expertos en codificación y compresión de video digital, y fueron también autores de los estándares anteriores. El estándar HEVC duplica la capacidad de compresión de su predecesor: el estándar H.264/AVC *Advanced Video Coding* (Sze et al. 2014: 3-4). Ello implica el doble de calidad (mayor resolución y más cuadros por segundo) por la misma cantidad de datos, o en su defecto, la mitad de datos por la misma calidad de video. En la figura 1.2 se puede observar los distintos estándares de video que fueron desarrollados por las organizaciones mencionadas hasta la actualidad.

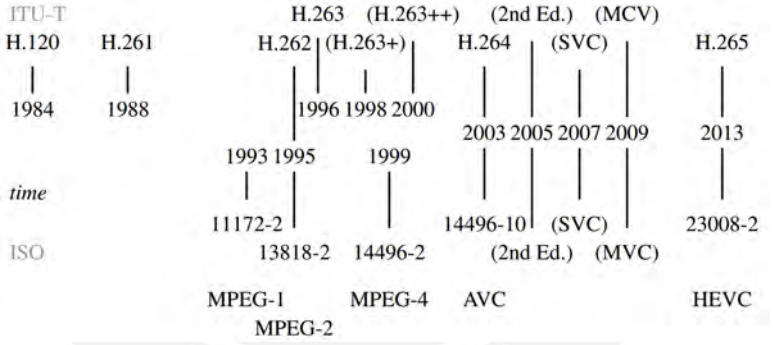


Figura 1.2: Línea de tiempo de los estándares publicados por ITU-T y ISO/IEC (Wien 2015: 3)

1.1.3. Necesidad de aceleradores dedicados

Debido a la demanda por cada vez mayores calidades de video (alto nivel de resolución) y debido al requisito de funcionar en tiempo real por parte de las aplicaciones multimedia más populares como televisión digital, *streaming* de video por comunicaciones inalámbricas (4G, wifi, etc.), es necesario desarrollar componentes del codificador y decodificador HEVC como aceleradores dedicados, sobre todo para las tareas más exhaustivas. Esta necesidad tiene su origen en el aumento de la complejidad de los últimos estándares de video (en particular, el HEVC) producto de las exigencias de la demanda mencionada. Por lo tanto, los aceleradores cumplen un papel importante al reducir la carga computacional asumida por los

codificadores/decodificadores, para que este logre satisfacer los requisitos de desempeño.

1.2. Plataformas de desarrollo

En la implementación de los aceleradores dedicados se puede considerar opciones tales como los FPGAs, ASICs y GPUs, siendo las dos primeras soluciones basadas en diseño de HW. Una implementación en HW está por sobre las que utilizan SW, las cuales son implementadas utilizando dispositivos tales como CPUs o GPUs (Kanupriya 2010: 7,10,19). Ello debido al alto nivel de paralelismo en cuanto a operaciones que se puede conseguir a través de una arquitectura HW, a diferencia del proceso secuencial que sigue las instrucciones del SW ejecutado en un CPU; sobre todo si se pretende implementar el proceso de ME, el cual consta de las operaciones computacionalmente más costosas para el codificador HEVC (Sullivan et al. 2012: 1661).

Cabe destacar que los GPUs de propósito general (GP-GPU del inglés, *General Purpouse*), poseen capacidades computacionales altas debido a que su arquitectura está compuesta por una gran cantidad de núcleos independientes entre sí, además de una gran capacidad de memoria permitiendo realizar un procesamiento en paralelo para aplicaciones de video, no obstante, suelen ser prohibitivos por motivo de su alto consumo de potencia, más aún cuando los códecs de video se implementen en dispositivos móviles (Kanupriya 2010: 18). Entonces, su uso es descartado para aplicaciones móviles o portátiles de alto desempeño. Asimismo, lo que finalmente resulta más óptimo, tanto en velocidad como en consumo, es implementar los algoritmos de compresión de video en ASICs. Sin embargo, la implementación en ASIC suele ser muy exhaustiva para una fase inicial de diseño y prototipado. Ante ello, la solución basada en FPGA surge como una alternativa óptima y viable para el propósito de desarrollo de arquitecturas HW.

1.3. Desafíos y consideraciones de diseño

En el diseño de la arquitectura propuesta se deben considerar dos aspectos principales: el desempeño y la potencia requerida por el circuito para su funcionamiento, los cuales son criterios fundamentales considerados para el diseño de cualquier sistema digital. Ya que en caso de no cumplir con el requerimiento de desempeño, se experimentaría un retardo en el receptor (en una transmisión en tiempo real); en su defecto, si no se cumple el requerimiento de potencia (si es muy alta), la plataforma física que energiza el circuito podría recalentarse o quedarse sin energía rápidamente (en el caso de un dispositivo móvil).

En tiempos anteriores se consideraba exclusivamente incrementar el desempeño del circuito, lo

cual se logra maximizando la frecuencia de reloj; sin embargo, en décadas recientes, la estrategia de diseño seguida es alcanzar el máximo desempeño sin excederse de ciertos límites de potencia, lo cual implica una optimización simultánea de ambos parámetros (Zlatanovici 2006: 1). En la presente propuesta, se considera únicamente una restricción de desempeño fijo (mayor a 30 fps), dado que no se especifica una plataforma en particular; no obstante, ello no implica incrementar indiscriminadamente la potencia consumida. Entonces, la estrategia de diseño a seguir, en este caso, se traduciría en maximizar el desempeño hasta conseguir superar los 30 fps propuestos, considerando en dicho proceso de diseño la minimización de área, lo cual se relaciona directamente con la potencia requerida pues la energía depende de la cantidad de compuertas (conmutación de estas) y la potencia de pérdida en estado de reposo (referida como *leakage power* en inglés) (Zlatanovici 2006: 46).

Las optimizaciones (mejora de una o más variables) y también *trade-offs* (mejora de una o más variables a costa de otras) se puede realizar en el proceso de síntesis de alto nivel —el cual mapea la descripción realizada en un HDL a un nivel RTL— y síntesis lógica, que se encarga de traducir la descripción RTL a nivel de compuertas (Damiani 1994: 1-2). Estas optimizaciones se efectúan tanto en el ámbito de la lógica combinacional (optimizaciones de lógica de 2 niveles, multi-nivel, heurísticas iterativas, componentes aritméticos del *datapath*, compartimiento de recursos, etc.) como en el ámbito de la lógica secuencial (reducción de estados equivalentes, codificación de estados, etc.), y son llevados a cabo por los sintetizadores modernos (Vahid 2010: 325-375).

Sin embargo, existen técnicas de diseño claves para un desempeño alto a nivel de la descripción RTL, el cual puede ser realizado en el código HDL. Tales criterios son el *pipelining* y la concurrencia, los cuales responden a agregar registros intermedios para dividir una tarea en etapas y en considerar subpartes que se ejecuten de forma simultánea, respectivamente. En el caso del *pipelining*, lo que se realiza es reducir el camino crítico máximo (entre dos registros) para aumentar la frecuencia de operación; lo cual resulta finalmente en un *throughput* (cantidad de datos que se puede procesar por unidad de tiempo) mayor. Adicionalmente, se pueden considerar optimizaciones a nivel de la FSM, como por ejemplo, eligiendo salidas de tipo Moore o Mealy. En general, la selección del algoritmo tiene el impacto más significativo en las diferentes variables (desempeño, potencia, área, etc.), ya que este puede determinar el nivel de concurrencia, cantidad de operaciones, posibilidad de agregar niveles de *pipeline*, entre otros (Vahid 2010: 388).

Capítulo 2

El estándar HEVC y la estimación de movimiento

2.1. Generalidades de la compresión de video digital

2.1.1. Dominios de una señal de video

Una señal de video contiene dos dominios fundamentales, ilustrado en la Figura 2.1: el espacial y el temporal (Bovik 2005: 556). El dominio espacial hace referencia a la información contenida en un mismo cuadro; por otro lado, el dominio temporal hace referencia a la información contenida entre cuadros adyacentes. Los métodos de compresión actúan en estos dominios para reducir la información redundante. De forma particular, el conjunto de técnicas que eliminan la redundancia temporal suelen ser referenciados como 'inter-predicción'. La inter-predicción abarca procedimientos —que finalmente son componentes o etapas del codificador/decodificador del estándar de compresión— tales como estimación de movimiento, predicción de vectores de movimiento, compensación de vectores de movimiento, entre otros.

2.1.2. Visión general del codificador HEVC

El presente trabajo se centra en la etapa inicial del módulo ME, el cual forma parte del codificador HEVC. En la Figura 2.2 se muestra el módulo de interés dentro de su contexto de operación. El procedimiento general que ejecuta el codificador HEVC se puede explicar con la siguiente secuencia de pasos (Xiao 2016: 4-6):

1. Un recuadro de la señal de video ingresante es particionada en bloques siguiendo un árbol de particionado, explicado anteriormente.

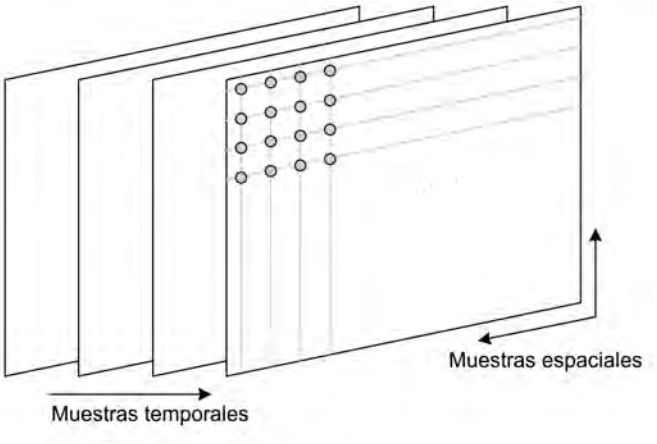


Figura 2.1: Dominio espacial y temporal de una secuencia de video (Richardson 2003: 10)

2. Cada bloque del recuadro es ingresado a una etapa de intra-predicción (busca la mejor correlación en vecinos del mismo recuadro) o inter-predicción; los cuales tienen como salida un bloque candidato que atiende a la redundancia espacial y temporal, respectivamente. Cabe resaltar que el estimador de movimiento brinda el MV a un componente interno de la etapa de Inter-predicción, el cual es referenciado como 'compensador de movimiento'. Dicho componente se encarga de reconstruir el bloque candidato a partir del MV y el cuadro de referencia (Lin et al. 2010: 5).

3. La diferencia entre el bloque candidato (predicho) y el bloque original, el cual es llamado residual, es ingresado al bloque de transformación, escalamiento y cuantización.

4. El proceso de transformación convierte la información residual ingresante en coeficientes de transformación en el dominio de la frecuencia, usando algoritmos como la transformada de coseno discreta (DCT, del inglés *Discrete Cosine Transform*); el cual elimina información redundante adicional.

5. El proceso de cuantización permite reducir el rango de la señal para ahorrar cierta cantidad de bits en la transmisión, esto implica una compresión que es ajustada por un parámetro de cuantización (QP, del inglés *Quantization Parameter*).

6. Finalmente, los coeficientes de transformación cuantizados y los datos de movimiento obtenidos en la etapa de inter-predicción son convertidos en un *bitstream* aún más comprimido por la unidad de codificación entrópica.

Es pertinente destacar que existe una cadena de componentes propios del decodificador HEVC —como la transformada inversa y el filtro de lazo (se encarga de eliminar artefactos de

compresión)— debido a que se requiere guardar imágenes que servirán como cuadros de referencia para bloques posteriores en un *buffer*.

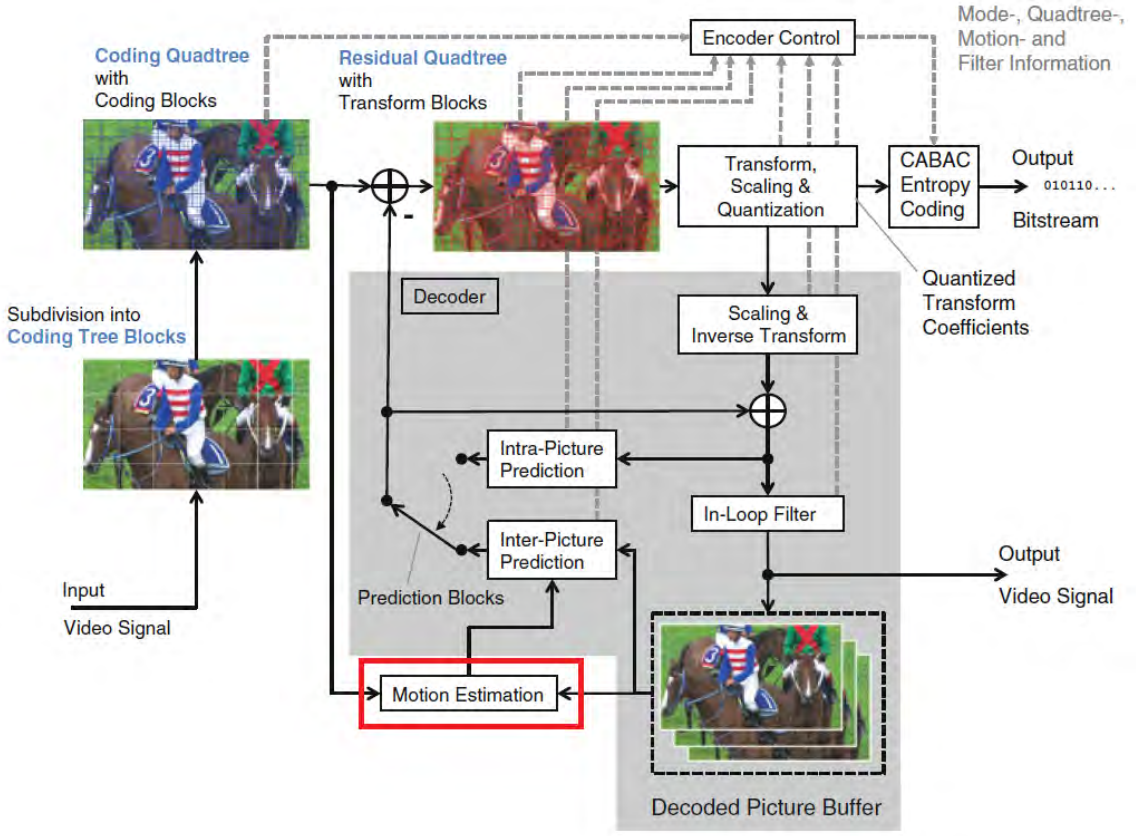


Figura 2.2: Esquema general del codificador HEVC con el módulo ME resaltado (Sze et al. 2014: 51)

2.2. Estimación de movimiento

2.2.1. Estimación de movimiento basado en *matching* de bloques

En la búsqueda por reducir la redundancia temporal, los últimos estándares de video utilizan una técnica denominada estimación de movimiento (ME del inglés *Motion Estimation*), particularmente se utiliza el de tipo '*matching* de bloques'. Dicha técnica consiste en buscar bloques de píxeles cuadrados (o rectangulares) similares a un bloque perteneciente al recuadro actual, el cual es también conocido como unidad de predicción (PU, del inglés *Prediction Unit*). Esta búsqueda de bloques similares es realizada en un área de búsqueda —que tiene como centro la ubicación del PU actual— perteneciente a un cuadro de referencia (lo cual suele ser un cuadro anterior). Finalmente, este proceso tiene como objetivo expresar el PU del cuadro actual como una traslación de un bloque presente en el cuadro de referencia. Esta traslación es expresada

como un movimiento en dos ejes (vertical y horizontal), y es referido como 'vector de movimiento' (MV del inglés *Motion Vector*), como se ilustra en la figura 2.3. Entonces, se disminuye la redundancia temporal al expresar el cuadro actual como un 'movimiento' de los bloques del cuadro de referencia; lo cual implica codificar únicamente los vectores de movimiento, en contraste con codificar todos los píxeles del bloque actual (Lin et al. 2010: 4).

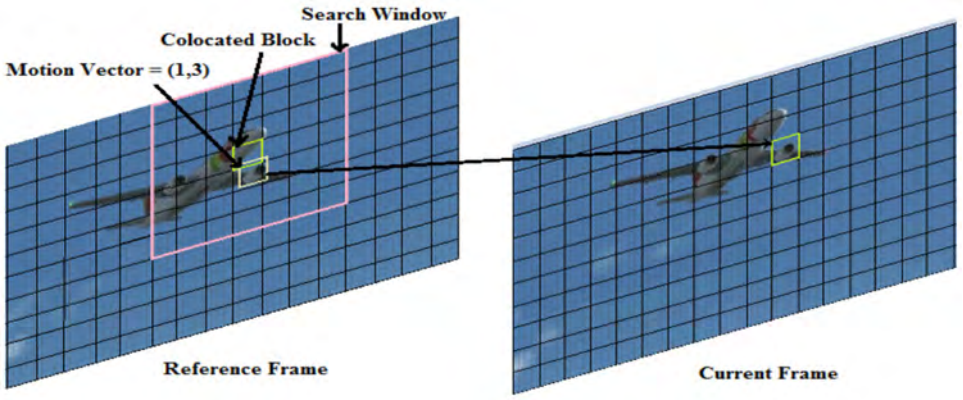


Figura 2.3: Proceso ME con un MV en particular (Nalluri 2016: 4)

2.2.2. Métricas de similaridad usadas en el proceso ME

Para evaluar la similaridad entre los bloques del cuadro de referencia con el PU del cuadro actual (en cada posición del rango de búsqueda) se pueden usar diversas métricas; sin embargo, las más usadas suelen ser el error cuadrático medio (MSE, por sus siglas en inglés) y la suma de diferencias absolutas (SAD por sus siglas en inglés). Luego, un menor valor de MSE o SAD, implica una mayor similaridad entre los píxeles de ambos bloques. En las ecuaciones 2.1 y 2.2 se muestran las relaciones correspondientes al MSE y SAD orientados a la etapa ME.

$$MSE = (\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}(C(i,j) - R(i,j))^2)^{\frac{1}{2}} \tag{2.1}$$

$$SAD = \sum_{i=0}^{M-1}\sum_{j=0}^{N-1}|C(i,j) - R(i,j)| \tag{2.2}$$

Donde

- M y N son las dimensiones del bloque PU (o el presente en el cuadro de referencia)
- $C(i,j)$ representa los pixeles del bloque PU (del cuadro actual)
- $R(i,j)$ representa los pixeles del bloque del cuadro de referencia

Cabe resaltar que el MSE es la métrica que mayor precisión brinda al evaluar la similaridad (Bing 2009: 3). Por otro lado, el SAD es utilizado ampliamente por arquitecturas en HW para la etapa ME como una aproximación al MSE, ya que prescinde de operaciones de multiplicación, los cuales son costosos en HW.

2.2.3. Particionado de bloques en el proceso ME del estándar HEVC

Es importante resaltar que el formato de codificación HEVC (y formato previos a este) dividen el recuadro actual en bloques rectangulares más pequeños. Particularmente, el estándar HEVC efectúa un particionado multi-nivel que sigue un diagrama de árbol (referido en inglés como *coding tree*). El recuadro completo es particionado, en un primer nivel, como un conjunto de unidades de codificación de árbol (CTU, del inglés *Codig Tree Unit*) de 64x64 generalmente. En niveles posteriores, de forma recursiva, el CTU es particionado en unidades de codificación (CU, del inglés *Codig Unit*); es decir, es divido en 4 partes iguales. En cada nivel, el CU es particionado en PUs, para el proceso de inter-predicción (o intra) (Sullivan et al. 2012: 1654-1656). El PU, que responde al interés particular de la presente propuesta, puede contar con divisiones simétricas, asimétricas o el mismo tamaño que el CU (Sze et al.2014: 63). El proceso descrito se ilustra en la figura 2.4.

Por lo tanto, en el estándar HEVC, el proceso ME se tiene que realizar para todos los PUs obtenidos en cada nivel del arbol de particionado, los cuales constituyen un total de 593 para un CTU de 64x64 (Nalluri 2016: 39). Entonces, el módulo ME debe soportar el cálculo de MVs para todos los tamaños de bloque posibles, esta característica es usualmente denominada como VBSME (*Variable Block Size Motion Estimation* por sus siglas en inglés).

2.2.4. Predicción de vectores de movimiento

Esta técnica consiste en utilizar los MVs calculados previamente para predecir el MV del PU actual. Esto es factible debido a que altamente probable que PUs cercanos se desplacen siguiendo el mismo patrón (tengan MVs parecidos). Cabe destacar que los MVs previamente calculados pueden pertenecer a PUs cercanos tanto espacial como temporalmente. Luego, ello trae consigo dos beneficios significativos. El primero es la posibilidad de codificar la diferencia de MVs del predicho y el real, como se muestra en la ecuación 2.3, en vez del MV real; lo cual se traduce en un ahorro en la cantidad de bits a utilizados para su codificación (Wien 2015: 184). La segunda utilidad radica en la determinación del centro del rango de búsqueda; esto significa un acercamiento al MV real, lo cual reduce en gran medida el costo computacional asociado a la

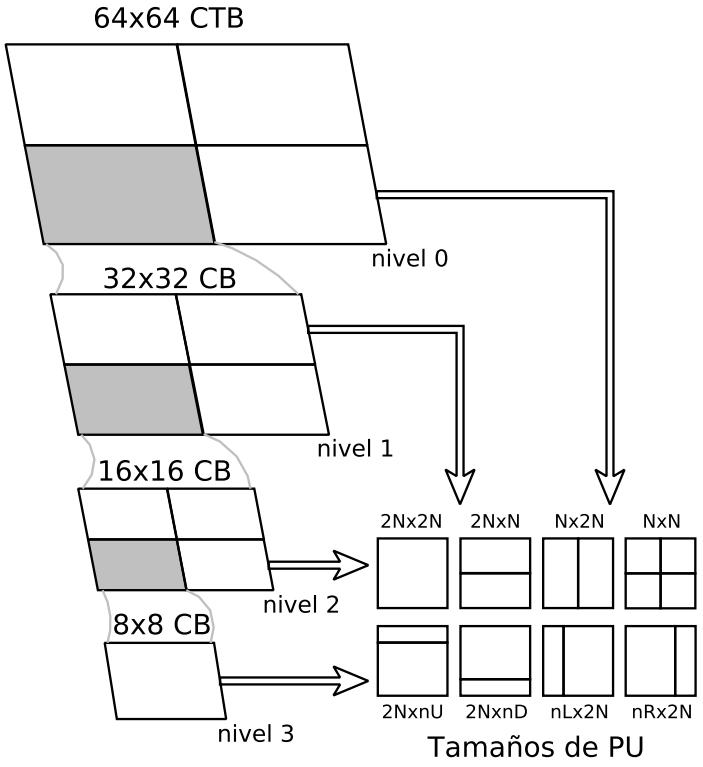


Figura 2.4: Particionado del CTU por niveles, y posibles PUs por nivel

búsqueda del MV real u óptimo.

$$MVD = MV_{real} - MVP \tag{2.3}$$

Donde

- MVD es la diferencia de MVs, del inglés *Motion Vector Difference*
- MV_{real} es el MV asociado al PU, o MV real u óptimo
- MVP es el MV predicho, del inglés *Motion Vector Predictor*

2.2.5. Predicción de MVs en el estándar HEVC: Etapa AMVP

La etapa AMVP corresponde al algoritmo seguido para determinar la lista de candidatos —de los cuales se extrae el mejor MVP- en el estándar HEVC. Dicha lista consta, finalmente, de dos elementos y es construida a partir de los candidatos espaciales (ubicados arriba y a la izquierda del PU actual) y temporales (el co-localizado y a la esquina derecha inferior del co-localizado) disponibles en las posiciones mostradas en la figura 2.5. El algoritmo comienza seleccionando

2 candidatos espaciales de los 5 posibles (1 del lado izquierdo y 1 de arriba), y 1 temporal de los 2 posibles. La selección se realiza de acuerdo a la disponibilidad (puede estar codificado en modo 'intra'); y para los espaciales, se considera adicionalmente el escalamiento (esto se explica posteriormente). Después, se procede a remover los MVs duplicados. Como paso final, en caso se disponga menos de 2 candidatos, se agregan MVs nulos de (0,0); y en caso se disponga de más de 2 candidatos, se priorizan los espaciales. Todo el proceso anterior es descrito en la figura 2.6.

Los candidatos que tengan un cuadro de referencia diferente al cuadro de referencia de PU actual deben ser escalados (al cuadro de referencia de PU actual). Este escalamiento es obligatoriamente aplicado para los candidatos temporales, y es aplicado a los espaciales solo en caso de que los cuadros de referencia difieran. Es pertinente destacar que en el proceso de selección descrito en el parrafo anterior, se priorizan los candidatos espaciales no escalados. Las ecuaciones 2.4, 2.5 y 2.6 describen la operación de escalamiento del MV.

$$MV_S = signo(MV_{cand}.f).((|MV_{cand}.f| + 2^7) \gg 8) \tag{2.4}$$

$$f = \begin{cases} 2^{12} - 1, & [(tb.tx + 2^5) \gg 6] > 2^{12} - 1 \\ (tb.tx + 2^5) \gg 6, & otros \\ 2^{-12}, & [(tb.tx + 2^5) \gg 6] < 2^{-12} \end{cases} \tag{2.5}$$

$$tx = \frac{2^{14} + \frac{td}{2}}{td} \tag{2.6}$$

Donde

- tb es la distancia temporal entre el cuadro de referencia y el cuadro actual del PU actual
- td es la distancia temporal entre el cuadro de referencia y el cuadro actual del PU candidato
- f es el factor de escalamiento
- MV_{cand} es el MV del candidato
- MV_S es el MV del candidato escalado

2.2.6. Concepto de *Rate-Distortion*

El concepto de RD (*Rate-Distortion* por sus siglas en inglés) en compresión de video, alude a la relación que existe entre la cantidad de bits codificados (R) y la calidad o fidelidad del video

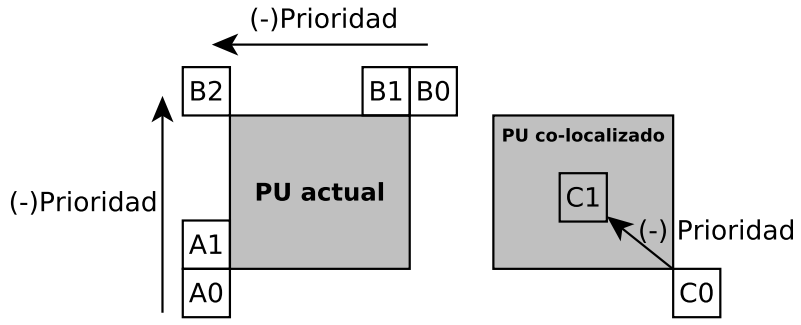


Figura 2.5: Posiciones de los candidatos espaciales y temporales considerados en la etapa AMVP, y su orden de prioridad

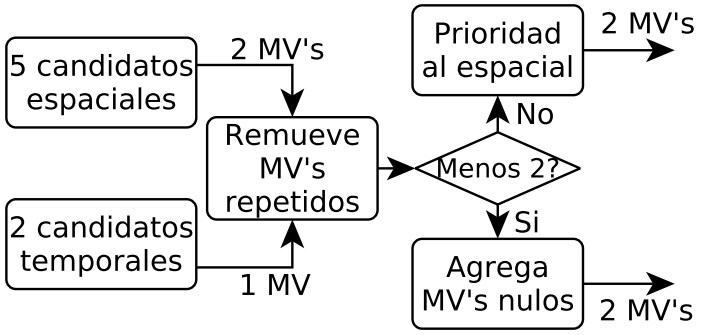


Figura 2.6: Selección de candidatos del algoritmo AMVP

digital (D). Este último puede ser representado por métricas que miden la distorsión (o similaridad) como el PSNR (*Peak to Signal Noise Ratio* por sus siglas en inglés), el MSE o el SAD (como se mencionaron en secciones anteriores). Esta relación es expresada por una curva RD que relaciona ambos parámetros, como se muestra en la figura 2.7, en este caso, cada punto de la curva implica un parámetro de cuantización (QP) determinado configurado en el *encoder*.

2.2.7. Métrica de Bjøntegaard

Es una métrica que calcula las diferencias en promedio de PSNR y *bit-rate* de dos curvas RD, los cuales se denominan BD-PSNR y BD-*rate* respectivamente. Lo mencionado se realiza ajustando una curva entre un grupo de puntos de R y D, y calculando la diferencia de las integrales de ambas curvas, respecto del eje R o D (para cada caso). Esta métrica es ampliamente usada para calcular la eficiencia o desempeño de algoritmos y técnicas de compresión (Hanhart & Ebrahimi 2014: 1-3).

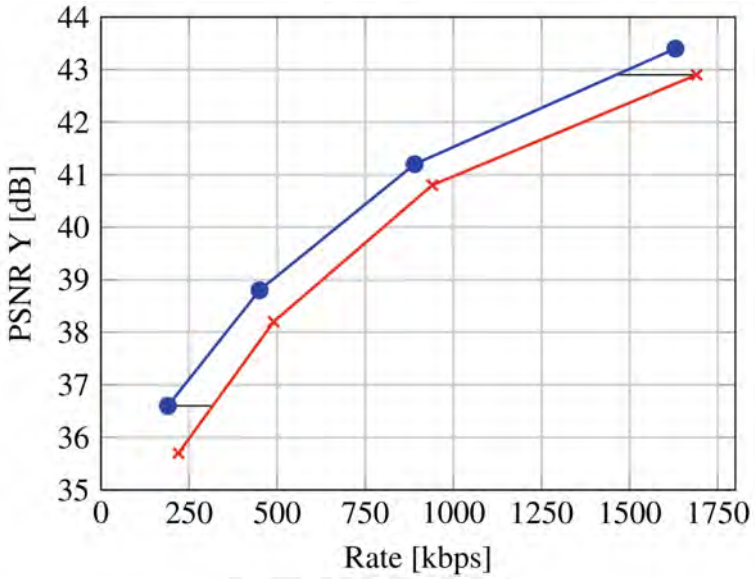


Figura 2.7: Ejemplo de curvas RD con cuatro puntos de QP (Wien 2015: 96)

2.2.8. Optimización de RD para estimación de movimiento

Consiste en obtener la mínima distorsión ante un determinado límite de cantidad de bits (o *bit-rate*, que son los bits por unidad de tiempo). Estos pares óptimos de R y D forman una curva convexa; entonces, el proceso de optimización consiste en conseguir pares que estén lo más cerca posible a dicha curva (Richardson 2002: 217-218).

Una forma de conseguir lo mencionado es con la 'optimización de Lagrange', el cual propone minimizar la función J, descrita en la ecuación 2.7. Ello, se realiza fijando un parámetro denominado 'multiplicador de Lagrange' (representado por λ); dado que cada valor de λ brinda la pendiente de una recta tangente a la curva óptima, y el mínimo valor de J equivale al punto de cruce con dicha curva (Richardson 2002: 218).

$$J = D + \lambda R \tag{2.7}$$

Esta técnica de optimización puede ser usada para elegir el mejor MV en el proceso de ME propiamente dicho o para el elegir el mejor candidato predictor en la etapa AMVP (de los dos seleccionados) (Laroche et al. 2008: 1249). Esto, debido a que las métricas de similaridad (mencionadas en secciones anteriores), el cual equivale al término D, no considera el parámetro R, el cual es importante en cualquier codificador. Particularmente, en la ecuaciónes 2.8 y 2.9 se muestra la aplicación de la optimización RD para la selección del MVP en la etapa AMVP usando el SAD como métrica de distorsión.

$$MVP = \underset{MV_n}{\text{argmín}}\, J(MV_n), \quad MV_n \in \{MV_1, MV_2\} \tag{2.8}$$

$$J(MV_n) = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} |C(i,j) - R(i + (MV_n)_x, j + (MV_n)_y)| + \lambda.R(MV_n) \tag{2.9}$$

Donde

- MV_n es uno de los dos MVs candidatos
- M y N son las dimensiones del bloque PU (o el presente en el cuadro de referencia)
- $C(i,j)$ representa los pixeles del bloque PU (del cuadro actual)
- $(MV_n)_x$ y $(MV_n)_y$ representan los componentes horizontales y verticales del MV candidato
- $R(i + (MV_n)_x, j + (MV_n)_y)$ representa los pixeles del bloque desplazado por el MV candidato en el cuadro de referencia del PU actual

Cabe precisar que el valor de λ depende de la configuración del *encoder*, el cual es determinado por la aplicación (el tipo de video a codificar) (Wien 2015: 66).

Capítulo 3

Propuesta, consideraciones y diseño

3.1. Propuesta de diseño

Visto que la técnica AMVP constituye un factor importante para el incremento en eficiencia conseguido por el estándar HEVC, y debido a la considerable capacidad de procesamiento requerida en el mismo, se propone realizar una implementación en HW digital del algoritmo AMVP. Asimismo, se propone incluir un módulo HW para la determinación del rango de búsqueda; puesto que el MVP y el rango son los parámetros iniciales requeridos por el proceso de ME.

Se plantea el FPGA como alternativa de desarrollo ya que permite un entorno más propicio para el prototipado en contraste con una implementación en ASIC; ello, tomando en consideranción el desarrollo de un estimador de movimiento completo para estándar HEVC en el futuro. Lo mencionado implica organizar los tiempos y las señales de comunicación del acelerador propuesto con la arquitectura ME externa (un acelerador de mayor cobertura), y con el encoder externo que realiza las otras operaciones concernientes al formato HEVC.

3.2. Objetivos

3.2.1. Objetivo principal

Diseño de una arquitectura en HW del predictor de MVs basado en algoritmo AMVP y del módulo de cálculo del rango de búsqueda, que incorpore las mejoras recientes sobre el estándar HEVC para secuencias de video UHD en tiempo real (mayor a 30 cuadros por segundo).

3.2.2. Objetivos específicos

1. Estudio de las mejoras recientes en relación a la etapa de predicción de MVs y cálculo del rango de búsqueda.
2. Estudio de las arquitecturas en HW correspondientes a la etapa inicial del proceso ME
3. Descripción y verificación de la arquitectura propuesta
4. Optimización del desempeño para cumplir los requerimientos planteados

3.3. Metodología

La metodología seguida para el diseño de la arquitectura será de tipo *top-down* (descripción gradual partiendo del sistema hacia sus componentes elementales), el cual es preferido en la actualidad por sobre el *bottom-top* (construcción del sistema a partir de la descripción de sus componentes elementales); lo mencionado se sustenta en las siguientes razones: requiere menor tiempo de diseño, el costo NRE es reducido, los diseños son fácilmente reutilizables, mayor flexibilidad para realizar cambios, mayor flexibilidad para cambiar de tecnología, mayor facilidad para verificar el diseño, etc. (Smith 1996: 3). Se realizará el diseño RTL en base al lenguaje de descripción de hardware (HDL) Verilog HDL, ya que su uso está más extendido en la industria y suele ser fácil de codificar (Palnitkar 2003: 23). Adicionalmente, se pretende emplear una verificación en Verilog basado en *assertions*, estímulos aleatorios y modelos en SW —correspondiente a un ambiente de sanidad— para los módulos y unidades que requieran una revisión más elaborada. Finalmente, se realizarán optimizaciones usando técnicas de *pipelining*, concurrencia, reducción de estados, entre otros para alcanzar los requerimientos plateados en base a los reportes de síntesis.

3.4. Consideraciones iniciales de la arquitectura en HW propuesta

3.4.1. Visión general de la arquitectura AMVP

En la figura 3.1 se muestra un diagrama conceptual de los componentes presentes en una arquitectura en HW que ejecutaría el algoritmo AMVP, basado en la implementación realizada por Abdelsalam de la misma (2017: 14). Luego, las sub-partes requeridas serían las siguientes:

- **Unidad de selección**: Se encarga de seleccionar dos MVs candidatos —con sus respectivos números de recuadro actual y de referencia— a partir de los candidatos espaciales y

temporales.

- **Unidad de escalamiento**: Se encarga de escalar los dos MVs seleccionados, en caso sea necesario.
- **Unidad de cálculo del número de bits**: Se encarga de calcular la cantidad de bits requeridos para codificar cada MV escogido.
- **Unidad de cálculo de distorsión**: Se encarga de calcular la distorsión asociada a los MVs seleccionados.
- **Unidad de cálculo del costo RD**: Se encarga de calcular la costos RD asociados a cada MV seleccionado.
- **Comparador**: Se encarga de seleccionar el MV con el menor costo RD como el MVP.

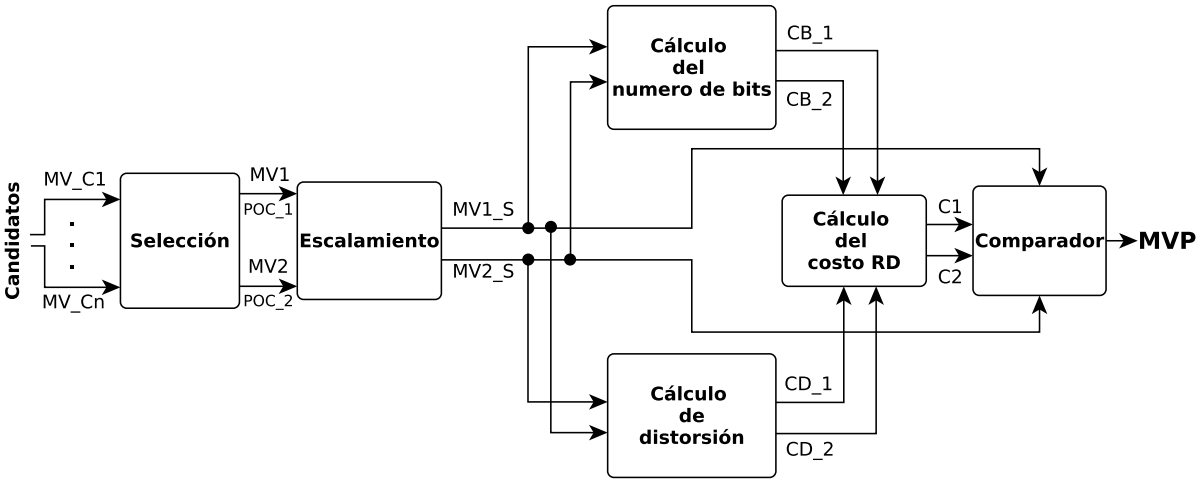


Figura 3.1: Diagrama conceptual de la arquitectura AMVP

3.4.2. Problema de la dependencia de datos entre etapa AMVP y ME

Dado que para iniciar el proceso ME, se precisa del centro de búsqueda, el cual es resultado de la etapa AMVP, y el hecho de que la etapa AMVP requiere de los MVs vecinos calculados por el proceso ME, existe un problema de dependencia de datos entre ellos, lo cual retarda el inicio del proceso ME dependiente e impide conformar un *pipeline* entre ambas etapas. Además, considerar el punto (0,0) (o el MV nulo) como centro —lo cual equivaldría a omitir la etapa AMVP— resulta, finalmente, en un tiempo de búsqueda considerablemente mayor, o valores altos de *bit-rate* (Jou et al. 2015: 1535). Por consiguiente, la resolución de este problema es relevante. El mencionado

inconveniente se ilustra de mejor forma en la figura 3.2, el cual muestra, de forma particular, la ruptura del *pipeline* en los PUs 32x32_2 y 32x32_3; ya que sus etapas AMVP (incluido el cálculo del costo RD) requieren de los cálculos de los MVs de 32x32_1 y 32x32_2, respectivamente.

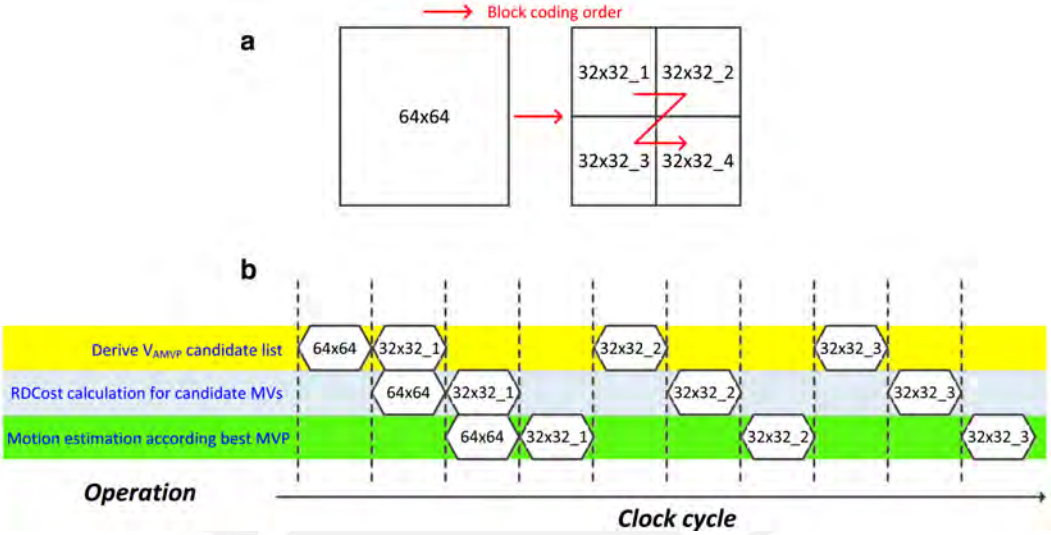


Figura 3.2: Ejemplo de dependencia entre etapas AMVP y ME (Wang & Li 2016: 4)

Para resolver este problema, Wang & Li (2016) proponen un método de interpolación de MVs a partir del MV del PU de mayor tamaño (nivel de particionado cero), el cual es referido como LCU. Ello, debido a que existe una relación lineal aproximada entre los MVPs de los PU más pequeños con el MVP del LCU. El modelo lineal para la interpolación de estos MVs es descrita por la ecuación 3.1.

$$MV_{int} \begin{cases} (MV_{int})_x = \alpha_x.(MVP_{LCU})_x + \beta_x \\ (MV_{int})_y = \alpha_y.(MVP_{LCU})_y + \beta_y \end{cases} \tag{3.1}$$

Donde

- $(MV_{int})_x$ y $(MV_{int})_y$ son los componentes horizontales y verticales del MV interpolado
- $(MVP_{LCU})_x$ y $(MVP_{LCU})_y$ son los componentes horizontales y verticale del MVP del LCU
- α_x y β_x son constantes para la componente horizontal (dependen del tamaño del PU)
- α_y y β_y son constantes para la componente vertical (dependen del tamaño del PU)

Por lo tanto, se propone utilizar estos MVs interpolados como reemplazo de los MVs no disponibles de los PUs vecinos (debido a que el proceso ME no culmina); es decir, el MV

interpolado es agregado a la lista de candidatos posibles. Cabe resaltar que este método de interpolación incrementa el BD-*rate* por 0.99 % en promedio a costa de incrementar el *throughput* del HW por más de 50 % aproximadamente (Wang & Li 2016: 4-8).

3.4.3. Elección de la métrica de distorsión

En cuantos a las métricas de distorsión, se suele preferir el SAD para implementaciones en HW por su menor utilización de recursos. Mientras que el MSE, siendo la opción que brinda mayor calidad, suele ser descartado para estas aplicaciones debido a que requiere de muchos recursos, tales como multiplicadores, arquitecturas de raiz cuadrada, etc. No obstante, existe una métrica que teóricamente se acerca al MSE en mayor medida que el SAD, y que demanda poca complejidad en cuanto a recursos de HW. La mencionada métrica es referida como WSAD, y fue propuesta por Bing (2009). El WSAD aplicado a vectores de dos elementos (o arreglos de 1x2) se muestra en la ecuación 3.2, donde d_1 es la diferencia de los elementos de la primera posición de los vectores a comparar y d_2, de los de segunda posición. Como se observa en la ecuación mencionada, la métrica WSAD únicamente requiere de comparadores, desplazadores y sumadores.

$$WSAD = \begin{cases} |d_1| + \frac{|d_2|}{2}, & |d_1| \geq |d_2| \\ \frac{|d_1|}{2} + |d_2|, & |d_1| \leq |d_2| \end{cases} \tag{3.2}$$

Sin embargo, se requiere utilizar la métrica para cualquier número de elementos; para ello, a partir de aproximaciones sucesivas del WSAD de dos elementos al MSE, se obtiene un árbol binario de WSADs de dos elementos (ilustrada en la figura 3.3), como herramienta de cálculo para un WSAD de n elementos (Bing 2009: 39).

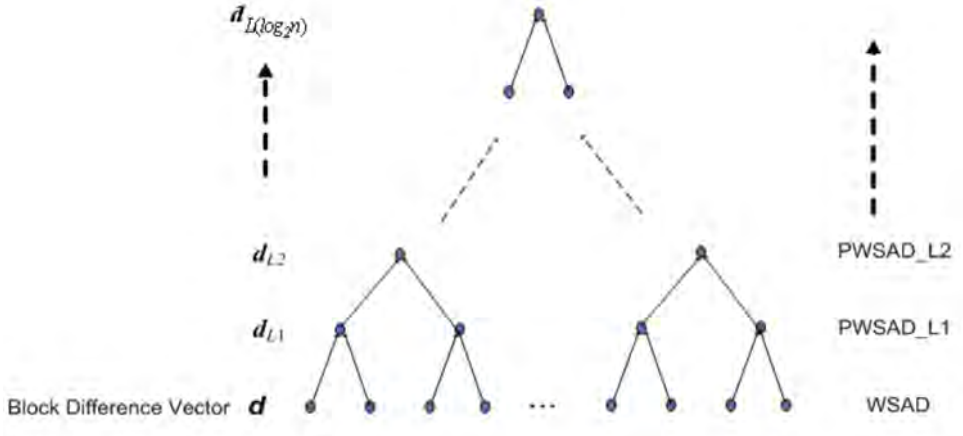


Figura 3.3: Árbol de cálculo del WSAD de n elementos (Bing 2009: 39)

3.4.4. Determinación del rango de búsqueda - Algoritmo DSR

El algoritmo DSR (*Dynamic Search Range*, por sus siglas en inglés) sugerido por Nalluri (2016), propone el uso de distancias euclidianas de los MVDs para la determinación del rango de búsqueda, como se indica en la ecuación 3.3 ($r(MVD)$ es el rango asociado al vector). Ello se justifica en el hecho de que la distancia mínima para que el centro de búsqueda (definido por el MVP) encuentre el MV óptimo es el MVD.

$$r(MVD) = \sqrt{MVD_x^2 + MVD_y^2} \tag{3.3}$$

De esta manera, se proponen 3 tipos de candidatos. El primer tipo se constituye por los MVDs de los PUs vecinos espaciales, y se consideran las posiciones izquierda-inferior, superior-izquierda y superior-derecha, los cuales equivalen a las posiciones A1, B2 y B0 (del AMVP), respectivamente. El segundo tipo está formado por los MVD de los PUs del nivel anterior al nivel del PU actual, los cuales ya fueron calculados. Para los PU rectangulares, se toma el MVD del PU de tipo 2Nx2N del nivel anterior; y para los PU cuadrados, se toma el máximo de los MVDs de los PUs de tipo 2NxN y Nx2N. Finalmente, el tercer tipo considera el candidato temporal co-localizado al PU actual (C0 en la etapa AMVP); aunque, para este caso, se toma en cuenta el MV en reemplazo del MVD, ya que su MVP tiene dependencia espacial de otro cuadro.

Luego, con los candidatos mencionados, se determina el rango de búsqueda de acuerdo a las ecuaciones 3.4, 3.5, 3.6, 3.7 y 3.8.

$$SR = \begin{cases} SR_{ENC}, & 2.SR_{pred} \geq SR_{ENC} \\ 2.SR_{pred}, & otros \\ \frac{SR_{ENC}}{4}, & 2.SR_{pred} \leq \frac{SR_{ENC}}{4} \end{cases} \tag{3.4}$$

$$SR_{pred} = max\{SR_{esp}, SR_{n.ant}, SR_{tem}\} \tag{3.5}$$

$$SR_{esp} = max\{r(MVD_{A1}), r(MVD_{B2}), r(MVD_{B0})\} \tag{3.6}$$

$$SR_{n.ant} = r(MVD_{n.ant}) \tag{3.7}$$

$$SR_{tem} = \frac{r(MV_{C0})}{N_{c.ref}} \tag{3.8}$$

Donde

- SR_{esp}, $SR_{n.ant}$ y SR_{tem} es el rango de búsqueda derivado de los candidatos espaciales, nivel anterior y temporal, respectivamente.
- SR_{pred} es el rango de búsqueda derivado a partir de todos los tipos de candidatos
- SR_{ENC} es el rango de búsqueda fijado por el *encoder*
- $N_{c.ref}$ es el número del cuadro de referencia del PU actual

Se elige el presente algoritmo como una referencia para el diseño de la arquitectura de cálculo del rango de búsqueda, en vista de que consigue una reducción de aproximadamente 50 % del tiempo de ejecución y número de puntos evaluados en el proceso ME, en comparación con el software de referencia del HEVC, el cual utiliza el algoritmo *TZSearch*. Además, consigue las mejoras mencionadas con una reducción de PSNR y incremento en *bit-rate* mínimas: 0.004 dB y 0.12 %, respectivamente (Nalluri 2016: 86).

3.5. Arquitectura en HW propuesta

3.5.1. Panorama general y alcance de la arquitectura propuesta

En la figura 3.4 se muestra un diagrama de bloques genérico de la arquitectura en HW propuesta, su alcance y comunicación con un posible entorno (que se pretende desarrollar en el futuro). Como se observa en la figura, las sub-partes consideradas en el presente trabajo son los siguientes: la unidad de interpolación o estimación de MVs, el módulo AMVP y el módulo DSR (que calcula el rango de búsqueda). Se contemplan componentes externos como el módulo ME, las memorias y controladores de las mismas para ilustrar el contexto de las entradas y salidas de la arquitectura a diseñar. Además, se incluyen mecanismos de selección para evitar problemas de dependencia, como el que se especificó para el módulo AMVP con el proceso ME; asimismo, se adicionan mecanismos para la dependencia en las entradas del módulo DSR (especificado en secciones posteriores). A continuación, se realizará una descripción breve de cada uno de los componentes del entorno propuesto:

- **Memoria de MVs de cuadros actuales + controlador:** Se encarga de almacenar los MVs y parámetros adicionales del cuadro actual, que al comunicarse con su controlador, brindarán la información de los candidatos espaciales (A0, A1, B0, B1 y B2) al módulo AMVP. También comunica información de los PUs del nivel anterior al módulo DSR.

Estos parámetros adicionales puede ser datos de disponibilidad y número de cuadro de referencia (se detalla en secciones posteriores).

- **Memoria de MVs de cuadros anteriores + controlador:** Operan de forma similar a los componentes mencionados, con la diferencia de que brindan información de los candidatos temporales (C0, C1) al módulo AMVP y al DSR. Adicionalmente, brindan las versiones co-localizadas de los candidatos A0, B0 y B2 (se precisa en la sección concerniente al módulo DSR).

- **Memoria de MVPs del cuadro actual + controlador:** Es análogo a los anteriores, pero a diferencia de brindar MVs y otros datos; se encarga de proveer de los MVPs de A0, B0, B2 y de los PUs del nivel anterior. Estos componentes se deben utilizar para proveer los MVDs requeridos en el módulo DSR. Asimismo, da información del MVP del LCU a la unidad de estimación de MVs.

- **Grupo de selectores - Sel1:** Se utilizan para reemplazar el candidato espacial que no esté disponible debido a que el proceso ME no terminó de calcular el MV requerido. Este MV no disponible es reemplazado por el MV estimado por la unidad de interpolación de MVs.

- **Grupo de selectores - Sel2:** De forma similar al anterior, se utilizan para sustituir a los MVs no disponibles por la dependencia con la etapa ME (con sus contrapartes co-localizadas en un cuadro anterior), que se deben disponer para calcular los MVDs requeridos por el módulo DSR.

- **Módulo ME:** Determina el MV óptimo o real del PU actual; para ello, recibe el MVP del módulo AMVP (centro de búsqueda) y el rango de búsqueda (SR) del módulo DSR. Puesto que para realizar estas operaciones se requiere de una memoria que guarde el PU actual y otra que guarde la región de búsqueda del mismo (considerando un rango máximo de entrada), se aprovechan dichas memorias para brindar los datos necesarios al módulo AMVP: el PU actual y los dos bloques de referencia (en base a los dos MVs posibles para elegir el MVP). Por otra parte, se encarga de controlar las señales de los grupos de selectores Sel1 y Sel2; dado que, dependiendo del PU actual, el particionado y orden de procesamiento de PUs, se debe determinar las combinaciones de estas señales (para resolver la dependencia particular presentada). También provee datos del tamaño del PU actual a la unidad de estimación de MVs y al módulo AMVP.

- ***Encoder*:** Representa a todos los componentes, herramientas o procesos adicionales posibles del *encoder*, los cuales podrían ejecutarse en un procesador. En este caso, se

encarga de brindar el valor de λ al módulo AMVP y el valor del rango de búsqueda de entrada al módulo DSR. Adicionalmente, recibe el MVD del PU actual para su posterior codificación.

Cabe resaltar que se omiten las interacciones adicionales que requiere el módulo ME para operar, como por ejemplo, cargar el PU actual y rango de búsqueda de una memoria que guarde el recuadro actual, conexiones extras con el *encoder*, etc.

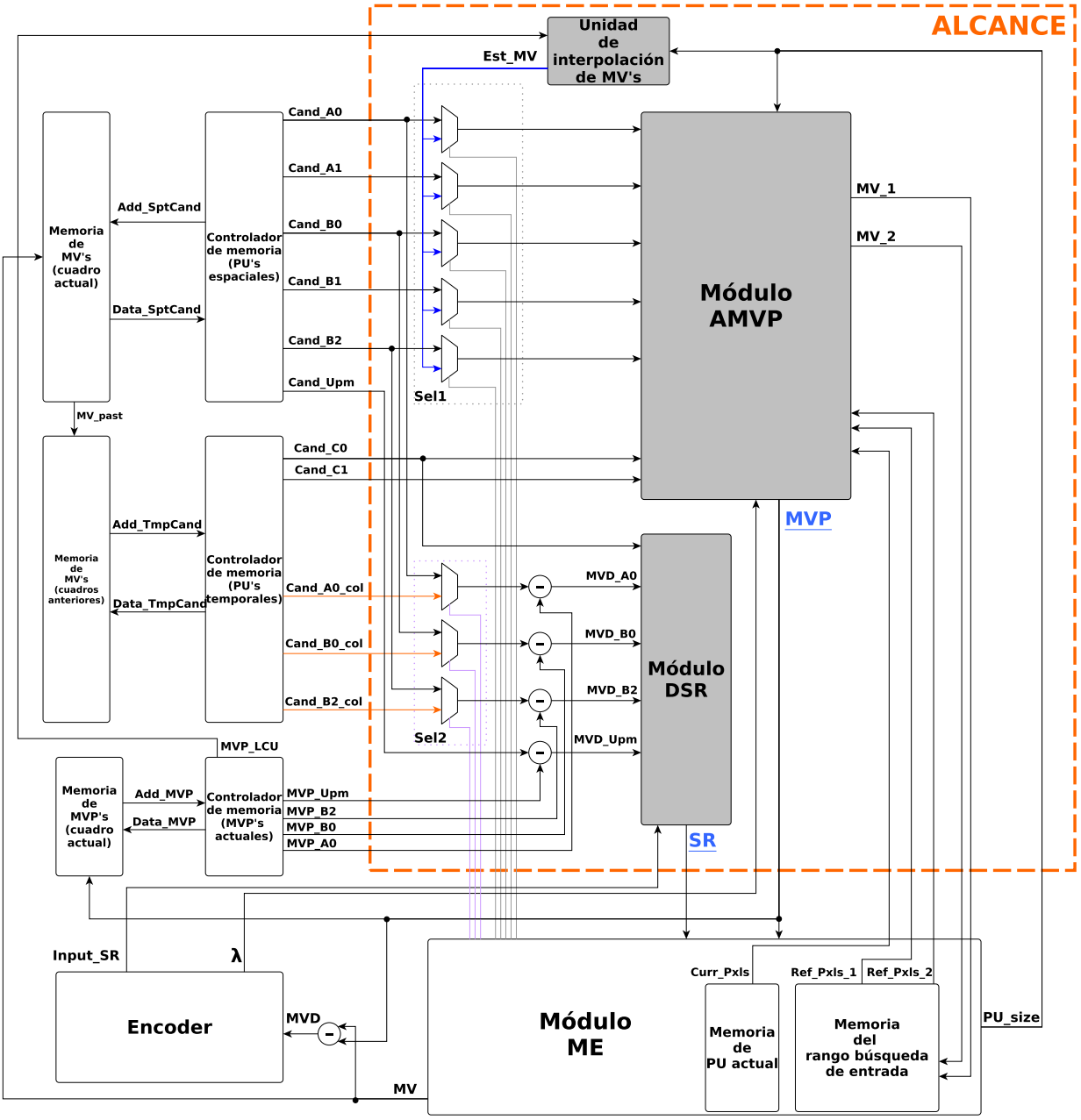


Figura 3.4: Alcance de la arquitectura HW y su entorno propuesto

3.5.2. Unidad de estimación de MVs

La unidad de estimación o interpolación de MVs está basada en la solución propuesta por Wang & Li (2016) para resolver el problema de dependencia descrito en la sección 3.4.2. De forma específica, la unidad se basa en la arquitectura en HW, mostrada en la figura 3.5, que los mismos autores proponen para implementar la interpolación lineal.

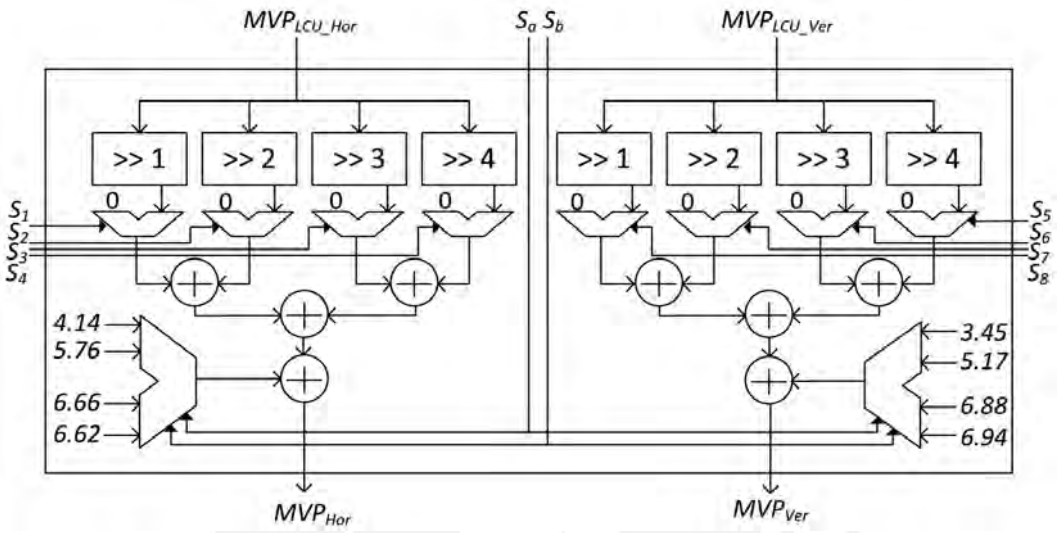


Figura 3.5: Arquitectura del método de interpolación de MVs (Wang & Li 2016: 8)

En la figura 3.6 se muestra el diagrama RTL de la unidad de estimación de MVs diseñada (solo la componente horizontal del MV) y en la tabla 3.1 se detallan las entradas y salidas de la unidad. Se propone utilizar una aritmética de punto fijo en formato Q(7,4) (su precisión es analizada en el capítulo de verificación y resultados) para trabajar con los valores fraccionales del β. Luego, los coeficientes β propuestos por Wang & Li son aproximados de acuerdo a su representación en Q(7,4). Cabe destacar que para la parte entera se usan 7 bits, debido a que se asume un rango de búsqueda máximo de ± 64 (representación en complemento a 2), lo cual es usual para ME.

La unidad de estimación de MVs opera de la siguiente manera:

1. Se determina el valor absoluto del MVP del LCU
2. Se añade un vector con 4 ceros correspondientes a la parte fraccional (en el lado del LSB de la parte entera)
3. Se realizan las operaciones necesarias según lo mostrado en la figura 3.5
4. Se redondea la señal en Q(7,4) a un entero de 7 bits (en complemento a 2). Para ello, de acuerdo al bit más significativo de la parte fraccional, se decide si se debe sumar 1 a la parte entera.

5. Se re-coloca el signo original del MVP del LCU, pues se trabajó con su valor absoluto.

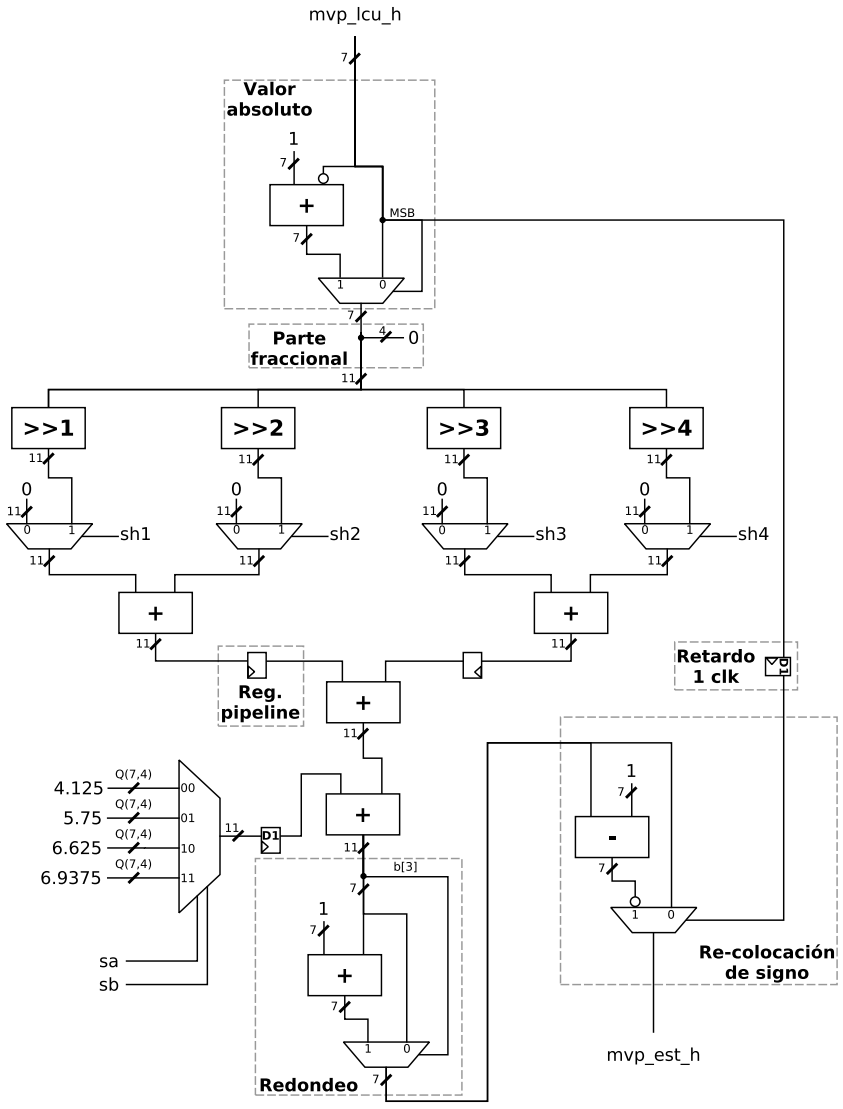


Figura 3.6: Diagrama RTL de la unidad de estimación de MVs - componente horizontal

Tabla 3.1: Entradas y salidas de la unidad de estimación de MVs

Señales	Tipo	Descripción
mvp_lcu_h[6:0]	Entrada	Componente horizontal del MVP del LCU
mvp_lcu_v[6:0]	Entrada	Componente vertical del MVP del LCU
sh1, sh2, sh3, sh4	Entradas	Señales de control para definir α del componente horizontal
sv1, sv2, sv3, sv4	Entradas	Señales de control para definir α del componente vertical
sa, sb	Entradas	Señales de control para definir β para ambos componentes
mvp_est_h[6:0]	Salida	Componente horizontal del MVP estimado
mvp_est_v[6:0]	Salida	Componente vertical del MVP estimado

Por otra parte, para el cálculo de la componente vertical, la arquitectura es idéntica a la mostrada en la figura 3.6, con el cambio de que las señales de control de los multiplexores

posteriores a los desplazadores son sv1, sv2, sv3 y sv4, de izquierda a derecha; y los coeficientes β representados en Q(7,4) son 3.4375, 5.125, 6.875 y 6.9375, de arriba hacia abajo.

3.5.3. Módulo AMVP

A. Unidad de selección de candidatos

Esta unidad se basa en un proceso de selección de dos etapas: primero se seleccionan candidatos únicos por cada tipo (A, B, C); y luego, a partir de estos, se eligen los dos candidatos de la lista considerada en el algoritmo AMVP. El diagrama RTL de la unidad es mostrado en la figura 3.7, las señales de entrada y salida se detallan en la tabla 3.2, y la lógica de selección de los componentes MUX A, B y de salida se precisan en las tabla 3.3, 3.4 y 3.5. Cabe señalar que los componentes horizontales y verticales del MV (mv_A0, mv_B0, etc.) y el número del recuadro de actual y de referencia concatenados (poc_A0, poc_B0, etc.) —referidos también como valores POC del candidado (del inglés *Picture Order Count*)— se agrupan en una señal (cnd_A0, cnd_B0, etc.), por lo que se consideran de 24 bits (7+7+5+5), y que las señales de 'disponibilidad' y 'no-escalado' se consideran en lógica positiva ('1' representa disponible y no escalado).

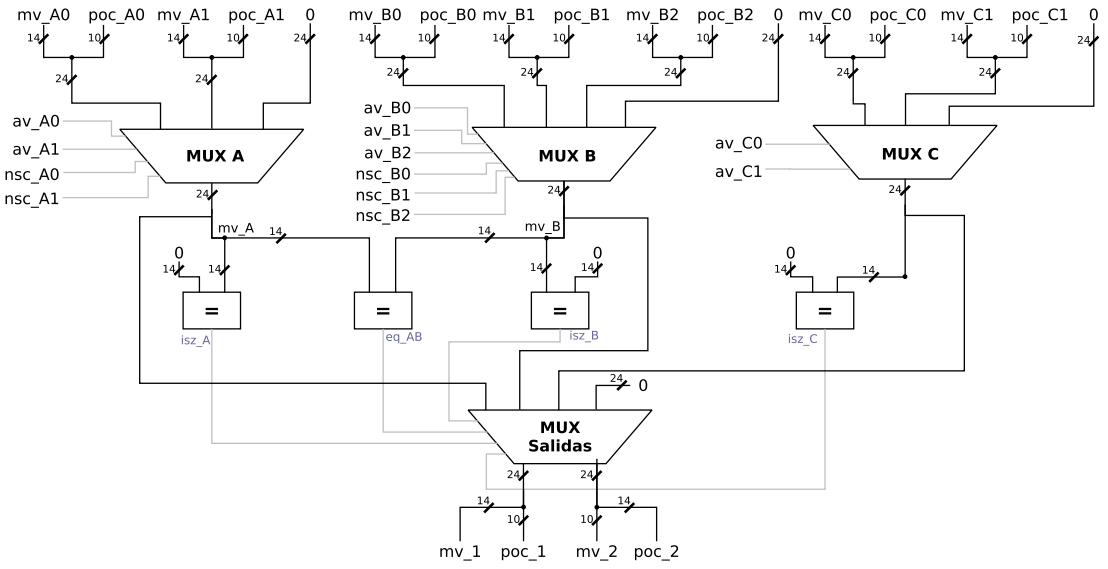


Figura 3.7: Diagrama RTL de la unidad de selección de MVs

B. Unidad de escalamiento

La unidad de escalamiento implementa las ecuaciones mencionadas en la sección 2.2.5, y sirve para escalar MVs de los candidatos temporales, o espaciales que tengan un cuadro de referencia diferente al PU actual. Además, se procede a utilizar la unidad de escalamiento

Tabla 3.2: Entradas y salidas de la unidad de selección de MVs

Señales	Tipo	Descripción
mv_A0[13:0], mv_A1[13:0]	Entradas	MV de los cand. tipo A
mv_B0[13:0], mv_B1[13:0], mv_B2[14:0]	Entradas	MV de los cand. tipo B
mv_C0[13:0], mv_C1[13:0]	Entradas	MV de los cand. tipo C
poc_A0[9:0], poc_A1[9:0]	Entradas	Núm. rec. actual y de ref. cand. A
poc_B0[9:0], poc_B1[9:0], poc_B2[9:0]	Entradas	Núm. rec. actual y de ref. cand. B
poc_C0[9:0], poc_C1[9:0]	Entradas	Núm. rec. actual y de ref. cand. C
av_A0, av_A1	Entradas	Disponibilidad de los cand. A
av_B0, av_B1, av_B2	Entradas	Disponibilidad de los cand. tipo B
av_C0, av_C1	Entradas	Disponibilidad de los cand. tipo C
nsc_A0, nsc_A1	Entradas	MV es no escalado (A)
nsc_B0, nsc_B1, nsc_B2	Entradas	MV es no escalado (B)
mv_1, mv_2	Salidas	MVs seleccionados

Tabla 3.3: Lógica de selección del MUX A

MUX A				
av_A0	**av_A1**	**nsc_A0**	**nsc_A1**	**cnd_A**
0	0	X	X	0
0	1	X	X	cnd_A1
1	0	X	X	cnd_A0
1	1	0	0	cnd_A0
1	1	0	1	cnd_A1
1	1	1	0	cnd_A0
1	1	1	1	cnd_A0
0	0	X	X	0

para candidatos espaciales con el mismo cuadro de referencia, dado que mantiene un factor de escalamiento unitario para dicho caso ($td = tb$). El diagrama RTL correspondiente al escalamiento de un MV es ilustrado en la figura 3.8, y las entradas y salidas en la tabla 3.8. Se procede a operar los valores absolutos de los MVs y se re-colocan los signos correspondientes (de forma análoga a la unidad de estimación de MVs). Las operaciones de división y multiplicación se implementan usando los *IP cores* disponibles en el catálogo de IPs de la herramienta Vivado; en las tablas 3.6 y 3.7 se muestran los parámetros más relevantes que definen las configuraciones escogidas para ambos IPs. Para el divisor, se ha empleado el algoritmo LUTMult, debido a que provee menos latencia, la cantidad de bits límite de los operandos cumplen con los requeridos y soporta un resultado por ciclo de reloj; y un protocolo de tipo *non blocking*, puesto que este modo no hace uso del protocolo *AXI-Stream* (se requeriría añadir lógica) y no bloquea la salida con señales de control adicionales (Xilinx 2016: 13-27). En cuanto al *core* multiplicador, se utilizó el tipo paralelo debido a que ambos operandos son variables, una construcción de tipo *Mults* para priorizar el uso de DSPs, una optimización enfocada en el desempeño (especificación más relevante), y 5 etapas de *pipeline* (Xilinx 2015: 9-12).

Tabla 3.4: Lógica de selección del MUX B

MUX B						
av_B0	**av_B1**	**av_B2**	**nsc_B0**	**nsc_B1**	**nsc_B2**	**cnd_B**
0	0	0	X	X	X	0
0	0	1	X	X	X	cnd_B2
0	1	0	X	X	X	cnd_B1
1	0	0	X	X	X	cnd_B0
0	1	1	X	0	0	cnd_B1
0	1	1	X	0	1	cnd_B2
0	1	1	X	1	0	cnd_B1
0	1	1	X	1	1	cnd_B1
1	0	1	0	X	0	cnd_B0
1	0	1	0	X	1	cnd_B2
1	0	1	1	X	0	cnd_B0
1	0	1	1	X	1	cnd_B0
1	1	0	0	0	X	cnd_B0
1	1	0	0	1	X	cnd_B1
1	1	0	1	0	X	cnd_B0
1	1	0	1	1	X	cnd_B0
1	1	1	0	0	0	cnd_B0
1	1	1	0	0	1	cnd_B2
1	1	1	0	1	X	cnd_B1
1	1	1	1	X	X	cnd_B0

Tabla 3.5: Lógica de selección del MUX de salida

MUX Salida					
isz_A	**isz_B**	**eq_AB**	**isz_C**	**cnd_1**	**cnd_2**
0	0	1	0	cnd_A	cnd_C
0	0	0	0	cnd_A	cnd_B
0	1	X	0	cnd_A	cnd_C
1	0	X	0	cnd_B	cnd_C
1	1	X	0	cnd_C	0
0	0	1	1	cnd_A	0
0	0	0	1	cnd_A	cnd_B
0	1	X	1	cnd_A	0
1	0	X	1	cnd_B	0
1	1	X	1	0	0

Tabla 3.6: Parámetros de configuración relevantes del *core* divisor

***Core* divisor (v5.1)**	
Tipo de algoritmo	LUTMult
Tipo de operandos	*Unsigned*
Protocolo	*Non blocking*
Latencia	8 ciclos

C. Unidad de cálculo del costo en bits

La arquitectura correspondiente al cálculo del costo en bits implementa las ecuaciones 3.9 y 3.10 —mv_x es el componente horizontal o vertical del MV— deducidas por Nalluri a partir del software de referencia (2016: 131-132). El diagrama RTL de la unidad mencionada se

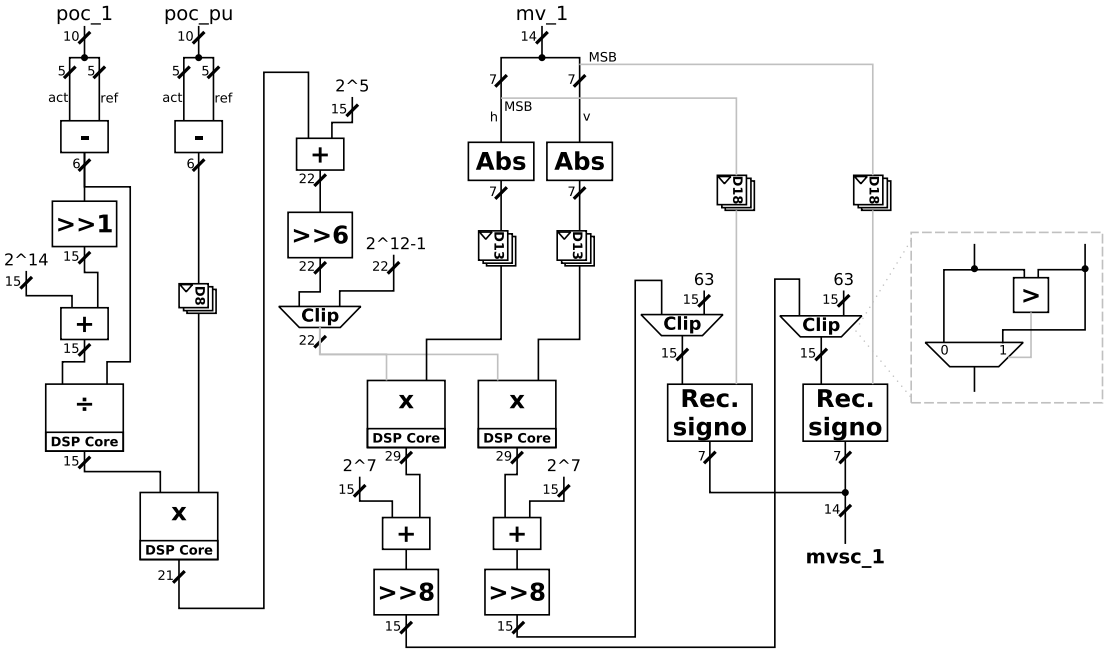


Figura 3.8: Diagrama RTL de la unidad de escalamiento (solo 1 MV)

Tabla 3.7: Parámetros de configuración relevantes del *core* multiplicador

***Core* multiplicador (v12.0)**	
Tipo	Paralelo
Tipo de operandos	*Unsigned*
Contrucción	Uso de *Mults*
Optimización	Desempeño
Pipeline	5 etapas

Tabla 3.8: Entradas y salidas de la unidad de escalamiento

Señales	**Tipo**	**Descripción**
mv_1 [13:0], mv_2[13:0]	Entradas	MVs escogidos a escalar
poc_1[9:0], poc_2[9:0], poc_pu[9:0]	Entradas	Núm. de recuadro MVs escogidos
poc_pu[9:0]	Entrada	Núm. recuadro del PU actual
clk, rst	Entradas	Señal de reloj y *reset*
mvsc_1 [13:0], mvsc_2[13:0]	Salidas	MVs escalados

ilustra en la figura 3.9 (para el costo de 1 MV). Las señales de entrada y salida son detalladas en la tabla 3.9. Es pertinente señalar que la operación de redondeo hacia abajo del logaritmo en base dos ($floor(log_2(n))$) es realizado con un *enconder*, el cual tiene como tabla de verdad lo especificado en 3.10.

$$costo_{bits}(mv_x) = \begin{cases} 2.floor(log_2(1 - 2mv_x)) + 1, & mv_x \leq 0 \\ 2.floor(log_2(2mv_x)) + 1, & mv_x > 0 \end{cases} \tag{3.9}$$

$$costo_{bits}(total) = costo_{bits}(mv_h) + costo_{bits}(mv_v) \tag{3.10}$$

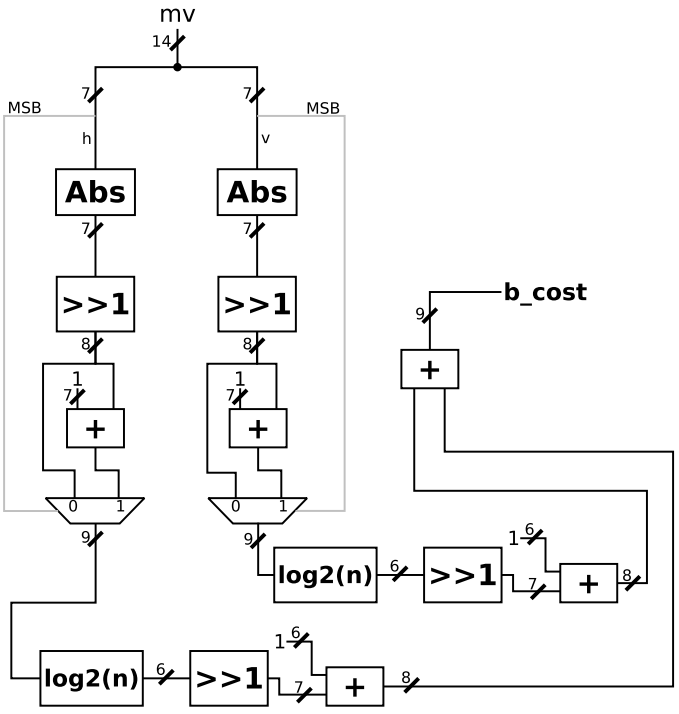


Figura 3.9: Diagrama RTL de la unidad de cálculo de costo en bits (solo 1 MV)

Tabla 3.9: Entradas y salidas de la unidad de cálculo de costo en bits

Señales	Tipo	Descripción
mv[13:0]	Entrada	MV como argumento
bcost[8:0]	Salida	Costo en bits del MV

Tabla 3.10: Tabla de verdad operación $floor(log_2(n))$

Entrada (binario)	Salida (decimal)
000000001x	1
00000001xx	2
0000001xxx	3
000001xxxx	4
00001xxxxx	5
0001xxxxxx	6
01xxxxxxx	7
1xxxxxxxx	8

D. Unidad de cálculo de distorsión

Esta unidad está compuesta por una unidad WSAD 4x4 (16 pixeles) y una unidad acumuladora de valores de distorsión parcial, mostrados en las figuras 3.10 y 3.11. Asimismo, se ilustran las señales de entrada y salida de ambas unidades en las tablas 3.11 y

3.12. Se debe precisar que se coloca la cuenta actual y el fin de contador (retardado 1 ciclo) como salidas, dado que estas se emplean en el control de la arquitectura integrada (detallado en secciones posteriores). La unidad WSAD4x4 está constituida por una lógica de diferencia absoluta basada en la propuesta optimizada para FPGA presentada por Kumm et al. (2016: 3) —ilustrada en la figura 3.12— y un arbol binario de WSADs de dos elementos separados por registros que definen etapas de *pipeline*.

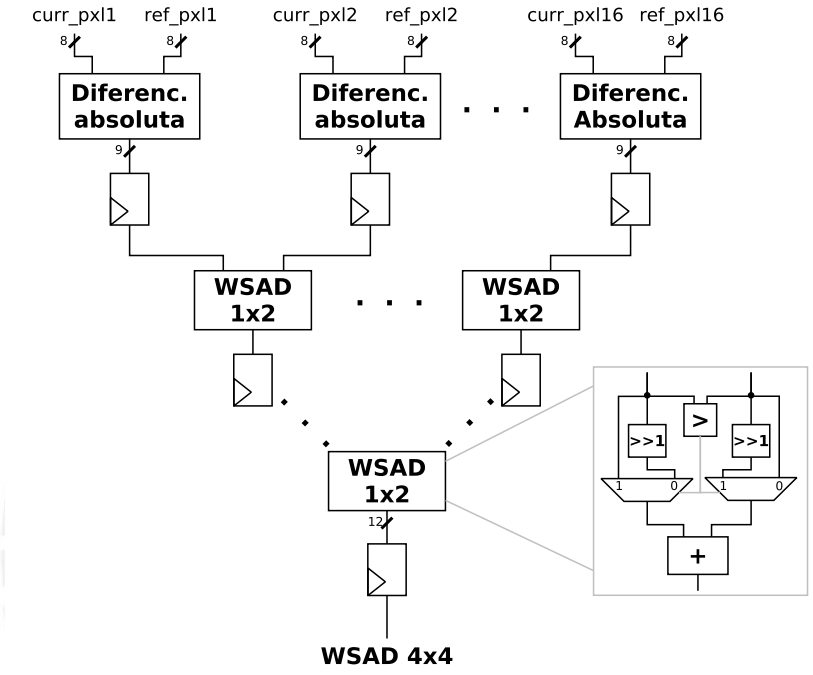


Figura 3.10: Diagrama RTL de la unidad de WSAD4x4

Tabla 3.11: Entradas y salidas de la unidad WSAD4x4

Señales	**Tipo**	**Descripción**
curr_pxls[127:0]	Entrada	16 pixeles del PU actual
ref_pxls[127:0]	Entrada	16 pixeles del bloque de ref.
clk, rst	Entradas	Señal de reloj y *reset*
en_stg1, en_stg2, en_stg3, en_stg4	Entradas	Habilitación de cada etapa
wsad4x4_out[12:0]	Salida	Salida WSAD4x4

Tabla 3.12: Entradas y salidas del acumulador

Señales	**Tipo**	**Descripción**
pdst[12:0]	Entrada	Distorsión parcial
nacc[8:0]	Entrada	Cantidad de dist. parciales
clk, rst	Entradas	Señal de reloj y *reset*
en_cnt	Entradas	Señal de habilit. contador
tdst[21:0]	Salida	Distorsión total
cnt[8:0]	Salida	Cuenta actual contador
done_ret	Salida	Fin contador retardado

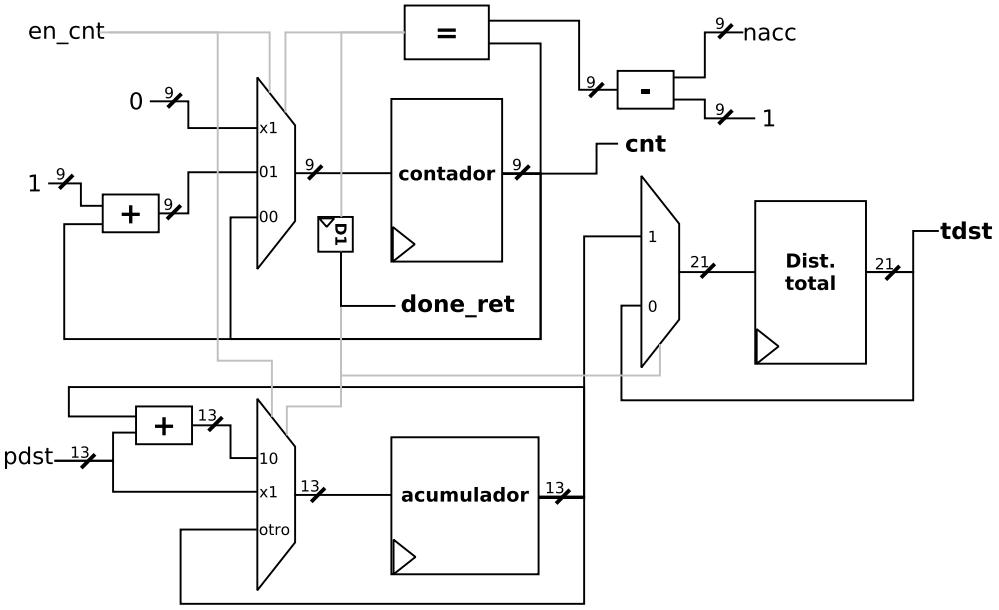


Figura 3.11: Diagrama RTL del acumulador

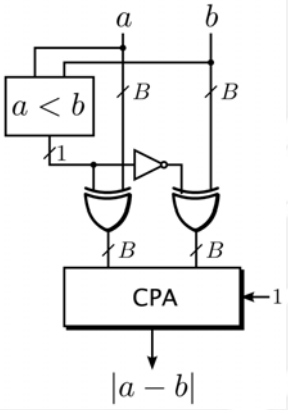


Figura 3.12: Circuito de diferencia absoluta optimizado para FPGA (Kumm et al. 2016: 3)

E. Unidad de comparación de costo RD

Esta unidad se encarga de calcular el costo RD de cada MV candidato y escoger el MV de menor costo; su diagrama RTL es presentado en la figura 3.13. Para realizar la multiplicación de λ con el costo en bits, se utiliza un *IP core* multiplicador, con una configuración similar al usado en la unidad de escalamiento. Por su parte, se describen las entradas y salidas de la unidad en la tabla 3.13. También, es pertinente destacar que se considera la señal 'en_sel_ret' como salida, ya que se utiliza para el control.

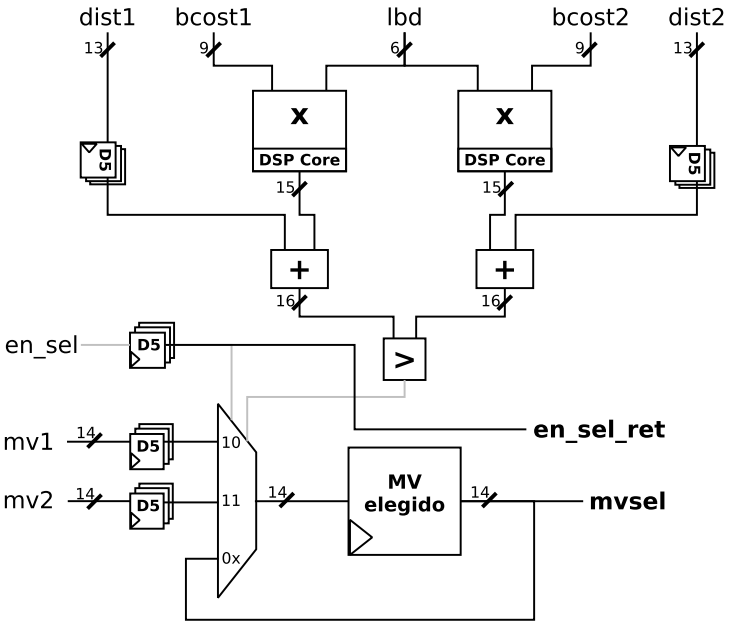


Figura 3.13: Diagrama RTL de la unidad de comparación de costo RD

Tabla 3.13: Entradas y salidas de la unidad de comparación de costo RD

Señales	Tipo	Descripción
mv1[13:0], mv2[13:0]	Entrada	MVs candidatos
dist1[12:0], dist2[12:0]	Entrada	Distorsión de candidatos
bcost1[8:0], bcost2[8:0]	Entradas	Costo en bits de candidatos
lbd[5:0]	Entrada	Parámetro λ
clk, rst	Entradas	Señal de reloj y *reset*
en_sel	Entradas	Habilitación de selección
mvsel[13:0]	Salida	MV escogido
en_sel_ret	Salida	Habilit. selección retardada

3.5.4. Módulo DSR

Se muestra el diagrama RTL del módulo DSR en la figura 3.14. El módulo mencionado parte calculando los MVDs de los candidatos espaciales y de los de nivel anterior realizando una resta en complemento a dos. Luego, sigue una lógica de valor absoluto similar a la usada en la unidad de estimación de MVs. De acuerdo a la dependencia y disponibilidad de los MVDs actuales y co-localizados, se selecciona la entrada de una unidad de WSAD1x2 (dos elementos), el cual se utiliza para estimar la distancia euclidiana; se procede de forma análoga para los candidatos de nivel anterior. Por otro lado, para el candidato temporal, un *IP core* de división es utilizado, con una configuración similar al presentado en la unidad de escalamiento. Debido a la latencia del *core* divisor, se añaden retardos en las distancias espacial y de nivel anterior (8 ciclos de reloj). Finalmente, se selecciona el máximo de las distancias obtenidas, y se realiza una operación de

recorte (en concordancia con lo indicado en la sección 3.4.4). Adicionalmente, se detallan las señales de entrada y salida del módulo en la tabla 3.14.

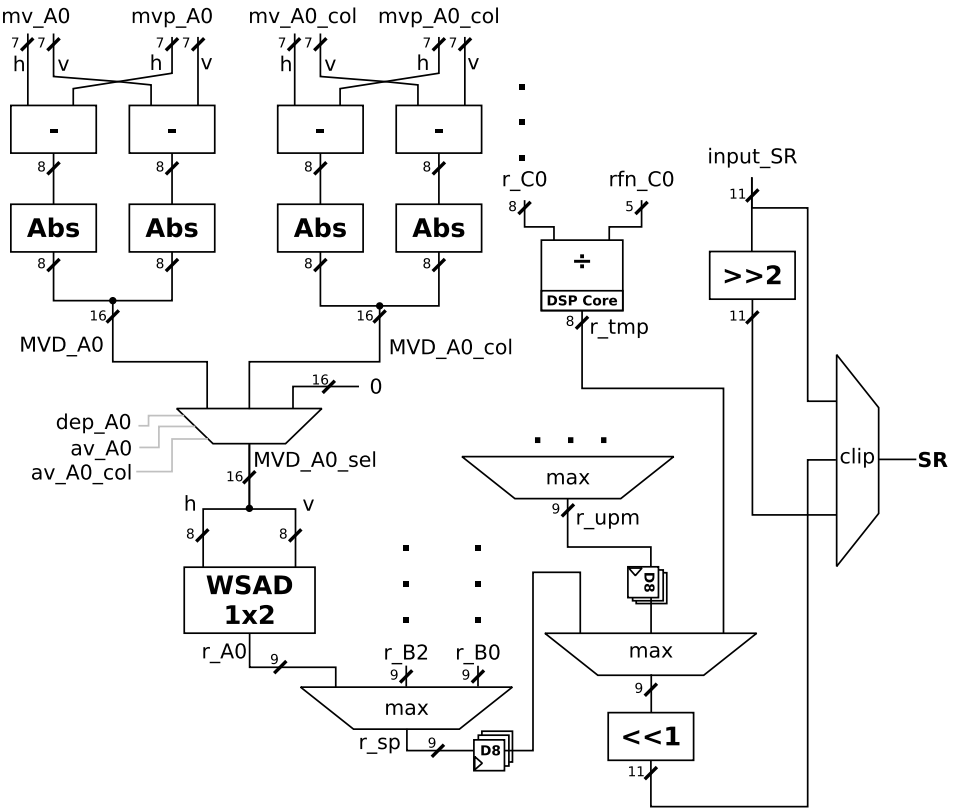


Figura 3.14: Diagrama RTL del módulo DSR

Tabla 3.14: Entradas y salidas del módulo DSR

Señales	Tipo	Descripción
mv_A0[13:0], ..B2, ..B0	Entradas	MVs candidatos espaciales
mv_A0_col[13:0], ..B2, ..B0	Entradas	MVs candidados espaciales co-local.
mv_upm_2nx2n[13:0], ..2nxn, ..nx2n	Entradas	MVs candidatos del nivel anterior
mpv_A0[13:0], ..B2, ..B0	Entradas	MVPs candidatos espaciales
mvp_upm_2nx2n[13:0], ..2nxn, ..nx2n,	Entradas	MVPs candidatos nivel anterior
mv_C0[13:0]	Entrada	MV candidato temporal
av_A0, av_B2, av_B0	Entradas	Disponibilidad MVs cand. esp.
av_A0_col, av_B2_col, av_B0_col	Entradas	Disponibilidad MVs cand. esp. co-local.
av_C0	Entrada	Disponibilidad MV temp.
sq_PU	Entrada	Es PU cuadrado
dep_A0, dep_B2, dep_B0	Entradas	Dependencia MVs candidatos espaciales
rfn_C0[4:0]	Entrada	Num. rec. de ref. candidato temporal
input_sr[10:0]	Entrada	Rango de búsqueda de entrada (máx.)
clk, rst	Entradas	Señal de reloj y *reset*
sr[10:0]	Salida	Rango de búsqueda escogido

3.5.5. Arquitectura integrada

Se presenta la arquitectura integrada en la figura 3.15, el cual incluye elementos adicionales para la interconexión y sincronización de las señales de las unidades descritas en la sección anterior. Se colocan registros a la salida de cada unidad —sobre todo de las que son enteramente combinacionales— para evitar el incremento del *delay* máximo (afecta a la frecuencia máxima). Para corregir el desfase de ciertos datos debido a las latencias (producto de etapas de *pipeline*), se hace uso de elementos de retardo constituidos por cadenas de *flip-flops*. No obstante, para sincronizar las señales que no se pueden alinear con un retardo fijo —producto de la latencia variable de la unidad de cálculo de distorsión— se utilizan FIFOs (del inglés, *First In First Out*), el cual se comporta como una pila con escrituras y lecturas controlables mediante señales de habilitación. Finalmente, se tiene un controlador/sincronizador que controla el pedido de dato de candidatos y datos de pixeles de PUs a las memorias correspondientes, controla las señales de habilitación de lectura/escritura de FIFOs, de la etapa WSAD4x4, del contador presente en el acumulador, del comparador de costo RD, entre otros (detallado posteriormente); adicionalmente, el controlador recibe algunas señales de las unidades de distorsión y costo RD para emplearlos en su lógica de control/sincronización.

Es pertinente resaltar que los FIFOs utilizados son *IP cores* disponibles en el catálogo de la herramienta Vivado. La configuración de los mismos se puede observar en la tabla 3.15. Se hace uso de una interfaz nativa del FIFO para evitar adicionar lógica de comunicación *AXI-Stream*, de una implementación basada en registros de desplazamiento con reloj común ya que la arquitectura posee un reloj único, y de una profundidad de 64 debido a que son pocos datos a almacenar —solo se utiliza para sincronizar las transiciones en el procesamiento PUs subsiguientes (Xilinx 2017: 5-16).

Tabla 3.15: Parámetros de configuración relevantes del *core* FIFO

***Core* FIFO (v13.1)**	
Interfaz	Nativo
Implementación	Reg. de desplazam. con reloj común
Modo de lectura	Estándar
Profundidad	64

A. Controlador/Sincronizador

El diagrama RTL del controlador/sincronizador es mostrado en la figura 3.16, el cual consta de contadores con tope fijo (*delay*) para contabilizar retardos definidos (latencias), contadores de tope variable y lógica adicional que define todas las señales de control requeridas. Cabe destacar que la lógica de temporización fue deducida en base a un modelo

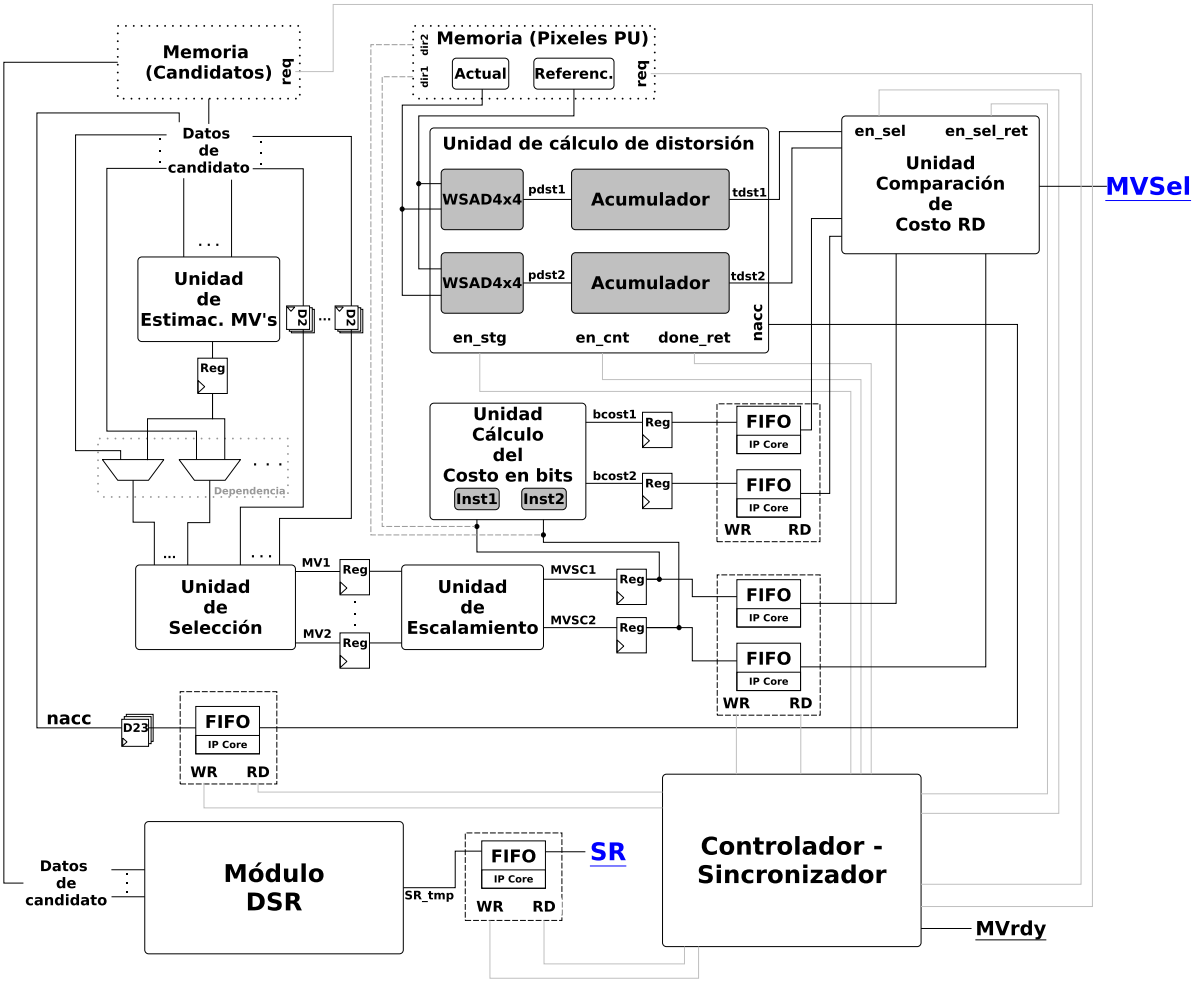


Figura 3.15: Diagrama general de la arquitectura integrada

realizado en SW (mayor detalle en el capítulo de verificación). Luego, se muestran las señales de entrada y salida de la unidad en la tabla 3.16.

B. Operación de la arquitectura

La arquitectura integrada opera, en términos generales, cargando datos de candidatos de la memoria de candidatos, manteniéndolo una cantidad de ciclos igual a 'nacc' (cantidad de distorsiones parciales); y luego de una determinada latencia fija, comienza a cargar datos de pixeles de los PUs correspondientes, en base a los dos MVs escogidos y escalados. Un diagrama de tiempos ejemplo se ilustra en la figura 3.17, en el cual se muestran las señales de petición de datos a memoria, habilitación de contadores, aviso de resultado disponible y algunas señales de lectura/escritura de FIFOs. Se puede observar que se cargan oportunamente al FIFO los rangos de búsqueda obtenidos al transcurrir una latencia de 8 ciclos de reloj; se comienza a pedir datos de los pixeles de PUs al pasar una latencia de 22 ciclos, se coloca la petición y el vector (que utilizaría para deducir la dirección de

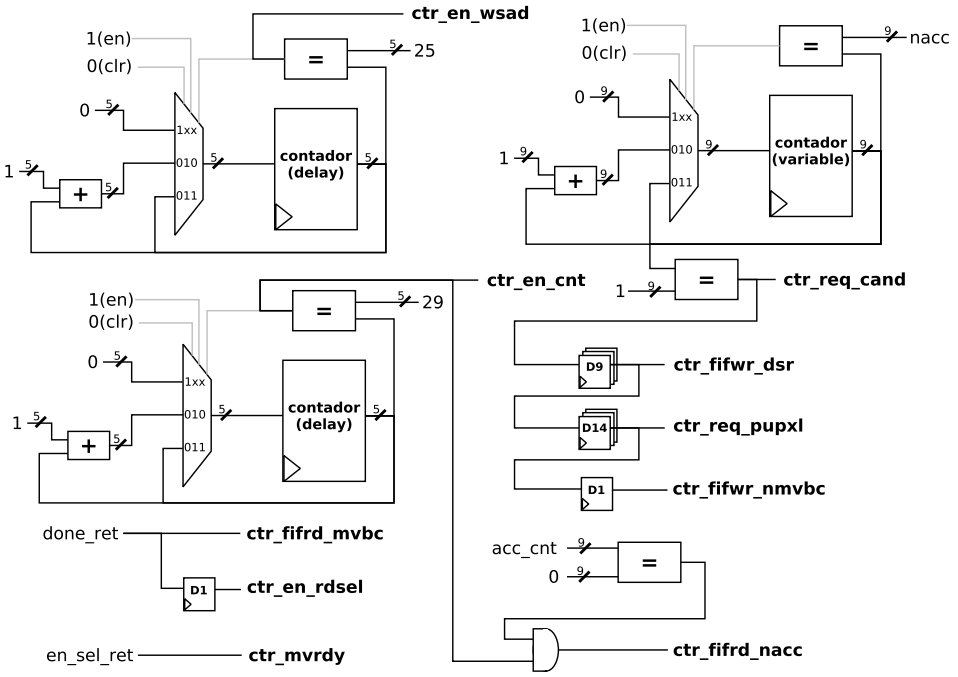


Figura 3.16: Diagrama RTL del controlador/sincronizador

Tabla 3.16: Entradas y salidas del controlador/sincronizador

Señales	Tipo	Descripción
done_ret	Entrada	Fin de contador retardado de la unidad acumul.
en_sel_ret	Entrada	Habilit. de selec. del comp. RD retardado
acc_cnt[8:0]	Entrada	Cuenta del acumulador
nacc[8:0]	Entrada	Num. dist. parc. del acumulador
clk, rst	Entradas	Señal de reloj y *reset*
ctr_en_wsad	Salida	Habilitador del WSAD4x4
ctr_en_cnt	Salida	Habilitador de cuenta del acumulador
ctr_en_rdsel	Salida	Habilitador de la unidad de comparación RD
ctr_fifrd_mvbc	Salida	Lectura de FIFOs de mv escal. y costo en bits
ctr_fifwr_nmvbc	Salida	Escritura de FIFOs de mv escal., c. bits y de 'nacc'
ctr_fifrd_nacc	Salida	Lectura de FIFOs de 'nacc'
ctr_fifwr_dsr	Salida	Escritura de FIFOs del rango de búsqueda
ctr_req_cand	Salida	Petición de datos de candidato
ctr_req_pupxl	Salida	Petición de pixeles del PU
ctr_mvrdy	Salida	Aviso de MV selecionado listo y lect. FIFO de SR

memoria) un ciclo antes al requerido. Se procede a guardar los datos de costo en bits y MV escalado en sus respectivas FIFOs; y luego de una latencia variable (debido al acumulador) se perciben las salidas de MV y rango de búsqueda seleccionados en compañia de una señal de aviso, que se activa un ciclo antes (como un protocolo *strobe* con una espera de 1 ciclo). Es pertinente resaltar que algunas señales de lectura/escritura de FIFOs (como el del contador en el acumulador) y de habilitación se han omitido por simplicidad.

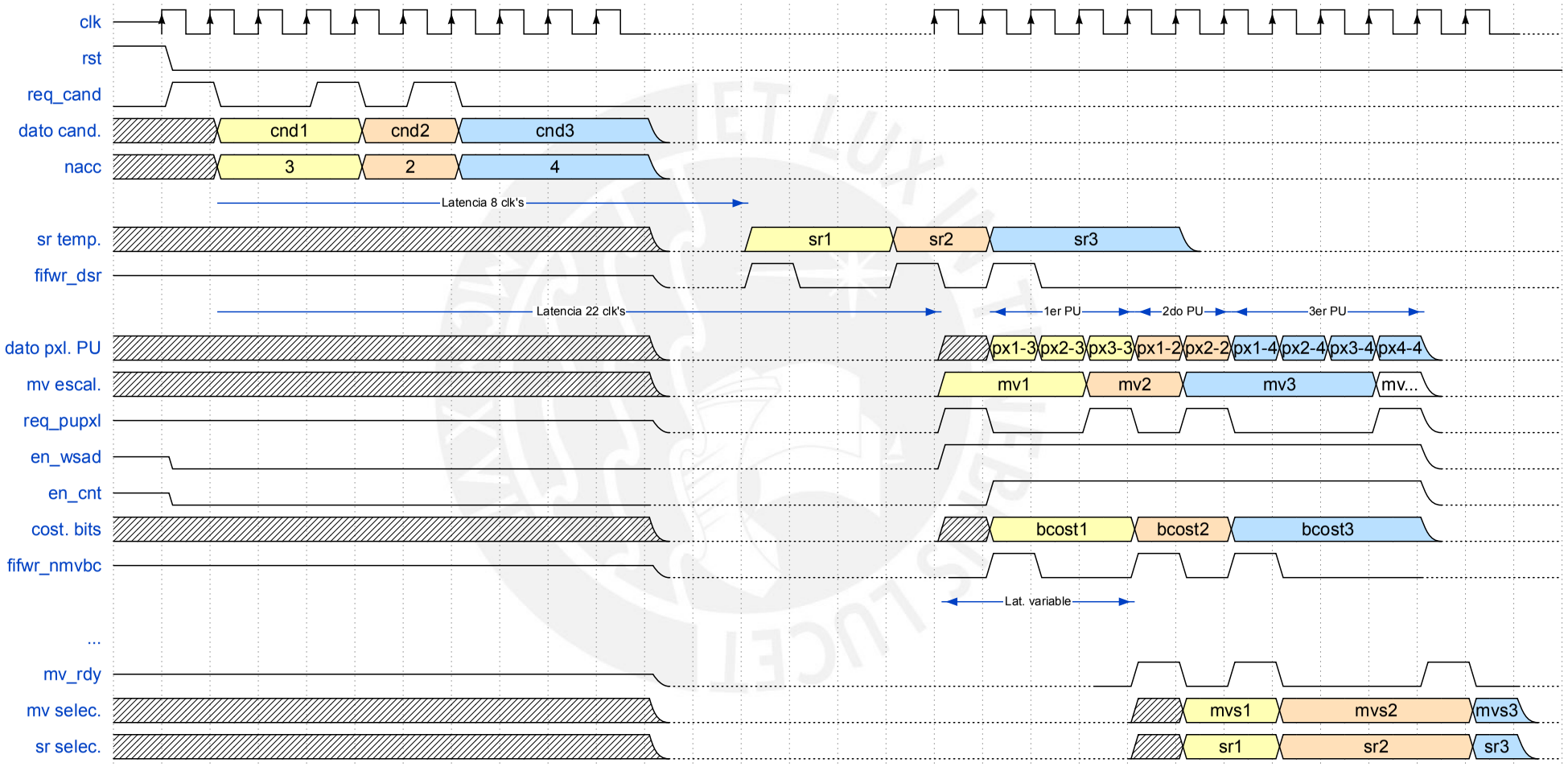


Figura 3.17: Diagrama de tiempos ejemplo de la arquitectura integrada

Capítulo 4

Verificación y resultados

4.1. Verificación de la arquitectura propuesta

4.1.1. Verificaciones con modelo en SW

Para este caso se ha codificado en Verilog un modelo en SW de la arquitectura en prueba, con el objetivo de contrastarlo con una referencia. El ambiente de verificación se muestra en la figura 4.1, en el cual se dispone de un revisor que compara las salidas obtenidas del modelo en SW y del HW en prueba —referido como DUV (*Design Under Verification*, por sus siglas en inglés)— ante estímulos aleatorios. Este revisor, finalmente, genera un reporte de los juegos de entrada probados, las salidas obtenidas e indica si las salida esperada (del modelo en SW) y obtenida (del DUV) coincidieron.

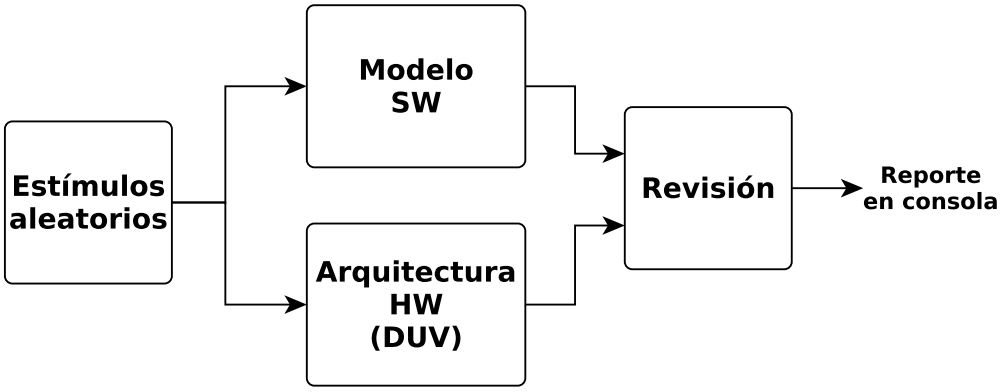


Figura 4.1: Diagrama de bloques del ambiente de verificación con modelo en SW

A. Unidad de estimación de MVs

Para comprobar la precisión del formato Q(7,4) propuesto, se procedió a codificar un

modelo en SW de la arquitectura que realice los cálculos correspondientes con números reales (tipo de dato *real* en Verilog), para luego comprobar si el valor redondeado coincide con el obtenido. En el cuadro 4.1 se ilustra el formato del reporte, el cual muestra el tamaño del PU generado, el valor real obtenido y los valores redondeados (en caso coincidan). Se verificó con más de 50 casos que el formato Q(7,4) provee resultados iguales al modelo en SW.

Cuadro 4.1: Formato de reporte de verificación (unidad de estimación de MVs)

```
...
Tamano de PU: 16x16, 16x8 o 8x16
    Real => (45.847500, 21.880000)
    (mvp_lcu_h, mvp_lcu_v) => (   57,   20)
    (mvp_est_h, mvp_est_v) => (   46,   22)
-------------------------------------------
Tamano de PU: 8x8, 8x4 o 4x8
    Real => (23.807500, 16.690000)
    (mvp_lcu_h, mvp_lcu_v) => (   25,  -13)
    (mvp_est_h, mvp_est_v) => (   24,  -17)
-------------------------------------------
Tamano de PU: 32x32, 16x32 o 32x16
    Real => (42.885000, 45.670000)
    (mvp_lcu_h, mvp_lcu_v) => (  -54,   54)
    (mvp_est_h, mvp_est_v) => (  -43,   46)
-------------------------------------------
Verificacion exitosa
```

B. Unidad WSAD4x4

De forma similar, se ha procedido a generar pixeles aleatorios actuales y de referencia, y se ha codificado un modelo en SW para verificar automáticamente varios juegos de entrada. El formato del reporte se muestra en el cuadro 4.2.

Cuadro 4.2: Formato de reporte de verificación (unidad WSAD4x4)

```
...
---------------Prueba N5---------------
Pixeles actuales:       |  58 |  83 | 165 |  84 |  86 | 131
                        |  79 | 153 | 125 | 216 | 107 | 244
                        |  59 | 109 |  15 | 133
Pixeles de referencia:  |  19 |   1 | 241 | 136 | 126 | 207
```

```
                        | 174 | 238 |  90 | 180 |  86 | 228
                        |  11 |  22 | 167 |  61
        Correcto!
--------------Prueba N6---------------
Pixeles actuales:       | 184 |  91 | 146 |  88 | 161 | 190
                        | 232 | 176 |  75 | 226 | 114 | 102
                        |   5 |  90 | 143 |  67
Pixeles de referencia:  |  39 |  35 | 123 |  57 | 211 | 194
                        | 135 |  59 |  64 | 123 | 253 | 242
                        | 216 | 185 | 130 | 183
        Correcto!
Verificacion exitosa
```

4.1.2. Verificaciones sin modelo en SW

Se utilizó este estilo de verificación para las unidades que poseen un juego de entradas complejo e implementan una operación fácilmente determinable; en su mayoría las unidades puramente combinacionales. En la figura 4.2 se describe el proceso seguido en la verificación.

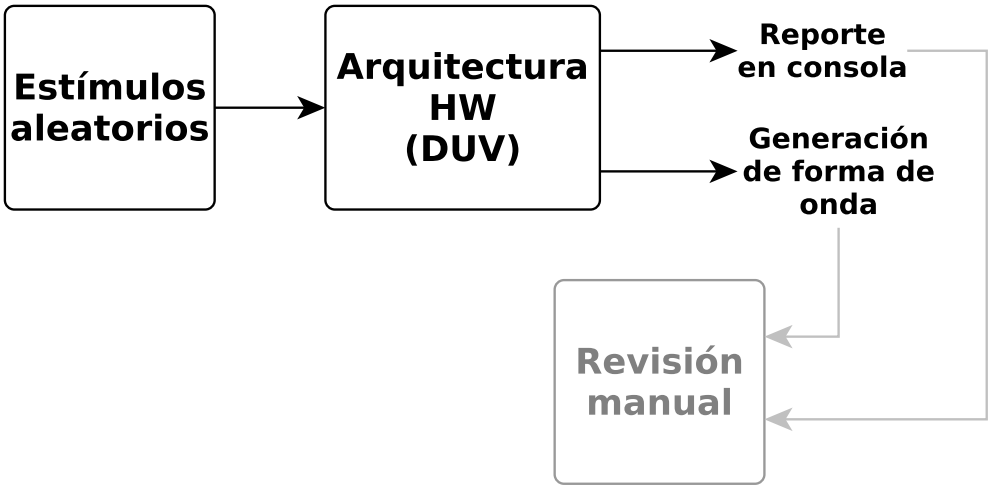


Figura 4.2: Diagrama de bloques del proceso de verificación sin modelo en SW

A. Unidad de selección de candidatos

Para esta unidad, se generaron estímulos aleatorios para las señales de disponibilidad y no-escalamiento, y estímulos fijos para los MVs de entrada. Para facilitar la visualización, se colocaron nombres en ASCII a los MVs y valores de POC fijos. Una porción del formato en consola se puede visualizar en el cuadro 4.3.

Cuadro 4.3: Formato de reporte de verificación (unidad de selección de candidatos)

```
...
-------------------------------------------------------
A0: No escalado
A1: No disponible
B0: No disponible
B1: No escalado
B2: No disponible
C0: Disponible
C1: Disponible
mv_1 => A0
mv_2 => B1
poc_1 =>    1
poc_2 =>    4
-------------------------------------------------------
A0: No escalado
A1: Escalado
B0: No escalado
B1: Escalado
B2: No disponible
C0: No disponible
C1: No disponible
mv_1 => A0
mv_2 => B0
poc_1 =>    1
poc_2 =>    3
-------------------------------------------------------
...
```

B. Unidad de escalamiento

En este caso, se generaron estímulos aleatorios de los MVs y POCs de forma inicial; y luego de un periodo de tiempo (debido a latencia del circuito), se revisaron las salidas (con impresiones en consola y en la forma de onda). En el cuadro 4.4 se puede observar una porción de los reportes.

Cuadro 4.4: Formato de reporte de verificación (unidad de escalamiento)

```
-----------------------Entradas-------------------------
```

```
(mv_1(h), mv_1(v)) => (   1,  -35)
(mv_2(h), mv_2(v)) => ( -13,  -18)
(poc_1(act), poc_1(ref)) => (16, 11)
(poc_2(act), poc_2(ref)) => (21, 18)
(poc_pu(act), poc_pu(ref)) => ( 7,  4)
-------------------------------------------------------
(mv_1(h), mv_1(v)) => (  -6,  -42)
(mv_2(h), mv_2(v)) => ( -55,   15)
(poc_1(act), poc_1(ref)) => (12, 11)
(poc_2(act), poc_2(ref)) => (22, 18)
(poc_pu(act), poc_pu(ref)) => (13,  9)
-------------------------------------------------------
...
-----------------------Salidas--------------------------
-------------------------------------------------------
(mv_sc_1(h), mv_sc_1(v)) => (   1,  -21)
(mv_sc_2(h), mv_sc_2(v)) => ( -13,  -18)
-------------------------------------------------------
(mv_sc_1(h), mv_sc_1(v)) => ( -24,  -63)
(mv_sc_2(h), mv_sc_2(v)) => ( -55,   15)
-------------------------------------------------------
...
```

C. Unidad de cálculo de costo en bits

Para esta unidad, se generaron MVs aleatorios y se revisaron manualmente los costos, de acuerdo a la ecuación que implementa de forma directa. El formato de reportes puede ser visto en el cuadro 4.5.

Cuadro 4.5: Formato de reporte de verificación (unidad de cálculo de costo en bits)

```
(mv(h), mv(v)) => (   1,  -35)
bit cost =>  16
-------------------------------------------------------
(mv(h), mv(v)) => ( -13,  -18)
bit cost =>  20
-------------------------------------------------------
...
```

D. **Unidad de comparación de costo RD**

La verificación de esta unidad es análoga a la realizada en la unidad de escalamiento, debido a que hace uso de un *IP core* multiplicador que le supone 5 ciclos de latencia. El formato del reporte es mostrado en el cuadro 4.6.

Cuadro 4.6: Formato de reporte de verificación (unidad de comparación de costo RD)

```
-----------------------------------------------
(mv1(h), mv1(v)) => (  -9,  -13)
(mv2(h), mv2(v)) => ( -37,    1)
(dist1, dist2) => (3341, 4470)
(bcost1, bcost2) => (317, 493)
-----------------------------------------------
(mv1(h), mv1(v)) => (  57,    5)
(mv2(h), mv2(v)) => (  37,  -18)
(dist1, dist2) => (7055, 2546)
(bcost1, bcost2) => (206, 232)
-----------------------------------------------
...
MV seleccionado: (  -9,  -13)
MV seleccionado: (  37,  -18)
...
```

E. **Módulo DSR**

De forma análoga al anterior, se usaron estímulos aleatorios y se revisaron las salidas con un determinado tiempo transcurrido (latencia). El cuadro 4.7 muestra el formato de reporte utilizado.

Cuadro 4.7: Formato de reporte de verificación (módulo DSR)

```
------------------------Entradas--------------------------
----------------------------------------------------------
Av A0|B2|B0|A0col|B2col|B0col|C0 => 0|1|0|0|1|0|0
Sq_PU => 1
Dep A0|B2|B0 => 0|0|1
rfn_C0 =>  3
         mv_A0:  => (  12,  -15)
         mv_B2:  => (  55,  -28)
```

```
            mv_B0:  => (  -9,  -16)
            mv_A0_col:  => ( -17,   12)
            mv_B2_col:  => (   8,  -61)
            mv_B0_col:  => (  62,  -57)
            mv_upm_2nx2n:  => ( -19,  -56)
            mv_upm_2nxn:  => (  -9,  -42)
            mv_upm_nx2n:  => (   6,   28)
            mvp_A0:  => (  38,  -35)
            mvp_B2:  => ( -51,   -4)
            mvp_B0:  => ( -11,   26)
            mvp_upm_2nx2n:  => ( -45,   37)
            mvp_upm_2nxn:  => ( -31,   -4)
            mvp_upm_nx2n:  => (  42,  -14)
            mv_C0:  => (  26,   -3)
-------------------------------------------------------
...
-----------------------Salidas--------------------------
SR =>   64
-------------------------------------------------------
...
```

4.1.3. Verificaciones simples

Las unidades con juegos de entrada/salida o modelos en SW muy complejos, o en su defecto, unidades con lógica relativamente simple, se verificaron empleando el método simple, descrito en la figura 4.3.

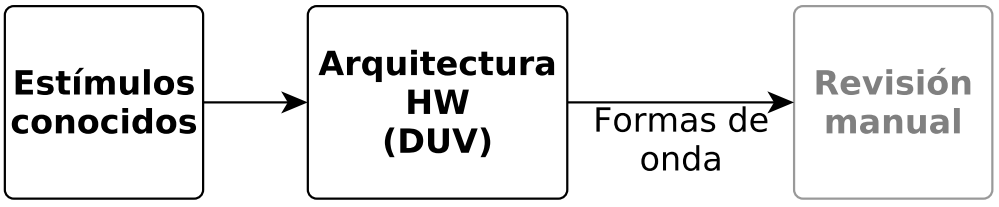


Figura 4.3: Diagrama de bloques - método simple de *test*

La unidad de control/sincronización, de acumulación y la arquitectura integrada fue verificada siguiendo el mencionado flujo. Por su parte, para el diseño del controlador/sincronizador se ha codificado un modelo de las señales de control conectada a la

arquitectura integrada (sin controlador) — esta a su vez, siendo estimulada con entradas conocidas— el cual sirvió para fijar sus especificaciones de temporización. Un determinado juego de señales de control fueron generadas por SW (para un determinado grupo de PUs procesados), y la unidad de control/sincronización fue verificada con las mismas. Es pertinente destacar que todas las simulaciones fueron realizadas utilizando el simulador ISE (ISim) de Vivado.

4.2. Resultados de síntesis

Los reportes de síntesis fueron generados configurando un dispositivo FPGA de gama media perteneciente a la familia Kintex 7 de Xilinx de modelo específico xc7k70tfbv676-1 en la herramienta Vivado. Cabe mencionar que la restricción de tiempo (referido también como *timing constraint*) —en este caso el periodo de la señal de reloj— fue configurada tomando como referencia una tasa de procesamiento mayor a 8K@60 (8K con 60 recuadros por segundo) en relación a la unidad WSAD4x4. Ello, debido a que para dicha unidad, el cálculo del *throughput* es directo, como se muestra en las ecuaciones 4.1, 4.2; donde $P_{T_{clk}}$ es la cantidad de pixeles que la unidad puede procesar por ciclo de reloj, fps es la cantidad de recuadros por segundo, R es resolución y fc es un factor asociado al espacio de color usado. Por lo que se escoge una frecuencia máxima mayor a 200 MHz como referencia, para tareas de optimización (por unidad).

$$P_{T_{clk}} = 16 \qquad f_{max} = \frac{fps.R.fc}{P_{T_{clk}}} \tag{4.1}$$

$$\begin{gathered} fps = 60 \quad R = 7680x4320 \quad fc = 1{,}5 \qquad \text{(8K@60 YCbCr)} \\ \Rightarrow f_{max} = 186{,}62\ MHz \end{gathered} \tag{4.2}$$

Luego, en la tablas 4.1 se muestra un resumen de los reportes obtenidos de las unidades optimizadas que originaban un mayor *delay* combinacional (su versión sin optimizar), de forma independiente.

Tabla 4.1: Reportes de síntesis de unidades independientes

	WSAD4x4 module	**MV Interp. unit**	**DSR module**
Slice LUTs	416 (1.01 %)	113 (0.28 %)	526 (1.28 %)
Slice registers	292 (0.36 %)	85 (0.10 %)	152 (0.19 %)
DSPs	0	0	2 (0.83 %)
Frecuencia máxima	367.38 MHz	317.66 MHz	403.06 MHz
Throughput	>8K@60 (YCbCr)	-	-

Finalmente, en la tabla 4.2, se resumen los reportes obtenidos en la síntesis de la arquitectura integrada. Además, debido a que la unidad WSAD4x4 en la arquitectura integrada opera continuamente —transcurrido el periodo de latencia (gracias al sincronizador desarrollado)— se puede calcular el *throughput* utilizando las ecuaciones 4.1, 4.2. Por otro lado, en cuanto a los FIFOs agregados, se percató que una implementación basada en registros de desplazamiento permite obtener una frecuencia máxima mayor, en comparación a una implementación basada en BRAM (referida también como *Block RAM*), por lo que se adoptó este primero.

Tabla 4.2: Reportes de síntesis de la arquitectura integrada

	Arquitectura integrada
Slice LUTs	2888 (7.04 %)
Slice registers	2551 (3.06 %)
BRAM Tile	1 (0.74 %)
DSPs	14 (5.83 %)
Frecuencia máxima	223.96 MHz
Throughput	8K@72fps (YCbCr)

Por otro lado, se adiciona una comparativa —a nivel de resultados de síntesis y características de la arquitectura— con el trabajo utilizado como referencia para el diseño del módulo AMVP (Abdelsalam 2017) en la tabla 4.2. Como se observa en la tabla, el diseño propuesto alcanza un mayor *throughput*; sin embargo, utiliza una cantidad ligeramente mayor de recursos del FPGA Virtex-7 (menor al 1 % aún). Ello, debido a que la presente arquitectura emplea una unidad de distorsión de mayor cantidad de elementos (WSAD 4x4) e implementa herramientas adicionales como el cálculo del rango de búsqueda y la lógica para resolver dependencias. Adicionalmente, el parámetro λ es considerado como una entrada en el presente trabajo, mientras que en el de Abdelsalam (2017), la multiplicación $\lambda.R$ es implementado como un *look-up-table* (usando un λ fijo).

Tabla 4.3: Comparativa de resultados de síntesis y características de la arquitectura

	Propuesto	**(Abdelsalam, 2016)**
FPGA	Virtex-7 xc7vx550t	
Slice LUTs	2781 (0.8 %)	1535 (<1 %)
Slice registers	2451 (0.35 %)	2155 (<1 %)
BRAM Tile	1 (0.08 %)	N/A
DSPs	14 (0.49 %)	6 (<1 %)
Frecuencia máxima	290.87 MHz	229 MHz
Throughput	8K@93fps (YCbCr)	4K@60fps (YCbCr)
Unidad de dist. parcial	WSAD de 16 elementos	SAD de 8 elementos
Tamaños de PU soportados	Todos	Todos
Herramientas implementadas	MVP y SR	Solo MVP
Problemas de dependencia	Solucionado	No se considera
Aspectos adicionales	λ variable	λ fijo

Conclusiones

- El presente trabajo ha conseguido diseñar una arquitectura correspondiente a la etapa inicial del proceso de ME, que abarca una unidad de predicción de MVs (el módulo AMVP) y una unidad de cálculo de rango de búsqueda (módulo DSR).
- Se han propuesto simplificaciones/mejoras nuevas a los cálculos requeridos (en los algoritmos AMVP y DSR) tal como la utilización del WSAD como métrica de distorsión en el módulo AMVP y su uso como una aproximación a la distancia euclidiana (requerida en el módulo DSR).
- Se ha presentado una arquitectura libre de dependencias con respecto a la etapa posterior: el proceso ME propiamente dicho (tanto en la etapa de determinación del MVP como del cálculo del SR). Ello, facilita el futuro diseño de una arquitectura integrada que incluya la etapa ME, puesto que las operaciones ejecutadas por los módulos correspondientes a la etapa inicial del ME y el ME propiamente dicho podrían ser agregadas a un *pipeline*, lo cual implica una mejora significativa en términos de desempeño.
- Se ha logrado prototipar la arquitectura en un FPGA de gama media (Kintex-7) consiguiendo una tasa de procesamiento en tiempo real mayor a 30 cuadros por segundo (72 fps, en particular) de secuencias de video en calidad UHD (8K, en específico) con una utilización de recursos del FPGA relativamente bajo (7 % como máximo).
- En suma, se ha propuesto una arquitectura que implementa los algoritmos DSR y AMVP, que alcanzan una reducción significativa en el tiempo requerido por el proceso ME y una adicional disminución apreciable del *bit-rate* para el caso del AMVP.

Recomendaciones

- El desarrollo de un ambiente de verificación formal de la arquitectura integrada sería provechoso para garantizar a cabalidad el correcto funcionamiento del diseño propuesto.
- El diseño de una unidad de control adicional que permita bloquear a la arquitectura —para luego retomar el procesamiento ante una señal de reinicio— es sugerida para flexibilizar su uso.
- Se recomienda realizar un análisis del comportamiento exacto en términos de *BD-rate* de los cambios propuestos —para facilitar la implementación en HW— en el software de referencia del estándar HEVC. Alguno de los cambios considerados son los realizados al algoritmo DSR (p.e. el uso del WSAD para el cálculo de distancia), en la métrica de distorsión del módulo AMVP, los mecanismos usados para solucionar problemas de dependencia tanto en la unidad AMVP como en el DSR, entre otros.
- Se plantea el estudio de la utilización y del flujo seguido por el software de referencia del estándar HEVC con el objetivo de posibilitar la proposición de mejoras al estándar y el diseño de ambientes de verificación con estímulos de secuencias de video reales.
- El desarrollo e integración de arquitecturas en HW de las etapas posteriores que conforman el proceso ME acordes con el estado del arte, y una posterior implementación sobre dispositivos FPGA o ASIC de los mismos son considerados en el futuro. También, se contempla la integración con los trabajos previos realizados en el Grupo de Microelectrónica (GμE) —tales como los de Soto (2016), Cano (2012) y Villegas (2011)—pertenecientes a la línea de diseño de arquitecturas para compresión de video.

Bibliografía

[1] ABDELSALAM, A. M., SHALABY, A., AND SAYED, M. S. Towards an FPGA-Based HEVC Encoder: A Low-Complexity Rate Distortion Scheme for AMVP. *CSSP 36* (2017), 4207–4226.

[2] BING, X. *Distortion Measure Analysis for Efficient Block Matching and Face Region Based Video Coding*. Tesis doctoral, Nanyang Technological University, 2009.

[3] BOVIK, A. C. *Handbook of Image and Video Processing (Communications, Networking and Multimedia)*. Academic Press, Inc., Orlando, FL, USA, 2005.

[4] CANO, C. *Diseño de una Arquitectura de un Filtro Digital sobre Muestreo de Imágenes, en Factor 2, de acuerdo al Formato H.263/AVC sobre FPGA*. Tesis de pregrado, Pontificia Universidad Católica del Perú, 2012.

[5] CISCO. The Zettabyte Era: Trends and Analysis. Tech. rep., 2014.

[6] DAMIANI, M. *Synthesis and Optimization of Synchronous Logic Circuits*. Tesis doctoral, Stanford University, 1994.

[7] GULATI, K., AND KHATRI, S. P. *Hardware Acceleration of EDA Algorithms: Custom ICs, FPGAs and GPUs*, 1st ed. Springer Publishing Company, Incorporated, 2010.

[8] HANHART, P., AND EBRAHIMI, T. Calculation of Average Coding Efficiency Based on Subjective Quality Scores. *J. Vis. Comun. Image Represent. 25*, 3 (Apr. 2014), 555–564.

[9] JIANG, X., SONG, T., SHIMAMOTO, T., AND WANG, L. AMVP Prediction Algorithm for Adaptive Parallel Improvement of HEVC. In *2014 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)* (Nov 2014), pp. 511–514.

[10] JOU, S. Y., CHANG, S. J., AND CHANG, T. S. Fast Motion Estimation Algorithm and Design for Real Time QFHD High Efficiency Video Coding. *IEEE Transactions on Circuits and Systems for Video Technology 25*, 9 (Sept 2015), 1533–1544.

[11] KUMM, M., KLEINLEIN, M., AND ZIPF, P. Efficient Sum of Absolute Difference Computation on FPGAs. In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)* (Aug 2016), pp. 1–4.

[12] LAROCHE, G., JUNG, J., AND PESQUET-POPESCU, B. RD Optimized Coding for Motion Vector Predictor Selection. *IEEE Transactions on Circuits and Systems for Video Technology 18*, 9 (Sept 2008), 1247–1257.

[13] LIN, Y.-L. S., KAO, C.-Y., KUO, H.-C., AND CHEN, J.-W. *VLSI Design for Video Coding: H.264/AVC Encoding from Standard Specification to Chip*, 1st ed. Springer Publishing Company, Incorporated, 2010.

[14] NALLURI, P. *A Fast Motion Estimation Algorithm and its VLSI Architecture for High Efficiency Video Coding*. Tesis doctoral, Universidad de Aveiro, 2016.

[15] PALNITKAR, S. *Verilog®Hdl: A Guide to Digital Design and Synthesis, Second Edition*, second ed. Prentice Hall Press, Upper Saddle River, NJ, USA, 2003.

[16] RICHARDSON, I. E. *Video Codec Design: Developing Image and Video Compression Systems*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[17] RICHARDSON, I. E. G. *Design and Performance, in H.264 and MPEG-4 Video Compression: Video Coding for Next-Generation Multimedia*. John Wiley & Sons, Ltd., Chichester, UK, 2003.

[18] SOTO, J. *Diseño de una Arquitectura para la Estimación de Movimiento Fraccional según el Estándar de Codificación HEVC para Video de Alta Resolución en Tiempo Real*. Tesis de pregrado, Pontificia Universidad Católica del Perú, 2016.

[19] SZE, V., BUDAGAVI, M., AND SULLIVAN, G. J. *High Efficiency Video Coding (HEVC): Algorithms and Architectures*. Springer Publishing Company, Incorporated, 2014.

[20] VILLEGAS, C. *Diseño de una Arquitectura para la Interpolación de Quarter-pixel para Estimación de Movimiento según el Formato H-264/AVC empleado en el Estándar SBTVD de Televisión Digital Terrestre*. Tesis de pregrado, Pontificia Universidad Católica del Perú, 2011.

[21] WANG, C.-C., AND LI, G.-L. Hardware-friendly Advanced Motion Vector Prediction Method and its Architecture Design for High Efficiency Video Coding. *Multimedia Tools and Applications* (Feb 2017).

[22] WIEN, M. *High Efficiency Video Coding: Coding Tools and Specification*. Springer Publishing Company, Incorporated, 2015.

[23] XIAO, G. *VLSI Architectures Design for Encoders of High Efficiency Video Coding (HEVC) standard*. Tesis doctoral, Politecnico di Torino, 2016.

[24] XILINX. Divider Generator v5.1 - LogiCORE IP Product Guide. Vivado Design Suite (oct. 2016). https://www.xilinx.com/support/documentation/ip_documentation/div_gen/v5_1/pg151-div-gen.pdf.

[25] XILINX. FIFO Generator v13.1 - LogiCORE IP Product Guide. Vivado Design Suite (ab. 2017). https://www.xilinx.com/support/documentation/ip_documentation/fifo_generator/v13_1/pg057-fifo-generator.pdf.

[26] XILINX. Multiplier v12.0 - LogiCORE IP Product Guide. Vivado Design Suite (nov. 2015). https://www.xilinx.com/support/documentation/ip_documentation/mult_gen/v12_0/pg108-mult-gen.pdf.

[27] YU, L., FU, G., MEN, A., LUO, B., AND ZHAO, H. A Novel Motion Compensated Prediction Framework Using Weighted AMVP Prediction for HEVC. In *2013 Visual Communications and Image Processing (VCIP)* (Nov 2013), pp. 1–6.

[28] ZLATANOVICI, R. *Power - Performance Optimization for Digital Circuits*. Tesis doctoral, University of California at Berkeley, 2006.