UNIVERSIDAD POLITÉCNICA DE VALENCIA

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

Estudio de Arquitecturas VLSI de la Etapa de Predicción de la Compensación de Movimiento, para Compresión de Imágenes y Video con Algoritmos *Full-Search*. Aplicación al Estándar H.264/AVC

TESIS DOCTORAL

Que presenta:

Armando Mora Campos

Dirigida por:

Dr. D. Francisco José Ballester Merelo
Dr. D. Marcos Antonio Martínez Peiró

Valencia, España, Octubre del 2008

Resumen

En esta tesis doctoral se presenta el diseño y realización de arquitecturas VLSI de estimación de movimiento, en sus versiones de pixeles enteros y fraccionarios, para la etapa de predicción de la compensación de movimiento del estándar de codificación de video H.264/AVC. Las arquitecturas propuestas son estructuras de procesamiento segmentadas-paralelas con alta eficiencia en su ruta de datos y una administración óptima de la memoria. Utilizando el algoritmo *full-search* de ajuste de bloques, los diseños cumplen los requerimientos de tamaño de bloque variable y resolución de ¼ de píxel del estándar con máxima calidad. Los estimadores de movimiento combinan las características de las arquitecturas consideradas en el estado del arte junto con la aplicación de nuevos esquemas y algoritmos hardware, en el proceso de codificación del componente luma de la señal de video. Diseñadas como co-procesadores de aceleración hardware para procesadores de 32 bits, las arquitecturas que se presentan han sido simuladas y sintetizadas para FPGA Virtex-4 de Xilinx, utilizando el lenguaje de descripción de hardware VHDL.

In this Ph. D. thesis is presented the design and implementation of integer-pel and fractional-pel motion estimation VLSI architectures, for the motion-compensation prediction stage of the H.264/AVC video-coding standard. The proposed architectures are pipeline-parallel processing structures with high data-path efficiency and optimal memory management. They make use of the full-search block-matching algorithm to fulfil the standard requirements like variable block size and quarter-pel resolution with maximum quality. The motion estimators blend the state-of-the-art architectures characteristics with new hardware schemes and algorithms for the luma video component coding process. The considered architectures were designed as hardware acceleration co-processors for 32-bit processors. They have been simulated and synthesized for Virtex-4 Xilinx FPGA using VHDL hardware description language.

En esta tesi doctoral es presenta el disseny i realització d'arquitectures VLSI d'estimació de moviment, en les seues versions de píxels sencers i fraccionaris, per a l'etapa de predicció de la compensació de moviment de l'estàndard de codificació de vídeo H.264/AVC. Les arquitectures proposades són estructures de processament segmentades-paral·leles amb alta eficiència en la seua ruta de dades i una administració òptima de la memòria. Utilitzant l'algoritme d'ajust de blocs amb busca completa, els dissenys complixen els requeriments de grandària de bloc variable i resolució de ¼ de píxel de l'estàndard amb màxima qualitat. Els estimadors de moviment combinen les característiques de les arquitectures considerades en l'estat de l'art junt amb l'aplicació de nous esquemes i algoritmes maquinari, en el procés de codificació del component luma del senyal de vídeo. Dissenyades com co-processadors d'acceleració maquinari per a processadors de 32 bits, les arquitectures que es presenten han sigut simulades i sintetitzades per a FPGA Virtex-4 de Xilinx, utilitzant el llenguatge de descripció de maquinari VHDL.

A Ma. Concepción, Gaby y Carmelita

A mis Padres, Suegros, Hermanos(a), Cuñados(as), Sobrinos(as)

Agradecimientos

Agradezco a todas las personas que han estado conmigo en el tiempo de mis estudios doctorales y que gracias a su apoyo y colaboración, hicieron posible la realización de esta Tesis.

A nuestro creador, que me ha concedido cumplir mi mayor sueño de juventud.

Al Ing. Carlos Fernández Pérez, por la confianza que me tuvo para que accediera al programa de Doctorado en Diseño de Sistemas Digitales, así como por su apoyo total en las siguientes fases de mis estudios.

A mis directores de tesis, Dr. D. Francisco José Ballester Merelo y Dr. D. Marcos Martínez Peiró, por su acertada definición y dirección del trabajo de investigación, así como por la paciencia que me han tenido a lo largo de varios años.

Al Ing. Oscar Armando López González y al Ing. David Hernández Ochoa, por mantener el apoyo institucional que me ha permitido concluir esta Tesis.

A mis profesores de la UPV, Dr. D. Ángel Sebastia Cortes, Dr. D. Rafael Gadea Gironés, Dr. D. Francisco José Mora Mas, Dr. D. Francisco José Ballester Merelo y Dr. D. Marcos Martínez Peiró, por sus enseñanzas que me encaminaron hacia el conocimiento científico y tecnológico que soporta esta investigación.

A mis compañeras del ITQ, Ing. Silvia González Aguilar, Ing. Martha Erika Duran Castellanos e Ing. Zita de Montserrat Valdés Barrón, por su gran comprensión ante mi situación de profesor/estudiante y por sus palabras de aliento que nunca me faltaron.

A los miembros del Tribunal designado para la defensa de la Tesis Doctoral, Dr. D. Matías J. Garrido González, Dr. D. César Sanz Álvaro, Dr. D. Juan Antonio Michell Martín, Dr. D. Ángel Sebastia Cortes y Dr. D. Gustavo Adolfo Ruiz Robredo, por sus comentarios y observaciones que enriquecieron el trabajo realizado.

A los profesores, Ing. Carlos Benítez Landa y Dr. D. René de Jesús Romero Troncoso, por haberme iniciado y preparado adecuadamente en el área de la Electrónica Digital, y que por su trabajo y dedicación siempre han sido ejemplos a seguir en el ejercicio de mis actividades docentes.

Al profesor Ing. Luis Mario Arenaza Quiñones, por su continua motivación para no ceder y llegar firme a la culminación de mis estudios.

A la T.S. Margarita Mirella Jiménez, por su gran vocación de servicio y su contribución en los siempre extenuantes tramites ante Dirección General.

A los profesores del Instituto Tecnológico de Celaya, M.I. Rubén Valles Villegas, M.I. Justo Navarro Venegas y Dr. D. Agustín Ramírez Agundis, por su

invitación a participar en el programa doctoral, por la excelente coordinación del mismo y por la amistad que siempre me han brindado.

A mis amigos y compañeros del Instituto Tecnológico de Querétaro y en especial a los del Departamento de Ingeniería Eléctrica y Electrónica, por su comprensión por los años en que me dedique exclusivamente a mi formación académica.

A la Dirección General de Educación Superior Tecnológica, por su compromiso permanente de brindar educación tecnológica de calidad a la juventud mexicana y por sus políticas de formación de profesores, que me han permitido realizar estos estudios.

A la Asociación Nacional de Universidades e Institutos de Educación Superior, por la beca que me proporcionó para apoyar los gastos de manutención en las 7 estancias que realicé en Valencia y en especial a los coordinadores de los convenios, Lic. Erik Morales Gutiérrez (SES-ANUIES) y Lic. Oswaldo Pérez Solís (ANUIES-SUPERA).

A mis amigos, familiares, vecinos y todas aquellas personas con las que he convivido en este tiempo de estudios, y que directa o indirectamente contribuyeron a su culminación.

Índice

Lista de Figuras

Lista de Tablas

Capítulo **1**

Introducción

1.1 Introducción.

La tendencia del mercado hacia las *aplicaciones multimedia en todos los equipos y todas las redes* [Hoffman07], ha presionado a la comunidad científica y tecnológica a re-examinar continuamente los algoritmos, arquitecturas, metodologías de diseño y tecnologías de los sistemas de procesamiento de señales, en los que se incluye la compresión de imágenes y video. Los consumidores, inicialmente motivados por los productos de las primeras generaciones, demandan cada vez mejores equipos de aplicaciones múltiples, con altas funcionalidades de tiempo real, mayor interconexión, menor coste y bajo consumo de energía.

Uno de los pilares que ha soportado el desarrollo de equipos y sistemas multimedia es la tecnología de codificación de imágenes y video. Desde su introducción comercial a inicios de la década de los 1990s, su uso ha sido clave para la optimización de los medios de transmisión y almacenamiento. Su contribución actual es de tal magnitud que se ha convertido en parte integral de la forma en que se crea, comunica y consume la información visual, lo que ha fomentado el manejo de otras tecnologías. Por ejemplo, el MPEG-2 (1993) fue el estándar de codificación que impulsó el auge de los sistemas de televisión digital alrededor del mundo, ya que permitió una transmisión eficiente de señales sobre satélites, cables y plataformas terrestres.

Siguiendo la línea de perfeccionamiento que han adoptado los sistemas de codificación de video, el último estándar de la ITU-T y de la ISO/IEC, el H.264 o MPEG-4 parte 10 *Advanced Video Coding* (AVC), incorpora los cambios más significativos desde la introducción del H.261 en 1990. Con el objetivo de mejorar la eficiencia de codificación de los estándares previos, la nueva norma utiliza algoritmos, técnicas y métodos que habían sido excluidos en el pasado por su alta complejidad computacional. Actualmente, la evolución de la tecnología VLSI solventa estas necesidades adicionales de cómputo, gracias al fenómeno conocido como la ley de Moore.

El H.264/AVC mejora en gran medida la capacidad de compresión de las normas anteriores, debido a las técnicas de codificación predictiva que utiliza, entre las que destacan:

* El tamaño de bloque variable.

* La resolución de ¼ de píxel en los vectores de movimiento.

* Los múltiples cuadros de referencia.

Con estas técnicas se busca eliminar la redundancia temporal de la secuencia de video, por medio de la determinación de los vectores de movimiento de menor coste tasa/distorsión.

El algoritmo utilizado para la selección de los vectores de movimiento óptimos en el modelo de referencia de la norma H.264/AVC, es el de ajuste de bloques de búsqueda completa *full-search block matching* (FSBM), en sus versiones de pixeles enteros y fraccionarios. Las necesidades de procesamiento y ancho de banda que demanda este algoritmo son tan altas, que su realización debe ser muy eficiente para cumplir los requerimientos de las aplicaciones de tiempo real.

Por estas razones, en esta tesis se presenta el diseño y realización de arquitecturas VLSI de estimación de movimiento (ME) con algoritmo FSBM, para la etapa de predicción de la compensación de movimiento del estándar H.264/AVC. La selección del algoritmo FSBM se debe al plan del grupo de trabajo en que participa el autor, de desarrollar primeramente arquitecturas ME dé alta regularidad de flujo de datos y direccionamiento, dejando las arquitecturas rápidas para una segunda etapa. El trabajo está dividido en las siguientes fases:

1. Estudio del estado del arte de las arquitecturas FSBM, con lo que se definen las características estructurales y de operación de donde deben partir los nuevos diseños.

2. Propuesta de arquitecturas hardware con estructura segmentada-paralela, para satisfacer los requerimientos de estimación de movimiento con tamaño variable de bloques y resolución entera y fraccionaria, para el componente luma de la señal de video.

3. Ejemplo de desarrollo de las arquitecturas propuestas, con descripción en VHDL y realización en tecnología de lógica programable (FPGA), para obtener co-procesadores de aceleración hardware de sistemas embebidos.

1.2 Objetivos.

1. Evaluar el grado de concurrencia que acepta la etapa de estimación de movimiento del codificador H.264/AVC, por medio del estudio del estándar y su modelo de referencia. Los resultados obtenidos apoyaran el diseño de arquitecturas altamente paralelas que conserven la calidad de codificación del modelo secuencial original, sin costes excesivos de área y ancho de banda.

2. Analizar las características estructurales y de operación de las arquitecturas hardware de estimación de movimiento con algoritmo FSBM que se encuentran en el estado del arte. Las características observadas se deben tomar como punto de referencia para la propuesta de nuevas arquitecturas enfocadas a la norma H.264/AVC.

3. Proponer técnicas y métodos de optimización del coste computacional y ancho de banda, en el diseño de arquitecturas de estimación de movimiento con algoritmo FSBM y resolución de pixeles enteros y fraccionarios, que no afecten significativamente la calidad y eficiencia de codificación de video del H.264/AVC.

4. Diseñar arquitecturas hardware para la etapa de estimación de movimiento del H.264/AVC, con algoritmo FSBM y resolución de pixeles enteros, para el componente luma de la señal de video. Se necesita que las arquitecturas cumplan las siguientes especificaciones de diseño, con la máxima eficiencia y menor ancho de banda: pixeles de 8 bits, formato de video hasta 1080 HD, velocidad de 30 fps, un cuadro de referencia y desplazamiento máximo de exploración de 56 pixeles. Las arquitecturas propuestas también deben satisfacer las siguientes características estructurales y de operación:

   * Procesamiento de cada macro-bloque candidato en un ciclo de reloj.

   * Operación independiente del tamaño de la ventana de búsqueda.

   * Latencia mínima y cero ciclos falsos.

   * Tamaño mínimo del *buffer* de referencia.

   * Alto grado de paralelismo.

   * Reutilización de datos al nivel de pixeles y bloques candidatos.

   * Unidad generadora de direcciones básica.

5. Diseñar arquitecturas hardware para la etapa de estimación de movimiento del H.264/AVC, utilizando el algoritmo FSBM con resolución de pixeles fraccionarios, para el componente luma de la señal de video, que cumplan con las siguientes características:

   * Secuencia de procesamiento *half* y *quarter* de las 41 particiones y sub-particiones de cada macro-bloque.

   * Desplazamiento de exploración vertical y horizontal de ± ½ píxel para la resolución *half* y de ± ¼ píxel para la resolución *quarter*.

   * Alto rendimiento para procesar a la misma o mayor velocidad, el flujo de datos que recibe de la arquitectura de estimación de movimiento con resolución entera previa.

   * Flujo de datos continuo en la entrada del módulo de interpolación.

   * Área mínima.

   * Bajo número de ciclos de latencia.

   * Alto nivel de reutilización de datos.

6. Realizar las arquitecturas propuestas sobre FPGA, utilizando técnicas de diseño para comportamiento y reutilización, para obtener núcleos de propiedad intelectual de estimación de movimiento que puedan ser integrados en un sistema completo de codificación H.264/AVC.

1.3 Metodología.

En el proceso de definición y realización de esta tesis, se ha seguido un ciclo de investigación apoyado en el método científico. A continuación se enuncian los pasos de su desarrollo, junto con las herramientas utilizadas en cada uno de ellos (Figura 1.1).

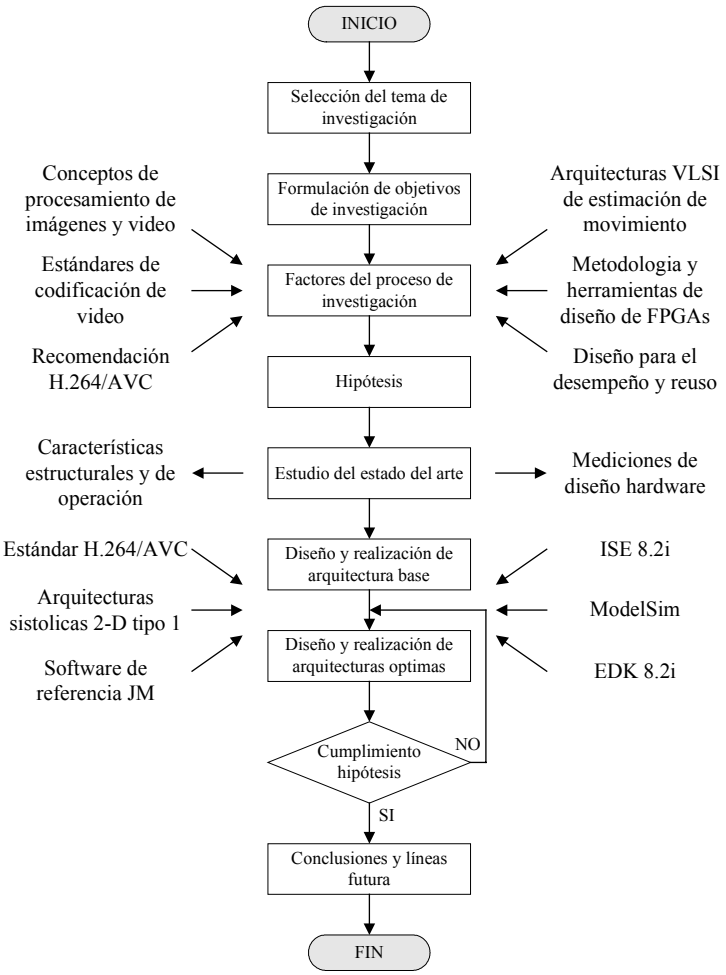


Figura 1.1. Metodología de desarrollo de la tesis.

El primer paso del trabajo de tesis fue la selección del tema de investigación. Después de evaluar varias áreas de estudio, con el apoyo de los directores de tesis se eligió el tema *Estudio de Arquitecturas VLSI de la Etapa de Predicción de la Compensación de Movimiento para Compresión de Imágenes y Video con Algoritmos Full-Search. Aplicación al Estándar H.264/AVC*, al ser este un tema original de actualidad, que aborda problemas que aportan resultados prácticos para la solución de los mismos. De igual forma, el enfoque del tema corresponde a la disciplina de los estudios de doctorado y refleja interés social y tecnológico, ya que gira alrededor de una de las problemáticas de mundo moderno, que es la optimización de los recursos de transmisión y almacenamiento de información visual.

Una vez definido el tema, en el segundo paso se han formulado una serie de objetivos concretos, a partir de los cuales se han obtenido resultados que se reflejan en

el documento de tesis. La utilidad de estos resultados se corrobora con las publicaciones obtenidas en congresos y revista.

El título del tema y los objetivos enuncian los factores que inciden en el proceso de investigación y que deben estar consignados en el marco teórico. Para ello, en el tercer paso de la metodología se han listado inicialmente todos los elementos de apoyo y consulta, para su posterior estudio, análisis e interpretación. Es en este punto donde se tiene un acercamiento con los conceptos de procesamiento de imágenes y video, y con las definiciones, algoritmos, técnicas y estándares de compresión, hasta llegar al estudio de la recomendación H.264/AVC. Asimismo se presentan las primeras herramientas, que por su importancia son primordiales para el desarrollo de la investigación: el documento oficial del estándar H.264/AVC [H.264.05] y el software de referencia [JM1106] del mismo. Otros factores que se consideran en esta etapa son la teoría de diseño de arquitecturas VLSI de estimación de movimiento y la metodología y herramientas de diseño sobre FPGA, ya que es la tecnología que se utiliza como ejemplo de realización de las arquitecturas propuestas.

En el cuarto paso del ciclo, se llega a la posición de establecer una respuesta posible al problema de investigación. Para el trabajo presente, esta conjetura o hipótesis plantea que con arquitecturas hardware de estimación de movimiento con algoritmo FSBM, se pueden satisfacer los requerimientos de tamaño de bloque variable, resolución de ¼ de píxel y múltiples cuadros de referencia del estándar H.264/AVC, para formatos de video de alta definición, utilizando tecnología VLSI. Esta solución exploratoria marca el inicio de la cadena de razonamientos que tienen como propósito su verificación.

La prueba de la hipótesis sigue el camino del cumplimiento de los objetivos planteados. En este punto se tiene el soporte para atender el estudio del estado del arte de las arquitecturas hardware de estimación de movimiento. El análisis de estas arquitecturas ha permitido identificar las características estructurales y de operación que las destacan, así como determinar las medidas de diseño para su evaluación y comparación, desde un punto de vista hardware.

Las comparaciónes de las arquitecturas propuestas con las arquitecturas previas, utilizando las medidas de diseño hardware, muestran las condiciones y rangos de operación donde es valida la hipótesis. Cabe destacar que para cada una de las fases de estimación de movimiento (resolución de pixeles enteros y fraccionarios) se ofrecen soluciones múltiples. La tesis presenta el diseño de una arquitectura base para cada caso y posteriormente se aplican en ellas otras técnicas y métodos de diseño y optimización. Al final se tiene un grupo de arquitecturas, donde cada una cubre diferentes aspectos, como máxima calidad de codificación, área mínima, menor número de ciclos de procesamiento, equilibrio área/velocidad, etc.

Después de las conclusiones, la metodología finaliza con las líneas de investigación futuras. En ellas se proponen algunas áreas de estudio que contemplan el diseño y realización de arquitecturas VLSI de estimación de movimiento. Su desarrollo planteará nuevas preguntas de investigación, cuya respuesta reforzará el trabajo realizado.

1.4 Principales aportaciones.

La principal aportación de esta tesis doctoral es el diseño de las siguientes arquitecturas VLSI, para la etapa de predicción de la compensación de movimiento de la recomendación H.264/AVC:

* Arquitectura de estimación de movimiento entero, con algoritmo *full-search block matching* (IME-FSBM).

* Arquitectura IME-FSBM con ventana de búsqueda de forma variable.

* Arquitectura de estimación de movimiento fraccionario, con algoritmo *full-search block matching* (FME-FSBM) y equilibrio área/velocidad.

* Arquitectura FME-FSBM con enfoque a velocidad.

Las arquitecturas propuestas optimizan la aplicación del algoritmo de estimación de movimiento FSBM en sus versiones de resolución de pixeles enteros y fraccionarios, para el componente de video luma. Su evaluación muestra excelentes resultados de velocidad, área, ancho de banda y eficiencia, gracias a su estructura segmentada-paralela, que incluye el cálculo del coste tasa/distorsión y la administración de las memorias de bloques actuales y candidatos.

En el diseño de la arquitectura IME-FSBM, se ha utilizado una estrategia de exploración de la ventana de búsqueda tipo serpenteo, con una matriz de 16×16 elementos procesadores y 16 registros auxiliares. Los resultados son alta velocidad de procesamiento, cero ciclos falsos en el cambio de columna de la ventana de búsqueda y bajo nivel de latencia. La dificultad de diseño de la técnica de tamaño de bloque variable se simplifica con el cálculo de la distorsión de los bloques 4×4 de cada macro-bloque candidato. La reutilización de los resultados permite determinar la distorsión de los bloques mayores utilizando un árbol sumador. El diseño propuesto presenta una operación independiente del tamaño de la ventana de búsqueda y la característica de modificar en línea el avance de la exploración vertical. Esta propiedad permite tomarlo como base para el desarrollo de la arquitectura IME-FSBM con ventana de búsqueda de forma variable. La idea es reducir el número de bloques candidatos a comparar, utilizando ventanas con formas geométricas que toman en cuenta la distribución más probable de los vectores de movimiento.

La primera arquitectura FME-FSBM es un diseño de buen equilibrio área/velocidad, gracias a los niveles de concurrencia, reutilización de datos y utilización en todos sus componentes. La clave es la aplicación eficiente de técnicas de descomposición 4×4 e integración vertical. La segunda arquitectura FME-FSBM se diseña a partir de la versión inicial, modificando la ruta de datos e incrementando su grado de paralelismo. Los cambios se enfocan a eliminar ciclos falsos en el flujo de procesamiento, incrementar la reutilización de datos con descomposiciones 8×4 y duplicar el nivel de paralelismo con secuencias simultáneas de procesamiento 4×4 y 8×4. Los resultados muestran una arquitectura con 3.255 veces la velocidad inicial y solo 2.5 veces el área original.

1.5 Esquema de la tesis.

De acuerdo al contenido y metodología, la presente tesis se puede dividir en tres secciones. La primera, compuesta por los capítulos 2 y 3, incluye el soporte teórico y documental que apoya este trabajo. La segunda se orienta al proceso de diseño y realización de las arquitecturas VLSI propuestas (capítulos 4 y 5). En la tercera sección, la tesis finaliza con las conclusiones generales, aportaciones principales y futuras líneas (capítulo 6), así como las referencias bibliográficas y la lista de publicaciones consideradas como los productos de investigación.

El capítulo 2 se centra en documentar los temas esenciales del trabajo de investigación: los sistemas de codificación de imágenes y video y el estándar H.264/AVC. Inicia con la introducción al video digital y continúa con una referencia al procesamiento digital de imágenes y video y a las técnicas y estándares de codificación JPEG y MPEG. Finaliza con una descripción general de la recomendación H.264/AVC, de sus técnicas de inter-predicción y del algoritmo utilizado en la estimación de movimiento.

Las metodologías y herramientas de diseño se revisan en el capítulo 3. Inicia con el estudio de los sistemas hardware de procesamiento de imágenes y video y las soluciones de codificación H.264/AVC que se ofrecen en el mercado. A continuación se atiende el concepto de sistemas embebidos, el cual engloba a la mayoría de las aplicaciones digitales, incluyendo la codificación de video. Ya que como ejemplo las arquitecturas propuestas se realizan sobre FPGA, se presentan las metodologías, lenguajes de descripción, herramientas y técnicas de diseño para esta tecnología de lógica programable. Se finaliza con la referencia a los procesadores MicroBlaze y PowerPC y su esquema de operación multiprocesamiento.

Los capítulos 4 y 5 inician con el estudio del estado del arte de las arquitecturas de estimación de movimiento con algoritmo FSBM, en sus versiones de pixeles enteros y fraccionarios. Se prosigue con la descripción del diseño de las arquitecturas propuestas y su posterior realización sobre FPGA. En cada caso se profundiza en la aplicación de otras técnicas y métodos para optimizar las primeras arquitecturas, dando como resultado diseños con un mejor comportamiento en velocidad. Los capítulos finalizan con las conclusiones de diseño, realización y optimización.

Capítulo **2**

Codificación de Video y Estándar H.264/AVC

2.1 Introducción al video digital.

En pocos años los sistemas de información visual emigraron de la tecnología analógica a la secuencia de imágenes y video digital, gracias al incremento de la capacidad de procesamiento de la tecnología digital y a las facilidades de la representación resultante, de ser convenientemente procesada, almacenada y transmitida, utilizando sistemas de cómputo de bajo coste. El video digital, confinado anteriormente a aplicaciones profesionales, se ha vuelto un tema de dominio público debido al consumo masivo de equipos como DVDs y videojuegos y a la introducción de la televisión digital. Las investigaciones de nuevas tecnologías, herramientas de diseño, algoritmos y arquitecturas de procesamiento, junto con el desarrollo de estándares, han dando como resultado aplicaciones cada vez más sofisticadas, que abarcan campos de acción no considerados previamente, lo que da soporte a mayor variedad de contenido e inter-conectividad, con altos niveles de calidad.

El término video hace referencia a una secuencia de cuadros de información visual natural (del mundo real), asociados con la banda de frecuencias visibles del espectro electromagnético. En el concepto de secuencia de imágenes, no existe tal restricción; por ejemplo, la secuencia de imágenes infrarrojas corresponde a una banda fuera de la parte visible del espectro. Esto hace que la expresión secuencia de imágenes tenga un alcance más amplio que el vocablo video y cuando se trata de la banda visible, ambos términos son intercambiables [Shi99]. En el ambiente de los sistemas de codificación, el término video puede significar un solo cuadro o una secuencia de cuadros, concepto que es más pertinente en sistemas multimedia; en cambio, la palabra imagen generalmente ubica un cuadro independiente.

La representación visual, por medio de una secuencia de imágenes, tiene una existencia de casi dos siglos. Una de las primeras máquinas para exhibir imágenes en movimiento, la rueda de la vida (Daedaleum), inventada por el matemático William George Horner en 1834, presenta una serie de imágenes, una a la vez, a través de cortes realizados en un tambor circular, cuando este gira. Su principio de operación ilustra conceptos que son la base de los sistemas de visión modernos [Reed04]:

* La impresión de movimiento es ilusoria. Este es el resultado de una propiedad del sistema visual humano referido como una persistencia de la visión: las imágenes percibidas permanecen un periodo de tiempo después de que son remplazadas.

* La percepción del movimiento es directamente proporcional a la velocidad de rotación del tambor: a baja velocidad, las imágenes aparecen como una secuencia de imágenes fijas disjuntas; al incrementar la velocidad, primero se llega a un punto donde se percibe el movimiento, pero acompañado de un parpadeo en la secuencia de imágenes y después se alcanza la llamada frecuencia crítica de fusión, donde el parpadeo desaparece y la impresión de movimiento llega a su mejor nivel.

* Los cortes en el tambor ilustran un aspecto importante de la ilusión: en orden de percibir el movimiento de una secuencia de imágenes, el estímulo que cada imagen individual representa debe ser removido por un periodo de tiempo entre cada presentación. Si no es así, la secuencia de imágenes se hace borrosa y cancela la percepción de movimiento.

Un sistema de captura de video se compone básicamente de dos sub-sistemas: el óptico y el de grabación. El sub-sistema óptico forma imágenes 2-D de la radiación electromagnética emitida y reflejada por los objetos de una escena 3-D y consiste en una serie de lentes que enfocan la radiación sobre una superficie llamada plano focal. El sub-sistema de grabación, en sus dos modalidades, químico y electrónico, se diseña para medir y almacenar las características de la radiación en el plano focal, la cual es función del espacio, tiempo y longitud de onda.

La película representa el elemento químico más común para grabar las imágenes proyectadas en el plano focal. Se compone de una emulsión de partículas sobre un substrato transparente flexible, que se altera cuando se expone a la luz, grabando la imagen en el material fotográfico. Un proceso químico expone las partículas para que la imagen pueda ser vista bajo iluminación normal. Un conjunto de películas, grabadas en instantes de tiempo espaciados uniformemente, constituye una secuencia de imágenes que forma una representación aproximada del patrón de intensidad de luz variante en el tiempo, que incide en el plano focal.

Los sistemas electrónicos son actualmente el medio normal para capturar y almacenar las imágenes del plano focal. Los tubos de exploración de video y más recientemente los sensores matriciales 2-D de estado sólido, como el dispositivo acoplado por carga “charge-coupled device” (CCD), el dispositivo de inyección de carga “charge-injection device” (CID) y los sensores CMOS, son utilizados para medir la iluminación incidente. Estos sensores usan materiales semiconductores para generar y recuperar las cargas producidas por la interacción de los fotones y el material sensible. La reproducción del color se obtiene con el uso de filtros matriciales, que aseguran la medición de la longitud de onda de cada píxel en bandas espectrales especificas.

A continuación se presentan algunos conceptos y terminologías que soportan el campo del video digital, como representación, espacios de color y formatos.

2.1.1 Representación del video digital.

Una imagen digital puede ser obtenida por la cuantización espacial y de amplitud de una imagen continua [Shi99]. El muestreo de la imagen se realiza con la digitalización de las coordenadas espaciales, mientras que la cuantización al nivel de grises se hace con la digitalización de la amplitud. Partiendo de que una imagen continua se denota por $g(x,y)$, donde la amplitud de g en el punto (x,y) es la intensidad o

brillo de la imagen en ese punto. La transformación de una imagen continua en una imagen digital puede ser expresada como *f(m,n)* (2.1), donde *Q* es un operador de cuantización, x_0 y y_0 son las coordenadas del origen del plano de imagen, *m* y *n* son los valores discretos 0, 1, 2,... y Δx y Δy son los intervalos de muestreo en las direcciones horizontal y vertical respectivamente.

$$f(m,n) = Q[g(x_0 + m\Delta x, y_0 + m\Delta y) \quad (2.1)$$

Si el proceso de muestreo es extendido a una tercera dirección temporal, se obtiene la secuencia *f(m,n,t)* (2.2), donde *t* toma los valores 0, 1, 2,... y Δt es el intervalo de tiempo.

$$f(m,n,t) = Q[g(x_0 + m\Delta x, y_0 + n\Delta y, t_0 + t\Delta t)] \quad (2.2)$$

Cada punto de la imagen o muestra espacial representa un píxel y cada imagen individual especifica un cuadro (Figura 2.1). Los cuadros son normalmente exhibidos en un intervalo de tiempo regular para que el ojo perciba un movimiento fluido.

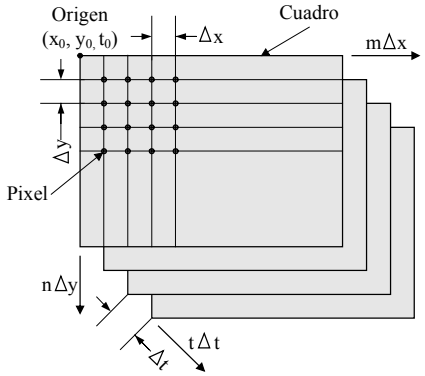


Figura 2.1. Representación de una secuencia de video.

2.1.2 Video progresivo y entrelazado.

Una señal de video puede ser muestreada como una secuencia de cuadros completos (video progresivo) o como una secuencia de campos entrelazados (video entrelazado). En los inicios de la televisión, la técnica de "entrelazado" fue utilizada para reducir la cantidad de información de cada imagen. Esta consiste en el muestreo de la mitad de los datos de un cuadro (un campo) en cada intervalo de muestreo temporal; el campo superior se forma con las líneas impares y el inferior con las líneas pares de un cuadro [Richarson03].

Una secuencia de video entrelazado consiste de una serie de campos superior-inferior-superior..., cada uno representando la mitad de la información del cuadro de video. Otra ventaja del muestreo entrelazado es que es posible transmitir y exhibir el doble de campos por segundo, con respecto a la transferencia de cuadros completos, para la misma tasa de datos, dando la apariencia de un movimiento más suave. Actualmente, la mayoría de las señales *broadcast* son transmitidas como entrelazadas y

la mayor parte de los monitores basados en CRT todavía son entrelazados, mientras que los monitores LCD y plasma generalmente son progresivos [Jack05] (Figura 2.2).

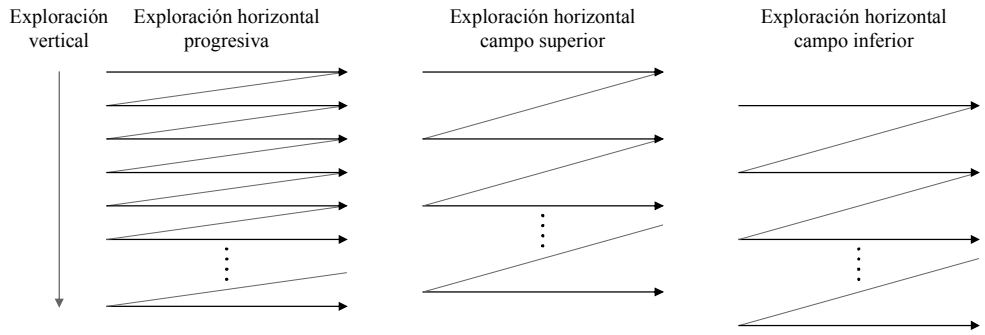


Figura 2.2. Exhibición progresiva y entrelazada (campos superior e inferior).

2.1.3 Espacios de color.

Los sistemas de video digital utilizan modelos matemáticos para representar la información de color, llamados espacios de color, siendo los más populares el RGB (red, green, blue) y el YCbCr (luma, croma blue, croma red) [Jack05]. Las imágenes monocromáticas requieren solo un número para indicar la brillantez o luminancia de cada píxel, mientras que las imágenes a color requieren un mínimo de tres números por cada posición del píxel, para representar el color adecuadamente.

2.1.3.1 Espacio de color RGB.

El espacio de color RGB representa cada muestra de una imagen a color con tres números, que indican la proporción relativa de los colores rojo, verde y azul. A partir de la suma de estos tres colores aditivos primarios de la luz, con igual ancho de banda y resolución, se puede crear cualquier color [Richarson03]. Su relación se dibuja como un cubo en un sistema de coordenadas cartesianas 3-D (Figura 2.3), donde la línea diagonal define los niveles de gris, al incluir la misma cantidad de componentes primarios. La captura de una imagen RGB se realiza filtrando los componentes rojo, verde y azul de la escena y usando un sensor matricial para cada color. La exhibición de una imagen RGB, ilumina los tres componentes de cada píxel de acuerdo a su intensidad, los cuales se mezclan para dar la apariencia de un color verdadero.

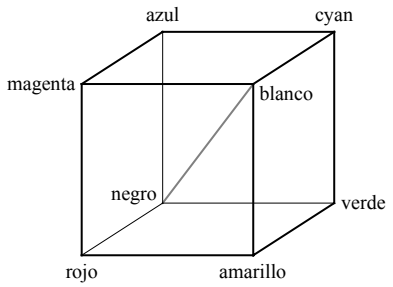


Figura 2.3. Cubo de color RGB, donde la línea diagonal representa los niveles de gris.

2.1.3.2 Espacio de color YCbCr.

El sistema de visión humana es más sensible a la iluminación (brillantez) que al color, por lo que es más eficiente representar el color separando la iluminación (luma) de la información de color (croma), y representando el componente luma con una mayor resolución que los componentes croma, sin tener una contracción evidente de la calidad visual.

Una imagen a color es descrita completamente por el componente luma *Y* y los componentes croma *Cb*, *Cr* y *Cg*, donde *Y* se puede calcular como la suma pesada de las intensidades de color *R*, *G* y *B* (2.3), siendo *kr*, *kg* y *kb* los factores de peso, donde *kr* + *kg* + *kb* = 1; *Cr*, *Cg* y *Cb* se definen como la diferencia entre *R*, *G* y *B* y *Y* respectivamente (2.4).

$$Y = kr\,R + kg\,G + kb\,B \tag{2.3}$$

$$\begin{aligned} Cr &= R - Y \\ Cg &= G - Y \\ Cb &= B - Y \end{aligned} \tag{2.4}$$

Ya que la suma *Cr* + *Cg* + *Cb* es una constante, solo se necesita almacenar o transmitir dos de los tres valores croma, siendo los componentes *Cb* y *Cr* los seleccionados en el espacio de color YCbCr. Un espectador aleatorio, no observa una diferencia clara entre una secuencia de imágenes RGB y su par YCbCr con resolución croma reducida, por lo que el uso de YCbCr es un método natural de compresión de video. Además, su utilización simplifica la realización de los controles de brillantez, contraste, saturación y tinte de la señal de video.

Puesto que los sistemas RGB predominan en los procesos de captura y exhibición de video, ya que simplifican la arquitectura y diseño del sistema, una imagen RGB puede ser convertida a YCbCr para su procesamiento, almacenamiento y/o transmisión y regresada al espacio de color original para su exhibición. La recomendación ITU-R BT.601 define *kb* = 0.114 y *kr* = 0.299, por lo que las expresiones (2.5) y (2.6) muestran las ecuaciones de conversión correspondientes [Richarson03].

$$\begin{aligned} Y &= 0.299\,R + 0.587\,G + 0.114\,B \\ Cb &= 0.564(B - Y) \\ Cr &= 0.713(R - Y) \end{aligned} \tag{2.5}$$

$$\begin{aligned} R &= Y + 1.402\,Cr \\ G &= Y - 0.344Cb - 0.714\,Cr \\ B &= Y + 1.772\,Cb \end{aligned} \tag{2.6}$$

2.1.3.3 Formatos de muestreo YCbCr .

Las imágenes en el espacio de color YCbCr se componen de tres matrices de muestras, una luma y dos croma, donde las estructuras del formato de muestreo croma más comunes son los patrones 4:4:4, 4:2:2 y 4:2:0 (Figura 2.4). En el muestreo 4:4:4, cada matriz croma tiene la misma altura y el mismo ancho que la matriz luma, es decir,

cada muestra tiene un valor *Y*, *Cb* y *Cr*, lo que preserva la fidelidad total de los componentes croma. Los números indican la tasa relativa de muestreo de cada componente en la dirección horizontal. En el patrón 4:2:2, cada matriz croma tiene la misma altura y la mitad del ancho de la matriz luma, por lo que se utiliza en reproducción de color de alta calidad [Richarson03]. Los números significan que por cada 4 muestras luma en la dirección horizontal, existen 2 muestras de cada componente croma.

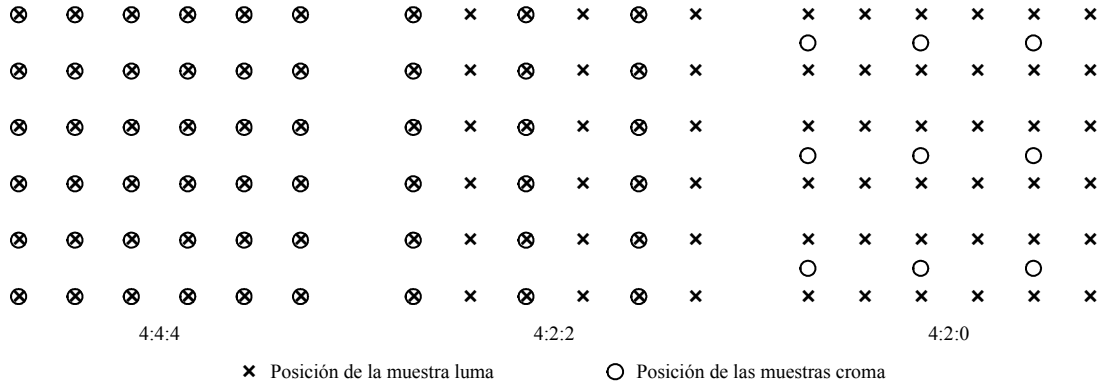


Figura 2.4. Posiciones nominales de las muestras luma y croma, en los patrones 4:4:4, 4:2:2 y 4:2:0 (imagen progresiva).

En el formato de muestreo comúnmente usado para compresión de video, y el más popular para aplicaciones de consumo como video conferencias, televisión digital y almacenamiento DVD (4:2:0), cada matriz croma tiene la mitad de la altura y la mitad del ancho de la matriz luma [H.264.05]. Los números 4:2:0 no definen una relación de muestreo, solo especifican el patrón donde cada componente *Cb* y *Cr* tiene la cuarta parte de las muestras del componente *Y*. En función de las aplicaciones, la posición de las muestras croma con respecto a las muestras luma puede variar, por lo que se definen diversos formatos de muestreo 4:2:0 (Figura 2.5).

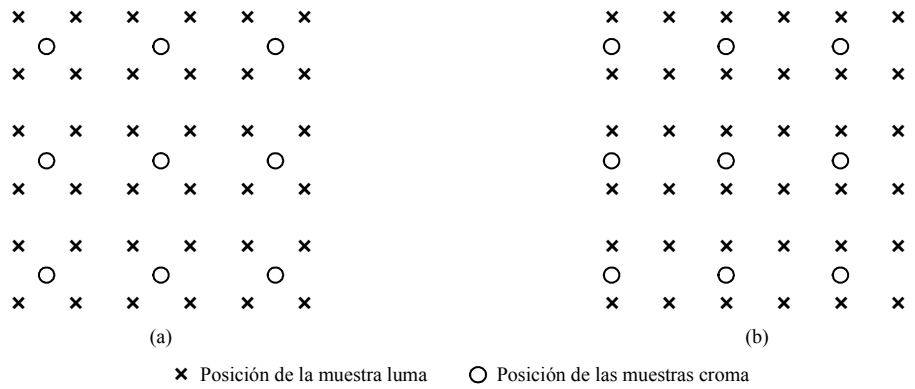


Figura 2.5. Formatos de muestreo 4:2:0 según su aplicación en los estándares de compresión de video a) H.261, H.263 y MPEG-1 y b) MPEG-2, MPEG-4 parte 2 y H.264 (imagen progresiva).

Al nivel de bits, cada pixel en los muestreos 4:4:4, 4:2:2 y 4:2:0 requiere un promedio de 24, 16 y 12 bits en aplicaciones de consumo (muestras de 8 bits) y 30, 20 y 15 bits en aplicaciones de video profesional (muestras de 10 bits) respectivamente [Jack05].

2.1.4 Formatos de video.

Existe una amplia variedad de formatos de cuadros de video, en función de su tamaño (ancho y alto) en píxeles (Tabla 2.1). Algunos de ellos son propios de sistemas como NTSC (National Television System Committee) o PAL (Phase Alternate Line) y otros son compatibles a varios de ellos. Su selección depende de las necesidades de la aplicación y de la disponibilidad de capacidad de almacenamiento, procesamiento y transmisión del sistema que los utiliza. La relación del número de píxeles por línea vs. el número de líneas/cuadro es conocida como la razón de aspecto.

Tabla 2.1. Tamaños del cuadro luma para algunos formatos de video.

Formato	Ancho luma (pixeles por línea)	Altura luma (líneas/cuadro)	# muestras luma
SQCIF	128	96	12 288
QCIF	176	144	25 344
QVGA	320	240	76 800
525 SIF	352	240	84 480
CIF	352	288	101 376
525 HHR	352	480	168 960
625 HHR	352	576	202 752
VGA	640	480	307 200
525 4SIF	704	480	337 920
525 SD	720	480	345 600
4CIF (D1)	704	576	405 504
625 SD	720	576	414 720
SVGA	800	600	486 400
XGA	1024	768	786 432
720p HD	1280	720	921 600
4VGA	1280	960	1 228 800
SXGA	1280	1024	1 310 720
525 16SIF	1408	960	1 351 680
16CIF	1408	1152	1 622 016
4SVGA	1600	1200	1 920 000
1080 HD	1920	1088	2 088 960
2K×1K	2048	1024	2 097 152
4XGA	2048	1536	3 145 728
16VGA	2560	1920	4 915 200
3616×1536(2.35:1)	3616	1536	5 554 176
3672×1536(2.39:1)	3680	1536	5 652 480
4K×2K	4096	2048	8 388 608
4096×2304(16:9)	4096	2304	9 437 184
4K×4K	4096	4096	16 777 216
UHDV	7680	4320	33 177 600

Por ejemplo, el formato intermedio común (CIF), define un grupo de formatos no entrelazados, desde el 4CIF hasta el Sub-QCIF (SQCIF), compatibles con los sistemas de televisión NTSC y PAL. El formato 4CIF es apropiado para televisión en definición estándar y video DVD, los CIF y QCIF son populares en videoconferencias y los formatos QCIF y SQCIF se aplican en video móvil [Richarson03].

2.2 Procesamiento digital de imágenes y video.

El desarrollo de la computadora basada en semiconductores y la carrera espacial de la década de los 1960s, acercaron el campo del procesamiento digital de imágenes y video a los centros académicos y de investigación. Posteriormente, en la década de los 1980s, los avances tecnológicos asociados con la industria VLSI y las herramientas de software, incrementaron la capacidad de las computadoras y redujeron su coste, con lo que el procesamiento digital de la secuencia de imágenes se convirtió en una área de investigación de gran interés económico, gracias a un mercado mundial, siempre ávido de aplicaciones de video en equipos de cómputo, consumo popular, telecomunicaciones, medicina, etc. [Shi99].

El procesamiento digital de video e imágenes consiste de un conjunto de algoritmos, métodos y técnicas, ejecutados en sistemas hardware/software, para el procesamiento de la información visual en medios donde se necesitan acciones como transformar imágenes de baja calidad en imágenes nítidas, producir efectos especiales, mezclar imágenes con gráficas y texto, comprimir la información, estandarizar el formato de video, corregir las no-linealidades de los dispositivos de entrada y salida, compatibilizar señales y equipos, etc.

Algunos ejemplos de procesamiento digital de video e imágenes son [Jack05]:

* Conversión del formato de muestreo, de video progresivo a entrelazado y viceversa, para compatibilizar señales y equipos.

* Escalado del video, de un tamaño de cuadro a otro (ejemplo, de SDTV a HDTV), modificando la resolución y las características de colorimetría.

* Conversión entre espacios de color (por ejemplo, de RGB a YCbCr).

* Limitación de rangos y eliminación de transitorios de los componentes Cb y Cr, para la eliminación de los colores erróneos y el realce de la exhibición.

* Conversión 4:4:4 YCbCr a 4:2:2 YCbCr, para el aprovechamiento de la mayor sencillez de las interfases de video, del formato de muestreo 4:2:2 YCbCr.

* Conversión de la tasa de exploración, en cuadros o campos por segundo.

* Corrección gamma, para la linealización de la salida de intensidad de un monitor de TV y la reducción del ruido inducido en la transmisión de la señal de video.

* Corrección de contraste, para la distribución de las áreas de luz y oscuridad en las imágenes o cuadros de video y la obtención de imágenes claras y de alto contraste, a partir de fuentes de poca variación de brillantez.

* Filtrado, para mejorar la nitidez, reducir la difusión de la imagen, detectar bordes, realizar efectos especiales, etc.

* Manipulación del orden del flujo de píxeles, para rotar, trasladar, recortar, etc., la información de video.

* Mezcla de video digital, para la combinación de imágenes por medio de operaciones de suma, promedio y multiplicación de los datos digitales y composición de la información de video con texto y gráficas generadas por computadora.

* Codificación, para la reducción de la cantidad de información de video y la consecuente simplificación de los procesos de manipulación, almacenamiento y transmisión.

La técnica de codificación revolucionó la industria del video digital, ya que además de reducir la cantidad de información, habilita un gran número de manipulaciones en el flujo de bits resultante del proceso de compresión, las cuales son complicadas o imposibles de ejecutar en el ambiente analógico. Por ejemplo, los datos de audio y video y la información de sincronía y temporización, pueden ser primero comprimidos y después combinados para formar un solo flujo de bits, proporcionando una solución multimedia para aplicaciones prácticas.

El nivel tecnológico actual permite realizar procesamientos de alto nivel en tiempo real, como la adaptación de los datos de entrada al rango dinámico de un monitor plano (corrección gamma), aplicando una curva de linealización a cada píxel en cada cuadro de la señal de video, lo cual mejora dramáticamente la calidad de la imagen final [Tusch06]. Pero no todo es perfecto en el procesamiento digital. La manipulación digital genera efectos no-deseados en los datos de video que deben ser minimizados. Por ejemplo, las técnicas de redondeo por truncamiento causan acumulación de errores, que provocan contornos visibles en áreas de colores sólidos.

2.2.1 Clasificación.

El procesamiento de imágenes puede clasificarse según el tipo de información de entrada y salida de la siguiente forma [Zuloaga98]:

* **Procesamiento imagen-imagen.** Cuando la entrada al proceso requiere una información de tipo imagen y la salida del mismo es una información de tipo imagen. Un ejemplo es el cambio de nivel de gris de una imagen.

* **Procesamiento imagen-dato.** Cuando la entrada al proceso requiere de información de tipo imagen y la salida es una información de tipo dato. Un ejemplo es la obtención del nivel medio de gris de una imagen.

* **Procesamiento dato-imagen.** Cuando la entrada al proceso requiere de información de tipo dato y la salida es información de tipo imagen, por ejemplo la reconstrucción de una imagen a partir de las coordenadas tridimensionales de las aristas de los objetos.

* **Procesamiento dato-dato.** Cuando la entrada al proceso requiere de información de tipo dato y la salida del mismo es una información de tipo dato, por ejemplo la determinación de las coordenadas del centro de un objeto representado por las coordenadas de sus aristas.

Eventualmente, se pueden presentar procesos que combinen imágenes y datos, tanto en las entradas como en las salidas.

El estudio del video puede considerarse como el procesamiento de una serie de imágenes, cuyo resultado será una serie de imágenes o datos. A la entrada del proceso se tiene una serie de imágenes y a la salida se pueden tener datos como las coordenadas de los objetos, los parámetros que definen el movimiento de los objetos y, en algunos casos, una serie de datos que definen el entorno de un espacio tridimensional. Las operaciones que se realizan sobre imágenes y que tienen como resultado otra imagen se pueden clasificar en tres grupos:

* **Operaciones puntuales.** Son aquellas en las que el valor de cada píxel $p_B(x,y)$ de la imagen resultante, se obtiene a partir del valor del píxel $p_A(x,y)$ de la imagen original, sin involucrar otros píxeles. Como ejemplos se tienen el realce de contraste, la modificación del histograma y el cambio de color.

* **Operaciones locales.** Son aquellas en las que el valor de cada píxel $p_B(x,y)$ de la imagen resultante, se obtiene a partir del valor del píxel $p_A(x,y)$ y de sus vecinos de la imagen original. El concepto puede ser extendido al campo 3-D, incluyendo la secuencia temporal de las imágenes. Entre las operaciones locales básicas se pueden mencionar las convoluciones, las operaciones morfológicas y las operaciones booleanas. Como ejemplos se pueden señalar los filtros por convolución, los detectores de bordes y los uniformizadores de textura.

* **Operaciones globales.** Son aquellas en las que el valor de cada píxel $p_B(x,y)$ de la imagen resultante, se obtiene a partir del valor de todos los píxeles de la imagen original. Como ejemplos se tienen las transformadas de Fourier y del Coseno.

2.2.2 Información de movimiento.

La información del movimiento 3-D de los objetos y el movimiento de la cámara tienen una gran importancia en el procesamiento de imágenes y video. Desde el punto de vista del procesamiento de señales, las trayectorias de movimiento representan la dirección espacial-temporal de los cambios de intensidad en la secuencia de imágenes. Desde el punto de vista de las mediciones, el movimiento 2-D en la secuencia de imágenes y video es una observación que incorpora información sobre la geometría y movimiento 3-D en el mundo real [Reed04].

Existen numerosas aplicaciones que incluyen estimación de movimiento, entre las que se encuentran la codificación de video, donde la información de movimiento sirve para eliminar la redundancia espacial-temporal de los datos de video; el filtrado digital, donde los procesos de reducción de ruido, interpolación y restauración pueden ser orientados en la dirección de las trayectorias de movimiento para una mayor eficiencia; el análisis de escena, en la cual la correspondencia de la posición de imágenes sobre el tiempo, que es establecida con la información 2-D, sirve para la medición de cantidades en 3-D como forma, desplazamiento, dirección, posición y velocidad de los objetos; el montaje y registro de imágenes, donde la información de movimiento es extraída de la secuencia de imágenes para producir imágenes con campos de vista agrandados o con resolución espacial mejorada.

El sistema de visión humano posee la habilidad de extraer la información de movimiento en forma instantánea y con una aparente facilidad. En contraste, los sistemas de visión artificial dedican gran parte de sus recursos y tiempo de cómputo a la

estimación de movimiento y aun cuando se han propuesto una gran variedad de algoritmos, este es un tema actual de investigación que depende del tipo de aplicación y los métodos de procesamiento asociados [Shi99].

2.2.2.1 Formación del movimiento.

La fuente más específica para la formación de movimiento visible en secuencias de imágenes y video, es el movimiento 3-D de un punto físico, proyectado en el plano de imagen [Reed04]. Si se considera el movimiento de un punto sobre el tiempo con respecto a un sistema de coordenadas 3-D, entonces su posición 3-D, como una función del tiempo t, describe una trayectoria $P(t) = (X(t), Y(t), Z(t))^T$. Sobre esta trayectoria, el movimiento entre los instantes de tiempo t y t+1 puede ser definido como

$$D(t) = P(t+1) - P(t) \tag{2.7}$$

Considerando ahora un sistema de adquisición de imágenes, que proyecta la escena en su plano de imágenes $p = (x,y)^T$ como

$$p(t) = \pi(P(t)) \tag{2.8}$$

donde $\pi(\cdot)$ denota proyección. Se observa que se pierde al menos una dimensión de la información de la escena espacial, por la proyección del mundo 3-D en el plano de imágenes 2-D; muchos de los desafíos del procesamiento de imágenes y video se originan por esta pérdida. La recuperación de la información perdida, hasta cierto grado, establece un problema inverso a menudo incompletamente definido.

Cuando se asume que el punto bajo consideración no experimenta oclusión en cualquiera de sus instantes de tiempo, el vector de movimiento 3-D es implícitamente proyectado sobre el plano de imagen como un vector de movimiento 2-D (2.9) (Figura 2.6).

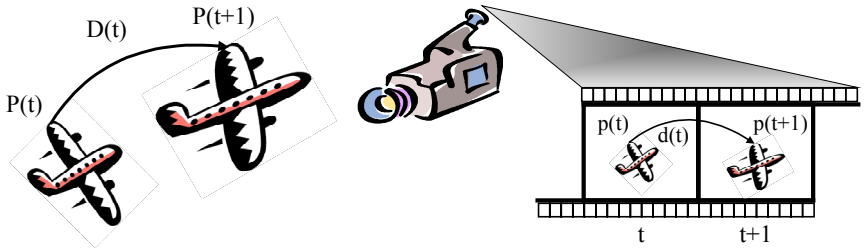


Figura 2.6. Proyección del movimiento 3-D sobre el plano de imágenes.

$$d(t) = \pi(P(t+1)) - \pi(P(t)) \tag{2.9}$$

En la práctica, el interés no es solo el movimiento de un solo punto, sino el movimiento de todos los puntos visibles en la primera imagen, donde se puede obtener un campo de vectores de movimiento d(p,t), el cual es bien definido por todas las posiciones espacio-tiempo de la imagen $p = \pi(P(t))$ que proyectan puntos 3-D que no experimentan obstrucción en tiempo t+1. Las posiciones de la imagen con movimiento 2-D indefinido son referenciadas como área de oclusión.

2.2.2.2 Modelos de estimación de movimiento.

En la literatura se reportan numerosos enfoques de estimación de movimiento, los cuales en general emplean uno o más de los siguientes modelos [Reed04]:

* **Modelo de observación.** Este modelo describe como interactúan el movimiento y la imagen, haciendo referencia a alguna característica de la imagen que permanece casi invariante a lo largo de las trayectorias de movimiento, como gradiente espacial, curvatura, color o justamente la intensidad de la imagen en sí misma. Por ejemplo, los métodos basados en gradiente especifican que las intensidades son constantes a lo largo de las trayectorias de movimiento, ignorando fenómenos como oclusión, fluidez y variaciones de la intensidad debido a cambios en la orientación e iluminación.

* **Modelo del campo de movimiento.** En esta clasificación se tienen modelos parametrizados y aleatorios, caracterizados por restringir en mayor o menor medida los grados de libertad espacial-temporal del movimiento. Los primeros definen que el campo de movimiento puede ser descrito por un conjunto finito de parámetros. El modelo paramétrico mejor conocido asume un movimiento de translación 2-D, describiendo el campo de movimiento con solo dos parámetros $d(p) = d$. Otros modelos incluyen parámetros adicionales con un significado físico, tales como rotación, corte y *zoom*. Mientras los modelos parametrizables imponen fuertes restricciones a los campos de movimiento, los modelos probabilísticos especifican restricciones suaves, al modelar los campos de movimiento como campos aleatorios cuya probabilidad se incrementa con la tersura de la imagen. Frecuentemente, los campos aleatorios Gibbs-Markob se utilizan para este propósito.

* **Modelo de región.** Modelo que describe la región de validez para un campo de movimiento particular. Se pueden utilizar regiones de tamaño y posición fijas, por ejemplo bloques, o regiones que se adapten a la forma de los objetos de movimiento independiente. Esto último permite una descripción del movimiento mas apropiada, gracias a la posibilidad de asignar y manipular propiedades significativas de los objetos, al coste de incrementar la complejidad de cómputo.

2.2.2.3 Método de ajuste de las regiones de intensidad.

El ajuste de regiones de intensidad es el método mas aplicado en el análisis de movimiento, debido a su fácil realización y alta eficiencia, en campos de movimiento grandes y pequeños.

En este método, una imagen es subdividida en regiones de tamaño pequeño (a menudo bloques rectangulares), tal que se pueda asumir un vector de desplazamiento (MV) para cada región. Las muestras de intensidad de la región A de una primera imagen son comparadas con las muestras de una región B del mismo tamaño, pero con un desplazamiento de traslación MV, en una segunda imagen. Los resultados de la diferencia entre las dos regiones A y B son simétricos ($d(A,B) = d(B,A)$) y no-negativos ($d(A,B) \geq 0 \; \forall \; A,B$), con $d(A,B) = 0 \Leftrightarrow A = B$.

Los métodos de ajuste de intensidad asumen que todos los píxeles en la región experimentan la misma traslación, lo cual no modela adecuadamente el movimiento, sobre todo el no-rígido. Esto es un dilema para la selección del tamaño del bloque: los

bloques grandes pueden representar pobremente el movimiento real; los bloques pequeños tal vez no incluyan suficiente información para una identificación única. En el último caso, la estimación de movimiento resulta ambigua si existe un gran número de bloques similares.

2.3 Codificación de video, técnicas y estándares.

La codificación es la ciencia de reducir la cantidad de datos utilizados para almacenar y transmitir información. Su desarrollo se apoya en el hecho que la información, por su naturaleza, no es aleatoria, ya que presenta orden y regularidad. Si el orden y la regularidad se determinan, se pueden eliminar la redundancia y la irrelevancia de la información y representarla con los datos mínimos suficientes para su posterior reconstrucción [Symes01]. Se debe tomar en cuenta que los datos y la información son conceptos relacionados pero diferentes. Los datos representan la información. Si la información es definida como el conocimiento, los datos son la representación del conocimiento [Shi99].

Existen diversas técnicas de compresión, que pueden utilizarse en forma individual o secuencial, para lograr la mayor reducción de datos. Algunas técnicas no producen pérdidas, ya que la compresión se realiza removiendo la información redundante, la cual puede ser recreada por los datos que permanecen, por lo que el proceso de compresión se puede revertir con exactitud y precisión. En otras técnicas, la compresión no se puede revertir totalmente, porque el proceso se realiza con pérdidas, eliminando la información irrelevante para el usuario final, la cual no se puede ni se quiere recuperar. Un ejemplo de la aplicación de algunas técnicas para la reducción de la información redundante e irrelevante de las señales digitales, lo muestra el diagrama de Schouten (Figura 2.7) [Chen02].

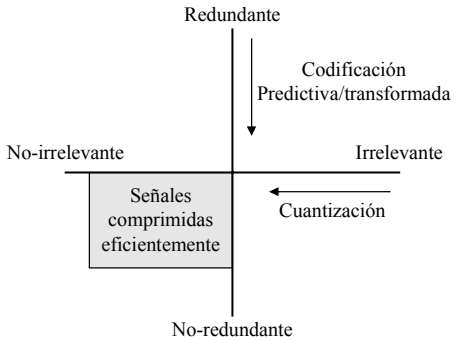


Figura 2.7. Diagrama de Schouten, que muestra la aplicación de algunas técnicas para la reducción de redundancia e irrelevancia de las señales digitales.

Las técnicas con pérdidas proporcionan tanto buena razón de compresión como adecuada reducción de la tasa en bits, a un coste mínimo. Se aplican cuando la información irrelevante eliminada no es totalmente necesaria y cuando el usuario final no percibe la reducción de la calidad por las mismas causas, como en la transferencia de audio y video. Sin embargo, la compresión con pérdidas no es simple, ya que las características dinámicas de los datos generan un gran número de opciones de realización, que hace difícil la medición de los resultados y de la calidad de la compresión.

2.3.1 Introducción a la codificación de video.

El rápido crecimiento de los servicios de comunicación digital ha generado un gran interés tecnológico y comercial en la transmisión de señales de video. Ya que las fuentes de video digital requieren una tasa de bits muy alta, que rebasa las capacidades de los sistemas hardware, el uso de algoritmos de codificación de video es una obligación [Shi99].

La compresión de datos de video hace referencia a un proceso en el cual se reduce la cantidad de datos utilizados en la representación del video, para satisfacer el requerimiento de tasa en bits, mientras la calidad del video reconstruido o recuperado satisface los objetivos de la aplicación. La figura 2.8 muestra la funcionalidad de la compresión de datos de la imagen y video en la transmisión y el almacenamiento de datos.

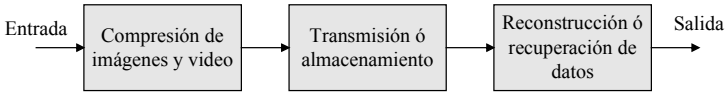


Figura 2.8. Compresión de imágenes y video para transmisión y almacenamiento de información visual.

La calidad requerida de la información reconstruida depende de la aplicación. Por ejemplo, en diagnósticos médicos y algunas mediciones científicas, las imágenes y videos reconstruidos necesitan ser idénticos al original, por lo que solo se permiten algoritmos reversibles, sin pérdidas, que preserven la información, a diferencia de aplicaciones como televisión, donde se permite una pérdida de cierta cantidad de información.

En el caso particular del procesamiento de video, la compresión con pérdidas produce varios cambios en las características de los cuadros, cada uno con una relación compleja, no lineal a la calidad subjetiva. El proceso de compresión con pérdidas aplicado al video genera dos efectos principales: objetos que deben estar en los cuadros pero que se pierden y objetos que están en los cuadros (artefactos) pero que no deben estar ahí. Las áreas donde se pueden esperar pérdidas incluyen la resolución espacial (luminancia y color) y los detalles de contraste. Los artefactos que pueden ser agregados incluyen bordes de bloques, ruido de mosquito, ruido de cuantización, escalones en la escala de grises, etc. [Symes01].

Una característica importante en la selección de un sistema de compresión de video es el grado de simetría o asimetría de la aplicación. Los sistemas simétricos, mantienen un equilibrio en la complejidad y coste de los procesos de compresión y descompresión, ya que el nivel de utilización de ambos es similar. Por ejemplo, en las video-conferencias, cada estación debe estar preparada para comprimir y descomprimir el video con la misma habilidad, por lo que el sistema es simétrico. El modelo asimétrico se tiene cuando un compresor alimenta varios descompresores, por lo que la complejidad y calidad del primero puede ser mucho mayor que la de los segundos. Los ejemplos se presentan en la industria de la televisión, donde un solo transmisor difunde la señal a un gran número de receptores y en la industria del DVD, donde un sistema de compresión prepara los datos para producir la versión maestra, con la que se graban los discos que se descomprimen en los equipos de los usuarios [Symes01].

El video digital, como una secuencia de cuadros, contiene una gran cantidad de datos. Por ejemplo, si se desea transmitir video en formato CIF (352×288 píxeles), espacio de color RGB, con 8 bits por píxel, 30 fps, sobre una línea telefónica, con un MODEM V90, el cual transfiere información a un máximo de 57 600 bps, la tasa en bits requerida es de 352×288×3×8×30 = 72 990 720 bps. Por lo tanto, de acuerdo a la tasa en bits requerida y la tasa disponible, se necesita comprimir el video 1290 veces para que el medio físico soporte la transmisión. Las necesidades actuales de servicios de video, como la televisión de alta definición (HDTV) hacen aun más necesaria la aplicación de las técnicas de compresión, las cuales son el enlace entre la gran cantidad de datos requeridos y las capacidades limitadas del hardware.

2.3.2 Redundancia en la señal de video.

La factibilidad de la compresión de video se apoya principalmente en la reducción de las redundancias que presenta la señal de video. Existen dos tipos principales, la estadística y la psicovisual, las cuales se sub-clasifican para cubrir la mayor parte del espectro de posibilidades de compresión (Figura 2.9) [Shi99].

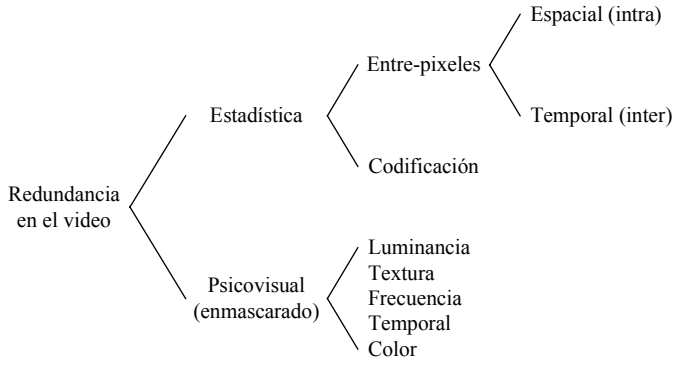


Figura 2.9. Clasificación de la redundancia en el video.

2.3.2.1 Redundancia estadística.

La redundancia estadística se clasifica en dos tipos: redundancia entre-píxeles y redundancia de codificación. La redundancia entre-píxeles se basa en que los píxeles de un cuadro de video y los píxeles de un grupo de cuadros sucesivos no son estadísticamente independientes, ya que presentan diversos grados de dependencia (correlación). La redundancia entre-píxeles se divide en dos categorías, redundancia espacial y redundancia temporal. La redundancia de codificación define la redundancia estadística asociada con las técnicas de codificación.

2.3.2.1.1 Redundancia espacial.

La redundancia espacial, también llamada redundancia intra-cuadros, representa la correlación estadística entre los píxeles de un cuadro de video. Es bien conocido que los valores de intensidad de los píxeles a lo largo de un renglón (ó una columna), tienen una correlación cercana a su valor máximo con los píxeles a lo largo del mismo renglón (ó columna), pero con un desplazamiento de un píxel. Uno de los primeros investigadores en estudiar las propiedades estadísticas del video fue Kretzmer, en los años 1950's, el cual después de varios experimentos encontró que la auto-correlación de

los píxeles de un cuadro de video en ambas direcciones horizontal y vertical exhiben rendimiento similar, con curvas de auto-correlación de cuadro a cuadro entre lineales y exponenciales y sin que la simetría central con respecto al eje vertical y la distribución tipo campana sufran cambios.

La redundancia espacial implica que el valor de intensidad de un píxel puede ser estimado a partir de sus píxeles vecinos, lo que hace referencia a que los píxeles de un cuadro de video no son independientes. La consecuencia directa del reconocimiento de la redundancia espacial es que al eliminarla, se puede representar un cuadro de video con menos datos, con una eficiencia similar a la compresión de una imagen fija.

2.3.2.1.2 Redundancia temporal.

La redundancia temporal, también llamada redundancia inter-cuadros, concierne a la correlación estadística entre píxeles de cuadros consecutivos de una secuencia de video. Esto permite predecir un cuadro a partir de los cuadros vecinos en la dimensión temporal. A partir de la década de los 1980s, la predicción inter-cuadros más eficiente utiliza análisis de movimiento, lo cual considera que los cambios de un cuadro a otro son debidos principalmente al movimiento de algunos objetos. La aplicación de este método, llamado formalmente codificación predictiva temporal con movimiento compensado, ha sido un factor vital para el creciente uso del video en medios digitales, gracias a los niveles de compresión que se obtienen con su uso.

2.3.2.1.3 Redundancia de codificación.

La redundancia de codificación busca cancelar la redundancia en la representación de la información, es decir, en la forma en que se codifican los datos resultantes al eliminar la redundancia de la información. Algunas de las técnicas que se aplican son los métodos de codificación de longitud variable, como el aritmético y el de Huffman.

2.3.2.2 Redundancia psicovisual.

La redundancia psicovisual hace referencia a las características del sistema visual humano “human visual system” (HVS), el cual percibe el mundo exterior de una forma compleja, con una respuesta no-lineal a la presencia de algunos atributos físicos del estímulo, como la intensidad y el color. En el HVS, la información visual se aprecia de diferentes formas: cierta información puede ser más importante que otra, por lo que si se utilizan menos datos para representar la información visual menos importante, la percepción no se afecta. Bajo esta perspectiva, se define que parte de la información visual es psicovisualmente redundante, por lo que su eliminación trae consigo la compresión de video.

Los aspectos del HVS que están relacionados con la compresión de video son los enmascarados dependientes de la imagen (luminancia y textura) y los independientes a ella (frecuencia, tiempo y color). Estas características de enmascarado del HVS se unifican en la que parece ser la más importante característica de la percepción visual humana: la sensibilidad diferencial.

El término enmascarado se refiere a la interacción destructiva o interferencia entre estímulos que están muy cercanamente acoplados en tiempo y espacio, lo que puede resultar en fallas de detección o errores de reconocimiento, por lo que es importante analizar la detectabilidad de un estímulo cuando otros están presentes. Un concepto importante en el enmascarado es el contraste mínimo o umbral crítico mínimo en la escala de grises entre los objetos y el fondo de una imagen, el cual se calcula como la diferencia en los niveles de gris que permite que el HVS discrimine los objetos con un 50% de confiabilidad.

2.3.2.2.1 Enmascarado en luminancia.

El enmascarado en luminancia es el aspecto del HVS que concierne a la percepción de la brillantez, en el que se estudian las condiciones de contraste que deben cumplir los objetos y el fondo de un cuadro de video para poder ser detectados, de acuerdo al umbral crítico en la escala de grises. Esto significa que la sensibilidad del ojo a un estímulo depende de la intensidad de otro estímulo. El impacto directo que el enmascarado de la luminancia tiene en la compresión de video se relaciona con el proceso de cuantización, su influencia en la tasa en bits resultante y la calidad del video reconstruido. El enmascarado en luminancia sugiere un esquema de cuantización no-uniforme que tome en consideración la función de sensibilidad al contraste.

2.3.2.2.2 Enmascarado en textura.

El enmascarado en textura muestra que el ojo humano es más sensible a las regiones suaves que a las regiones con textura, donde la intensidad exhibe una alta variación. Su estudio establece que el umbral de discriminación se incrementa cuando aumentan los detalles de la imagen, es decir, a una mayor textura, un mayor umbral de discriminación. El enmascarado en textura, aplicado a la compresión de video, indica que se puede obtener una optimización en la tasa en bits, cuando el número de niveles de cuantización se adapta a las variaciones en intensidad de las regiones de una imagen.

2.3.2.2.3 Enmascarado en frecuencia.

El enmascarado en frecuencia es una característica independiente de la imagen, que establece que el umbral de discriminación se incrementa a la par que la frecuencia, lo que se considera un proceso de dependencia que no altera la percepción visual, gracias al funcionamiento del ojo humano como un filtro pasa-bajos. La aplicación del enmascarado en frecuencia en la compresión de video se realiza en el dominio de la transformada. Por ejemplo, varios codificadores de video utilizan una etapa de transformada discreta del coseno (DCT), en la cual se cancelan los coeficientes de alta frecuencia de magnitud pequeña, sin afectar significativamente la percepción del HVS.

2.3.2.2.4 Enmascarado temporal.

El enmascarado temporal se origina por el tiempo que el HVS tarda en adaptarse a los cambios abruptos de la escena. Durante la transición, el HVS no es sensible a los detalles. El enmascarado toma lugar antes o después del cambio, por lo que se debe tomar en cuenta cuando se distribuyen los datos en la codificación de video.

2.3.2.2.5 Enmascarado en color.

La luz visible corresponde a una distribución del espectro electromagnético. Por lo tanto un color, como una sensación de luz visible, es una energía con una intensidad y un conjunto de longitudes de onda asociadas al espectro. La intensidad es un atributo de la luz visible. La composición de las longitudes de onda, la croma, es otro atributo. El atributo croma se compone de dos elementos: el tinte y la saturación. El tinte de un color es caracterizado por la longitud de onda dominante de la composición. La saturación es una medida de la pureza del color: un color puro tiene una saturación del 100 %, la saturación del blanco es 0 %. El espacio de color RGB es el que más se aproxima a la característica humana de la percepción del color, ya que el área sensible al color del HVS consiste de tres diferentes conjuntos de conos, cada grupo sensible a uno de los colores primarios rojo, verde y azul, por lo que cualquier color recibido por el HVS puede ser considerado como una combinación lineal particular de los mismos.

Sin embargo, en el procesamiento de la señal de color, incluyendo la compresión de video, el espacio RGB no es muy eficiente. Esto tiene que ver con la percepción del color en el HVS: el ojo humano es más sensible al verde que al rojo y menos sensible al azul, por lo que la representación de los colores primarios con el mismo peso, no es lo mejor cuando el HVS es el usuario final. La especificación de los datos de color en función de la percepción del HVS, hace más eficiente la codificación del video, lo cual se realiza con más sencillez en el espacio de color luma-croma, donde el componente luma se identifica con la percepción de brillantez y los componentes croma con la percepción de tinte y saturación del color.

El impacto del enmascarado en color en la codificación de video se refleja en la resolución de los componentes luma y croma: se pueden asignar mas bits al componente luma que a los croma, en una relación 2 a 1, sin que se altere proporcionalmente la apreciación del video. Este método de compresión ha sido adoptado en los estándares internacionales de codificación JPEG y MPEG.

2.3.3. Técnicas de codificación de video.

Los sistemas de codificación de video utilizan un conjunto de técnicas que normalmente son ejecutadas en cascada, para proporcionar una solución de compresión óptima. Algunas de ellas no comprimen la información de video, sino solamente preparan los datos para las etapas posteriores. Otras producen pérdidas de información, ya que no son totalmente reversibles. Algunas más eliminan la información irrelevante, otras la estructura, y muchas la redundancia. Pero en su conjunto, las técnicas de codificación agrupan los elementos que han revolucionado el almacenamiento y transmisión de información visual, por lo que a continuación se hace una referencia conceptual a las de mayor importancia.

2.3.3.1 Codificación predictiva.

Las técnicas de codificación predictiva procesan la diferencia entre la señal misma y su predicción, en vez de procesar la señal directamente, por lo que también son conocidas como técnicas de codificación diferencial [Shi99]. Al utilizar la correlación temporal y/o espacial entre píxeles y datos, la codificación predictiva es una técnica eficiente y computacionalmente simple. Cuando la señal diferencial (también conocida

como error de predicción) es cuantizada, la codificación diferencial toma el nombre de modulación de codificación de pulsos diferencial (DPCM). Los sistemas de codificación diferencial consisten de dos componentes principales: predicción y cuantización. A continuación se documentan los elementos del componente de predicción que tienen como objetivo reducir la redundancia temporal, espacial y de codificación, por medio de la creación de modelos de predicción de la información de video [Richarson03].

2.3.3.1.1 Predicción temporal.

La predicción temporal o inter-predicción, tiene como objetivo reducir la redundancia temporal, a partir de un modelo de predicción y de la compensación de diferencias (movimiento), entre uno o más cuadros pasados y/o futuros (cuadros de referencia) y el cuadro actual. El resultado de la inter-predicción es un cuadro de residuos con la menor energía posible, creado al restar el cuadro actual menos la predicción, además de un conjunto de vectores que describen la compensación de movimiento. El método más simple de predicción temporal es la utilización del cuadro previo como modelo de predicción del cuadro actual. Su inconveniente es que una gran cantidad de energía permanece en el cuadro de residuos, lo que significa que aun existe un exceso de información redundante.

Se puede realizar una mejor predicción si se considera la diferencia entre ambos cuadros, causada por el movimiento de los objetos, el movimiento de la cámara, las regiones no cubiertas y los cambios de iluminación. Con excepción de las regiones no cubiertas y los cambios de iluminación, las cuales son difíciles de predecir, las diferencias corresponden al movimiento de los píxeles entre los cuadros, por lo que se puede obtener un modelo de predicción más exacto si se estima la trayectoria de cada uno de ellos.

La estimación de la trayectoria de cada píxel, conocida como flujo óptico, es un proceso computacionalmente intensivo que genera gran cantidad de datos, por lo que no es un método adecuado de compensación de movimiento. Otros métodos, más prácticos, consideran la estimación del movimiento en bloques de píxeles o por regiones de la imagen. El método más ampliamente usado es el de compensación de movimiento de una sección rectangular o bloque de M×N píxeles del cuadro actual, utilizando el algoritmo de ajuste de bloques.

2.3.3.1.2 Predicción espacial.

La predicción espacial o intra-predicción, tiene como objetivo reducir la redundancia espacial presente en los cuadros de video, por medio de la estimación de la intensidad de un píxel o de un grupo de píxeles, a partir de una combinación de los píxeles vecinos previamente codificados y del cálculo de la diferencia entre el valor original y el de predicción. La intra-predicción se apoya en la correlación estadística de los píxeles de los cuadros de video.

2.3.3.1.3 Predicción de codificación.

Ya que algunos símbolos están altamente correlacionados en las regiones locales de una imagen, la eficiencia de codificación se puede mejorar al predecir los elementos

de la región actual con los datos codificados previamente y codificar la diferencia entre la predicción y los valores actuales.

Un modelo de la predicción de codificación, se observa en los vectores de movimiento del algoritmo de ajuste de bloques. Los vectores de bloques vecinos están normalmente correlacionados, ya que el movimiento de los objetos puede extenderse a lo largo de grandes regiones del cuadro. La compresión del campo de vectores puede ser mejorada, al predecir cada vector de movimiento, con los vectores de los bloques vecinos previamente codificados. Por ejemplo, la predicción del vector de movimiento del bloque X (MVP_X), se puede calcular con los vectores de los bloques vecinos A, B y C (Figura 2.10), es decir $MVP_X = f(MV_A, MV_B, MV_C)$. Una vez calculado el vector del bloque X (MV_X), se obtiene una mayor compresión si se codifica la diferencia de vectores $MVD = MV_X - MVP_X$.

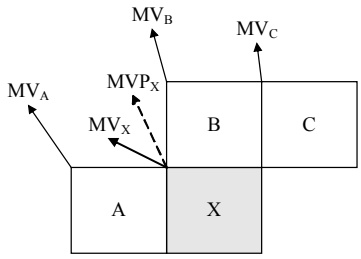


Figura 2.10. Predicción de codificación del vector de movimiento del bloque X.

2.3.3.2 Transformación.

El objetivo de la etapa de transformación de un codificador de video es cambiar el dominio de los datos residuales de la compensación de movimiento, a un dominio donde los datos sean representados por coeficientes de la transformada, en los cuales es más factible la eliminación de la redundancia espacial, facilitando su codificación [Guan01].

Una transformada es una regla matemática que puede ser usada para convertir un conjunto de valores en otro conjunto diferente, creando una nueva forma de representar la misma información, en un proceso que puede ser con pérdidas o sin ellas. Los sistemas de compresión con más éxito a la fecha se basan en transformadas que se mueven entre el dominio del tiempo o espacio y el dominio de la frecuencia o frecuencia espacial [Symes01]. La base fundamental de estas transformadas es el teorema de Fourier, el cual establece que cualquier función periódica puede ser representada por la suma de las amplitudes y fases apropiadas de ondas sinusoidales, de una frecuencia y de múltiplos enteros de esa frecuencia.

La selección de la transformada en los sistemas de codificación de video depende de los siguientes criterios [Richarson03][Guan01]:

1. La transformada debe de-correlacionar los datos (separarlos en componentes con mínima interdependencia) y compactarlos (la mayoría de la energía de los datos transformados se debe concentrar en un número pequeño de valores, en la región de baja frecuencia).

2. La transformada debe ser reversible.

3. La transformada debe tener un bajo coste computacional (mínimos requerimientos de memoria, uso de aritmética de baja precisión, bajo número de operaciones, etc.).

Varias transformadas han sido propuestas para la compresión de video. Las más populares se ubican en dos categorías: las que se basan en bloques y las que procesan imágenes completas [Richarson03]. Algunos ejemplos de las transformadas basadas en bloques son la transformada Karhunen-Loeve (KLT), la descomposición de valor singular (SVD) y la muy popular transformada DCT, las cuales se aplican en bloques de N×N píxeles, por lo que las imágenes se procesan en unidades de bloque. Estas transformadas tienen requerimientos mínimos de memoria y son adecuadas para la compresión de los residuos de la compensación de movimiento, pero tienden a producir artefactos en los bordes de los bloques. Las transformadas basadas en imágenes procesan toda o una gran sección de la imagen o cuadro. La más popular es la transformada discreta Wavelet (DWT), la cual muestra un excelente rendimiento en la compresión de imágenes fijas, pero tiene un alto requerimiento de memoria y no es muy compatible con la compensación de movimiento basada en bloques.

2.3.3.3 Cuantización.

La cuantización es el proceso de convertir una señal con un rango de valores X, en otra señal con un reducido rango de valores Y, donde el menor número de posibles valores de la señal cuantizada permite representarla con menos bits que la señal original, produciendo un efecto de compresión de datos. El cuantizador es un componente necesario en la codificación y tiene un impacto directo en la tasa en bits y en la distorsión del video reconstruido [Shi99].

El proceso de cuantización se realiza en ambos sentidos, en forma directa para cuantizar los datos de entrada y en forma inversa, para recuperar los valores originales a partir de los datos cuantizados, operación que no tiene total exactitud, al ser la cuantización una técnica de compresión con pérdidas. La cuantización puede ser usada para reducir la precisión de los datos de la imagen después de aplicar una transformada y eliminar valores insignificantes, como los coeficientes cercanos a cero [Richarson03].

Existen dos tipos principales de cuantizadores, escalares y vectoriales. Los escalares mapean un valor de entrada a un valor de salida. Los vectoriales mapean un grupo de muestras de entrada a un grupo de valores cuantizados, operación que puede ser usada como un método independiente de compresión. Los cuantizadores escalares se clasifican en uniformes y no-uniformes.

Los cuantizadores uniformes son los más simples y a menudo los más efectivos. En este componente, ambos niveles de decisión y reconstrucción están espaciados uniformemente, por lo que pueden ser comparados con una función escalera de igual tamaño de pasos (Figura 2.11).

En el diseño de los cuantizadores uniformes, el tamaño del paso de cuantización es usualmente el único parámetro que necesita ser especificado [Shi99], por lo que su valor es crítico, ya que controla la relación entre la eficiencia de compresión y la calidad del video. En un codificador de video en tiempo real, puede ser necesario modificar el

tamaño del paso, por ejemplo, para alterar la razón de compresión, en orden de igualar la tasa en bits de salida con la tasa del canal de transmisión [Richardson03]. Si el tamaño de paso es grande, el rango de valores cuantizados es pequeño, por lo que pueden ser representados eficientemente durante la transmisión (compresión alta), pero la reconstrucción de los valores originales, con la cuantización inversa, se ejecuta con un exceso de error. Si el tamaño del paso de cuantización es pequeño, los valores de la cuantización inversa son muy aproximados a la señal original, pero el rango amplio de los valores cuantizados reduce la eficiencia de compresión.

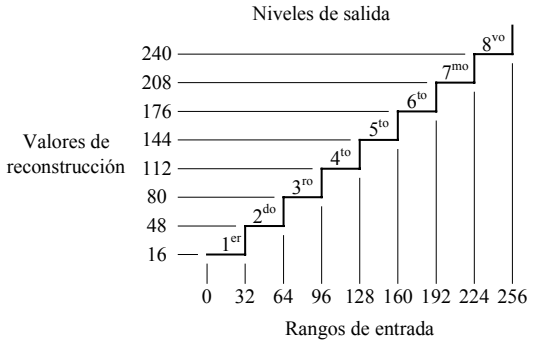


Figura 2.11. Ejemplo de un cuantizador escalar uniforme.

Los cuantizadores escalares no-uniformes toman en cuenta que la distribución de los valores de entrada no es regular, sino que puede ser irregular o presentar una distribución de probabilidad, por lo que en el diseño del cuantizador, los rangos de los valores de entrada se definen en función de la concentración de los mismos, evaluando su calidad a partir del error cuadrático medio, para un dado número de niveles de salida. Ejemplos de ellos son el cuantizador Lloyd-Max y el cuantizador de restricción de entropía [Symes01].

2.3.3.4 Reordenamiento.

En un codificador de video basado en transformadas, la salida del cuantizador es una matriz que conteniene algunos coeficientes diferentes de cero y un gran número de coeficientes cero. La agrupación de los coeficientes diferentes a cero (reordenamiento) y la representación eficiente de los coeficientes cero, define otro método de codificación, que se puede aplicar previamente a la codificación de entropía.

Por ejemplo, el resultado de la transformada DCT de un bloque de muestras residuales de imagen, es una matriz de coeficientes, con los coeficientes de baja frecuencia (los más significativos), alrededor del coeficiente DC ubicado en la posición (0,0). La cuantización remueve los coeficientes de alta frecuencia poco significativos, por lo que la matriz resultante contiene una gran cantidad de ceros. En el proceso de codificación, la matriz cuantizada se reordena por medio de su exploración en zig-zag diagonal (de arriba hacia abajo y de izquierda a derecha), iniciando en el coeficiente DC. Cada coeficiente cuantizado pasa a formar parte de un vector unidimensional, en donde los coeficientes diferentes a cero tienden a agruparse al inicio del vector, por lo que al final se tiene una gran secuencia de ceros.

El número de ceros en un vector de coeficientes cuantizados reordenados, puede ser codificado para compactar su representación, con la técnica más ampliamente utilizada: la codificación *run-length* (RLC). El termino *run* es usado para indicar la repetición de un símbolo, mientras que el termino *length* es usado para representar el número de símbolos repetidos, en otras palabras, el número de símbolos consecutivos del mismo valor [Shi99]. Con la técnica RLC, el vector unidimensional de coeficientes cuantizados se representa eficientemente como una serie de pares (run, level), donde *run* indica el número de ceros precediendo un coeficiente no-cero y *level* indica la magnitud del coeficiente no-cero.

2.3.3.5 Codificación de entropía.

La entropía es el proceso que convierte los datos que representan la información, en un grupo de símbolos que representan la misma información pero con menos datos, que se puede especificar como un flujo de bits adecuado para transmisión o almacenamiento.

Por definición, la entropía es una medida del desorden o la impredecibilidad [Symes01]. En los sistemas de información, el grado de impredecibilidad de un mensaje puede ser usado como una medida de la información contenida en el mensaje. Si el mensaje no contiene información, su grado de impredecibilidad es cero. La entropía de una fuente de información se especifica como la cantidad promedio de información que contiene cada símbolo a la salida de la fuente, en unidades de bits/símbolo. Ya que la probabilidad de cada elemento del mensaje determina su frecuencia, la información media por símbolo es la suma de la información de cada elemento multiplicada por su probabilidad.

El principio de la aplicación de la codificación de entropía en la compresión de información se apoya con un ejemplo, en el cual los datos son representados por valores entre 0 y 255 (bytes). Si se quiere transmitir un mensaje, se escoge un símbolo para representar cada posible valor y se transmite la secuencia de símbolos hasta que todos los valores se hayan enviado. La forma más simple de codificar el dato es enviando los bytes directamente, en donde cada símbolo es el mismo que el valor que representa, por lo que no hay distinción entre el valor y el símbolo. Pero esto no es eficiente. Si ambos el transmisor y el receptor conocen y aplican las mismas reglas, se puede usar cualquier conjunto de 256 símbolos únicos para representar los 256 valores posibles, con la ventaja de que no todos los símbolos necesitan tener la misma longitud. Esta es la esencia de la codificación de entropía.

La codificación de entropía comprime la información al usar símbolos cortos para representar los valores que ocurren frecuentemente y símbolos de mayor longitud para representar los valores menos frecuentes. La relación entre los valores y los símbolos es conocida como la tabla de código. A menos que los datos de entrada tengan una distribución de valores muy aleatoria, en la aplicación de esta técnica, el total de los datos utilizados para los símbolos será menor que el total de los valores originales.

Existen varios algoritmos y esquemas de entropía, en función de la manera en que se construye el código, de la forma en que se maneja y actualiza la tabla de código, del método de estimación y actualización de los cambios de probabilidad de los

elementos de entrada, etc., siendo los más representativos los métodos de codificación de longitud variable (VLC) y la codificación aritmética.

2.3.4 Análisis tasa/distorsión en la codificación de video.

El análisis de tasa/distorsión (R/D) ha sido uno de los principales enfoques de investigación en la teoría de la información y las comunicaciones en las últimas décadas, desde el teorema de Shannon para la codificación de fuentes, hasta el modelado R/D de los sistemas modernos de codificación de video [He05].

En las aplicaciones de codificación de video, las principales restricciones para la compresión están en la forma de ancho de banda de transmisión y espacio de almacenamiento, lo cual determina la salida en la tasa de bits del codificador. Por lo tanto, la meta fundamental de estos sistemas es optimizar la calidad del video bajo la restricción de la tasa. Para este fin, han sido desarrolladas varias teorías R/D, que modelan la relación entre la tasa de bits de codificación y la distorsión de la señal.

Las teorías R/D describen el rendimiento de los sistemas de compresión de datos con pérdidas y responden la siguiente pregunta clave: ¿cuál es el mínimo número de bits necesarios para la compresión de la fuente de datos, para un nivel específico de calidad?. Se debe tomar en cuenta que los análisis R/D teóricos y analíticos se han realizado en fuentes y esquemas de codificación simples. Para fuentes complejas, como el video y sistemas sofisticados de codificación como MPEG-2, H.263, MPEG-4 y H.264, este tipo de análisis de comportamiento teórico es a menudo inviable, por la alta correlación de los datos de la fuente y por la difícil modelación matemática de las técnicas de predicción y representación de datos.

La dificultad del modelado matemático de las características de la fuente y del sistema de compresión, crea un hueco entre el análisis R/D y la optimización de la tasa y calidad de la codificación de video. Para llenar este hueco, en las pasadas dos décadas se han desarrollado un conjunto de técnicas de análisis R/D, control de tasa y algoritmos de optimización de calidad para sistemas prácticos de compresión.

2.3.4.1 Análisis R/D de sistemas prácticos.

En la codificación de video, los dos factores más importantes son la tasa en bits de codificación y la calidad de la imagen. La tasa en bits de codificación, denotada por R, determina el ancho de banda del canal de transmisión o el espacio de almacenamiento requerido para almacenar el video codificado. El error de reconstrucción introducido por la compresión con pérdidas, normalmente referido como distorsión, es denotado por D. En un sistema típico de codificación de video, ambos R y D son controlados por el parámetro de cuantización, denotado por q. El rendimiento R/D del codificador de video es caracterizado por las funciones tasa/cuantización (R/Q) y distorsión/cuantización (D/Q), denotadas por $R(q)$ y $D(q)$ respectivamente, las cuales son llamadas funciones R/D. El parámetro de cuantización q puede ser determinado a partir de las funciones R/D, para alcanzar las metas de tasa en bits o de calidad del video.

Existen dos enfoques básicos para el modelado R/D. El primero es el *enfoque predictivo*, en el cual la fuente y el sistema de codificación son descompuestos en

componentes cuyos modelos estadísticos son conocidos. Estos modelos individuales son entonces combinados para formar un modelo analítico del sistema de codificación. El segundo es el *enfoque operacional*, donde las funciones R/D son definidas por un procesamiento matemático de los datos observados. La predicción R/D es adecuada para codificación de video en tiempo real, mientras que el enfoque operacional puede ser usado en compresión de video y optimización de calidad fuera de línea.

El control y optimización R/D puede operar en varios niveles de compresión de video, llamados nivel de secuencia, nivel de cuadro y nivel de bloque. En general, las aplicaciones de compresión de video pueden ser clasificadas en tres categorías principales: video interactivo, flujo de un solo sentido y compresión fuera de línea.

En el video interactivo en tiempo real, tal como las video-conferencias, el retardo de extremo a extremo permitido es muy pequeño, a menudo menor a 100 ms. Por lo tanto el tamaño del *buffer* tiene que ser muy pequeño. En este caso, el principal objetivo del control R/D es igualar la tasa de bits de codificación con el ancho de banda del canal de transmisión. La optimización de calidad se realiza al nivel de bloque, ya que la optimización al nivel de cuadro requiere las estadísticas de varios cuadros, lo que puede introducir un mayor retardo de codificación.

En el flujo de video de un solo sentido, el sistema captura la información de video sobre el ambiente físico, lo comprime y lo transmite al destino remoto. En este caso, la aplicación a menudo tolera una cierta cantidad de retardo, por ejemplo, de algunos segundos. Bajo esta condición, el codificador puede respaldar un gran número de cuadros de video, para recoger las estadísticas de cada uno y utilizarlas para asignación de bits y optimización de calidad, por lo que se puede obtener un control de tasa más exacto y una mejor calidad de imagen que en el video interactivo.

El escenario ideal para el control y optimización R/D es la compresión de video fuera de línea, donde el codificador tiene acceso a la secuencia de video completa. En este caso, la codificación se realiza en dos pasos. En el primero, el codificador estima las curvas R/D de cada cuadro y recauda las estadísticas de toda la secuencia. En el segundo paso, el codificador ejecuta el control y optimización R/D global, para maximizar la calidad de todo el video.

2.3.4.2 Método de multiplicación de Lagrange.

Dentro de las herramientas matemáticas para la optimización R/D se encuentran el método de multiplicación de Lagrange, el cual es utilizado para convertir un problema de optimización restringido en uno sin restricciones [Guan01].

El método de multiplicación de Lagrange opera bajo el concepto de transferir una o más restricciones a la función objetivo a minimizar. En el contexto de la codificación de video, la función objetivo más común es la distorsión, siendo la restricción la tasa en bits. La solución que ofrece el método es sumar la restricción a la función objetivo. Matemáticamente, esta idea se establece de la siguiente forma: Si B es un modo de codificación que pertenece a todos los posibles modos S_B generados por un algoritmo dado y para una fuente de video específica y $R(\cdot)$ y $D(\cdot)$ son las funciones asociadas de tasa y distorsión, encontrar la solución óptima $B^*(\lambda)$ de la expresión:

$$\min_{B \in S_B} (D(B) + \lambda \cdot R(B)) \quad (2.10)$$

donde λ es un número real positivo, es equivalente a resolver el siguiente problema de optimización restringido:

$$\min_{B \in S_B} D(B), \text{ sujeto a: } R(B) \leq Rmax \quad (2.11)$$

Claramente, la solución óptima $B^*(\lambda)$ es una función de λ, el multiplicador de Lagrange. Es necesario notar que lo inverso no siempre es verdad. No todas las soluciones al problema de restricción pueden ser encontradas con la formulación no restringida. Es decir, se pueden tener valores de Rmax para los cuales λ no existe y por lo tanto, la minimización en (2.10) no es realizable. Ya que en la práctica, el problema de restricción necesita resolverse para un Rmax dado, un paso crítico en este método es seleccionar λ apropiadamente, así que $R(B^*(\lambda)) \approx Rmax$. La selección de λ puede ser ubicada como la determinación de la relación entre la tasa y la distorsión, lo cual es específico a la aplicación.

2.3.5 Calidad del video.

La calidad del video es un factor importante al trabajar con sistemas de compresión. Por ejemplo, al medir la eficiencia de dos algoritmos de compresión, la evaluación puede apoyarse en un patrón de calidad de video [Shi99]. Si ambos algoritmos presentan la misma calidad de reconstrucción de video, bajo las mismas condiciones de prueba, el algoritmo que requiera menos datos puede considerarse superior al otro. Alternativamente, para la misma cantidad de datos, el mejor algoritmo puede ser aquel que ofrece la mayor calidad en el video reconstruido. Cabe destacar que al comparar la operación de varios sistemas de compresión, la cantidad de datos y la calidad de reconstrucción no son los únicos criterios de rendimiento, ya que también se pueden consideran otros agentes como la complejidad de cómputo, el consumo de energía, el número de ciclos de reloj de ejecución, la cantidad necesaria de elementos de almacenamiento, etc.

Para la medición de la calidad de video existen dos criterios, uno es la medición subjetiva (utilizando observadores humanos) y el otro la medición objetiva (por medio de algoritmos), cada uno con ventajas y desventajas. En la práctica, se utiliza una combinación de ambos.

2.3.5.1 Medición subjetiva de la calidad.

La medición subjetiva de la calidad se refiere a la evaluación de la calidad del video reconstruido por observadores humanos, al ser estos los últimos receptores de los datos. Para esta medición, se prepara un conjunto de muestras de video con diferentes parámetros de codificación y se les pide a varios observadores que tasen las diversas secuencias de video con alguna medida de calidad subjetiva. Alternativamente, se les proporciona una escala de calidad para que asignen a cada secuencia de video el valor que ellos consideren adecuado, desde un nivel de calidad excelente, hasta el de mala calidad. Algunos procedimientos de prueba para la evaluación de la calidad subjetiva son definidos en la recomendación ITU-R BT.500.11 [Richardson03].

Un punto importante es que la evaluación subjetiva de la calidad es costosa, difícil e imprecisa, ya que existen varios factores que pueden afectar el resultado: se necesita un gran número de secuencias de video y observadores, la evaluación toma mucho tiempo porque el sistema visual humano se fatiga y aburre fácilmente, la medición es dependiente del ambiente donde se ubica el observador y de su estado de ánimo, el observador se puede convertir en un experto y alterar la medición. etc.

2.3.5.2 Medición objetiva de la calidad.

La medición objetiva de la calidad de video es aquella que se mide automáticamente, a partir de un algoritmo. La técnica más popular es la razón del pico de la señal al ruido (PSNR), pero su rigidez y la desviación de sus resultados con respecto a las evaluaciones subjetivas, han motivado el desarrollo de nuevos algoritmos, más apegados a la percepción visual humana. Ya que las propuestas no han logrado obtener resultados aproximados a los observadores humanos, el concepto PSNR sigue vigente.

Considérese un sistema de procesamiento de video donde f(x,y) es el cuadro de entrada y g(x,y) el cuadro de salida degradado por el procesamiento. Para evaluar la calidad de g(x,y), se define la función de error e(x,y) como la diferencia entre la entrada y la salida. El error medio cuadrado es definido como MSE (2.12), donde *M* y *N* son las dimensiones de la imagen en las direcciones horizontal y vertical.

$$MSE = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} e(x,y)^2 \qquad (2.12)$$

El término PSNR se define en una escala logarítmica y depende del error MSE, relativo al cuadrado del mayor valor de la señal en la imagen (2.13), donde *n* es el número de bits por píxel.

$$PSNR_{dB} = 10 \log_{10} \frac{(2^n - 1)^2}{MSE} \qquad (2.13)$$

La interpretación del PSNR es que mientras más grande sea su valor, mejor es la calidad de la imagen procesada g(x,y) y aun cuando no siempre proporciona una evaluación confiable de la calidad de la imagen, su evaluación es mucho más rápida y sencilla que la medición subjetiva, con un ingrediente extra, su repetibilidad.

Una aplicación del criterio de calidad PSNR es la comparación del rendimiento tasa/distorsión de varios codificadores de video, por medio de un grupo de gráficas que describen la tendencia entre la calidad y la tasa de bits. Graficando el valor medio del PSNR del componente luma *Y*, para una secuencia de video de entrada, contra diferentes valores de tasa de bits de codificación, produce una curva tasa/distorsión que caracteriza cada codificador (Figura 2.12). Cuando la tasa en bits disminuye, la calidad (valor PSNR) sufre un decaimiento no lineal. El mejor rendimiento lo tiene el sistema con una gráfica tasa/distorsión más hacia arriba y a la izquierda.

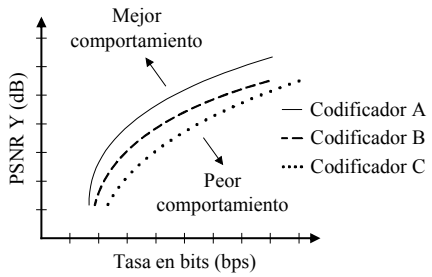


Figura 2.12. Ejemplo de curvas tasa/distorsión.

2.3.6 Modelo genérico para la codificación de video.

La mayoría de los estándares actuales de codificación de video se basan en un modelo genérico de procesamiento basado en bloques. Este incorpora funciones de estimación (ME) y compensación de movimiento (MC), transformación (T), cuantización (Q), reordenamiento y codificación de entropía, en una operación hibrida apoyada en los principios de reducción de las redundancias espacial, temporal y de codificación, de la eliminación de la información irrelevante en el codificador y de la reconstrucción de la información original en el decodificador (Figura 2.13).

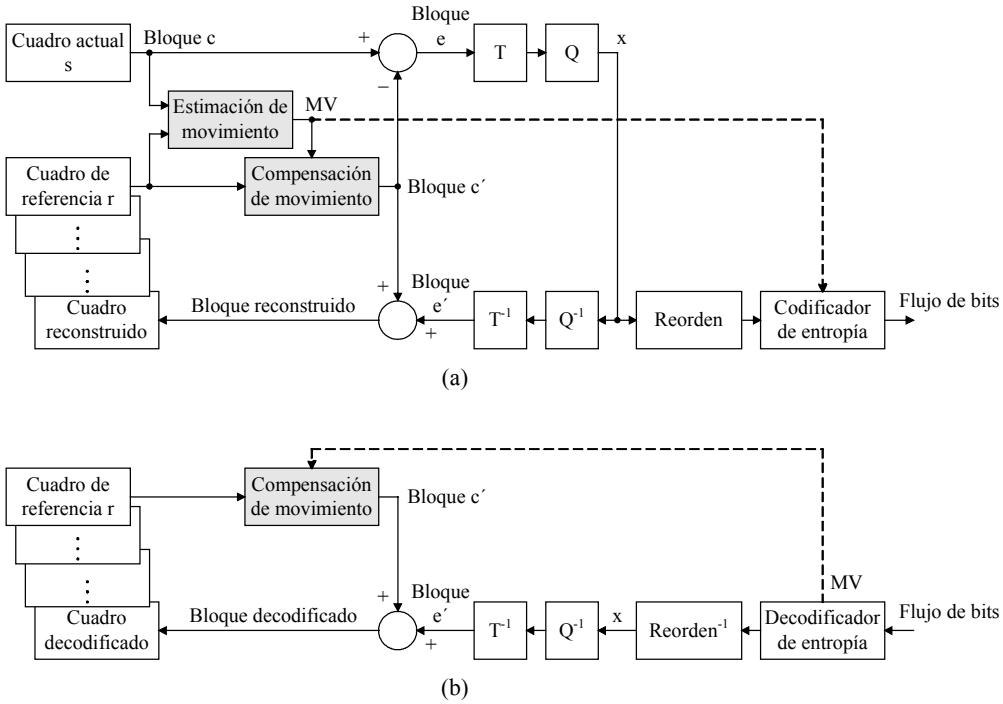


Figura 2.13. Modelo genérico de codificación de video, a) codificador y b) decodificador.

La primera etapa, que incluye las técnicas de estimación y compensación de movimiento (ME/MC) y transformación 2-D, tiene como objetivo tomar ventaja de la correlación temporal y espacial de la secuencia de video, en orden de optimizar el rendimiento R/D de las etapas posteriores: la cuantización y la codificación de entropía. La técnica más popular para ME/MC es el ajuste de bloques y la más popular de transformación espacial es la DCT. En el decodificador, todos los procesamientos

anteriores son invertidos exactamente, excepto la cuantización, en donde se tienen pérdidas de información.

El codificador recibe un cuadro de video a la vez (*s*), que es dividido y procesado en unidades de bloques *c*, para su comparación con los cuadros previamente decodificados (cuadros de referencia *r*). Por cada bloque *c*, se explora una región del cuadro *r*, para determinar el bloque del mismo tamaño que mejor lo iguala. La diferencia de posición entre el bloque actual y el bloque estimado (vector de movimiento, *MV*), se envía como información lateral al decodificador. La misma información de movimiento es utilizada para obtener el bloque compensado *c´*, el cual se resta del cuadro actual para obtener el bloque de diferencias, lo cual es propiamente el error de predicción *e*. El bloque de diferencias es transformado y cuantizado (*x*) y los coeficientes son reordenados para su codificación de entropía, junto con los vectores de movimiento y la información auxiliar, para generar el flujo de bits codificado.

Para que el proceso de predicción de la compensación de movimiento se realice con los mismos cuadros de referencia que se tendrán en el decodificador, los datos *x* son cuantizados y transformados inversamente para producir un bloque decodificado *é*, el cual es sumado al bloque compensado *c´* para producir un bloque reconstruido. El conjunto de los bloques reconstruidos de un mismo cuadro actual forma el cuadro reconstruido, el que se almacena como cuadro de referencia.

El decodificador recibe el flujo de bits que representa al video codificado y realiza un proceso inverso al codificador, para extraer la información auxiliar y de movimiento después de la decodificación de entropía, el cuadro de diferencias transformado y cuantizado *x* después del reordenamiento inverso, el bloque decodificado *é* después de la transformación inversa y el bloque reconstruido después de la suma de *é* con el bloque compensado *c´*, obtenido con el vector de movimiento decodificado *MV* y el cuadro de referencia *r*. Los bloques decodificados de un mismo cuadro actual forman el cuadro decodificado, el cual esta listo para ser exhibido y almacenado para servir como referencia en la decodificación de los cuadros siguientes.

2.3.7 Estándares de codificación de video.

Todo sistema de codificación de video requiere el uso de un número de herramientas y algoritmos de compresión, las cuales son combinadas para proporcionar una solución óptima en coste. Si la codificación es utilizada en un sistema o equipo de video particular, sin necesidad de que sea conocida por otros fuera del fabricante, el diseñador tiene toda la libertad de seleccionar sus componentes, su conexión y su forma de operación. Pero si la información necesita ser procesada por diferentes fabricantes y usuarios, todos deben conocer, en diferentes grados, el proceso aplicado, por lo que se necesita uno o más sistemas de codificación estandarizados.

En el campo de las imágenes electrónicas, el trabajo de estandarización ha sido emprendido principalmente por la organización internacional para la estandarización (ISO), en cooperación con la comisión electro-técnica internacional (IEC) y el sector de estandarización de las telecomunicaciones, de la unión internacional de telecomunicaciones (ITU-T). Dentro de estas organizaciones, los sectores que han elaborado la mayoría de los estándares actuales son el grupo de expertos en imágenes en movimiento (MPEG), el grupo de la junta de expertos en fotografía (JPEG) y el grupo

de expertos en codificación de video (VCEG), los cuales han sido reconocidos internacionalmente por la gran calidad de sus contribuciones, que han revolucionado la industria del video.

A continuación se hace una breve descripción de los estándares internacionales desarrollados por los grupos JPEG, MPEG, VCEG y SG15. Se debe tomar en cuenta que los estándares JPEG se usan normalmente para la codificación de imágenes fijas y que se incluyen en esta sección porque también son usados para la codificación de imágenes en movimiento, si bien con baja eficiencia [Shi99].

* **JPEG.** Estándar para la compresión de imágenes fijas de tono continuo, con un algoritmo de codificación basado en la transformada DCT de bloques de 8×8 píxeles, seguido por la cuantización de los coeficientes DCT y la codificación de entropía estilo Huffman (JPEG línea-base). Iniciado a mediados de la década de los 1980s y completado en 1992, el JPEG se convirtió en el exitoso estándar ISO/IEC 10918-1 y la recomendación ITU-T T.81.JPEG "Digital compression and coding of continuous-tone still images".

* **JPEG-2000.** Sistema que especifica el proceso de decodificación para convertir datos de imágenes comprimidas en datos de imágenes reconstruidas, así como una guía del proceso de codificación para convertir datos de imágenes fuente en datos de imágenes comprimidas, utilizando una técnica basada en transformada Wavelet. Completado en el 2000, se convirtió en el estándar ISO/IEC 15444-1:2000 y la recomendación ITU-T Rec. T.800.

* **MPEG-1.** Estándar para la representación codificada de imágenes en movimiento y la información asociada de audio, para almacenamiento en medios digitales. Iniciado en 1988 y completado en 1991, el MPEG-1 se convirtió en el estándar ISO/IEC 11172-2, "Coding of moving picture and associated audio for digital storage media". Las principales características técnicas que cubre son: espacio de color YCbCr, esquema de muestreo 4:2:0, bloques 8×8, transformada DCT, señal de video progresiva, tasa y tamaño de cuadros flexible, cuadros I, P y B, estimación de movimiento con resolución *half*, matriz de cuantización, etc. Su aplicación principal es el almacenamiento de audio y video en medios digitales (CD-ROM), a una tasa de hasta 1.5 Mbps. El MPEG-1 no especifica el proceso de codificación. Este solo especifica la sintaxis y semántica del flujo de bits y el procesamiento de la señal en el decodificador (Figura 2.14) [Chen02].

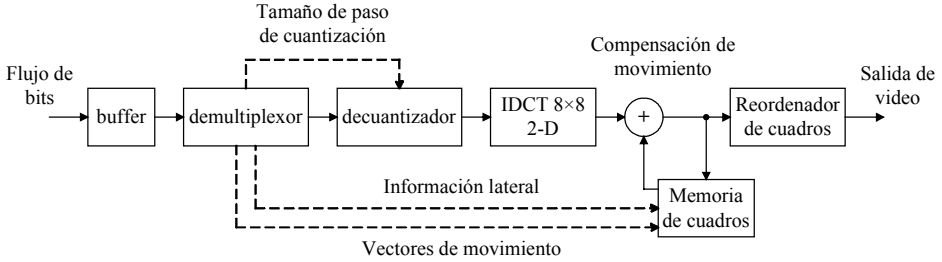


Figura 2.14. Diagrama a bloques simplificado del decodificador de video MPEG-1.

* **MPEG-2.** Extensión del primer estándar MPEG, que mantiene el objetivo de obtener la máxima eficiencia de codificación de video agregando más flexibilidad en el

formato del cuadro de entrada, mayor tasa de datos y mejor oposición al error. Iniciado en 1990 y completado en 1994, el MPEG-2 se convirtió en el estándar ISO/IEC 13818-2 y la recomendación ITU-T H.262, "Generic coding of moving pictures and associated audio information". Las principales características técnicas que agrega son: señal de video entrelazada y progresiva, esquemas de muestreo 4:2:0, 4:2:2 y 4:4:4, tasa de hasta 100 Mbps, factor de cuantización no-lineal, etc. Entre todos los estándares, el MPEG-2 es el que tiene el mayor impacto en la industria de la electrónica de consumo, ya que el DVD y la televisión digital lo han adoptado como su tecnología de compresión.

* **MPEG-4.** Grupo de estándares que tienen como objetivo estandarizar los algoritmos de codificación audiovisual en aplicaciones multimedia, por medio de funcionalidades como accesibilidad universal y robustez en ambientes propensos a errores, interactividad, codificación de datos naturales y sintéticos, alta eficiencia de compresión, etc. En contraste a sus homólogos MPEG-1 y MPEG-2 que se basan en cuadros, el MPEG-4 propone un procesamiento basado en objetos para la representación de la escena. Iniciado en 1993, fue completado en su versión inicial en 1999, como el estándar ISO/IEC 14496. La mayoría de sus características no tienen que estar disponibles en todas las aplicaciones, al punto que no es posible que existan realizaciones completas. Para manejar esta variedad, el estándar incluye los conceptos de *perfil* y *nivel*, lo que permite definir conjuntos específicos de capacidades que pueden ser realizados para cumplir objetivos particulares. Sus aplicaciones principales son los flujos de medios audiovisuales, la distribución en CD, la transmisión bidireccional y la emisión de televisión. Las partes de este grupo de estándares que se enfocan a la codificación de video son el MPEG-4, parte 2 (ISO/IEC 14496-2) y el MPEG-4 parte 10 (ISO/IEC 14496-10), llamados también MPEG-4 visual y MPEG-4 AVC (Advanced Video Coding), respectivamente.

* **H.120.** Primer estándar internacional de codificación de video digital, desarrollado por la CCITT (ahora la ITU-T) y aprobado en 1984. Definido originalmente como un codificador de reconstrucción condicional, utilizando técnicas DPCM, cuantización escalar y codificación de entropía VLC, a tasas de 1544 y 2048 Kbps [Sullivan98], su segunda versión (1988) agregó compensación de movimiento y predicción del fondo de los cuadros.

* **H.261.** Primer estándar internacional de compresión de video con un gran éxito práctico, desarrollado por el grupo de estudio de la ITU-T XV (SG15) para aplicaciones de comunicación bidireccional, por lo que enfatiza en bajas tasas de bits (de 64 a 1920 Kbps) y bajo retardo de codificación (150 ms máximo). Diseñado originalmente para transmisión sobre las líneas de la red digital de servicios integrados (ISDN), el H.261 utiliza varias técnicas, métodos y algoritmos de codificación que aun prevalecen: procesamiento de macro-bloques 16×16, dos modos de operación (intra e inter), un esquema de compensación de movimiento basado en DCT 8×8 con procesamiento del residuo, dos tamaños de cuadro (CIF y QCIF), esquema de muestreo 4:2:0 para cuadros I y P, codificación *run-length*, entropía VLC, etc. Iniciado en 1984 y completado en 1990 como estándar ITU-T, sus aplicaciones principales son la videotelefonía y las videoconferencias.

* **H.262.** Estándar de codificación de video digital, diseñado originalmente para la transmisión de video sobre redes ATM, a una tasa de 2 a 4 Mbps. Posteriormente se

alineo con el estándar MPEG-2, por lo que es sostenido por los grupos MPEG y VCEG.

* **H.263.** Estándar de compresión de video derivado del H.261, definido por el mismo grupo ITU-T SG15, que agrega estimación de movimiento con resolución *half* y tamaño de bloque variable, tablas VLC 3-D, cambio del paso de cuantización en cada bloque, codificación aritmética, predicción de MVs, etc. El H.263 soporta formatos de cuadro sub-QCIF, QCIF, CIF, 4CIF y 16CIF, con el espacio de color YCbCr muestreado en formato 4:2:0, a una tasa de 29.97 fps en modo progresivo. Iniciado en 1993 y completado en 1995, el estándar fué diseñado para comunicaciones de baja velocidad (menores a 64 Kbps) sobre la línea telefónica normal. Posteriormente se desarrollaron las versiones 2 (H.263+) y 3 (H.263++). La versión 2, completada en 1998, incluye características que mejoran la eficiencia de codificación (formato de video flexible, robustez contra pérdida de datos en el canal de transmisión, escalabilidad, etc.). La versión 3, liberada en el 2000, adiciona cuatro anexos que especifican modos avanzados de selección del cuadro de referencia, partición de datos y especificación de información, así como la definición de perfiles y niveles.

* **H.264.** Estándar de codificación de video digital, desarrollado conjuntamente por los grupos MPEG y VCEG, con un contenido idéntico al MPEG-4 parte 10.

Cabe destacar que los estándares MPEG y H.26x no especifican al codificador, solo al decodificador, por lo que no restringen las técnicas y tecnologías aplicadas en la codificación. De acuerdo a la definición, el codificador es un dispositivo realizado en hardware y/o software, que produce un flujo de bits semántica y sintacticamente adecuado, que cuando alimenta a un decodificador compatible, se obtiene la salida reconstruida correcta. Esto significa que el codificador es un sistema abierto, que puede utilizar algoritmos, tecnologías y arquitecturas diversas, con la restricción de que los resultados deben ser compatibles con el decodificador estandarizado que reconstruye la información visual.

En términos de los métodos de compresión de video, existe una tendencia hacia los algoritmos DCT y la predicción de la compensación de movimiento inter-cuadros. Por otro lado, las técnicas basadas en Wavelets han tenido un éxito reciente en la codificación de imágenes fijas, debido a sus características de alta eficiencia de codificación, excelente escalabilidad espacial y calidad. Pero en la codificación de video, la transformada Wavelet no ha tenido el mismo brillo debido a varias dificultades. Una de ellas es que no es claro como eliminar la redundancia temporal en este dominio, ya que la transformada Wavelet procesa bloques grandes o hasta el cuadro completo, pero la compensación de movimiento se realiza normalmente en bloques pequeños.

2.4 Estándar H.264/AVC.

2.4.1 Introducción.

El estándar de codificación MPEG-2 (1993) fue la tecnología que impulsó el auge de los sistemas de televisión digital alrededor del mundo, ya que permitió una transmisión eficiente de señales de televisión sobre satélite (digital video broadcast-

satellite, DVB-S), cable (DVB-C) y plataforma terrestre (DVB-T). Sin embargo, la eficiencia del MPEG-2 no fue suficiente para soportar otras tecnologías de comunicación de menor capacidad de transmisión, como cable-modem, xDLS o UMTS, por lo que se precisó una mayor compresión de video.

En 1998, el grupo VCEG de la ITU-T inició un proyecto llamado H.26L, con el objetivo de duplicar la eficiencia de codificación con respecto a los estándares existentes, es decir, reducir a la mitad la tasa de bits necesaria para el mismo nivel de fidelidad, sin aumentar proporcionalmente la complejidad de realización [Schäfer03].

El primer borrador del proyecto H.26L fue terminado en octubre de 1999 y en diciembre del 2001, el VCEG unió esfuerzos con el grupo MPEG de la ISO/IEC, formando el equipo JVT (Joint Video Team), con el propósito de finalizar el nuevo estándar de codificación de video, llamado H.264 o MPEG-4 parte 10 *Advanced Video Coding* (AVC), el cual fue aprobado en su primera versión en marzo del 2003. Posteriormente, en marzo del 2005, se oficializó la segunda versión, que incluyó las extensiones de rango de fidelidad (FREXt), agregando al estándar cuatro perfiles de alto comportamiento. Finalmente, en abril del 2007 se incluyeron cuatro perfiles, enfocados a la codificación intra. En cuanto a las actividades actuales del JVT, a partir de enero del 2005, se inició el proyecto del anexo "Scalable Video Coding" (SVC) y desde julio del 2006, se trabaja en una extensión del estándar hacia la codificación de video multi-vistas (MVC).

Al inicio del proceso de diseño del nuevo estándar, el grupo de expertos adoptó las siguientes premisas, para soportar y agilizar el desarrollo del proyecto [WikipediaH264]:

* La estructura "DCT + compensación de movimiento" de las versiones anteriores era superior a otros estándares, por lo que no había necesidad de hacer cambios fundamentales en ella.

* Algunos algoritmos, técnicas y métodos de codificación de video que habían sido excluidos en el pasado debido a su complejidad y su alto coste de realización, se re-examinarian para su posible inclusión, ya que la tecnología VLSI había sufrido un adelanto considerable, emparejada con una reducción de costes.

* Para permitir una libertad máxima en la codificación y evitar restricciones que comprometan la eficiencia, no se contempló mantener la compatibilidad con normas anteriores.

Como ha sido el caso de todos los estándares de codificación de video de la ITU-T y de la ISO/IEC, el estándar H.264/AVC solo define la sintaxis del flujo de bits del video codificado y el método de decodificación de este flujo de datos, de tal forma que todos los sistemas diseñados conforme al estándar, producen una salida correcta cuando reciben un flujo de bits codificado adecuadamente. Este alcance del estándar ofrece gran libertad de realización, principalmente en aplicaciones influenciadas por los costes de diseño, tiempo al mercado, nivel de calidad, etc. Sin embargo, esto no garantiza una reproducción de la calidad de extremo a extremo, ya que abre las puertas para la consideración de una amplia variedad de técnicas de codificación.

El estándar H.264/AVC comprende una capa de codificación de video (VCL), para la representación eficiente del contenido de video y una capa de abstracción de red (NAL), la que formatea la representación VCL y estipula la información de cabecera de una manera apropiada, para su transferencia a una capa de transporte particular o a un medio de almacenamiento.

El diseño de la VCL se fundamenta en el enfoque de codificación hibrida basada en bloques. El algoritmo básico de codificación es una combinación de inter-predicción, para explotar las dependencias estadísticas temporales, y transformación del residuo de predicción, para explotar las dependencias estadísticas espaciales. No existe un elemento de codificación único que proporcione la mayoría de las mejoras en la eficiencia de compresión, mas bien es un conjunto de pequeñas mejoras que sumadas hacen cumplir el objetivo de diseño.

La NAL se diseño con el fin de obtener un resultado de codificación amigable a diversos ambientes de redes, habilitando un mapeo simple y efectivo de los datos de la VCL en varios tipos de capas de transporte, para su uso en sistemas de almacenamiento y en servicios de Internet, conversacionales y de difusión, alambricos e inalambricos [Wiegand03a].

La primera versión del H.264/AVC cubre una gama amplia de aplicaciones de contenido de video incluyendo, pero no limitado, a las siguientes [H.264.05]:

CATV	Televisión por cable en redes ópticas, cobre, etc.
DBS	Servicios de video de difusión directa por satélite.
DSL	Servicios de video por línea de suscripción digital.
DTTB	Difusión de televisión digital terrestre.
ISM	Medios interactivos de almacenamiento (discos ópticos, etc.).
MMM	Correo multimedia.
MSPN	Servicios multimedia sobre redes de paquetes.
RTC	Servicios conversacionales en tiempo real (videoconferencias, videoteléfono, etc.).
RVS	Video vigilancia remota.
SSM	Medios series de almacenamiento (VTR digital, etc.).

Estas y otras aplicaciones han permitido que a pocos años de aprobarse, el estándar esté siendo adoptado por diversas asociaciones, foros, empresas, países, etc., como la asociación Blu-ray Disc, el foro HD-DVD, los servicios DBV-S en Inglaterra, Estados Unidos, Alemania y Japón, el proveedor de servicios de Internet TPG en Australia, etc.

2.4.2 Estructura básica.

Aun cuando el estándar solo especifica el flujo codificado de video y el proceso para su decodificación, en la práctica se definen los diagramas a bloques del codificador y el decodificador (Figuras 2.15 y 2.16), con sus estructuras y elementos funcionales básicos (predicción, transformación, cuantización, codificación de entropía, etc.).

El proceso de codificación principia con la presentación de un cuadro de video, el cual es dividido en unidades de macro-bloques (MBs) actuales, correspondientes a

16×16 píxeles de la imagen original. Los MBs de la primera imagen de una secuencia o de un punto de acceso aleatorio son típicamente codificados intra, utilizando la información contenida en la misma imagen. Los MBs de las imágenes siguientes o entre puntos de acceso aleatorio, son típicamente codificados inter, empleando predicción de la compensación de movimiento, a partir de los cuadros reconstruidos previamente (cuadros de referencia). En ambos casos, se genera un MB de predicción, que se resta del MB actual para producir un residuo o error de predicción, el cual es transformado (T) y cuantizado (Q), para obtener una matriz de coeficientes transformados cuantizados. La matriz es reordenada en una topología 1-D, para acceder a la codificación de entropía junto con la información lateral de control y movimiento. El codificador de entropía genera el flujo de datos codificado que pasa a la capa NAL para su transmisión o almacenamiento.

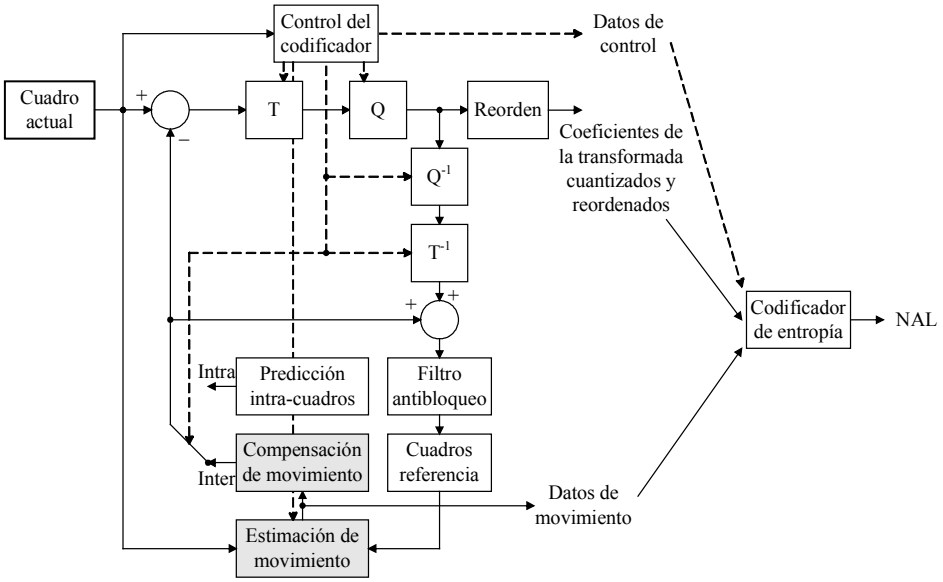


Figura 2.15. Estructura básica de codificación del H.264/AVC.

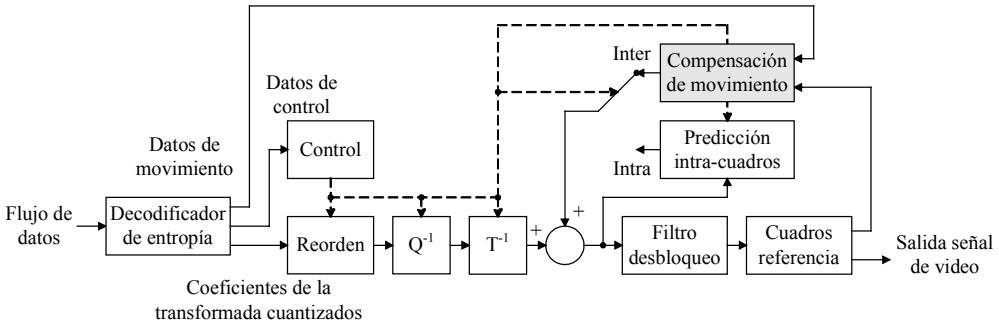


Figura 2.16. Estructura básica de decodificación del H.264/AVC.

En orden de reconstruir los cuadros para la predicción de los siguientes MBs, los coeficientes transformados cuantizados son re-escalados (Q^{-1}) y transformados inversamente (T^{-1}), para producir un MB de diferencias que se suma al MB de predicción, para crear un MB decodificado que se depura con un filtro antibloqueo antes de su respaldo, en la memoria de cuadros de referencia.

El proceso de decodificación inicia cuando el decodificador recibe un flujo de datos de la capa NAL, vía un medio de transmisión o un módulo de almacenamiento. Los datos de video, control y movimiento son decodificados de entropía y distribuidos entre los diferentes bloques del proceso. Los datos de video son reordenados para producir una matriz de coeficientes transformados y cuantizados, la cual pasa por un proceso inverso (Q^{-1} y T^{-1}), para obtener el MB de residuo. Utilizando la información de cabecera del flujo de datos, el decodificador crea un MB de predicción, idéntico al formado en el codificador y lo suma al MB de residuo para producir el MB decodificado, el que pasa por un filtro antibloqueo antes de integrase a los cuadros de referencia y formar la salida de la señal de video.

2.4.3 Características y técnicas generales.

A continuación se listan las técnicas mas importantes utilizadas en el H.264/AVC [Wiegand03a][WikipediaH264]. Algunas de ellas son innovaciones del estándar y otras ya han sido consideradas en recomendaciones previas, como la resolución de un cuarto de píxel en MPEG-4 parte 2 y los vectores de movimiento sobre los límites de la imagen en H.263 y MPEG-4 parte 2.

a) Codificación predictiva.

* **Compensación de movimiento con tamaño variable de bloque,** lo que proporciona una alta flexibilidad en la selección de tamaños de bloques, tan grandes como 16×16 píxeles y tan pequeños como 4×4, para lograr una segmentación precisa de las regiones en movimiento.

* **Resolución de un cuarto de píxel para compensación de movimiento,** en el componente luma y de un octavo en los componentes croma, lo que habilita una descripción precisa del desplazamiento de las áreas en movimiento, aplicando procesos de interpolación de baja complejidad.

* **Compensación de movimiento con múltiples imágenes de referencia,** utilizando hasta 32 cuadros de referencia para predicción y bi-predicción, lo que mejora la eficiencia de compresión, principalmente en escenas con rápido parpadeo repetitivo, corte de escenas de adelanto-atraso o con áreas no cubiertas del fondo.

* **Vectores de movimiento sobre los límites de la imagen,** aplicando una técnica de extrapolación en los límites de los cuadros de referencia.

* **Predicción pesada,** innovación en el H.264/AVC que permite que el codificador especifique el uso de escalado y desplazamiento en la compensación de movimiento, mejorando su rendimiento en casos de escenas de desvanecimiento.

* **Predicción intra,** con los procesos de predicción INTRA-4×4, INTRA-8×8 e INTRA-16×16 para las muestras luma y el proceso de predicción intra para las muestras croma aplicado por separado a cada bloque Cb y Cr.

* **Desacoplamiento del orden de los cuadros de referencia del orden de exhibición,** para que el codificador seleccione el orden de los cuadros para propósitos de referencia y exhibición, lo que proporciona un alto grado de

flexibilidad, restringido solamente por la capacidad de memoria que asegura la factibilidad de la decodificación.

* **Desacoplamiento de los métodos de representación de imágenes de la capacidad de referencia de imágenes,** lo que facilita que los cuadros codificados como bi-predictivos puedan usarse como referencias para la predicción de otros cuadros en la secuencia de video.

* **Mejora de la inferencia de movimiento "skipped" y "direct",** para deducir movimiento en áreas "skipped", cuando se codifica video conteniendo movimiento global y para optimizar la compensación de movimiento "direct" en áreas codificadas con bi-predicción.

b) Transformación.

* **Transformada entera del coseno (ICT) de bloques 4×4,** conceptualmente similar a la DCT, pero especificada de una manera simple y con una relación exacta entre la transformación directa e inversa. Su diseño requiere aritmética de 16 bits, con operaciones de sumas y desplazamientos binarios.

* **Transformada entera del coseno de bloques 8×8,** que permite que las regiones altamente correlacionadas sean comprimidas más eficientemente que con la transformada 4×4.

* **Transformada de Hadamard,** para el procesamiento de los coeficientes DC de la primera transformada espacial, aplicada a los coeficientes croma DC y a los luma en casos especiales. Con su uso se obtiene una mayor compresión en regiones suaves.

c) Cuantización.

* **Control logarítmico del tamaño de paso,** para facilitar la administración de la tasa de bits en el codificador y simplificar el re-escalado de cuantización inverso.

* **Particularización en frecuencia de las matrices de escalado,** seleccionadas por el codificador para la optimización de la cuantización basada en percepción.

d) Codificación de entropía.

* **Codificación aritmética binaria adaptable al contexto (CABAC),** para la compresión sin pérdidas de los elementos de sintaxis en el flujo de video, conociendo sus probabilidades en un contexto específico. Método más eficiente pero con mayor coste computacional que el CAVLC en la decodificación.

* **Codificación de longitud variable adaptable al contexto (CAVLC),** el cual es una alternativa de baja complejidad con respecto al CABAC, utilizado para la codificación de los coeficientes transformados cuantizados.

* **Codificación de longitud variable (VLC),** para la codificación de los elementos de sintaxis nos codificados por CAVLC.

e) Codificación de macro-bloques sin pérdidas.

* **Modo de representación de macro-bloques PCM sin pérdidas,** en el cual los datos de las muestras de video son representados directamente, admitiendo una representación perfecta de regiones específicas, con un límite estricto de la cantidad de datos codificados por cada MB.

* **Modo avanzado de representación de macro-bloques sin pérdidas,** permitiendo una representación perfecta de regiones específicas, con menos bits que el modo PCM.

f) Codificación de video entrelazado.

* **Codificación cuadro-campo adaptable al MB (MBAFF),** utilizando un par estructurado de MBs, para imágenes codificadas como cuadros y permitiendo MBs 16×16 en video entrelazado.

* **Codificación cuadro-campo adaptable a la imagen (PAFF),** permitiendo una mezcla de selección libre de imágenes codificadas como campos únicos individuales (medios cuadros) de video entrelazado.

g) Robustez a errores y pérdidas de datos y flexibilidad para la operación sobre una variedad de redes.

* **Estructura del conjunto de parámetros,** que trata de la separación de la información de cabecera del flujo codificado, para su manejo de una manera flexible y más especializada, ya que su pérdida o degradación produce un gran impacto negativo en el proceso de decodificación

* **Estructura de la sintaxis en unidades de NAL,** las cuales son paquetes de datos que permiten una mayor particularización del método de transporte del contenido de video, de una manera apropiada para cada red.

* **Ordenación flexible de MBs (FMO),** que es la habilidad de particionar la imagen en regiones llamadas grupo de sectores, las cuales contienen uno o más sectores, donde cada sector es decodificable independientemente.

* **Sectores I, P y B,** los cuales definen una región de la imagen compuesta por un número entero de MBs o pares de MBs ordenados consecutivamente en exploración de rastreo, dentro de un grupo de sectores. Un sector I contiene solo MBs codificados intra. El sector P incluye MBs codificados intra, inter o *skipped*. El sector B incluye MBs codificados intra, o inter, este último utilizando dos cuadros de referencia simultáneamente.

* **Tamaño flexible de sectores,** para aumentar la eficiencia de codificación, al reducir la cantidad de datos de cabecera e incrementar la eficacia de predicción.

* **Ordenación arbitraria de sectores (ASO),** para la habilitación del envío y recepción de los sectores de la imagen en cualquier orden, lo cual reduce el retardo de extremo a extremo en aplicaciones de tiempo real, particularmente cuando se

usa sobre redes que tienen un comportamiento de entrega fuera de orden, como las redes con protocolos de Internet.

* **Sectores redundantes (RS),** una característica de robustez al error o pérdida que permite al codificador enviar una representación extra de una región de la imagen (típicamente a baja fidelidad), que puede ser usada si la representación primaria se corrompe o pierde.

* **Partición de datos (DP)**, una característica que proporciona la habilidad de separar los elementos de sintaxis en diferentes paquetes de datos, de acuerdo a su grado de importancia, permitiendo la aplicación de varios niveles de protección al error.

* **Sectores de conmutación SP/SI,** como nuevos tipos de elementos que sirven para codificar la transición entre dos flujos de video, en los cambios de la tasa de bits, en la recuperación cuando se tienen errores o pérdida de datos, en la habilitación de modos de operación de adelanto y atraso rápidos, etc.

* **Numeración de cuadros,** característica para la creación de sub-secuencias, habilitando el escalado temporal con la inclusión opcional de cuadros extras y la detección y encubrimiento de la pérdida de cuadros completos, lo cual puede ocurrir debido a la pérdida de paquetes de red o errores en el canal.

h) Otras características.

* **Filtro antibloqueo en el lazo de predicción,** para eliminar los artefactos producidos por la codificación de video basada en bloques y para mejorar la calidad subjetiva y objetiva del video resultante.

* **Soporte de formatos de muestreo 4:0:0 (monocromático), 4:2:0, 4:2:2 y 4:4:4** (en función del perfil seleccionado).

* **Soporte de unidades básicas de video (píxeles) de 8 a 14 bits** (dependiendo del perfil seleccionado).

2.4.4 Perfiles y niveles.

El estándar fue diseñado en forma genérica, en el sentido de que este soporta una amplia gama de aplicaciones, tasa de bits, formatos de cuadro, resoluciones, herramientas, calidades y servicios. En el proceso de la creación de la norma, se consideraron varios requerimientos de aplicaciones típicas y se desarrollaron los elementos algorítmicos necesarios, lo que se integro en una sola sintaxis, por lo que la recomendación facilita el intercambio de datos de video entre diferentes aplicaciones. Considerando la complejidad de realización de toda la sintaxis, se estipuló un número de subconjuntos por medio de perfiles y niveles. Un perfil es un subconjunto de la sintaxis completa del estándar, mientras que un nivel es un grupo de restricciones impuestas en los valores de los elementos de la sintaxis de cada perfil.

2.4.4.1 Perfiles.

Actualmente, el estándar incluye los siguientes perfiles generales, enfocados a diferentes aplicaciones:

* **Línea base (BP).** Perfil para aplicaciones de bajo coste y recursos de cómputo limitados; utilizado en videoconferencias y aplicaciones móviles.

* **Principal (MP).** Definido originalmente como el perfil para aplicaciones de difusión y almacenamiento, su importancia se debilitó cuando se desarrollaron los perfiles altos para las mismas aplicaciones.

* **Extendido (XP).** Especificado como el perfil del video *streaming*, con capacidades de alta compresión, robustez a la pérdida de datos y funciones auxiliares para la conmutación de flujo en servidores.

* **Alto (HiP).** Perfil principal para aplicaciones de difusión y almacenamiento en disco, particularmente para aplicaciones de televisión de alta definición (por ejemplo, este es el perfil adoptado en HD DVD y Blu-ray).

* **Alto 10 (Hi10P).** Nivel superior del perfil HiP, que soporta 10 bits por muestra en la precisión de las imágenes decodificadas.

* **Alto 4:2:2 (Hi422P).** Perfil construido sobre el perfil Hi10P, en el que se agrega soporte para formatos de muestreo hasta 4:2:2. Orientado hacia aplicaciones profesionales que utilizan video entrelazado.

* **Alto 4:4:4 predictivo (Hi444PP).** Perfil construido sobre el perfil Hi422P, para soportar formatos de muestreo hasta 4:4:4 y 14 bits por muestra. Incluye soporte adicional para la codificación sin pérdidas y la codificación de cada cuadro como tres planos de color separados.

Para precisar el alcance de cada perfil, la tabla 2.2 muestra las funciones y herramientas de codificación que soporta cada uno de ellos. En adición a los perfiles generales, el estándar incluye cuatro perfiles totalmente intra, especificados para aplicaciones profesionales (perfiles High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra y CAVLC 4:4:4 Intra).

2.4.4.2 Niveles.

Cada nivel especifica un conjunto de límites en los valores que pueden tomar los elementos de sintaxis del estándar. Todos los perfiles utilizan el mismo conjunto de definición de niveles (Tabla 2.3), pero las realizaciones individuales pueden soportar un nivel diferente para cada perfil soportado. Para un perfil específico, los niveles corresponden generalmente a la carga de procesamiento del decodificador y a la capacidad de memoria.

Tabla 2.2. Funciones y herramientas de los perfiles del estándar H.264/AVC.

Función o herramienta	BP	XP	MP	HiP	Hi10P	Hi422P	Hi444P
Sectores I y P	Si	Si	Si	Si	Si	Si	Si
Sector B	No	Si	Si	Si	Si	Si	Si
Sectores SI y SP	No	Si	No	No	No	No	No
Múltiples cuadros de referencia	Si	Si	Si	Si	Si	Si	Si
Filtro antibloqueo	Si	Si	Si	Si	Si	Si	Si
Codificación CAVLC	Si	Si	Si	Si	Si	Si	Si
Codificación CABAC	No	No	Si	Si	Si	Si	Si
FMO	Si	Si	No	No	No	No	No
ASO	Si	Si	No	No	No	No	No
Sectores redundantes (RS)	Si	Si	No	No	No	No	No
Partición de datos	No	Si	No	No	No	No	No
Codificación entrelazada (MBAFF y PAFF)	No	Si	Si	Si	Si	Si	Si
Formato 4:0:0	No	No	No	Si	Si	Si	Si
Formato 4:2:0	Si	Si	Si	Si	Si	Si	Si
Formato 4:2:2	No	No	No	No	No	Si	Si
Formato 4:4:4	No	No	No	No	No	No	Si
Pixeles de 8 bits	Si	Si	Si	Si	Si	Si	Si
Pixeles de 9 y 10 bits	No	No	No	No	Si	Si	Si
Pixeles de 11 a 14 bits	No	No	No	No	No	No	Si
Transformada adaptable 4×4 ó 8×8	No	No	No	Si	Si	Si	Si
Matrices de escalado de cuantización	No	No	No	Si	Si	Si	Si
Paso de cuantización separado para Cb y Cr	No	No	No	Si	Si	Si	Si
Codificación separada del plano de color	No	No	No	No	No	No	Si
Codificación predictiva sin pérdidas	No	No	No	No	No	No	Si

2.5 Técnicas de estimación de movimiento en el estándar H.264/AVC.

En las siguientes secciones se documentan las técnicas de estimación de movimiento que más contribuyen en el incremento de la eficiencia de codificación del estándar H.264/AVC. Estas técnicas, no consideradas en las normas previas, mejoran significativamente la habilidad de predecir la acción de movimiento del contenido de una secuencia de imágenes. Si bien su realización utiliza recursos adicionales de cómputo, esto ha sido compensado por el fenómeno conocido como la ley de Moore, por lo que la complejidad computacional extra no es una restricción para el uso de esta norma en un gran número de aplicaciones.

Tabla 2.3. Límites de los niveles del estándar H.264/AVC.

Número de nivel	Tasa máxima de procesa-miento de MBs (MB/s)	Tamaño máximo de cuadro (MBs)	Tamaño máximo buffer imagen deco-dificada (1024 bytes para 4:2:0)	Tasa máxima en bits del video (1000 ó 1200 bps)	Tamaño máximo buffer de la imagen codificada (1000 ó 1200 bits)	Máximo rango componente vertical MV (muestras de cuadro luma)	Razón de compresión mínima	Máximo número de MVs por 2 MBs consecu-tivos
1	1 485	99	148.5	64	175	[–64,+63.75]	2	-
1b	1 485	99	148.5	128	350	[–64,+63.75]	2	-
1.1	3 000	396	337.5	192	500	[–128,+127.75]	2	-
1.2	6000	396	891.0	384	1 000	[–128,+127.75]	2	-
1.3	11 880	396	891.0	768	2 000	[–128,+127.75]	2	-
2	11 880	396	891.0	2 000	2 000	[–128,+127.75]	2	-
2.1	19 800	792	1 782.0	4 000	4 000	[–256,+255.75]	2	-
2.2	20 250	1 620	3 037.5	4 000	4 000	[–256,+255.75]	2	-
3	40 500	1 620	3 037.5	10 000	10 000	[–256,+255.75]	2	32
3.1	108 000	3 600	6 750.0	14 000	14 000	[–512,+511.75]	4	16
3.2	216 000	5 120	7 680.0	20 000	20 000	[–512,+511.75]	4	16
4	245 760	8 192	12 288.0	20 000	25 000	[–512,+511.75]	4	16
4.1	245 760	8 192	12 288.0	50 000	62 500	[–512,+511.75]	2	16
4.2	491 520	8 192	12 288.0	50 000	62 500	[–512,+511.75]	2	16
5	589 824	22 080	41 310.0	135 000	135 000	[–512,+511.75]	2	16
5.1	983 040	36 864	69 120.0	240 000	240 000	[–512,+511.75]	2	16

2.5.1 Tamaño de bloque variable.

La norma H.264 soporta compensación de movimiento con tamaño de bloque variable. El componente luma de cada MB puede ser dividido en cuatro modos para compensar el movimiento: como una partición 16×16, dos particiones 8×16, dos particiones 16×8 ó cuatro particiones 8×8. Si se selecciona el modo 8×8, cada partición 8×8 puede ser dividida en otras cuatro particiones de sub-macrobloque: una 8×8, dos 4×8, dos 8×4 ó cuatro 4×4. El método de dividir el MB en bloques y sub-bloques de tamaño variable es conocido como compensación de movimiento estructurada en árbol (Figura 2.17) [H.264.05].

Cada bloque o sub-bloque derivado de la partición del MB requiere su propio MV. La selección del tipo de partición, así como los MVs resultantes son codificados e incluidos en el flujo de bits de salida, por lo que esta operación tiene gran impacto en el rendimiento del sistema de compresión. La selección de una partición de gran tamaño, necesita un pequeño número de bits para indicar el MV y el tipo de partición, pero el residuo de la compensación de movimiento puede contener una cantidad de energía significativa en áreas del cuadro con alto detalle. La elección de particiones de tamaño pequeño puede ofrecer un residuo de baja energía después de la compensación de movimiento, pero utiliza un gran número de bits para señalar los MVs y la partición. En

general, las particiones de gran tamaño son apropiadas para áreas homogéneas y las de tamaño pequeño pueden utilizarse en áreas de detalle de los cuadros de video.

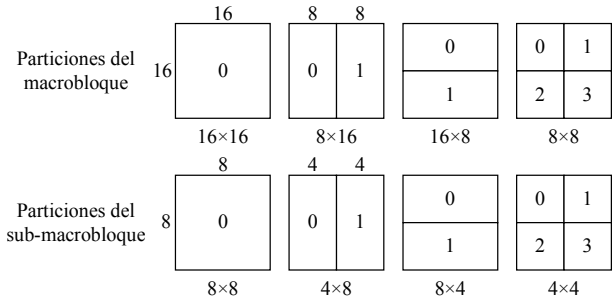


Figura 2.17. Particiones macro-bloque (16×16, 8×16, 16×8 y 8×8) y sub-macrobloque (8×8, 4×8, 8×4 y 4×4) para la compensación de movimiento estructurada en árbol.

Ya que cada componente croma de un MB (Cb y Cr) tiene la mitad de la resolución horizontal y vertical del componente luma, su partición tiene la misma forma que el bloque luma, pero con la mitad de la resolución horizontal y vertical (por ejemplo, una partición luma 16×8 se relaciona con una partición croma 8×4). De igual forma, el rango de los componentes x y y de los MVs de las particiones croma, corresponden a la mitad de los respectivos luma.

2.5.2 Resolución de un cuarto de muestra.

Una de las principales mejoras del estándar H.264/AVC con respecto a las normas previas, es el aumento en la resolución de la representación del movimiento, al utilizar muestras en posiciones enteras y fraccionarias (media y un cuarto), con lo que se tiene una alta exactitud en la compensación de movimiento [Wiegan03].

La exactitud de la compensación de movimiento es de un cuarto de la distancia entre las muestras luma y de un octavo entre las muestras croma. En caso de que el MV apunte a una posición de muestra entera, la señal de predicción se especifica directamente con las muestras del cuadro de referencia. De otra manera, las muestras son obtenidas utilizando interpolación para generar posiciones no enteras. Los valores de predicción en posiciones de un medio de muestra son obtenidos aplicando un filtro FIR unidimensional de 6-taps, horizontal y verticalmente. Los valores de predicción en posiciones de un cuarto de muestra son generados por el promedio de muestras en posiciones enteras y de un medio. Los valores de predicción para los componentes croma son siempre obtenidos por interpolación lineal [Schäfer03].

2.5.2.1 Interpolación de muestras luma.

Dadas las muestras ‘A’ a ‘U’ en las posiciones de muestras enteras (Figura 2.18), las muestras luma ‘a’ a ‘s’ en las posiciones de muestras fraccionarias se calculan como se describe a continuación [H.264.05]. Los valores de predicción luma en posiciones de media muestra se calculan aplicando un filtro de 6 coeficientes cuyos valores son (1, –5, 20, 20, –5, 1). Los valores de predicción luma en las posiciones de un cuarto de muestras se calculan con el valor medio de las muestras en las posiciones de muestra entera y de media muestra.

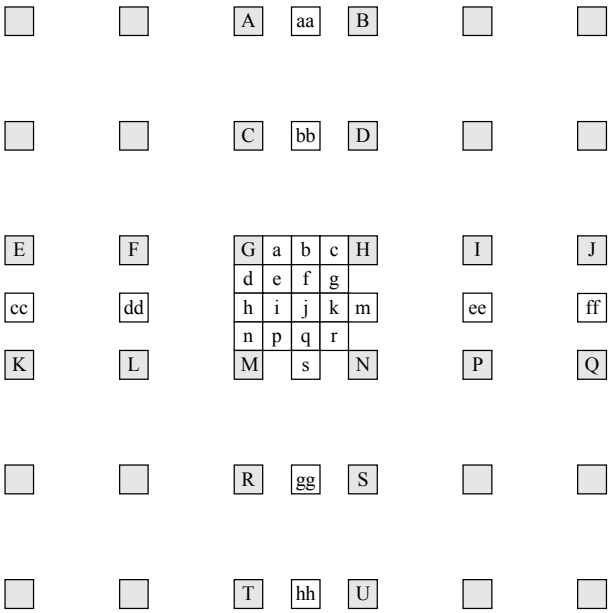


Figura 2.18. Posiciones de muestras enteras y fraccionarias.

2.5.2.1.1 Posiciones de media muestra.

Para calcular las muestras en la posicion de media muestra indicada con b, se calculan en primer lugar los valores intermedios indicados mediante b_1, para lo cual se aplica un filtro de 6 coeficientes a las muestras de posición entera más cercanas, en sentido horizontal (2.14). Para calcular las muestras en la posicion de media muestra indicada mediante h, se calculan en primer lugar los valores intermedios indicados como h_1, para lo cual se aplica un filtro de 6 coeficientes a las muestras de posición entera más cercanas, en sentido vertical (2.15).

$$b_1 = (E - 5 * F + 20 * G + 20 * H - 5 * I + J) \quad (2.14)$$

$$h_1 = (A - 5 * C + 20 * G + 20 * M - 5 * R + T) \quad (2.15)$$

Los valores finales de predicción b y h se calculan mediante las siguientes expresiones:

$$b = Clip1_Y((b_1 + 16) >> 5) \quad (2.16)$$

$$h = Clip1_Y((h_1 + 16) >> 5) \quad (2.17)$$

siendo $Clip1_Y$ una función matemática que recorta su argumento entre los valores 0 y $2^{BitDepthY} - 1$, donde BitDepthY el número de bits de las muestras luma.

Para calcular las muestras en la posición de media muestra indicada mediante j, se calcula en primer lugar el valor intermedio indicado como j_1, para lo cual se aplica un filtro de 6 coeficientes a los valores intermedios de las posiciones de media muestra más cercanas en el sentido horizontal (2.18) o vertical (2.19), dado que el resultado es el mismo:

$$j_1 = cc - 5 * dd + 20 * h_1 + 20 * m_1 - 5 * ee + ff \quad (2.18)$$

$$j_1 = aa - 5 * bb + 20 * b_1 + 20 * s_1 - 5 * gg + hh \quad (2.19)$$

donde los valores intermedios indicados como aa, bb, gg, s_1 y hh se calculan aplicando un filtro de 6 coeficientes horizontalmente, del mismo modo que en el cálculo de b_1, y los valores intermedios indicados como cc, dd, ee, m_1 y ff se calculan aplicando un filtro de 6 coeficientes en sentido vertical, del mismo modo que en el cálculo de h_1. El valor de predicción final j se calcula mediante la siguiente expresión:

$$j = Clip1_Y((j_1 + 512) >> 10) \quad (2.20)$$

Los valores de predicciones finales de s y m se calculan a partir de s_1 y m_1 de igual manera que en el cálculo de b y h, es decir:

$$s = Clip1_Y((s_1 + 16) >> 5) \quad (2.21)$$

$$m = Clip1_Y((m_1 + 16) >> 5) \quad (2.22)$$

2.5.2.1.2 Posiciones de un cuarto de muestra.

Las muestras en las posiciones de un cuarto de muestra indicadas mediante a, c, d, n, f, i, k y q, se calculan con el promedio redondeando hacia arriba, de las dos muestras enteras y/o medias más cercanas mediante las siguientes expresiones:

$$a = (G + b + 1) >> 1 \quad (2.23)$$

$$c = (H + b + 1) >> 1 \quad (2.24)$$

$$d = (G + h + 1) >> 1 \quad (2.25)$$

$$n = (M + h + 1) >> 1 \quad (2.26)$$

$$f = (b + j + 1) >> 1 \quad (2.27)$$

$$i = (h + j + 1) >> 1 \quad (2.28)$$

$$k = (j + m + 1) >> 1 \quad (2.29)$$

$$q = (j + s + 1) >> 1 \quad (2.30)$$

Las muestras en las posiciones de un cuarto de muestra indicadas mediante e, g, p, y r, se calculan con el promedio redondeado haciendo hacia arriba, de las dos medias muestras más cercanas en dirección diagonal, mediante las siguientes expresiones:

$$e = (b + h + 1) >> 1 \quad (2.31)$$

$$g = (b + m + 1) >> 1 \quad (2.32)$$

$$p = (h + s + 1) >> 1 \quad (2.33)$$

$$r = (m + s + 1) >> 1 \quad (2.34)$$

2.5.2.2 Interpolación de muestras croma.

Las sub-muestras croma son generadas en intervalos de un octavo entre muestras enteras en cada componente Cb y Cr, utilizando interpolación lineal. Cada sub-muestra se calcula como una combinación lineal de sus muestras enteras vecinas A, B, C y D, en función de la posición ($xFrac_C$, $yFrac_C$) que se tiene con respecto a la muestra entera superior izquierda (Figura 2.19) (2.35).

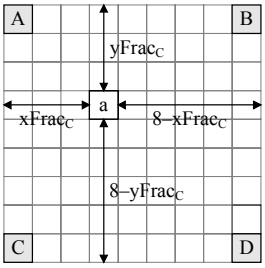


Figura 2.19. Interpolación croma en la posición de un octavo de muestra.

$$pred[x_C, y_C] = ((8 - xFrac_C) * (8 - yFrac_C) * A + xFrac_C * (8 - yFrac_C) * B + (8 - xFrac_C) * yFrac_C * C + xFrac_C * yFrac_C * D + 32) >> 6 \quad (2.35)$$

Si como ejemplo se considera la muestra 'a' en la posición de un octavo $xFrac_C = 3$ y $yFrac_C = 3$, de la figura 2.19, se tiene:

$$pred[x_C, y_C] = (25 * A + 15 * B + 15 * C + 9 * D + 32) >> 6 \quad (2.36)$$

2.5.3 Múltiples imágenes de referencia.

El estándar H.264/AVC soporta predicción de la compensación de movimiento multi-imagen. Esto es, puede utilizar mas de una imagen previamente codificada como referencia para la predicción de la compensación de movimiento [Schäfer03]. Esto habilita al codificador a buscar el mejor ajuste para la partición del MB actual en un amplio conjunto de imágenes. (Figura 2.20).

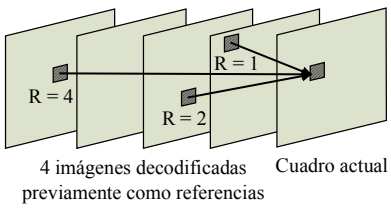


Figura 2.20. Ejemplo del uso de referencias múltiples para la predicción de la compensación de movimiento de una partición del MB actual.

Las imágenes codificadas en forma predictiva (imágenes P) en MPEG-2 y sus predecesores, usan solo una imagen previa para predecir el valor del cuadro actual. El H.264/AVC extiende la selección de cuadros de referencia utilizado en H.263++ para

habilitar una codificación eficiente, al permitir que un codificador seleccione, para propósitos de compensación de movimiento, entre un gran número de imágenes que han sido decodificadas y almacenadas. La misma extensión de capacidad de referencia es también aplicada en la compensación de movimiento bi-predictiva, la cual es restringida en MPEG-2 a utilizar dos imágenes específicas solamente, una de ellas siendo la imagen previa intra (I) o P en el orden de exhibición y otra siendo la imagen siguiente I o P en el orden de exhibición [Wiegan03].

El codificador y el decodificador mantienen una o dos listas de imágenes de referencia en un *buffer* multi-imágenes, conteniendo imágenes que han sido codificadas y decodificadas previamente, en un orden de exhibición antes o después de la imagen actual. Los MBs codificados inter y las particiones de MB en los sectores P se predicen de imágenes en una lista única, lista 0. Los MB codificados inter y las particiones de MB en un sector B se predicen de dos listas, lista 0 y lista 1 [Richardson03]. A menos que el tamaño del *buffer* multi-imagen sea establecido en una imagen, el índice en el cual la imagen de referencia es localizada dentro del *buffer* tiene que ser señalado y transmitido para cada bloque luma compensado en movimiento de tamaño 16×16 a 8×8. La compensación de movimiento para regiones más pequeñas que 8×8, usan el mismo índice de referencia que todos los bloques dentro de la región 8×8.

2.5.4 Predicción inter en sectores B.

En comparación con los estándares previos, el estándar H.264/AVC generaliza el concepto de sectores B, al permitir que las imágenes codificadas con predicción inter bi-predictiva también se puedan utilizar como referencias para la codificación inter de otras imágenes [Wiegand03b].

Cada partición o sub-partición de un MB, en un MB codificado inter de un sector B, puede predecirse a partir de uno o dos cuadros de referencia (I, P o B), antes o después de la imagen actual, en el orden de exhibición. En función de las imágenes almacenadas en el *buffer* del codificador y del decodificador, se tienen varias opciones para seleccionar las referencias de predicción de las particiones del MB. Por ejemplo, se pueden seleccionar una referencia pasada y una futura, dos referencias pasadas y dos referencias futuras, utilizando dos listas de imágenes de referencia previamente codificadas, lista 0 y lista 1 (Figura 2.21).

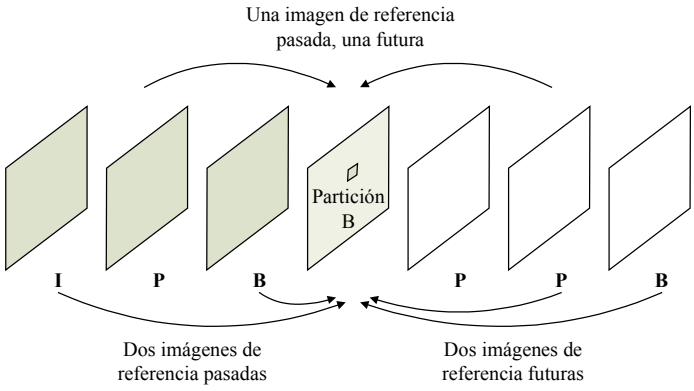


Figura 2.21. Ejemplos de bi-predicción, utilizando imágenes I, P y B.

Los sectores B soportan cuatro tipos de predicción inter: lista 0, lista 1, bi-predictivo y predicción directa. Los diferentes modos de predicción pueden ser escogidos para cada partición; si se utiliza el tamaño de partición 8×8, el modo elegido es aplicado a todas las sub-particiones dentro de la partición.

En modo bi-predictivo, se crea un bloque de referencia del mismo tamaño que la partición o sub-partición actual, a partir de una imagen de referencia de la lista 0 y otra de la lista 1, por lo que se requieren dos MVs. Cada muestra del bloque de predicción se calcula como un promedio de las muestras de predicción de las listas 0 y 1, excepto cuando se utiliza predicción pesada. Después de calcular las muestras de predicción, el residuo de la compensación de movimiento es formado como en los sectores P.

En el modo directo, no se transmite un vector de movimiento para el MB o partición del MB del sector B. En su lugar, el decodificador calcula los vectores de las listas 0 y 1 a partir de los vectores previamente decodificados.

La predicción pesada es un método para escalar las muestras de predicción de la compensación de movimiento en MBs de sectores P y B. Existen tres tipos de predicción pesada en el H.264/AVC:

* Predicción pesada explicita en MB de sector P.

* Predicción pesada explicita en MB de sector B.

* Predicción pesada implicita en MB de sector B.

Cada muestra de predicción de la lista 0 ó de la lista 1 puede ser escalada por un factor de peso w_0 ó w_1 previo a la predicción de la compensación de movimiento. En los tipos explícitos, los factores de peso son determinados por el codificador y transmitidos en la cabecera del sector. En la predicción implícita, los factores de peso son calculados a partir de la posición temporal relativa de los cuadros de referencia en las listas 0 y 1. Si la imagen de referencia esta temporalmente cercana al cuadro actual, se aplica un factor de peso grande y si esta temporalmente alejada, el factor de peso es pequeño. Una aplicación de la predicción pesada es el permitir un control de la contribución relativa de cada imagen de referencia, por ejemplo en la codificación de transiciones de desvanecimiento, donde una escena se desvanece en otra.

2.6 Algoritmo *block-matching* y su aplicación en el estándar H.264/AVC.

La estimación de movimiento juega un papel preponderante en los sistemas de codificación de video, ya que es el proceso de mayor intensidad computacional del sistema (de 60% a 80%), además de tener un alto impacto en la calidad del video. Ya que la evaluación del movimiento no es estandarizada, la libre competencia ha generado un gran conjunto de métodos de estimación, que pueden ser clasificados en algoritmos en el dominio del tiempo y algoritmos en el dominio de la frecuencia (Figura 2.22) [Kuhn99].

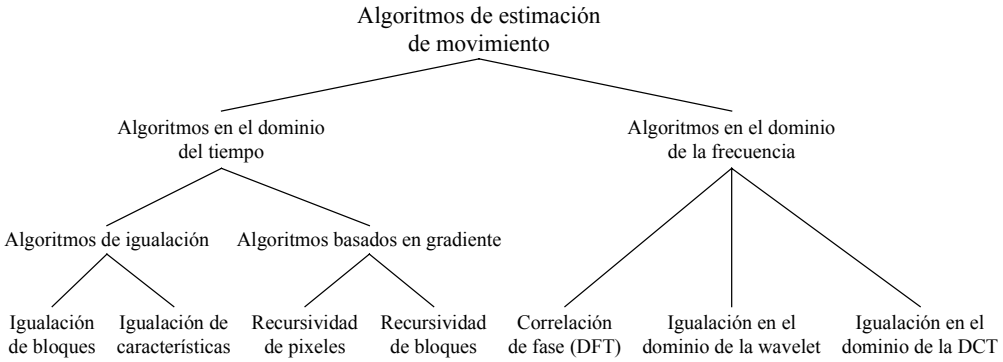


Figura 2.22. Clasificación de los algoritmos de estimación de movimiento.

Los algoritmos en el dominio del tiempo utilizan técnicas de ajuste y recursividad, mientras que los algoritmos en el dominio de la frecuencia aplican técnicas de transformación y correlación de fase. Ya que el algoritmo de ajuste de bloques "block matching" (BM) ha sido adoptado por todos los estándares de codificación de video ISO e ITU-T, incluyendo el H.264/AVC, gracias a su operación simple, directa y muy eficiente, las siguientes secciones se enfocan a describir esta popular técnica de estimación de movimiento.

La alta capacidad de compresión del H.264/AVC se obtiene utilizando algoritmos de intra e inter codificación. La codificación intra explota las dependencias estadísticas espaciales de la señal de video en una sola imagen. La codificación inter explota las dependencias estadísticas temporales entre diferentes imágenes [H.264.05]. La codificación inter utiliza una técnica de estimación de movimiento basada en bloques, la cual compara el bloque actual con los bloques candidatos dentro de una ventana de búsqueda "search window" (SW) en un cuadro de referencia. El desplazamiento entre el bloque actual y el bloque más similar en la SW es llamado el vector de movimiento (MV). El algoritmo BM se ejecuta primero con una resolución de muestras enteras y después con muestras fraccionarias, para todos los tamaños de macro-bloque y sub-macrobloque, en cada uno de los cuadros de referencia especificados, en los sectores P y B de la secuencia de imágenes.

2.6.1 Principio de operación.

La técnica BM fue propuesta por Jain y Jain [Jain81] a partir de un modelo simple de movimiento: una imagen actual es fraccionada en un conjunto de bloques rectangulares pequeños no-traslapados, igualmente espaciados y de tamaño fijo (bloques actuales), considerandose un movimiento de traslación uniforme e independiente en cada bloque, entre la imagen actual y la imagen previa. Aun cuando el modelo solo considera el movimiento de traslación, otros tipos de movimiento, tales como la rotación y el *zoom* de objetos grandes, pueden ser aproximados por la traslación de los bloques, pensando que estos son suficientemente pequeños. Esta observación, realizada originalmente por Jain y Jain, ha sido confirmada una y otra vez desde entonces [Shi99].

Si se considera que en el transmisor se define un cuadro de referencia, a partir de la codificación y decodificación de la imagen previa, se identifica en este cuadro de referencia una "área de búsqueda", alrededor de un bloque con el mismo tamaño y

coordenadas que el bloque actual. La hipótesis del algoritmo es que dentro del área de búsqueda existe un bloque que iguala al bloque actual en tamaño y contenido. Entonces, en lugar de transmitir todos los píxeles del bloque actual, solo se necesita especificar el desplazamiento entre la ubicación del bloque actual y el bloque igualado en el cuadro de referencia, por lo que en el receptor, que previamente recibió y decodifico el cuadro de referencia, se puede cortar y pegar esta área del cuadro de referencia a la posición del bloque actual, para reconstruir la imagen actual sin recibir otros bits mas que el valor de desplazamiento, pensando que el ajuste de bloques es perfecto [Guan01].

En la realización del algoritmo BM, el cuadro actual, una imagen con un tamaño horizontal y vertical de Nh×Nv píxeles, es dividido en (Nh/M)×(Nv/N) bloques de M×N píxeles. Cada bloque (llamado bloque actual *c*) es exhaustivamente comparado con todos los bloques candidatos *r* dentro de una SW típicamente rectangular, de tamaño (M+2Ph)×(N+2Pv) en un cuadro de referencia, donde 2Ph y 2Pv son los desplazamientos horizontal y vertical máximos permitidos (Figura 2.23). El MV de menor coste (dx,dy), puede ser obtenido cuando se encuentra un bloque candidato que mejor iguala al bloque actual, bajo un criterio que minimiza una medición de error.

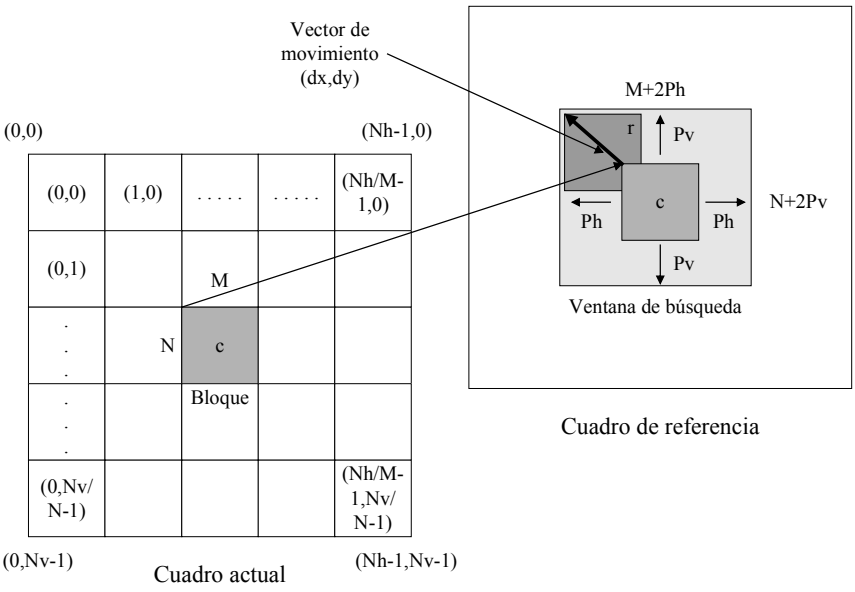


Figura 2.23. Realización del algoritmo de ajuste de bloques.

Para un bloque M×N, el algoritmo BM puede ser descrito por un lazo anidado de seis niveles, utilizando lenguaje tipo C (Figura 2.24), donde c(i,j) y r(i+m, j+n) son píxeles del bloque actual y del bloque candidato respectivamente.

El método de optimización de la estimación de movimiento en el software de referencia JM [JM1106] es el criterio de coste tasa/distorsión (R/D) "J = D + λ*R", donde D es la medida de distorsión, R es el número de bits asociados con la transmisión del vector de movimiento (evaluada usando la tabla universal de codificación de longitud variable, UVLC) y λ es el multiplicador de Lagrange para el cálculo del coste de la tasa [Wiegand03b].

El algoritmo BM no restringe la aplicación de las nuevas técnicas de predicción de la compensación de movimiento del H.264/AVC, como el tamaño de bloque

variable, la resolución de un cuarto de muestra en la estimación de movimiento y el uso de múltiples cuadros de referencia. Solo que el manejo de cada técnica incrementa la intensidad de cómputo y el tiempo de ejecución del algoritmo, por lo que se necesitan utilizar arquitecturas y tecnologías altamente eficientes que minimicen estos efectos.

```
for(v=0; v<Nv/N; v++) { //Nivel de cuadro.
   for(h=0; h<Nh/M; h++) {
      MV(h, v) = (0,0);  //Inicialización del vector de movimiento.
      mcost_min(h, v) = maxvalue; //Inicialización del coste de movimiento mínimo.
      for(m=0; m≤2Ph; m++)  {  //Nivel de la ventana de búsqueda.
         for(n=0; n≤2Pv; n++){
            D(m, n) = 0;  //Inicialización de la medida de distorsión.
            for(j=0; j<N; j++) {  //Nivel de bloque.
               for(i=0; i<M; i++) {
                  D(m,n) = D(m,n) + |c(i,j) – r(i+m, j+n)| //Distorsión.
               }  //end for i.
            }  //end for j.
            R(m,n) = UVLC_TABLA[Diferencia_vector_movimiento_bits]  //Tasa.
            mcost(m,n) = D(m,n) + λ_D R(m,n) //Coste del movimiento.
            if(mcost(m,n) < mcost_min(h, v) {
               mcost_min(h, v) = mcost(m, n); //Nuevo coste de movimiento mínimo.
               MV(h, v) = (m, n);  //Mejor posición.
            }  //end if.
         }  //end for n.
      }  //end for m.
   }  //end for h.
}  //end for v.
```

Figura 2.24. Descripción del algoritmo de ajuste de bloques, resolución de píxeles enteros.

2.6.2 Criterios de ajuste.

La comparación de los bloques actual y candidato en el algoritmo BM se realiza evaluando el nivel de disimilaridad o error de ajuste entre ambos bloques [Shi99]. Para una región en forma de bloque de tamaño M×N muestras, la medición de la disimilaridad (también referida como error, distorsión, diferencia, distancia o energía residual) incluye entre otros los métodos de suma de errores al cuadrado "sum of square errors" (SSE) (2.37), error absoluto medio "mean absolute error" (MAE) (2.38), suma de diferencias absolutas "sum of absolute differences" (SAD) (2.39) y suma de diferencias transformadas absolutas "sum of absolute transformed differences" (SATD) (2.40), donde Cij y Rij son las muestras de intensidad en el bloque actual y el bloque candidato respectivamente [Richardson03] y H_{2D} es la transformada Hadamard 2-D.

$$SSE = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} (Cij - Rij)^2 \tag{2.37}$$

$$MAE = \frac{1}{M \times N} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} |Cij - Rij| \tag{2.38}$$

$$SAD = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} |Cij - Rij| \tag{2.39}$$

$$SATD = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} \left| H_{2D}(Cij - Rij) \right| \quad (2.40)$$

Un estudio basado en trabajos experimentales reportó que el criterio de ajuste no afecta significativamente la búsqueda del bloque menos disimilar [Srinivasan85], por lo que el método del SAD es preferido debido a su simplicidad de realización.

El software de referencia del H.264/AVC [JM1106] utiliza opcionalmente los criterios SAD, SSE y SATD en la estimación de movimiento (SAD como criterio por defecto en píxeles enteros y SATD como criterio por defecto en píxeles fraccionarios). La suma de diferencias transformadas absolutas (SATD) es un método de medición de disimilaridad que utiliza una transformada básica libre de multiplicaciones (Hadamard 2-D) en el lazo de decisión de movimiento. Transformando la diferencia en cada posición de búsqueda incrementa la complejidad de cómputo, pero mejora la exactitud de la medición del residuo de energía.

2.6.3 Procedimientos de búsqueda.

El camino más directo y regular para estimar el movimiento en un codificador de video, es aplicando el algoritmo BM con una estrategia de búsqueda completa "full search" (FSBM), la cual evalúa todos los bloques candidatos dentro de la SW. Este es un proceso de fuerza bruta que logra los mejores resultados, a costa de un procesamiento exhaustivo, que puede derivar en un alto coste computacional, un gran número de accesos a memorias y, en consecuencia, un tiempo de ejecución alto y un creciente consumo de energía.

Para evaluar en forma genérica la carga computacional y la necesidad de ancho de banda del algoritmo FSBM con resolución de píxeles enteros, se define una secuencia de video a una velocidad de f cuadros por segundo (fps), con cuadros de tamaño Nh×Nv píxeles, en donde se realiza la predicción de la compensación de movimiento utilizando bloques de tamaño M×N, dentro de una SW rectangular de tamaño (M+2Ph)×(N+2Pv) y aplicando el criterio de error SAD (el cual consiste básicamente en tres operaciones por píxel candidato: resta, valor absoluto y suma). El número de operaciones por segundo para el proceso de búsqueda se obtiene con la expresión:

$$Op = 3 * (2Ph+1) * (2Pv+1) * Nh * Nv * f \quad (2.41)$$

El número de accesos a memoria, para la lectura de los valores de los píxeles actual y candidato es:

$$Mem = (1 + (2Ph+1) * (2Pv+1)) * Nh * Nv * f \quad (2.42)$$

Como ejemplo, para un formato de video 1080 HD (Nh = 1920, Nv = 1088), con Ph = Pv = 16 y f = 30 fps, se tienen 204×10^9 operaciones por segundo y 68×10^9 accesos por segundo. Estos resultados no toman en cuenta el número de operaciones dependientes de la realización, el número de operaciones y accesos a las memorias para el cálculo del direccionamiento, los resultados de comparación, las decisiones de codificación y el control. Si a lo anterior se le agregan las funciones y herramientas que

incluye el H.264/AVC en sus diferentes perfiles, como el número de bloques y sub-bloques a evaluar para cada MB (41 en total), la ejecución del algoritmo en las resoluciones entera, media y un cuarto de muestra, el uso de múltiples imágenes de referencia, la predicción bi-predictiva que utiliza dos cuadros de referencia en cada predicción y la resolución de los píxeles, de 8 a 14 bits, se presenta un esquema de predicción que no puede aplicarse por su intensidad de cómputo en todos los ámbitos de codificación.

Una alternativa es utilizar algoritmos rápidos, los cuales pueden reducir la carga computacional del FSBM con aceptable calidad de video, utilizando técnicas como la reducción de las posiciones de búsqueda, la simplificación de los criterios de comparación, la búsqueda jerarquica, etc. [Huang06]. En la literatura se han propuesto un gran número de algoritmos, varios de ellos enfocados a su realización software y otros para su desarrollo hardware. Estos últimos buscan una realización VLSI óptima en área y consumo de energía. La figura 2.25 muestra una clasificación de los algoritmos rápidos en función de su principio de operación [Kuhn99].

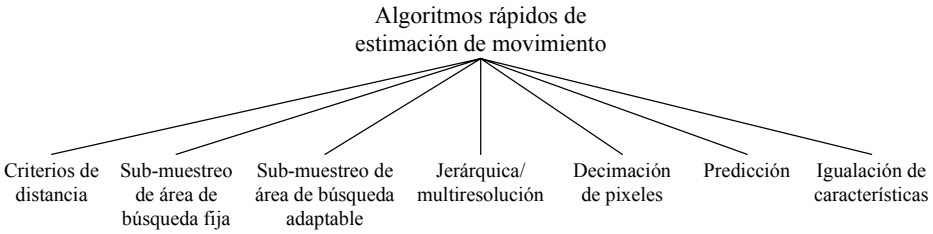


Figura 2.25. Clasificación de los algoritmos rápidos de estimación de movimiento, en función de su principio de operación.

La selección del algoritmo a utilizar es función de la aplicación. Por ejemplo, en el video interactivo en tiempo real, donde cada estación debe estar preparada para comprimir y descomprimir el video con la misma habilidad y retardo mínimo, se prefieren sistemas de codificación con algoritmos rápidos. En los sistemas de flujo de video de un solo sentido, donde en el proceso de captura, codificación y transmisión de la información de video a los destinos remotos se puede tolerar una cierta cantidad de retardo, el codificador tiene tiempo para optimizar el proceso de compresión usando el algoritmo FSBM. Pero los escenarios ideales son la codificación de video fuera de línea para el algoritmo FSBM y el video interactivo en equipos móviles para los algoritmos rápidos, donde los objetivos son la máxima optimización compresión/calidad y consumo de energía/retardo, respectivamente.

El software de referencia JM incluye los siguientes algoritmos de estimación de movimiento:

* Full Search.

* Fast Full Search (algoritmo por defecto).

* Uneven Multi-Hexagon Search (UMHex).

* Simplified Hexagon Search.

* Enhanced Predictive Zonal Search (EPZS).

El primero utiliza la lógica de operación que se definió previamente. El *Fast Full Search* evalúa el coste de distorsión de los sub-bloques 4×4 y con los resultados calcula el coste de distorsión de los bloques y sub-bloques restantes, reduciendo significativamente el coste del cómputo. Los algoritmos UMHex y *Simplified Hexagon Search*, son los métodos rápidos más eficientes que propone el equipo de estandarización JVT. Finalmente, el EPZS es un esquema genérico que se agrega para propósitos académicos y que permite al usuario depurar el algoritmo en función de la aplicación destino, al incluir el siguiente grupo de patrones de refinamiento primario alrededor del mejor predictor:

* Diamond.

* Square.

* Extended Diamond (algoritmo por defecto).

* Large Diamond.

* Subpixel Diamond.

2.6.4 Limitaciones de la técnica *Block Matching*.

El algoritmo BM es la técnica más sencilla, directa y eficiente de estimación de movimiento para codificación de video, pero aun con todas estas cualidades, su modelo básico de movimiento tiene varias limitaciones, que los estándares de codificación deben sobrellevar. Las más importantes son las siguientes [Shi99]:

* El campo de MVs es poco fiable con respecto al movimiento 3-D real, básicamente por la estimación imperfecta de movimiento a lo largo de los límites de movimiento.

* El algoritmo causa artefactos de bloque en forma inherente.

* Se tiene la necesidad de manejar información lateral, al codificar y transmitir los MVs, por lo que es difícil hacer uso de bloques pequeños para obtener una mayor exactitud en la estimación de movimiento.

La estimación de movimiento poco fiable entre los límites de movimiento, causa mayor error de predicción, lo que reduce la eficiencia de codificación. Los artefactos de bloque pueden pasar de una degradación no perceptible para el HVS a altas tasas de bits, donde se transmite un error de predicción suficiente para mejorar el efecto visual subjetivo, hasta una visión desagradable a tasas menores a 64 Kbps. La aceptación de que el movimiento dentro de cada bloque es uniforme requiere utilizar un tamaño de bloque pequeño, lo cual redunda en una gran cantidad de MVs, resultando en una sobrecarga de información lateral.

El estándar H.264/AVC resuelve satisfactoriamente las restricciones de origen del algoritmo BM con el conjunto de herramientas, técnicas y métodos que incluye. La primera, al permitir que los MVs rebasen los límites de la imagen, aplicando una técnica de extrapolación en los límites de los cuadros de referencia. La segunda, con el uso de un filtro antibloqueo en el lazo de predicción, que elimina los artefactos de bloque y

mejora la calidad subjetiva y objetiva del video resultante. Y la tercera, con el uso del tamaño de bloque variable, para relacionar el tamaño de los bloques con el nivel de detalle de la imagen y optimizar el número de MVs.

Capítulo **3**

Metodología y Herramientas de Diseño

3.1 Sistemas hardware de procesamiento de imágenes y video.

La demanda de los consumidores hacia las aplicaciones multimedia con altas funcionalidades de tiempo real, mayor interconexión, menor coste y bajo consumo de energía, impone gran presión en los diseñadores, quienes deben satisfacer estos requerimientos, mientras mejoran la facilidad de desarrollo y la flexibilidad del producto. Y lo deben de hacer en un ambiente de complejidad creciente y dentro del criterio de más prioridad del proceso, el tiempo al mercado [Adams02].

En el ámbito de los sistemas de imágenes y video, la industria se está moviendo hacia las aplicaciones de alta definición (HD) y baja tasa de bits, donde se tienen seis veces más datos a procesar con respecto a la definición estándar (SD) y hasta 5 veces mas complejidad computacional para reducir a la mitad la tasa de bits, utilizando las normas del estado del arte en la codificación de video (H.264/AVC, AVS, AV-1, etc.), con respecto al tradicional MPEG2 [Greene07]. Afortunadamente, han proliferado una gran variedad de plataformas tecnológicas, que en conjunto, satisfacen los requerimientos de las aplicaciones modernas.

En los sistemas de procesamiento de imágenes y video, las alternativas de realización hardware incluyen circuitos de aplicación especifica, soluciones programables y procesadores configurables, entre los que se encuentran el circuito integrado de aplicación específica (ASIC), el procesador de señales de aplicación específica (ASSP), el procesador digital de señales (DSP), la matriz de puertas programable en campo (FPGA) y los procesadores de proposito general (GPP). Cada enfoque tiene ventajas y desventajas, con la selección final dependiendo de la satisfacción de las necesidades de la aplicación y la factibilidad de la solución.

3.1.1 Características de los sistemas.

De acuerdo a los requerimientos del mercado, un sistema hardware de procesamiento de imágenes y video debe tener características de alto rendimiento, buen precio, facilidad de desarrollo, bajo consumo de energía, flexibilidad, disponibilidad de periféricos, etc. [Jentz06]. Estos requisitos, junto con el criterio de tiempo al mercado, pueden utilizarse para determinar la mejor plataforma y dispositivo para una aplicación específica [Adam02].

El tiempo al mercado es el motor que está activando a la industria, por lo que es el punto más crítico en el ciclo de desarrollo de un producto, alimentado por el estilo

moderno de vida de la población, de consumir y desechar, lo que está reduciendo el ciclo de vida de muchos productos, de años a meses.

El rendimiento normalmente determina la capacidad de un producto. A menudo los productos son diferenciados por algunas características que dependen del rendimiento: cantidad de funciones que ofrecen, área de cobertura, velocidad de operación, calidad de compresión, etc. Generalmente el rendimiento se mide en millones de operaciones por segundo (MIPS), millones de multiplicaciones y acumulaciones por segundo (MMACS), o por los millones de ciclos por segundo (MHz) de la frecuencia del reloj.

El precio es en muchos casos el criterio de prioridad mas obvio, pero generalmente se ubica ligeramente atrás del tiempo al mercado y rendimiento. El precio impacta la lista de materiales y los costes del producto en la producción recurrente. A mayor volumen de producción y consumo popular, mas alta es la prioridad del precio. Por otro lado, mientras el producto es más especializado y dedicado a un nicho específico del mercado, se tiene menor sensibilidad al precio, pero se incrementa la atención en los costes de derechos de propiedad.

La facilidad de desarrollo es un criterio que está aumentando de importancia, por su relación con el soporte, herramientas y costes. Sus componentes principales son el soporte técnico, las relaciones con terceras partes, el entrenamiento técnico, las herramientas de software, la documentación, el tiempo de ingeniería, los costes de ingeniería no-recurrente (NRE), etc. Mientras mejor soporte técnico tenga un diseñador, es más fácil que se enfoque en la innovación y en la disminución del tiempo al mercado. Los diseñadores que tengan la opción de comprar contra hacer o gastar en consultar a un experto contra hacer todo por ellos mismos pueden evaluar la mejor solución, ahorrando esfuerzo y dinero. La facilidad de desarrollo permite obtener mayor calidad en el producto final, ya que el tiempo y el dinero se invierten en su diferenciación más que en generar los principios de operación.

El consumo de energía también está creciendo en importancia, principalmente en dos segmentos del mercado: en equipos portátiles, para aumentar el tiempo de vida de las baterías, característica muy significativa para el usuario final, y en aplicaciones de infraestructura, para reducir la disipación de calor, lo que permite incrementar la densidad de equipo electrónico en un mismo gabinete.

La flexibilidad es la capacidad de modificar o agregar nuevas características para satisfacer el cambio de requerimientos, lo cual ocurre frecuentemente en la época actual. Por ejemplo, la entrada al mercado de productos basados en un estándar, antes de que este madure, es de suma importancia para que los fabricantes encabecen el mercado. Sin embargo, los diseñadores deben desarrollar productos que puedan ser actualizados fácil y eficientemente para cumplir con la sintaxis final del estándar. La flexibilidad da a los diseñadores la capacidad de formalizar la convergencia entre los consumidores, las comunicaciones y la funcionalidad de cómputo, según lo dicta el mercado.

Otro criterio de evaluación lo representan los periféricos, los cuales son elementos muy relevantes en los diseños y aplicaciones actuales. Mientras los productos y estándares evolucionan, los tipos y capacidades de los periféricos deben crecer

exponencialmente. Por lo tanto, la disponibilidad o carencia de periféricos se convierte en un punto clave en el desarrollo del sistema.

3.1.2 Soluciones del mercado a los sistemas de codificación H.264/AVC.

Desde la aprobación de su primera versión (marzo 2003), el H.264/AVC ha pasado por un proceso continuo de crecimiento y maduración, que lo ha convertido en un estándar de línea en los sistemas multimedia actuales. Es por ello que un gran número de empresas tecnológicas se han enfocado en desarrollar productos bajo esta recomendación. Una muestra de los sistemas hardware disponibles se presenta en la tabla 3.1, donde se listan varias plataformas y dispositivos que ofrecen compañías de clase mundial. De acuerdo a esta información, se tienen las siguientes observaciones:

* Las empresas han desarrollado soluciones de codificación H.264/AVC para las principales plataformas de sistemas digitales, a partir de cores IP para lógica programable, aceleradores hardware para SoC, software para DSPs especializados, tarjetas PCI para PC y procesadores multimedia para equipos en general.

* Para la realización del H.264/AVC, se tienen las opciones de circuitos de aplicación especifica (ASIC), soluciones programables (DSP, FPGA, FPOA, GPP) y procesadores configurables, por lo que el mercado no esta dominado por una tecnología en particular.

* La tendencia del mercado es hacia los formatos de alta definición, con perfil alto, y nivel de 4 en adelante, sin descuidar el formato de perfil base y niveles 3 y 4, con versiones de video entrelazado y progresivo.

* Los diseñadores reducen la complejidad de cómputo de la estimación de movimiento con resolución entera utilizando algoritmos rápidos propietarios [XilinxMEE07] [On2_07], algoritmos rápidos generales [MathStar07][OL06][LSI03] y algoritmos no especificados [Ambarella07][Freescale07][Fujitsu07][GDT07][TI05][Vitecmm07] [W&W06][4i2i05].

* En la estimación de movimiento con resolución fraccionaria, aun cuando la mayoría de los fabricantes indica el cumplimiento de la resolución de ¼ de píxel, solo uno de ellos [4i2i05] documenta el algoritmo utilizado (FSBM).

* Otros métodos que los diseñadores aplican para reducir la complejidad computacional consisten en limitar la cantidad de particiones y sub-particiones del MB [XilinxMEE07][MathStar07][GDT07][Freescale07][OL06][LSI03], mantener al mínimo nivel el número de cuadros de referencia [XilinxMEE07][MathStar07] [Freescale07][OL06][LSI03] y minimizar el tamaño de la ventana de búsqueda [MathStar07][On2_07][GDT07][Freescale07][OL06].

En función de estas observaciones, se puede señalar que el mercado ofrece toda una gama de codificadores H.264/AVC, que cubren la mayoría de las plataformas de sistemas digitales, sin el predominio de una tecnología especifica, para aplicaciones que van desde equipos portátiles de baja tasa de bits hasta sistemas profesionales. El inconveniente es que estas soluciones distan de ser optimas, no sólo por el uso de algoritmos rápidos, sino también por la forma en que se aplican dos de las principales

Tabla 3.1. Plataformas y dispositivos de codificación H.264/AVC en el mercado.

Plataforma y/ó dispositivo	Compañía y fecha de introducción	Tecnología	Características de codificación H.264	Características de estimación de movimiento	Aplicaciones principales
Core IP H.264 Motion Estimation engine	Xilinx Agosto 2007 [XilinxMEE07]	FPGA	Perfiles base, principal y alto Nivel 4.2 1080p HD 30 fps 275 MHz	Píxeles enteros Part. 8×4 960 puntos por MB 1 cuadro de ref. Ph=56, Pv=48	Sistemas de vigilancia, videoconferencias, difusión de video
SoC MB86H51	Fujitsu julio 2007 [Fujitsu07]	ASIC 90 nm	Perfil alto nivel 4.0 1080i HD 60/50 fps	-	Equipos portátiles, equipos de red en casa
Core IP MIP-H2E02-P12	MathStar abril 2007 [MathStar07]	FPOA	Perfil alto 4:2:2 Nivel 4.1 1080i/720p HD 60 fps 800 MHz	Búsqueda jerárquica Part. 16×16 a 8×8 2 cuadros de ref. Ph=512, Pv=256	Equipo profesional de video, servidores de video
PC (Tarjeta PCI) VMC-5400	VITEC 2007 [Vitecmm07]	Multi-DSPs	Perfil principal Nivel 3.0	-	Aplicaciones profesionales de video
Core IP (RTL) Hantro 6280	On2 2007 [On2_07]	FPGA	Perfil base Nivel 3.1 720p HD 30 fps	Full-search indexado Ph=Pv=16	SoCs inalámbricos
Core IP H.264EBH	GDT 2007 [GDT07]	ASIC 0.18 µm ó FPGA	Perfil base Nivel 3.1 SXGA ASIC 4CIF FPGA 30 fps	4 MVs por MB Part. 16×16 a 8×8 Ph=Pv=16	Video vigilancia, HDTV, equipos portátiles
Procesador de aplicaciones multimedia i.MX27	Freescale 2007 [Freescale07]	ASIC CMOS 90 nm	Perfil base 525 SD, 4CIF 25 fps	Part. 16×16 a 8×8 1 cuadro de ref. Ph=Pv=16	Video y voz sobre IP, equipos portátiles sistemas embebidos
SoC A2 Chip	Ambarella 2007 [Ambarella07]	ASIC	Perfiles principal y alto 480i SD 720p, 1080i 60 fps	Un sector por cuadro	Cámaras de video
SoC WW10000BA	W & W Communications 2006 [W&W06]	ASIC	Perfil base 1080p HD 30 fps 110 MHz	Paso único múltiples bloques Ph=184, Pv=120	Video conferencias
Core IP OL_H264MCE	Ocean Logic 2006 [OL06]	0.13 µm ASIC/ FPGA	Perfil base Nivel 4.1 1080p HD ASIC 720p HD FPGA 30 fps 250 MHz	Algoritmo 4 pasos Part. 16×16 1 cuadro de ref. Ph=Pv=16	Equipos portátiles, Sistemas de vigilancia, HDTV

Tabla 3.1. Continuación.

Plataforma y/ó dispositivo	Compañía y fecha de introducción	Tecnología	Características de codificación H.264	Características de estimación de movimiento	Aplicaciones principales
Core IP H.264 Encoder	4i2i Communications Ltd. Agosto 2005 [4i2i05]	FPGA	Perfil base 4CIF 30 fps 56 MHz	Píxeles enteros: Algoritmo rápido 1 a 5 cuadros de referencia Ph=Pv=256 Píxeles fracc: FSBM	CCTV, video-conferencias, video-telefonía, video-cámaras
SoC TSM320DM6446	Texas Instruments 2005 [TI05]	Procesador + DSP	Perfil base 4CIF 30 fps	-	Sistemas multimedia
PC (Tarjeta PCI) VLE4000	LSI LOGIC 2003 [LSI03]	FPGA	Perfil principal Nivel 3 4CIF 30 fps	Búsqueda jerárquica Part. 16×16 a 8×8 2 cuadros ref.	Difusión de video Video sobre IP DVD

técnicas que permiten alcanzar la eficiencia del H.264/AVC con respecto a sus predecesores: la compensación de movimiento con tamaño variable de bloques y el uso de múltiples imágenes de referencia.

Aun cuando los fabricantes no incluyen la evaluación de la eficiencia de compresión con respecto a la aplicación completa del estándar, los resultados experimentales muestran que la cancelación de las sub-particiones menores a 8×8 píxeles incrementa 2.12% en promedio la tasa de bits con respecto al uso de todos los tamaños de bloque, para una calidad similar [Puri04], mientras que la utilización de una imagen de referencia con respecto al manejo de 2, 3 y 5 cuadros de referencia, aumenta un promedio de 2.68, 3.6 y 4.17% la tasa de bits, respectivamente. Si a esto se le agrega el uso de algoritmos rápidos de estimación de movimiento, que no son precisamente los más eficientes en su tipo, la degradación total es significativa.

Los resultados de este estudio básico, revelan que la investigación y desarrollo de sistemas hardware, que cumplan íntegramente con la sintaxis de la norma H.264/AVC y en particular, con los métodos y técnicas de predicción de la compensación de movimiento, es un campo joven que necesita el esfuerzo de investigadores y diseñadores, para obtener equipos y dispositivos que satisfagan adecuadamente las expectativas del mercado.

3.1.3 Realización del estándar H.264/AVC sobre dispositivos digitales.

Dentro de las soluciones tecnológicas, las FPGAs, los CPUs/DSPs embebidos y los GPPs, son los componentes digitales más comunes para la realización del estándar H.264/AVC. La selección del dispositivo depende de la aplicación, la cual dicta los requerimientos de densidad de procesamiento, consumo de energía y costes, entre otros.

En este tipo de soluciones, el tiempo de realización del sistema y el coste dominante son determinados por el desarrollo de software [Hawkins06].

Los GPPs se caracterizan por su naturaleza de procesamiento genérico y su topología jerárquica de memoria, con el uso de unidades de manejo de memoria para proporcionar espacios de direccionamiento virtual, pero con una configuración de acceso aleatorio que los hace inadecuados para algunas aplicaciones de tiempo real. Sin embargo, el gran número de ingenieros de software familiares con esta plataforma y las tarjetas electrónicas disponibles como co-procesadores, los hacen adecuados para la codificación H.264/AVC.

Los DSPs son dispositivos perfeccionados para el procesamiento *pipeline* de un flujo continuó de señales muestreadas, aunque generalmente son poco eficientes en las tareas de control, por lo que una alternativa que ofrecen algunos fabricantes es la conjunción de un GPP y un DSP de alto rendimiento en un ambiente SoC, lo cual es una solución avanzada para la codificación H.264/AVC.

Recientemente, la FPGA se ha convertido en un dispositivo muy adecuado para el procesamiento de imágenes y video, debido a su gran capacidad de procesamiento paralelo, para la ejecución de operaciones repetitivas sobre grandes volúmenes de datos, a frecuencias de operación en el rango de 50 a 300 MHz. Su alta capacidad se debe a la gran densidad de celdas lógicas y la eficiencia de los elementos de interconexión, con la desventaja de una disipación de potencia significativa. La FPGA puede utilizarse para realizar codificadores H.264/AVC en soluciones *stand-alone*, que incluyen un GPP integrado por hardware o software, o pueden servir como co-procesador de DSP o GPP para la misma aplicación.

Los sistemas a gran escala consisten generalmente de una combinación jerárquica de GPPs, DSPs y FPGAs. Una política en el diseño de estos sistemas, es enfocar el desarrollo de hardware a las tareas de tiempo real que excedan la capacidad del GPP. Las tareas en tiempo real, como la codificación H.264/AVC, pueden ser realizadas por dispositivos DSPs o FPGAs, dedicando el GPP a la interfase hombre-máquina y al control de los elementos de bajo nivel.

Las arquitecturas de estimación de movimiento propuestas en este trabajo son realizadas en dispositivos FPGAs, por lo que en las siguientes secciones se documentan algunos conceptos introductorios, junto con la metodología y las herramientas utilizadas en el diseño.

3.2 Sistemas embebidos.

Los sistemas embebidos son sistemas de cómputo diseñados para realizar funciones específicas. Generalmente no son usados o percibidos como computadoras, ya que típicamente no tienen la interfase teclado-monitor para interactuar con el usuario, no ejecutan sistemas operativos estándar y no pueden ser programados por el consumidor final, sino que incluyen una interfase más natural, un sistema operativo de tiempo real (RTOS) y son programados en fabrica, con programas residentes en firmware (en memoria flash o ROM). Algunas veces, estos sistemas constituyen un producto auto-contenido, como un teléfono móvil o forman parte de otro sistema,

agregando funcionalidades avanzadas a los equipos existentes, como en los automóviles [Zurawski06].

Desde el sistema de gobierno de los cohetes espaciales del proyecto *Apollo* de la NASA, en la década de los 1960s [WikipediaES], hasta la última generación de equipos de comunicación y electrodomésticos, los sistemas embebido se han desarrollado espectacularmente, principalmente porque llevan las ventajas de la ley de Moore a la vida diaria, con un incremento exponencial de su funcionalidad y rendimiento y un decremento continuo de su coste. Esto es posible gracias a la capacidad tecnológica y de manufactura de los circuitos integrados, que permite fabricar dispositivos cada vez más complejos, y al desarrollo de nuevas metodologías de diseño, que contribuyen al uso eficiente e inteligente de estos dispositivos y su aplicación en casi todos los productos de uso común. A partir de la década de los 1980s, los sistemas embebidos han sido la norma, mas que la excepción, para casi todos los equipos electrónicos, una tendencia que ha continuado hasta la fecha.

Existe gran diversidad de sistemas embebidos. Por ejemplo, un automóvil moderno contiene decenas de componentes electrónicos que realizan tareas relacionadas al aspecto mecánico, como el control del motor, el sistema de frenos antibloqueo y el control de la suspensión y transmisión, además de tareas enfocadas al confort del conductor y los pasajeros, como los sistemas de navegación, los aparatos de audio y video, el control del clima, etc. Otros ejemplos vienen de la industria de la comunicación: un teléfono celular es una computadora sofisticada, cuya principal tarea es enviar y recibir voz, imágenes y mensajes multimedia, y que incluye otras funciones atractivas para el usuario, como cámara fotográfica, asistente personal, consola de juegos y equipo de conexión a Internet. Otros sistemas embebidos que han impactado el estilo de vida de la población son las máquinas de punto de venta, que modificaron la forma de pagar las compras y los sistemas multimedia, que cambiaron la manera en que se disfruta la música y los videos.

3.2.1 Diseño de sistemas embebidos.

Los sistemas embebidos pueden ser definidos como una colección de componentes programables alrededor de un ASIC y otros dispositivos estándar (Figura 3.1), que interactúan continuamente con el entorno a través de sensores y actuadores [Zurawski06]. La colección esta compuesta por un grupo de *chips* en una tarjeta o un conjunto de módulos IP en un circuito integrado. El software es utilizado para configurar las características con alta flexibilidad y el hardware para incrementar el comportamiento y reducir el consumo de energía. Los componentes programables principales son los microprocesadores y DSPs, donde se lleva a cabo la partición de software del sistema. Se pueden incluir componentes reconfigurables en tiempo de ejecución (FPGA), que exhiben características intermedias de área, coste, comportamiento y consumo entre el hardware dedicado y los procesadores. Todos los componentes son conectados vía buses y redes estándar y/o dedicadas, y los datos son almacenados en un grupo de memorias.

Los sistemas embebidos presentan típicamente las siguientes características:

* Son sistemas de baja flexibilidad diseñados para realizar siempre la misma tarea.

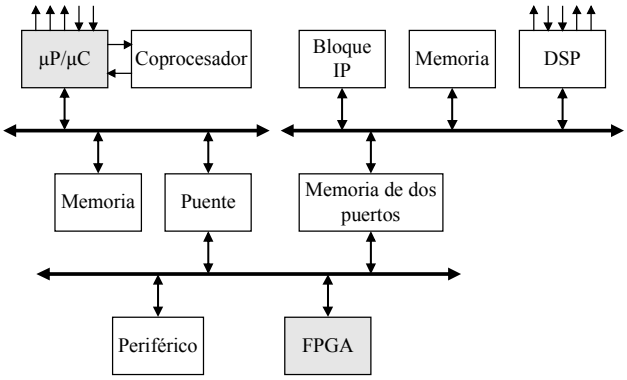


Figura 3.1. Arquitectura general de un sistema embebido.

* Generalmente son parte de un gran sistema de control.

* Los criterios de coste, confiabilidad y seguridad, son a menudo más importantes que el rendimiento, ya que el usuario, que no está conciente de su presencia en el sistema, valora características como coste, facilidad de uso y tiempo de vida del producto.

* Los sistemas embebidos necesitan ser diseñados en un periodo extremadamente corto, para satisfacer el criterio de tiempo al mercado. Un pequeño retardo en el ciclo de diseño puede ser la diferencia entre el éxito o el fracaso.

* Su desarrollo esta basado en la experiencia previa con productos similares y en el diseño manual, a partir de un proceso que requiere diversas iteraciones hasta llegar a la convergencia, ya que normalmente el sistema no está especificado rigurosamente.

* La gran mayoría son sistemas de tiempo real, por lo que las especificaciones de tiempo son críticas.

3.2.1.1 Metodología de diseño abstracción-agrupación.

La metodología tradicional de diseño de sistemas embebidos se compone de una secuencia de dos pasos: abstracción y agrupación. En la abstracción se describe un objeto utilizando un modelo, donde normalmente se ignoran los detalles de bajo nivel. En la agrupación se conectan varios modelos del mismo nivel de abstracción para obtener un objeto nuevo, con propiedades que no son parte de los modelos independientes que lo constituyen. Con la aplicación sucesiva de estos dos pasos, los diseños de electrónica digital han pasado de la distribución de capas de silicio, a esquemas con transistores, a conexiones de puertas lógicas y a descripciones al nivel de transferencia de registros (RTL) y de comportamiento.

La metodología de abstracción-agrupación se apoya en el concepto de plataforma. Una plataforma consiste en una serie de componentes junto con las reglas de operación que aseguran su correcta interoperabilidad. Utilizada para acelerar el tiempo al mercado, la plataforma garantiza el rápido desarrollo de arquitecturas complejas, al definir modelos abstractos que ocultan los detalles de realización e integración de los elementos de bajo nivel. Un ejemplo de plataforma es una familia de microprocesadores, periféricos y protocolos de bus, que permite a los desarrolladores

diseñar aplicaciones sin necesidad de un conocimiento detallado de la operación interna de cada elemento, compartiendo los costes de diseño y fabricación entre un amplio rango de usuarios potenciales.

La figura 3.2 muestra la representación esquemática de la metodología de diseño, derivada de la secuencia abstracción-agrupación. En el nivel funcional, se especifica, diseña y analiza el funcionamiento del sistema. En paralelo, se seleccionan y agrupan los componentes de la plataforma que formaran su arquitectura. Los componentes se obtienen de librerías existentes o pueden ser objetos a diseñar posteriormente.

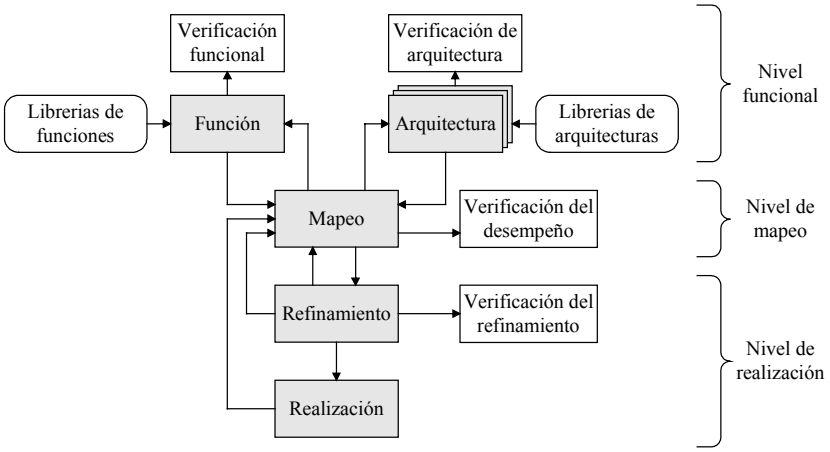


Figura 3.2. Metodología de diseño abstracción-agrupación de un sistema embebido.

En el nivel de mapeo, donde se define la relación entre la aplicación y los elementos de la plataforma, las operaciones funcionales son asignadas a los componentes de la arquitectura y los patrones proporcionados por la misma son seleccionados para las comunicaciones. En este nivel se verifica el comportamiento de la realización seleccionada, con un mayor grado de detalle con respecto a la verificación funcional. Diferentes mapeos a la misma arquitectura o el mapeo a diferentes arquitecturas, permiten explorar el espacio de diseño para encontrar la mejor solución a los más importantes cambios del diseño. Estos análisis permiten identificar y corregir problemas potenciales al inicio del ciclo de diseño, para eliminar errores y defectos posibles. En esta etapa, es muy importante definir la organización de las unidades de almacenamiento de datos del sistema, balanceando el coste y rendimiento de las memorias seleccionadas. El mapeo de las estructuras de datos a memorias diferentes y el cambio de la organización y distribución de las redes puede tener un gran impacto en la reducción del tiempo de ejecución de un algoritmo.

En el nivel de realización, en un proceso inverso a la abstracción y agrupación, se genera una especificación de bajo nivel del sistema embebido, por medio de una serie de refinamientos y modificaciones manuales o automáticas que agregan sucesivamente más detalles, mientras se verifica la satisfacción de los requerimientos de alto nivel. Este paso no necesariamente genera un producto para manufactura, ya que también puede describir un nuevo componente para una aplicación mayor, en un proceso recursivo.

3.2.1.2 Diseño al nivel de sistema electrónico.

Paralelamente a los avances en los semiconductores, las metodologías de diseño también han evolucionado. En particular, un nuevo paradigma conocido como diseño electrónico al nivel de sistema (ESL) está revolucionando la industria, con herramientas y metodologías que elevan el grado de abstracción del diseño sobre los lenguajes actuales del nivel RTL. El éxito del diseño ESL se debe principalmente a que satisface los siguientes escenarios [Lass06]:

* Las herramientas ESL proporcionan una conversión automática de lenguajes de alto nivel (como C/C++ o MATLAB) a RTL y puertas.

* El diseño ESL ofrece una solución lógica a las limitaciones de potencia y rendimiento de los procesadores, para manejar la complejidad computacional de las aplicaciones actuales y futuras, al definir una ruta fácil y automática a la aceleración basada en hardware.

* Las metodologías ESL habilitan una rápida simulación del sistema, al utilizar modelos basados en procesos de muy alta velocidad, que permiten verificar la funcionalidad y ejecutar análisis del comportamiento hardware/software en forma prematura en el ciclo de diseño.

* Cuando los cambios en el diseño se presentan en las ultimas etapas del ciclo de desarrollo del producto, la metodología ESL apoya firmemente al diseñador para responder con rapidez.

Pero el punto central del diseño ESL no son las herramientas y niveles de abstracción, sino la productividad, por lo que tiene un fuerte impacto en el tiempo al mercado del producto final. En las últimas dos décadas, el diseño de hardware electrónico ha sido muy ineficiente, ya que mientras el número de puertas de las plataformas de diseño digital se ha incrementado por un factor de 2000, cumpliendo la ley de Moore, los avances de la automatización del diseño electrónico (EDA) han logrado aumentar solo 400 veces el número de puertas aplicadas por diseñador por día, en el mismo periodo de tiempo [Morris06]. Uno de los objetivos de las metodologías ESL es que la productividad de los diseñadores pase de un crecimiento lineal a uno exponencial, como el de puertas en los *chips*.

3.2.2 Sistemas en un solo *chip*.

En los últimos años se ha tenido una notable evolución de los sistemas embebidos, al pasar de componentes discretos ensamblados sobre tarjetas de circuito impreso a componentes de propiedad intelectual (IP), alojados en el silicio de un solo *chip* (sistema en un solo *chip*, SoC), con lo que se obtiene un mayor potencial para integrar funciones complejas, en operaciones independientes o distribuidas [Zurawski06]. El SoC ha emergido como el reemplazo ideal de los equipos multi-*chips*, prometiendo una solución con menor consumo de energía, menor espacio en tarjeta, integración más simple y más ancho de banda de E/S, con el menor número de componentes, todo lo cual es muy atractivo para los productos embebidos [Siewert07].

3.2.2.1 Conceptos de sistemas en un solo *chip*.

La definición de SoC es simplemente un *chip* donde se diseña un sistema completo en una sola ASIC configurable, con una reutilización significativa de componentes IP, lo cual acelera el tiempo al mercado. Alternativamente, el SoC es realizado como ASIC reconfigurable, utilizando una combinación de FPGA y GPP por hardware o software para una mayor flexibilidad. El desarrollo del SoC sigue claramente las leyes de Moore, donde las cada vez mayores escalas de integración han hecho que las tarjetas de circuito impreso se conviertan en grupos de *chips* y estos en un solo componente.

Uno de los conceptos más poderosos atrás del diseño SoC es que la funcionalidad puede ser especificada y asignada no sólo al software, sino también al hardware. A menudo, en el curso de desarrollo de un SoC, las funciones son primero definidas en software y entonces trasladadas a hardware para efectos de aceleración. Cuando la funcionalidad es menos madura, es útil realizarla como software para que pueda ser fácilmente actualizado. Una vez que la funcionalidad es bien conocida y optimizada, la realización en hardware a menudo toma sentido.

Específicamente, los arquitectos del SoC inician su trabajo utilizando herramientas de análisis de alto nivel, para explorar la funcionalidad completa del sistema y obtener la mejor asignación de funciones a hardware y software, para definir el tamaño correcto de los recursos de hardware, de tal forma que el software pueda satisfacer los requerimientos de rendimiento. Algunas decisiones de la configuración del SoC alteran el rendimiento del software, incluyendo:

* La asignación de funciones en software para la ejecución en un GPP o su asignación a una máquina de estados.

* La potencia de cómputo requerida por las funciones software, para el uso de un solo GPP o el manejo de multiprocesamiento.

* La jerarquía de memoria.

* La interconexión con elementos externos y periféricos de E/S.

El desarrollo de software para SoC involucra la partición de la aplicación entre los diversos elementos procesadores, de acuerdo al modelo computacional más eficiente. Para establecer la mejor partición, se puede requerir un proceso de prueba y error. En un alto nivel, la lógica de partición puede ser la siguiente [Oshana07]:

* Colocar el software de la máquina de estado (esos algoritmos que proporcionan el control de la aplicación, la secuenciación, el control de las interfases del usuario, etc.) en un GPP.

* Ubicar el software de procesamiento de señales en un DSP.

* Situar los algoritmos de mayor tasa y alto coste de cómputo en aceleradores hardware.

El SoC pueden soportar una gama de aplicaciones que va desde aquellas que solo necesitan maximizar la tasa de transferencia hasta las que deben satisfacer requerimientos de tiempo muy específicos. En general, los SoC son sistemas de tiempo real (RTS), siendo los RTS la aplicación dominante de las tecnologías de cómputo, ya que el 99% de los procesadores fabricados son usados en dispositivos embebidos. No existe un acuerdo sobre la definición de un RTS, pero la siguiente caracterización es una aproximación aceptada:

* Los RTS son sistemas de cómputo que interactúan físicamente con el mundo real, típicamente vía sensores y actuadores.

* Los RTS tienen requerimientos temporales debido a que la interacción con el mundo real debe ocurrir en un tiempo específico.

Los sistemas en tiempo real son siempre más fáciles de realizar en un ambiente donde abundan los recursos. Si es posible, se debe mantener un diseño SoC simple, de tal forma que cada servicio que proporcione el sistema tenga recursos dedicados. Pero el coste, consumo de energía, disipación térmica y los requerimientos distribuidos restringen la simplificación del diseño [Siewert07].

3.2.2.2 Aplicaciones SoC en el procesamiento de video.

Los medios digitales, incluyendo audio, video y gráficos en tiempo real, ofrecen a los diseños SoC un mercado amplio en oportunidades. En el ámbito del video, estos sistemas pueden acelerar significativamente la revolución del procesamiento digital del video, ya que gran parte de las soluciones de video digital solo podrían llegar a los consumidores como arquitecturas embebidas realizadas sobre SoC. Estos sistemas están bien posicionados para satisfacer las necesidades humanas de generar, usar y compartir los contenidos de alta fidelidad que involucra el sentido de mayor ancho de banda, la vista, con fiabilidad, calidad, seguridad y privacidad [Siewert07].

Un ejemplo se tiene en la tecnología de sensores de imagen inteligentes [Anafocus07], los cuales integran todas las estructuras para conformar un sistema de visión en un solo *chip* (VSoC), el cual es definido como un sistema cuyas entradas son un conjunto de señales analógicas 2-D representando un flujo visual, en las que se realiza un pre-procesamiento analógico antes de su conversión al dominio digital para su post-procesamiento. Las operaciones típicas de los VSoC son:

* Capturar (sensar) imágenes.

* Mejorar la operación del sensor.

* Realizar un procesamiento espacial-temporal en el flujo de imágenes.

* Interpretar la información contenida en el flujo de imágenes.

* Tomar decisiones en función de los resultados de la interpretación.

Las especificaciones de los VSoC compiten con las soluciones multi-*chip* convencionales, agregando nuevas características que los hacen muy atractivos para el mercado:

* Adquisición de imágenes con alto rango dinámico.

* Arquitectura todo-en-uno de propósito general, incluyendo sensores ópticos, procesadores, memorias, convertidores de datos y periféricos de control y comunicaciones.

* Alta potencia computacional con bajo consumo de energía, en un sistema compacto.

* Operación de gran flexibilidad, con programación en lenguajes de alto nivel para satisfacer aplicaciones específicas.

3.3 Dispositivos FPGA.

Uno de los sectores con mas crecimiento en la industria de los semiconductores es el de los dispositivos de lógica programable (PLD), como los CPLD (Dispositivos de Lógica Programable Complejos) y la FPGA (Matriz de Puertas Programables en Campo). Desde su creación, en la década de los 1970s, los PLD se han caracterizado por ofrecer la mayor flexibilidad de diseño, ya que son dispositivos cuya arquitectura interna es predeterminada por el fabricante, pero que son creados de tal forma que puedan ser configurados por los ingenieros en campo, para ejecutar una variedad de funciones [XilinxDK03].

3.3.1 Definición.

Después de las primeras versiones de PLDs, donde se ofrecieron arquitecturas con planos AND/OR programables y diversos niveles de interconexión de suma de productos y arquitecturas complejas (CPLDs), que extendieron la densidad de los primeros componentes, en 1984 salió al mercado un nuevo dispositivo programable, la FPGA XC2000 de Xilinx, con una arquitectura que superó los estándares de su tiempo. Este dispositivo, concebido por Ross Freeman, fundador de Xilinx, revolucionó la tecnología al abandonar la restricción de suma de productos y utilizar módulos combinacionales configurables a los que llamó LUT (*Look-Up-Table*), cada uno acompañado por un *Flip-Flop* (FF) e interconectado por conexiones programables [Xcell01].

Los dispositivos FPGA son circuitos integrados digitales que contienen bloques de lógica programable e interconexiones configurables entre estos bloques. Dependiendo de su realización, algunas FPGA pueden ser programadas una sola vez y otras pueden ser reprogramadas un sinnúmero de veces. El concepto de “programables en campo”, se refiere al hecho de que su configuración puede tomar lugar en el laboratorio o en el sistema electrónico que lo incluye [Maxfield04] y el concepto “matriz de puertas”, hace alusión a la estructura interna que posibilita su reprogramación.

3.3.2 Arquitectura.

Las FPGA se caracterizan principalmente por su estructura lógica, tamaño (número de bloques lógicos y terminales de E/S), recursos especiales de procesamiento, almacenamiento y control y por su velocidad y consumo de energía.

Las FPGA están basadas en una matriz de bloques lógicos programables (también llamados bloques lógicos configurables CLBs o bloques matriciales lógicos LABs) acoplados por líneas de interconexión y cajas de conmutadores (Figura 3.3). Las funciones digitales que el usuario define son mapeadas a uno o más de los bloques lógicos. Las líneas de interconexión configurables forman parte de los recursos de rutado del dispositivo, los cuales son la clave de la flexibilidad de la FPGA, pero que también representan un compromiso entre flexibilidad de programación y eficiencia de área. El rutado incluye típicamente una jerarquía de canales que van desde las líneas de alta velocidad, hasta las dedicadas para la difusión de la señal de reloj y *reset*. Los conmutadores programables, que pueden ser basados en RAM, borrables eléctricamente o programables una sola vez, habilitan la conexión de las líneas de rutado y de los recursos internos y elementos externos, reduciendo al mínimo el retardo de red.

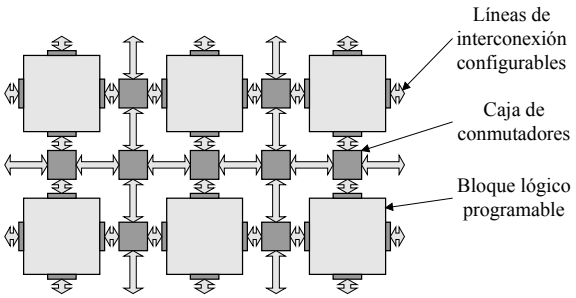


Figura 3.3. Estructura lógica de la FPGA.

Los bloques lógicos son formados por un grupo de celdas o elementos lógicos (LC o LE), compuestos generalmente de un registro FF, un multiplexor y un elemento programable, la tabla *Lookup* (LUT), que realiza cualquier función combinacional de *n* entradas-una salida, siendo *n* igual a 3, 4, 5 ó 6 (Figura 3.4). Cada celda lógica en la FPGA puede ser configurada para tomar un valor inicial de 1 ó 0 y actuar como un FF o un candado. El multiplexor que alimenta al FF puede ser programado para aceptar la salida de la LUT o la entrada independiente. Si la opción seleccionada es el FF, el registro puede ser configurado para actuar en el borde positivo o negativo del reloj. La estructura detallada de este elemento lógico básico depende de cada fabricante.

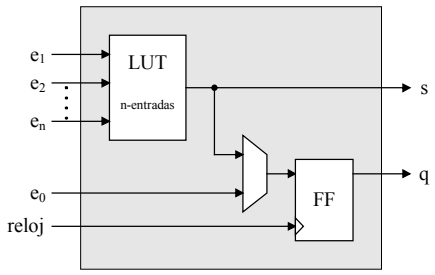


Figura 3.4. Celda o bloque lógico programable de la FPGA.

Los dispositivos FPGA más modernos incorporan recursos especiales, como bloques de RAM, multiplicadores, elementos de multiplicación y acumulación (MAC), módulos de comunicación gigabits, bloques DSP y hasta procesadores embebidos en el silicio. Estos componentes extras hacen que las FPGA sean consideradas para el desarrollo de SoC y para la realización de sistemas DSP. Ya que el tipo y capacidad de los recursos que incluye cada fabricante y cada familia de dispositivos es muy variante, el proceso de selección de la FPGA para una aplicación en particular debe ser muy metódico.

3.4 Metodología de diseño.

La metodología de diseño inicia con la lógica para atender el proceso de descripción y representación de un producto final o un componente para un sistema mayor. En términos generales, existen dos métodos básicos, el diseño de abajo hacia arriba (*bottom-up*) y el diseño de arriba hacia abajo (*top-down*) [Pardo00]. Para un diseño hardware, una vez concebida la idea, el flujo de diseño se apoya en las herramientas de diseño asistido por computadora (CAD). En un nivel superior, que puede incluir también el diseño software, las herramientas se engloban en los sistemas de automatización del diseño electrónico (EDA), las cuales integran en el mismo marco de trabajo, tanto las herramientas de descripción como las de simulación, síntesis, realización y verificación

3.4.1 Metodologías de descripción.

En la metodología *botton-up*, la descripción del sistema inicia con los componentes más pequeños, para posteriormente agruparlos en diferentes módulos y estos en otros módulos hasta llegar a la representación del sistema completo. El proceso no implica una estructuración jerárquica de los elementos del sistema, y esta solo se puede definir al termino de la descripción del módulo superior. La descripción parte de los componentes de más bajo nivel, que representan unidades funcionales con significado propio dentro del diseño (primitivas). En un sistema electrónico, las primitivas pueden ser *chips*, transistores, resistencias y condensadores, entre otros elementos [Pardo00].

La metodología *top-down* consiste en capturar una idea en un alto nivel de abstracción y realizarla, dividiéndola en módulos y sub-módulos cada vez con mayor grado de detalle y con una funcionalidad determinada, hasta llegar a las primitivas del sistema. Este es el principio de que un problema muy complejo puede ser dividido en varios sub-problemas y estos en otros problemas mucho más sencillos de resolver [Pardo00]. Los sistemas EDA han evolucionado hacia la metodología *top-down*, ya que presenta las ventajas de incrementar la productividad de los diseñadores, aumentar la reutilización de los módulos intermedios del diseño y detectar los errores en etapas tempranas del ciclo de desarrollo.

3.4.2 Flujo de diseño FPGA.

La disponibilidad de las herramientas EDA han facilitado el desarrollo de sistemas digitales con lógica programable. En el caso de las FPGA, se tiene un flujo de

diseño que incluye las etapas de especificación, verificación, realización y depuración, en un proceso totalmente automatizado (Figura 3.5) [XilinxDK03].

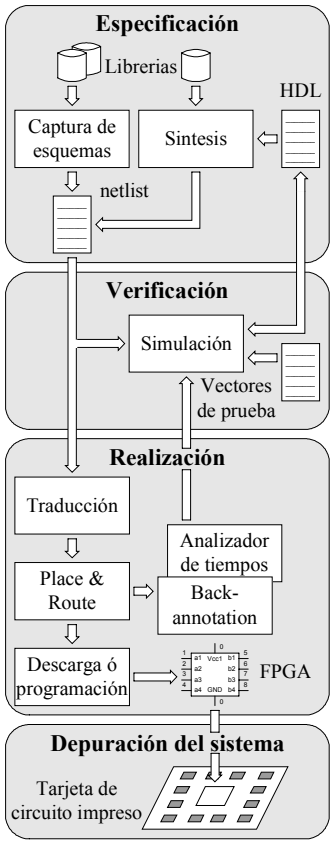


Figura 3.5. Flujo de diseño de los dispositivos FPGA.

Anteriormente, los diseños se especificaban completamente por medio de esquemas, pero dada la complejidad de los sistemas actuales, se prefiere utilizar lenguajes de descripción de hardware (HDL) como el VHDL y el Verilog, dejando los esquemas para la representación de los módulos de jerarquías superiores. Los HDLs permiten describir en forma de texto la función o comportamiento del circuito, en vez de hacerlo gráficamente y a bajo nivel.

La descripción HDL es procesada por la herramienta de síntesis para obtener la representación del comportamiento al nivel de componentes electrónicos. La síntesis es el proceso de adaptación de la lógica de diseño a los recursos lógicos disponibles en el *chip*, partiendo de la especificación de entrada con alto grado de abstracción y de las restricciones de área y tiempo definidas por el diseñador y llegando a una descripción menos abstracta, mas enfocada al dispositivo. En el proceso de síntesis se comprueba la sintaxis del código y se analiza la jerarquía del diseño para asegurar su optimización para la arquitectura seleccionada.

El proceso básico de verificación consiste en el uso de un simulador, el cual es un programa de software que auxilia al diseñador a comprobar el cumplimiento de las especificaciones del diseño. El simulador recibe un *nestlist* (archivo de texto equivalente al circuito) y un banco de vectores de prueba y genera las salidas que el diseño produce bajo estos patrones de estímulo. En este punto del flujo de diseño, solo se puede realizar

una simulación funcional, en la cual se verifica la operación correcta de la descripción HDL. Si los resultados de salida no son correctos, se debe modificar la descripción del diseño y repetir el ciclo hasta que la funcionalidad sea satisfactoria.

La etapa de realización tiene como propósito obtener un dispositivo FPGA programado con la operación lógica que describe el *netlist*, por medio de la secuencia de procesos de traducción (*translate*), mapeo *(map*), emplazamiento (*place*), rutado (*route*) y programación (*programming*). En la traducción se integran los programas para importar el *netlist* y fusionarlo con las restricciones del diseño definidas por el usuario, para obtener un archivo que describe el diseño lógico a partir de primitivas. En el mapeo, el diseño se ajusta a los recursos disponibles del dispositivo meta, relacionando los elementos lógicos del diseño con los recursos físicos del *chip*. El emplazamiento es el proceso de asignar los módulos o bloques lógicos del diseño a elementos específicos de la FPGA. En el rutado se interconectan los elementos lógicos seleccionados para que cumplan con la función lógica del diseño. La programación consiste en descargar a la FPGA la información de configuración que define la funcionalidad diseñada.

En este punto del ciclo el dispositivo puede estar ya trabajando, pero todavía es necesario depurar su funcionamiento dentro del sistema que lo contiene y bajo las condiciones del entorno donde trabajará. Este es el momento de resolver situaciones como funcionalidades no consideradas, operaciones incorrectas o requerimientos de E/S fuera de norma. La ventaja de la tecnología FPGA es que se puede regresar a las etapas iniciales del ciclo de desarrollo para resolver este tipo de anomalías, sin perjudicar significativamente los planes de tiempos y costes del proyecto.

3.5 Lenguaje de descripción de hardware VHDL.

Uno de los lenguajes de descripción de hardware más populares para el modelado, simulación y síntesis de circuitos y sistemas es el VHDL (*Very High Speed Integrated Circuit Hardware Description Language*), gracias a sus abundantes y flexibles elementos sintácticos y a su estructura que permite varios niveles de abstracción, incluyendo diseños específicos a la arquitectura. El VHDL puede ser utilizado para el diseño de sistemas utilizando las librerías estándar, independientes de algún fabricante o dispositivo, con la ventaja de que los diseños pueden ser reutilizados en otros diseños o por otros diseñadores, pero también existe la opción de familiarizarse con la arquitectura de un dispositivo específico y codificar el diseño para esa arquitectura, utilizando librerías del fabricante. En ambos casos la estructura del diseño es el resultado directo de la estructura del código VHDL, por lo que se debe poner mucha atención a las técnicas de codificación, para obtener la solución que ofrezca el comportamiento deseado al menor coste [Actel00].

3.5.1 Niveles de abstracción.

La funcionalidad de un circuito digital puede ser representada en diferentes niveles de abstracción, por lo que los HDL como el VHDL necesitan soportar una gama amplia de niveles, para ofrecer al usuario todas las posibilidades de diseño (Figura 3.6) [Maxfield04][Pardo00].

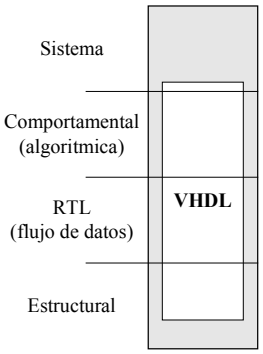


Figura 3.6. Niveles de abstracción del VHDL.

El más bajo nivel de abstracción del VHDL es el nivel de puertas, donde se describe el circuito a partir de un *netlist* de puertas y bloques lógicos. Este nivel de abstracción, en donde se especifican los componentes que incluye el circuito, su interconexión y los nombres que reciben los pines de E/S se conoce como descripción estructural. Aun cuando la mayoría de las descripciones son de un nivel superior, la descripción estructural permite incluir en el diseño elementos de librería y realizar diseños jerárquicos a partir de componentes.

El siguiente nivel de abstracción incluye la habilidad para describir funciones con ecuaciones booleanas y diseños con registros enlazados por lógica combinacional, por lo que se conoce como la descripción del nivel de transferencia de registros (RTL). Esta descripción también es conocida como flujo de datos, ya que todas las instrucciones son de asignación, por lo que los datos gobiernan el flujo de ejecución. La característica más importante de esta descripción es la interpretación o ejecución concurrente de las sentencias, para explotar el paralelismo de las arquitecturas hardware.

El más alto nivel de abstracción del VHDL es la descripción comportamental o algorítmica, la cual incluye tanto la ejecución concurrente como la programación serie, para describir un circuito en forma similar a los diseños software, dentro de bloques indicados con la palabra clave *process*. Un bloque *process* contiene instrucciones que se ejecutan en serie, como ecuaciones aritméticas, asignaciones, condiciones, bucles, etc. Si el programa tiene varios bloques *process*, cada uno de ellos equivale a una instrucción concurrente.

Aun cuando el VHDL soporta algunas construcciones de diseño al nivel de sistema, existen ciertos problemas asociados con su flujo de diseño en los ambientes donde el tiempo al mercado es una prioridad [Maxfield04]: la captura de la descripción y su verificación consumen demasiado tiempo; la evaluación de alternativas de realización es difícil; los cambios de especificaciones en el curso del proyecto son complicados; la descripción VHDL es especifica a la realización sobre FPGA o ASIC; el manejo de la descripción VHDL en ambientes de co-diseño hardware-software es problemático. Es por ello que la industria esta combinando la tecnología de descripción HDL con las metodologías y herramientas que cubren completamente el nivel de sistema, como el diseño ESL.

3.5.2 Características generales del VHDL.

El VHDL se utiliza tanto para la descripción de modelos de simulación como para la síntesis automática de circuitos. El modelo de simulación se utiliza para verificar la funcionalidad de un circuito mediante el uso de una herramienta de simulación. En simulación, el nivel de abstracción no es de gran importancia y las restricciones son mínimas, además de incluir una serie de elementos que solo tienen significado en este entorno, como retrasos, señalización de errores y bancos de prueba. La síntesis consiste en reducir el nivel de abstracción de la descripción VHDL hasta convertirla en una descripción puramente estructural, cuyos componentes son los elementos de la biblioteca indicada. La descripción VHDL para síntesis tiene varias restricciones, ya que el objetivo es obtener un circuito real que realice la función descrita en el código.

Como todo lenguaje, el VHDL tiene elementos sintácticos, tipos de datos y operadores. En los elementos sintácticos se ubican entre otros las palabras reservadas, los identificadores y los números. El lenguaje no tiene tipos de datos propios, pero sí los mecanismos para poder definir tipos escalares, compuestos y subtipos. Los operadores que se incluyen son de concatenación, aritméticos, de desplazamiento, relacionales y lógicos. En diseños complejos, el VHDL permite la estructuración de la descripción por medio de bloques, componentes y subprogramas (funciones y procedimientos) y, a un nivel más elevado, en estructuras por encima de la propia descripción, llamadas paquetes y estos a su vez en bibliotecas.

La descripción de un diseño hardware puede empezar declarando su símbolo: una caja negra con las entradas y salidas del circuito, que en la sintaxis de VHDL se denomina entidad (palabra reservada *entity*). La descripción del funcionamiento del circuito simbolizado por la entidad se realiza en la definición de arquitectura (palabra reservada *architecture*). El mecanismo para incorporar elementos de las bibliotecas y hacerlos visibles al diseño, consiste en la inclusión de la cláusula *library* y la lista de bibliotecas que se desea sean visibles, usando la palabra reservada *use* (Figura 3.7).

```
-- Declaración de librerias.
LIBRARY nombre;
USE nombre_libreria.visibilidad;
   :                :
USE nombre_librería.visibilidad;

-- Declaración de entidad.
ENTITY nombre IS
    [GENERIC(lista de parámetros);]
    [PORT(lista de puertos);]
    [declaraciones]
[BEGIN
    sentencias]
END  [ENTITY] [nombre];

-- Declaración de arquitectura.
ARCHITECTURE nombre OF nombre_entidad IS
    [declaraciones]
BEGIN
    [sentencias concurrentes]
END [ARCHITECTURE] [nombre]
```

Figura 3.7. Declaración general de librerías, entidad y arquitectura de un programa en VHDL.

La forma general de declaración de entidad incluye las palabras reservadas *generic* y *port*, que aun cuando son opcionales, tienen gran importancia. La instrucción *generic* se utiliza para definir y declarar propiedades o constantes del módulo declarado en la entidad. Con la palabra *port*, se definen las entradas y salidas del módulo.

En la sintaxis general de la arquitectura, existe una parte declarativa opcional donde se pueden definir subprogramas (funciones, procedimientos, etc.) y declarar tipos de datos, constantes, señales, componentes, etc. Después del *begin* se encuentran las sentencias concurrentes que definen el comportamiento, funcionalidad y estructura del circuito.

3.6 Herramientas de diseño para FPGA.

En el mercado se encuentran gran cantidad de herramientas EDA para el diseño con FPGA y VHDL. Algunas de ellas son de uso publico (*open-source*), otras cubren casi todos los dispositivos del mercado, como las de Mentor Graphics, Exemplar, Synopsys y Synplicity. Las más son especificas de las compañías de dispositivos, como las de Actel, Altera, Cypress y Xilinx. La mayoría de las herramientas funcionan sobre PC y algunas están disponibles para *workstation*.

Ya que las herramientas específicas de cada compañía están optimizadas para las arquitecturas de sus dispositivos, además de cubrir toda la gama de posibilidades de diseño que sus FPGA ofrecen, desde la síntesis hasta el desarrollo de sistemas embebidos, esta sección se orienta a las herramientas de la empresa Xilinx, al ser esta el fabricante de los dispositivos utilizados en la realización de las arquitecturas hardware que se proponen en este trabajo.

3.6.1 Herramientas de síntesis.

La síntesis es uno de los pasos más esenciales en la metodología de diseño con FPGA, por lo que se necesitan utilizar las técnicas del estado del arte para generar la mejor representación lógica para el dispositivo seleccionado, a partir de la definición conceptual del diseño.

3.6.1.1 Síntesis HDL y síntesis con enfoque físico.

Las herramientas de síntesis que salieron al mercado a mediados de los 1980s, referidas como tecnología de síntesis HDL, toman una descripción RTL de un diseño ASIC junto con un conjunto de restricciones de tiempo y generan un *netlist* al nivel de puertas, en un proceso de minimización y optimización. A mediados de los 1990s, estas herramientas fueron extendidas para incluir las arquitecturas de las FPGA, para producir un *netlist* al nivel de LUTs y bloques lógicos configurables (LUT/CLB).

En esa época, los *chips* eran diseñados con tecnología del estado sólido de baja resolución, por lo que el retardo de rutado tenia poco peso en el retardo de red. Por lo mismo, las herramientas de síntesis utilizaban modelos simples para evaluar los efectos de los retardos de rutado. En los *chips* actuales con tecnología de sub-micrones, donde el retardo de rutado representa hasta el 80% del retardo de red, estas herramientas no estimarian adecuadamente los tiempos del diseño.

A partir de 1996 se consideraron herramientas de síntesis con enfoque físico en el flujo de diseño ASIC. El flujo de diseño de las FPGA también las adoptó a principios del 2000. Estas herramientas de síntesis utilizan información de emplazamiento de los elementos lógicos del diseño asociada al dispositivo objetivo, para estimar los retardos de rutado lo más pronto posible en el proceso de síntesis. Actualmente, las herramientas EDA inician el proceso de síntesis con un paso de síntesis HDL (para obtener la información de emplazamiento de los elementos lógicos) y continúan con un paso de síntesis con enfoque físico (Figura 3.8) [Maxfield04].

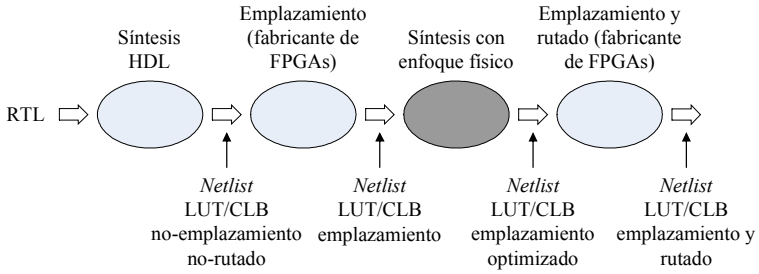


Figura 3.8. Proceso moderno de síntesis para el flujo de diseño de las FPGA.

3.6.1.2 Tecnología de síntesis de Xilinx.

La empresa Xilinx ofrece el software ISE (ambiente de software integrado), que conjunta las herramientas de todas las etapas del flujo de diseño (especificación, realización, verificación y depuración del sistema), para las familias de FPGA y CPLD que tiene en el mercado (Virtex-5, Virtex-4, Virtex-II PRO, Virtex-II, Virtex, Spartan-III, Spartan-II, CoolRunner-II, CoolRunner-XPLA3, XC9500, etc.). Además de incluir una tecnología de síntesis propietaria, el ISE proporciona una integración directa con los motores de síntesis de las compañías lideres (Leonardo Spectrum, Synplify Pro, Precision RTL Synthesis, etc.), facilitando el uso de múltiples herramientas para obtener el mejor resultado.

La tecnología de síntesis de Xilinx (XST), sintetiza descripciones en VHDL, Verilog o una mezcla de ambos lenguajes, para crear un archivo *netlist* específico de Xilinx (NGC), el cual contienen los datos del diseño lógico y las restricciones que procesa el paso de traducción de la etapa de realización [XilinxISE8.2i]. El XST puede recibir núcleos IP en forma de archivos *netlist* en formato NGC o EDIF (*Electronic Data Interchange Format*) y a la vez genera un reporte de síntesis (archivo LOG) y los esquemas RTL (archivo NGR) y tecnológico del diseño (incluido en el archivo NGC) (Figura 3.9).

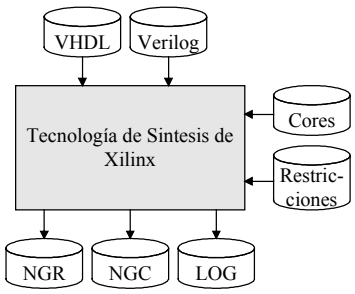


Figura 3.9. Archivos de entrada y salida de la herramienta de síntesis XST.

En el paso de análisis HDL del flujo de diseño XST (Figura 3.10), se verifica la escritura correcta del código HDL, reportándose cualquier error de sintaxis. En la síntesis, el XST analiza el código HDL e intenta inferir macros, tales como multiplexores, RAMs, sumadores y restadores, para los cuales se crean realizaciones tecnológicamente eficientes. Para reducir la cantidad de macros inferidos, el XST ejecuta un reconocimiento de recursos compartidos, lo que puede producir una reducción del área así como un incremento en la frecuencia del reloj.

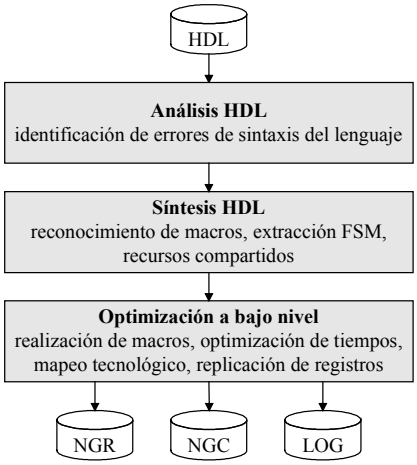


Figura 3.10. Flujo de diseño del XST.

El reconocimiento de las máquinas de estado finito (FSM) forma parte del proceso de síntesis. El XST reconoce las FSM independientemente de su estilo de modelación. Para generar la realización más eficiente, el XST hace referencia a las propiedades del proceso de síntesis definidas por el usuario, como la meta y el esfuerzo de optimización, lo que define el algoritmo de codificación FSM más adecuado.

Las restricciones controlan el proceso de síntesis HDL, las cuales pueden ser introducidas en el archivo HDL fuente (como atributos del VHDL o meta-comentarios del Verilog), por medio del archivo de restricciones de Xilinx (XCF) o por las propiedades del proceso de síntesis.

En la optimización a bajo nivel, el XST transforma los macros inferidos y la lógica general en una realización específica a la tecnología. El proceso puede ser controlado por restricciones de tiempo, como periodo y desplazamiento. Debido a la complejidad del mapeo, no todas las características de la FPGA son utilizadas [XilinxISE8.2i].

3.6.2 Herramientas de verificación.

La verificación es la operación con la que se comprueba si la realización de un producto se ajusta a las normas o especificaciones técnicas establecidas, o si posee las condiciones de funcionamiento exigidas. En el ambiente de los diseños electrónicos, la verificación se realiza en diferentes fases del flujo de diseño, para comprobar el cumplimiento de la funcionalidad y rendimiento. Pero mientras las tareas de diseño crecen linealmente con la complejidad, el proceso de verificación crece exponencialmente, consumiendo cada vez más tiempo y recursos.

Los *chips* complejos emplean bloques funcionales, procesadores, periféricos, protocolos e interfases múltiples, etc., por lo que la verificación debe probar cada elemento y su interacción con los demás, en todas las combinaciones y modos de operación posibles. La verificación incluye la realización de un ambiente de verificación, crear un *testbench*, realizar la simulación lógica, analizar los resultados para detectar y aislar los problemas, etc.

La verificación es un parte integral de todo proyecto de diseño con FPGA. Gran cantidad de modelos de verificación tradicionales ya no son apropiados para los nuevos diseños con dispositivos de millones de puertas y los métodos modernos deben ser estudiados para determinar su efecto en el tiempo al mercado del producto [Tessier02].

3.6.2.1 Métodos de verificación.

La verificación puede consumir el 70% o más del esfuerzo total de desarrollo de un sistema digital, desde el concepto inicial hasta la realización final, por lo que es necesario optimizar el proceso utilizando las mejores herramientas y metodologías, entre las que se encuentran la simulación, la verificación IP y la verificación formal.

La simulación lógica es la herramienta tradicional de verificación. Entre los algoritmos más populares se encuentran la simulación guiada por eventos y la simulación basada en ciclos. En la simulación guiada por eventos, el simulador guarda una lista de todos los eventos que se generan y los ordena según el momento que se tengan que procesar. Al producirse cada evento, se pueden generar nuevos eventos que se colocaran en la lista según el retraso asociado. En cada evento se ejecutan una o varias sentencias concurrentes, incluidos los procesos. La simulación termina cuando no quedan mas eventos en la lista o cuando se llega a un tiempo especificado de antemano. Para acelerar la simulación, algunas herramientas utilizan la técnica de avance de tiempo por incremento fijo [Pardo00].

La simulación basada en ciclos es particularmente adecuada para diseños *pipeline*, en los que se tienen islas de lógica combinacional entre bloques de registros. En este caso, el simulador analiza la información de tiempo asociada con las puertas que forman la lógica combinacional y convierte esta lógica en una serie de operaciones booleanas que son realizadas directamente por las instrucciones lógicas del GPP [Maxfield04].

La ventaja de la simulación guiada por eventos es que los simuladores basados en esta técnica pueden ser usados para representar casi cualquier forma de diseño, incluyendo circuitos síncronos, asíncronos, lazos de retroalimentación combinacional, etc. Estos simuladores también ofrecen una buena visibilidad del diseño para propósitos de depuración y pueden evaluar los efectos de pulsos cortos relacionados al retardo y detectar malfuncionamientos que son muy difíciles de encontrar con otras técnicas. Su desventaja es que son computacionalmente intensivos y por lo tanto lentos.

El simulador basado en ciclos ofrece ventajas significativas de tiempo de simulación sobre el simulador activado por eventos. Sus desventajas son que solo trabaja con valores lógicos 1 y 0 (no maneja los niveles de alta impedancia y *don´t care*) y no tiene capacidad de resolución en situaciones de contención. Otra desventaja es que no puede representar lógica asíncrona o lazos combinacionales de retroalimentación.

Los simuladores modernos han sido mejorados para tener ambas capacidades, por lo que pueden verificar alguna parte del circuito usando un enfoque de activación por eventos y otra parte usando técnicas basadas en ciclos.

Normalmente en la simulación se utiliza un *testbench*, el cual describe las señales precisas que son aplicadas en los pines de entrada y evalúa los valores de salida. Sin embargo, los diseños actuales pueden ser tan complejos que es difícil crear manualmente los vectores de prueba, por lo que han aparecido en el mercado ambientes y lenguajes de verificación sofisticados, algunos conocidos como lenguajes de verificación de hardware (HVL), los cuales declaran rangos validos, secuencias de valores de entradas y estrategias de verificación de alto nivel.

La técnica de verificación IP forma el sistema bajo prueba utilizando modelos funcionales de bus (BFM), para representar módulos complejos como procesadores y agentes de E/S. Un BFM no replica la funcionalidad completa del dispositivo que representa, pero emula la forma que trabaja el dispositivo y el nivel de interfase de bus, generando y aceptando transacciones [Maxfield04].

La verificación formal es un término amplio que comprende técnicas para explorar el estado espacial de un sistema y probar cuando o no ciertas propiedades, típicamente especificadas en la forma de aserciones, son verdaderas. También comprende el uso de lógica para probar, de una manera similar a una prueba matemática, que una realización satisface una especificación asociada.

Existen dos tipos principales de verificación formal: la comprobación de equivalencias y la comprobación del modelo. La comprobación de equivalencias es una herramienta que verifica la equivalencia funcional de dos diseños que están en el mismo o en diferente nivel de abstracción (por ejemplo, RTL a RTL, RTL a puertas y puertas a puertas), para determinar si tienen la misma funcionalidad entrada/salida. La comprobación del modelo verifica que la realización satisface las propiedades del diseño. Esta es utilizada al inicio de la fase de creación del diseño para descubrir defectos funcionales. Aun cuando las grandes compañías de computadoras y *chips* han estado desarrollando y usando herramientas formales desde mediados de 1980s, el concepto es relativamente nuevo en el ambiente de las FPGA.

3.6.2.2 Tecnologías de verificación de Xilinx.

El software de diseño de Xilinx ISE [XilinxISE8.2i] incluye todos los elementos para verificar y depurar un diseño en cualquier punto del flujo del desarrollo, desde simuladores hasta herramientas de verificación formal (comprobación de equivalencias). La verificación que soporta la divide en varios niveles: funcional, de tiempo, avanzada y al nivel de tarjeta.

La verificación funcional verifica la sintaxis y funcionalidad del diseño utilizando análisis HDL, simulación y elementos de generación de *testbench*. Las herramientas de software que se integran, algunas propias de Xilinx y otras de sus socios tecnológicos, son ModelSim, simulador ISE, *Design Rule Check* (DRC), HDL *Advisor*, HDL *Bencher* y LEDA PRO. El ModelSim es un ambiente completo de simulación para verificar los modelos funcionales y de tiempo de un diseño. El DRC ofrece una serie de comprobaciones predefinidas o definidas por el usuario para

encontrar los errores físicos del diseño. El HDL *Advisor* ofrece sugerencias interesantes sobre como codificar el diseño para reducir su tamaño y satisfacer los requerimientos de tiempo. El HDL *Bencher* realiza una verificación automática de diseños FPGA basados en HDL, para acelerar el tiempo al mercado e incrementar la productividad. El LEDA PRO es una herramienta de Synopsys que ofrece la verificación del código HDL para reducir los esfuerzos de depuración y mejorar la calidad del diseño.

En la verificación de tiempo se comprueban las especificaciones de retardo utilizando las herramientas de cálculo de retardo y tiempo estático del ISE y de los socios de Xilinx, como *Delay Calculator*, *Prime Time*, *Time Analyzer* y TRACE. El *Delay Calculator* calcula y exhibe los retardos asociados con la carga y los pines de activación en una red o ruta dada. El *Prime Time* es una herramienta de *Synopsys* que utiliza el ISE para identificar y resolver violaciones de tiempo del diseño. El *Time Analyzer* es una herramienta con interfase gráfica del usuario que realiza análisis estáticos de tiempo de un diseño FPGA o CPLD. El TRACE (*Timing Reporter and Circuit Evaluator*) realiza un análisis estático de tiempos del diseño en función de las restricciones de tiempo de entrada.

La verificación avanzada va mas allá de las verificaciones tradicionales, al usar herramientas de análisis térmicos, depuración lógica en tiempo real, análisis de bus y verificación formal, como *ChipScope Pro*, *Power Analysis*, *IBISWriter*, modelos *Spice*, y *Tau*. El *ChipScope Pro* es una herramienta de verificación y depuración en tiempo real para FPGA de Xilinx, que puede realizar depuraciones en el *chip* a la velocidad del sistema operativo. La herramienta *Power Analysis* ofrece un estimador de energía previo a la realización para dispositivos lógicos programables. El *IBISWriter* simplifica la exportación del diseño en las herramientas de análisis de integridad de señal. Los modelos *Spice* ofrecen un ambiente de simulación con un circuito de alta exactitud, combinando los modelos de dispositivos validados más exactos. El *Tau* es una herramienta de *Mentor Graphic* que detecta el tiempo del peor caso del diseño al nivel de tarjeta.

La verificación al nivel de tarjeta asegura que el diseño se comporte como se proyectó una vez integrado en el PCB, utilizando modelación de E/S, análisis estático de tiempos y tecnologías de depuración de hardware, con herramientas del ISE y de sus los socios de Xilinx, como las ya mencionadas *ChipScope Pro* y *Power Analysis* y la FPGA *Editor Probe*. Esta última ejecuta una depuración física extremadamente eficiente apoyada en el acceso al estado interno del dispositivo.

3.6.3 Herramienta de diseño embebido EDK.

Las plataformas de sistemas programables, que incluyen buses, memoria, aceleradores hardware y núcleos de procesadores *hard* y *soft*, para realizar sistemas embebidos en la forma de SoC sobre FPGA, son alternativas que no implican los retardos de tiempo y los riesgos y restricciones en recursos y costes propios de los diseños con ASIC. Los SoC realizados en plataformas FPGA proporcionan una gran oportunidad para desarrollar sistemas óptimos y de alto rendimiento. Sin embargo, para realizar estos sistemas, se necesita una cadena completa de herramientas que partan desde el concepto y lleguen hasta la depuración del sistema en tarjeta, en un ambiente cooperativo hardware/software.

Para este fin, la empresa Xilinx ofrece el kit de desarrollo de sistemas embebidos (EDK), como parte de sus productos de línea, para facilitar y modernizar el proceso de creación de sistemas SoC sobre FPGA. El kit integra todas las herramientas para el desarrollo y depuración de sistemas hardware/software, con las que se pueden realizar las siguientes funciones:

* Definir plataformas hardware para un sistema programable, utilizando núcleos de procesadores (PowerPC o MicroBlaze), IPs parametrizables y buses de interconexión.

* Desarrollar plataformas software para adaptarlas al hardware definido.

* Utilizar herramientas de verificación e interfases a los simuladores HDL soportados por Xilinx.

El EDK es un ambiente de desarrollo integrado, donde se conjuntan los siguientes elementos, que facilitan el diseño al concentrar toda la potencia ofrecida por una plataforma programable:

a) Grupo de herramientas Xilinx *Platform Studio* (XPS).

   Soporte de herramientas gráficas y en línea de comandos para desarrollar y depurar plataformas hardware/software para aplicaciones embebidas; ayudas de diseño inteligentes para configurar rápidamente arquitectura, buses y periféricos del sistema embebido.

b) Herramientas de desarrollo de software para MicroBlaze y PowerPC.

   * Compilador y depurador GNU C/C++ .

   * Servidor de depuración de microprocesadores Xilinx (XMD).

   * Utilería Data2MEM, para carga y actualización del flujo de bits.

   * Ayuda de configuración del sistema *Base System Builder* (BSB).

   * Kit de desarrollo y depuración de software *Platform Studio* (SDK).

c) Paquetes de soporte de tarjeta (BSPs).

   * BSP independientes, para sistemas que no son de tiempo real.

   * Wind River VxWorks, para PowerPC en plataforma FPGA.

   * Monta Vista Linux, para PowerPC en plataforma FPGA.

   * Soporte para sistemas de MicroKernel Xilinx (XMK)

d) Procesadores IP.

   * Catalogo de IPs pre-verificados.

* Núcleo del procesador *soft* MicroBlaze.

Las herramientas del EDK permiten configurar automáticamente la plataforma de hardware y crear un diseño software que incluye las librerías apropiadas y los controladores de dispositivos. Este ambiente productivo ahorra tiempo al utilizar una plataforma programable flexible, donde se pueden optimizar los diseños para obtener los mejores resultados precio-rendimiento. Esto significa que el tiempo del desarrollo se utilizara en crear productos con características únicas y de alto valor, que lo diferenciaran de sus competidores. La programación, reprogramación y actualización del sistema en campo significa que el producto llegara al mercado más rápido y podrá tener un mayor tiempo de vida, regresando un beneficio mayor.

El flujo de diseño EDK inicia con el desarrollo hardware (Figura 3.11), donde se recomienda utilizar la herramienta BSB para configurar la plataforma programable [XilinxEDK8.2i]. Si el diseño incluye aceleradores hardware, estos se pueden definir como periféricos y conectarlos al diseño junto con los componentes básicos. En este punto ya se tienen los elementos para iniciar el desarrollo de la plataforma software y las aplicaciones, con el uso de las herramientas XPS y SDK. Una vez que el hardware y software estén disponibles, ya depurados, se procede a configurar el dispositivo FPGA, descargando el flujo de bits que conjunta todos los componentes del diseño, para finalizar con la prueba y depuración de la tarjeta electrónica, en laboratorio y en campo.

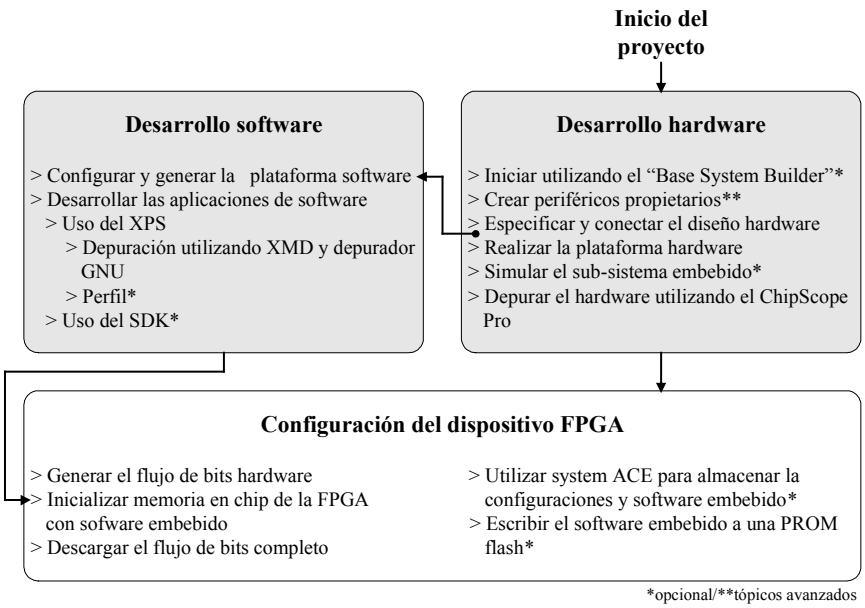


Figura 3.11. Flujo de diseño del EDK.

3.7 Diseño para comportamiento y reutilización.

Aun cuando las FPGA son muy flexibles para adaptarse a diversos estilos de diseño, tipos de codificación HDL y opciones de configuración de las herramientas, no todas las alternativas resultan en un diseño optimo en términos de área, comportamiento y consumo de energía. Es por ello que los fabricantes de FPGA y desarrolladores de herramientas presentan una serie de recomendaciones para que los diseñadores alcancen sus objetivos en pocas iteraciones. Las primeras recomendaciones son básicas para el

buen desarrollo del ciclo de diseño y se sugiere atenderlas antes de escribir las primeras líneas de código:

* Estudiar la guía del usuario y la hoja de datos de la FPGA, para entender y visualizar como aprovechar lo mejor posible su arquitectura y tomar las decisiones de codificación más adecuadas.

* Revisar los manuales del usuario, notas de aplicación y otro material relevante, para conocer las capacidades y limitaciones del software de diseño y determinar el escenario que más optimiza el proyecto.

En las siguientes secciones se listan otras recomendaciones al nivel de prácticas de codificación y herramientas. Su atención redundara en la obtención de un diseño óptimo y un tiempo de desarrollo mínimo.

3.7.1 Prácticas de codificación.

* Preferir *resets* síncronos y locales a *resets* asíncronos globales, para reducir el consumo de recursos lógicos y de rutado, facilitar la inferencia de recursos altamente eficientes y aumentar la confiabilidad del diseño [XilinxWP272].

* Segmentar el diseño, para mejorar el comportamiento de un diseño, al reestructurar rutas largas en varios niveles de lógica distribuidos en diversos ciclos de reloj (Figura 3.12) [XilinxReuse02] [Man02] [Frank02a].

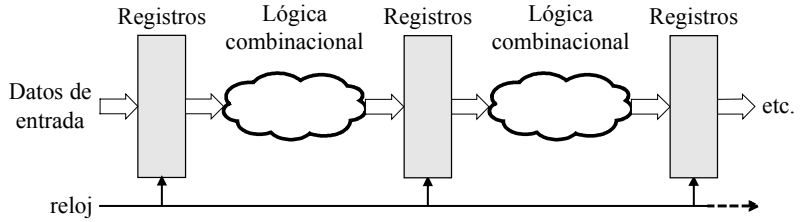


Figura 3.12. Segmentación (*pipeline*) del diseño.

* Balancear retardos de ruta, porque la máxima velocidad del reloj del sistema es función del mayor retardo. El retardo en exceso de una ruta significa que se han gastado recursos en otras rutas que corren a mayor velocidad y que no lo van a poder hacer [Aldec98].

* Duplicar registros, para reducir la carga excesiva (*fanout*) de las rutas críticas y mejorar el rendimiento en velocidad [XilinxReuse02]. Según [Actel00], se debe mantener el *fanout* de una red menor a 16.

* Definir límites con registros, registrando entradas y/o salidas de bloques lógicos, para que la temporización sea completamente local.

* Diseñar adecuadamente las máquinas FSM, optimizando la cantidad de lógica combinacional, la codificación de los estados, la carga de entrada de cada registro (*fanin*), el número de registros y el retardo de propagación entre registros [XilinxISE8.2i] [Pereira05] [Frank02b].

* Utilizar la estructura *case* para decodificaciones complejas e *if* para rutas de velocidad crítica, tomando en cuenta que la estructura *case* crea lógica balanceada y la *if* produce generalmente lógica para codificación de prioridad. En el caso de estructuras anidadas, si se desea mayor velocidad, sustituir los *if-else-if* por estructuras *case*.

* Aplicar la polaridad en las señales de control (reloj, habilitaciones, *reset*, etc.) que recomienda el fabricante, lo que normalmente disminuye el consumo de recursos y mejora el comportamiento y empaquetado de bloques lógicos [Philofsky06].

* Utilizar registros de salida, para mejorar las características de tiempo de las memorias RAM distribuida y bloques de RAM [Philofsky06].

* En los bloques de RAM de dos puertos, considerar los modos de operación que gobiernan la salida de la memoria, cuando se produce una operación de escritura. Los diferentes modos definen el comportamiento de la memoria y afectan su operación [Garrault05]. Un consejo es evitar el modo *read before write* para obtener el máximo rendimiento de los bloques de RAM [XilinxWP231].

* Utilizar los patrones (*templates*) del lenguaje que incluye el fabricante en sus herramientas de diseño.

* El mantener el código genérico y no restringirlo a una arquitectura en particular, auxilia en la reutilización y portabilidad del diseño [Pereira05].

* Ubicar todos los componentes de E/S (*buffers*, registros, circuitos de tasa de datos doble, elementos de retardo, etc.) en el nivel superior del diseño. Si no es así, asegurar que todos ellos estén contenidos dentro de una jerarquía única.

* Utilizar los recursos de E/S que el dispositivo permita, tales como registros de entradas y salidas, control variable de tiempos de *set-up* y *hold*, nivel de *slew-rate*, etc.

* Utilizar habilitaciones de reloj en vez de controlar el reloj.

* Evitar la presencia de candados (*latch*) no deseados, asignando valores a las salidas en todos los casos. Los *latch* consumen silicio, tienen un pobre rendimiento de tiempo y a menudo modifican el funcionamiento del circuito.

3.7.2 Optimización al nivel de herramientas.

3.7.2.1 Restricciones y propiedades de la síntesis.

* Definir todas las velocidades de reloj.

* Experimentar con el diseño sobre-restringido.

* Especificar excepciones de tiempo, tales como rutas falsas y multiciclos. Con esta información, la herramienta puede ignorar estas rutas y concentrarse en las rutas críticas reales [Pereira05].

* Tratar restricciones de rutado.

* Incluir CoreGen EDIF o modelos de tiempo para cajas negras. Si se utilizan cajas negras IP en el diseño, asegurar que se incluyan sus *netlist* EDIF y NGC y los modelos de tiempo, para que las herramientas puedan acceder a su descripción [Pereira05].

* Experimentar con las propiedades de síntesis en el ambiente ISE (Tabla 3.2).

Tabla 3.2. Valor de las propiedades del proceso de síntesis para la optimización del diseño.

	Propiedad	Valor
Opciones de síntesis	Optimization Effort	Alto
	Synthesis Constraint File	
	Create a XCF File (UCF Syntax)	Si
	NET clock PERIOD = 5 ns	
	Read Cores	Si
Opciones de HDL	Resource Sharing	Experimentar
Opciones especificas de Xilinx	Register Balancing	Si
	Pack I/O Registers into IOBs	Experimentar

3.7.2.2 Restricciones y propiedades de la realización.

* Primero, experimentar sin restricciones.

* Aplicar restricciones, pero no sobre-restringir más del 15% [Pereira05].

* Relajar relojes no críticos.

* Experimentar con las propiedades del proceso de realización en el ambiente ISE (Tabla 3.3) [Bixler05].

Tabla 3.3. Valor de las propiedades del proceso de realización para la optimización del diseño.

	Propiedad	Valor
Propiedades del mapeo	Perform Timing Driven Packing and Placement	Si
	Map Effort level	Alto
	Logic Optimization	Si
	Global Optimization	Si
	Retiming	Si
Propiedades de la ubicación y rutado	Place and Route Effect Level	Alto
	Use Timing Constraints	Si

3.7.2.3 Consideraciones generales en el uso de las herramientas.

Si el diseño tiene una cantidad excesiva de niveles lógicos [Fernandez06]:

* Tratar la opción de síntesis lógica en el mapeo.

* Regresar al proceso de síntesis y verificar que las rutas críticas reportadas en la realización se igualen con las reportadas en la síntesis.

* Revisar la inferencia en la síntesis del código HDL.

Si se tienen pocos niveles de lógica, pero ciertas rutas de datos no satisfacen los requerimientos de tiempo:

* Evaluar el *fanout* de las rutas con grandes retardos.

* Considerar el uso de bloques IP *hard* como bloques de RAM y módulos DSP48.

* Si las rutas críticas contienen bloques IP *hard*, verificar que el diseño tome total ventaja de los registros embebidos. También entender cuando utilizar estos bloques *hard* o usar lógica general.

* Analizar el *skew* del reloj.

3.7.3 Reutilización.

Una de los conceptos que más se ha extendiendo para agilizar el desarrollo de un producto es la filosofía de la reutilización. Su metodología, que es similar en ASIC y FPGA, agrega nuevos niveles de libertad a los diseñadores, permitiendo una alta productividad que reduce costes y tiempo al mercado, ya que su aplicación hace innecesario cubrir toda la curva de aprendizaje para la creación de cada producto, al reutilizar en los nuevos diseños, los bloques funcionales existentes (IPs). La necesidad de diseño para la reutilización tiene su historia. A ninguna compañía le gusta invertir en diseños que se usan una sola vez. En el presente, miles de diseñadores están creando IPs que soportan una amplia gama de tecnologías [XilinxReuse00] [XilinxReuse02].

La reutilización no sucede por si misma, aunque los diseñadores tengan las mejores intenciones y sea justificable económicamente, ya que siempre hay obstáculos que franquear antes de que esta estrategia se convierta en estilo de diseño. El diseño para la reutilización es más una función de la infraestructura administrativa que de los diseñadores mismos.

Tal como se observa en las reglas y guías de reutilización para ambientes de diseño con FPGA [Xilinxreuse00], la reutilización misma es un estilo de codificación, por lo que las estrategias de diseño y reutilización comunes ofrecen la flexibilidad para seleccionar el mejor método para diseñar un sistema, sin producir una sobrecarga de trabajo en el equipo de diseño.

3.8 Procesadores embebidos en las FPGA de Xilinx.

Los SoC realizados sobre FPGA Xilinx pueden incluir dos tipos de microprocesadores embebidos, el MicroBlaze y el PowerPC 405, para crear sistemas que conjuntan la versatilidad de las funciones realizadas por software, con el alto rendimiento de las tareas hardware. El MicroBlaze es un procesador *soft*, diseñado para realizarse en la lógica de propósito general de las FPGA Xilinx. El PowerPC 405 es un procesador *hard* que incluyen algunas FPGA Xilinx de alto nivel, como un bloque de silicio dedicado y que no consume recursos lógicos del dispositivo.

3.8.1 Procesador MicroBlaze.

El MicroBlaze es un procesador *soft* de 32 bits con arquitectura Harvard RISC, optimizado para las familias de FPGA Xilinx e incluido como un núcleo IP en el software EDK en forma de un *netlist* parametrizable [XilinxUG081]. Su estructura básica consiste de 32 registros de propósito general, una unidad aritmética lógica (ALU), una unidad de desplazamiento, registros de propósito especial, decodificador de instrucciones, interfases de bus y dos niveles de interrupción (Figura 3.13).

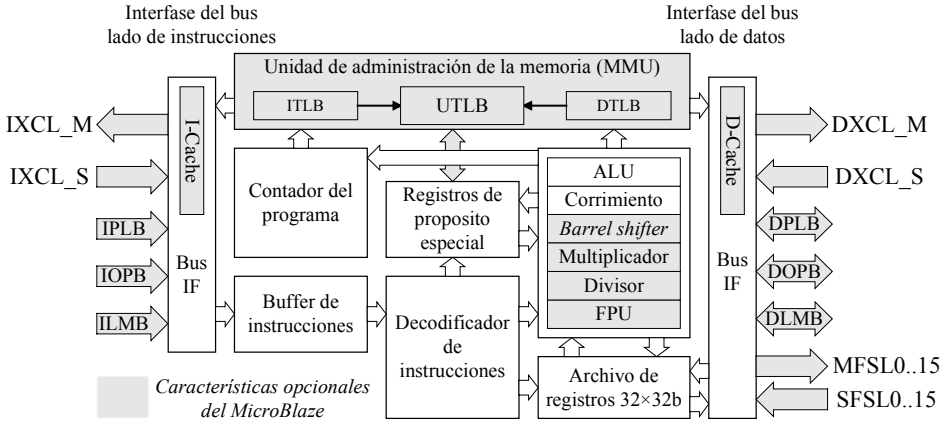


Figura 3.13. Diagrama a bloques del MicroBlaze.

El diseño elemental del MicroBlaze puede ser configurado por el usuario para habilitar, definir el tamaño o seleccionar algunas características y componentes opcionales, como unidades de administración y protección de la memoria, unidad de punto flotante, divisores, multiplicadores, *barrel shifter*, *caches*, manejo de excepciones, lógica de depuración, etc., con los cuales se logra una flexibilidad que permite balancear el comportamiento requerido con el coste en área.

3.8.1.1 Características generales.

Las características generales del MicroBlaze son:

* Arquitectura completamente ortogonal.

* Representación de los datos con formato Big-Endian bit-inverso, con soporte de datos tipo palabra (MSBit 0: LSBit 31), media palabra (0:15) y byte (0:7).

* Palabras de instrucciones de 32 bits (tipo A o tipo B), con tres operandos y dos modos de direccionamiento. Las instrucciones tipo A tienen hasta dos operandos fuente y un operando destino. Las instrucciones tipo B tienen un registro fuente y un operando inmediato de 16 bits (el cual puede ser extendido a 32 bits al ser precedido por la instrucción *imm*). Las instrucciones tipo B tienen un solo operando destino. Las instrucciones se clasifican en aritméticas, lógicas, de bifurcación, cargar/almacenar y especiales, algunas de ellas privilegiadas.

* Treinta y dos registros de 32 bits de propósito general, numerados de R0 a R31, inicializados a 0x00000000 en la descarga del flujo de bits de configuración (los registros no se inicializan por las entradas de *reset*).

* Hasta 18 registros de 32 bits de propósito especial, dependiendo de la configuración del procesador (contador del programa, registro de estado de la máquina, registro de excepción de direcciones, registro de estado de excepciones, registro del objetivo de la bifurcación, etc.)

* Arquitectura de ejecución de instrucciones tipo *pipeline*, con opción de tres estados para minimizar el coste del hardware y de cinco estados para maximizar el rendimiento (Figura 3.14).

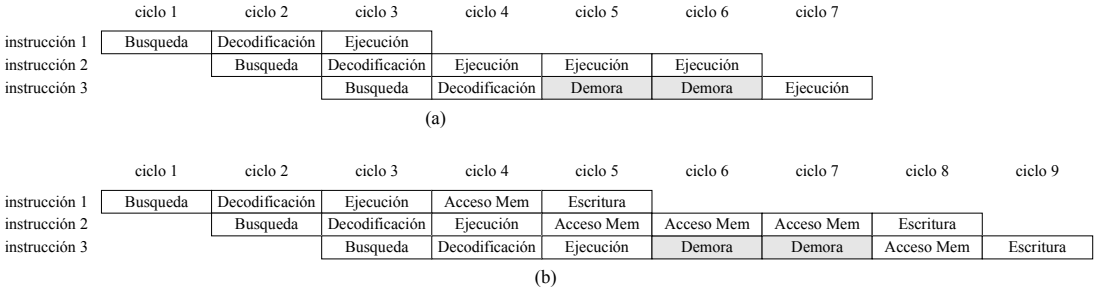


Figura 3.14. Ejecución de las instrucciones con a) 3 estados *pipeline*, b) 5 estados.

* Ejecución de programas utilizando direccionamiento efectivo, para acceder a un espacio de direccionamiento plano de 4 Gb. El procesador puede interpretar este espacio de direcciones de dos maneras, dependiendo del modo de traslación:

  + En modo real, utilizando direcciones efectivas para acceder directamente a memoria física.

  + En modo virtual, las direcciones efectivas son trasladadas a direcciones físicas por el hardware de administración de la memoria virtual del procesador.

* Soporte de *reset*, excepciones de hardware, rompimientos no-enmascarables, rompimientos, interrupciones y vectores del usuario (en este orden, de mas a menos prioridad).

* Posibilidad de usar un *cache* de instrucciones, para mejorar el rendimiento cuando se ejecuta código que reside fuera del rango de direcciones del bus LMB.

* Uso opcional de un *cache* de datos, para mejorar el rendimiento cuando se leen datos que residen fuera del rango de direcciones del bus LMB.

* Unidad de punto flotante (FPU) opcional, basada en el estándar IEEE 754.

* Capacidad de desarrollo de software en lenguaje ensamblador, por medio de la interfase de aplicación binaria (ABI).

3.8.1.2 Buses, puertos e interfases.

El MicroBlaze está organizado como una arquitectura Harvard, con buses separados para datos e instrucciones. Las interfases a memoria que soporta son el bus de memoria local (LMB), el bus local del procesador (PLB) o el bus de periféricos en *chip*

(OPB) y el Xilinx CacheLink (XCL). El procesador también incluye puertos de enlace simple de alta velocidad (FSL), cada uno con una interfase maestra (MFSL) y una esclava (SFSL). Otros elementos que contiene son la interfase de depuración y la interfase de trazo, para el análisis de comportamiento.

El LMB es un bus síncrono, cuya función principal es el acceso a los bloques RAM internos, utilizando un protocolo simple con un número mínimo de señales de control, para asegurar la lectura o escritura de la memoria en un solo ciclo.

Las interfases PLB y OPB son versiones de 32 bits de los buses IBM correspondientes, utilizados para proporcionar una conexión a bloques RAM y periféricos internos y externos. Ambas son interfases maestras con capacidad de habilitación de bytes.

La interfase XCL es una solución de alto rendimiento para accesos a memoria externa, que consiste en una interfase de alta velocidad, arbitrada del lado del esclavo, para la conexión directa del procesador a un controlador de memoria que incluye *buffers* FSL. Este el método de comunicación con la latencia más baja y el menor número de instanciaciones, que solo esta disponible cuando se habilita la memoria *cache*, en el lado de instrucciones, datos o en ambos.

El enlace FSL provee un canal de comunicación punto a punto no arbitrado de 32 bits, entre una FIFO de salida y una FIFO de entrada, por lo que representan el medio ideal para extender la capacidad del procesador, utilizando aceleradores hardware diseñados por el usuario. Esto es equivalente a la agregación de instrucciones especiales, ejecutadas por la lógica de la FPGA (Figura 3.15).

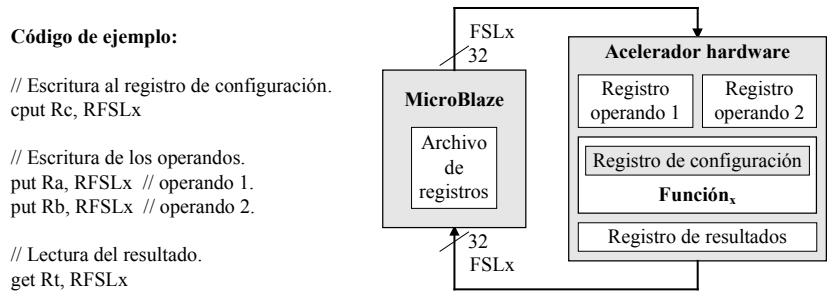


Figura 3.15. Función de aceleración de hardware del MicroBlaze vía FSL.

La interfase de depuración está diseñada para trabajar con el núcleo IP de depuración de microprocesadores (MDM), el cual es controlado por el depurador de microprocesadores Xilinx (XMD), a través del puerto JTAG de la FPGA. Para el análisis de comportamiento, el núcleo del MicroBlaze exporta una gran cantidad de señales internas por medio de la interfase de trazo, como la información de la instrucción actual, dirección destino, contador del programa, registros especiales, etc.

3.8.2 Procesador PowerPC 405.

El procesador PowerPC 405 es una realización embebida *hard* de 32 bits, de la arquitectura estándar de procesamiento industrial PowerPC Harvard RISC, con versiones 405D5 para FPGA Virtex-II Pro y 405F6 para Virtex-4 y Virtex-5. Estas

versiones utilizan un modelo de software que asegura la compatibilidad con toda la familia de procesadores PowerPC, al nivel de programas de aplicación. Su manejo le proporciona al usuario la posibilidad de desarrollar sistemas embebidos completos, con alta flexibilidad para las particiones hardware/software y gran capacidad de aceleración y co-procesamiento hardware [XilinxUG018].

3.8.2.1 Organización y características.

La organización del PowerPC 405 se divide en unidades de *cache*, unidad de administración de memorias (MMU), el GPP y los elementos de temporización y depuración (Figura 3.16), en una arquitectura que presenta las siguientes características generales:

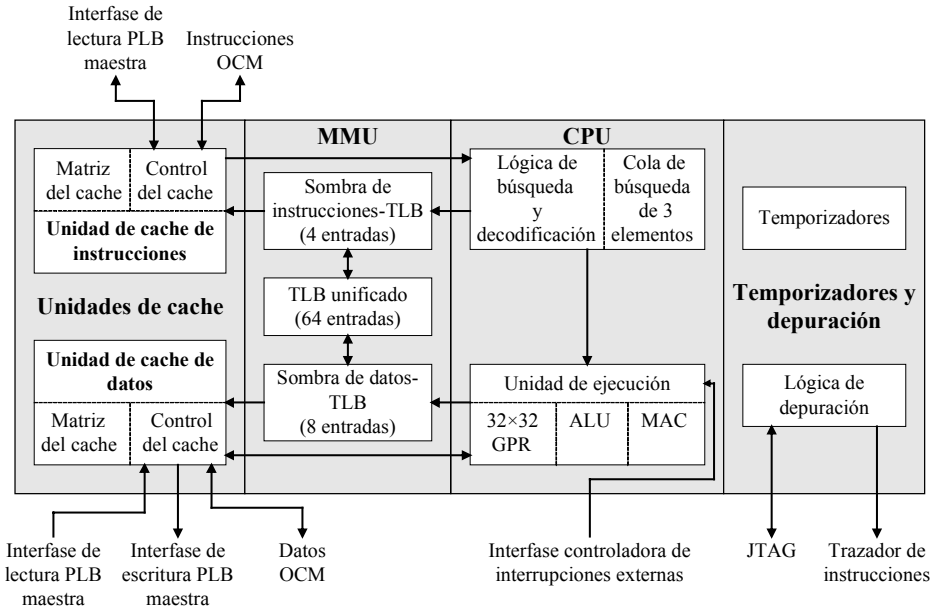


Figura 3.16. Organización del PowerPC 405.

* Núcleo *hard* de procesador embebido de 450 MHz, 700+ DMIPS.

* *Pipeline* de datos de 5 estados (búsqueda, decodificación, ejecución, re-escritura y carga de re-escritura), con ejecución de ciclo único de la mayoría de las instrucciones.

* Unidad de ejecución de punto fijo, con arquitectura de 32 bits, conteniendo 32 registros de 32 bits de propósito general.

* Módulos hardware de multiplicación/división, para acelerar la aritmética entera (multiplicación en 4 ciclos, división en 35 ciclos).

* Caches de datos e instrucciones, de conjuntos asociativos de 16 KB y 2 vías.

* Unidad de administración de memoria (MMU), que habilita la realización de sistemas operativos en tiempo real (RTOS), a partir de múltiples tamaños de páginas y una variedad de atributos de protección de almacenamiento y opciones de control de acceso.

* *Buffers* de traslación *Look-aside* (TLB) de 64 entradas unificadas.

* Tamaño de pagina variable (1 KB – 16 KB).

* Controladores avanzados de datos e instrucciones de memoria en *chip* (OCM), con interfase directa a los bloques de RAM.

* Soporte de arquitectura de bus IBM *CoreConnect*.

* Lógica de depuración utilizando la interfase JTAG y soporte de trazo.

3.8.2.2 Características de la arquitectura en el ambiente embebido.

El PowerPC 405 es una realización optimizada para ambientes embebidos, por lo que presenta características especificas que lo hacen propio para el diseño de SoC sobre FPGA:

* Administración de memoria optimizada para ambientes de software embebidos.

* Instrucciones de administración del *cache* para optimizar el rendimiento y el control de la memoria en aplicaciones gráfica y numéricamente intensivas.

* Extensiónes de la arquitectura del PowerPC para el soporte de aplicaciones embebidas: operaciones *true little-endian*, administración flexible de memoria, instrucciones de multiplicación-acumulación para aplicaciones de cómputo intensivas, capacidad de depuración avanzada, base de tiempo de 64 bits, temporizadores de intervalos programables y fijos y temporizador guardián.

* Registros de propósito especial para controlar el uso de los recursos de depuración, recursos del temporizador, interrupciones, atributos de almacenamiento en modo real, facilidades de administración de memoria y otros recursos del procesador.

* Un espacio de direcciones del registro de control del dispositivo para la administración de periféricos en *chip* tales como controladores de memoria.

* Estructura de interrupción de nivel dual e instrucción de control de interrupciones.

* Recursos de depuración que habilitan las funciones de depuración hardware y software, tales como puntos de rompimiento de instrucciones, puntos de rompimiento de datos y ejecución de programas paso a paso.

3.8.2.3 Interfases de entrada y salida.

El procesador PowerPC 405 cuanta con un grupo completo de interfases, puertos y líneas de entrada y salida, para su control, depuración e interconexión con la lógica y los componentes internos y externos de la FPGA:

* Interfase de reloj y administración de energía (CPM), que habilita que las aplicaciones sensibles al consumo de energía controlen el reloj del procesador utilizando lógica externa.

* Interfase de control del GPP, para proporcionar información de la configuración del GPP y reportar la detección de una condición de verificación de máquina al procesador.

* Interfase de *reset*, para iniciar por hardware el bloque de procesamiento, en la energización o en cualquier momento durante la operación normal. En este último caso, la ejecución de instrucciones es detenida inmediatamente y el procesador pierde su estado.

* Interfases del bus local del procesador (PLB), en los lados de instrucciones (ISPLB) y datos (DSPLB). El ISPLB habilita a la unidad de *cache* de instrucciones del procesador (ICU) para leer instrucciones como elemento maestro, desde cualquier dispositivo conectado al PLB, con un bus de direcciones de 30 bits y un bus de datos de 32 ó 64 bits. El DSPLB habilita a la unidad de *cache* de datos del procesador (DCU) para leer y escribir datos como elemento maestro, desde cualquier dispositivo de memoria conectado al PLB, con un bus de direcciones de 32 bits y dos buses de datos de 32 ó 64 bits (uno para lectura y otro de escritura).

* Interfase del registro de control del dispositivo (DCR), en su versión normal e interna. La interfase DCR normal proporciona un mecanismo para que el bloque procesador inicialice y controle los dispositivos periféricos que residen en la misma FPGA. La DCR interna puede ser usada para controlar, configurar y mantener el estado de las diversas unidades funcionales del bloque procesador.

* Interfase de bus DCR externo, la cual consiste en un bus de direcciones de 10 bits, buses de datos de entrada y salida separados de 32 bits y las señales de control y reconocimiento de lectura y escritura, para la conexión de una cadena distribuida y multiplexada de periféricos.

* Interfase controladora de interrupciones externas, para el manejo de las dos clases de interrupciones externas que soporta el procesador: criticas (de mayor prioridad) y no criticas.

* Puerto de depuración JTAG, para el soporte de depuración de software por medio de herramientas como el depurador GNU (GDB) incluido en el EDK.

* Interfase de depuración, que habilita a las herramientas de depuración externas para operar los recursos del procesador en modo de depuración externa, en el cual se puede alterar la ejecución normal del programa para depurar el sistema en hardware y software.

* Interfases de trazo, para la operación del modo trazo-depuración en tiempo real.

* Interfase del registro de versión del procesador (PVR), para el bloque procesador de la Virtex-4-FX, con la cual se accede a 8 bits del PVR, para identificar a cada procesador en un ambiente de multiprocesamiento o para codificar alguna descripción especial de procesamiento.

3.8.3 Multiprocesamiento.

Una de las alternativas para satisfacer los crecientes requerimientos de tiempo y procesamiento de SoC sobre FPGA es el uso de más de un procesador, en un ambiente de *chip* multiprocesador (MPC). Si bien esta es una solución de gran complejidad hardware/software, el manejo de herramientas como el EDK de Xilinx y la utilización de procesadores como el MicroBlaze y el PowerPC la hacen factible, de una forma rápida y flexible.

3.8.3.1 Factores que justifican el multiprocesamiento.

Antes de iniciar el desarrollo de un MPC, se debe justificar el uso de los dos o más procesadores que satisfacen eficientemente los requerimientos del diseño. Los argumentos que soportan la decisión final se pueden ubicar en los siguientes escenarios [XilinxWP262]:

* **Funciones independientes múltiples.** En diseños que necesitan ejecutar gran cantidad de tareas de procesamiento independientes, una solución puede ser repartir estas funciones en varios módulos de procesamiento, cada uno asignado a un procesador con sus respectivos periféricos.

* **Descarga del plano de control.** El manejo en un solo procesador de tareas de tiempo real y tareas que no lo son puede ser inadecuado. Una alternativa es utilizar un procesador esclavo para las tareas de tiempo real y un procesador maestro, para la ejecución de tareas no críticas, el monitoreo y control del procesador esclavo y la interfase con un nivel superior de control.

* **Descarga del plano de datos.** En un escenario de un gran número de tareas especiales o de procesamiento de protocolos, en conjunción con actividades regulares, la solución maestro/esclavo puede ser muy eficiente, ya que el procesador esclavo se puede dedicar a satisfacer los requerimientos del procesamiento intensivo de datos y el maestro puede atender la configuración del sistema, la coordinación del flujo de datos de entrada y salida y la interfase con el usuario.

* **Procesamiento de interfases.** En sistemas que actúan como un puente o conmutador entre múltiples interfases, como en un sistema de procesamiento de red, un procesador esclavo puede ser dedicado al procesamiento de datos de cada interfase, mientras que un procesador maestro puede realizar las tareas de alto nivel.

* **Procesamiento de flujos de datos.** En sistemas computacionales orientados al flujo de datos, se puede aumentar la tasa de procesamiento con un sistema *pipeline* de procesadores, donde cada procesador ejecuta una parte de las operaciones sobre el flujo de datos antes de pasar los resultados al procesador de la siguiente etapa.

* **Procesamiento simétrico.** Cuando un procesador único no proporciona él suficiente rendimiento, la solución puede ser el multiprocesamiento simétrico (SMP), donde un sistema operativo que administra tareas paralelas, como el Linux, reparte las funciones de una aplicación entre varios procesadores.

* **Confiabilidad y redundancia.** En sistemas donde la confiabilidad debe ser máxima, una solución es replicar varias veces el sistema de procesamiento, para obtener la redundancia que permita que el sistema sea tolerante a la falla de alguno de los sistemas replicados.

3.8.3.2 Arquitecturas de multiprocesamiento.

Un sistema de multiprocesamiento puede estar compuesto por una gran diversidad de procesadores, memorias, periféricos y componentes especiales, cuya topología, cantidad, tipo y características son funciones de los requerimientos del diseño. Ya que las posibilidades bajo este panorama son muy extensas, en [XilinxWP262] se presentan los conceptos de las arquitecturas de multiprocesamiento a partir de una topología de dos procesadores (PowerPC y MicroBlaze), en la cual se estudian los procesos de comunicación y coordinación típicos de estos sistemas (Figura 3.17).

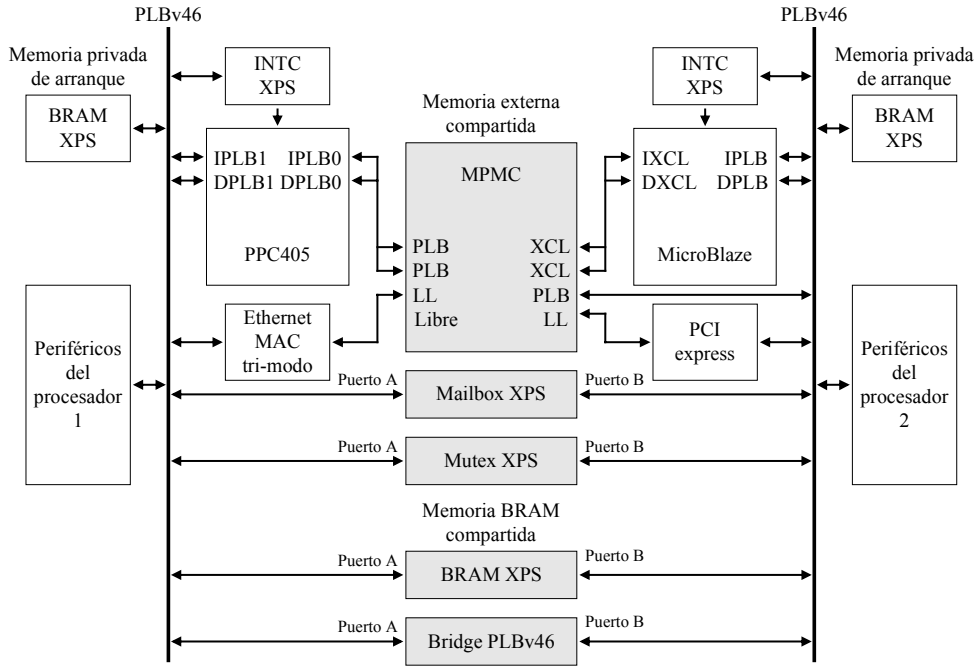


Figura 3.17. Arquitectura de multiprocesamiento, con procesadores PowerPC y MicroBlaze.

Los conceptos clave de la arquitectura de procesamiento dual son:

* La arquitectura multiprocesamiento se forma con la inclusión de componentes compartidos entre dos subsistemas de procesamiento independientes.

* Los subsistemas de procesamiento pueden ser heterogéneos, cada uno con su propia arquitectura, sistema operativo y sus dominios de *reset* y reloj. Este modelo de multiprocesamiento heterogéneo es muy propicio para sistemas embebidos sobre FPGA.

* Los componentes compartidos son de naturaleza multipuerto, lo que permite que cada bus del sistema (PLBv46) sea independiente uno del otro, en términos de carga estática y dinámica. El aislamiento de cada subsistema de procesamiento asegura que

un bus no sea corrompido por un procesador o periférico debido a la operación del otro procesador.

* Ya que la memoria compartida es la forma más adecuada de pasar información entre subsistemas de procesamiento, el principal periférico común es el controlador de memoria externa MPMC, ya que es el único controlador soportado por la herramienta XPS que ofrece hasta 8 interfases (puertos) a la misma memoria. Su uso permite que 3 ó 4 subsistemas de procesamiento puedan acceder simultáneamente a la memoria externa compartida, con mínima latencia y alto ancho de banda.

* También es posible compartir memoria BRAM interna entre procesadores, lo cual proporciona un canal de alta velocidad para el intercambio de datos. La BRAM se puede conectar al bus PLB o a las interfases locales de memoria de los procesadores (OCM en el PowerPC y LMB en el MicroBlaze).

* El uso del *Mailbox* es un método eficiente para transferir apuntadores y datos actuales entre procesadores, con mensajes menores a 100 bytes, por medio de un canal FIFO para transmitir y otro canal para recibir, en operación síncrona o asíncrona. Si bien este componente está disponible como un núcleo en el XPS, una alternativa para su realización con un funcionamiento similar es la interfase FSL, disponible en los procesadores PowerPC y MicroBlaze.

* Para la sincronización de los nodos de procesamiento, en el acceso a recursos compartidos (periféricos, memoria, etc.), se incluye el módulo de sincronización por hardware *Mutex*, el cual proporciona la habilidad para crear regiones de exclusión mutua entre varios procesadores.

* Cuando se necesita compartir un elemento que no es multipuerto, como los periféricos UART, SPI y I2C, la solución es conectar el periférico a un sistema procesador y utilizar un puente de bus a bus (por ejemplo de PLB a PLB), para que el otro procesador pueda acceder al componente.

* Para que el XPS genere automáticamente un mapa de memoria sin conflictos, cada subsistema incluye una sección de memoria privada, para ubicar el código de arranque y las rutinas de servicio a interrupciones y excepciones. El resto del software de cada procesador puede ubicarse en otros segmentos de memoria, privados o compartidos.

Otra opción de arquitectura de multiprocesamiento es la conexión de todos los procesadores a un solo bus de sistema, lo que ofrece un ahorro en área, al eliminar los buses particulares y reducir la necesidad de puertos múltiples en los componentes compartidos. El resultado es un sistema poco deterministico, con una sobrecarga de ejecución en el bus común. Una arquitectura intermedia puede tener un procesador de alto rendimiento en un bus independiente y varios procesadores de menor rendimiento en un bus compartido.

3.8.3.3 Realización de sistemas multiprocesamiento.

Los pasos para la definición hardware de sistemas de dos procesadores en el ambiente del EDK son los siguientes [Asokan07]:

1. Diseñar uno de los sub-sistema de procesamiento con todos sus componentes (procesador, periféricos y memorias locales y externas), para una tarjeta específica, utilizando la herramienta *Base System Builder* del EDK.

2. Incorporar al diseño el segundo procesador, con operaciones de arrastrar-soltar desde el catálogo de IPs; conectarlo a un nuevo bus de sistema.

3. Conectar la interfase de depuración del primer procesador al periférico de depuración. Si es necesario, configurar el periférico para acceder a ambos procesadores.

4. Adicionar la memoria local del segundo procesador. Empezar con los buses de memoria local y continuar con las conexiones al procesador y al controlador de memoria.

5. Conectar el segundo procesador al controlador de memoria externa MPMC, configurando nuevos puertos. Si se requiere, conectar los enlaces de *cache*.

6. Anexar los componentes de comunicación entre procesadores (*Mutex* y *Mailbox*) y realizar su configuración.

7. Agregar la memoria local compartida, con conexión al bus DLMB (MicroBlaze), al bus DOCM (PowerPC) o al bus PLB.

8. Si es necesario, utilizar periféricos puente entre los subsistemas.

9. Incluir los periféricos misceláneos, tales como temporizadores y controladores de interrupciones.

10. Ir a la sección de direcciones del panel de ensamble del sistema y configurar un rango de direcciones valido para todos los periféricos y memorias incluidas.

11. Verificar las conexiones de reloj y *reset* de todos los periféricos.

Capítulo **4**

Arquitecturas *Full-Search* de Estimación de Movimiento con Resolución de Pixeles Enteros

4.1 Introducción.

La estimación de movimiento con resolución de pixeles enteros “integer motion estimation” (IME) y algoritmo de ajuste de bloques con búsqueda completa “full-search block matching” (FSBM) es la técnica utilizada para la selección del mejor vector de movimiento en pixeles enteros de la recomendación H.264/AVC. Valorado como el proceso de mayor grado de complejidad computacional del sistema de codificación de video [Chen04a], su realización hardware debe ser altamente paralela para cumplir los requerimientos que demandan las aplicaciones en tiempo real.

Aun cuando el estándar H.264/AVC [H.264.05] no define su operación, el sistema IME-FSBM debe cumplir con una funcionalidad especifica para generar adecuadamente la información que recibe, interpreta y procesa un decodificador normalizado. La base para la realización y evaluación de diseños que soportan el estándar es el software de referencia [JM1106]. Desarrollado y actualizado por el grupo de trabajo del estándar, este modelo funcional documenta con programas en lenguaje C las técnicas, métodos y algoritmos que dan cumplimiento al codificador/decodificador. Los diseños propuestos en este capítulo se fundamentan en las funciones C incluidas en el archivo *mv-search.c* del software de referencia, el cual integra los programas de búsqueda del vector de movimiento para cuadros de video tipo predictivo (P) y bi-predictivo (B).

La figura 4.1 muestra el procedimiento general que utiliza el software de referencia en el flujo de predicción de bloques tipo P, con resolución entera y algoritmo FSBM (función C *FullPelBlockMotionSearch*). Este mismo software maneja un método rápido, donde la distorsión de bloques grandes es calculada a partir del coste del residuo de bloques 4×4 (función C *FastFullPelBlockMotionSearch*). En el caso de estimación de movimiento bi-predictivo (función C *FullPelBlockMotionBiPred*), el flujo de predicción es similar al mostrado e incluye dos bloques candidatos de diferentes cuadros cuyos pixeles son combinados linealmente antes de su comparación con los pixeles del bloque actual.

El proceso de predicción de cada macro-bloque (MB), lazo 2, del cuadro de video actual (lazo 1) se realiza en múltiples cuadros de referencia (lazo 3) y tamaño de bloque variable (lazo 4). El bloque actual se compara con todos los bloques candidatos del mismo tamaño dentro de una ventana de búsqueda “search window” (SW), lazo 5,

en un orden de barrido en espiral partiendo del centro de la ventana. La comparación se realiza utilizando un proceso de optimización tasa/distorsión, donde la tasa se obtiene con el coste de codificación de los bits del vector de movimiento y la distorsión de acuerdo al coste del residuo de predicción (lazo 6) en base al método de suma de diferencias absolutas "sum of absolute differences" (SAD). El resultado final es la posición del bloque candidato con mínimo coste con respecto al bloque actual (vector de movimiento), para cada partición y sub-partición del MB.

```
Lazo 1 (Cuadro actual)
  Lazo 2 (Macro-bloque en el cuadro actual)
    Lazo 3 (Cuadro de referencia)
      Lazo 4 (41 bloques y sub-bloques)
        Lazo 5 (Candidato en la ventana de búsqueda)
        {
          Coste del vector de movimiento
           Lazo 6 (Pixeles en los bloques actuales y candidatos)
           {
             Coste del residuo
             Candidato con mínimo coste total
           }
        }
```

Figura. 4.1. Lazos del procedimiento IME-FSBM en el software de referencia.

La naturaleza secuencial del procedimiento software anterior resuelve satisfactoriamente la dependencia de datos entre bloques vecinos, generando el flujo optimo de bits en términos de eficiencia de codificación y máximo nivel de compresión. En una realización hardware, esta dependencia de datos se puede resolver al nivel algorítmico [Huang03] o al nivel arquitectural [Zhao06] y así permitir un alto grado de procesamiento concurrente y segmentado. El coste es un resultado de codificación sub-optimo o una mayor área por la cantidad de procesadores necesarios, respectivamente.

Además de la dependencia de datos, un diseño hardware debe solucionar la complejidad computacional y el gran requerimiento de ancho de banda del sistema, que se incrementa linealmente con el número de cuadros de referencia. El nivel de computación puede resolverse utilizando arquitecturas de procesamiento paralelo diseñadas para altos niveles de flujo de datos y/o bajo consumo de energía. El ancho de banda del sistema puede reducirse con estrategias de reutilización de datos, utilizando memorias locales en los lazos 2 a 5 del procedimiento software anterior [Chen07].

Una arquitectura hardware de estimación de movimiento, diseñada como un periférico para un sistema microprocesador, está formada típicamente por los siguientes elementos: las memorias RAM locales para recibir los datos de la SW y el MB actual desde memorias externas, la ruta de procesamiento de datos para la comparación de bloques y el cálculo tasa/distorsión, la unidad generadora de direcciones para seleccionar localmente los datos de los bloques candidatos y actuales y la unidad de control del sistema, que generalmente es una FSM (Figura 4.2) [Kuhn99]. Alrededor de esta estructura genérica, se deben conjuntar las técnicas, métodos de diseño y procesamiento de datos, para satisfacer los requerimientos del H.264/AVC en la estimación de movimiento de pixeles enteros con algoritmo FSBM.

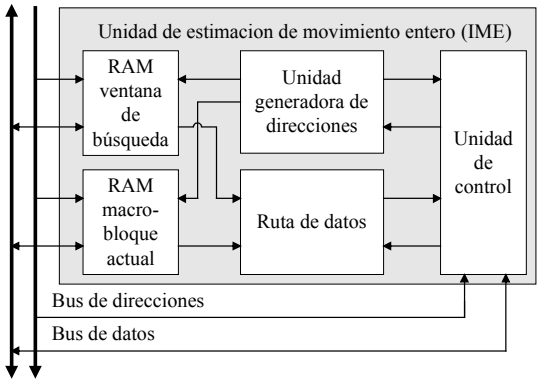


Figura. 4.2. Estructura general de una arquitectura hardware de estimación de movimiento entero para un sistema con microprocesadores.

Debido al progreso continuo de los circuitos integrados digitales, es posible diseñar una arquitectura FSBM con resolución de pixeles enteros, para satisfacer la tasa de procesamiento de formatos de video de alta definición (HD), utilizando tecnología VLSI. En las secciones siguientes se propone una arquitectura hardware que satisface la alta demanda computacional y ancho de banda, con una red sistólica de elementos procesadores y una eficiente estructura de memoria. Su optimo rendimiento y la regularidad del flujo de datos, trae beneficios como alta calidad en la estimación de movimiento y bajo direccionamiento de memoria, sin un gasto excesivo en el esquema de control.

A continuación, en la sección 4.2 se presenta el estado del arte de arquitecturas hardware IME-FSBM, en las secciones 4.3 y 4.4 se explican los diseños propuestos y en la 4.5 las conclusiones del trabajo realizado.

4.2 Estado del arte.

En la literatura técnica se han propuesto un gran número de arquitecturas hardware IME-FSBM, las cuales se pueden clasificar de acuerdo a la estructura y funcionalidad de su bloque más importante: la matriz de elementos procesadores "processor elements" (PEs). Se han catalogado en sistólicas y semi-sistólicas, de una dimensión (1-D) o dos dimensiones (2-D) y si cada PE es responsable del cálculo de distorsión de un píxel del bloque candidato (tipo 1) o del cálculo de distorsión de un píxel del bloque actual, para todos los bloques candidatos (tipo 2) [Vos89]. Aquí solo se consideran las arquitecturas sistólicas 2-D tipo 1, ya que estas son reconocidas como las más eficientes por su excelente comportamiento área/velocidad [Chen06a] [Huang03] [Vos89].

4.2.1 Arquitecturas clásicas.

A continuación se presentan algunas arquitecturas IME-FSBM clásicas que son consideradas como el estado del arte en la codificación de video. Aun cuando su estructura necesita ser reorganizada para soportar los más nuevos estándares de codificación de video, sus principios siguen siendo utilizados en arquitecturas recientes. Tres de ellas son documentadas en [Vos89], [Komarek89] y [Roma02].

Vos y Stegherr [Vos89] presentan una de las primeras y más eficientes arquitecturas sistólicas 2-D (Figura 4.3), donde el número de PEs es igual al tamaño del bloque actual (N×N). La estructura incluye m renglones de registros auxiliares en la parte superior e inferior de la matriz de procesamiento, donde M es el valor total del rango de exploración vertical y horizontal con valor $M = \lambda N - 1$, siendo λ un entero positivo. Para una SW de tamaño (M+N)×(M+N), la arquitectura procesa los datos evitando tiempos muertos, y calculando el valor SAD de la distorsión de cada bloque candidato en un ciclo de reloj.

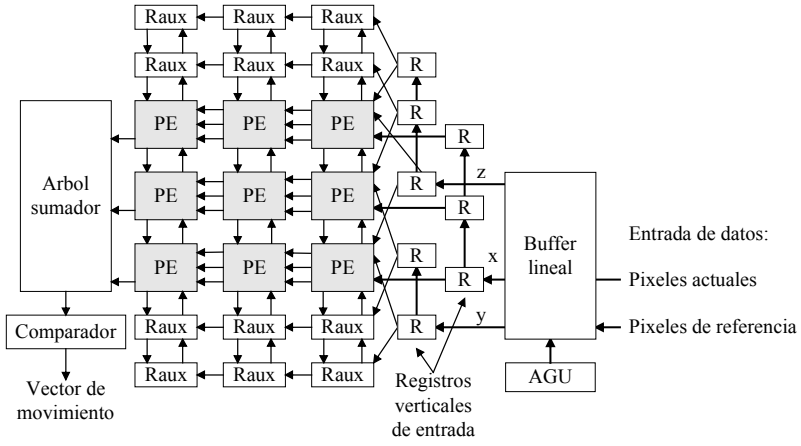


Figura 4.3. Arquitectura de Vos y Stegherr con N = 3, λ = 1 y M = 2.

La arquitectura recibe los datos del bloque actual por el puerto de entrada *x* y los datos de los bloques candidatos por los puertos de entrada *y* y *z*; estos últimos desde una estructura dinámica de memoria lineal, direccionada por una unidad generadora de direcciones "address generate unit" (AGU) tipo registro de desplazamiento. Dentro de la matriz de procesamiento, el bloque actual permanece fijo y los bloques candidatos se mueven hacia arriba, hacia abajo o a la izquierda con el apoyo de los registros auxiliares, en una secuencia de exploración de la SW tipo serpenteo. En el movimiento a la izquierda, la matriz de PEs y registros auxiliares reciben una columna completa de pixeles de la SW desde una red vertical de registros de entrada. La diferencia absoluta entre el píxel actual y el píxel candidato, calculada en cada PE, es acumulada horizontalmente y propagada hacia un árbol sumador.

Cada PE esta formado por un multiplexor que selecciona el píxel candidato desde los PEs superior, inferior o derecho adyacentes, tres registros para almacenar temporalmente los pixeles en proceso y una unidad aritmética para el cálculo de la diferencia absoluta entre los píxeles de los cuadros actual y previo (Figura 4.4). También incluye un conmutador (SWa) para transferir el píxel actual circulante a un registro de respaldo y un sumador para procesar las diferencias absolutas en forma horizontal.

Esta arquitectura esta diseñada para un rango de exploración vertical fijo de valor M, del cual depende el número de renglones de registros auxiliares arriba y abajo de la matriz de PEs. Para un tamaño vertical de SW menor al diseño, la arquitectura es sub-utilizada y para un tamaño mayor, su latencia, velocidad de procesamiento y eficiencia originales se degradan considerablemente.

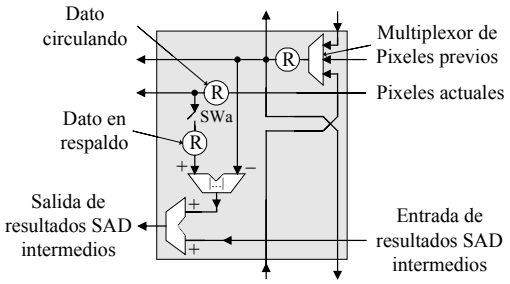


Figura 4.4. Elemento de procesamiento propuesto por Vos y Stegherr.

Komarek y Pirsch [Komarek89] utilizan una metodología con nodos de procesamiento y arcos de dependencia de datos para diseñar una matriz de PEs, a la que llaman AB2 (Figura 4.5). Cada PE calcula la diferencia absoluta "absolute difference" (AD) entre un píxel del bloque de datos actual, que se almacena y permanece fijo en la matriz y los píxeles del bloque de referencia que fluyen horizontalmente desde una RAM local. Los PEs acumulan los resultados AD parciales y los propagan verticalmente hacia los nodos de acumulación horizontal, que alimentan a un nodo de comparación. El flujo horizontal de datos permite un procesamiento secuencial de líneas consecutivas de la SW. La estrategia de exploración utiliza un direccionamiento básico de memoria, pero penaliza a la arquitectura con ciclos extras de reloj, debido a los registros que se ubican entre los bloques AD.

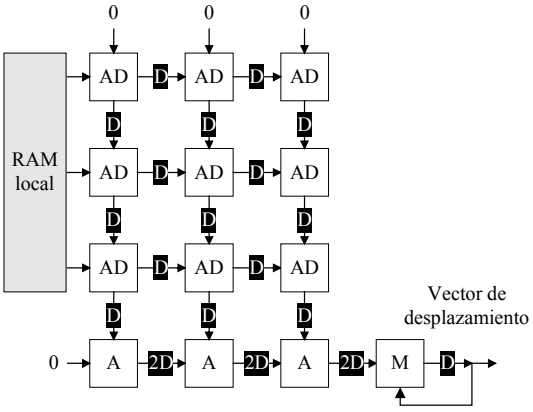


Figura 4.5. Arquitectura AB2 de Komarek y Pirsch, donde AD son los nodos SAD intermedios, A son los nodos de acumulación del SAD de columna, M es un nodo comparador y D son los registros de retardo para ciclos falsos, necesarios para la estrategia de exploración.

Roma y Sousa [Roma02] proponen un diseño mejorado de la arquitectura de Vos y Stegherr, al presentar una estructura que reduce la cantidad de memoria necesaria para almacenar la SW, logrando a la vez una mayor utilización de los recursos hardware (Figura 4.6). Al colocar la arquitectura original sobre una superficie cilíndrica, Roma y Sousa observaron que el grupo de elementos pasivos se sobreponen y aun cuando se descarte uno de ellos, la eficiencia del esquema de Vos y Stegherr se preserva, con una reducción del 50% en la cantidad de registros auxiliares. Esta modificación mantiene las demás características de la arquitectura clásica, incluyendo las limitaciones por el tamaño fijo de la SW.

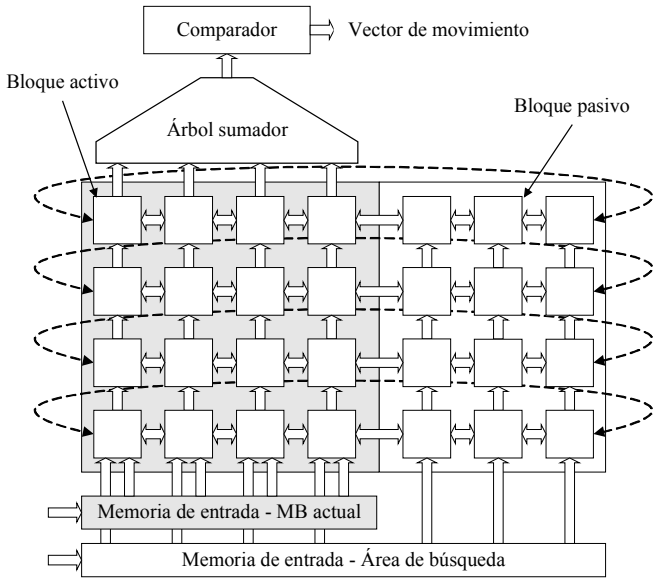


Figura 4.6. Arquitectura de Roma y Sousa, la cual es una versión mejorada del diseño de Vos y Stegherr.

Las arquitecturas clásicas citadas incluyen desde un mínimo número de registros auxiliares (Komarek y Pirsch), hasta una gran cantidad de ellos (Vos y Stegherr), observándose que mientras más elementos pasivos tienen, mayor es la degradación de sus características de operación al cambiar el tamaño de la SW. Una arquitectura eficiente debe optimizar su área con un mínimo número de elementos pasivos y un rendimiento independiente al tamaño de la SW.

4.2.2 Arquitecturas modernas.

A continuación se analizan las características de tres arquitecturas IME-FSBM diseñadas para la recomendación H.264/AVC. Estas estructuras modernas son normalmente versiones modificadas de los sistemas clásicos, para cumplir los requerimientos del nuevo estándar.

Zhang y Gao [Zhang05] describen una arquitectura similar al diseño de Roma y Sousa, con una matriz de N×N PEs, un grupo de registros auxiliares y un árbol sumador que auxilia en el cumplimiento del tamaño de bloque variable del estándar H.264/AVC (Figura 4.7). La estructura también incluye la optimización tasa/distorsión y un algoritmo adaptable que reduce el tamaño de la SW en función del grado de actividad de movimiento. La arquitectura fue diseñada para rangos pequeños de exploración vertical. Para desplazamientos mayores a 8, la SW se divide en dos partes para procesar los datos secuencialmente, con una penalización de 16 ciclos de reloj. La memoria de los pixeles de referencia debe tener el número suficiente de puertos para alimentar una columna de la matriz en un ciclo de reloj.

Huang et al. [Huang03] desarrollaron una arquitectura IME-FSBM utilizando una matriz sistólica de 16×16 PEs, un flujo de datos avanzado y un proceso de reutilización de resultados parciales (Figura 4.8). La matriz de PEs es la encargada de calcular el SAD de 16 bloques de 4×4 pixeles, que un árbol sumador reutiliza para obtener la distorsión de los bloques mayores. Para minimizar el tiempo de carga del

bloque candidato cuando la posición de búsqueda cambia en dirección horizontal, los autores presentan un flujo de datos complejo, donde 32 módulos locales de memoria SRAM alimentan a la matriz de procesamiento en un ciclo de reloj. Para ello, cada PE incluye dos puertos de entrada, con registros de 8 bits para los píxeles del área de búsqueda. Con un rango de búsqueda fijo, la arquitectura únicamente calcula el coste de la distorsión para determinar los mejores bloques candidatos.

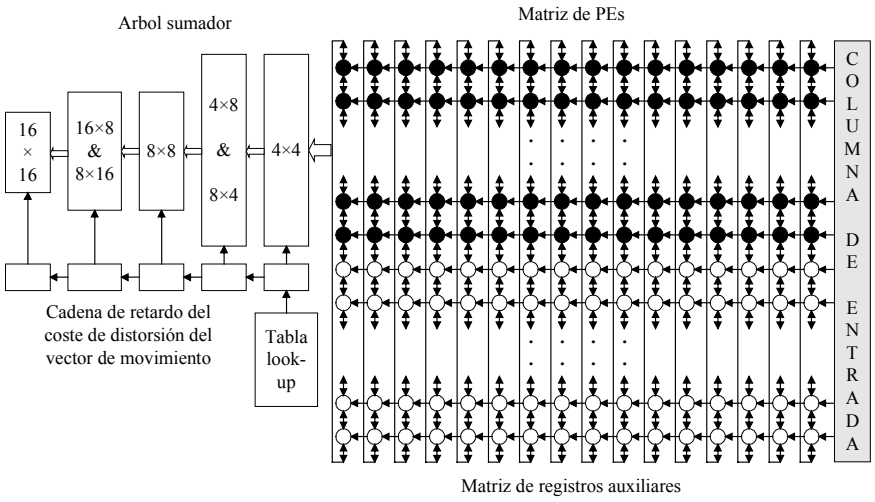


Figura 4.7. Arquitectura de Zhang y Gao, diseñada para satisfacer el tamaño de bloques variables del estándar H.264/AVC.

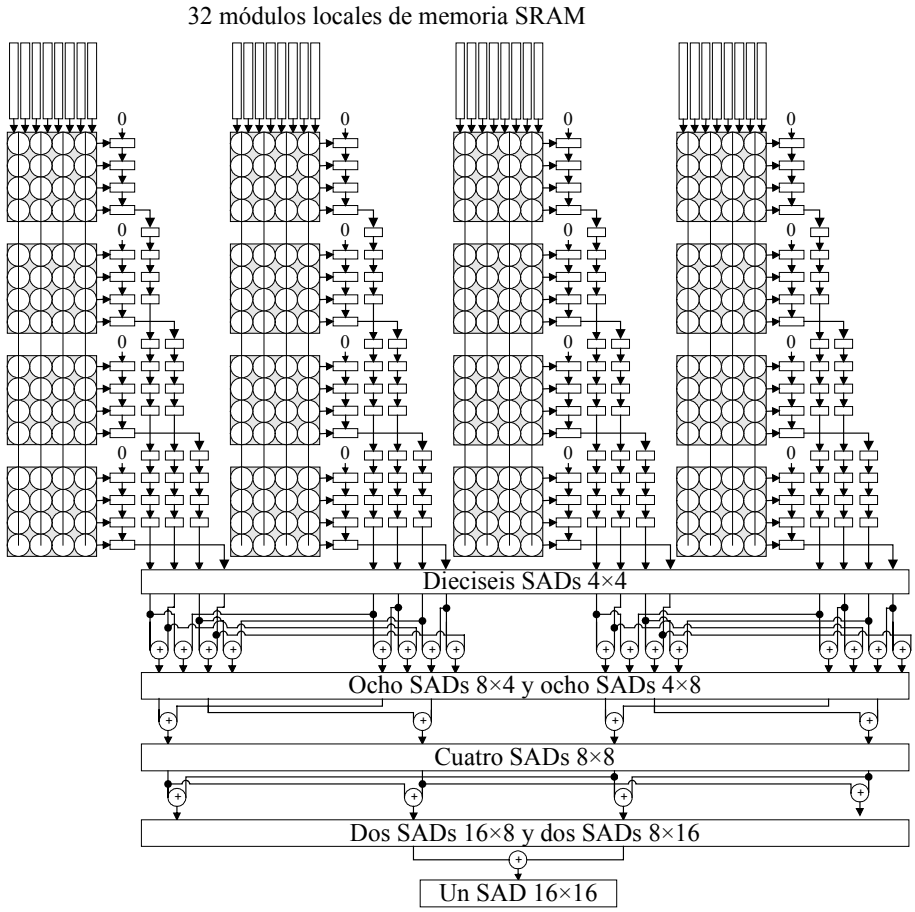


Figura 4.8. Arquitectura de Huang et al. con una matriz de procesamiento 16×16, un flujo de datos avanzado y un proceso de reutilización de resultados parciales.

Chen et al. [Chen06a] proponen un diseño que satisface los requerimientos del H.264/AVC con una matriz sistólica de PEs, un árbol sumador y un *buffer* de referencia reconfigurable (Figura 4.9), con lo cual se maximiza la reutilización de datos entre comparaciones sucesivas de bloques con un diseño de bajo coste, alta utilización y área mínima. El rendimiento de la arquitectura es independiente al tamaño del área de búsqueda. Los autores no incluyen la estructura y la lógica de administración de la memoria que soporta el flujo de datos.

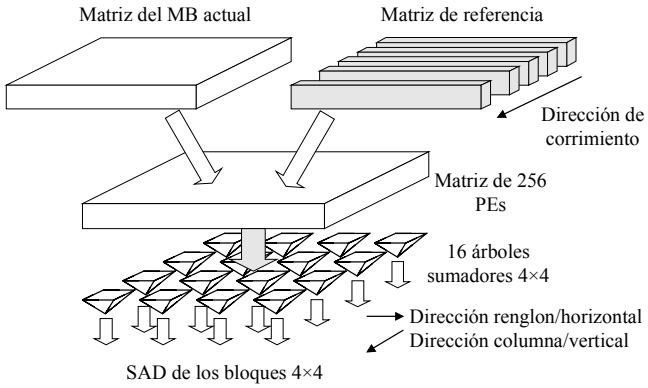


Figura 4.9. Diagrama básico de la arquitectura de Chen et al.

4.2.3 Evaluación de las arquitecturas previas y conclusiones.

4.2.3.1 Evaluación.

En esta sección, las arquitecturas previas son evaluadas desde un punto de vista hardware, utilizando mediciones de diseño como regularidad en la secuencia de direcciones, latencia, velocidad de procesamiento, eficiencia, ancho de banda de entrada y salida (E/S) [Chen06a] [Kuhn99], etc. La tabla 4.1 concentra la información de las 6 arquitecturas, para una matriz de procesamiento de N×N PEs y rangos de búsqueda [-Ph, +Ph] y [-Pv, +Pv], donde Ph y Pv son los desplazamientos de búsqueda horizontal y vertical respectivamente. Se incluye el número de registros para los pixeles candidatos (*buffer* de referencia) en la matriz de procesamiento, ya que es interesante observar su relación con el ancho de banda de E/S, con el ancho en bytes de la memoria de la SW y con la flexibilidad de la arquitectura ante el cambio de tamaño de la SW. También se señala la inclusión en el diseño de la estructura y administración de la memoria de la SW. Para una comparación menos abstracta, la tabla 4.2 muestra el valor de las mediciones de diseño para N = 16, Ph = 24 y Pv = 16.

La regularidad del direccionamiento de la memoria de la SW está en función del número de bits de cada palabra de dirección generados por lógica adicional, con respecto a la cantidad de bits que son generados en forma básica por un contador [Kuhn99]. Considerando que en las arquitecturas evaluadas no se describe su lógica de direccionamiento, su nivel de direccionamiento se puede aproximar por el grado de regularidad del flujo de datos de entrada a la matriz de PEs. Los diseños de Huang et al. [Huang03] y Chen et al. [Chen06a] tienen un flujo de datos complejo y no-regular. La arquitectura de Zhang et al. [Zhang05] presenta un flujo de datos con regularidad media en función del rango de búsqueda vertical. Las propuestas de Vos et al. [Vos89],

Komarek et al. [Komarek89] y Roma et al. [Roma02] utilizan un flujo de datos lineal con alta regularidad.

Tabla 4.1. Evaluación hardware de las arquitecturas consideradas como el estado del arte en el procesamiento IME-FSBM.

Arquitectura	Regularidad del direccionamiento RAM_{SW}	Velocidad de procesamiento (ciclos/MB)	Latencia (ciclos)	Ancho RAM_{SW} (bytes)	Buffer de referencia (bytes)	Incluye E/A[b] memoria SW
Vos89	alta	$(2Ph+1)(2Pv+1)+N(2Pv+1)^a$	$N(2Pv+1)$	2^a	$N^2+4PvN+2Pv+N$	parcial/ parcial
Komarek89	alta	$(2Ph+1)(2Pv+N)^a$	$2Ph+N$	N	$N(N-1)$	parcial/ no
Roma02	alta	$(2Ph+1)(2Pv+1)+N(2Pv+1)^a$	$N(2Pv+1)$	2^a	$N^2+2PvN+2Pv+N$	no/no
Zhang05	media	$(2Ph+1)(2Pv+1)+N^a$	N	$2Pv+N$	N^2+2PvN	no/no
Huang03	baja	$(2Ph+1)(2Pv+1)+N-1^a$	N	$2N^a$	$2N^2$	parcial/ no
Chen06	baja	$(2Ph+1)(2Pv+1)+N-1^a$	N	$N+1^a$	$N(N+1)$	parcial/ no

a-Peor caso, b-Estructura/Administración.

Tabla 4.2. Características estructurales de las arquitecturas evaluadas para N=16, Ph=24 y Pv=16.

Arquitectura	Velocidad de procesamiento (ciclos/MB)	Latencia (ciclos)	Eficiencia (%)	Ancho RAM_{SW} (bytes)	Ancho de banda* E/S (bytes/MB)	Buffer de Referencia (bytes)
Vos89	2145	528	75.38	2	7 362	1328
Komarek89	2352	64	68.75	16	40 704	31
Roma02	2145	528	75.38	2	7 362	816
Zhang05	1633	16	99.02	48	81 456	768
Huang03	1632	16	99.08	32	55 296	512
Chen06	1632	16	99.08	17	30 816	272

* Para el peor caso de ancho de la RAM_{SW}.

La velocidad de procesamiento (N_{ciclos}) define el número necesario de ciclos de reloj para realizar una secuencia de exploración de la SW. N_{ciclos} consta de tres términos: latencia, ciclos activos de operación y ciclos falsos. La latencia es el número de ciclos de reloj que el hardware necesita para generar el primer resultado. Los ciclos activos de operación corresponden al tiempo en el cual los PEs están ocupados. Los ciclos falsos son aquellos ciclos extras utilizados para preservar un flujo de datos regular. En las expresiones de N_{ciclos} de la tabla 4.1 se considera el peor caso, cuando no existe solapamiento entre las SWs de los MBs actuales adyacentes. Esto hace que las exploraciones de las SWs para MBs actuales consecutivos sean totalmente independientes y que cada una incluya la latencia y los ciclos activos y falsos.

La eficiencia (η) se define como la razón entre el número de ciclos activos y el número total de ciclos de operación. Para las arquitecturas evaluadas, su cálculo se realiza con la formula (4.1).

$$\eta = (2Ph+1)(2Pv+1)/ N_{ciclos} \quad (4.1)$$

El ancho de banda de E/S (B) se define como el número de accesos (lectura y escritura) a la memoria de búsqueda (RAM_{SW}), que se realizan en una exploración completa de la SW [Kuhn99] (4.2).

$$B = \text{tamaño de la SW} + (N_{ciclos} \times \text{ancho } RAM_{SW}) \quad \text{bytes/MB} \qquad (4.2)$$

El ancho de la RAM_{SW} es definido como el número de bytes que una matriz sistólica puede recibir simultáneamente desde la RAM_{SW}, durante el proceso de estimación de movimiento [Chen06a]. Para un alto grado de paralelismo, se necesita un gran ancho en bytes de la memoria de búsqueda.

Para el número de PEs y los rangos de búsqueda utilizados en la tabla 4.2, los diseños propuestos en [Zhang05], [Huang03] y [Chen06a] tienen valores casi ideales de velocidad de procesamiento, latencia y eficiencia. Pero, en forma similar a las arquitecturas en [Vos89] y [Roma02], el diseño en [Zhang05] tiene un rendimiento dependiente de su realización: para un desplazamiento vertical de exploración Pv mayor al valor utilizado, la arquitectura aumenta su latencia y agrega ciclos falsos, a diferencia de los diseños de Huang et al. y Chen et al. que son independientes del tamaño de la SW.

En los diseños presentados en [Vos89] y [Roma02], se puede observar la relación entre el tamaño del *buffer* de referencia, el ancho de la RAM_{SW} y el ancho de banda de E/S: para un *buffer* de reutilización de datos de gran tamaño, el ancho de la RAM_{SW} y el ancho de banda de E/S son bajos; lo contrario también es correcto (Chen06). Pero si una arquitectura tiene un *buffer* grande y valores altos en el ancho de la RAM_{SW} y en el ancho de banda de E/S, posiblemente existe un problema estructural para los rangos de búsqueda considerados, como el diseño en [Zhang05].

4.2.3.2 Conclusiones.

Como resultado del estudio y evaluación de las arquitecturas consideradas como el estado del arte, se tienen las siguientes conclusiones de su estructura, operación y rendimiento, que pueden tomarse en cuenta para diseñar nuevos sistemas IME-FSBM, enfocados a satisfacer los requerimientos de los codificadores modernos como el H.264/AVC:

* La regularidad en el direccionamiento de los pixeles candidatos depende directamente de la estrategia de exploración de la SW, de la estructura y administración de la memoria y del acoplamiento entre la memoria y la matriz de PEs, por lo que en un nuevo diseño se deben atender globalmente estos conceptos, para minimizar la complejidad de las unidades generadoras de direcciones.

* Para optimizar la velocidad de procesamiento de una arquitectura IME-FSBM, se necesita disminuir su latencia y eliminar los ciclos falsos de operación, además de ejecutar el proceso de comparación de bloques en el menor tiempo posible. La latencia corresponde básicamente al proceso de carga inicial de la matriz de PEs y los ciclos falsos se utilizan para preservar un flujo regular de datos, por lo que se debe prestar mucha atención al movimiento de los datos en la matriz de procesamiento para minimizar estos términos. El tiempo de procesamiento se puede controlar con el nivel de paralelismo de la arquitectura.

* El nivel de paralelismo de una arquitectura debe ser proporcional al ancho en bytes y al ancho de banda de la memoria de búsqueda.

* Una arquitectura mantiene los valores de las mediciones de diseño en toda la gama de rangos de búsqueda vertical y horizontal, solo si su operación es independiente del tamaño de la SW.

* El *buffer* de referencia apoya el flujo de pixeles en la matriz de PEs. Típicamente su tamaño y estructura es inversamente proporcional al ancho de la RAM_{SW}, al ancho de banda de E/S, a la flexibilidad de la arquitectura ante cambios en el tamaño de la SW y al nivel de reutilización de datos deseado.

* El diseño de un estimador de movimiento entero está incompleto si solo se considera la matriz de procesamiento. Es necesario incluir en la propuesta la estructura y administración de las memorias de pixeles actuales y candidatos y los elementos lógicos necesarios para su integración en el sistema.

* El uso de memoria local para almacenar los bloques candidatos de la SW aumenta el nivel de reutilización de datos, por lo que no se incrementa el ancho de banda de E/S del sistema cuando el diseño necesita acceder repetidamente a determinados pixeles.

4.3 Arquitectura *integer full-search* propuesta.

En esta sección se presenta una nueva arquitectura hardware que satisface los requerimientos de estimación de movimiento con resolución de pixeles enteros del estándar H.264/AVC. Buscando obtener una estructura óptima, en su diseño se tomaron en cuenta las conclusiones del estudio y evaluación de las arquitecturas del estado del arte, resultados que pueden considerarse como guías generales para el desarrollo de esta arquitectura (sistólica 2-D, tipo 1). De acuerdo a lo anterior, las principales características del diseño IME-FSBM que se presenta son:

* Procesamiento de cada bloque candidato en un ciclo de reloj.

* Latencia mínima.

* Cero ciclos falsos.

* Operación independiente del tamaño de la SW.

* Tamaño mínimo del *buffer* de referencia.

* Alto grado de paralelismo.

* Reutilización de datos al nivel de pixeles y bloques candidatos.

* Unidad generadora de direcciones básica.

El diseño propuesto utiliza una estrategia de exploración de la SW tipo serpenteo de izquierda a derecha, con la RAM_{SW} compuesta por 4 bloques de memoria

direccionados en paralelo y un módulo auxiliar para la selección de los pixeles que necesita la matriz de PEs en cada ciclo de reloj. El manejo conjunto de estos tres conceptos permite una alta regularidad en el direccionamiento de los datos a partir de una ecuación de direcciones simple.

Para lograr la mayor velocidad de procesamiento, la arquitectura propuesta fue diseñada con un valor mínimo de latencia, cero ciclos falsos y el procesamiento de cada MB candidato en un ciclo de reloj. Esto se obtiene gracias a un ancho de 32 bytes de la RAM_{SW}, lo que permite un alto nivel de paralelismo y a un flujo eficiente de datos a la entrada y dentro de la matriz de procesamiento.

La arquitectura mantiene sus valores de latencia, ciclos falsos y tiempo de procesamiento al modificarse los rangos de exploración, ya que por diseño, su operación es independiente del tamaño de la ventana de búsqueda. Esta flexibilidad ante cambios en el tamaño de la SW se debe principalmente a la estructura y tamaño del *buffer* de referencia.

Los pixeles del MB actual y de la SW se almacenan en memorias locales para permitir la reutilización de los datos, sin incrementar el ancho de banda de E/S del sistema. El diseño incluye la estructura y administración de estas memorias y las unidades generadoras de direcciones correspondientes.

Antes de documentar la estructura, organización y operación de la arquitectura propuesta, en las siguientes secciones se presentan algunos conceptos críticos en el diseño de sistemas FSBM: el ancho de banda de las memorias, la reutilización de datos y la orientación hardware del software de referencia. La aplicación de estos tópicos tiene el objetivo de considerar todos las variables que conlleven a obtener un producto optimizado en área y velocidad, que cumpla con el perfil de funcionalidad deseado.

4.3.1 Ancho de banda y reutilización de datos.

En el diseño de arquitecturas hardware de estimación de movimiento con algoritmo *full-search* se deben atender dos aspectos de suma relevancia: la complejidad computacional y el ancho de banda de las memorias. El primero se puede resolver con estructuras paralelas de múltiples PEs, pero si las memorias del sistema no tienen el suficiente ancho de banda para proporcionar la tasa de datos que necesitan los elementos procesadores, estos estarán subutilizados.

El ancho de banda de las memorias es uno de los principales cuellos de botella en la realización de sistemas de codificación de video con algoritmo FSBM como el H.264/AVC. Su valor se incrementa al ritmo de las necesidades cada vez mayores de procesamiento de video de alta calidad y resolución, ya que es función de factores como la velocidad (en fps), el formato, el número de cuadros de referencia considerados en la estimación de movimiento, los rangos de búsqueda, el tipo de algoritmo de predicción utilizado y el grado de redundancia en el acceso de los datos (Ra). Todos ellos son parámetros de diseño y normalmente no está a consideración del diseñador modificarlos, excepto Ra.

Ra define cuantas veces en promedio se necesita acceder a cada píxel en memoria, para realizar la tarea de estimación de movimiento (4.3) [Tuan02]. Lo ideal es

acceder solo una vez ($Ra_{min} = 1$) , pero si la arquitectura es poco eficiente, su valor puede llegar a ser igual al producto de los desplazamientos de exploración ($Ra_{max} = Ph \times Pv$).

Ra = número total de accesos a memoria en la tarea / número de pixeles en la tarea (4.3)

El ancho de banda de las memorias puede ser reducido con la planeación cuidadosa de la secuencia de datos, el uso apropiado de memorias locales y la aplicación de técnicas de reutilización Estos métodos y técnicas permiten minimizar el grado de redundancia del sistema hasta cumplir los requerimientos de ancho de banda de la aplicación. Por ejemplo, arquitecturas con bajos niveles de reutilización presentan un alto valor en Ra y demandan un mayor ancho de banda y aquellas con altos niveles de reutilización presentan un valor de Ra pequeño y menor ancho de banda.

Las técnicas de reutilización se aplican principalmente en procesos donde se tienen datos comunes entre operaciones sucesivas o en procesos con alto grado de manipulación de información. Los datos del MB actual no son susceptibles para la aplicación de estos métodos, por sus características de independencia (los pixeles de un MB actual no se traslapan con los de otro MB) y manejo (los pixeles de un MB actual normalmente se mantienen fijos en la matriz de procesamiento durante su tiempo de vida), por lo que se tienen mejores resultados usando memorias o registros locales. En cambio, para los pixeles de la SW, las técnicas de reutilización representan la mejor solución para reducir sus accesos redundantes.

Por la naturaleza del procesamiento FSBM, los pixeles de la SW muestran un alto grado de redundancia debido a sus traslapes internos (entre MB candidatos adyacentes) y externos (entre SW adyacentes), por lo que se pueden aplicar varios niveles de reutilización, que disminuyan el ancho de banda de la memoria de los cuadros de referencia. El primer nivel se presenta entre MB candidatos adyacentes, en donde para el primer MB se debe acceder a la totalidad de sus pixeles y para los siguientes MBs en la misma dirección de exploración, solamente se necesita acceder al renglón (ó columna) de pixeles que lo completan. En el segundo nivel se aprovecha el traslape entre renglones (o columnas) adyacentes de MBs dentro de la SW (Figura 4.10), en un proceso que requiere mayor cantidad de memoria local y un direccionamiento complejo. El tercer y cuarto nivel son similares a los dos anteriores pera ahora en vez de MBs candidatos, se considera toda el área de la SW y su traslape vertical u horizontal con la SW del MB actual adyacente.

La reutilización al nivel de SW se aplica cuando para MBs actuales adyacentes, sus SWs son a la vez adyacentes. Esta condición no es siempre valida, ya que en el software de referencia del estándar H.264 [JM1106], se logra una máxima ganancia de codificación con baja complejidad cuando el centro de la SW se predice con los vectores de movimiento de los bloques vecinos ya codificados, procurando que el vector de desplazamiento cero se incluya en la búsqueda. Esto hace que los niveles de reutilización con un traslape ordenado de las SW no se consideren en este trabajo.

En [Chen07] se propone un quinto nivel de reutilización, donde los datos de una SW en un cuadro de referencia son usados por los MBs actuales en la misma posición en diferentes cuadros originales (un solo cuadro de referencia, múltiples MBs actuales), logrando gran reducción en el tamaño de la memoria local y en el ancho de banda de las

memorias externas (Figura 4.11), lo que permite soportar múltiples cuadros de referencia con los mismos recursos hardware que los diseños para un único cuadro de referencia. Este esquema de reutilización es valido cuando los MBs actuales en la misma posición y diferentes cuadros originales usan la misma SW (por ejemplo, para estimación de movimiento con la SW centrada en el vector de desplazamiento cero), pero si los MBs actuales ubican su SW según la predicción del vector de movimiento, las SWs ya no coinciden y este método deja de ser operacional.

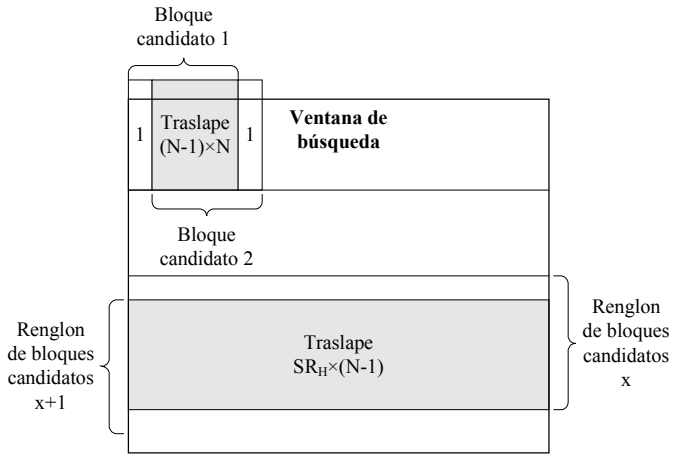


Figura 4.10. Niveles de reutilización entre MBs adyacentes y entre renglones adyacentes de MBs.

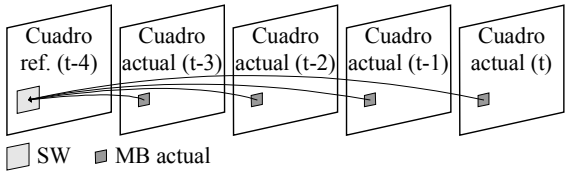


Figura. 4.11. Nivel de reutilización de un solo cuadro de referencia, múltiples MBs actuales

La reutilización de datos no siempre se enfoca a disminuir el ancho de banda. En sistemas de tamaño de bloque variable, el cálculo de la distorsión de los bloques grandes a partir de la distorsión de los bloques pequeños representa un esquema de reutilización enfocado a reducir el coste computacional del módulo de estimación.

Al igual que para el MB actual, el uso de memorias locales para los datos de la SW permite tener un grado de redundancia unitario (Ra = 1) visto desde el lado de la memoria externa, por lo que los métodos de reutilización se pueden aplicar en los accesos a las memorias locales para aumentar la eficiencia del diseño y disminuir el número de ciclos de reloj de la tarea de estimación. La memoria local para los pixeles actuales es igual al tamaño del MB y la memoria mínima para los pixeles candidatos es igual al tamaño de la región de traslape en cada nivel de reutilización de datos.

4.3.2 Algoritmo orientado a hardware.

El software de referencia del H.264/AVC puede ser tomado como modelo para el desarrollo de codificadores de video, ya que genera el flujo de bits que necesita un decodificador estandarizado, con alta eficiencia y máxima tasa de compresión. Para el

diseño de codificadores hardware bajo este software, puede ser preciso modificar su operación secuencial, para que sus algoritmos sean factibles de realizar en arquitecturas altamente segmentadas-paralelas, sin sacrificar la calidad de compresión.

En la estimación de movimiento con resolución entera, el modo de decisión de Lagrange en el software de referencia adopta un procesamiento secuencial de cada bloque en el MB, lo cual mejora el comportamiento del codificador, pero causa dependencia de datos, que hace del procesamiento hardware una tarea compleja. La dependencia se presenta básicamente entre el bloque a procesar y su bloque vecino izquierdo, ya que para la definición del centro de la SW para cada MB, bloque y sub-bloque a estimar, se necesita calcular el MV de predicción (MVp) con el valor medio de los MVs de los bloques izquierdo, superior y superior derecho (Figura 4.12a). Si alguna de las particiones no está disponible (por ejemplo en los bordes del cuadro o de un sector), el cálculo del MVp se modifica consecuentemente [H.264.05].

Un método para resolver la retroalimentación de datos es utilizar como centro de la SW el vector de desplazamiento cero [Huang03], lo que permite aplicar técnicas de reutilización al nivel de las SWs. Otra solución es aproximar el MVp con el valor medio de los vectores de los MBs superior izquierdo, superior y superior derecho, para su uso en el procesamiento de estimación de movimiento de todos los bloques y sub-bloques que forman el MB (Figura 4.12b). Esta y otras modificaciones pueden causar una degradación de la razón pico de señal a ruido "peak signal-to-noise ratio" (PSNR) hasta de 1.65 dB a bajas tasas de bits y en secuencias de video donde los MVs reales son mayores al rango de búsqueda. Para minimizar este efecto, se propone utilizar un rango de búsqueda de al menos [-24, +24] pixeles y seleccionar el centro de la SW con el valor aproximado del MVp.

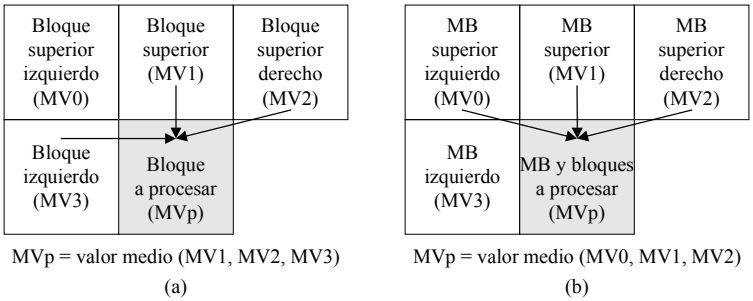


Figura 4.12. Cálculo del MVp cuando estan disponibles todas las particiones, a) al nivel de bloques, de acuerdo al estándar y b) aproximación con el valor medio de los MVs de los MBs superiores.

No siempre la solución es eliminar los lazos de retroalimentación. Existe un método que mantiene el procesamiento secuencial y la dependencia de datos del software de referencia, preservando la continuidad en el campo de movimiento, aplicando un algoritmo de procesamiento paralelo llamado *Wavefront Parallelization* [Zhao06]. Basado en técnicas de partición de datos y planeación de tareas en un sistema multiprocesador, el algoritmo opera concurrentemente en varios cuadros actuales para lograr alta eficiencia y velocidad de codificación (Figura 4.13).

4.3.3 Descripción del diseño IME-FSBM.

La arquitectura hardware propuesta satisface los requerimientos de estimación de movimiento con resolución entera del estándar H.264/AVC, utilizando una matriz sistólica de 16×16 PEs, un árbol sumador para calcular la distorsión SAD de 41 bloques

y sub-bloques, un módulo de optimización tasa/distorsión, una unidad generadora de direcciones y una unidad de control, así como de la lógica de manejo y selección de los datos de entrada (Figura 4.14). En esta sección se describe el diseño de cada componente y su integración en el estimador, independientemente de la tecnología utilizada para su realización.

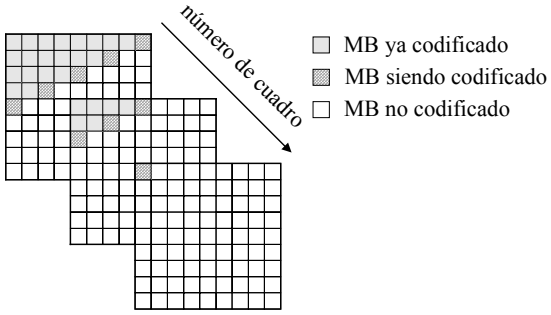


Figura 4.13. Procesamiento concurrente de varios cuadros actuales con el algoritmo *Wavefront Parallelization*.

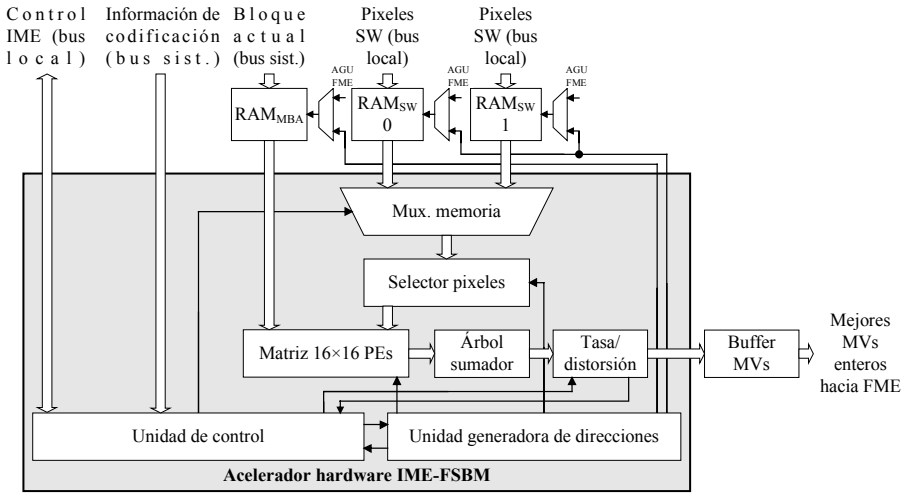


Figura 4.14. Diagrama a bloques de la arquitectura IME-FSBM propuesta.

Esta arquitectura fue desarrollada como un acelerador hardware que funciona como un periférico para dos procesadores, el local para la etapa de estimación de movimiento y el del sistema para todo el codificador. El primer procesador se comunica con la unidad de control para administrar su operación y controlar la transferencia de los pixeles desde la memoria de cuadros de referencia (RAM externa) hasta las memorias de la SW (RAM_{SW}). Antes de que el procesador local dé la orden de inicio de operaciones e indique la posición de los pixeles candidatos, el procesador del sistema envía la información de configuración a los registros de control.

El módulo IME lee los pixeles de referencia de las memorias $RAM_{SW,}$ conectadas al bus local y los pixeles del MB actual de la memoria RAM_{MBA}, conectada al bus del sistema. Estas memorias no forman parte del acelerador, ya que son memorias comunes compartidas con la etapa de estimación de movimiento fraccionario "fraccional motion estimation" (FME). Una vez que el multiplexor de memorias selecciona la RAM_{SW} indicada por el procesador local y esta es direccionada por su

AGU, el módulo selector elige los pixeles candidatos que alimentan la matriz de 16×16 PEs en un flujo de datos continuo.

La matriz calcula los valores de distorsión de 16 bloques 4×4 en cada ciclo de reloj. El árbol sumador utiliza los resultados SAD de las 16 sub-particiones 4×4 para calcular la distorsión de los bloques mayores, ahorrando el proceso de comparación para seis tamaños de bloques. El componente de tasa/distorsión R/D recibe los valores de distorsión de 41 bloques candidatos y les suma el coste de transmitir su vector de movimiento. El coste total de cada bloque candidato es comparado con el coste mínimo obtenido de los bloques similares previamente igualados.

Los resultados, al final de cada ciclo IME, son los MVs con resolución entera de las 41 particiones y sub-particiones candidatas con coste mínimo, dentro de la SW para el MB actual. El diagrama de flujo de la figura 4.15 documenta gráficamente el proceso anterior y la figura 4.16 muestra una rutina de programa tipo lenguaje C, que describe con 6 niveles de lazos anidados la operación IME-FSBM para bloques de tamaño M×N, donde c(i,j) y r(i+m, j+n) son los pixeles del bloque actual y del bloque candidato respectivamente.

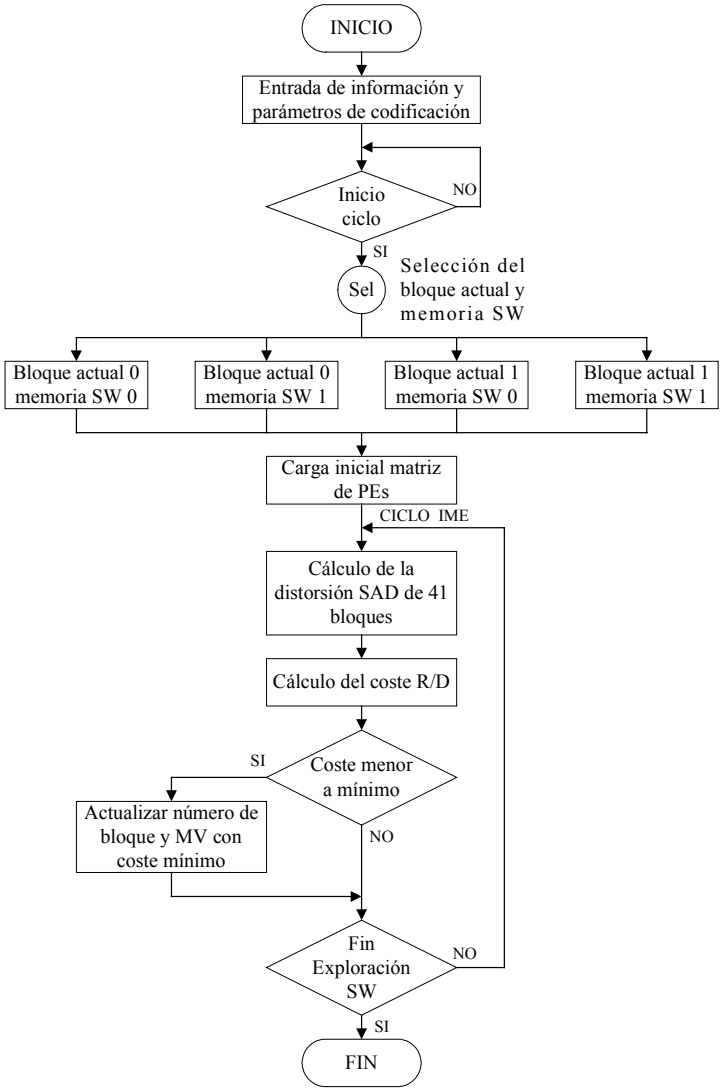


Figura 4.15. Secuencia del ciclo de operación IME-FSBM.

Conceptos importantes en el diseño de sistemas IME-FSBM son el tamaño de la SW y la estrategia de rastreo. La arquitectura que se presenta utiliza una ventana de búsqueda de tamaño (2Ph+16)×(2Pv+16) para un total de (2Ph+1)×(2Pv+1) bloques candidatos de 16×16 pixeles (Figura 4.17a), con una lógica de operación independiente de los valores de Ph y Pv. El método de exploración (espiral, barrido horizontal y vertical, serpenteo, etc.) es seleccionado en función del mecanismo de flujo de datos de una arquitectura, para reducir el nivel de computación y/o para aplicar algoritmos de terminación temprana. El diseño propuesto optimiza el flujo de datos y reduce el número de registros auxiliares con un mínimo nivel de control, gracias a la estrategia de exploración tipo serpenteo utilizada (Figura 4.17b). La exploración inicia con el MB candidato ubicado en la posición (0,0) (esquina superior izquierda de la SW) y continúa hasta terminar en el MB con posición (2Ph, 2Pv).

```
for(v=0; v<Nv/N; v++) { //Nivel del cuadro de video.
  for(h=0; h<Nh/M; h++) {
    MV (h, v) = (0,0);  //Inicialización del vector de movimiento.
    mcost_min(h, v) = maxvalue; //Valor inicial del coste de movimiento mínimo.
    for(m=0; m≤2Ph; m++)  {  //Nivel de la ventana de busqueda (exploración tipo serpenteo).
      for (n=0; n≤2Pv; n++){
        SAD(m, n) = 0;  //Inicialización del valor de distorsión.
        for(j=0; j<N; j++) {  //Nivel de bloque.
          for(i=0; i<M; i++) {
            SAD(m,n) = SAD(m,n) + |c(i,j) – r(i+m, j+n)| //Distorsión.
          }  //end for i.
        }  //end for j.
        R(m,n) = UVLC_TABLE[Motion_Vector_Difference_bits]  //Tasa.
        mcost(m,n) = SAD(m,n) + λ_SAD R(m,n) //Coste del movimiento.
        if(mcost(m,n) < mcost_min(h, v) {
          mcost_min(h, v) = mcost(m, n); //Nuevo coste de movimiento mínimo.
          MV(h, v) = (m, n);  //Mejor posición.
        }  //end if.
      }  //end for n.
    }  //end for m.
  }  //end for h.
}  //end for v.
```

Figura 4.16. Rutina de programa tipo lenguaje C con 6 niveles de lazos anidados, que describe la operación IME-FSBM para bloques de tamaño M×N.

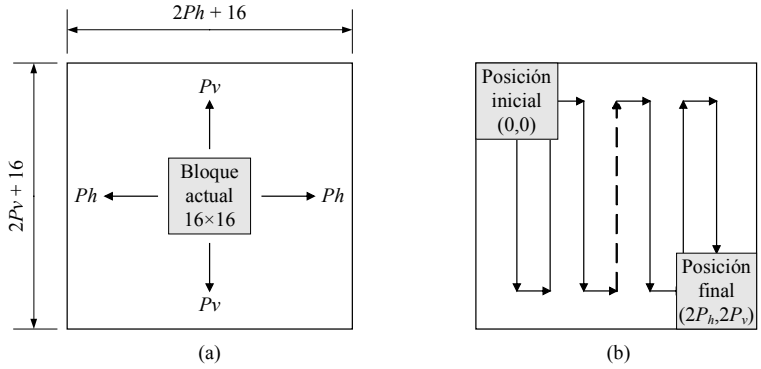


Figura 4.17. Ventana de búsqueda, a) rangos y desplazamientos de exploración vertical y horizontal, b) estrategia de rastreo tipo serpenteo.

Cabe destacar que, en el proceso de rastreo, se observa que el MB actual realiza el movimiento de serpenteo en una SW fija. En la realidad, el MB actual permanece fijo en la matriz de PEs y los pixeles de la SW son los que se mueven hacia arriba, hacia abajo y a la izquierda, emulando la estrategia de exploración indicada.

A continuación se describe la operación y diseño de cada módulo que compone la arquitectura propuesta.

4.3.3.1 Unidad de control.

La unidad de control es el elemento de interconexión entre los componentes de procesamiento y direccionamiento del IME y los procesadores local y del sistema. Diseñada como un periférico IP, la arquitectura IME se conecta a los buses de los procesadores local y del sistema para intercambiar información por medio de registros esclavos contenidos en la unidad de control, la cual también debe integrar las interfases y la lógica de operación global (Figura 4.18).

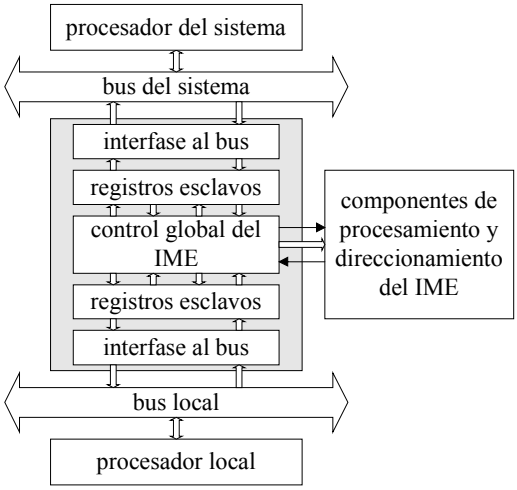


Figura 4.18. Diagrama a bloques de la unidad de control y su interconexión con los procesadores y buses local y del sistema, y con la lógica de procesamiento y direccionamiento del acelerador IME.

En cada ciclo de comparación, el IME debe utilizar los datos y variables de configuración que optimizan el proceso de búsqueda de los mejores bloques candidatos. Ya que el procesador del sistema tiene control sobre la operación global del codificador y recibe toda la información de su estado, es el encargado de definir y escribir en los registros del acelerador hardware los siguientes parámetros de ejecución del proceso de búsqueda y comparación:

* Los desplazamientos de exploración horizontal (Ph) y vertical (Pv) de la SW.

* La posición en RAM local del MB actual.

* La información para el cálculo del coste tasa/distorsión.

De forma similar, el procesador local, que controla el proceso de estimación IME y el acceso a los pixeles candidatos, indica el número de la memoria local donde se

encuentran los pixeles de la SW para el MB actual y señala el inicio del ciclo de búsqueda y comparación.

El tipo y capacidad de los procesadores y buses, está en función de las alternativas que proporcione la tecnología de realización. Su selección depende principalmente de que satisfagan el ancho de banda requerido por el estimador, bajo las condiciones de codificación de una aplicación específica (formato de video, velocidad en fps, tamaño del píxel, número de cuadros de referencia, dimensiones de la SW, etc.*)*.

Cada ciclo de exploración inicia con la validación de los parámetros y datos de operación y la señalización de arranque por parte de los procesadores. La actividad que abre el rastreo es la carga del MB actual y del primer candidato en la matriz de PEs. Una vez procesado el primer MB candidato, los siguientes MBs son cargados y comparados progresivamente hasta terminar de evaluar todos los MBs en la SW (Figura 4.19).

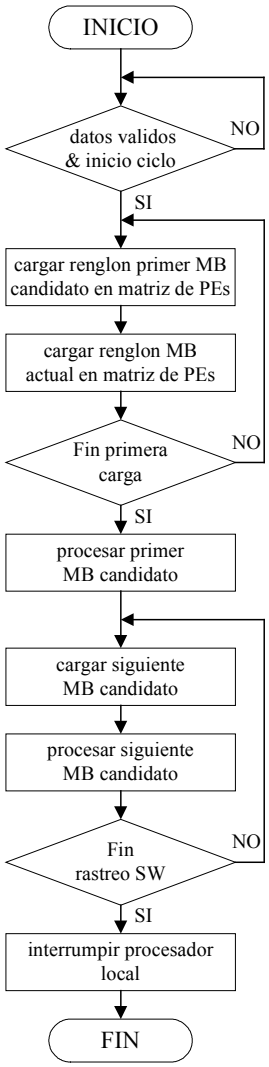
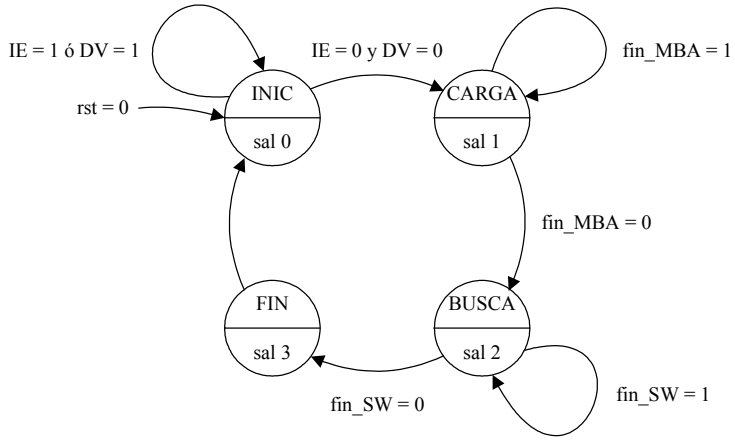


Figura 4.19. Diagrama de flujo de la exploración de la ventana de exploración.

La unidad de control interrumpe al procesador local para indicar el fin del ciclo y la disponibilidad del acelerador para iniciar otro. Cabe destacar que en el momento de la interrupción, los resultados de la estimación entera aun no están disponibles, ya que

por efectos de latencia, los 41 vectores de movimiento de los bloques de menor coste tardaran algunos ciclos de reloj en llegar a la etapa de estimación fraccionaria. Esto no condiciona el inicio de un nuevo ciclo de exploración, ya que la arquitectura *pipeline* del sistema permite operaciones simultaneas en diferentes etapas del estimador. El uso de la interrupción trata de reducir el tiempo muerto entre ciclos, ya que la exploración de la SW es la actividad crítica en los sistemas de estimación de movimiento.

Para su realización hardware, el control de la exploración de la SW se construye mediante una FSM tipo Moore de 4 estados y un contador de eventos (Figura 4.20). En el estado INIC se espera por las señales de validez de datos (DV) e inicio de operaciones (IE). En el estado CARGA se indica a las AGUs que carguen en la matriz de PEs los primeros MBs. Ya en el estado BUSCA, se espera por la señal de fin de la exploración, indicando a los componentes posteriores a la matriz la validez de los resultados SAD. Al final se genera la señal de interrupción en el estado FIN y la máquina se prepara para volver a repetir el ciclo.



Señales de salida	sal 0	sal 1	sal 2	sal 3
inicio_MBA	1	0	1	1
inicio_SW	1	0	0	1
SAD_valido	1	1	0	0
interrupción_borde	0	0	0	1

Figura 4.20. Diagrama de la máquina de estados del control de exploración de la SW.

4.3.3.2 Matriz de 16×16 PEs.

La matriz de procesamiento (Figura 4.21) es un grupo sistólico de 16×16 PEs y 16 registros auxiliares de 8 bits con los PEs agrupados en 16 bloques 4×4 (B_{PE}) y los registros auxiliares reunidos en 4 bloques 1×4 (B_{RA}). Los B_{PE} (Figura 4.22a) incluyen la lógica para el cómputo del SAD, con los que la matriz calcula la distorsión de 16 bloques 4×4 en un ciclo de reloj. Los B_{RA} (Figura 4.22b) siempre almacenan la columna de 16 pixeles a la derecha del MB candidato siendo procesado.

El elemento procesador (Figura 4.23a) selecciona con un multiplexor uno de los tres pixeles que llegan de los PEs adyacentes, de acuerdo al movimiento de datos que marca la estrategia de exploración. Dos registros respaldan los pixeles actual (R_C) y candidato (R_R), mientras que una unidad aritmética calcula su diferencia absoluta. El

resultado, respaldado en el registro (R_{AD}), alimenta al árbol sumador que procesa el SAD del bloque 4×4. Los pixeles registrados actual y candidato están disponibles para los PEs superior, inferior e izquierdo. Los registros auxiliares (Figura 4.23b) utilizan un multiplexor de 8 bits para seleccionar un píxel candidato del registro auxiliar superior o inferior. Después de ser almacenado, el píxel elegido está disponible para los elementos contiguos.

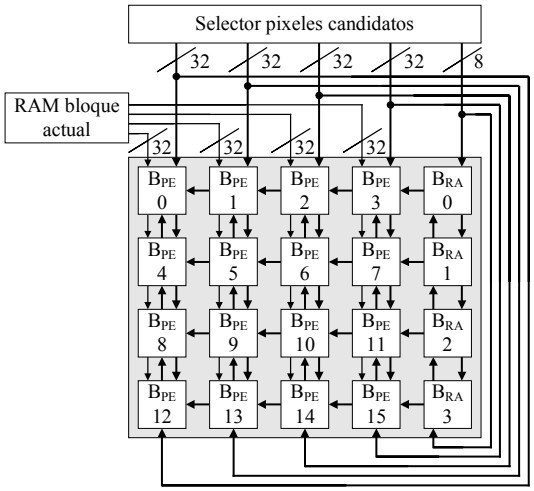


Figura 4.21. Matriz de 256 elementos procesadores y 16 registros auxiliares, agrupados en 16 bloques 4×4 de PEs (B_{PE}) y 4 bloques 1×4 de registros auxiliares (B_{RA}).

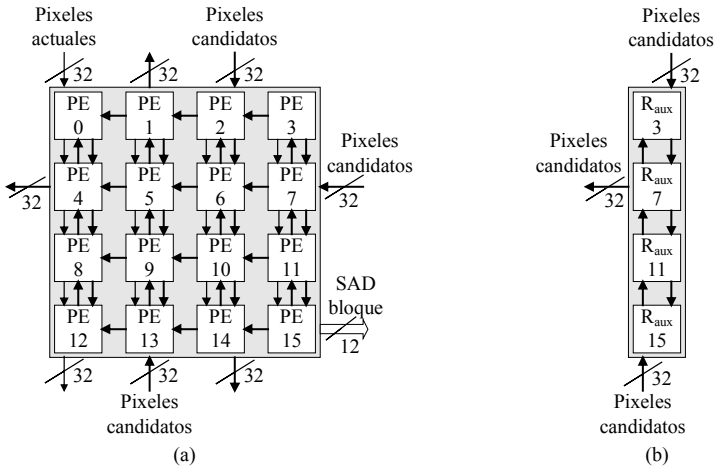


Figura 4.22. Bloques de la matriz de procesamiento: a) bloque B_{PE} de 4×4 PEs y b) bloque B_{RA} de 1×4 registros auxiliares.

Previo al inicio de un ciclo IME, la matriz de PEs recibe al MB actual, al primer MB candidato y a una columna auxiliar de pixeles candidatos desde las RAM locales (Figura 4.24). En un proceso de desplazamiento renglón por renglón, el MB actual se propaga por la matriz a través de los PEs superiores y sus registros R_C. Simultáneamente, la matriz es alimentada con los pixeles del primer MB candidato vía los PEs inferiores y sus registros R_R. Otra columna de pixeles candidatos es recibida por los registros R_{aux}. Después de una latencia de 16 ciclos de reloj, cada PE en la matriz entrega un resultado de distorsión AD.

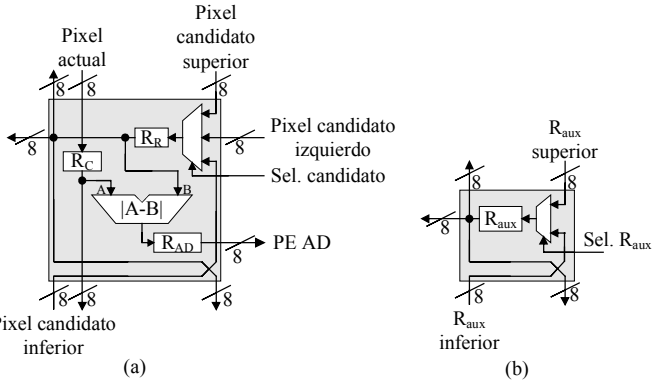


Figura 4.23. Estructura interna del a) elemento procesador y del b) registro auxiliar.

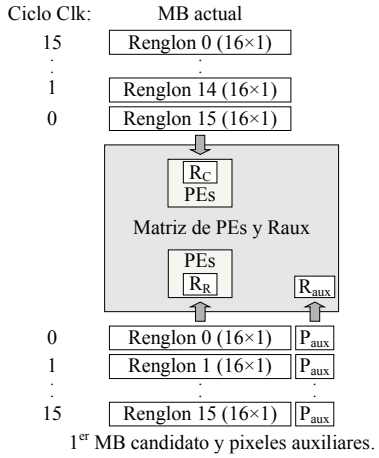


Figura 4.24. Carga inicial de la matriz de PEs con el MB actual, el primer MB candidato y los pixeles auxiliares.

Ya dentro del ciclo de procesamiento y de acuerdo a la exploración de la SW, en el siguiente ciclo de reloj el primer MB candidato es sustituido por el candidato inmediato inferior. Ya que ambos MBs comparten 15 renglones de pixeles (Figura 4.25), para completar el segundo MB solo se necesita realizar un desplazamiento hacia arriba del primer MB y alimentar a la matriz con el renglón faltante del segundo MB. En un total de 2Pv+16 ciclos de reloj, la matriz procesa la primera columna de MBs de la SW.

El procesamiento de la segunda columna inicia con un desplazamiento a la izquierda de los pixeles candidatos dentro de la matriz de PEs. Con esta operación, la columna de 1×16 pixeles de los registros auxiliares pasa a formar parte de la matriz de PEs. Ya que los registros auxiliares siempre almacenan la columna de pixeles de referencia a la derecha del MB candidato que procesa la matriz, esta operación completa el primer MB candidato de la segunda columna para que el procesamiento de la distorsión continúe sin ciclos falsos (Figura 4.26).

Los MBs candidatos de las siguientes columnas son procesados en la SW de una forma similar, sin ciclos extras en el cambio de columna. El módulo IME explora cualquier columna (no la primera) en (2Pv+1) ciclos de reloj y toda la SW de dimensiones (2Pv+16)×(2Ph+16) pixeles enteros en (2Pv+1)×(2Ph+1)+15 ciclos.

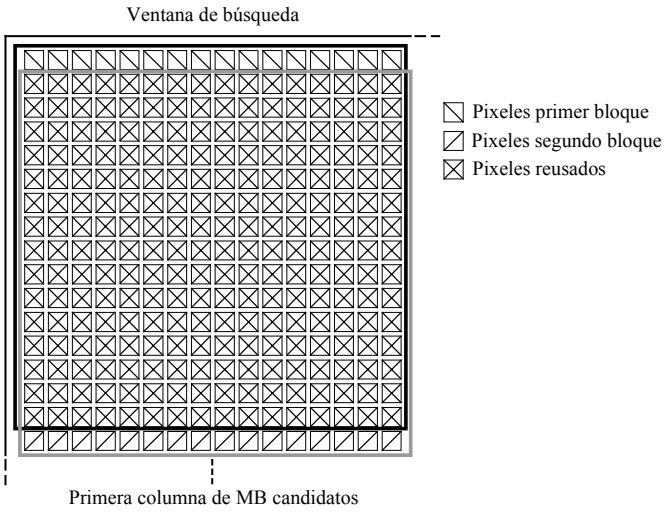


Figura 4.25. Pixeles de reutilización de los MBs candidatos de la primera columna de la SW.

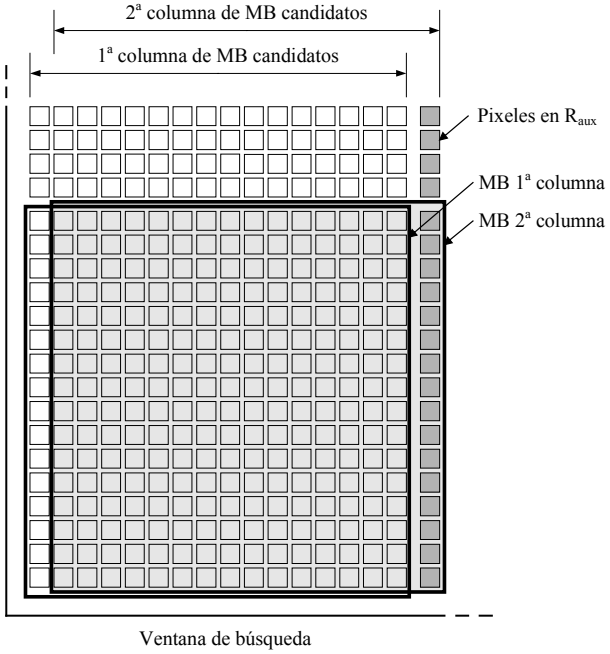


Figura 4.26. Cambio de la 1ª a la 2ª columna en la exploración de la SW, donde los pixeles en los R_{aux} completan el MB candidato de la 2ª columna, sin ciclos extras.

4.3.3.3 Memorias RAM_{BA} y RAM_{SW}.

Aun cuando las memorias RAM de los MBs actuales y candidatos no pertenecen directamente al módulo IME, ya que son compartidas por las arquitecturas de estimación de movimiento de pixeles enteros y fraccionarios, en este apartado se documentan sus características y estructura, porque su operación debe soportar la capacidad y tasa de datos que necesita el acelerador hardware.

Los 256 pixeles del MB actual se almacenan en una RAM local (RAM_{MBA}) que necesita acceder independientemente al bus del microprocesador del sistema y a la

matriz de PEs. Para la transferencia de los pixeles desde la memoria externa del cuadro de video actual, la memoria local requiere un puerto de 32 bits (bus del microprocesador de 32 bits), lo que permite una tasa de 4 pixeles por ciclo de reloj. Para la conexión a la matriz de procesamiento se precisa un puerto de 128 bits, que proporcione una tasa de 16 pixeles por ciclo de reloj. Para el cumplimiento de estas características de operación, se propone utilizar 4 bloques SRAM de 2 puertos, cada una con una organización de 128×8 en el puerto A (Tabla 4.3) y de 32×32 en el puerto B (Tabla 4.4). La memoria completa tiene una organización de 128×32 en el puerto A y de 32×128 en el puerto B, para almacenar 512 pixeles de 8 bits equivalentes a 2 MBs actuales (Figura 4.27).

Tabla 4.3. Organización de la memoria RAM_{MBA} (lado del puerto A), donde B es el número del bloque actual.

Dirección	Datos SRAM 3	Datos SRAM 2	Datos SRAM 1	Datos SRAM 0
(64×B)+0	pixel(0,3)	pixel (0,2)	pixel(0,1)	pixel(0,0)
(64×B)+1	pixel(0,7)	pixel (0,6)	pixel(0,5)	pixel(0,4)
(64×B)+2	pixel(0,11)	pixel (0,10)	pixel(0,9)	pixel(0,8)
(64×B)+3	pixel(0,15)	pixel (0,14)	pixel(0,13)	pixel(0,12)
(64×B)+4	pixel(1,3)	pixel (1,2)	pixel(1,1)	pixel(1,0)
:	:	:	:	:
(64×B)+59	pixel(14,15)	pixel (14,14)	pixel(14,13)	pixel(14,12)
(64×B)+60	pixel(15,3)	pixel (15,2)	pixel(15,1)	pixel(15,0)
(64×B)+61	pixel(15,7)	pixel (15,6)	pixel(15,5)	pixel(15,4)
(64×B)+62	pixel(15,11)	pixel (15,10)	pixel(15,9)	pixel(15,8)
(64×B)+63	pixel(15,15)	pixel (15,14)	pixel(15,13)	pixel(15,12)

Tabla 4.4. Organización de la memoria RAM_{MBA} (lado del puerto B), donde B es el número del bloque actual.

Dirección	Datos SRAM 3	Datos SRAM 2	Datos SRAM 1	Datos SRAM 0
(16×B)+0	P(0,15), P(0,14), P(0,13), P(0,12)	P(0,11), P(0,10), P(0,9), P(0,8)	P(0,7), P(0,6), P(0,5), P(0,4)	P(0,3), P(0,2), P(0,1), P(0,0)
(16×B)+1	P(1,15), P(1,14), P(1,13), P(1,12)	P(1,11), P(1,10), P(1,9), P(1,8)	P(1,7), P(1,6), P(1,5), P(1,4)	P(1,3), P(1,2), P(1,1), P(1,0)
(16×B)+2	P(2,15), P(2,14), P(2,13), P(2,12)	P(2,11), P(2,10), P(2,9), P(2,8)	P(2,7), P(2,6), P(2,5), P(2,4)	P(2,3), P(2,2), P(2,1), P(2,0)
(16×B)+3	P(3,15), P(3,14), P(3,13), P(3,12)	P(3,11), P(3,10), P(3,9), P(3,8)	P(3,7), P(3,6), P(3,5), P(3,4)	P(3,3), P(3,2), P(3,1), P(3,0)
(16×B)+4	P(4,15), P(4,14), P(4,13), P(4,12)	P(4,11), P(4,10), P(4,9), P(4,8)	P(4,7), P(4,6), P(4,5), P(4,4)	P(4,3), P(4,2), P(4,1), P(4,0)
(16×B)+5	P(5,15), P(5,14), P(5,13), P(5,12)	P(5,11), P(5,10), P(5,9), P(5,8)	P(5,7), P(5,6), P(5,5), P(5,4)	P(5,3), P(5,2), P(5,1), P(5,0)
(16×B)+6	P(6,15), P(6,14), P(6,13), P(6,12)	P(6,11), P(6,10), P(6,9), P(6,8)	P(6,7), P(6,6), P(6,5), P(6,4)	P(6,3), P(6,2), P(6,1), P(6,0)
(16×B)+7	P(7,15), P(7,14), P(7,13), P(7,12)	P(7,11), P(7,10), P(7,9), P(7,8)	P(7,7), P(7,6), P(7,5), P(7,4)	P(7,3), P(7,2), P(7,1), P(7,0)
(16×B)+8	P(8,15), P(8,14), P(8,13), P(8,12)	P(8,11), P(8,10), P(8,9), P(8,8)	P(8,7), P(8,6), P(8,5), P(8,4)	P(8,3), P(8,2), P(8,1), P(8,0)
(16×B)+9	P(9,15), P(9,14), P(9,13), P(9,12)	P(9,11), P(9,10), P(9,9), P(9,8)	P(9,7), P(9,6), P(9,5), P(9,4)	P(9,3), P(9,2), P(9,1), P(9,0)
(16×B)+10	P(10,15), P10(,14), P(10,13), P(10,12)	P(10,11), P(10,10), P(10,9), P(10,8)	P(10,7), P(10,6), P(10,5), P(10,4)	P(10,3), P(10,2), P(10,1), P(10,0)
(16×B)+11	P(11,15), P(11,14), P(11,13), P(11,12)	P(11,11), P(11,10), P(11,9), P(11,8)	P(11,7), P(11,6), P(11,5), P(11,4)	P(11,3), P(11,2), P(11,1), P(11,0)
(16×B)+12	P(12,15), P(12,14), P(12,13), P(12,12)	P(12,11), P(12,10), P(12,9), P(12,8)	P(12,7), P(12,6), P(12,5), P(12,4)	P(12,3), P(12,2), P(12,1), P(12,0)
(16×B)+13	P(13,15), P(13,14), P(13,13), P(13,12)	P(13,11), P(13,10), P(13,9), P(13,8)	P(13,7), P(13,6), P(13,5), P(13,4)	P(13,3), P(13,2), P(13,1), P(13,0)
(16×B)+14	P(14,15), P(14,14), P(14,13), P(14,12)	P(14,11), P(14,10), P(14,9), P(14,8)	P(14,7), P(14,6), P(14,5), P(14,4)	P(14,3), P(14,2), P(14,1), P(14,0)
(16×B)+15	P(15,15), P(15,14), P(15,13), P(15,12)	P(15,11), P(15,10), P(15,9), P(15,8)	P(15,7), P(15,6), P(15,5), P(15,4)	P(15,3), P(15,2), P(15,1), P(15,0)

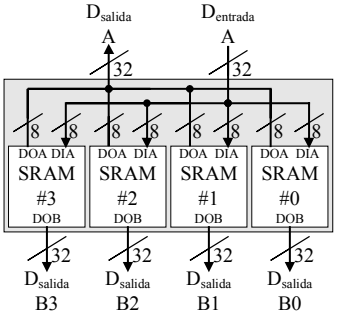


Figura 4.27. Estructura de la memoria RAM del MB actual, con cuatro SRAMs de 2 puertos.

La arquitectura propuesta incluye dos memorias para almacenar localmente los MBs candidatos (RAM_{SW} #0 y RAM_{SW} #1), cada una estructurada como 4 bloques SRAMs de 2 puertos, A y B, con una organización de ((2Ph+16)(2Pv+16)/16)×32 en ambos puertos, en función de los desplazamientos de exploración (Ph, Pv). La organización de la memoria completa es de ((2Ph+16)(2Pv+16)/4)×32 en el puerto A y de ((2Ph+16)(2Pv+16)/16)×128 en el puerto B. Las expresiones anteriores indican que la capacidad de almacenamiento de cada RAM_{SW} es función del tamaño de la SW (4.4).

Capacidad en bytes de la $RAM_{SW} \geq (2Ph+16)(2Pv+16)$ (4.4)

El diseño de las RAM_{SW} esta enfocado a la obtención de su máximo ancho en bytes, conectando el puerto A al bus local de 32 ó 64 bits y administrando ambos puertos A y B para transferir 32 pixeles candidatos hacia el selector de pixeles, vía los multiplexores de selección de memorias (Figura 4.28). La tabla 4.5 muestra un ejemplo de la organización y distribución de los datos en las cuatro SRAMs para Ph = 32.

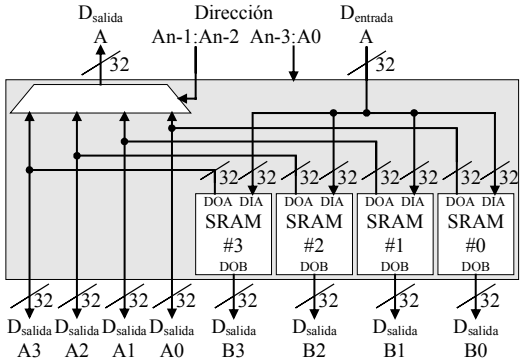


Figura 4.28. Estructura de la memoria RAM_{SW}.

Tabla 4.5. Ejemplo de organización y distribución de los datos en memorias RAM_{SW} para las primeras 16 localidades y un rango de exploración horizontal Ph = 32.

Dirección	Datos SRAM 3	Datos SRAM 2	Datos SRAM 1	Datos SRAM 0
0	P(0,15), P(0,14), P(0,13), P(0,12)	P(0,11), P(0,10), P(0,9), P(0,8)	P(0,7), P(0,6), P(0,5), P(0,4)	P(0,3), P(0,2), P(0,1), P(0,0)
1	P(0,31), P(0,30), P(0,29), P(0,28)	P(0,27), P(0,26), P(0,25), P(0,24)	P(0,23), P(0,22), P(0,21), P(0,20)	P(0,19), P(0,18), P(0,17), P(0,16)
2	P(0,47), P(0,46), P(0,45), P(0,44)	P(0,43), P(0,42), P(0,41), P(0,40)	P(0,39), P(0,38), P(0,37), P(0,36)	P(0,35), P(0,34), P(0,33), P(0,32)
3	P(0,63), P(0,62), P(0,61), P(0,60)	P(0,59), P(0,58), P(0,57), P(0,56)	P(0,55), P(0,54), P(0,53), P(0,52)	P(0,51), P(0,50), P(0,49), P(0,48)
4	P(0,79), P(0,78), P(0,77), P(0,76)	P(0,75), P(0,74), P(0,73), P(0,72)	P(0,71), P(0,70), P(0,69), P(0,68)	P(0,67), P(0,66), P(0,65), P(0,64)
5	P(1,15), P(1,14), P(1,13), P(1,12)	P(1,11), P(1,10), P(1,9), P(1,8)	P(1,7), P(1,6), P(1,5), P(1,4)	P(1,3), P(1,2), P(1,1), P(1,0)
6	P(1,31), P(1,30), P(1,29), P(1,28)	P(1,27), P(1,26), P(1,25), P(1,24)	P(1,23), P(1,22), P(1,21), P(1,20)	P(1,19), P(1,18), P(1,17), P(1,16)
7	P(1,47), P(1,46), P(1,45), P(1,44)	P(1,43), P(1,42), P(1,41), P(1,40)	P(1,39), P(1,38), P(1,37), P(1,36)	P(1,35), P(1,34), P(1,33), P(1,32)
8	P(1,63), P(1,62), P(1,61), P(1,60)	P(1,59), P(1,58), P(1,57), P(1,56)	P(1,55), P(1,54), P(1,53), P(1,52)	P(1,51), P(1,50), P(1,49), P(1,48)
9	P(1,79), P(1,78), P(1,77), P(1,76)	P(1,75), P(1,74), P(1,73), P(1,72)	P(1,71), P(1,70), P(1,69), P(1,68)	P(1,67), P(1,66), P(1,65), P(1,64)
10	P(2,15), P(2,14), P(2,13), P(2,12)	P(2,11), P(2,10), P(2,9), P(2,8)	P(2,7), P(2,6), P(2,5), P(2,4)	P(2,3), P(2,2), P(2,1), P(2,0)
11	P(2,31), P(2,30), P(2,29), P(2,28)	P(2,27), P(2,26), P(2,25), P(2,24)	P(2,23), P(2,22), P(2,21), P(2,20)	P(2,19), P(2,18), P(2,17), P(2,16)
12	P(2,47), P(2,46), P(2,45), P(2,44)	P(2,43), P(2,42), P(2,41), P(2,40)	P(2,39), P(2,38), P(2,37), P(2,36)	P(2,35), P(2,34), P(2,33), P(2,32)
13	P(2,63), P(2,62), P(2,61), P(2,60)	P(2,59), P(2,58), P(2,57), P(2,56)	P(2,55), P(2,54), P(2,53), P(2,52)	P(2,51), P(2,50), P(2,49), P(2,48)
14	P(2,79), P(2,78), P(2,77), P(2,76)	P(2,75), P(2,74), P(2,73), P(2,72)	P(2,71), P(2,70), P(2,69), P(2,68)	P(2,67), P(2,66), P(2,65), P(2,64)
15	P(3,15), P(3,14), P(3,13), P(3,12)	P(3,11), P(3,10), P(3,9), P(3,8)	P(3,7), P(3,6), P(3,5), P(3,4)	P(3,3), P(3,2), P(3,1), P(3,0)
:	:	:	:	:

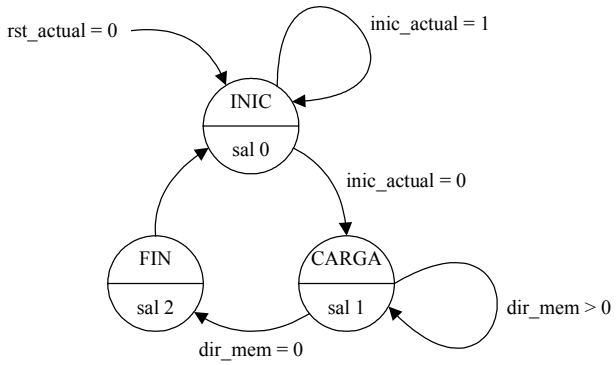
Al inicio del ciclo IME, la RAM_{SW} #0 recibe los pixeles de la SW desde la memoria externa que almacena los cuadros de referencia. Los datos son organizados en la memoria local de izquierda a derecha y de arriba hacia abajo. Mientras el módulo lee y procesa estos pixeles, el microprocesador local almacena en la RAM_{SW} #1 los pixeles de la siguiente SW. Esta operación ayuda a evitar cuellos de botella en ciclos consecutivos. Por la misma razón, el tiempo de escritura de la RAM_{SW} en ciclos de reloj necesita ser menor al tiempo de exploración de la SW (4.5).

Tiempo escritura RAM_{SW} = (2Ph+16) (2Pv+16)/4 < (2Ph+1) (2Ph+1)+15 ciclos (4.5)

4.3.3.4. Unidades generadoras de direcciones.

El módulo IME incluye 2 unidades generadoras de direcciones (AGU), la AGU_{MBA} de la memoria del MB actual y la AGU_{SW} de las memorias que almacenan los MB candidatos. La AGU_{MBA} tiene la función de controlar el bus de direcciones del puerto B de la memoria RAM_{MBA}, para leer 16 pixeles pertenecientes al mismo renglón del MB actual y alimentarlos a la matriz de SW. La AGU_{SW} administra el bus de direcciones de los puertos A y B de las memorias RAM_{SW} para leer 32 pixeles de un mismo renglón de la SW, de los cuales son seleccionados los 17 pixeles que necesita la matriz de PEs en cada ciclo de reloj.

La AGU_{MBA} asume que los 256 pixeles de un MB actual son leídos en modo de exploración en línea [Vos89], del renglón 15 al renglón 0. Ya que el orden de direcciones es fijo y solo depende de un punto de inicio, su arquitectura se clasifica como una secuencia de direcciones predefinida con realización incremental [Kuhn99], alrededor de una FSM Moore de tres estados (Figura 4.29) y un contador de direcciones. La FSM comienza su operación en el estado INIC esperando la señal de inicio *inic_actual*. En el estado CARGA, la FSM inicializa y habilita al contador de direcciones *dir_mem* a '1111' (contador binario descendente de 4 bits), habilita la lectura de la RAM_{MBA} (señal *hab_mem_actual*) y activa la señal *cargar_Rc* para cargar cada renglón de pixeles actuales en la matriz de PEs. Una vez que termina la lectura del MB actual, cuando el contador llega al valor '0000', 16 ciclos de reloj después, la máquina de estados pasa al estado FIN donde se activa la señal *fin_actual.*

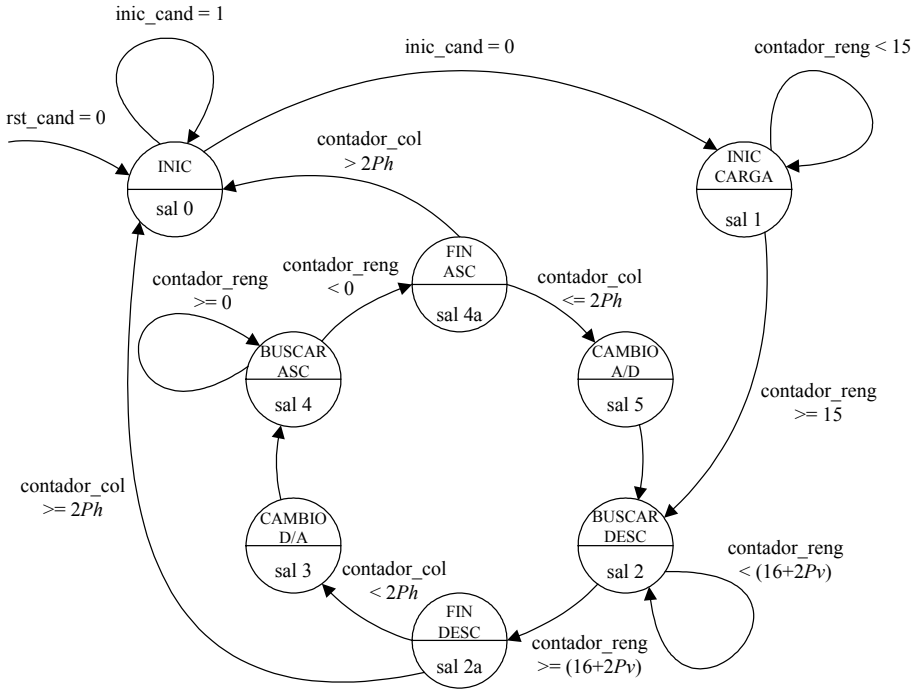


Señales de salida	sal 0	sal 1	sal 2
hab_mem_actual	0	1	0
hab_dir_mem_actual	1	0	1
cargar_Rc	1	0	0
fin_actual	1	1	0

Figura 4.29. Diagrama de flujo de la FSM Moore que define la operación de la AGU de la memoria RAM_{MBA}.

La AGU_{SW} controla el proceso de rastreo tipo serpenteo de la SW a partir de un modo de exploración en línea y una secuencia parametrizable de direcciones con realización incremental. Diseñada como una FSM Moore de 8 estados (Figura 4.30) y dos contadores binarios principales (contadores de renglones y columnas) y cuatro auxiliares (contadores de desplazamiento, bloques, *x* y *y*), la AGU_{SW} recibe de la unidad

de control los rangos de exploración horizontal y vertical (Ph, Pv) e inicia su operación cuando se activa la señal *inic_cand* y termina cuando el contador de columnas es igual o mayor que 2Ph.



Señales de salida	sal 0	sal 1	sal 2/sal 2a	sal 3	sal 4/sal 4a	sal 5
cargar_R_R_cand	1	0	0	0	0	0
cargar_R_{aux}_cand	1	0	0	1	0	1
sel_cand	00	10	10	01	00	01
sel_aux_cand	0	1	1	0	0	0
fin_cand	0	1	1	1	1	0

Figura 4.30. Diagrama de flujo de la FSM Moore (FSM_{SW}) que define la operación de la AGU de las memorias RAM_{SW}.

Las especificaciones de los contadores binarios son:

a) Contador de renglones (*contador_reng*). Contador ascendente/descendente que apoya la exploración vertical de la SW. En el sentido de exploración vertical descendente (estado BUSCAR_DESC), en cada ciclo de reloj, la matriz de procesamiento recibe un renglón de pixeles candidatos en sus PEs inferiores y el contador se incrementa a partir de un valor inicial 0 para terminar cuando llega al valor (16+2Pv). En el sentido de exploración vertical ascendente (estado BUSCAR_ASC), en cada ciclo de reloj, la matriz de procesamiento recibe un renglón de pixeles candidatos en sus PEs superiores y el contador se decrementa a partir de un valor inicial 2Pv para terminar cuando alcanza el valor de 0. Este contador forma parte de la ecuación de direcciones.

b) Contador de columnas (*contador_col*). Contador ascendente que con su valor indica el avance en la exploración de las columnas de pixeles de la SW. Su cuenta inicia en 0 y termina en el valor 2Ph, indicando el final del ciclo.

c) Contador de desplazamiento (*contador_corr*). Contador ascendente que controla al módulo selector de pixeles. Con un valor inicial de 0, este contador se incrementa en cada cambio de exploración de columnas hasta alcanzar un valor máximo de 15, después del cual regresa a su valor inicial para repetir su operación.

d) Contador de bloques (*contador_bloques*). Contador ascendente que con su valor indica la columna de MBs a explorar. Este contador se incrementa de 0 a ((16+2Ph)/16)−1 cada vez que se procesan 16 columnas de pixeles de la SW. Junto con el contador de renglones, este contador define la ecuación de direcciones.

e) Contador *x* (*contador_x*). Contador ascendente que indica la coordenada *x* con respecto al origen de la SW (punto superior izquierdo) del MB candidato que esta siendo procesado. Su valor, que se deriva del contador de columnas, se envía al componente de tasa/distorsión para calcular el coste del MV.

f) Contador *y* (*contador_y*). Contador ascendente que indica la coordenada *y* con respecto al origen de la SW (punto superior izquierdo) del MB candidato que esta siendo procesado. Su valor, que se deriva del contador de renglones, también se envía al componente de tasa/distorsión para calcular el coste del MV.

En el estado INIC, la FSM_{SW} inicializa los contadores y espera la señal *inic_cand* para comenzar su operación. En el estado INIC_CARGA, se realiza la carga de los 16 renglones del primer MB candidato en la matriz de PEs, con el apoyo del contador de renglones. Los estados BUSCAR_DESC y BUSCAR_ASC controlan el barrido vertical (descendente y ascendente respectivamente) de la SW, en donde se procesan 2Pv+1 MBs candidatos por columna. El rastreo de cada columna de MBs termina en los estados FIN_DESC y FIN_ASC, donde se determina el final de la exploración de la SW. Si el valor del contador de columnas alcanza un valor máximo, se salta al estado INIC y se activa la señal *fin_cand*. En otra situación, al terminar el procesamiento del último MB de cada columna, se realiza el cambio inmediato a la siguiente columna en los estados CAMBIO_D_A y CAMBIO_A_D.

Para la organización y distribución de los datos en la memorias RAM_{SW}, la ecuación que calcula la dirección de los pixeles candidatos es una función del tamaño horizontal de la SW (2Ph + 16) y de los contadores de renglones y bloques (4.6). El rango vertical Pv puede tomar un valor mayor a 0, pero el rango horizontal Ph solo puede tomar valores 8x, donde x = 1, 2, 3, 4..., debido a que la división (2Ph + 16) / 16 debe tener un resultado entero (restricción de diseño). Los valores de Ph y Pv estaran limitados por la tecnología y recursos de realización. La tabla 4.6 muestra un ejemplo de aplicación de la ecuación de direcciones para los datos de la tabla 4.5. Cuando el contador de renglones toma valores de 0 a 15, el contador de bloques es igual a 0 y (2Ph + 16) / 16 = 5, para Ph = 32, lo que resulta en la lectura del primer MB de la SW.

$$\text{dirección } RAM_{SW} = (\text{contador\_reng} \times (2Ph + 16) / 16) + \text{contador\_bloques} \quad (4.6)$$

4.3.3.5 Selector de memorias y selector de pixeles.

Para la transferencia de los pixeles candidatos desde las memorias RAM_{SW} hasta la matriz de procesamiento se cuenta con dos componentes: el selector de memorias y el selector de pixeles (Figura 4.31).

Tabla 4.6. Aplicación de la ecuación de direcciones para la lectura del primer MB de la SW.

Dirección	Datos SRAM 3	Datos SRAM 2	Datos SRAM 1	Datos SRAM 0
0×5+0	P(0,15), P(0,14), P(0,13), P(0,12)	P(0,11), P(0,10), P(0,9), P(0,8)	P(0,7), P(0,6), P(0,5), P(0,4)	P(0,3), P(0,2), P(0,1), P(0,0)
1×5+0	P(1,15), P(1,14), P(1,13), P(1,12)	P(1,11), P(1,10), P(1,9), P(1,8)	P(1,7), P(1,6), P(1,5), P(1,4)	P(1,3), P(1,2), P(1,1), P(1,0)
2×5+0	P(2,15), P(2,14), P(2,13), P(2,12)	P(2,11), P(2,10), P(2,9), P(2,8)	P(2,7), P(2,6), P(2,5), P(2,4)	P(2,3), P(2,2), P(2,1), P(2,0)
3×5+0	P(3,15), P(3,14), P(3,13), P(3,12)	P(3,11), P(3,10), P(3,9), P(3,8)	P(3,7), P(3,6), P(3,5), P(3,4)	P(3,3), P(3,2), P(3,1), P(3,0)
4×5+0	P(4,15), P(4,14), P(4,13), P(4,12)	P(4,11), P(4,10), P(4,9), P(4,8)	P(4,7), P(4,6), P(4,5), P(4,4)	P(4,3), P(4,2), P(4,1), P(4,0)
5×5+0	P(5,15), P(5,14), P(5,13), P(5,12)	P(5,11), P(5,10), P(5,9), P(5,8)	P(5,7), P(5,6), P(5,5), P(5,4)	P(5,3), P(5,2), P(5,1), P(5,0)
6×5+0	P(6,15), P(6,14), P(6,13), P(6,12)	P(6,11), P(6,10), P(6,9), P(6,8)	P(6,7), P(6,6), P(6,5), P(6,4)	P(6,3), P(6,2), P(6,1), P(6,0)
7×5+0	P(7,15), P(7,14), P(7,13), P(7,12)	P(7,11), P(7,10), P(7,9), P(7,8)	P(7,7), P(7,6), P(7,5), P(7,4)	P(7,3), P(7,2), P(7,1), P(7,0)
8×5+0	P(8,15), P(8,14), P(8,13), P(8,12)	P(8,11), P(8,10), P(8,9), P(8,8)	P(8,7), P(8,6), P(8,5), P(8,4)	P(8,3), P(8,2), P(8,1), P(8,0)
9×5+0	P(9,15), P(9,14), P(9,13), P(9,12)	P(9,11), P(9,10), P(9,9), P(9,8)	P(9,7), P(9,6), P(9,5), P(9,4)	P(9,3), P(9,2), P(9,1), P(9,0)
10×5+0	P(10,15),P(10,14)P(10,13) P(10,12)	P(10,11), P(10,10), P10,9), P(10,8)	P(10,7), P(10,6), P(10,5), P(10,4)	P(10,3), P(10,2), P(10,1), P(10,0)
11×5+0	P(11,15),P(11,14)P(11,13),P(11,12)	P(11,11), P(11,10), P(11,9), P(11,8)	P(11,7), P(11,6), P(11,5), P(11,4)	P(11,3), P(11,2), P(11,1), P(11,0)
12×5+0	P(12,15) P(12,14)P(12,13) P(12,12)	P(12,11), P(12,10), P(12,9), P(12,8)	P(12,7), P(12,6), P(12,5), P(12,4)	P(12,3), P(12,2), P(12,1), P(12,0)
13×5+0	P(13,15) P(13,14)P(13,13) P(13,12)	P(13,11), P(13,10), P(13,9), P(13,8)	P(13,7), P(13,6), P(13,5), P(13,4)	P(13,3), P(13,2), P(13,1), P(13,0)
14×5+0	P(14,15) P(14,14)P(14,13) P(14,12)	P(14,11), P(14,10), P(14,9), P(14,8)	P(14,7), P(14,6), P(14,5), P(14,4)	P(14,3), P(14,2), P(14,1), P(14,0)
15×5+0	P(15,15) P(15,14)P(15,13) P(15,12)	P(15,11), P(15,10), P(15,9), P(15,8)	P(15,7), P(15,6), P(15,5), P(15,4)	P(15,3), P(15,2), P(15,1), P(15,0)
:	:	:	:	:

El selector de memorias elije los datos de salida de una de las RAM_{SW} (#0 ó #1) en función de la información que recibe del procesador local. Está compuesto de 8 multiplexores de 32 bits, de dos entradas y una salida, que seleccionan los bloques SRAM de una u otra memoria.

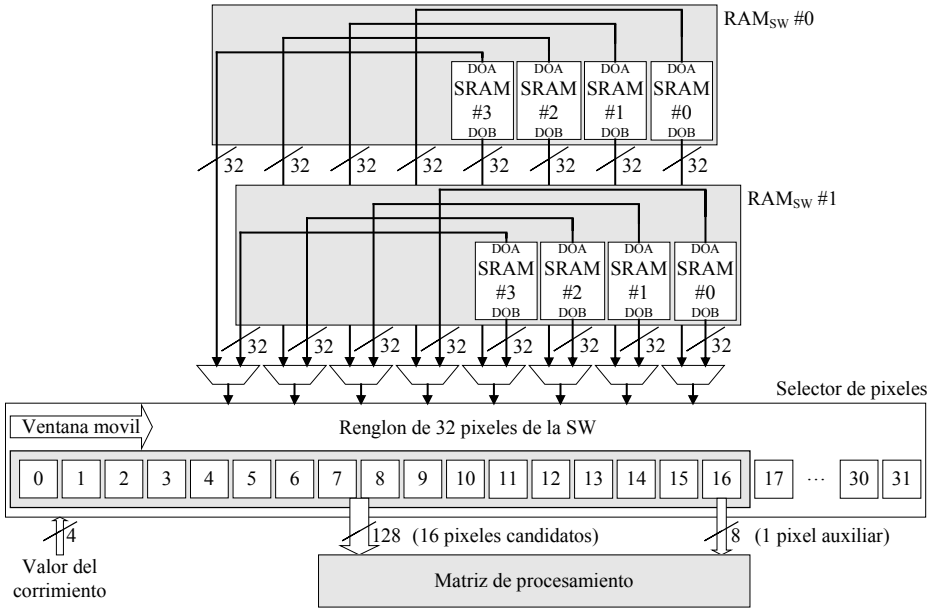


Figura 4.31. Componentes para la transferencia de los pixeles candidatos desde las memorias RAM_{SW} hasta la matriz de procesamiento: selector de memorias (multiplexores 2 a 1) y selector de pixeles.

El selector de pixeles tiene la función de simplificar el direccionamiento de los pixeles candidatos que necesita la matriz de procesamiento en cada ciclo de reloj, al permitir que el acceso a la memoria RAM_{SW} se alinee con las localidades de 32 bits de los bloques SRAM que la forman, y que del lado de la matriz de procesamiento se tengan accesos alineados con bytes. Su operación inicia al recibir 32 pixeles de la memoria RAM_{SW} seleccionada, 16 del puerto A y 16 del puerto B, los cuales pertenecen al mismo renglón de la SW (Figura 4.32). De estos datos, el componente selecciona los 17 pixeles que demanda la matriz de procesamiento utilizando una ventana móvil, cuyo desplazamiento es especificado por el valor de un contador controlado por la AGU_{SW} (*contador_corr*).

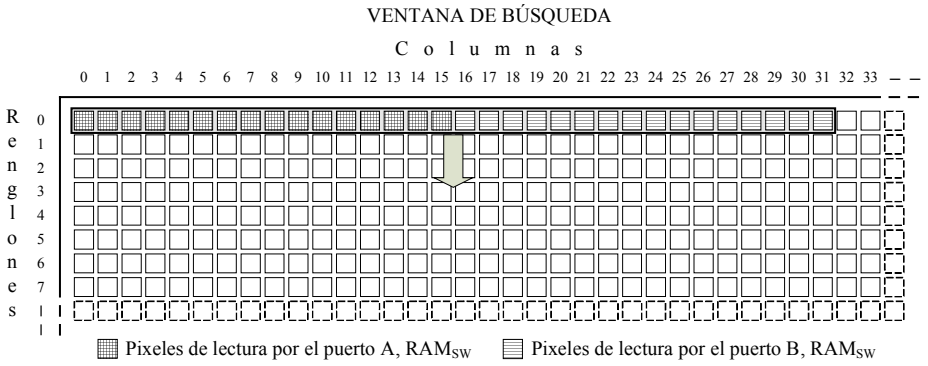


Figura 4.32. Ejemplo de un renglón de 32 pixeles de la ventana de búsqueda que recibe el selector de pixeles y dirección de lectura de los siguientes renglones.

El orden de lectura de los renglones de 32 pixeles se realiza en forma vertical descendente y ascendente, con desplazamientos horizontales cada 16 columnas, de acuerdo a la estrategia de exploración tipo serpenteo aquí utilizada. Al terminar de explorar una columna, el *contador_corr* se incrementa en uno para que la ventana móvil realice un desplazamiento a la derecha y seleccione los 17 pixeles de la siguiente columna de MBs candidatos. Cuando se terminan de procesar las primeras 16 columnas, el contador de desplazamiento se inicializa a cero y el renglón de pixeles se recorre 16 posiciones a la derecha para incluir las siguientes 16 columnas de pixeles, en un proceso que se repite en todo el rastreo de la SW.

4.3.3.6 Árbol sumador.

En el proceso de estimación de movimiento con tamaño de bloque variable del H.264/AVC, cada MB se particiona en un total de 41 bloques y sub-bloques (1-16×16, 2-16×8, 2-8×16, 4-8×8, 8-8×4, 8-4×8 y 16-4×4), lo que origina un alto coste computacional en el cálculo de los valores de distorsión SAD. Para minimizar este coste, el diseño propuesto utiliza la técnica que se aplica en la función C *FastFullPelBlockMotionSearch* del software de referencia [JM1106], donde se calcula la distorsión para el menor tamaño de bloque (4×4) y se obtiene la distorsión de los 6 tamaños de bloques restantes, reusando los 16 valores de distorsión de los bloques 4×4.

Cada PE de la matriz de procesamiento calcula la diferencia absoluta (AD) entre el píxel actual y el candidato. Estos valores de AD son procesados en los 16 bloques de 4×4 PEs, utilizando un árbol sumador para obtener la distorsión SAD_Bx de cada bloque (Figura 4.33). En una operación paralela, los 16 SAD_Bx son procesados nuevamente por un árbol sumador para obtener la distorsión de los siguientes bloques, en un orden de menor a mayor tamaño (Figura 4.34). Las figuras no incluyen la segmentación para reducir los niveles lógicos, ni los elementos de retardo para sincronizar los resultados. El número y posición de los registros correspondientes dependerá de la tecnología de realización.

4.3.3.7 Módulo de tasa/distorsión.

El enfoque mas ampliamente aceptado en los recientes estándares para el control operacional del codificador de video es el uso de técnicas de optimización de Lagrange,

las cuales tratan de contestar la pregunta de ¿qué parte de la señal de video debe ser codificada usando que método y que valor de parámetros? [Sullivan98]. En el caso de la estimación de movimiento en el estándar H.264/AVC, el software de referencia aplica esta técnica minimizando la función de coste $J_{MOVIMIENTO}$ (4.7), en la cual la distorsión SAD(MV) del bloque candidato con vector de movimiento MV se suma al coste de la tasa λ_{SAD} R(MV – MVp), la cual consiste en el coste de transmitir los bits asociados con la diferencia entre el vector de movimiento y su predicción MVp, siendo λ_{SAD} el multiplicador de Lagrange para la estimación de movimiento [Wiegand03b].

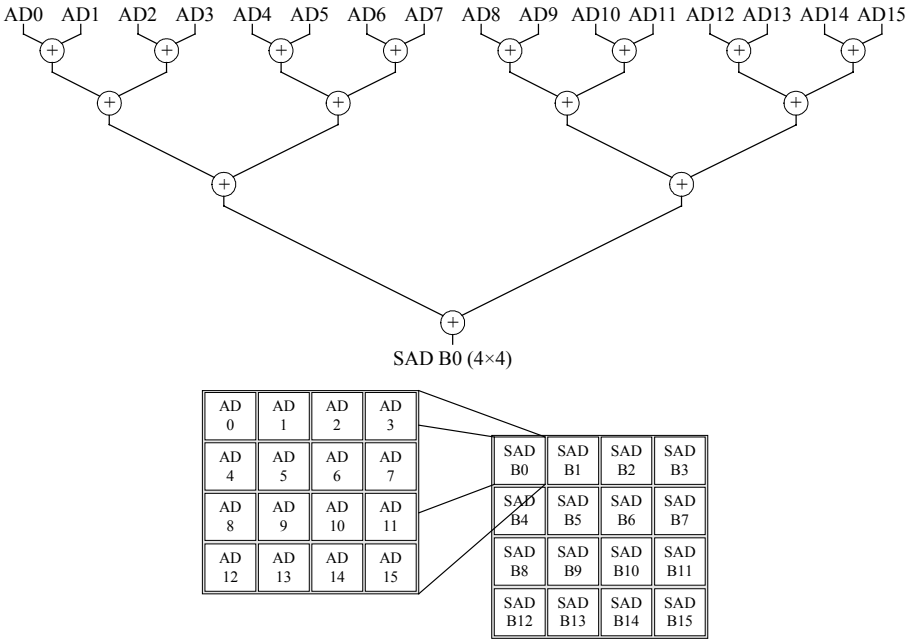


Figura 4.33. Árbol sumador para obtener la distorsión SAD del bloque 0 (4×4).

$$J_{MOVIMIENTO}(MV \mid \lambda_{SAD}, MVp) = SAD(MV) + \lambda_{SAD}\, R(MV - MVp) \qquad (4.7)$$

En cada ciclo de reloj durante la exploración de la SW, el árbol sumador calcula los valores de distorsión SAD de los bloques y sub-bloques del último MB candidato evaluado y la AGU_{SW} indica las coordenadas (x,y) de este MB con respecto al origen de la SW (punto superior izquierdo). Junto con esta información, el módulo R/D utiliza los siguientes datos que se recibe del microprocesador del sistema, para encontrar el vector de movimiento que minimiza el criterio de coste de cada bloque y sub-bloque:

* Las coordenadas del MB actual (pic_pix_x, pic_pix_y) en unidades de pixeles enteros con respecto al origen del cuadro de video (enteros sin signo).

* Las coordenadas de la SW (search_x, search_y) en unidades de pixeles enteros con respecto al origen del cuadro de video (enteros sin signo).

* Los componentes *x* y *y* de la predicción del vector de movimiento MVp (pred_mv_x, pred_mv_y) en unidades de ¼ de píxel (enteros con signo)

* El multiplicador de Lagrange λ_{SAD} (punto flotante).

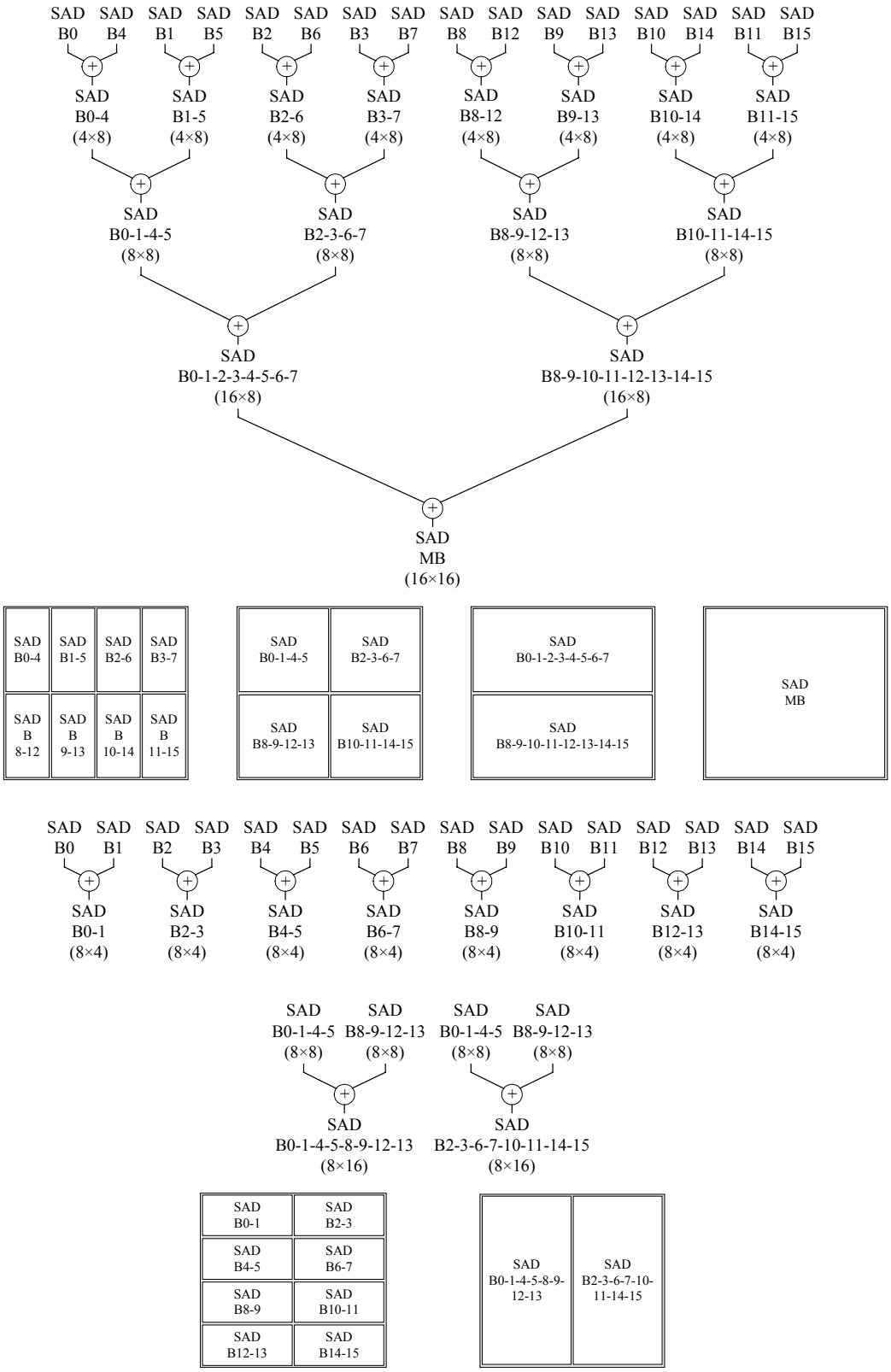


Figura 4.34. Procesamiento de la distorsión de los bloques 4×4 para obtener el SAD de los siguientes 6 tamaños de bloque, para un total de 41 bloques y sub-bloques.

Para facilitar el procesamiento, en [JM1106] se define un equivalente entero del valor en punto flotante λ_{SAD} (4.8), donde LAMBDA_ACCURACY_BITS (LAB) es una constante entera, con un valor por defecto de 16, que define la resolución de λ_{SAD}. El valor de $\lambda_{SADinteger}$ es procesado por el microprocesador del sistema y enviado al estimador, como un dato entero de 24 bits.

$$\lambda_{SADinteger} = ((int)((double)(1 << LAMBDA\_ACCURACY\_BITS) * \lambda_{SAD} + 0.5)) \quad (4.8)$$

El procedimiento para el cálculo de $J_{MOVIMIENTO}$ es el siguiente:

a) Calcular las coordenadas del MB candidato (cand_x, cand_y) con respecto al origen del cuadro en unidades de pixeles (4.9). Los resultados son siempre positivos.

cand_x = coordenada x de la SW + valor x estrategia de exploración (4.9)
= search_x + exploracion_x
cand_y = coordenada y de la SW + valor y estrategia de exploración
= search_y + exploracion_y

b) Determinar el punto final de la predicción del vector de movimiento (MVp) del MB (pred_x, pred_y) con respecto al origen de la trama en unidades de ¼ de píxel (4.10). Los resultados pueden ser positivos o negativos de acuerdo a los valores con signo de pred_mv_x y pred_mv_y.

pred_x = (pic_pix_x << 2) + pred_mv_x (4.10)
pred_y = (pic_pix_y << 2) + pred_mv_y

c) Calcular la diferencia absoluta |MV – MVp| para ambos ejes (bits_x, bits_y) en unidades de ¼ de píxel (4.11). El valor máximo de los resultados depende de los desplazamientos de exploración P_H y P_V (Tabla 4.7).

bits_x = |(cand_x << 2) – pred_x| (4.11)
bits_y = |(cand_y << 2) – pred_y|

d) Estimar la tasa *R* a partir del código de longitud variable universal (UVLC), indexando el vector mvbits[] (Tabla 4.8) con los valores de bits_x y bits_y (4.12).

R = mvbits[bits_x] + mvbits[bits_y] (4.12)

Tabla 4.7. Dependencia de los valores |bits_x| y |bits_y| con respecto al desplazamiento horizontal (P_H) o vertical (P_V) de exploración de la SW.

Valor de P_H o P_V	Valor de bits
0	0 to 31
1, 2	0 to 63
3 to 6	0 to 127
7 to 14	0 to 255
15 to 30	0 to 511
31 to 62	0 to 1023
63 to 126	0 to 2047
127 to 254	0 to 4095

Tabla 4.8. Tabla del código de longitud variable universal para estimar el número de bits asociados con la tasa *R*.

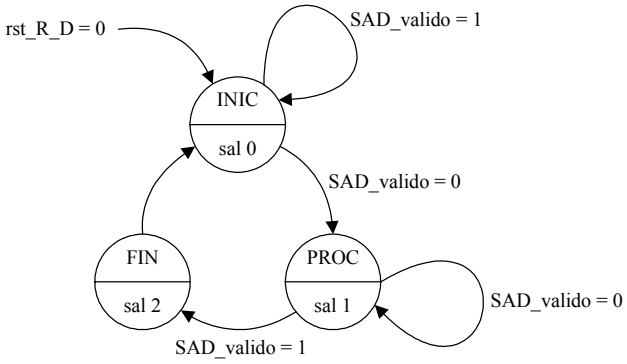
bits	mvbits[bits]
0	1
1	3
2,3	5
4-7	7
8-15	9
16-31	11
32-63	13
64-127	15
128-255	17
256-511	19
512-1023	21
1024-2047	23
2048-4095	25

e) Calcular el coste de la tasa λR (4.13). Este valor se suma a la distorsión SAD para obtener el coste total.

$$\lambda R = (\lambda_{SAD\ integer} * R) >> LAMBDA\_ACCURACY\_BITS \quad (4.13)$$

Una vez obtenido el coste R/D de los 41 bloques y sub-bloques del MB candidato presente, estos resultados se comparan con el coste mínimo de los candidatos previos para actualizar la posición de los mejores candidatos.

La operación R/D es controlada por una FSM Moore (FSM_{RDI}) de tres estados (Figura 4.35). En el estado INIC, la FSM inicializa los registros de mejor posición y mínimo coste con la salida *limpiar_mejor*. Al activarse la entrada *SAD_valido* proveniente del árbol sumador, se salta al estado PROC donde se activa el cálculo y comparación del coste R/D con la salida *procesar_mejor_costo*.



Señales de salida	sal 0	sal 1	sal 2
limpiar_mejor	0	1	1
procesa_mejor_costo	1	0	1
mejor_pos_valida	1	1	0

Figura 4.35. Diagrama de flujo de la FSM que controla la operación del módulo R/D.

Una vez inactiva la señal *SAD_valido*, se salta al estado FIN, donde se indica la validez de los resultados con la salida *mejor_pos_valida*. Al final de su operación, el módulo R/D entrega a la etapa de estimación de movimiento fraccional las coordenadas (x,y) y el coste de los mejores 41 bloques y sub-bloques candidatos con resolución entera. Con esto concluye la evaluación de las particiones y sub-particiones de los MB candidatos de la SW.

4.3.3.8 Evaluación del diseño.

La evaluación del diseño propuesto se realiza utilizando las mediciones aplicadas en las arquitecturas previas (Tablas 4.1 y 4.2), para una matriz de procesamiento de 16×16 PEs y rangos de búsqueda ±Ph y ±Pv (Tabla 4.9).

Tabla 4.9. Evaluación hardware de la arquitectura propuesta para una matriz de procesamiento de 16×16 PEs y rangos de búsqueda ±Ph y ±Pv.

Regularidad del direccionamiento RAM_{SW}	Velocidad de procesamiento (ciclos/MB)	Latencia (ciclos)	Eficiencia %[a]	Ancho RAM_{SW} (bytes)	Ancho de banda E/S[a] (bytes/MB)	Buffer de referencia (bytes)	Incluye E/A[b] RAM_{SW}
muy alta	(2Ph+1)(2Pv+1) +15	16	99.08	32	55 296	272	completa/ completa

a-para Ph = 24 y Pv = 16, b-Estructura/Administración.

El uso del selector de pixeles, con la ventana móvil que selecciona 17 datos en cada ciclo de reloj, de los registros auxiliares, que simplifican el cambio de columna en la exploración de la SW y el acceso a la matriz de PEs por la parte superior e inferior, permiten un flujo de datos muy uniforme y una muy alta regularidad en el direccionamiento de la memoria de pixeles candidatos (RAM_{SW}). La expresión de velocidad presenta casi el valor ideal de procesamiento para arquitecturas de este tipo: un MB candidato por ciclo de reloj. Sin necesidad de ciclos extra, el número de ciclos de operación solo se altera por los 16 ciclos de latencia necesarios para cargar el MB actual y el primer MB candidato en la matriz de PEs. Esto da como resultado una muy alta eficiencia (99.08 %). La memoria RAM_{SW} provee al acelerador hardware con 32 bytes por ciclo de reloj, para un total de 55 296 bytes/MB cuando Ph = 24 y Pv = 16 . Estos valores satisfacen el alto grado de paralelismo que requiere la arquitectura sin la necesidad de un *buffer* de referencia de gran tamaño (272 bytes).

En una comparación con los diseños previos, la arquitectura propuesta, al igual que los trabajos presentados en [Zhang05], [Huang03] y [Chen06a], ofrece valores casi ideales de velocidad de procesamiento, latencia y eficiencia. Su rendimiento es independiente del tamaño de la SW como los diseños de Huang et al. y Chen et al. y sin problemas estructurales para los rangos de búsqueda considerados, como la arquitectura de Zhang et al. Con la más alta regularidad de direccionamiento de todas las arquitecturas analizadas, nuestro diseño presenta una característica única, que es la de permitir esquemas de busqueda con SW no cuadradas al nivel de MB. Se incluye la estructura y administración de la memoria que permite esta funcionalidad a partir de bloques básicos de RAM.

La evaluación del cumplimiento de los objetivos de diseño (pixeles de 8 bits, formato de video de hasta 1080 HD, velocidad de 30 fps, 1 cuadro de referencia y desplazamiento de exploración máximo de 56 pixeles), se realiza en función de la

frecuencia de operación del estimador (Tabla 4.10). La arquitectura IME-FSBM puede satisfacer el procesamiento de diversos formatos de video con varios valores de desplazamiento de exploración, en una operación independiente de la tecnología. Solo basta alcanzar la frecuencia de operación necesaria en su realización. Esto es muy factible para desplazamientos de exploración pequeños y/o formatos de baja resolución.

Tabla 4.10. Frecuencia de operación en MHz de la arquitectura IME-FSBM para procesar varios formatos de video y desplazamientos de exploración de la SW, operación independiente de la tecnología.

	Desplazamientos de exploración (Ph, Pv)					
Formato	(8,8)	(16,16)	(24,16)	(24,24)	(48, 32)	(56,56)
525 SD	12.312	44.712	66.096	97.848	255.960	517.752
625 SD	14.774	53.654	79.315	117.417	307.152	621.302
720p HD	32.832	119.232	176.256	260.928	682.56	1380.672
1080 HD	74.419	270.259	399.513	591.436	1547.136	3129.523

4.3.4 Realización sobre FPGA.

El acelerador hardware IME-FSBM es un diseño general que puede realizarse en cualquiera de las tecnologías de fabricación de sistemas digitales. Al enfocar este trabajo en el diseño con dispositivos FPGA y lenguaje VHDL, se utiliza esta tecnología y lenguaje de programación para la realización del diseño propuesto. Para su síntesis y realización, se han seleccionado los dispositivos Virtex-4 de la empresa Xilinx, por ser una familia de FPGA de alto nivel y última generación, con una amplia gama de recursos para soportar todos los requerimientos hardware y software de este diseño. Un ejemplo de ello son los procesadores local y del sistema a los que se conecta el acelerador hardware, para los cuales la familia Virtex-4 ofrece los microprocesadores MicroBlaze (procesador *soft*) y PowerPC (procesador *hard*).

Para optimizar el uso de las FPGA en la escritura del código RTL y obtener programas más portables y fáciles de interpretar, se han considerado tópicos de diseño para comportamiento tales como la selección adecuada del tipo de *reset*, el uso de técnicas *pipeline* para la reducción del número de niveles lógicos, la minimización de código anidado tipo *if-else*, etc. [Garrault05]. La aplicación de estas técnicas, junto con el uso de las herramientas de Xilinx para síntesis (ISE 8.2i, sintetizador XST), diseño embebido (EDK 8.2i) y simulación (ModelSim Xilinx edition-III v6.1e), permitirán alcanzar las metas de diseño al menor coste posible, para todo el sistema de estimación de movimiento.

A continuación se documenta la codificación RTL y la síntesis y realización sobre la FPGA Virtex-4 XC4VFX60-10-FF1152 de cada uno de los componentes del sistema IME-FSBM propuesto. Se muestran los segmentos de código más importantes, así como los resultados de tiempo y área después de *Place & Route*, usando valores por defecto de los parámetros de síntesis y realización del *ISE 8.2i* y sin definir restricciones de tiempo, para que las operaciones P&R se ejecuten en modo de evaluación del comportamiento (si no se indica otra cosa) [ManualISE8.2i]. Se incluye la conexión entre componentes para observar su interacción y transferencia de información, además de la descripción de las señales de entrada y salida.

Se ha seleccionado al dispositivo XC4V60, de la plataforma FX, con grado de velocidad -10 y encapsulado FF1152 debido a que su arquitectura incluye los recursos hardware suficientes (Tabla 4.11) para alojar simultáneamente todas los diseños y elementos que comprende este trabajo (arquitecturas IME y FME, memorias, buses, procesadores *hard* y *soft*, etc.). Se utiliza el menor grado de velocidad (-10) para buscar obtener el código RTL más eficiente que no dependa totalmente de las características del dispositivo.

Tabla 4.11. Resumen de los recursos hardware de la FPGA Virtex-4 XC4VFX60-10-FF1152 disponibles para la realización del sistema de estimación de movimiento propuesto.

Slices	CLB Flip-flops	Bits RAM distribuidos	Bloques RAM (18 kb)	Select E/S	Xtreme DSP slices	Procesadores PowerPC
25280	50560	404480	232	576	128	2

4.3.4.1 Unidad de control.

La unidad de control es el componente del acelerador hardware que interactúa con el procesador del sistema, para recibir la información de codificación (datos y parámetros) y con el procesador local que controla su operación. Su realización inicia con una referencia a los procesadores y buses disponibles en la tecnología utilizada.

Xilinx ofrece dos procesadores embebidos de alto nivel, para crear sistemas hardware/software alrededor de sus FPGA: el procesador *soft* MicroBlaze y el procesador *hard* PowerPC 405. El MicroBlaze es un procesador configurable tipo RISC y arquitectura Harvard, con buses separados para datos e instrucciones de 32 bits, optimizado para residir en las FPGA Xilinx como un componente software [ManualEDK8.2i]. El PowerPC 405, disponible en algunas FPGA como un bloque hardware (PowerPC 405D5 en la Virtex-II Pro y PowerPC 405F6 en la Virtex-4 plataforma FX), es una versión de 32 bits de la familia de procesadores RISC PowerPC de 64 bits.

Diseñado como un elemento periférico, el acelerador IME puede conectarse a los procesadores PowerPC y/o MicroBlaze a través de los buses para periféricos de propósito general PLB, OPB y FSL (Tabla 4.12). La interfase PLB (*Processor Local Bus*) es un bus local de gran rendimiento diseñado para la conexión de los procesadores PowerPC a dispositivos periféricos maestros y esclavos de alto rendimiento. El OPB (*On-chip Peripheral Bus*) es un bus de propósito general diseñado para una fácil conexión de dispositivos periféricos maestros y esclavos a los procesadores MicroBlaze y PowerPC. El FSL (*Fast Simplex Link*) es un canal de comunicación punto a punto unidireccional basado en memorias FIFO, utilizado para realizar transferencias de datos en 2 ciclos de reloj entre una interfase FSL maestra y otra esclava. En el MicroBlaze se pueden configurar hasta 8 interfases FSL.

La selección del procesador y el bus de periféricos dependerá en primera instancia de su capacidad para realizar todas las funciones que le son asignadas en el sistema estimador, para los datos de entrada y parámetros de operación que le exigen los mayores niveles de procesamiento y transferencia de datos por el bus (máximo tamaño del píxel, formato de video de mayor definición, máxima velocidad en cuadros por segundo, mayor dimensión de la SW, etc.).

Tabla 4.12. Buses para periféricos de los procesadores embebidos de Xilinx.

Especificación	Bus periférico		
	PLB	OPB	FSL
Familia de procesadores	PPC405	PPC405/MicroBlaze	MicroBlaze
Ancho del bus de datos	64	32	32
Ancho del bus de direcciones	32	32	-
Tasa de datos tipica (MB/s)	533	167	-
Tasa de datos pico (MB/s)	1600	500	-
Recursos utilizados (slices)	223 a 1645	46 a 410	21 a 451

En el caso del procesador del sistema, además de calcular y proporcionar al estimador los datos y parámetros de codificación, se encarga de transferir los pixeles actuales de la memoria externa a la memoria local RAM_{MBA}. Para los datos y parámetros máximos especificados en la arquitectura propuesta (pixeles de 8 bits, formato de video hasta 1080 HD, velocidad de 30 fps, 1 cuadro de referencia y desplazamientos de exploración maximos de Ph = 56 y Pv = 56), los requerimientos de operación son mínimos (bajo nivel de procesamiento y tasa de transferencia de datos de 63.65 MBytes/s). Ya que todos los procesadores y buses que ofrece Xilinx cumplen estos requisitos, como procesador del sistema se prefiere al MicroBlaze, por su versatilidad de integrarse a la mayoría de las FPGA disponibles, a diferencia del PowerPC que solo lo incluyen algunos dispositivos de gran densidad. Para el bus del sistema se elige al OPB por su universalidad, sencillez y bajo coste, utilizando registros esclavos de 32 bits accesibles por software como interfase con el periférico.

Ya que las necesidades de operación entre el procesador local y la unidad de control se pueden cumplir con otro registro esclavo, el tipo de procesador y bus dependerá de la selección que satisfaga los requerimientos de la principal función de estos elementos locales: la transferencia de pixeles candidatos de la RAM externa de cuadros de referencia a las dos RAM_{SW}. A manera de ejercicio, la tabla 4.13 presenta las tasas de transferencia en MBytes/s que se necesitan para procesar la estimación de movimiento entero para varios formatos de video y desplazamientos de exploración (pixeles de 8 bits, 30 fps y 1 cuadro de referencia). De los valores de ancho de banda que se observan, el bus OPB (tasa típica de 167 MBytes/s) satisface 7 y el bus PLB (tasa típica de 533 MBytes/s) 15. Estos resultados muestran la necesidad de utilizar localmente al MicroBlaze o al PowerPC, con sus buses OPB y PLB, delimitando el formato de video y rango de operación máximos para cada combinación bus-procesador como se indica en la tabla 4.13.

Tabla 4.13. Tasas de transferencia en MBytes/s para algunos formatos de video y rangos de exploración de la ventana de búsqueda.

Formato	Desplazamientos de exploración (Ph, Pv)					
	(8,8)	(16,16)	(24,16)	(24,24)	(48, 32)	(56,56)
525 SD	41.47	93.31	124.41	165.888	362.88	663.55
625 SD	49.76	111.97	149.30	199.065	435.45	796.26
720p HD	110.59	248.83	331.77	442.368	967.68	1769.47
1080 HD	250.67	564.02	752.02	1002.7	2193.41	4010.80

La arquitectura IME-FSBM, que como periférico se conecta a los buses OPB y PLB a través de la unidad de control, incluye los siguientes componentes (Figura 4.36):

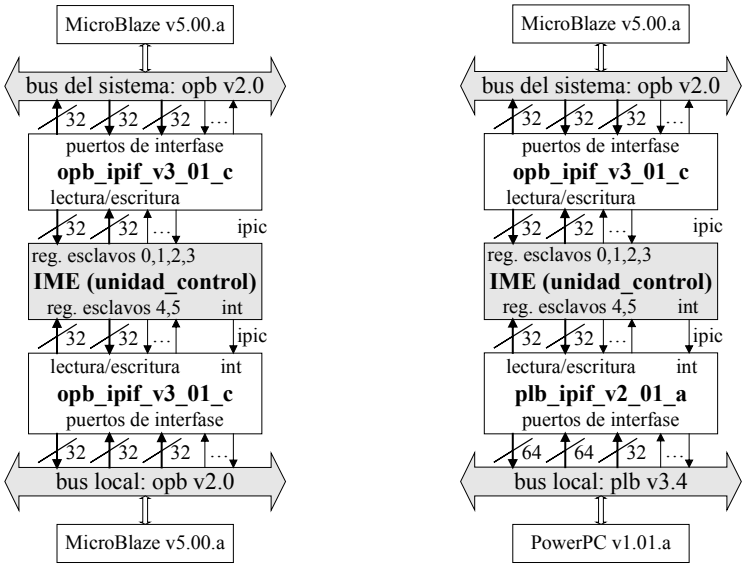


Figura 4.36. Interconexión de la unidad de control del periférico IME con el procesador y bus del sistema (MicroBlaze y OPB) y con el procesador y bus local (MicroBlaze-OPB o PowerPC-PLB) utilizando las interfases IPIF y sus puertos correspondientes.

* Puertos de interfase al bus.

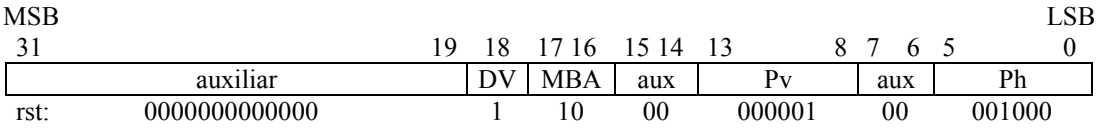
* Interfase de propiedad intelectual (IPIF) para periféricos OPB y PLB.

* Lógica del usuario, que se conecta a la IPIF mediante un conjunto de puertos llamados interconexión de propiedad intelectual (IPIC).

Los puertos de interfase y la IPIF se generan con la herramienta EDK 8.2i de Xilinx, seleccionando las funciones que necesita la lógica del usuario como decodificación de direcciones, registros, interrupciones, FIFOs de lectura y escritura y DMAs. Para el estimador IME, se establece una interrupción por borde que señala al procesador local el fin de la exploración de la SW. También se especifican registros esclavos de 32 bits para el bus OPB y de 64 bits para el bus PLB, con los cuales se realiza la transferencia de información y control entre los procesadores y el acelerador hardware.

El procesador del sistema (MicroBlaze) escribe en el registro de configuración 0 (Figura 4.37) los desplazamientos de exploración, el número de MB actual a utilizar y la validez de los datos y en los registros 1 a 4 (Figura 4.38), los datos para el cálculo del coste R/D (pic_pix_x, pic_pix_y, search_x, search_y, pred_mv_x, pred_mv_y y lambda_factor).

El procesador local indica en el registro 5 (Figura 4.39) la memoria RAM_{SW} valida, activando el inicio de operaciones con el bit IE. La unidad de control le indica al procesador local el fin del ciclo de exploración por dos métodos: estableciendo el bit IE y por interrupción externa. Cuando se utiliza al PowerPC como procesador local, el registro 5 se concatena a la derecha de un registro auxiliar de 32 bits para formar el registro de 64 bits que maneja el bus PLB.



Ph Desplazamiento de exploración horizontal. Valores: 8, 16, 24, 32, 40, 48 y 56.
Pv Desplazamiento de exploración vertical. Valores: 1, 2, 3, 4, 5,....., 56.
MBA 00 Posición del MB actual en la memoria RAM_{MBA}: 0.
01 Posición del MB actual en la memoria RAM_{MBA}: 1.
1x Posición del MB actual no valida.
DV 0 Datos y parámetros validos en los registros 0 a 4.
1 Datos y parámetros no validos o ya leídos.

Figura 4.37. Especificación del registro de configuración 0, bus del sistema.

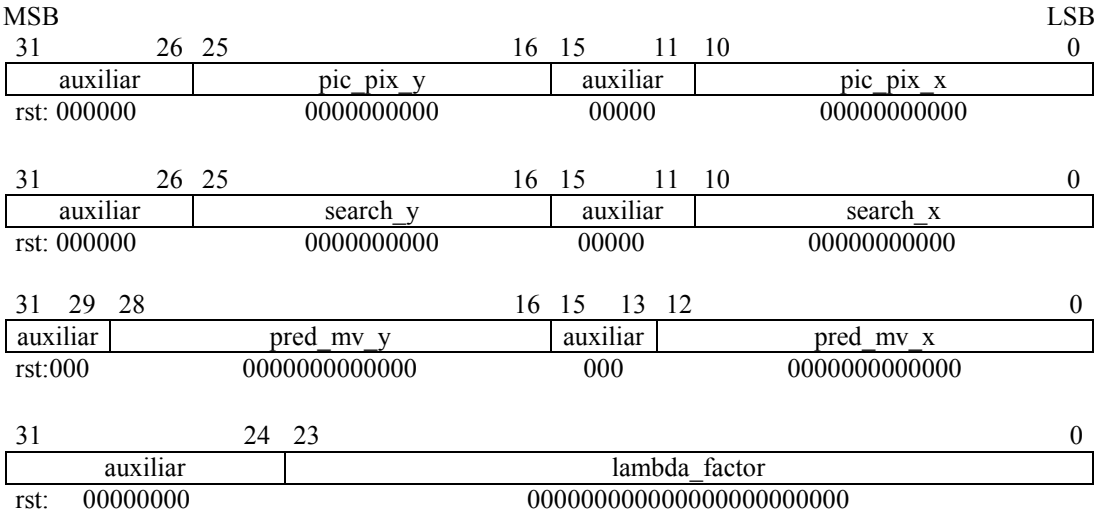
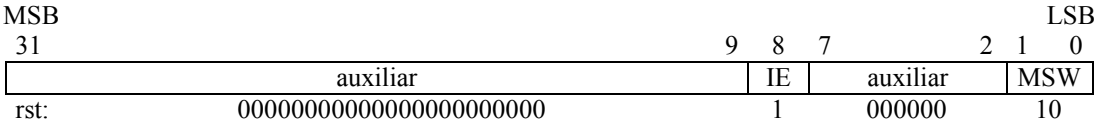


Figura 4.38. Especificación de los registros de configuración 1, 2, 3 y 4, bus del sistema.



MSW 00 Validación de datos y habilitación de la memoria RAM_{sw} #0.
01 Validación de datos y habilitación de la memoria RAM_{SW} #1.
1x No validación de datos ni habilitación de las memoria RAM_{SW}.
IE 0 Iniciar exploración (cambia de 0 a 1 cuando la exploración termina).
1 Fin de exploración.

Figura 4.39. Especificación del registro de configuración 5, bus local.

Además de interactuar con los buses de los procesadores por medio de las IPIF, la unidad de control se conecta a otros módulos (Figura 4.40) para enviar y recibir las señales de inicio, fin y configuración de la operación (AGU_MBA, AGU_SW), validar resultados intermedios (arbol_sum) y transferir parámetros del coste R/D (R_D_IME). A manera de ejemplo, la tabla 4.14 muestra todas las señales de entrada/salida (E/S) de la unidad de control junto con las de interconexión a los buses (puertos de la IPIF), para una aplicación que usa dos procesadores MicroBlaze.

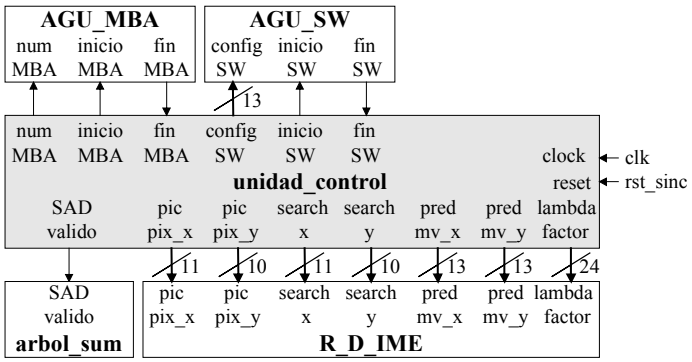


Figura 4.40. Interconexión de la unidad de control con los componentes del acelerador hardware.

Tabla 4.14. Descripción de las señales de E/S de la unidad de control, para una aplicación con dos procesadores MicroBlaze.

Señal	Tipo	Descripción
clock	E	Reloj, con operación en su borde de subida
reset	E	Reset síncrono, activo en bajo
Bus2IP_Data_0(0 to 31)	E	Datos del bus a la *IPIF* (*OPB* del sistema)
Bus2IP_BE_0(0 to 3)	E	Habilitación de bytes del bus a la *IPIF* (*OPB* del sistema)
Bus2IP_RdCE_0(0 to 3)	E	Habilitación de lectura, del bus a la *IPIF* (*OPB* del sistema)
Bus2IP_WrCE_0(0 to 3)	E	Habilitación de escritura, del bus a la *IPIF* (*OPB* del sistema)
Bus2IP_Data_1(0 to 31)	E	Datos del bus a la *IPIF* (*OPB* local)
Bus2IP_BE_1(0 to 3)	E	Habilitación de bytes del bus a la *IPIF* (*OPB* local)
Bus2IP_RdCE_1(0 to 0)	E	Habilitación de lectura, del bus a la *IPIF* (*OPB* local)
Bus2IP_WrCE_1(0 to 0)	E	Habilitación de escritura, del bus a la *IPIF* (*OPB* local)
fin_MBA	E	Fin de la operación de la AGU del MB actual
fin_SW	E	Fin de la operación de la AGU de la SW
IP2Bus_IntrEvent_1(0 to 0)	S	Evento de interrupción de la *IPIF* al bus (*OPB* local)
IP2Bus_Data_0(0 to 31)	S	Datos de la *IPIF* al bus (*OPB* del sistema)
IP2Bus_Ack_0	S	Reconocimiento de la *IPIF* al bus (*OPB* del sistema)
IP2Bus_Retry_0	S	Respuesta de recuperación de la *IPIF* al bus (*OPB* del sistema)
IP2Bus_Error_0	S	Respuesta de error de la *IPIF* al bus (*OPB* del sistema)
IP2Bus_ToutSup_0	S	Supresión de tiempo fuera de la *IPIF* al bus (*OPB* del sistema)
IP2Bus_Data_1(0 to 31)	S	Datos de la *IPIF* al bus (*OPB* local)
IP2Bus_Ack_1	S	Reconocimiento de la *IPIF* al bus (*OPB* local)
IP2Bus_Retry_1	S	Respuesta de recuperación de la *IPIF* al bus (*OPB* local)
IP2Bus_Error_1	S	Respuesta de error de la *IPIF* al bus (*OPB* del sistema)
IP2Bus_ToutSup_1	S	Supresión de tiempo fuera de la *IPIF* al bus (*OPB* local)
inicio_MBA	S	Inicio de la operación de la AGU del MB actual
inicio_SW	S	Inicio de la operación de la AGU de la SW
num_MBA	S	Número de MB actual (0 - MB #0, 1- MB #1)
config_SW(12 downto 0)	S	Información del número de RAM_{SW} y rangos de operación
SAD_valido	S	Validación de los valores de SAD que recibe el arbol sumador
pic_pix_x(10 downto 0)	E	Coordenada x del MB actual en unidades de pixeles
pic_pix_y(9 downto 0)	E	Coordenada y del MB actual en unidades de pixeles
search_x(10 downto 0)	E	Coordenada x de la SW en unidades de pixeles
search_y(9 downto 0)	E	Coordenada y de la SW en unidades de pixeles
pred_mv_x(12 downto 0)	E	Valor x de la predicción del MV, en unidades de ¼ de pixel
pred_mv_y(12 downto 0)	E	Valor y de la predicción del MV, en unidades de ¼ de píxel
lambda_factor(23 downto 0)	E	Equivalente entero de λ_{SAD}

La FSM que define estas señales de control y retroalimentación se realiza como una FSM Moore, utilizando tres procesos VHDL. La figura 4.41 muestra el proceso de

decodificación de las señales de salida, utilizando el siguiente estado como la variable de decisión para reducir la latencia de la máquina.

```
process (sig_estado)
    begin
        -- Valores de salida por defecto.
        inicio_MBA            <= '1';
        inicio_SW             <= '1';
        SAD_valido            <= '1';
        interrupcion_borde    <= '0';
        case (sig_estado) is
            when init =>
                null;
            when load =>
                inicio_MBA    <= '0';
                inicio_SW     <= '0';
            when search =>
                inicio_SW     <= '0';
                SAD_valido    <= '0';
            when others =>
                SAD_valido            <= '0';
                interrupcion_borde    <= '1';
        end case;
    end process;
```

Figura 4.41. Proceso de decodificación de salidas de la FSM Moore de la unidad de control.

Como un ejemplo de los resultados de síntesis y realización, la tabla 4.15 muestra las estadísticas post-P&R de la codificación de la unidad de control, con los componentes IPIF que lo conectan a los procesadores MicroBlaze vía los buses OPB. Se observa que es un componente optimizado en área y velocidad.

Tabla 4.15. Estadísticas de síntesis y realización post-P&R de la unidad de control, con las dos interfases a los buses OPB para los procesadores MicroBlaze.

Recurso/especificación	Valor
Slices	168
Flip-flops	162
4-input LUTs	201
Puertas equivalentes	3601
Frecuencia (MHz)	267

4.3.4.2 Matriz de 16×16 PEs.

La matriz de procesamiento esta compuesta por 256 PEs, 16 registros auxiliares (Raux) y 16 árboles sumadores de bloque (arbol_sum_bl), donde se obtienen los valores de distorsión SAD de 16 bloques 4×4, utilizando una codificación en dos niveles jerárquicos de descripción estructural. El primer nivel define un bloque de 4×4 PEs (bloque_PEs) instanciando 16 PEs y el segundo hace referencia a 16 bloque_PEs para describir la matriz completa (matriz_PEs). El componente matriz_PEs interactúa con los módulos RAM_MBA, AGU y_MBA, AGU_SW, mux_sel_pix y arbol_sum (Figura 4.42) utilizando sus señales de E/S (Tabla 4.16).

Los PEs están diseñados para conectarse entre sí y con los registros auxiliares (Figura 4.43), utilizando señales de datos de pixeles actuales y candidatos (Tabla 4.17).

Su control y temporización se realiza con señales generales que recibe la matriz desde las AGUs del sistema.

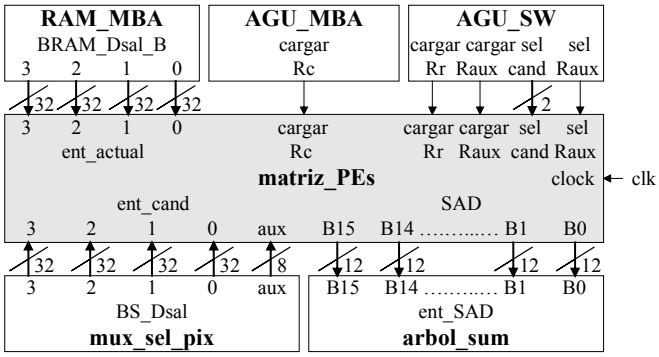


Figura 4.42. Componente matriz_PEs y su conexión con los módulos con quienes intercambia información.

Tabla 4.16. Descripción de las señales de E/S de la matriz de PEs.

Señal	Tipo	Descripción
clock	E	Reloj, con operación en su borde de subida
ent_actual_0...3(31 downto 0)	E	Renglón de 16 pixeles del MB actual
ent_cand_0...3(31 downto 0)	E	Renglón de 16 pixeles del MB candidato
ent_cand_aux(7 downto 0)	E	Pixel auxiliar para el cambio de columna
cargar_Rc	E	Habilitación de la carga de pixeles actuales en los registros *Rc* de los PEs. Activa en bajo
cargar_Rr	E	Habilitación de la carga de los pixeles candidatos en los registros *Rr* de los PEs. Activa en bajo
cargar_Raux	E	Habilitación de la carga de los pixeles auxiliares en los registros Raux. Activa en bajo
sel_cand(1 downto 0)	E	Selección del píxel candidato superior (*sel_cand* = "00"), derecho ("01"), inferior ("10") o ninguno ("11) en cada PE
sel_Raux	E	Selección del píxel auxiliar superior (*sel_Raux* = '0') o inferior ('1') en cada PE
SAD_B0...B15(11 downto 0)	S	Valor de la distorsión (SAD) de los bloques candidatos 4×4

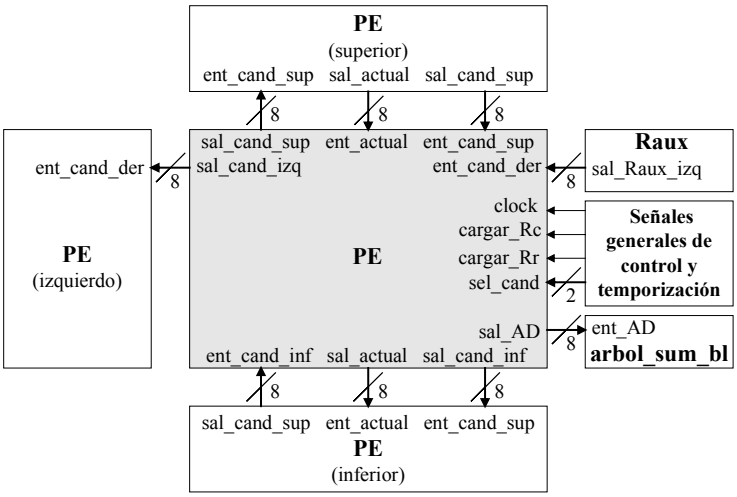


Figura 4.43. Elemento PE y su conexión con el registro auxiliar y los PEs superior, inferior e izquierdo.

Tabla 4.17. Descripción de las señales de E/S del elemento procesador.

Señal	Tipo	Descripción
clock	E	Reloj, con operación en su borde de subida
ent_actual(7 downto 0)	E	Píxel del MB actual
ent_cand_sup(7 downto 0)	E	Pixel candidato del PE superior
ent_cand_der(7 downto 0)	E	Pixel candidato del PE o registro auxiliar derecho
ent_cand_inf(7 downto 0)	E	Pixel candidato del PE inferior
cargar_Rc	E	Habilitación de la carga del pixel actual en el registro Rc. Activa en bajo
cargar_Rr	E	Habilitación de la carga del pixel candidato en el registro *Rr*. Activa en bajo
sel_cand(1 downto 0)	E	Selección del píxel candidato superior (*sel_cand* = "00"), derecho ("01"), inferior ("10") o ninguno ("11)
sal_actual(7 downto 0)	S	Píxel del MB actual
sal_cand_sup(7 downto 0)	S	Pixel candidato para el PE superior
sal_cand_izq(7 downto 0)	S	Pixel candidato para el PE izquierdo
sal_cand_inf(7 downto 0)	S	Pixel candidato para el PE inferior
sal_AD(7 downto 0)	S	Valor de la distorsión (*AD*) del píxel candidato

La codificación del PE se realiza con una descripción algorítmica de 5 elementos: multiplexor del píxel candidato, unidad de diferencias absolutas y registros Rr, Rc y Raux. El multiplexor del píxel candidato se describe con una declaración *case*, lo que implica una codificación paralela puramente combinacional (Figura 4.44). Los registros Rr, Rc y Rad son codificados en forma independiente infiriendo FF tipo D sin *reset*, activados en el borde de subida del reloj (Figura 4.45). Las opciones que se presentan para la codificación de los registros están en función de la polaridad de las señales de control. Xilinx especifica en sus plantillas de lenguaje (ISE 8.2i) señales de control normalmente activas en alto, por lo que se comprobara con resultados de síntesis y realización de la matriz de PEs si esta aseveración es correcta en este caso.

```
process(sel_cand, ent_cand_sup, ent_cand_der, ent_cand_inf)
    begin
        case sel_cand is
            when "00"   => Dr <= ent_cand_sup;
            when "01"   => Dr <= ent_cand_der;
            when "10"   => Dr <= ent_cand_inf;
            when others => Dr <= (others => '0');
        end case;
    end process;
```

Figura 4.44. Proceso del multiplexor para la selección del píxel candidato del PE, donde Dr es la entrada D del registro Rr.

Los datos de entrada de la diferencia absoluta |Rc-Rr| son pixeles de 8 bits, por lo que el resultado tiene un rango dinámico de 0 a 255, con una representación en 8 bits. Para su codificación se tienen tres alternativas en función del cálculo del valor absoluto (Figura 4.46). En la opción (a) se utiliza el operador correspondiente definido en el lenguaje VHDL (abs()). La opción (b) usa un comparador y dos restas con diferente orden de los operandos para obtener siempre un resultado positivo o cero. En la opción (c) se manipulan los operandos de una resta única empleando un comparador y variables locales. La mejor alternativa se definirá posteriormente en función de los resultados de síntesis y realización de la matriz de PEs.

```
process (clock)
    begin
       if (clock'event and clock = '1') then
          -- Registro Rr.
          if cargar_Rr = '0' then
             Qr <= Dr;
          end if;
          -- Registro Rc
          if cargar_Rc = '0' then
             Qc_PE <= ent_actual;
          end if;
          -- Registro Rad.
          AD <= sal_AD(7 downto 0);
       end if;
    end process;
```

Figura 4.45. Codificación de los registros Rr, Rc y Rad del elemento procesador.

a) Descripción utilizando la función *abs()* y una resta.

```
process (Qc, Qr)
    begin
       sal_dif_abs <= abs( ('0'&Qc) - ('0'&Qr) );
    end process;
```

b) Descripción con una comparación y dos restas.

```
process (Qc, Qr)
    begin
       if (Qc > Qr) then
          resta <= Qc - Qr;
       else
          resta <= Qr - Qc;
       end if;
    end process;
```

c) Descripción con una comparación y una resta utilizando variables.

```
process (Qc_PE, Qr_PE)
    variable aux_resta_0, aux_resta_1 : std_logic_vector(7 downto 0);
    begin
       if (Qc > Qr) then
          aux_resta_0 := Qc;
          aux_resta_1 := Qr;
       else
          aux_resta_0 := Qr;
          aux_resta_1 := Qc;
       end if;
          resta <= aux_resta_0 - aux_resta_1;
    end process;
```

Figura 4.46. Opciones de codificación de la diferencia absoluta |Rc-Rr| en los PEs.

Dentro de la matriz de procesamiento, los registros auxiliares interactúan verticalmente entre sí y horizontalmente con los PE a su izquierda (Figura 4.47). El flujo de datos para su intercambio de pixeles candidatos se controla con señales provenientes del componente AGU_{SW} (Tabla 4.18). La arquitectura del *Raux* incluye un multiplexor 2 a 1 y un registro de 8 bits, por lo que su codificación algorítmica solo presenta la duda sobre la polaridad de las señales de control.

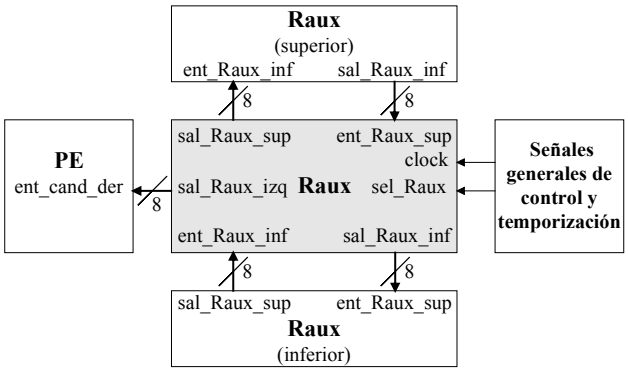


Figura 4.47. Elemento Raux y su interfase con los registros auxiliares superior e inferior y el PE izquierdo.

Tabla 4.18. Descripción de las señales de E/S del registro auxiliar.

Señal	Tipo	Descripción
clock	E	Reloj, con operación en su borde de subida
ent_Raux_sup(7 downto 0)	E	Píxel candidato auxiliar del Raux superior
ent_Raux_inf(7 downto 0)	E	Pixel candidato auxiliar del Raux inferior
sel_Raux	E	Selección del píxel candidato auxiliar superior (*sel_Raux* = '0') o inferior ('1')
sal_Raux_sup(7 downto 0)	S	Píxel del MB actual
sal_Raux_izq(7 downto 0)	S	Pixel candidato para el PE superior
sal_Raux_inf(7 downto 0)	S	Pixel candidato para el PE izquierdo

Cada árbol sumador de bloque (arbol_sum_bl) recibe 16 resultados sal_AD de los PEs que agrupa y los procesa paralelamente en 4 niveles de sumadores, hasta obtener el valor SAD_Bx. Ya que el número de niveles lógicos combinacionales puede impedir cumplir las metas de diseño con el dispositivo FPGA utilizado, se hace necesaria una segmentación a la mitad del árbol junto con el registro del resultado de distorsión (Figura 4.48). El coste es el aumento de la latencia en 2 ciclos de reloj. En su código se utilizan sumadores de 9, 10, 11 y 12 bits y registros con FF D sin *reset*.

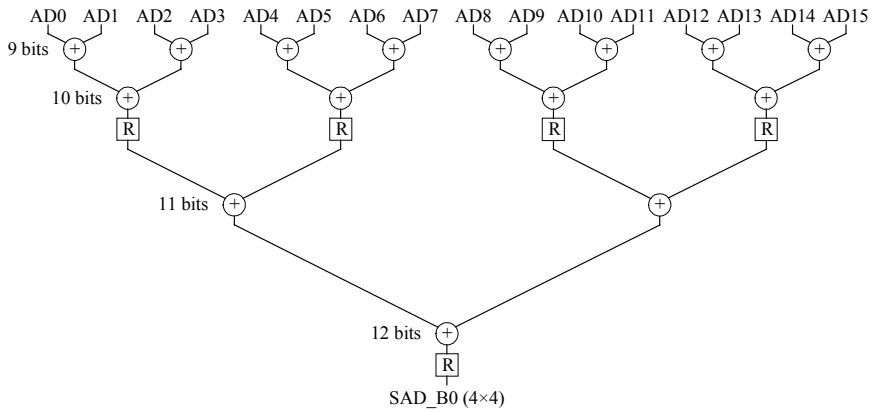


Figura 4.48. Árbol para el cálculo del *SAD* del bloque 0, con indicación del tamaño de los sumadores y la segmentación cada dos niveles de operaciones.

La tabla 4.19 exhibe las estadísticas post-P&R del componente matriz_PEs, para las alternativas de polaridad de las señales de control y las opciones de codificación del

proceso de diferencia absoluta. Los resultados muestran en promedio un menor número de *slices* y una mayor frecuencia de operación con las señales de control activas en bajo. De acuerdo a las alternativas de codificación de |Rc−Rr|, las opciones (b) y (c) ofrecen resultados idénticos, pero de menor eficiencia que la opción (a). En conclusión, el componente matriz_PEs tiene un mejor comportamiento área-velocidad con señales de control activas en bajo y con el uso de la función *abs()*.

Tabla 4.19. Estadísticas de síntesis y realización post-P&R de las alternativas de codificación del componente matriz_PEs.

	Polaridad de las señales de control					
	Activas en bajo			Activas en alto		
Recurso ó	Alternativa de codificación de |Rc−Rr|			Alternativa de codificación de |Rc−Rr|		
especificación	a	b	c	a	b	c
Slices	7489	7755	7755	7496	7766	7766
Flip-flops	6800	6800	6800	6800	6800	6800
4-input LUTs	12060	13084	13084	12068	13092	13092
Puertas equivalentes	156696	167448	167448	156744	167496	167496
Frec. (MHz)	129.17	127.76	127.76	125.05	128.06	128.06

4.3.4.3 Árbol sumador.

El árbol sumador calcula el SAD de 6 tamaños de bloque, a partir de 16 SAD de los bloques 4×4 y entrega a la siguiente etapa 41 valores SAD y su señal de validación (Figura 4.49). En su codificación RTL, se considera la necesidad de cumplir la meta de velocidad del sistema y el requerimiento de la siguiente etapa de recibir los resultados simultáneamente, por lo que se agregan registros de segmentación en cada nivel de sumadores y registros de retardo en los valores SAD iniciales e intermedios, con un coste de 4 ciclos de reloj en latencia. Se incluye la descripción de las señales de E/S (Tabla 4.20) y las estadísticas de síntesis y realización post-P&R (Tabla 4.21).

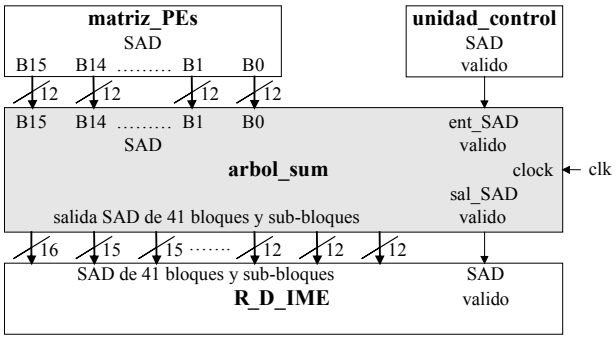


Figura 4.49. Árbol sumador y su interfase con otros componentes del sistema.

4.3.4.4 Módulo de tasa/distorsión.

Al inicio del ciclo de exploración, el módulo de tasa/distorsión (R/D) recibe de la unidad de control los parámetros del coste λR y, en cada ciclo de reloj, los valores SAD y las coordenadas de las 41 particiones y sub-particiones del MB candidato actual, desde los componentes arbol_sum y AGU_SW (Figura 4.50). Al término de la exploración, la señal *mejor_pos_valida* (Tabla 4.22) activa el almacenamiento de las coordenadas de los bloques candidatos de menor coste en el *buffer* IME-FME. Junto con

los resultados, se respaldan los datos de configuración y coste del procesamiento IME, para que la etapa FME realice su operación con la información adecuada.

Tabla 4.20. Descripción de las señales de E/S del árbol sumador.

Señal	Tipo	Descripción
clock	E	Reloj, con operación en su borde de subida
SAD_B0…15(11 downto 0)	E	Valor SAD de los bloques 0 a 15 (4×4)
ent_SAD_valido	E	Validación de los valores SAD de entrada
SAD_B0…15(11 downto 0)	S	Valor SAD de los bloques 0 a 15 (4×4)
SAD_B0_4(12 downto 0)	S	Valor SAD del bloque 0_4 (4×8)
SAD_B1_5(12 downto 0)	S	Valor SAD del bloque 1_5 (4×8)
SAD_B2_6(12 downto 0)	S	Valor SAD del bloque 2_6 (4×8)
SAD_B3_7(12 downto 0)	S	Valor SAD del bloque 3_7 (4×8)
SAD_B8_12(12 downto 0)	S	Valor SAD del bloque 8_12 (4×8)
SAD_B9_13(12 downto 0)	S	Valor SAD del bloque 9_13 (4×8)
SAD_B10_14(12 downto 0)	S	Valor SAD del bloque 10_14 (4×8)
SAD_B11_15(12 downto 0)	S	Valor SAD del bloque 11_15 (4×8)
SAD_B0_1(12 downto 0)	S	Valor SAD del bloque 0_1 (8×4)
SAD_B2_3(12 downto 0)	S	Valor SAD del bloque 2_3 (8×4)
SAD_B4_5(12 downto 0)	S	Valor SAD del bloque 4_5 (8×4)
SAD_B6_7(12 downto 0)	S	Valor SAD del bloque 6_7 (8×4)
SAD_B8_9(12 downto 0)	S	Valor SAD del bloque 8_9 (8×4)
SAD_B10_11(12 downto 0)	S	Valor SAD del bloque 10_11 (8×4)
SAD_B12_13(12 downto 0)	S	Valor SAD del bloque 12_13 (8×4)
SAD_B14_15(12 downto 0)	S	Valor SAD del bloque 14_15 (8×4)
SAD_B0_1_4_5(13 downto 0)	S	Valor SAD del bloque 0_1_4_5 (8×8)
SAD_B2_3_6_7(13 downto 0)	S	Valor SAD del bloque 2_3_6_7 (8×8)
SAD_B8_9_12_13(13 downto 0)	S	Valor SAD del bloque 8_9_12_13 (8×8)
SAD_B10_11_14_15(13 downto 0)	S	Valor SAD del bloque 10_11_14_15 (8×8)
SAD_B0_1_4_5_8_9_12_13 (14 downto 0)	S	Valor SAD del bloque 0_1_4_5_8_9_12_13 (8×16)
SAD_B2_3_6_7_10_11_14_15 (14 downto 0)	S	Valor SAD del bloque 2_3_6_7_10_11_14_15 (8×16)
SAD_B0_1_4_5_2_3_6_7 (14 downto 0)	S	Valor SAD del bloque 0_1_4_5_2_3_6_7 (16×8)
SAD_B8_9_12_13_10_11_14_15 (14 downto 0)	S	Valor SAD del bloque 8_9_12_13_10_11_14_15 (16×8)
SAD_MB(15 downto 0)	S	Valor SAD del MB (16×16)
sal_SAD_valido	S	Validación de los valores SAD de salida

Tabla 4.21. Estadísticas de síntesis y realización post-P&R del árbol sumador.

Recurso/especificación	Valor
Slices	631
Flip-flops	219
4-input LUTs	315
Puertas equivalentes	39089
Frecuencia (MHz)	173.85

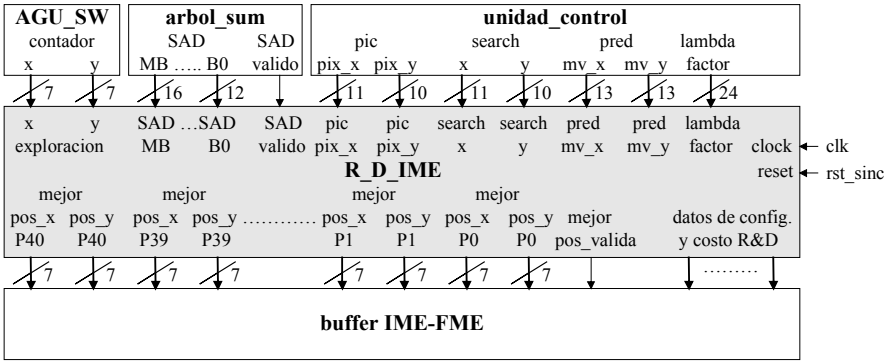


Figura 4.50. Componente de coste R/D y su interfase con otros módulos.

Tabla 4.22. Descripción de las señales de E/S del componente de coste tasa/distorsión.

Señal	Tipo	Descripción
clock	E	Reloj, con operación en su borde de subida
reset	E	Reset síncrono, activo en bajo
exploracion_x(6 downto 0)	E	Coordenada x del MB candidato en evaluación
exploracion_y(6 downto 0)	E	Coordenada y del MB candidato en evaluación
SAD_MB...B0(15..11 downto 0)	E	Valor SAD de 41 bloques y sub-bloques
SAD_valido	E	Validación de los valores SAD de entrada
pic_pix_x(10 downto 0)	E	Coordenada x del MB actual en unidades de pixeles
pic_pix_y(9 downto 0)	E	Coordenada y del MB actual en unidades de pixeles
search_x(10 downto 0)	E	Coordenada x de la SW en unidades de pixeles
search_y(9 downto 0)	E	Coordenada y de la SW en unidades de pixeles
pred_mv_x(12 downto 0)	E	Valor x de la predicción del MV, en unidades de ¼ de pixel
pred_mv_y(12 downto 0)	E	Valor y de la predicción del MV, en unidades de ¼ de pixel
lambda_factor(23 downto 0)	E	Equivalente entero de λSAD
mejor_pos_x_P40(6 downto 0)	S	Posición x del MB con la mejor partición 40 (16×16)
mejor_pos_y_P40(6 downto 0)	S	Posición y del MB con la mejor partición 40 (16×16)
:	:	:
mejor_pos_x_P1(6 downto 0)	S	Posición x del MB con la mejor partición 1 (4×4)
mejor_pos_y_P1(6 downto 0)	S	Posición y del MB con la mejor partición 1 (4×4)
mejor_pos_x_P0(6 downto 0)	S	Posición x del MB con la mejor partición 0 (4×4)
mejor_pos_y_P0(6 downto 0)	S	Posición y del MB con la mejor partición 0 (4×4)
mejor_posición_valida	S	Validación de los resultados de R/D
dato_num_MB_actual	S	Dato config.: número MB actual en RAM_{MBA} (0 ó 1)
dato_num_mem_cand	S	Dato config.: número RAM_{SW} (0 ó 1)
dato_rango_Ph(5 downto 0)	S	Dato config.: rango de búsqueda horizontal
dato_rango_Pv(5 downto 0)	S	Dato config.: rango de búsqueda vertical
dato_pic_pix_x(9 downto 0)	S	Dato coste: coordenada x del MB actual en unidades de pixeles
dato_pic_pix_y(9 downto 0)	S	Dato coste: coordenada y del MB actual en unidades de pixeles
dato_search_x(9 downto 0)	S	Dato coste: coordenada x de la SW en unidades de pixeles
dato_search_y(9 downto 0)	S	Dato coste: coordenada y de la SW en unidades de pixeles
dato_pred_mv_x(12 downto 0)	S	Dato coste: componente x de la predicción del MV, en unidades de ¼ de píxel
dato_pred_mv_y(12 downto 0)	S	Dato coste: componente y de la predicción del MV, en unidades de ¼ de pixel
dato_lambda_factor(23 downto 0)	S	Dato coste: equivalente entero de λSAD

Las principales secciones del programa en VHDL corresponden al cálculo del coste de la tasa λR (Figura 4.51) y a la suma y comparación del coste total (Figura 4.52). La primera es una descripcion RTL/algoritmica con operación *pipeline* que se ejecuta una sola vez por cada MB candidato. La segunda sección se describe algoritmicamente y se instancia 41 veces, para obtener las coordenadas de los bloques y sub-bloques candidatos con mínimo coste. La sintesis y realización por defecto del ISE 8.2i (Tabla 4.23) resulta en un componente no optimo, por lo que se necesitaran emplear las especificaciones de optimización de la herramienta en el sistema completo.

```
-- Cálculo de coordenadas del MB candidato.
cand_x <= ('0'&search_x) + ("0000"&exploracion_x); --(0 a +1150).
cand_y <= ('0'&search_y) + ("0000"&exploracion_y); --(0 a +1150).

-- Determinacion del punto final de la prediccion del vector de movimiento mvp.
pred_x <= ("00"&pic_pix_x&"00") + SXT(pred_mv_x, 14); --(-4096 a +8187).
pred_y <= ("00"&pic_pix_y&"00") + SXT(pred_mv_y, 14); --(-4096 a +8187).

-- Cálculo de la diferencia MV-MVp.
bits_x <= ('0'&cand_x_r &"00") - pred_x_r ; --(-8187 a +5246).
bits_y <= ('0'&cand_y_r &"00") - pred_y_r ; --(-8187 a +5246).

-- Estimación de la tasa R a partir de la tabla UVLC.
process(bits_r)
    variable bits_int : int_mv;
    begin
        bits_int := abs(CONV_INTEGER(bits_r)); --(0 a +8187).
        case bits_int is
            when 0 => Mv_bits <= "00001"; --(1 to 25).
            when 1 => Mv_bits <= "00011";
            when 2 | 3   => Mv_bits <= "00101";
            when 4 to 7 => Mv_bits <= "00111";
            when 8 to 15   => Mv_bits <= "01001";
            when 16 to 31 => Mv_bits <= "01011";
            when 32 to 63   => Mv_bits <= "01101";
            when 64 to 127 => Mv_bits <= "01111";
            when 128 to 255  => Mv_bits <= "10001";
            when 256 to 511  => Mv_bits <= "10011";
            when 512 to 1023   => Mv_bits <= "10101";
            when 1024 to 2047 => Mv_bits <= "10111";
            when others => Mv_bits <= "11001";
        end case;
    end process;
Rcost <= ('0'&Mv_bits_x_r) + ('0'&Mv_bits_y_r); --(0 a 50).

-- Multiplication lambda_factor * Rcost.
lambda_Rcost_0 <= ('0'&lambda_factor) * ('0'&Rcost_r); --(0 a 838 860 750).

-- Cambio de escala: (lambda_factor * Rcost) >> LAMBDA_ACCURACY_BITS.
process (lambda_Rcost_0_r, LAB)
    variable LAB_int : int_lambda;
    variable lambda_Rcost_0_bv, lambda_Rcost_1_bv : bit_vector(31 downto 0);
    begin
        LAB_int :=  CONV_INTEGER('0'&LAB);
        lambda_Rcost_0_bv := To_bitvector(lambda_Rcost_0_r);
        lambda_Rcost_1_bv := lambda_Rcost_0_bv srl LAB_int;
        lambda_Rcost <= To_StdLogicVector(lambda_Rcost_1_bv);
    end process;
```

Figura 4.51. Código VHDL para el cálculo del coste de la tasa λR.

```
-- Cálculo del coste R/D:  SAD + (lambda * R)
RD_cost <= ("00000"&SAD_cost) + ('0'&lambdaR_cost);

-- Comparación y actualización de las coordenadas del bloque con coste mínimo.
process(clock)
    begin
        if (clock'event and clock = '1') then
            if (reset = '0') then
                best_pos_x <= (others => '0');
                best_pos_y <= (others => '0');
                min_mcost <= (others => '1');
            elsif ((RD_cost_r < min_mcost) and (process_best_cost = '0')) then
                best_pos_x <= exploracion_x;
                best_pos_y <= exploracion_y;
                min_mcost <= RD_cost_r;
            end if;
        end if;
    end process;
```

Figura 4.52. Código VHDL del cálculo de la función $J_{MOVIMIENTO}$ y comparación de costes para la actualización de las coordenadas del bloque con coste mínimo.

Tabla 4.23. Estadísticas de síntesis y realización post-P&R del componente de coste R/D.

Recurso/especificación	Valor
Slices	1359
Flip-flops	1625
4-input LUTs	1698
DSP48s	2
Puertas equivalentes	33704
Frecuencia (MHz)	103

4.3.4.5 Memoria local del macro-bloque actual.

Para almacenar localmente los MBs actuales, se necesita una memoria RAM de 2 puertos (RAM_{MBA}), con capacidad de 512 bytes y organización de 128×32 en el puerto A y de 32×128 en el puerto B. Con esta estructura, el puerto A se puede conectar en todo su ancho al bus OPB (32 bits) del procesador del sistema (MicroBlaze) y el puerto B puede proveer los 16 pixeles por ciclo de reloj que requiere la matriz de PEs.

De acuerdo a la tecnología utilizada (FPGA), se tienen 2 alternativas para la realización de la RAM_{MBA}: RAM distribuida y bloques de RAM (BRAM). La primera opción se construye con la lógica de los *slices* y se utiliza para almacenar pequeñas cantidades de datos, por lo que es muy rápida y escalable. La segunda se presenta como recurso dedicado, integrado en el silicio y se utiliza para almacenar grandes cantidades de datos. Una u otra alternativa es viable, ya que para los dos tipos de memoria, el dispositivo utilizado (XC4VFX60-10FF1152) dispone de amplios recursos (232 bloques de RAM de 18 Kb c/u, equivalentes a 4,276,224 bits, y hasta 404,480 bits de RAM distribuida). Su incorporación en el diseño no presenta diferencias, ya que ambas memorias se pueden crear e integrar con los mismos métodos: instanciación, inferencia o con el software *Coregen* incluido en la herramienta ISE.

Tomando como referencia la facilidad que ofrece el software EDK para especificar memorias BRAM como periférico de los procesadores MicroBlaze y

PowerPC, la disponibilidad de recursos dedicados que no consumen *slices* y la capacidad de los BRAM de satisfacer ampliamente las metas del diseño, con recursos como puertos duales verdaderos, en este ejercicio se utilizan BRAMs para la realización de la memoria de los MB actuales, sin demeritar la opción de usar RAM distribuida, que sigue siendo valida (como una observación, una memoria de dos puertos con organización 128×32 realizada en RAM distribuida consume 328 *slices*).

El periférico RAM_{MBA} consiste de la interfase al bus OPB, del controlador de memoria y de la memoria BRAM que satisface los requerimientos de capacidad de almacenamiento, organización de los puertos y ancho de banda (Figura 4.53). Para su incorporación al sistema, el controlador se agrega como componente IP en el ambiente de desarrollo EDK 8.2i y la memoria se realiza con el software *Coregen* (dual port block memory v6.3) del ISE 8.2i. De esta manera se tiene un buen control del diseño de la memoria, con el inconveniente de complicar su portabilidad. La tabla 4.24, muestra la descripción de las señales de E/S de la RAM_{MBA}, con los niveles activos de las señales de control de acuerdo al controlador y a los componentes del acelerador con los que interactúa.

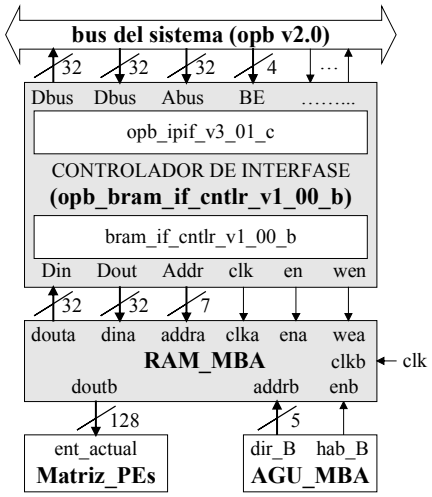


Figura 4.53. Memoria RAM_{MBA} con la interfase al bus (IPIF), el controlador de memoria y los componentes matriz_PEs y AGU_BMA.

Tabla 4.24. Descripción de las señales de E/S de la memoria RAM_{MBA}.

Señal	Tipo	Descripción
clka	E	Reloj, con operación en su borde de subida (puerto A)
ena	E	Habilitación memoria, activa en alto (puerto A)
wea	E	Habilitación de escritura, activa en alto (puerto A)
addra (6 downto 0)	E	Bus direcciones (puerto A)
dina (31 downto 0)	E	Datos de entrada puerto A
douta (31 downto 0)	S	Datos de salida (puerto A)
clkb	E	Reloj, con operación en su borde de subida (puerto B)
enb	E	Habilitación memoria, activa en bajo (puerto B)
addrb (4 downto 0)	E	Bus direcciones (puerto B)
doutb (127 downto 0)	S	Datos de salida (puerto B)

En las estadísticas de síntesis y realización post-P&R del periférico RAM_{MBA} (Tabla 4.25), se observa un bajo consumo de elementos hardware del dispositivo utilizado: 25 *slices* del controlador de interfase de la memoria (1.72 % del total) y 4

bloques RAMB16s de la memoria de MBs actuales (0.01 % del total), por lo que la solución con BRAMs es muy eficiente para los recursos y características de la FPGA Virtex-4 utilizada.

Tabla 4.25. Estadísticas de síntesis y realización post-P&R del periférico RAM_{MBA} con interfase al bus OPB.

Recurso/especificación	Valor
Slices	25
Flip-flops	35
4-input LUTs	16
RAMB16s	4
Frecuencia (MHz)	135

4.3.4.6 Memorias locales de la ventana de búsqueda.

La realización de las dos memorias RAM_{SW} parte del cumplimento total de los requerimientos de capacidad de almacenamiento, estructura, organización y velocidad de acceso que demanda la arquitectura IME, además de la facilidad de adaptarse como periférico a un bus de 32 ó 64 bits.

La capacidad de la memoria es función directa del tamaño de la SW. Para los valores máximos de los rangos de exploración horizontal y vertical considerados en este desarrollo (Ph = 56, Pv = 56), la memoria necesita almacenar 16 KB de datos (SW de 128×128 pixeles). Para conectar la memoria al bus del procesador y al acelerador IME simultáneamente, se requiere una estructura de dos puertos, con una organización que satisface la capacidad de almacenamiento: 4096×32 (bus de 32 bits) ó 2048×64 (bus de 64 bits) en el puerto A y 512×256 en el puerto B. En cuanto a velocidad, los procesos de lectura y de escritura se necesitan realizar en un borde de reloj. La conexión del puerto A a los buses de 32 ó 64 bits está en función del procesador y bus local necesario para la aplicación: MicroBlaze y su bus OPB o el PowerPC con su bus PLB, por lo que la memoria seleccionada debe ser configurable para adecuarse a uno u otro.

En el diseño del acelerador se proponen dos memorias RAM_{SW} estructuradas como 4 bloques SRAM de dos puertos de 32 bits de ancho, con una lógica que multiplexa los 4 bloques para conectarse al bus del procesador y que administra la lectura de ambos puertos para transferir 32 pixeles candidatos al acelerador hardware. Gracias a la tecnología de realización (FPGA Virtex-4) y sus herramientas software, esta topología de memoria se puede realizar muy eficientemente. En primer lugar, de las alternativas de memoria de estos dispositivos (distribuida y BRAMs), se seleccionan los bloques de RAM, debido a que son los mas adecuados para el nivel de almacenamiento de esta aplicación. En segundo lugar, las características de operación de la RAM_{SW} se pueden satisfacer con un módulo único de memoria de dos puertos independientes, con la capacidad y organización previamente mencionadas, utilizando la herramienta *Coregen* (Dual Port Block Memory v6.3) integrada en el ISE 8.2i.

Cada RAM_{SW} es un periférico del procesador, con su interfase al bus y su controlador, que se conecta a los componentes del acelerador que procesan la información almacenada (Figura 4.54), a través de sus señales de datos, direcciones y control (Tabla 4.26). La tabla 4.27 muestra las estadísticas de área y tiempo post-P&R

de una de las RAM_{SW} cuando se conecta al bus OPB, en donde se observa el uso significativo de bloques de memoria BRAM.

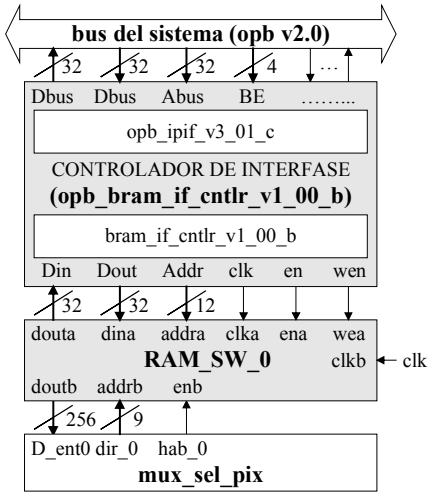


Figura 4.54. Memoria RAM_{sw} #0 con la interfase al bus OPB, el controlador de memoria y el componente mux_sel_pix que controla, direcciona y recibe los datos.

Tabla 4.26. Descripción de las señales de E/S del módulo de memoria de 16 KB y puerto dual.

Señal	Tipo	Descripción
clka	E	Reloj, con operación en su borde de subida (puerto A)
ena	E	Habilitación memoria, activa en alto (puerto A)
wea	E	Habilitación de escritura, activa en alto (puerto A)
addra (11 downto 0)	E	Bus direcciones (puerto A)
dina (31 downto 0)	E	Datos de entrada puerto A
douta (31 downto 0)	S	Datos de salida (puerto A)
clkb	E	Reloj, con operación en su borde de subida (puerto B)
enb	E	Habilitación memoria, activa en bajo (puerto B)
addrb (8 downto 0)	E	Bus direcciones (puerto B)
doutb (255 downto 0)	S	Datos de salida, (puerto B)

Tabla 4.27. Estadísticas de síntesis y realización post-P&R del periférico RAM_{SW} (interfase al bus OPB).

Recurso/especificación	Valor
Slices	25
Flip-flops	35
4-input LUTs	16
RAMB16s	8
Frecuencia (MHz)	135

4.3.4.7 Unidades generadoras de direcciones.

La AGU_{MBA} es el componente que habilita y direcciona la memoria RAM_{MBA} para leer los pixeles del MB actual y cargarlos en la matriz de PEs (Figura 4.55). Su operación comienza al recibir la señal de inicio (Tabla 4.28) desde la unidad de control y termina al transferir los 256 pixeles del MB actual, 16 ciclos de reloj después. La máquina Moore de tres estados y el contador binario descendente de 4 bits que la forman se codifican en VHDL mediante los procesos de decodificación del siguiente estado, actualización del estado actual y decodificación de salidas. Sus estadísticas post-

P&R (Tabla 4.29) muestran un componente altamente optimizado en área y velocidad. Cabe destacar que el vector de direcciones dir_B se forma de la concatenación de la entrada num_BMA (bit más significativo) con el contador binario de 4 bits y así acceder a uno de los dos MBs almacenados en memoria.

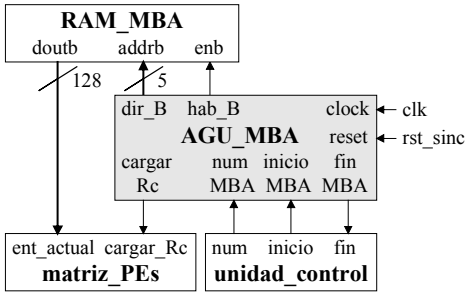


Figura 4.55. Interconexión de AGU_{MBA} con los componentes del acelerador hardware IME.

Tabla 4.28. Descripción de las señales de E/S de la unidad generadora de direcciones AGU_{MBA}.

Señal	Tipo	Descripción
clock	E	Reloj, con operación en su borde de subida
reset	E	Reset síncrono, activo en bajo
num_MBA	E	Número de MB actual (0 - #0, 1- #1)
inicio_MBA	E	Inicio de la operación de la AGU del MB actual, activa en bajo
fin_MBA	S	Fin de la operación de la AGU del MB actual, activa en bajo
cargar_Rc	S	Habilitación de la carga de los pixeles candidatos en los registros Rc de los PEs, activa en bajo
dir_B(4 downto 0)	S	Dirección de lectura de la RAM_{MBA} (puerto B)
hab_B	S	Habilitación de la memoria RAM_{MBA}, activa en bajo (puerto B)

Tabla 4.29. Estadísticas de síntesis y realización post-P&R de la AGU_{MBA}.

Recurso/especificación	Valor
Slices	9
Flip-flops	6
4-input LUTs	17
Puertas equivalentes	174
Frecuencia (MHz)	308

La unidad generadora de direcciones AGU_{SW} es el componente que administra la exploración de la SW, al direccionar la memoria RAM_{SW} y controlar el movimiento de los pixeles candidatos en la matriz de PEs (Figura 4.56), utilizando una máquina de estados Moore y un grupo de contadores. En el inicio de operaciones, este elemento recibe de la unidad de control los parámetros de operación y en el proceso de rastreo, envía las coordenadas del MB candidato al elemento RD_IME para su evaluación de costes. Al termino de la exploración, la AGU_{SW} indica el fin de operaciones con la señal *fin_SW* (Tabla 4.30).

La máquina de estados Moore se codifica algorítmicamente en VHDL, con dos procesos combinacionales y un proceso secuencial, mientras que los contadores se codifican en un proceso secuencial para su incremento/decremento, con las señales de control definidas en función del siguiente estado de la máquina. Las estadísticas post-P&R (Tabla 4.31) muestran un componente de baja velocidad debido al reducido nivel

de segmentación en la lógica de la ecuación de direccionamiento, que evita una alta latencia en la exploración de la SW. Para cumplir con la meta de velocidad, se necesitará aplicar en el sistema completo las acciones de depuración de la herramienta ISE utilizada.

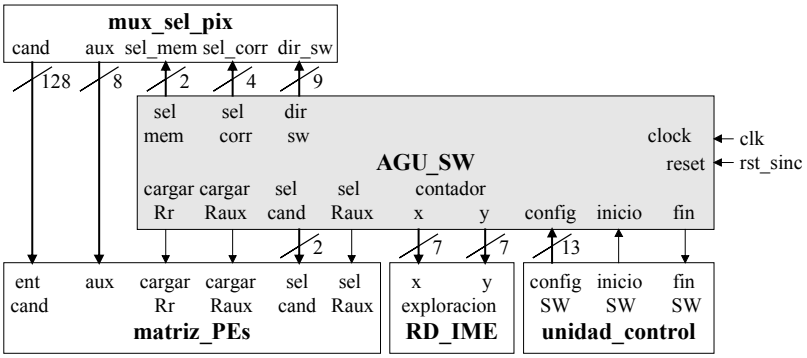


Figura 4.56. Componente AGU_{SW} y su interfase con sus elementos vecinos.

Tabla 4.30. Descripción de las señales de E/S de la unidad generadora de direcciones AGU_{SW}.

Señal	Tipo	Descripción
clock	E	Reloj, con operación en su borde de subida
reset	E	Reset síncrono, activo en bajo
config(12 downto 0)	E	Parámetros de configuración: número de RAM_{SW} y valores de Ph y Pv
inicio_SW	E	Inicio de la operación de la AGU_{SW}, activa en bajo
fin_SW	S	Fin de la operación de la AGUsw, activa en bajo
sel_mem(1 downto 0)	S	Número de RAM_{SW} a utilizar en este ciclo (00- #0, 01- #1, 1x- ninguna)
sel_corr(3 downto 0)	S	Desplazamiento de la ventana del selector de pixeles
dir_sw(8 downto 0)	S	Dirección de lectura de la RAM_{SW}
cargar_Rr	S	Habilitación de la carga de los pixeles candidatos en los registros Rr de los PEs, activa en bajo
cargar_Raux	S	Habilitación de la carga de los pixeles candidatos en los registros auxiliares de la matriz de PEs, activa en bajo
sel_cand(1 downto 0)	S	Selección del pixel candidato superior (00), derecho (01) o inferior (1x) a cargar en los registros Rr de los PEs, activa en bajo
sel_auxiliar	S	Selección del píxel candidato auxiliar superior (0) o inferior (1)
contador_x(6 downto 0)	S	Coordenada x con respecto al origen de la SW del MB candidato actual
contador_y(6 downto 0)	S	Coordenada y con respecto al origen de la SW del MB candidato actual

Tabla 4.31. Estadísticas de síntesis y realización post-P&R de la AGU_{SW}.

Recurso/especificación	Valor
Slices	83
Flip-flops	46
4-input LUTs	152
Puertas equivalentes	1834
Frecuencia (MHz)	87.5

4.3.4.8 Multiplexor de memorias y selector de pixeles.

El multiplexor de memorias y el selector de pixeles son los componentes que realizan la transferencia de pixeles desde las memorias RAM_{SW} hasta la matriz de PEs (Figura 4.57). El primer elemento selecciona una de las RAM_{SW} (#0 ó #1) según lo establece el procesador local y el selector de pixeles permite una alineación en bytes del bloque de 32 pixeles que recibe de la RAM_{SW} seleccionada

La selección de memorias se realiza con un multiplexor 2 a 1 de 256 bits, el cual se codifica en VHDL con la sentencia algorítmica *case-when*. El selector de pixeles se realiza como una estructura tipo *barrel-shifter* al nivel de bytes, codificado igualmente en VHDL con sentencias *case-when*. La tabla 4.32 muestra las señales de E/S y la tabla 4.33 las estadísticas post-P&R, en donde se observa un pobre comportamiento en velocidad por la gran cantidad de niveles lógicos (6). Para cumplir las metas de diseño, se necesitarán aplicar las acciones de depuración de la herramienta ISE o resolver el problema con la segmentación del diseño, con el consiguiente aumento de latencia.

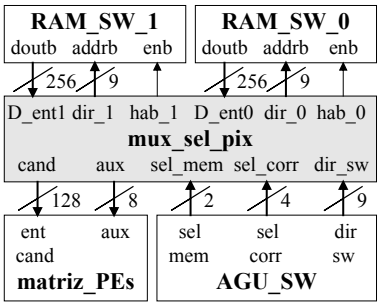


Figura 4.57. Componente mux_sel_pix y su interconexión con los elementos del acelerador IME.

Tabla 4.32. Descripción de las señales de E/S del multiplexor de memorias y selector de pixeles.

Señal	Tipo	Descripción
sel_mem(1 downto 0)	E	Número de RAM_{SW} a utilizar en este ciclo (00- #0, 01- #1, 1x-ninguna)
sel_corr(3 downto 0)	E	Desplazamiento de la ventana del selector de pixeles
dir_sw(8 downto 0)	E	Dirección de lectura de las RAM_{SW}
D_ent0(256 downto 0)	E	Datos de la RAM_{SW} #0
D_ent1(256 downto 0)	E	Datos de la RAM_{SW} #1
dir_0(8 downto 0)	S	Dirección de lectura de la RAM_{SW} #0
dir_1(8 downto 0)	S	Dirección de lectura de la RAM_{SW} #1
hab_0	S	Habilitación de la RAM_{SW} #0, activa en bajo
hab_1	S	Habilitación de la RAM_{SW} #1, activa en bajo
cand(127 downto 0)	S	16 Pixeles candidatos a la matriz de PEs
aux(7 downto 0)	S	Pixel auxiliar a la matriz de PEs

Tabla 4.33. Estadísticas de síntesis y realización post-P&R del multiplexor de memorias y selector de pixeles.

Recurso/especificación	Valor
Slices	538
4-input LUTs	1011
Puertas equivalentes	7410
Frecuencia (MHz)	69.62

4.3.4.9 Estadísticas del sistema completo.

La integración de los componentes del acelerador hardware IME permite aplicar en el conjunto el flujo de diseño del software ISE 8.2i. Para optimizar el diseño (máxima frecuencia de operación y área mínima), el primer paso consiste en la ejecución iterativa de los procesos de síntesis y realización con diferentes configuraciones en sus propiedades. Si con la aplicación intensiva de la herramienta no se logran los resultados esperados, se realizará un análisis del diseño para perfeccionar la codificación de las partes criticas, hasta cumplir con las expectativas marcadas.

La tabla 4.34 muestra las estadísticas post-P&R para varias configuraciones de la herramienta, desde aquella en la que se utilizan las propiedades por defecto de la síntesis y realización hasta en la que se logran los resultados deseados. Para todos los casos, se especifican restricciones globales de tiempo *period*, *pad to setup* y *clock to pad* de 10 ns, con un ciclo de trabajo del 50% en la señal de reloj. El análisis no incluye los procesadores, buses y memorias, ya que estos elementos se consideran externos al diseño. Ya que con los resultados finales se cumplen los objetivos de tiempo, no se requiere modificar el estilo de codificación VHDL del sistema.

Para evaluar la frecuencia de operación obtenida (101 MHz), en la tabla 4.35 se muestra la frecuencia en MHz que necesita la arquitectura propuesta en su realización en FPGA, para cumplir la estimación de movimiento de varios formatos de video y desplazamientos de exploración. Se observa que con el resultado alcanzado, el acelerador hardware puede procesar secuencias de video de la mayoría de los formatos, para valores de desplazamiento de 8 y 16 pixeles.

Tabla 4.34. Estadísticas de síntesis y realización post-P&R del acelerador hardware IME.

	Iteración				
Recurso/especificación	1	2	3	4	5
Slices	9542	9559	9561	9464	9525
Flip-flop	9725	9739	9770	9197	9478
4-input LUTs	14490	14485	14489	14311	14451
Puertas equivalentes	626212	626315	626590	623562	625890
Periodo (ns)	10.105	10.398	9.944	10.274	9.883
Offset before clock (ns)	7.835	6.599	5.801	5.695	8.638
Offset after clock (ns)	10.542	10.542	10.544	14.963	9.952

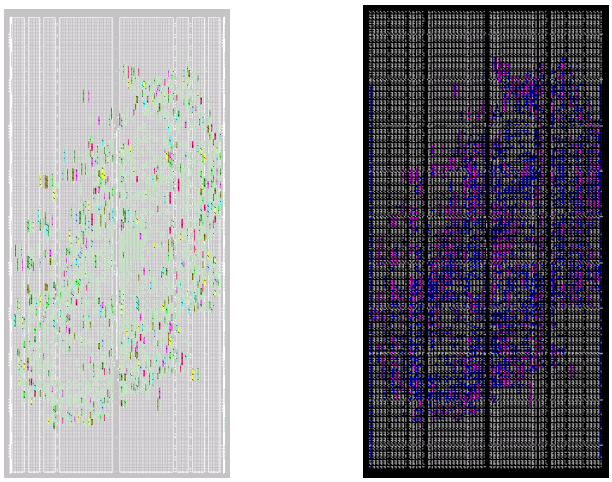
Iteración #1: Propiedades de síntesis y realización por defecto.
Iteración #2: Síntesis con esfuerzo de optimización alto, no balance de registros, realización por defecto.
Iteración #3: Síntesis con esfuerzo de optimización alto, si balance de registros, realización por defecto.
Iteración #4: Síntesis por defecto, realización con esfuerzo de mapeo alto, optimización lógica y global, retiming y nivel de P&R alto.
Iteración #5: Síntesis con esfuerzo de optimización alto, si balance de registros, realización con esfuerzo de mapeo alto, optimización lógica y global, retiming y nivel de P&R alto.

Tabla 4.35. Frecuencia de operación en MHz del acelerador hardware para procesar varios formatos de video y desplazamientos de exploración de la SW, realización en FPGA.

	Desplazamientos de exploración (Ph, Pv)					
Formato	(8,8)	(16,16)	(24,16)	(24,24)	(48, 32)	(56,56)
525 SD	12.879	45.279	66.663	98.415	256.527	518.319
625 SD	15.454	54.334	79.995	118.098	307.832	621.982
720p HD	34.344	120.744	177.768	262.44	684.072	1382.184
1080 HD	77.846	273.69	402.940	594.864	1550.56	3132.95

Otro resultado de interés es la latencia total. Antes de realización, el diseño presenta una latencia fija de 16 ciclos de reloj, por el proceso de carga en la matriz de procesamiento del MB actual y del primer MB candidato. En la realización con FPGA, este efecto aumenta 14 ciclos, debido a la segmentación que se necesita para alcanzar el comportamiento deseado. Aun cuando el incremento en latencia puede ser menor aplicando tecnologías que no requieren altos niveles de *pipeline*, como las ASIC, su efecto se minimiza por la misma operación segmentada, ya que no es necesario esperar a que termine totalmente un ciclo de procesamiento para iniciar el siguiente.

La Figura 4.58 muestra los resultados de emplazamiento y rutado de la arquitectura propuesta sobre la FPGA XC4VFX60, en donde se observa una gran complejidad en estas operaciones para la herramienta de síntesis y realización ISE 8.2i. El acelerador hardware se simuló con la herramienta ModelSim XE III 6.1e. Al no incluirse los buses y procesadores, se simularon las señales para escribir a los registros de configuración en la entrada de parámetros (Figura 4.59). Los datos a procesar se definieron como valores iniciales en las memorias RAM_{MBA} y RAM_{SW}. En la figura 4.60 se observan los resultados de la estimación IME (posición con respecto al origen de la SW, de los 41 bloques de menor coste), los cuales se validan con un nivel bajo en la señal *best_pos_valid*.



(a) (b)

Figura 4.58. Emplazamiento (a) y rutado (b) del sistema completo sobre Virtex-4 XC4VFX60.

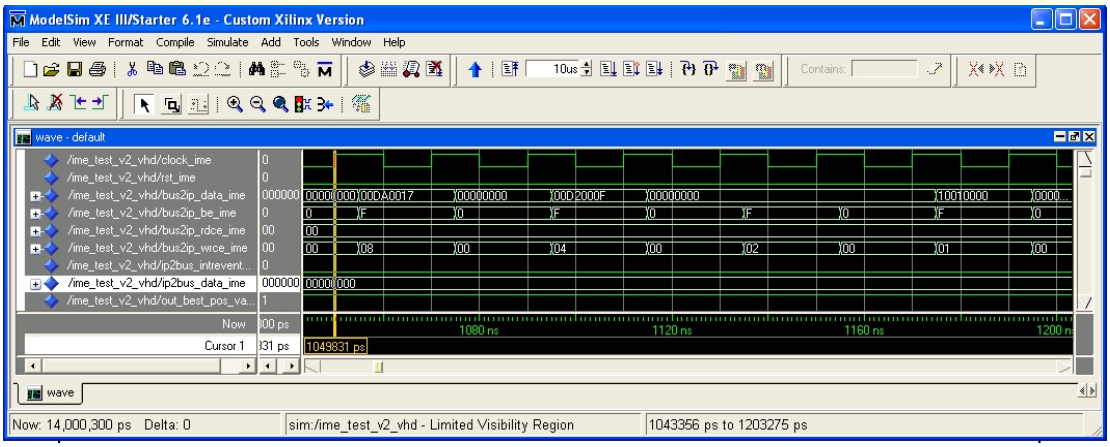


Figura 4.59. Simulación de la entrada de parámetros a través de los registros de configuración.

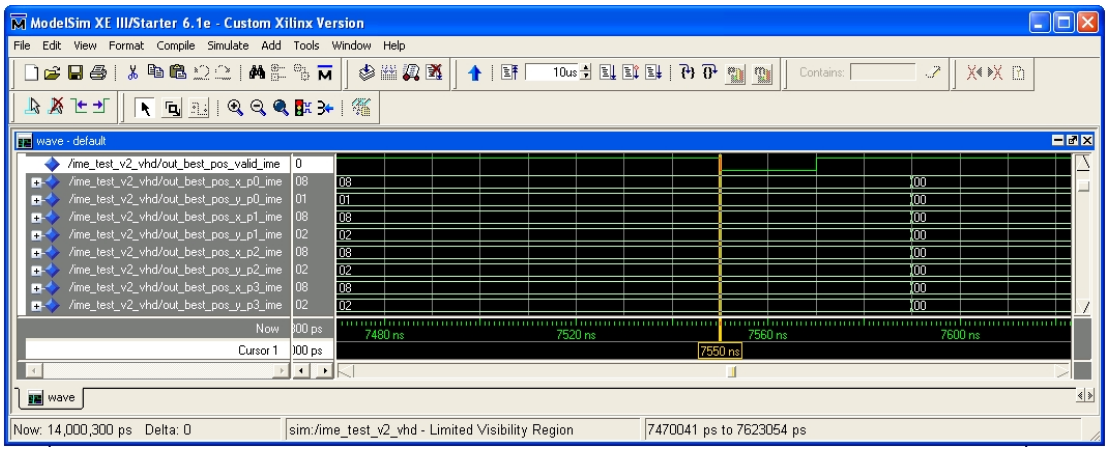


Figura 4.60. Resultados de simulación: MVs de los bloques de menor coste.

4.4 Arquitectura con ventana de búsqueda de forma geométrica variable.

Además de los métodos de reutilización y terminación temprana utilizados en el perfeccionamiento de las arquitecturas IME-FSBM, se tienen las siguientes técnicas para reducir el coste computacional y el ancho de banda, sin afectar significativamente la calidad de codificación: truncamiento de pixeles, decimación de pixeles y ajuste adaptable de los rangos de búsqueda. Para continuar con la optimización de los sistemas de estimación de movimiento, en esta sección se propone un método complementario para decrecer la complejidad de estas arquitecturas, reduciendo el área de la SW cuadrada, al utilizar SWs con otras formas geométricas (circular, rómbica, cruz, elíptica, etc.). Para satisfacer esta función, se utiliza la característica de la arquitectura propuesta, de modificar el desplazamiento vertical en cualquier momento de la exploración.

4.4.1 Técnicas de reducción de la complejidad FSBM.

4.4.1.1 Truncamiento de pixeles.

La técnica de truncamiento de pixeles reduce la carga computacional y el consumo de energía, modificando la resolución de los pixeles durante el cálculo de los vectores de movimiento, al recortar sus bits menos significativos en forma fija o adaptable. Los experimentos muestran que al truncar en forma fija 4 de 8 bits, se logran ahorros de 56% en el consumo de energía, con una reducción promedio del PSNR de 0.25 dB (secuencia foreman.qcif) [He00]; un truncamiento de 2 bits resulta en 23% de ahorro de energía con una degradación en la calidad de 0.07 dB. La aplicación de esta técnica en el diseño de un codificador H.264/AVC para formato HDTV720p @ 30 fps, con un truncamiento de 3 bits en la estimación de movimiento entero, ofrece excelentes resultados [Chen06b].

En sistemas con requerimiento de tasa en bits constante, el truncamiento adaptable puede actuar al nivel de cuadro o bloque en tiempo real, a partir del tamaño del paso de cuantización (QP). El valor de QP es modificado por el mecanismo de control de la tasa en bits en función del residuo entre el cuadro actual y el cuadro de

predicción. Normalmente, los escenarios complejos y con demasiada movilidad entre cuadros tienen un residuo alto y necesitan un QP grande, mientras que los de baja complejidad y movimientos pequeños tienen un residuo bajo y utilizan un QP pequeño. Al truncar cierto número de bits en los pixeles, el residuo se incrementa, por lo que probablemente el tamaño de QP aumente. Si QP aumenta considerablemente, se debe reducir el número de bits de truncamiento para reducir el error en el residuo. De otra manera, se puede mantener o incrementar el número de bits recortados, mientras QP no cambie o lo haga en forma poco significativa. Los experimentos en la secuencia foreman.qcif muestran un promedio de 5.14 bits truncados, con una reducción de 0.55 dB en PSNR con respecto a la secuencia original de pixeles de 8 bits, lo que permite un ahorro de 65% en el consumo de energía [He00].

4.4.1.2 Decimación de pixeles.

La técnica de decimación de pixeles disminuye tanto el coste computacional como el ancho de banda, al reducir el número de datos utilizados en el cálculo del mejor MV. En la comparación de un bloque actual con un bloque candidato, lo usual es evaluar N×N pixeles, pero ya que el algoritmo FSBM asume que todos los pixeles en un bloque se mueven a la par, se puede estimar el movimiento usando una fracción ordenada de los mismos (sub-muestra), en un proceso que preserve la exactitud de la estimación de movimiento y que reduzca el coste computacional por un factor entero. Por ejemplo, en [Liu93], un sub-muestreo de 4 a 1 reduce el coste computacional y el ancho de banda por un factor de 4, manteniendo la calidad de la estimación con una técnica de alternancia de los patrones de decimación. La arquitectura propuesta en [Chen06b] también reduce el coste del hardware por un factor de 2 a partir de un sub-muestreo 2 a 1, al evaluar solo 128 pixeles del total de 256 de cada MB (Figura 4.61), con excelentes resultados R/D, al compararse con otros diseños.

a	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b
b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**
a	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b
b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**
a	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b
b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**
a	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b
b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**
a	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b
b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**
a	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b
b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**
a	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b
b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**
a	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b
b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**	b	**a**

Figura 4.61. Patrones de sub-muestreo 2 a 1, a y b, para una reducción del coste computacional y ancho de banda por un factor de 2, al procesar 128 pixeles de los 256 de cada MB.

4.4.1.3 Ajuste adaptable de los rangos de búsqueda.

Las técnicas adaptables de los rangos de búsqueda para arquitecturas FSBM, evitan las operaciones de cómputo innecesarias y reducen el ancho de banda del sistema, sin alterar la eficiencia de codificación, al configurar dinámicamente el tamaño de la SW.

Una alternativa para su realización [Saponara04], utiliza los resultados parciales de estimación de movimiento (SAD_{min}) para determinar el valor optimo del desplazamiento de búsqueda, a partir de la correlación temporal y espacial en el campo de movimiento de los bloques vecinos, en un algoritmo llamado "seguidor de ventana". Los valores de SAD_{min} son comparados con un valor de umbral, para seleccionar el mejor tamaño de la SW para los siguientes bloques. Las pruebas muestran un ahorro del 73% en consumo de energía para la secuencia de video foreman.qcif.

Otros autores [Bailo04] proponen aplicar una exploración exhaustiva solo en las secuencias complejas de video y no en todos los cuadros y bloques por igual. Para ello, aproximan el nivel de movimiento con el algoritmo *blob-coloring*, en orden de decidir el tamaño de la SW para cada MB: los MBs donde se detecte mayor movimiento se procesan con SWs de gran tamaño; los MBs con poco o sin movimiento, se procesan con SW pequeñas. Los resultados muestran un ahorro en tiempo de ejecución del software de referencia del 42%, con una degradación promedio de 0.0625 dB en el PSNR.

4.4.2 Arquitectura con SW de forma variable.

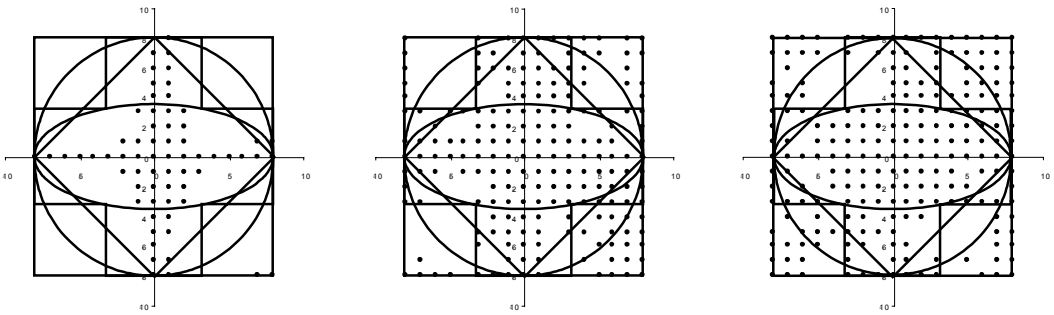
4.4.2.1 Factibilidad de nuevas formas geométricas de la SW.

En el algoritmo FSBM, el tamaño optimo de la SW puede ser determinado por un conocimiento a-priori sobre el movimiento de translación, el cual incluye la velocidad de movimiento de los objetos y el intervalo temporal entre cuadros consecutivos. Pero la actividad de movimiento es mas un concepto de probabilidad que uno estadístico. Algunos autores proponen métodos de estimación de movimiento, para adaptar el tamaño de la SW a la acción de cambio de posición y así reducir la complejidad computacional del algoritmo. El propósito de este apartado es mostrar que, al igual que el tamaño, la forma geométrica de la SW puede ser considerada para adaptarse al movimiento.

En primer lugar, la distribución de los MVs es normalmente centrada, con una dirección de movimiento horizontal [Kuhn99], por lo que la mayoría de los MVs globales se posicionan alrededor y a la izquierda y derecha del vector cero o del vector de predicción, sin una relación directa con una distribución siempre cuadrada o rectangular, lo que permite analizar varias opciones en la forma geométrica de la SW. Si las nuevas SWs contienen la principal parte de los MVs, no habrá una gran perdida de exactitud en la búsqueda.

Para ubicar el nivel de movimiento que se presenta en el procesamiento, la figura 4.62 muestra ejemplos de la distribución de los MVs globales en el plano XY, alrededor del vector de predicción del componente luma, en tres secuencias de video (news.qcif, carphone.qcif y foreman.qcif [YUVqcif]). Se consideran 50 cuadros con predicción IPPP..., a una tasa de 100 Kbps y desplazamiento de búsqueda p = 8 pixeles, bajo el modelo de prueba JM11 (perfil *baseline*, nivel 1.0, estimación fraccional desactivada, un cuadro de referencia y control de tasa habilitado). Se incluyen las formas de SWs original y propuestas: cuadrada (lado = 2p+1), rómbica (diagonal = 2p+1), circular (radio = p+½), cruz ($|x| \leq$ ½p ó $|y| \leq$ ½p) y elíptica (eje mayor = 2p+1, eje menor = p+½). Para una mayor definición, las figuras 4.63, 4.64 y 4.65 muestran la distribución XYZ de los vectores globales. El porcentaje que contiene cada una de las

SWs con respecto a la forma cuadrada se muestra en la tabla 4.36 para p = 8 y en la tabla 4.37 para p = 16.



(a) (b) (c)

Figura 4.62. Ejemplos de distribución de MVs globales en el plano del vector de predicción y formas propuestas de la SW, para las secuencias de video a) news.qcif, b) carphone.qcif y c) foreman.qcif.

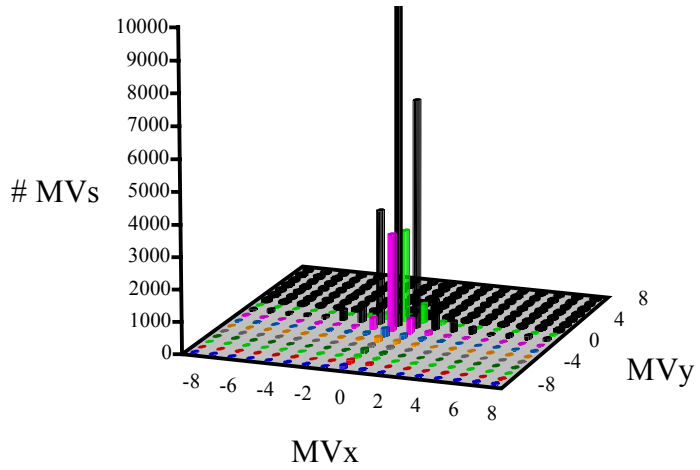


Figura 4.63. Distribución XYZ de los MVs alrededor del vector de predicción, secuencia news.qcif.

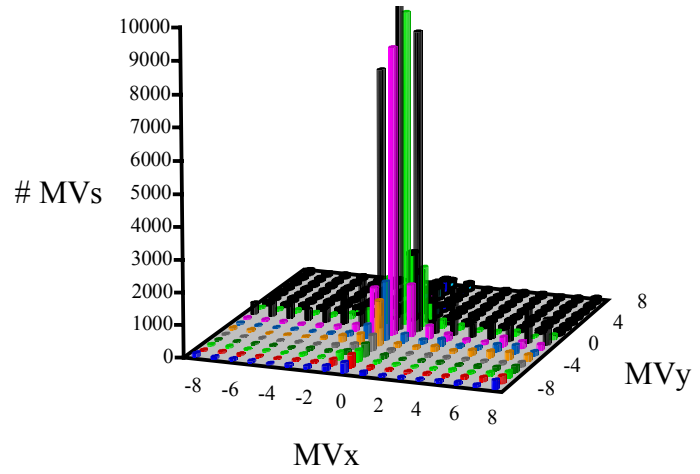


Figura 4.64. Distribución XYZ de los MVs alrededor del vector de predicción, secuencia carphone.qcif

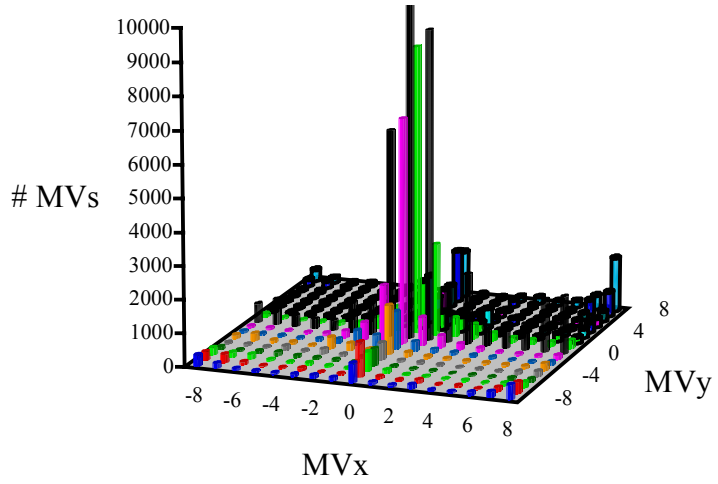


Figura.4.65. Distribución XYZ de los MVs alrededor del vector de predicción, secuencia foreman.qcif.

Tabla 4.36. Porcentaje de los MVs globales que contienen las SWs propuestas, para 50 cuadros de las secuencias de video evaluadas y p = 8.

Secuencia de video	MVs incluidos (%)				
	Cuadrada	Rómbica	Circular	Cruz	Elíptica
News	100	99.54	99.65	99.82	98.95
Carphone	100	95.09	96.56	97.82	92.17
Foreman	100	91.28	93.07	94.77	86.52
Promedio	100	95.30	96.43	97.47	92.55

Tabla 4.37. Porcentaje de los MVs globales que contienen las SWs propuestas, para 50 cuadros de las secuencias de video evaluadas y p = 16.

Secuencia de video	MVs incluidos (%)				
	Cuadrada	Rómbica	Circular	Cruz	Elíptica
News	100	98.99	99.05	99.72	98.52
Carphone	100	91.19	91.98	96.96	89.02
Foreman	100	89.40	90.43	95.55	86.97
Promedio	100	93.19	93.82	97.41	91.50

En los resultados anteriores, se observa que las nuevas SWs comprenden un promedio de 92.55% a 97.47% (p = 8) y de 91.5% a 97.41% (p = 16) de los MVs globales de la SW cuadrada, con un alto porcentaje para las secuencias de bajo nivel de movimiento, por lo que es muy factible considerarlas para reducir el coste computacional de la arquitectura IME-FSBM, en función del grado de actividad de cada MB. La reducción del coste de cómputo es directamente proporcional al menor número de MBs procesados con las formas de SWs no cuadradas (Tabla 4.38).

En segundo lugar, el uso de nuevas formas de SW no debe alterar significativamente la calidad de codificación, ya que si esto sucede, esta técnica no es una opción valida para optimizar el algoritmo FSBM. Para tener una referencia de la calidad de codificación con las SW propuestas, se hizo un análisis tasa/distorsión (R/D) de las tres secuencias de video previamente consideradas, manteniendo la configuración del modelo JM11 para p = 8. En los resultados (Figuras 4.66, 4.67 y 4.68), se observa gráficamente que no existen perdidas significativas en el PSNR para el rango de tasa en

bits evaluado, al cambiar la forma de la SW de cuadrada a rómbica, circular, cruz o elíptica.

Tabla 4.38. Número de MBs que se procesan en cada forma de SW con respecto a la forma cuadrada, para diferentes valores de desplazamiento de exploración p de la SW.

p	MBs procesados (%)				
(pixeles)	Cuadrada	Rómbica	Circular	Cruz	Elíptica
8	289 (100)	145 (50)	197 (68.17)	225 (77.85)	97 (33.56)
16	1089 (100)	545 (50)	797 (73.19)	833 (76.49)	393 (36.09)
24	2401 (100)	1201 (50)	1793 (74.68)	1825 (76.01)	893 (37.19)
48	9409 (100)	4705 (50)	7213 (76.66)	7105 (75.51)	3601 (38.27)
56	12769 (100)	6385 (50)	9845 (77.10)	9633 (75.44)	4913 (38.48)

Para una mejor revisión, las tablas 4.39, 4.40 y 4.41 muestran numéricamente la diferencia en dBs de las formas propuestas con respecto a la SW cuadrada. La diferencia promedio máxima (−0.173 dB) se tiene en la secuencia de mayor grado de movimiento (foreman.qcif) y con la forma de SW que evalúa menos MBs candidatos (elipse), por lo que se puede expresar que al utilizar una forma diferente a la cuadrada, la degradación de la calidad es mínima, para una reducción en el coste computacional de hasta 66 %.

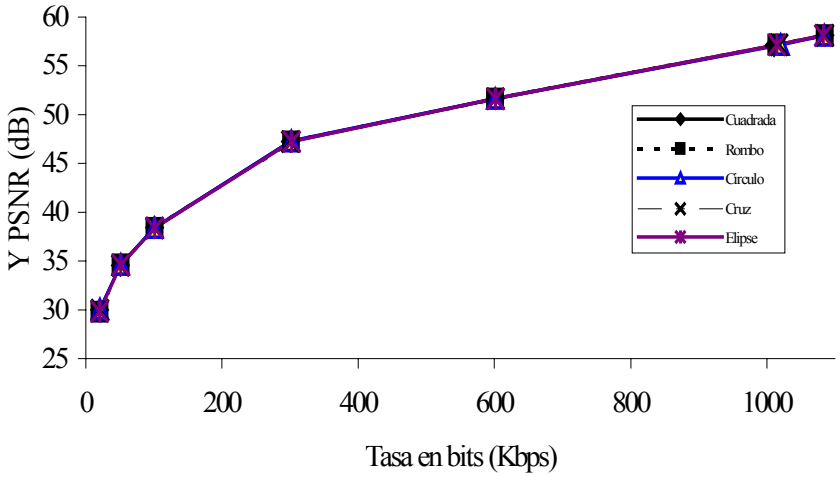


Figura 4.66. Comparación de tasa/distorsión, secuencia news.qcif.

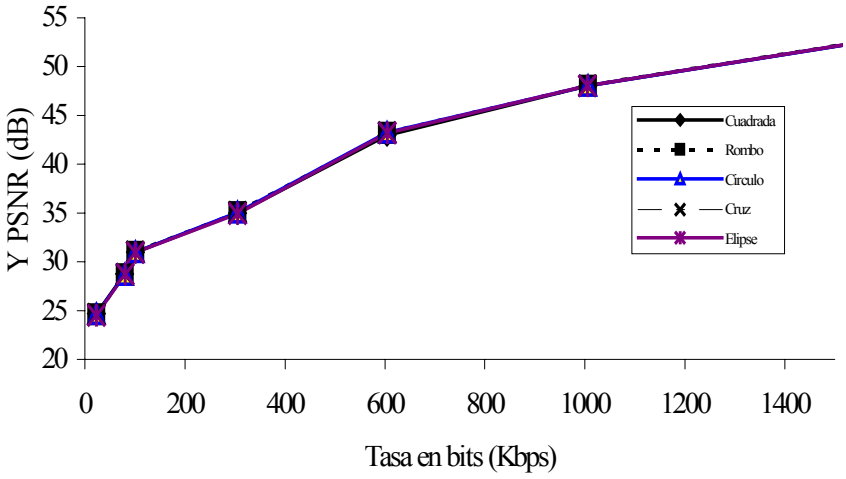


Figura 4.67. Comparación de tasa/distorsión, secuencia carphone.qcif.

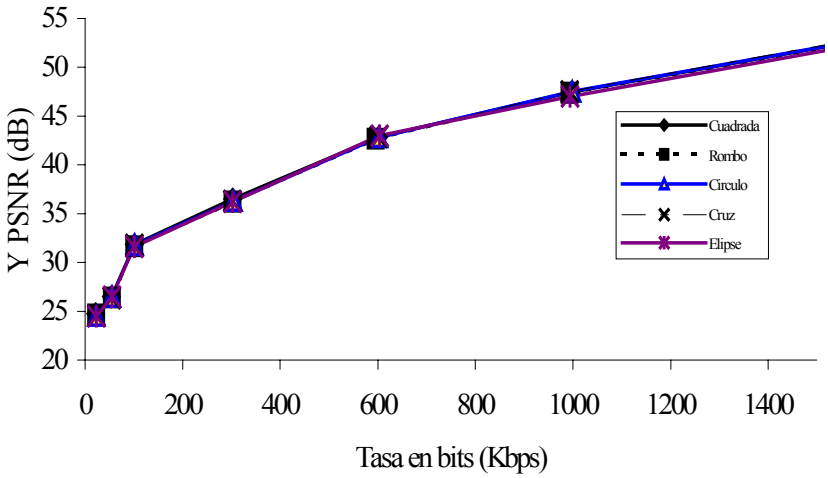


Figura 4.68. Comparación de tasa/distorsión, secuencia foreman.qcif.

Tabla 4.39. Diferencias en la comparación tasa/distorsión, secuencia news.qcif.

Tasa (Kbps)	Cuadrado (dB)	Diferencias (dB) Rómbica	Circular	Cruz	Elíptica
20	29.98	–0.17	0	0	–0.12
50	34.57	0	0	0	0
100	38.42	–0.01	–0.01	–0.02	–0.01
300	47.25	0	0	–0.01	0
600	51.65	–0.01	0	0	–0.03
1000	57.08	0	0	0	0
Promedio		–0.032	–0.0017	–0.005	–0.027

Tabla 4.40. Diferencias en la comparación tasa/distorsión, secuencia carphone.qcif.

Tasa (Kbps)	Cuadrado (dB)	Diferencias (dB) Rómbica	Circular	Cruz	Elíptica
20	24.69	–0.08	–0.04	–0.07	–0.16
50	28.78	–0.05	–0.11	–0.07	–0.04
100	31.02	0	0	–0.01	–0.06
300	35.12	–0.05	–0.04	–0.05	–0.17
600	43.23	–0.01	0	0	–0.03
1000	48.06	0	–0.01	0	0
Promedio		–0.032	–0.034	–0.034	–0.094

Tabla 4.41. Diferencias en la comparación tasa/distorsión, secuencia foreman.qcif.

Tasa (Kbps)	Cuadrada (dB)	Diferencias (dB) Rómbica	Circular	Cruz	Elíptica
20	24.795	–0.14	–0.25	–0.23	–0.29
50	26.48	–0.1	–0.02	–0.13	–0.01
100	31.84	–0.21	–0.06	–0.05	–0.23
300	36.48	–0.27	–0.19	–0.15	–0.26
600	42.86	–0.21	–0.03	–0.01	–0.13
1000	47.48	–0.09	0	0	–0.12
Promedio		–0.17	–0.091	–0.095	–0.173

Como otro punto de análisis, la figura 4.69 y la tabla 4.42 muestran la comparación tasa/distorsión para la secuencia foreman.qcif, cuando las formas de SW incluyen el mismo número de MBs candidatos (361). Esta evaluación busca definir si las formas no cuadradas de SW mejoran la calidad de codificación, cuando el nivel de complejidad computacional es el mismo. En los resultados se observa que para estas condiciones de prueba, la variación en PSNR con respecto a la forma cuadrada es mínima, por lo que no es muy adecuado alterar la regularidad del algoritmo al considerar otras formas de SW cuando no se tienen beneficios que lo compensen.

Un tercer elemento de estudio, es la complejidad en la estructura y control que puede presentar un estimador de movimiento entero, al adaptar la región de búsqueda a nuevas formas de SW. Si bien el algoritmo FSBM presenta una gran regularidad en el flujo de datos, el cambio geométrico de la forma de la ventana puede producir cuellos de botella en los accesos a memoria. La solución es diseñar una arquitectura que cumpla adecuadamente con la operación deseada y que en el cambio de forma de SW, no altere sus mediciones de eficiencia, latencia, velocidad de procesamiento y regularidad de direccionamiento. Es por ello que en la siguiente sección se presenta la arquitectura que desarrolla este nuevo método de optimización.

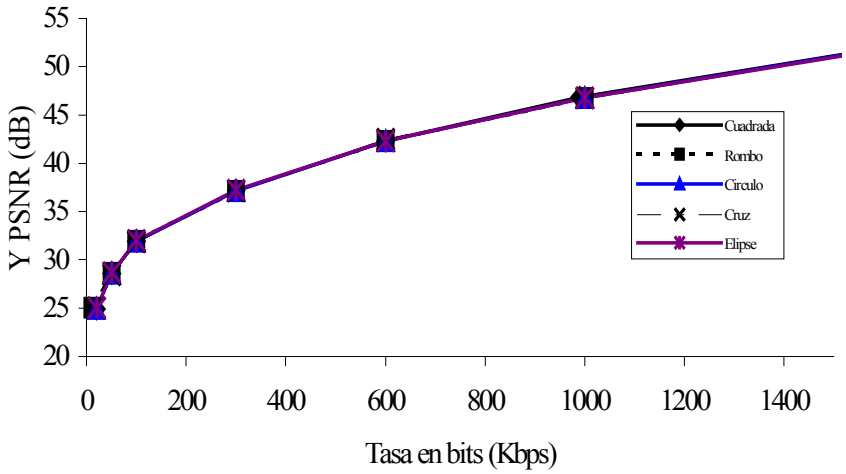


Figura 4.69. Comparación tasa/distorsión de 5 formas geométricas de SW para la misma complejidad computacional (361 MBs), secuencia foreman.qcif.

Tabla 4.42. Diferencias en la comparación tasa/distorsión para un mismo nivel de complejidad computacional, secuencia foreman.qcif.

Tasa (Kbps)	Cuadrada (dB)	Diferencias (dB) Rómbica	Circular	Cruz	Elíptica
20	24.89	+0.15	+0.06	+0.16	+0.16
50	28.60	+0.05	0	–0.13	+0.06
100	31.91	+0.07	0	–0.06	+0.09
300	37.16	–0.01	–0.03	–0.03	+0.05
600	42.30	+0.01	–0.01	0	–0.02
1000	46.83	–0.09	–0.05	–0.04	–0.04
Promedio		+0.03	–0.005	–0.016	+0.05

4.4.2.2 Diseño de la arquitectura.

En la primera sección de este capítulo se documentó una arquitectura con una ruta de datos muy eficiente, compuesta por un multiplexor y selector de pixeles, una matriz de 16×16 PEs, un árbol sumador y un módulo R/D, junto con la unidad de control y las unidades de manejo de memoria. Este sistema ejecuta el algoritmo IME-FSBM utilizando una estrategia de exploración tipo serpenteo, que rastrea la SW a una tasa de 1 MB/ciclo, sin consumir ciclos extras de reloj, gracias a la estructura de la matriz de procesamiento.

Además de 256 PEs, la matriz de procesamiento se compone de 16 registros auxiliares, los cuales siempre respaldan los pixeles necesarios para completar al MB candidato en el cambio de columna de la exploración tipo serpenteo. Ya que este cambio puede ser ejecutado en cualquier momento (Figura 4.70), el valor del desplazamiento vertical Pv se considera variable. Por otro lado, la unidad de manejo de memoria AGU_{SW} está diseñada alrededor de cuatro contadores binarios: contador de renglones, contador de bloques, contador de columnas y contador de desplazamiento; la ecuación de direcciones es función de los primeros dos contadores y del desplazamiento horizontal de exploración Ph, pero no de Pv. El desplazamiento vertical de búsqueda solo define los límites superior e inferior del contador de renglones en la lógica de operación de la AGU_{SW}.

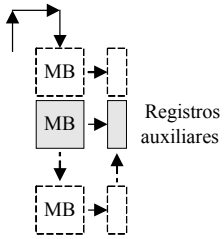


Figura 4.70. Cambio de columna en la exploración tipo serpenteo, con el apoyo de los registros auxiliares.

Los conceptos anteriores muestran que Pv puede ser regulado fácilmente y que en general, la arquitectura previa incluye de una manera natural la función de cambiar la forma de la SW, al tener la capacidad de modificar en cualquier momento el avance de la exploración vertical. Esto permite utilizar, con mínimos cambios, el diseño del acelerador IME-FSBM para explorar diferentes formas de SWs (Figura 4.71). En este apartado se documentan los cambios necesarios para alcanzar este objetivo, dejando el desarrollo del sistema como propuesta de trabajo futuro.

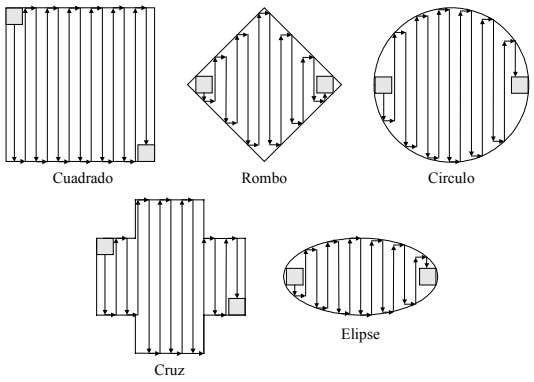


Figura 4.71. Exploración tipo serpenteo de las formas de SW propuestas.

En el acelerador hardware propuesto, el procesador del sistema recibe la información de las diferentes etapas de codificación, por lo que es el indicado para definir la forma de SW mas adecuada del siguiente MB, a partir de valores como QP, coste R/D o promedio de MVs de los últimos MBs actuales estimados. Una vez determinada, el procesador transmite al acelerador IME la información de la forma de SW en el registro de configuración 0 (Figura 4.72).

MSB 31 … 22	21 … 19	18	17 16	15 14	13 … 8	7 6	5 … LSB 0
auxiliar	SW	DV	MBA	aux	Pv	aux	Ph
rst: 0000000000	000	1	10	00	000001	00	001000

Ph Desplazamiento de exploración horizontal. Valores: 8, 16, 24, 32, 40, 48 y 56.
Pv Desplazamiento de exploración vertical. Valores: 1, 2, 3, 4, 5,....., 56.
MBA 00 Posición del MB actual en la memoria RAM_{MBA}: 0.
01 Posición del MB actual en la memoria RAM_{MBA}: 1.
1x Posición del MB actual no valida.
DV 0 Datos y parámetros validos en los registros 0 a 4.
1 Datos y parámetros no validos o ya leídos.
SW 000 Cuadrada/rectangular.
001 Rómbica.
010 Circular.
011 Cruz.
1xx Elíptica.

Figura 4.72. Nueva especificación del registro de configuración 0, con la información de la forma de la SW para el siguiente MB actual.

La arquitectura IME procesa la forma de SW en un nuevo módulo VHDL (Figura 4.73). Con este elemento extra, el componente AGU_SW ya no recibe un valor fijo de desplazamiento vertical Pv, sino que para cada valor del rango de exploración horizontal [0,2Ph], el módulo "control_forma_SW" calcula los límites inferior y superior de Pv y los envía como señales de salida (Tabla 4.43) a la unidad de manejo de memoria de la RAM_{SW}. Para esta función, se hace necesario modificar la máquina de estados de la AGU_SW, en los valores iniciales y de comparación del contador de renglones (Figura 4.74).

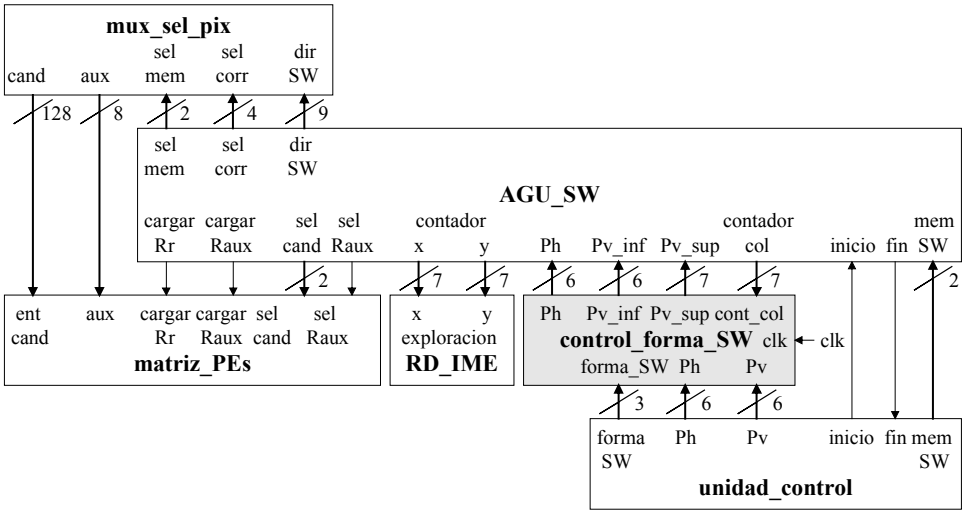


Figura 4.73. Ubicación del módulo "control_forma_SW" en el diseño IME-FSBM.

Tabla 4.43. Descripción de las señales de E/S del módulo de control de forma de la SW.

Señal	Tipo	Descripción
clk	E	Reloj, con operación en su borde de subida
forma_SW(2 downto 0)	E	Forma SW: 000-cuadrada/rectangular, 001- rómbica, 010-circular, 011-cruz y 1xx-elíptica
Ph(5 downto 0)	E	Desplazamiento horizontal de exploración (constante)
Pv(5 downto 0)	E	Desplazamiento vertical de exploración (constante)
cont_col(6 downto 0)	E	Contador de columnas
Ph(5 downto 0)	S	Desplazamiento horizontal de exploración (constante)
Pv_inf(5 downto 0)	S	Límite inferior de exploración vertical (variable)
Pv_sup(6 downto 0)	S	Límite superior de exploración vertical (variable)

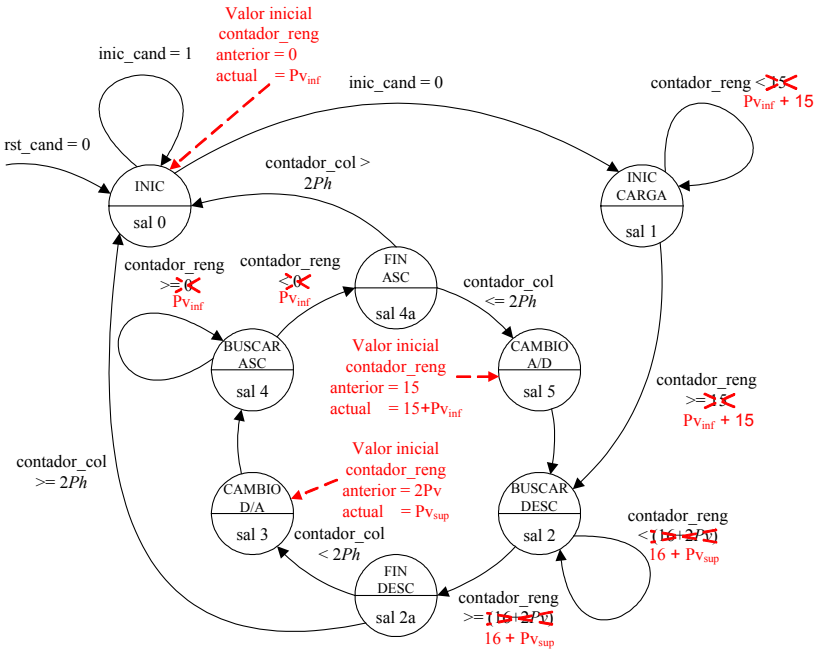
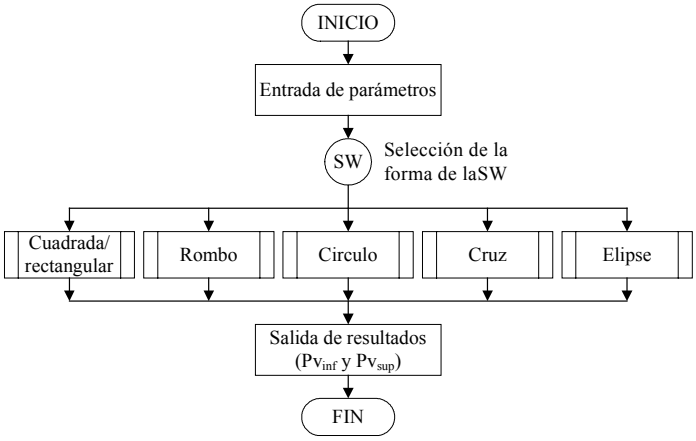


Figura 4.74. Modificaciones de la FSM del componente AGU_SW, para soportar la función de forma de SW variable.

El nuevo elemento recibe la información de la forma de SW, los valores originales de Pv y Ph y el valor actual del contador de columnas, para calcular los limites Pv_inf y Pv_sup con la lógica y el código VHDL que se exhiben en la figura 4.75. Las estadísticas post-P&R se muestran en la tabla 4.44.

En el programa se utiliza una raíz cuadrada de 12 bits (sqrt(contador_col×(2Pv – contador_col))), para calcular los límites de las formas circular y eliptica, que es generada en el ambiente *Coregen* del ISE8.2i con el software CORDIC v3.0. En su generación se elije el modo *pipeline* optimo, lo que incluye una latencia de 5 ciclos. Para evitar que este retardo inicial se agregue a la latencia total, la raíz cuadrada se calcula en el tiempo de la carga inicial de la matriz de PEs. Si la forma de SW es circular o eliptica, los valores de Ph y Pv, junto con un contador binario de 7 bits (0 a 2Ph), alimentan al nucleo generado y los resultados se almacenan en una memoria RAM distribuida de dos puertos y lectura asíncrona. En el programa de control de forma ya no se calcula esta operación aritmética, ahora solo se lee la memoria SQRT_RAM, direccionandola con el valor presente del contador de columnas.




```
sqrt_out := SQRT_RAM(conv_integer(contador_col_in));

case forma_SW_in is
  when "000" =>  -- CUADRADA.
    Pv_inf  := "00000000";
    Pv_sup := '0'&Pv_in&'0';
  when "001" =>  -- RÓMBICA.
    if (contador_col_in <= ('0'&Ph_in)) then
      Pv_inf  := ("00"&Pv_in) – ('0'&contador_col_in);
      Pv_sup := ("00"&Pv_in) + ('0'&contador_col_in);
    else
      Pv_inf := ('0'&contador_col_in) – ("00"&Pv_in);
      Pv_sup_a := ( "0000011" * Pv_in) – ("0000000"&contador_col_in);
      Pv_sup := Pv_sup_a(7 downto 0);
    end if;
  when "010" =>  -- CIRCULAR.
    if (contador_col_in = "0000000") then
      Pv_inf  := "00"&Pv_in;
      Pv_sup := "00"&Pv_in;
    else
      Pv_inf  := ("00"&Pv_in) – ('0'&sqrt_out);
      Pv_sup := ("00"&Pv_in) + ('0'&sqrt_out);
    end if;
  when "011" =>  -- CRUZ.
    if ( ((contador_col_in&'0') >= ("00"&Ph_in))
         and ((contador_col_in&'0') < ("00000011"*("00"&Ph_in))) ) then
      Pv_inf  := "00000000";
      Pv_sup := '0'&Pv_in & '0';
    else
      Pv_inf := "000" & Pv_in(5 downto 1);
      Pv_sup_a := "0000011" * ("00"&Pv_in(5 downto 1) );
      Pv_sup := Pv_sup_a(7 downto 0);
    end if;
  when others => -- ELÍPTICA.
    if (contador_col_in = "0000000") then
      Pv_inf  := "00"&Pv_in;
      Pv_sup := "00"&Pv_in;
    else
      Pv_inf  := ("00"&Pv_in) – ("00"&sqrt_out(6 downto 1));
      Pv_sup := ("00"&Pv_in) + ("00"&sqrt_out(6 downto 1));
    end if;
end case;
```

Figura 4.75. Diagrama de flujo y código VHDL del módulo de control de forma de la SW.

Tabla 4.44. Estadísticas de síntesis y realización post-P&R del módulo de control de forma de la SW.

Recurso/especificación	Valor
Slices	192
Flip-flops	54
4-input LUTs	347
DSP48s	2
Puertas equivalentes	10461
Frecuencia (MHz)	100

La arquitectura con SWs de forma variable mejora el cumplimiento de los objetivos de diseño del acelerador IME-FSBM original, sin menoscabo de la calidad de codificación. Esto se puede evaluar con la reducción de la frecuencia de operación al procesar menor número de MBs en cada forma no cuadrada de SW (tablas 4.45 a 4.48). Se observa que la operación de esta arquitectura con independencia de la tecnología puede cubrir una gama mayor de formatos de video que el diseño de SW cuadrada, para los desplazamientos de exploración considerados.

Tabla 4.45. Frecuencia de operación en MHz de la arquitectura IME-FSBM para procesar varios formatos de video, con diversos desplazamientos de exploración, para SW de forma rómbica, operación independiente de la tecnología.

	Desplazamientos de exploración (Ph, Pv), SW rómbica				
Formato	(8,8)	(16,16)	(24,24)	(48, 48)	(56,56)
525 SD	6.48	22.68	49.248	191.160	259.200
625 SD	7.776	27.216	59.097	229.392	311.040
720p HD	17.28	60.48	131.328	509.760	691.200
1080 HD	39.168	137.088	297.677	1155.456	1566.720

Tabla 4.46. Frecuencia de operación en MHz de la arquitectura IME-FSBM para procesar varios formatos de video, con diversos desplazamientos de exploración, para SW de forma circular, operación independiente de la tecnología.

	Desplazamientos de exploración (Ph, Pv), SW circular				
Formato	(8,8)	(16,16)	(24,24)	(48, 48)	(56,56)
525 SD	8.586	32.886	73.224	292.734	399.330
625 SD	10.303	39.463	87.869	351.280	479.196
720p HD	22.896	87.696	195.264	780.624	1064.880
1080 HD	51.898	198.777	442.598	1769.414	2413.728

Tabla 4.47. Frecuencia de operación en MHz de la arquitectura IME-FSBM para procesar varios formatos de video, con diversos desplazamientos de exploración, para SW en forma de cruz, operación independiente de la tecnología.

	Desplazamientos de exploración (Ph, Pv), SW cruz				
Formato	(8,8)	(16,16)	(24,24)	(48, 48)	(56,56)
525 SD	9.720	34.344	74.520	288.360	390.744
625 SD	11.664	41.213	89.424	346.032	468.893
720p HD	25.920	91.584	198.720	768.960	1041.984
1080 HD	58.752	207.590	450.432	1742.298	2361.830

Tabla 4.48. Frecuencia de operación en MHz de la arquitectura IME-FSBM para procesar varios formatos de video, con diversos desplazamientos de exploración, para SW de forma elíptica, operación independiente de la tecnología.

	Desplazamientos de exploración (Ph, Pv), SW elíptica				
Formato	(8,8)	(16,16)	(24,24)	(48, 48)	(56,56)
525 SD	4.536	16.524	36.774	146.448	199.584
625 SD	5.443	19.829	44.129	175.738	239.501
720p HD	12.096	44.064	98.064	390.528	532.224
1080 HD	27.418	99.878	222.278	885.197	1206.374

4.5 Conclusiones.

A continuación se presenta un sumario de los tópicos más relevantes del diseño y realización de la arquitectura propuesta y las modificaciones de la misma para su aplicación en el cambio de la forma geométrica de la SW. Se incluyen los conceptos, métodos y técnicas aplicadas, así como los alcances de las mismas en la realización del sistema de estimación de movimiento con resolución de pixeles enteros.

4.5.1 Diseño IME-FSBM.

Se ha diseñado una arquitectura que satisface los requerimientos de estimación de movimiento con resolución de pixeles enteros del estándar H.264/AVC, utilizando como componentes principales un selector de pixeles, una matriz de 16×16 PEs, un árbol sumador y un módulo R/D, elementos diseñados bajo la premisa de cumplir los objetivos de diseño (pixeles de 8 bits, formato de video de hasta 1080 HD, velocidad de 30 fps, 1 cuadro de referencia y desplazamiento de exploración máximo de 56 pixeles), con la mayor eficiencia posible.

Las actividades previas al diseño, consistieron en el análisis del software de referencia JM11 y el estudio del estado del arte de las arquitecturas de estimación de movimiento. De acuerdo al análisis del modelo de referencia, se precisaron algunas modificaciones en su operación secuencial, para que sus algoritmos sean factibles de realizar en arquitecturas segmetadas-paralelas, sin sacrificar la calidad de compresión. El estudio del estado del arte proporcionó las características de estructura y operación que predominan en estos diseños, las cuales se resumen en los siguientes puntos:

* La velocidad de procesamiento se optimiza reduciendo al mínimo la latencia, eliminando los ciclos falsos y ejecutando el proceso de comparación de bloques en el menor tiempo posible con una arquitectura altamente paralela.

* El nivel de paralelismo debe ser proporcional al ancho en bytes y al ancho de banda de la memoria de búsqueda.

* En las arquitecturas cuya operación es independiente al tamaño de la SW, los valores de las mediciones de diseño pueden ser constantes en toda la gama de rangos de exploración.

* Mientras más elementos pasivos se tienen, mayor puede ser la degradación de las características de operación al cambiar el tamaño de la SW.

* El tamaño del *buffer* de referencia es inversamente proporcional al ancho de la memoria de la SW, al ancho de banda de E/S, a la flexibilidad de la arquitectura ante cambios en el tamaño de la SW y al nivel de reutilización de datos.

* El uso de memorias locales optimiza el ancho de banda de E/S del sistema, cuando el diseño necesita acceder repetidamente a determinados pixeles.

* La reutilización de datos se aplica para reducir la complejidad computacional y el ancho de banda de las memorias locales.

* La regularidad en el direccionamiento de los pixeles candidatos depende directamente de la estructura y administración de la memoria, del acoplamiento entre la memoria y la matriz de PEs y de la estrategia de exploración de la SW.

Los conceptos anteriores se consideraron como guías generales para definir las características estructurales y de operación de la arquitectura propuesta:

* Máxima uniformidad en el flujo de datos.

* Procesamiento de cada bloque candidato en un ciclo de reloj.

* Cero ciclos falsos.

* Latencia mínima.

* Rendimiento independiente del tamaño de la SW.

* Bajo número de elementos pasivos.

* Alto grado de paralelismo.

* Reutilización de datos a nivel local.

* Alta regularidad en el direccionamiento de los pixeles candidatos.

El diseño utiliza una estrategia de exploración de la SW tipo serpenteo, con un grupo de registros auxiliares que simplifican el cambio de columna y un acceso a la matriz de PEs por la parte superior e inferior, lo que permite un flujo de datos muy uniforme.

El estimador de movimiento presenta una alta velocidad de operación, debido al procesamiento de cada MB candidato en un ciclo de reloj, sin necesidad de ciclos extras y a un bajo valor de latencia (16 ciclos). Esto se logró con un acceso de 32 bytes a la memoria de la SW y al flujo eficiente de datos a la entrada y dentro de la matriz de PEs.

La arquitectura mantiene sus valores de medición en el cambio de los rangos de exploración, ya que su operación es independiente del tamaño de la ventana de búsqueda. Esta flexibilidad se debe principalmente a la estructura y tamaño del *buffer* de referencia.

El acelerador hardware utiliza una estructura altamente paralela de 16×16 PEs, alimentada por memorias con suficiente ancho de banda, que proporcionan la tasa de datos que necesitan los elementos procesadores.

El sistema de tamaño de bloque variable reusa los resultados de distorsión de los bloques 4×4 en el cálculo de la distorsión de los bloques mayores. Esto representa un esquema de reutilización para reducir el coste computacional del módulo de estimación.

El ancho de banda de las memorias se reduce con la planeación cuidadosa de la secuencia de datos, el uso apropiado de memorias locales y la aplicación de técnicas de reutilización de pixeles entre MBs candidatos adyacentes, en ambas direcciones de exploración.

El uso del selector de pixeles, que evita acceder a la memoria al nivel de bytes, así como el control de la estrategia de exploración, permiten una alta regularidad en el direccionamiento de los datos, a partir de una ecuación de direcciones simple.

La arquitectura propuesta, al igual que los trabajos presentados en [Zhang05], [Huang03] y [Chen06a], ofrece valores casi ideales de velocidad de procesamiento, latencia y eficiencia, con un rendimiento independiente del tamaño de la SW, como los diseños de Huang et al. y Chen et al. y sin problemas estructurales para los rangos de búsqueda considerados, como la arquitectura de Zhang et al. Esto implica que el nivel de cumplimiento de los objetivos de diseño (pixeles de 8 bits, formato de video de hasta 1080 HD, velocidad de 30 fps, 1 cuadro de referencia y desplazamiento de exploración máximo de 56 pixeles) solo es función de la frecuencia de operación del estimador y por lo tanto de la tecnología de realización. Como una referencia, la tabla 4.10 (sección 4.3.3.8) muestra la frecuencia de operación de la arquitectura IME-FSBM para satisfacer el procesamiento de diversos formatos de video con varios valores de desplazamiento de exploración, en una operación independiente de la tecnología.

4.5.2 Realización IME-FSBM.

El acelerador hardware se realizó programando en VHDL la FPGA XC4VFX60-10 de la familia Virtex-4 de Xilinx. Para optimizar el uso del dispositivo en la escritura del código RTL, se consideraron técnicas de diseño para comportamiento, tales como la selección del tipo de *reset* (activo en alto, en bajo o sin *reset*), el uso de técnicas *pipeline* para la reducción del número de niveles lógicos, la minimización de código anidado tipo *if-else*, etc. La aplicación de estas técnicas, junto con el uso adecuado de las herramientas de Xilinx para síntesis (ISE 8.2i, sintetizador XST), diseño embebido (EDK 8.2i) y simulación (ModelSim Xilinx edition-III v6.1e), permitieron alcanzar 100 MHz en la frecuencia del reloj, con un bajo coste en área.

La selección de los procesadores y buses local y del sistema se basó principalmente, en el cumplimiento de la tasa de transferencia de datos, entre las memorias externas y locales, para las condiciones máximas de operación de la arquitectura (formato 1080 HD y desplazamientos de búsqueda de 56 pixeles).

En la sección del sistema, se presenta una tasa de transferencia de hasta 63.65 MBytes/s, por lo que se utiliza un procesador MicroBlaze y su bus OPB, ya que con su tasa típica de 167 MBytes/s, cumple satisfactoriamente con las condiciones de

operación, con la ventaja de que son elementos definidos por software que pueden integrarse en la mayoría de las FPGA de Xilinx.

En la etapa local, se tienen las alternativas del procesador por software MicroBlaze y el bus OPB o del procesador hardware PowerPC y el bus PLB. Considerando las tasas típicas de transferencia de los buses OPB y PLB (167 y 533 MBytes/s respectivamente), con la ayuda de la tabla 4.12 (sección 4.3.4.1), se puede determinar el formato y el desplazamiento de exploración que cubre cada par procesador/bus. Se observa que con los valores de tasa típicos se está lejos de satisfacer el formato 1080 HD y desplazamiento de 56 pixeles, por lo que se deben aplicar otros métodos de diseño, que permitan aproximar las tasas de transferencia a los valores máximos teóricos de cada bus (OPB-500 MBytes/s, PLB-1600 MBytes/s) y así satisfacer un mayor número de formatos y rangos de exploración.

Los resultados de la aplicación del flujo de diseño del software ISE 8.2i en el sistema IME-FSBM especifican una frecuencia máxima de operación de 101 MHz y un área de 9525 *slices*. Para definir el alcance de estos resultados, en la tabla 4.35 (sección 4.3.4.9) se indica la frecuencia de operación que necesita la arquitectura realizada, para el procesamiento de varios formatos de video y desplazamientos de operación. Se observa que el acelerador hardware puede procesar secuencias de video con los formatos marcados solo para valores pequeños y medianos de desplazamiento, por lo que es necesario optimizar la realización del diseño para su operación a mayor frecuencia y/o aplicar técnicas de reducción del número de MBs procesados sin alterar la calidad de codificación.

4.5.3 Arquitectura con forma de SW variable.

Se propuso un método para decrecer la complejidad de las arquitecturas IME-FSBM, reduciendo el área de la SW cuadrada, al utilizar SWs con otras formas geométricas (circular, rómbica, cruz, elíptica, etc.). Para justificar que al igual que el tamaño, la forma de la SW puede ser considerada para adaptarse al movimiento, se realizaron los análisis de distribución de MBs y del rendimiento tasa/distorsión en tres secuencias de video QCIF, para cada forma de SW propuesta.

El primer estudio parte de la consideración de que la distribución de los MVs globales es normalmente centrada, con una dirección de movimiento horizontal, alrededor del vector de predicción. Los resultados del análisis muestran que las nuevas SWs comprenden un promedio de 92.55% a 97.47% (desplazamiento de búsqueda p = 8) y de 91.5% a 97.41% (p = 16) de los MVs globales de la SW cuadrada.

El segundo estudio toma en cuenta que las nuevas formas de SWs no deben alterar significativamente la calidad de codificación, ya que si esto sucede, esta propuesta no es una opción valida para optimizar el algoritmo FSBM. Los resultados del análisis indican perdidas promedio máximas de –0.173 dB en PSNR, para el rango de tasa en bits evaluado, al cambiar la forma de la SW de cuadrada a rómbica, circular, cruz o elíptica.

Los resultados anteriores indican la factibilidad de considerar otras formas de SW en función del grado de actividad de cada MB, ya que las nuevas SWs contienen la principal parte de los MVs globales, reduciendo el coste computacional de la

arquitectura IME-FSBM al evaluar menos MBs candidatos (Tabla 4.38, sección 4.4.2.1), con una perdida mínima en la calidad de codificación.

Para la realización de la arquitectura con forma de SW variable, se tomó en cuenta que el acelerador hardware de resolución entera tiene la capacidad de modificar en línea el avance de la exploración vertical. Esto permite utilizar, con mínimos cambios, el diseño IME-FSBM original para explorar diferentes formas de SWs. Los cambios necesarios son en el registro de configuración 0, para que el procesador del sistema indique al acelerador el tipo de SW a explorar, en el control de la AGU_{SW}, para que la exploración vertical se limite de acuerdo al contorno de la SW indicada y en la integración de un componente extra, que calcule los desplazamientos verticales variables.

En cuanto al cumplimiento de los objetivos de diseño (pixeles de 8 bits, formato de video de hasta 1080 HD, velocidad de 30 fps, 1 cuadro de referencia y desplazamiento de exploración máximo de 56 pixeles), el menor número de MBs que se procesan en cada forma de SW con respecto a la cuadrada reduce la frecuencia de operación del acelerador IME-FSBM para satisfacer estos objetivos. La operación de esta arquitectura con independencia de la tecnología cubre una gama mayor de formatos de video que el diseño de SW cuadrada, para los desplazamientos de exploración considerados (tablas 4.45 a 4.48, sección 4.4.2.2).

Capítulo **5**

Arquitecturas *Full-Search* de Estimación de Movimiento con Resolución de Pixeles Fraccionarios

5.1 Introducción.

El movimiento continuo de los objetos en una escena de video, puede ser evaluado con mayor exactitud al aumentar la resolución espacial del muestreo. Consecuentemente, se puede obtener una mejor codificación de video si en la etapa de estimación de movimiento los MVs apuntan a posiciones fraccionarias y no solo a enteras. Mientras mayor sea la resolución de la estimación, mejor será la compensación de movimiento, al disminuir el residuo entre el cuadro actual y el de predicción. La aplicación exacta de este precepto pudiese ser la solución a las necesidades modernas de codificación, a no ser por sus costes implícitos, como son el incremento del cómputo en la interpolación de pixeles y en la comparación de bloques, así como la penalización en la tasa, al manejar una mayor cantidad de bits para representar a los vectores de movimiento fraccionario.

En el establecimiento del grado de exactitud de la compensación de movimiento, se debe buscar un equilibrio entre complejidad de la estimación y eficiencia de compresión. Una compensación muy exacta utiliza menos bits para codificar el residuo pero más bits para codificar el campo de MVs, mientras que en una poco exacta, se requieren más bits para el residuo y menos bits para el campo de vectores. Además, al ir incrementando los pasos de resolución en la estimación, la ganancia en rendimiento disminuye exponencialmente y los costes computacionales aumentan de la misma forma. El proceso de inter-predicción fraccionaria del estándar H.264/AVC hace buen uso de este equilibrio, al utilizar una resolución de ¼ de pixel para la estimación de movimiento del componente luma y de $^1/_8$ para la estimación de los componentes croma.

En la estimación de movimiento fraccionario (FME), la exploración de toda la ventana de búsqueda con sub-pixeles de video no es normalmente necesaria, primero porque las sub-muestras no existen y su cálculo a partir de pixeles enteros origina un gran coste computacional y segundo, porque generalmente es suficiente con la interpolación de pixeles y ajuste de bloques alrededor de los mejores bloques de pixeles enteros. En el caso de la estimación de movimiento con resolución de ¼ de píxel, el primer paso es encontrar el mejor ajuste entero, para después determinar el bloque *half*

de menor coste en su vecindad y finalmente realizar el ajuste de bloques *quarter* alrededor del mejor bloque *half*.

El procedimiento general de inter-predicción con resolución fraccionaria y algoritmo FSBM para el estándar H.264/AVC, está codificado en la función C *SubPelBlockMotionSearch* del software de referencia JM [JM1106], la cual se compone de una secuencia de lazos *for*, para una operación bloque por bloque (Figura 5.1).

```
Interpolación cuadro de referencia
            :
Lazo 1 (Cuadro actual)
  Lazo 2 (Macro-bloque en el cuadro actual)
    Lazo 3 (Cuadro de referencia)
      Lazo 4 (41 bloques y sub-bloques)
        Lazo 5 (Estimación de movimiento en pixeles enteros)
          Lazo 6 (Estimación de movimiento half en 9 bloques candidatos)
          {
            Coste del vector de movimiento
            Lazo 7 (Pixeles en el bloque actual y bloque half candidato)
            {
              SATD
              Coste R/D
            }
            Comparación coste R/D
          }
          Lazo 8 (Estimación de movimiento quarter en 9 bloques candidatos)
          {  Coste del vector de movimiento
            Lazo 9 (Pixeles en el bloque actual y bloque quarter candidato)
            {
              SATD
              Coste R/D
            }
            Comparación coste R/D
          }
```

Figura 5.1. Lazos del procesamiento FME-FSBM en el software de referencia.

El procesamiento fraccionario inicia al terminar la estimación entera del bloque en curso (lazo 5), con la secuencia para la obtención del mejor bloque de resolución *half*. En una estrategia de exploración en espiral, se igualan y comparan los bloques *half* alrededor del mejor bloque entero (lazo 6), evaluando el coste del MV fraccionario y calculando la suma de diferencias transformadas absolutas "sum of absolute transform differences" (SATD), entre los pixeles del bloque actual y los sub-pixeles de cada bloque candidato (lazo 7). En una segunda etapa, el mejor bloque *half* sirve como centro para la evaluación de los bloques *quarter* (lazo 8), en un procesamiento similar al de resolución *half* (lazo 9). El cálculo de los pixeles fraccionarios no se incluye en los lazos secuenciales, ya que el software de referencia genera previamente los cuadros de referencia a ¼ de píxel de resolución.

La figura 5.1 puntualiza la operación secuencial y la dependencia de datos de la estimación de movimiento, ya que cada bloque debe pasar rigurosamente por las etapas de estimación entera, *half* y *quarter*, aun en una realización hardware. El diseño de una arquitectura FME se debe enfocar a la paralelización de cada una de las fases de

estimación, con una operación *pipeline* entre ellas, buscando la eficiencia del procesamiento bloque por bloque.

Una arquitectura hardware FME típica, se compone de las secciones de memoria, procesamiento y control (Figura 5.2). La sección de memoria incluye las memorias RAM de la SW, del MB actual, de los mejores MVs enteros y de los resultados fraccionarios, junto con sus unidades de manejo (AGUs). La sección de procesamiento se integra con los bloques de interpolación, distorsión SATD y coste R/D. La sección de control incluye la unidad de control y la interfase al sistema microprocesador, por donde se reciben los parámetros y comandos de operación. A partir de esta estructura genérica, se debe diseñar una arquitectura VLSI que satisfaga los requerimientos de resolución fraccionaria (*half* y *quarter*), tamaño variable de bloque y múltiples cuadros de referencia del estándar H.264/AVC, con una operación óptima en tiempo y área.

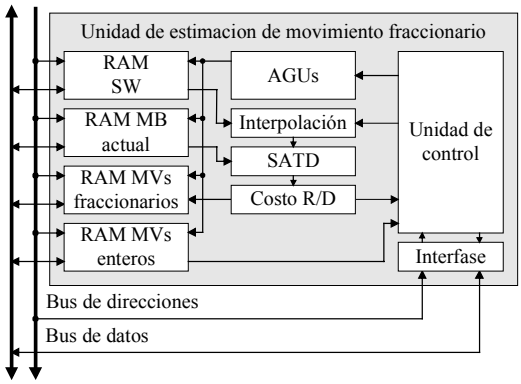


Figura 5.2. Estructura general de una arquitectura hardware de estimación de movimiento fraccionario.

A continuación se presenta el estado del arte de arquitecturas hardware FME-FSBM en la sección 5.2, el desarrollo y realización de un primer diseño FME en la sección 5.3, en la 5.4 una segunda propuesta enfocada a reducir los tiempos de estimación fraccionaria y en la 5.5 las conclusiones del trabajo realizado.

5.2 Estado del arte.

En este apartado se estudian varias arquitecturas que cumplen con alta eficiencia los requerimientos de la estimación de movimiento fraccionario del estándar H.264/AVC. Se inicia con una referencia a las estructuras FME-FSBM que utilizan algoritmos orientados a hardware, basados en el modelo de referencia JM. Posteriormente se analizan los mejores diseños de las etapas más críticas de la estimación fraccionaria, como son la interpolación y la transformada Hadamard "Hadamard transform" (HT). El objetivo del estudio es determinar las características de estructura y operación de las arquitecturas del estado del arte, que hacen que su comportamiento área/velocidad sea optimo, para su consideración en el desarrollo de nuevos sistemas FME.

5.2.1 Arquitecturas FME-FSBM.

Chen et al. [Chen04b] presentaron una de las primeras arquitecturas FME con tecnología VLSI que cumple los requerimientos del estándar H.264/AVC. Los autores paralelizaron varios lazos secuenciales del algoritmo y aplicaron técnicas de integración vertical y descomposición de bloques 4×4, para obtener un diseño con un flujo de datos muy regular, una alta utilización del hardware y un buen nivel de reutilización (Figura 5.3).

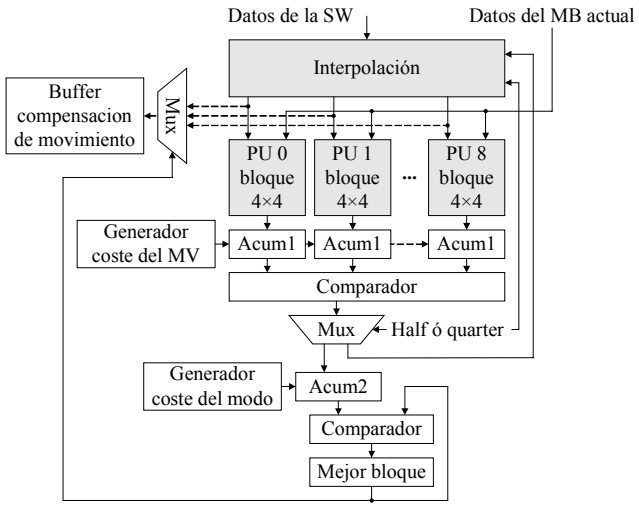


Figura 5.3. Arquitectura FME propuesta por Chen et al., con los elementos principales: una unidad de interpolación y 9 unidades procesadoras de bloques 4×4 para el cálculo de la distorsión SATD.

El diseño descompone las 41 mejores particiones y sub-particiones enteras en bloques 4×4, para su procesamiento en un módulo de interpolación y en 9 unidades procesadoras "process unit" (PU), para ese tamaño de bloques. La reutilización de datos se obtiene con el procesamiento continuo de los bloques 4×4 vecinos, ubicados verticalmente en los bloques enteros. La arquitectura paraleliza el lazo 6 y procesa al mismo tiempo 4 pixeles del lazo 7 (Figura 5.1), utilizando 16 filtros FIR de 6-taps y 32 filtros bilineales en el módulo de interpolación y 9 transformadas Hadamard 4×4 en las PUs. El diseño incluye dos grupos acumuladores (Acum1 y Acum2) para el cálculo del coste R/D en bloques mayores a 4×4 pixeles y dos comparadores, para determinar los 41 bloques de menor coste con resolución de ¼ de pixel. Su realización en tecnología UMC 0.18µ procesa 49 000 MB/s a una frecuencia de 100 MHz, para una velocidad de 2041 ciclos/MB. Esto es suficiente para soportar secuencias de video con formato SDTV @ 30 fps, para una trama de referencia.

Rahman y Badawy [Rahman05] diseñaron una arquitectura FME bajo el estándar H.264/AVC, capaz de procesar secuencias de video CIF @ 30 fps, considerando 5 cuadros de referencia y un rango de búsqueda de [–3,75, +4.00] (Figura 5.4). Operando a 120 MHz, el diseño procesa 59 400 MB/s, a una velocidad de 2020 ciclos/MB. La arquitectura está integrada principalmente por una SW fraccionaria compuesta por 23 bloques de RAM de un puerto, 9 PUs para el cálculo del SAD y una unidad de comparación. No incluye el módulo de interpolación y el componente de coste de la tasa. Si bien este diseño compara los bloques fraccionarios en un rango de búsqueda mayor al del modelo de referencia, su desventaja es que omite la unidad de interpolación, por lo que su propuesta es incompleta.

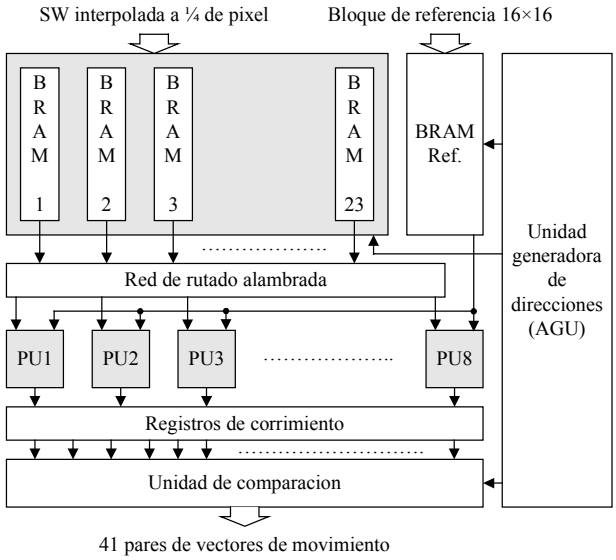


Figura 5.4. Arquitectura FME diseñada por Rahman y Badawy, que utiliza 23 memorias RAM para la realización de una SW fraccionaria y 9 PUs para el cálculo del SAD.

Yang et al. [Yang06] desarrollaron una arquitectura FME de alto rendimiento, que codifica secuencias de video con formato 1080 HD @ 30 fps (Figura 5.5). Basada en la arquitectura de [Chen04b], la nueva propuesta acelera la estimación fraccionaria, al procesar bloques completos y simultáneos, sin necesidad de descomposición, por medio de una unidad de interpolación *half* con 52 filtros FIR de 6-taps, una unidad de interpolación *quarter* de 128 filtros bilineales y 36 transformadas Hadamard 4×4 agrupadas en 9 PEs 16×16. Las características de operación *pipeline* en dirección vertical y diagonal de la unidad de interpolación *half*, permiten que el diseño trabaje a 200 MHz en el peor caso, para una velocidad de procesamiento de 790 ciclos/MB. El precio es una latencia extra de 6 ciclos en la unidad de interpolación y 2.38 veces más área con respecto al diseño de Chen et al.

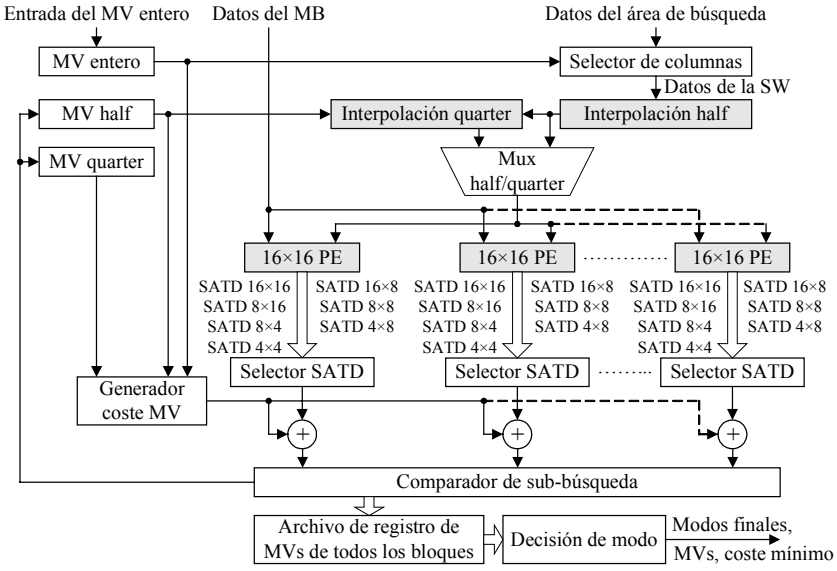


Figura 5.5. Diseño FME propuesto por Yang et al., con unidades de interpolación de alto rendimiento y matriz de 16×16 PEs.

5.2.2 Unidad de interpolación.

La unidad de interpolación, es el elemento que calcula sub-muestras de video a partir de pixeles enteros, utilizando filtros digitales. El estándar H.264/AVC especifica el cálculo de sub-pixeles por medio de filtros FIR de 6-taps y bilineales, para las posiciones de ½ y ¼ de píxel respectivamente. Debido a la complejidad de los filtros de 6-taps, así como de la gran cantidad de filtros FIR y bilineales necesarios para paralelizar los lazos *for* del algoritmo FME, la unidad de interpolación se debe diseñar con alta eficiencia en el flujo de datos y un equilibrio entre el nivel de concurrencia y el coste del área, como se observa en la clasificación y estado del arte de este tipo de arquitecturas.

5.2.2.1 Clasificación.

En función de los niveles de paralelismo y *pipeline*, las arquitecturas de interpolación se clasifican en estructuras de filtrado 1-D, 2-D y 1-D separable [Wang05]. Las arquitecturas 1-D son estructuras series de área mínima y baja velocidad de procesamiento, con poca eficiencia en los accesos a memoria y bajo nivel de reutilización de datos [Song05]. Las arquitecturas 2-D son estructuras paralelas de muy alta velocidad, alta latencia y gran coste en área, donde los sub-pixeles se calculan con una combinación de filtros 1-D horizontales y verticales [Lappalainen03]. Las arquitecturas 1-D separables son estructuras *pipeline* de alto nivel de reutilización de datos, con una primera etapa de procesamiento horizontal y una segunda de procesamiento vertical, ambas con filtros 1-D [Lappalainen03] (Figura 5.6).

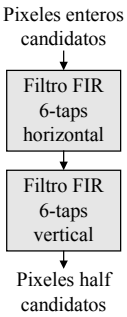


Figura 5.6. Estructura 1-D separable para la interpolación *half*.

Las arquitecturas de interpolación también se pueden clasificar de acuerdo al momento en que se procesan los sub-pixeles, que puede ser en el instante de su uso (*on-demand*) o previo a su uso (*before-hand*). En orden de ahorrar memoria, lo mejor es interpolar cada sub-pixel en el momento en que se usa, con el inconveniente de que si el píxel se requiere varias veces, se incrementa la carga computacional porque el mismo valor se calculara varias veces. Si no se tienen problemas de memoria, y se tiene la información correspondiente, los sub-pixeles se pueden calcular antes de su uso.

5.2.2.2 Estado del arte.

Song et al. [Song05] desarrollaron una arquitectura de interpolación 1-D *before-hand* para un codificador H.264/AVC, compuesta por dos memorias SRAM de puerto

dual (BMRAM e ISRAM), un filtro FIR de 6-taps, un filtro bilineal y un sumador de sub-muestras *half* (Figura 5.7).

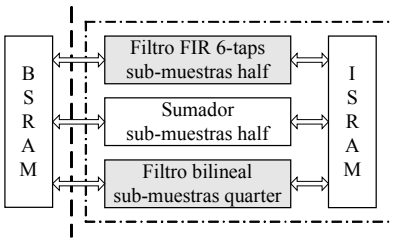


Figura 5.7. Diagrama a bloques de la arquitectura de interpolación de Song et al.

En un primer estado, el filtro de 6-taps y el sumador generan los sub-pixeles *half* y los almacenan en las memorias BSRAM e ISRAM. Cuando se decide el mejor ajuste de bloques de ½ resolución, el filtro bilineal calcula las sub-muestras *quarter* y las almacena en la memoria BSRAM. La arquitectura soporta todos los tamaños de bloque y resolución de sub-muestras sin simplificar el algoritmo de interpolación, pero necesita trabajar a muy alta frecuencia de reloj por su operación serie *pipeline* con solo dos filtros.

Chen, Huang y Chen [Chen04b] utilizaron en su arquitectura FME una estructura de interpolación 1-D separable *on-demand*, que genera los sub-pixeles de 9 bloques candidatos *half* 4×4 alrededor de una descomposición 4×4 de un bloque entero, para paralelizar el lazo *for* 6 de la figura 5.1. Los autores diseñaron el módulo de interpolación *half* como una secuencia de 5 filtros horizontales y 11 verticales tipo FIR de 6-taps con realización en árbol sumador, con un *buffer* de interpolación intermedio (Figura 5.8). Los filtros horizontales reciben 10 pixeles enteros adyacentes de la SW y generan 5 sub-pixeles *half*, los cuales son registrados y desplazados por el *buffer* junto con 6 de los pixeles enteros. Las columnas del *buffer* alimentan a 11 filtros verticales que generan 11 sub-pixeles *half*, que también se registran y desplazan por el *buffer*. Una vez que se decide cual es el mejor bloque candidato *half*, se utiliza una estructura de interpolación 2-D *on-demand* con 32 filtros bilineales, para la generación de 9 bloques candidatos *quarter* 4×4 alrededor de cada descomposición 4×4 del mejor bloque *half*, lo que apoya la paralelización del lazo *for* 8 del algoritmo FME.

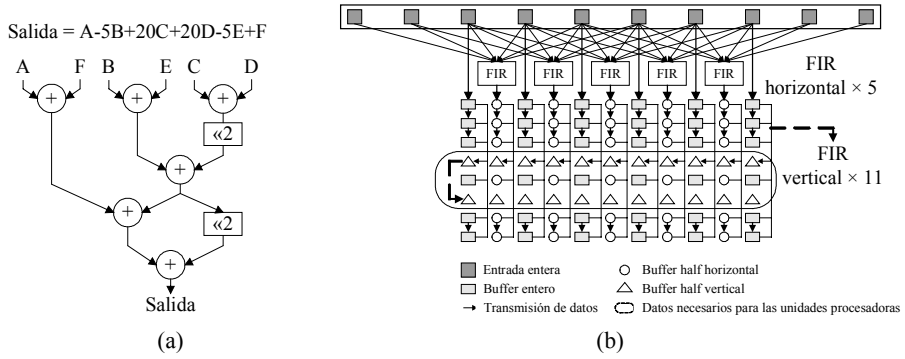


Figura 5.8. Unidad de interpolación de Chen et al., a) realización como árbol sumador del filtro FIR de 6-taps , b) estructura del interpolador *half*.

Yang, Goto e Ikenaga [Yang06] diseñaron una unidad de interpolación 1-D separable *on-demand*, para procesar bloques de hasta 16×16 pixeles (Figura 5.9). Para incrementar la utilización del hardware y ahorrar ciclos de reloj, ya que la unidad es completamente utilizada solo en la interpolación de los bloques 16×16 y 16×8, los bloques 4×8 y 4×4 se procesan simultáneamente. El uso de filtros FIR de 6-taps horizontales, verticales y diagonales incrementa la frecuencia de reloj, al reducir significativamente el número de niveles lógicos. El coste es una latencia extra de 6 ciclos y un esquema de control y filtrado más complejo, con respecto a las estructuras sin filtros diagonales.

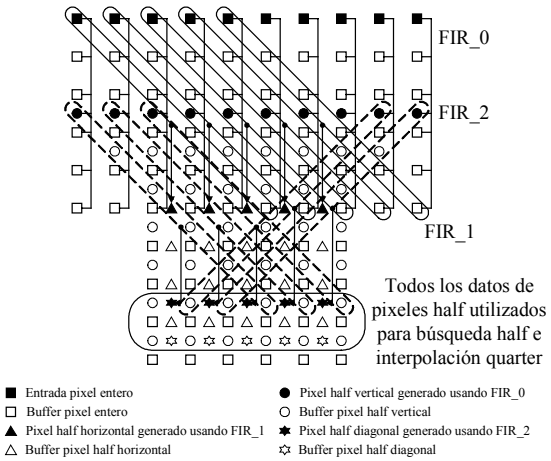


Figura 5.9. Arquitectura de la unidad de interpolación de Yang, Goto e Ikenaga.

Li, Wang y Wu [Li05] desarrollaron una arquitectura de interpolación 1-D separable *on-demand* segmentada-paralela, para un decodificador de cuadros HD bajo el estándar H.264/AVC (Figura 5.10). El diseño utiliza una estrategia de acceso a la memoria que optimiza la lectura de bloques de tamaño variable, un *buffer* de entrada que evita transferencias redundantes y una estructura en árbol sumador para la realización de los filtros FIR de 6-taps. La idea principal es leer cualquier tamaño de bloque desde la memoria de cuadros de referencia y dividirlo en bloques 4×4 antes de almacenarlos en el *buffer* de entrada, junto con los pixeles necesarios para su interpolación *half*, en un proceso similar a las técnicas de descomposición e integración de bloques 4×4 que utiliza el diseño en [Chen04b]. Los resultados muestran una velocidad de interpolación de 356 ciclos/MB, con un 60% de reducción en el ancho de banda de la memoria de referencia con respecto a diseños similares.

5.2.3 Módulos de transformada Hadamard.

La transformada Hadamard (HT) es una clase de transformada de Fourier que realiza una operación lineal sobre números reales o complejos. Esta es una transformada libre de multiplicaciones y operaciones de desplazamiento, que puede reemplazar a la transformada entera del coseno, en aplicaciones de menor rango dinámico y complejidad [Hallapuro02]. El modelo de referencia JM utiliza la HT en los lazos enteros y fraccionarios de inter-predicción, para calcular el coste de distorsión de los bloques candidatos, en un proceso de suma de diferencias transformadas absolutas (SATD), donde las diferencias entre el bloque actual y el bloque candidato se procesan con una HT 2-D.

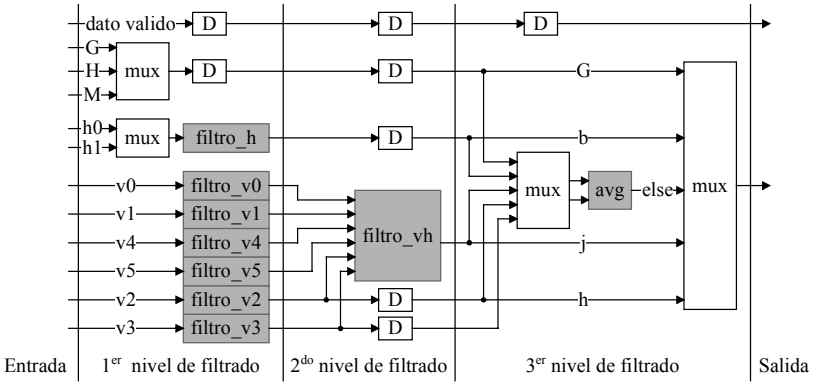


Figura 5.10. Unidad de interpolación segmentada-paralela de Li, Wang y Wu.

5.2.3.1 Clasificación.

En función del formato de datos de entrada, las HT se clasifican en estructuras de datos de entrada series o paralelos. De acuerdo al algoritmo de realización, las HT se clasifican en transformadas 2-D separable renglón-columna [Malvar01], transformada 2-D directa y transformada 2-D por multiplicación de matrices [Hallapuro02].

Si $Y = HXH^T = HM$ define la transformada Hadamard 2-D, donde X y Y son las matrices de entrada y salida respectivamente, entonces $M = XH^T$ representa la transformada 1-D de renglones sobre la matriz de entrada X y $Y = HM$ especifica la transformada 1-D de columnas sobre la matriz intermedia M. Este proceso es el llamado transformada separable renglón-columna. Si la matriz de salida $Y = HXH^T$ es computada ejecutando la multiplicación matriz-vector sobre los elementos de X, el resultado es la transformada 2-D directa (Figura 5.11). La última alternativa, la transformada por multiplicación de matrices, se considera en el estándar H.264/AVC para tomar ventaja de las instrucciones de multiplicación/acumulación de los microprocesadores modernos.

$$\begin{bmatrix} Y_{00} \\ Y_{01} \\ Y_{02} \\ Y_{03} \\ Y_{10} \\ Y_{11} \\ Y_{12} \\ Y_{13} \\ Y_{20} \\ Y_{21} \\ Y_{22} \\ Y_{23} \\ Y_{30} \\ Y_{31} \\ Y_{32} \\ Y_{33} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} X_{00} \\ X_{01} \\ X_{02} \\ X_{03} \\ X_{10} \\ X_{11} \\ X_{12} \\ X_{13} \\ X_{20} \\ X_{21} \\ X_{22} \\ X_{23} \\ X_{30} \\ X_{31} \\ X_{32} \\ X_{33} \end{bmatrix}$$

Figura 5.11. Algoritmo de realización de la transformada Hadamard 2-D 4×4 directa.

Para el cálculo de transformadas 1-D 4×4, se tiene un método que solo requiere 8 sumas por renglón o columna y que se ejecuta en un ciclo de reloj (Figura 5.12). Esta técnica, conocida como estructura de mariposa, es la realización más rápida de la HT 4×4 que existe.

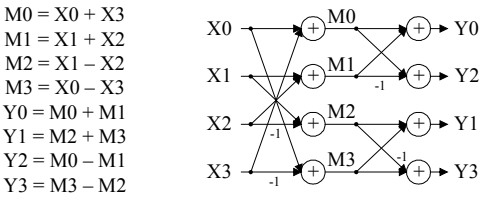


Figura 5.12. Estructura de mariposa para la realización de la transformada rápida Hadamard 1-D 4×4.

5.2.3.2 Estado del arte.

Kuo, Lin, Liu y Jen [Kuo05] propusieron una arquitectura serie 2-D directa 4×4, que puede aplicarse para la realización de transformadas Hadamard e ICT (Figura 5.13). En su operación, cada dato de entrada se procesa durante 16 ciclos, en una secuencia de multiplicación, acumulación y desplazamiento, con una duración total de 256 ciclos de reloj. Esta arquitectura es muy eficiente en área, pero su elevada latencia no permite su aplicación en sistemas de tiempo real.

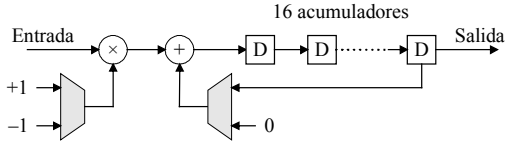


Figura 5.13. Arquitectura HT serie 2-D 4×4 directa de Kuo et al.

Porto et al. [Porto05] diseñaron una arquitectura HT serie 2-D renglón-columna, utilizando dos transformadas 1-D y un *buffer* de transposición (Figura 5.14). Diseñado con dos memorias RAM de 16 palabras, el *buffer* transpone los resultados de la primera HT 1-D y envía la matriz transpuesta a la segunda HT 1-D. Para mantener la continuidad de la recepción de datos series, los autores duplicaron los registros de cada etapa *pipeline* en una operación *ping-pong*. La arquitectura puede generar un resultado valido cada ciclo de reloj, después de una latencia de 24 ciclos.

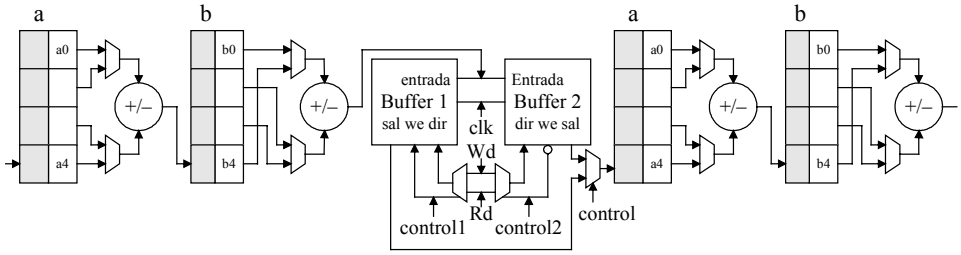


Figura 5.14. Arquitectura HT serie 2-D 4×4 renglón-columna de Porto et al.

Wang et al. [Wang03] describieron una arquitectura HT paralela 2-D renglón-columna que procesa 4 entradas simultaneas, con dos HT 1-D rápidas y una matriz de transposición 4×4 (Figura 5.15). Un multiplexor dentro de los registros de la matriz modifica la dirección de los datos cada 4 ciclos, para que la transformada 2-D se realice continuamente, con una latencia total no acumulable de 4 ciclos. El mismo algoritmo puede ser utilizado para procesar 8 y 16 datos simultáneos de la matriz 4×4 de entrada. La arquitectura de 8 datos usa 4 HT 1-D para procesar los datos en 2 ciclos de reloj con una latencia mínima (2 ciclos). La arquitectura de 16 datos utiliza 8 HT 1-D, para una operación en un ciclo de reloj sin latencia. Esta última estructura no necesita una matriz de transposición porque las salidas de las transformadas de los renglones son cableadas directamente a las entradas de las transformadas de las columnas [Amer04].

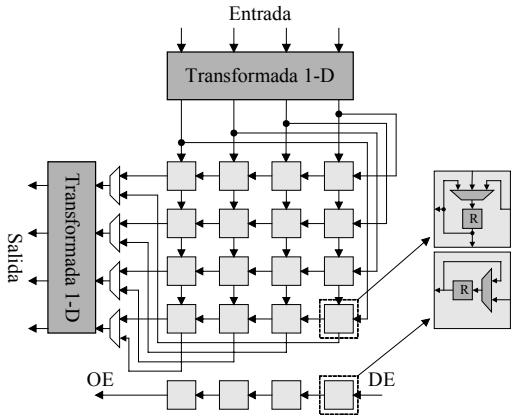


Figura 5.15. Arquitectura HT paralela 2-D 4×4 renglón-columna de Wang et al.

Cheng, Chen, Liu y Yang [Cheng04] desarrollaron la arquitectura HT paralela 2-D directa donde todos los datos de la matriz 4×4 de entrada se reciben simultáneamente (Figura 5.16). El diseño presenta una lógica de dos etapas, con 2 HT 1-D, 16 sumadores y sin memoria de transposición, por lo que los resultados se completan en dos ciclos de reloj. El procesamiento en dos etapas requiere un esquema de control que indique la operación del nodo de entrada, en sus alternativas de sumador y restador.

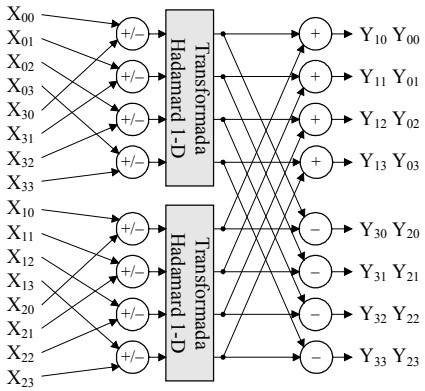


Figura 5.16. Arquitectura HT paralela 2-D 4×4 directa de Cheng et al.

5.2.4 Evaluación de las arquitecturas previas y conclusiones.

5.2.4.1 Arquitecturas completas.

Las arquitecturas FME del estado del arte son esencialmente estructuras secuenciales con alto nivel de paralelización en los lazos de interpolación y distorsión SATD. En su desarrollo los autores buscaron cumplir con el algoritmo del modelo JM a la mayor velocidad y menor área posible, a partir de la optimización del flujo de procesamiento, la reutilización de datos y la concurrencia de operaciones. Su evaluación se enfoca a determinar las características estructurales y de operación, que les permiten alcanzar su alto nivel de operación, para tomarlas como punto de partida en el diseño de nuevas arquitecturas.

La evaluación se hace en las arquitecturas que se presentan en [Chen04b] y [Yang06]. Ambos son muy buenos ejemplos de la realización hardware del algoritmo de inter-predicción P fraccionaria, del modelo JM del estándar H.264/AVC. Ya que el trabajo de Yang et al. es una versión avanzada del diseño original de Chen et al., con un mayor nivel de paralelización en las unidades de interpolación y distorsión SATD, resulta interesante observar las estadísticas área-velocidad de estos diseños (Tabla 5.1).

Tabla 5.1. Estadísticas área/velocidad de las arquitecturas FME del estado del arte.

Arquitectura	Tecnología	Área (puertas)	Velocidad de Procesamiento (ciclos/MB)	Latencia + ciclos falsos	# filtros FIR 6-taps	# filtros bilineales	# HT 4×4
Chen04b	UMC 0.18µ	79372	1664 + latencia + ciclos falsos	377	16	32	9
Yang06	TSMC 0.18µ	188456	732 + latencia + ciclos falsos	58	52	128	36

Los siguientes puntos resumen los resultados del análisis:

* La velocidad de procesamiento en ciclos/MB es directamente proporcional al nivel de concurrencia de la unidad de interpolación

* La mayor velocidad para una unidad de interpolación especifica se consigue al reducir los ciclos falsos y la latencia en la secuencia de procesamiento, principalmente entre las etapas *pipeline half* y *quarter*.

* El módulo de distorsión SATD debe ser diseñado para procesar el flujo de sub-pixeles a la tasa que los genera la unidad de interpolación.

* Para mantener una secuencia de procesamiento continua, el sistema debe preparar con anticipación los datos que necesita la unidad de interpolación

* El mayor nivel de reutilización de datos se consigue cuando la unidad de interpolación tiene capacidad para procesar los bloques de mayor tamaño.

5.2.4.2 Unidades de interpolación.

En la evaluación de las unidades de interpolación del estado del arte, se observa que sus arquitecturas son realizaciones directas de los esquemas de la clasificación, en cuanto a los niveles de segmentación-paralelismo y del momento en que generan los sub-pixeles, por lo que su análisis se hace en forma subjetiva, a partir de la categoría a la que pertenecen (Tabla 5.2), como se concreta en las siguientes aseveraciones:

Tabla 5.2. Evaluación de las unidades de interpolación del estado del arte en función de su clasificación.

Clasificación	Requerimiento de memoria	Velocidad de procesamiento	Latencia	Carga computacional	Coste del área	Nivel de Reuso	Ejemplo
1-D before-hand	muy alto	muy baja	nula	muy baja	muy bajo	nulo	[Song05] (half/quarter)
1-D separable before-hand	muy alto	media	nula	alta	medio	medio	[Rahman05] (half)
1-D separable on-demand	bajo	alta	media	alta	alto	alto	[Chen04b] [Yang06] [Li05] (half)
2-D on-demand	bajo	muy alta	media	muy alta	muy alto	alto	[Chen04b] [Yang06] (quarter)

* Las arquitecturas 1-D before-hand son estructuras optimas en área y coste computacional, pero que no reúnen los requisitos de operación en tiempo real, debido a su baja velocidad de procesamiento derivada de su operación serie.

* Las arquitecturas 1-D separables before-hand pueden ser consideradas para generar las sub-muestras *half*, pero únicamente en el caso de que la unidad de interpolación procese bloques completos, que permitan tener toda la información en el momento que lo requiere el siguiente nivel de interpolación.

* Las arquitecturas 1-D separables on-demand, en combinación con técnicas de descomposición de bloques, son muy adecuadas para el procesamiento de los sub-pixeles *half*, ya que bajo este entorno mantienen un alto equilibrio área/velocidad, al aplicarse en los sub-pixeles del segmento de bloque que se procesa en tiempo real.

* Las arquitecturas 2-D on-demand tienen las mejores características de velocidad de procesamiento, al precio de una alta carga computacional y el mayor coste en área, por lo que su aplicación se recomienda en procesos de interpolación con filtros básicos, como el bilineal, para la obtención de los sub-pixeles *quarter*.

* En el codificador de video, es más eficiente calcular previamente los sub-pixeles necesarios para la estimación de movimiento, mientras que en el decodificador es más eficiente calcular solo los sub-pixeles a los que apuntan los MVs.

* En la aplicación de estas estructuras a los filtros FIR de 6-taps y bilineal del H.264/AVC, se observa que los requerimientos de redondeo de los filtros FIR solo

permiten su realización en las estructuras 1-D y 1-D separable y que la sencillez de los filtros bilineales permite su diseño con cualquiera de las tres opciones.

* Un enfoque intermedio, es interpolar las sub-muestras *half* necesarias con la estrategia before-hand, almacenar los resultados temporalmente en memoria y calcular las sub-muestras *quarter* on-demand al conocer cual es el bloque *half* que se utilizara como centro para la estimación *quarter*.

5.2.4.3 Arquitecturas de transformada Hadamard.

La evaluación de las arquitecturas de transformada Hadamard se realiza para matrices de entrada 4×4. Aun cuando el cálculo de la distorsión SATD se ejecuta en bloques de diversos tamaños, de 4×4 a 16×16, la HT 4×4 es la base de los diseños hardware, ya que el uso de HTs de mayor tamaño es prohibitivo por la complejidad de cómputo que incluyen (por ejemplo, la transformada rápida HT 8×8 utiliza 6 veces más sumas que su similar 4×4 [JM1106]). El estudio se realiza de una forma objetiva, de acuerdo a la clasificación y las mediciones de rendimiento de las arquitecturas modelo (Tabla 5.3). Las conclusiones más importantes se listan a continuación:

Tabla 5.3. Mediciones de rendimiento de las arquitecturas de transformada Hadamard 4×4.

Algoritmo	Tipo	Ejemplo	Tasa de datos (pixeles/ciclo)	Velocidad de procesamiento (ciclos)	Latencia (ciclos)	# Sumas	Buffer de transposición
renglon-columna	serie	[Porto05]	1	16 + latencia	24	4	si
directo	paralelo	[Wang03]	4	4 + latencia	4	16	si
	paralelo	[Wang03]	8	2 + latencia	2	32	si
	paralelo	[Amer04]	16	1 + latencia	0	64	no
	serie	[Kuo05]	1	16 + latencia	256	1	no
	paralelo	[Cheng04]	8	2 + latencia	0	32	no
	paralelo	-	16	1 + latencia	0	240	no

* Las arquitecturas HT deben tener la capacidad de procesar los bloques *half* y *quarter* que genera la unidad de interpolación, con mínima latencia y sin ciclos falsos.

* La paralelización del cálculo SATD en el algoritmo FME requiere una gran cantidad de elementos HTs 4×4 (por ejemplo 9 en [Chen04b] y 36 en [Yang06]), por lo que el número de sumas, la inclusión del *buffer* de transposición o la necesidad de un mayor esquema de control en la arquitectura seleccionada, tienen un gran impacto en el área del sistema de estimación fraccionaria.

* Las arquitecturas HT serie solo son aplicables en sistemas de codificación de video, donde la prioridad sea la optimización en área o en aquellos que puedan operan a muy alta frecuencia.

* Las realizaciones HT directas paralelas son críticas en área, por la gran cantidad de sumas que utilizan o por la necesidad de una lógica de control, por lo que bajo los mismos requerimiento de operación, se deben preferir otros diseños.

* Las arquitecturas HT renglón-columna paralelas, proporcionan todas las posibilidades en el formato de los datos de entrada y salida, con posibilidades de un procesamiento continuo sin acumulación de latencia, a un coste medió en el área.

5.3 Arquitectura *fractional full-search* propuesta.

En esta sección se presenta una arquitectura hardware FSBM para la estimación del movimiento fraccionario del componente de video luma, que satisface los requerimientos de tamaño de bloque variable y múltiples cuadros de referencia del algoritmo de inter-predicción P del estándar H.264/AVC. El diseño parte del estudio de las arquitecturas del estado del arte, con el que se determinaron las características estructurales y de operación que se deben buscar en los nuevos diseños, las cuales se exponen en los siguientes puntos:

* Alta velocidad de procesamiento, con la paralelización de la mayor cantidad de lazos *for* del algoritmo FME.

* Secuencia de procesamiento *half* y *quarter* de los 41 bloques enteros, con el menor número de ciclos falsos.

* Latencia mínima entre las etapas *pipeline half* y *quarter*.

* Flujo de datos continuo en la entrada del interpolador.

* Área total mínima.

* Reutilización de datos en la unidad de interpolación.

* Realización óptima de los filtros de interpolación.

* Selección y uso eficiente de los módulos de transformada Hadamard.

La arquitectura propuesta cumple estos requerimientos con las siguientes especificaciones de diseño:

* Alta velocidad de procesamiento, con la paralelización de las etapas de interpolación y distorsión SATD, para procesar 9 bloques 4×4 simultáneamente.

* Sincronización óptima de la secuencia de procesamiento, para la reducción de ciclos extras.

* Minimización de la latencia, con el diseño de unidades de interpolación y distorsión SATD de operación continua, con latencia no acumulable.

* Flujo de datos continuo a la entrada del interpolador, por medio de una lógica de lectura y selección de los datos de la memoria de la SW que envía 10 pixeles por ciclo de reloj al interpolador.

* Reducción del área total, con la aplicación de técnicas de descomposición de los bloques mayores en sub-bloques 4×4, lo que permite utilizar unidades de interpolación y distorsión SATD de bajo coste en área.

* Reutilización de datos en la unidad de interpolación, con la aplicación de la técnica de integración vertical de bloques 4×4.

* Desarrollo de la unidad de interpolación *half* (1-D separable) y *quarter* (2-D) on-demand, utilizando filtros FIR de 6-taps horizontales y verticales, con realización tipo árbol sumador, en una sola etapa *pipeline* y filtros bilineales paralelos.

* Diseño eficiente de los módulos SATD utilizando HTs 4×4 de arquitectura paralela renglón-columna.

A continuación se documenta el diseño de la arquitectura FME propuesta y de cada uno de los componentes que la integran, con un enfoque independiente de la tecnología de realización y considerando los tópicos de ancho de banda, reutilización de datos y orientación hardware del software de referencia, aplicados en el diseño del IME.

5.3.1 Descripción del diseño FME-FSBM.

La arquitectura propuesta esta diseñada como un acelerador hardware para resolver los requerimientos de estimación de movimiento con resolución fraccionaria del estándar H.264/AVC, a partir de las secciones de control, procesamiento, generación de direcciones y selección de datos (Figura 5.17). La sección de control se forma con la lógica de control general y la interfase al bus local. La de procesamiento se compone de una unidad de interpolación y 9 módulos de distorsión SATD, así como de los componentes de coste y comparación R/D. La sección de generación de direcciones incluye principalmente las AGUs de las memorias del bloque actual y de la SW. Finalmente, la sección de selección integra los circuitos que manejan los datos de entrada: el multiplexor de memorias y el selector de pixeles.

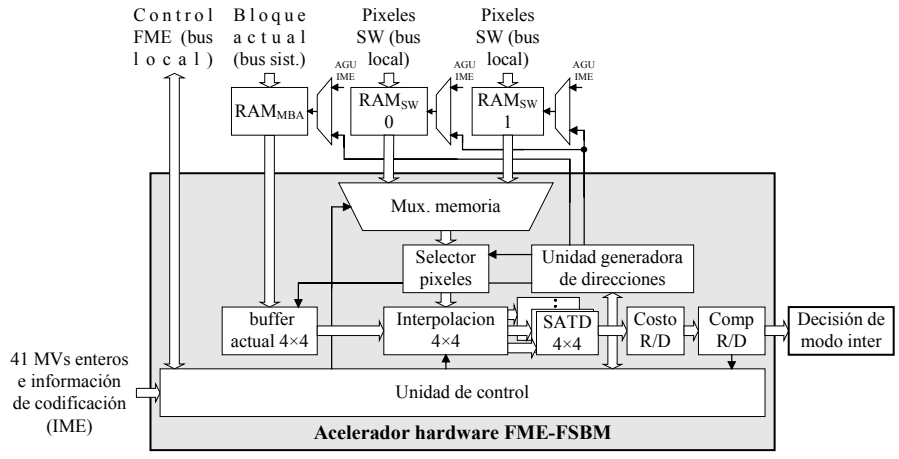


Figura 5.17. Diagrama a bloques de la arquitectura FME-FSBM propuesta.

Al igual que el diseño IME, la arquitectura FME se desarrolla como un periférico, pero solo para el procesador local, el cual supervisa su inicio y fin de operaciones. El procesador del sistema influye indirectamente a través del estimador de

movimiento entero y de la memoria RAM_{MBA}. El ciclo de estimación fraccionaria inicia con la recepción de la señal de visto bueno del procesador local y la indicación del acelerador IME de la validez de la información en su *buffer* de salida, al termino del ciclo de estimación entera. Al detectar estas señales, la primera acción del FME es leer del *buffer* la siguiente información, para preparar el arranque del nuevo ciclo:

* Los pares de coordenadas (x,y) de los 41 bloques enteros de menor coste, con respecto al origen de la SW.

* Memoria RAM_{SW} valida para el procesamiento fraccionario (#0 ó #1).

* Posición en RAM_{MBA} del MB actual (posición 0 ó 1).

* Desplazamiento de búsqueda Ph y parámetros de coste R/D.

Las coordenadas de los bloques enteros de menor coste definen su posición en la ventana de búsqueda, y por lo tanto, su ubicación en la memoria RAMsw. Cabe destacar que las 41 posiciones son totalmente aleatorias, sin depender unas de otras y solo en función del movimiento con resolución entera detectado en cada tamaño de bloque, por lo que al aplicar el algoritmo FME-FSBM, cada bloque se procesa en forma autónoma y secuencial (Figura 5.18).

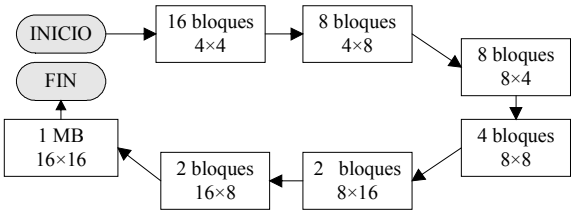


Figura 5.18. Secuencia de procesamiento FME-FSBM de los 41 mejores bloques enteros.

El número de RAM_{SW} define la memoria que usó el acelerador IME en su último ciclo de operación y que contiene los bloques candidatos de la SW, para los que son validas las 41coordenadas anteriores. Se recordara que se tienen dos RAM_{SW} compartidas por los aceleradores IME y FME, y que mientras el primero lee una de ellas, la otra se actualiza o es leída por el otro, en una alternancia controlada por el procesador local, de tal forma que se maximice el nivel de ocupación del sistema de estimación. Para disponer de la información que se necesita en el proceso de interpolación *half*, utilizando el filtro FIR de 6-taps que marca el estándar H.264/AVC, el acelerador FME requiere un cinturón de 3 pixeles de ancho alrededor de la SW. La franja extra de datos puede ubicarse dentro de la SW, redefiniendo la magnitud de los rangos de desplazamiento en la etapa de estimación entera, para eliminar los efectos de reducción del área de la SW (Figura 5.19).

Los parámetros para el cálculo del coste R/D son los mismos que se utilizaron en la etapa IME y que se recibieron para este ciclo de estimación desde el procesador del sistema. La idea de que el acelerador IME transfiera directamente estos parámetros, es para asegurar que el ciclo FME se ejecute bajo las mismas condiciones de operación.

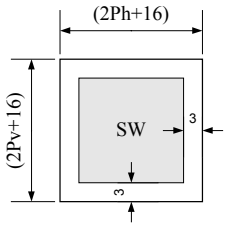
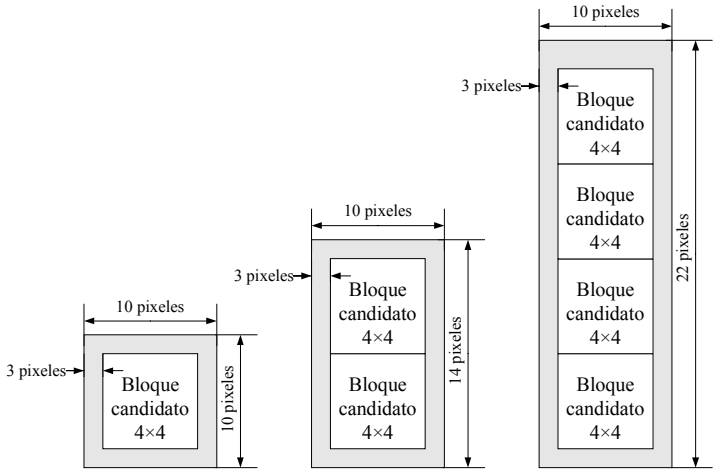


Figura 5.19. Modificación del tamaño de la SW, para incluir los pixeles necesarios para la interpolación de los bloques en la periferia de la ventana de búsqueda.

La posición en RAM_{MBA} del MB actual apunta hacia el MB que ya se estimó en forma entera y que ahora se procesara en estimación fraccionaria. Mientras tanto, la otra posición se actualiza con el nuevo MB actual, para su posterior lectura y carga en la matriz de PEs del estimador IME.

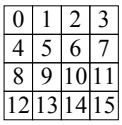
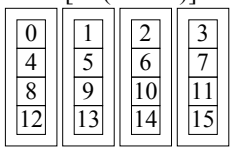
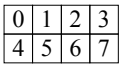
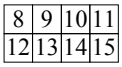
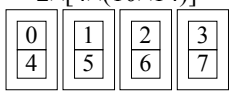
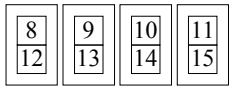
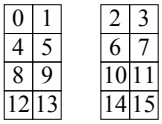
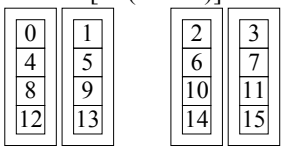
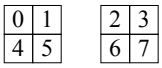
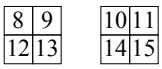
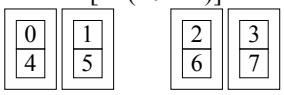
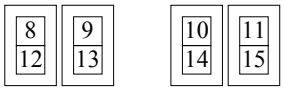
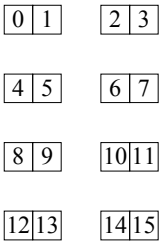
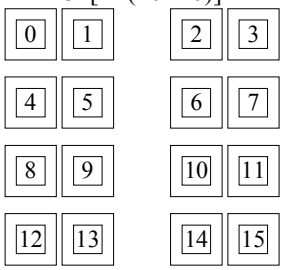
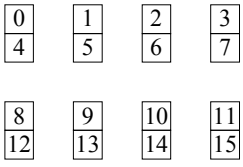
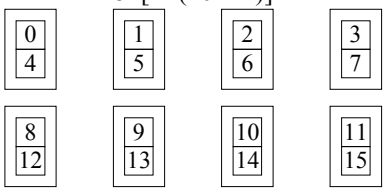
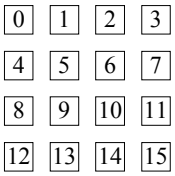
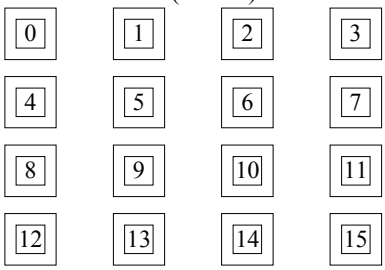
Las técnicas de descomposición 4×4 e integración vertical [Chen04b] son métodos de optimización computacional y reutilización de datos, que se utilizan cuando las unidades de procesamiento (interpolación y SATD) se diseñan para los bloques de menor tamaño. Su aplicación reduce el ancho de banda de la memoria RAM_{SW} y la carga computacional de la unidad de interpolación, al disminuir el número de accesos a memoria de 1120 a 832 y el número de pixeles enteros a procesar en el ciclo FME de 11200 a 8320. La descomposición consiste en desglosar los bloques mayores en sus componentes 4×4, para su procesamiento individual (interpolación y distorsión). La integración busca reutilizar los pixeles enteros y fraccionarios que se manipulan y procesan en la vecindad vertical de los bloques 4×4, lo que permite una reducción en el número de datos hasta del 45% (Figura 5.20). La tabla 5.4 muestra la aplicación de estas técnicas en los 7 tamaños de bloques, donde se observa la principal utilización en los bloques mayores.



# de bloques 4×4 verticales	Pixeles a procesar sin integración	Pixeles a procesar con integración	% de reducción
1	100	100	0
2	200	140	30
4	400	220	45

Figura 5.20. Reutilización de datos en la integración vertical de bloques 4×4 y porcentaje de reducción del número de pixeles manipulados y procesados.

Tabla 5.4. Descomposición e integración en bloques 4×4 de cada tamaño de bloque entero.

Bloque o sub-bloque	Integración	Descomposición e integración
1×(16×16) 	4 bloques verticales	1×[4×(10×22)] 
2×(16×8)  	2 bloques verticales	2×[4×(10×14)]  
2×(8×16) 	4 bloques vertical	2×[2×(10×22)] 
4×(8×8)  	2 bloques verticales	4×[2×(10×22)]  
8×(8×4) 	Ninguna	8×[2×(10×10)] 
8×(4×8) 	2 bloques verticales	8×[1×(10×14)] 
16×(4×4) 	Ninguna	16×(10×10) 

Una vez definida la ubicación de los datos en las memorias RAM_{MBA} y RAM_{SW}, el sistema lee y procesa los pixeles del MB actual. Simultáneamente, para cada mejor bloque entero candidato y cada una de sus descomposiciones 4×4, se leen 25 pixeles/ciclo y el selector elige los 10 pixeles que procesa la unidad de interpolación en cada ciclo de reloj. La unidad de interpolación calcula los sub-pixeles *half* y los sincroniza con los pixeles actuales, para enviar paquetes de 4 sub-pixeles *half* y 4 pixeles actuales por ciclo de reloj a cada uno de los 9 módulos SATD, hasta completar la interpolación de los candidatos *half* alrededor del bloque entero.

Los módulos SATD calculan la distorsión de cada descomposición 4×4 en 4 ciclos de reloj, con una latencia de 4 ciclos. En la integración vertical, los datos son procesados continuamente, con la latencia de un solo bloque 4×4. El componente de coste R/D recibe la distorsión SATD de cada descomposición 4×4 y la acumula para obtener el SATD de los bloques fraccionarios candidatos. Las distorsiones totales se suman con el coste de transmitir su MV fraccionario y los 9 resultados se comparan para definir el mejor candidato *half*. Al final, las coordenadas del bloque seleccionado se envían al módulo de interpolación vía la unidad de control, para realizar un proceso similar para los bloques *quarter* candidatos, alrededor del bloque *half* elegido.

Los resultados, al final de la estimación *half* y *quarter*, son los costes R/D y coordenadas con respecto al origen de la SW, en ¼ de píxel de resolución, de los mejores 41 bloques *quarter*, información que se respalda para su transferencia a la etapa de modo de decisión inter, donde se determina la combinación de particiones y sub-particiones que mejor estima el movimiento de los pixeles del MB actual. La figura 5.21 muestra gráficamente el proceso anterior.

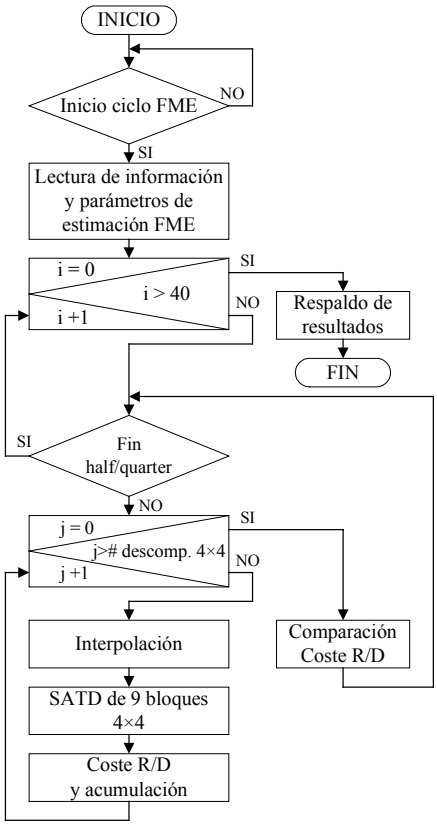


Figura 5.21. Diagrama de flujo del ciclo de estimación FME.

5.3.1.1 Unidad de control FME.

La unidad de control es el componente que coordina la operación de los elementos de procesamiento y direccionamiento del estimador FME, con el objetivo de calcular los 41 bloques *quarter* de menor coste R/D, buscando obtener una alta reutilización de datos, la máxima utilización de las unidades de procesamiento y el menor número de ciclos de ejecución.

El módulo de control se conecta al *buffer* de salida de la etapa IME, para recibir la información de configuración (coordenadas de los 41 bloques enteros de menor coste, posición del MB actual y de la SW y parámetros de coste) y a un registro esclavo del procesador local, para intercambiar señales de validación y estado y así conjuntar los elementos necesarios para generar las señales de control del sistema FME (Figura 5.22).

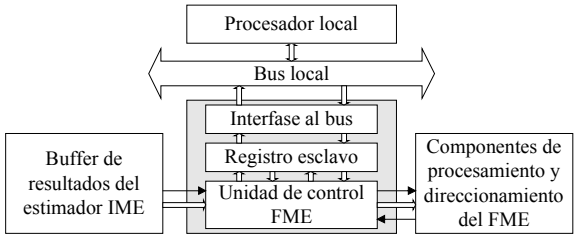


Figura 5.22. Diagrama a bloques de la unidad de control y su interconexión con el procesador local, el *buffer* de resultados del estimador IME y la lógica de procesamiento y direccionamiento del acelerador FME.

La secuencia de procesamiento FME (Figura 5.18) utiliza un algoritmo de control sofisticado (Figura 5.21), debido a su naturaleza serie, a la diversidad en la posición de los bloques a evaluar dentro de la SW y a la variedad de tamaños de bloque. Solo se logra cierta homogeneidad al aplicar la técnica de descomposición, que uniforma el diseño de las unidades de interpolación y distorsión SATD para el procesamiento de bloques de un solo tamaño (4×4), aun cuando la técnica de integración vertical que la acompaña, agrega otro nivel de complejidad al algoritmo.

Para facilitar el esquema de control, la sección de procesamiento se diseña en forma *pipeline*, con un flujo de datos controlado por las señales de validación de datos de entrada y de salida de las etapas que intervienen (Figura 5.23).

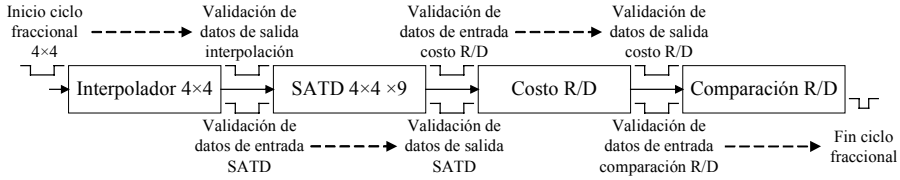


Figura 5.23. Secuencia *pipeline* de la sección de procesamiento FME.

Al inicio del ciclo fraccionario 4×4, la unidad de interpolación recibe los pixeles del bloque 4×4 actual y del bloque 10×10 candidato, la indicación de integración vertical y la información del tipo de ciclo *half* o *quarter*; si el que inicia es el segundo, se debe señalar el número del mejor bloque *half* alrededor del cual se realizará el procesamiento *quarter*. El módulo de coste R/D acumula el SATD de las

descomposiciones 4×4 hasta completar la distorsión del bloque completo, al cual le suma el coste de transmitir el MV fraccionario. Después de la comparación, la señal de fin de ciclo señala la validez del número del bloque fraccionario de menor coste.

La unidad de control se diseña con una FSM Moore y 3 contadores binarios. La máquina de estados realiza la secuencia de procesamiento de los 41 bloques, de mayor a menor tamaño, con 151 estados repartidos según el tamaño del bloque (Tabla 5.5), con el cambio de estado definido en función del valor de los contadores, los cuales presentan las siguientes especificaciones:

a) Contador de renglones del bloque actual (*cont_reng_act*). Contador binario de 2 bits descendente, con habilitación de cuenta, que contabiliza los 4 renglones del bloque 4×4 actual que se cargan en la unidad de interpolación, en las secuencias *half* y *quarter*. Su valor se transfiere a la AGU de la memoria RAM_{MBA} para indicar el renglón que se lee de la memoria.

b) Contador de renglones del bloque candidato (*cont_reng_cand*). Contador binario de 5 bits descendente, con carga y habilitación de cuenta, que contabiliza los renglones de los bloques candidatos que se cargan en el interpolador, para ambas secuencias *half* y *quarter*. El valor de la cuenta se envía a la AGU para direccionar el renglón que se lee de la memoria RAM_{SW}.

c) Contador de bloques (*cont_bloques*). Contador binario de 4 bits ascendente, con carga y habilitación de cuenta, que contabiliza el número de bloques que se procesan. Define el fin de los ciclos *half* y *quarter* para cada tamaño de bloques.

Tabla 5.5. Número de estados de la FSM Moore de la unidad de control, según el tamaño de bloque en la secuencia de procesamiento.

	16×16	16×8	8×16	8×8	8×4	4×8	4×4
Número de estados	29	18	30	30	18	16	10

La figura 5.24 muestra la sección del diagrama de estados para el procesamiento de los 8 bloques 4×8, el cual puede tomarse como referencia para la secuencia de los otros tamaños de bloque. Por ser un esquema general, no se muestran las señales de entrada y salida y las condiciones de control para el cambio de estado. Cada bloque 4×8 se descompone en 2 bloques 4×4, en los que se aplica la técnica de integración vertical.

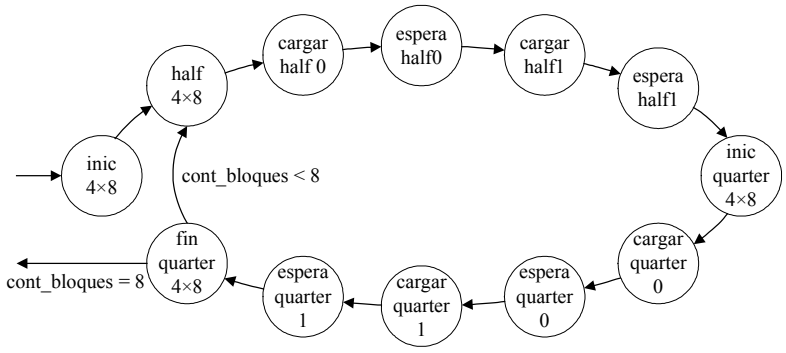


Figura 5.24. Secuencias de estados para el procesamiento FME de un bloque 4×8.

En el estado "inic_4×8" se definen las condiciones de procesamiento, como el valor inicial de los contadores y la indicación de la integración vertical de 2 bloques 4×4. Un ciclo después, en el estado "*half*_4×8", se habilita la lectura de las memorias del bloque actual y candidato, para que en el estado "cargar_*half*_0" se cargue el bloque 4×4 actual y los 10 renglones del bloque candidato en la unidad de interpolación, con el apoyo de los contadores de renglones. Al término de la carga de los pixeles de la primera descomposición 4×4, se tiene un estado de espera, para sincronizar los datos con los módulos de procesamiento y preparar las condiciones de operación para la segunda descomposición. La aplicación de la integración vertical, permite que en el siguiente estado "cargar_*half*1", se carguen solamente 4 renglones del bloque candidato junto con el segundo bloque 4×4 actual. Finalizada la carga, en el estado "espera_*half*1", se aguarda la conclusión del procesamiento *half*, hasta la obtención del número de bloque 4×8 *half* de menor coste, el cual se toma como centro para la secuencia de procesamiento *quarter*, que inicia en el estado "inic_*quarter*_4×8" y que continúa de una manera similar.

Al terminar el procesamiento *quarter*, en el estado "fin_*quarter*_4×8", el valor del contador de bloques indica el número de bloques ya procesados. Si el contador es menor a 8, la secuencia se repite hasta procesar los 8 bloques 4×8, para posteriormente saltar al estado de inicio del siguiente tamaño de bloque.

5.3.1.2 Unidad de interpolación 4×4.

La unidad de interpolación es el componente que genera los sub-pixeles *half* y *quarter* para la estimación de movimiento fraccionario bajo el estándar H.264/AVC. Su diseño tiene como objetivo paralelizar el lazo de interpolación del algoritmo FME-FSBM, con una arquitectura que genere bloques fraccionarios de tamaño 4×4, a la mayor velocidad y menor área posible.

Los sub-pixeles *half* ubicados en la vecindad horizontal de los pixeles enteros (ejemplo x, figura 5.25), se interpolan con las 6 muestras horizontales adyacentes (E, F, G y H, I, J), aplicando un filtro FIR de 6-taps con pesos (1, -5, 20, 20, -5 y 1) y el promedio y recorte del resultado entre 0 y 255 (5.1), para una resolución de 8 bits por sub-píxel, según lo especifica el estándar (interpolación *half* horizontal, FIR_H).

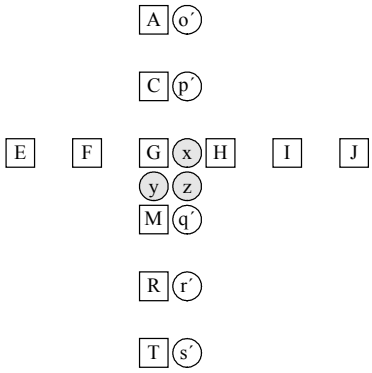


Figura 5.25. Pixeles y sub-pixeles para la interpolación *half* horizontal y vertical.

$$x' = FIR_H \text{ intermedio} = (E - 5*F + 20*G + 20*H - 5*I + J)$$
$$x = FIR_H = \min(255, \max(0, (x' + 16) >> 5)) \quad (5.1)$$

Los sub-pixeles *half* ubicados en la vecindad vertical de los pixeles enteros (ejemplo y, figura 5.25), se interpolan con las 6 muestras verticales adyacentes (A, C, G y M, R, T), en un proceso similar al de la interpolación *half* horizontal ($FIR_{V1,}$ 5.2).

$$y' = FIR_{V1} \text{ intermedio} = (A - 5*C + 20*G + 20*M - 5*R + T)$$
$$y = FIR_{V1} = \min(255, \max(0, (y' + 16) >> 5)) \quad (5.2)$$

Los sub-pixeles *half* ubicados en la vecindad vertical de los sub-pixeles *half* previamente calculados (ejemplo z, figura 5.25), se interpolan con los valores intermedios de los 6 sub-pixeles verticales adyacentes *half* (o′, p′, x′ *y* q′, r′, s′), de acuerdo a la expresión (5.3), en la llamada interpolación *half* vertical 2 (FIR_{V2}).

$$z' = FIR_{V2} \text{ intermedio} = (o' - 5*p' + 20*x' + 20*q' - 5*r' + s')$$
$$z = FIR_{V2} = \min(255, \max(0, (z' + 512) >> 10)) \quad (5.3)$$

Las ecuaciones (5.1) y (5.2) difieren de la (5.3) en que en las primeras se utilizan pixeles enteros y en la tercera valores intermedios de los pixeles *half*. Si a cada valor intermedio de z (o′, p′, x′, q′, r′ y s′) se le suma el número 16, se tiene:

$$\begin{aligned} z'' &= ((o'+16) - 5*(p'+16) + 20*(x'+16) + 20*(q'+16) - 5*(r'+16) + (s'+16)) \\ &= (o' - 5*p' + 20*x' + 20*q' - 5*r' + s') + 512 \\ &= z' + 512 \end{aligned}$$

lo cual corresponde a la expresión z antes del desplazamiento de 10 a la derecha y del recorte entre 0 y 255. La operación "valor intermedio + 16" también se observa en las ecuaciones 5.1 y 5.2, por lo que esta es la expresión común de los filtros FIR verticales y horizontales.

Manipulando algebraicamente la expresión "x′ + 16", se tiene:

$$\begin{aligned} x' + 16 &= E - 5*F + 20*G + 20*H - 5*I + J + 16 \\ &= (E + J) + 20*(G + H) - 5*(F + I) + 16 \\ &= (E + J) + (4 + 1) * [4*(G + H) - (F + I)] + 16 \\ &= [(E + J) + 16] + 4 * [4*(G + H) - (F + I)] + [4*(G + H) - (F + I)] \end{aligned} \quad (5.4)$$

donde (5.4) define la realización en árbol sumador del componente común "valor intermedio + 16" de los filtros FIR_H, FIR_{V1} y FIR_{V2}, los cuales solo difieren en el valor de desplazamiento (Figura 5.26).

El diseño del interpolador completo, parte del análisis de los pixeles y sub-pixeles que se involucran en el proceso de generación de los 9 candidatos *half* alrededor del mejor bloque 4×4 entero y de los 9 candidatos *quarter* en la vecindad del bloque 4×4 *half* de menor coste (Figura 5.27), observándose lo siguiente:

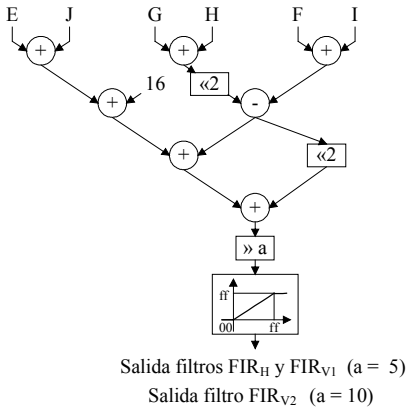


Figura 5.26. Realización en árbol sumador del filtro FIR de 6-taps, para la interpolación *half*.

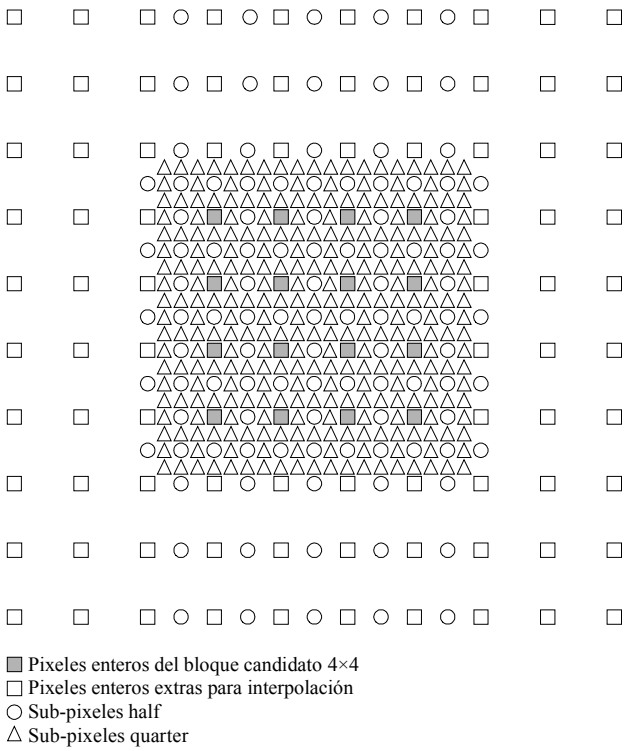


Figura 5.27. Pixeles y sub-pixeles en la interpolación de un bloque candidato 4×4.

* Los datos de entrada del interpolador son los pixeles del bloque 4×4 candidato más una franja de 3 pixeles de ancho a su alrededor, para un total de 100 pixeles enteros.

* Aun cuando la etapa anterior tuviese el ancho de banda suficiente para proporcionar simultáneamente los 100 pixeles enteros, el algoritmo de interpolación *half* no permite su procesamiento paralelo, por los requerimientos de redondeo que define, por lo que necesariamente se deben procesar en varios ciclos.

* En un procesamiento de 2 ciclos, se pueden generar en el primero los sub-pixeles *half* en la vecindad horizontal de los pixeles enteros, utilizando 50 filtros FIR$_H$ y en el segundo los sub-pixeles *half* restantes con 55 filtros FIR$_{V1}$ y FIR$_{V2}$. Este método es

muy eficiente en tiempo, pero demasiado oneroso en área, por lo que no es factible para una unidad de interpolación 4×4.

* El aumento del número de ciclos reduce la cantidad de filtros *half*, pero también incrementa la latencia, aun cuando se considere enviar a la salida los sub-pixeles inmediatamente después de su cálculo, para acelerar su procesamiento en las siguientes etapas (Tabla 5.6). En cualquier caso, la tasa de salida es irregular.

* Un factor básico de comparación se puede obtener con el producto "ciclos × # filtros", incluido en la tabla 5.6, donde el mejor rendimiento bajo este punto de vista se presenta en 8 y 11 ciclos.

Tabla 5.6. Rendimiento del interpolador *half* 4×4 en función del número de ciclos de ejecución.

ciclos	FIR_H	FIR_{V1} + FIR_{V2}	Latencia (ciclos)	Tasas de entrada (pixeles/ciclo)	Tasas de salida (sub-pixeles/ciclo)	Producto ciclos × # filtros
2	50	55	1	100	36, 45	210
3	25	55	1	50	18, 45	240
4	20	33	1	20, 40	9, 27	212
5	15	22	2	10, 30	18, 27	185
6	10	22	2	20	9, 18	192
8	10	11	2	20	9, 18	168
11	5	11	4	10	9, 18	176

Ya que las etapas precedentes influyen directamente en la unidad de interpolación, en este diseño se eligió la opción de 11 ciclos, por ser una de las más eficientes en ciclos × # filtros y por tener una tasa de entrada compatible con el formato de datos de la memoria RAM_{SW}. La tasa de salida de datos se regulariza con elementos de retardo, para hacerla correspondiente al flujo de 4 sub-pixeles/ciclo que requieren los siguientes 9 módulos SATD 4×4.

La figura 5.27 también incluye 280 sub-pixeles de los bloques *quarter* alrededor de los 8 bloques *half* 4×4 candidatos, pero solo se necesitan calcular 128 para los 8 bloques *quarter* 4×4 alrededor del mejor bloque *half*. El número de filtros bilineales necesarios se define en función de la disponibilidad de pixeles enteros y sub-pixeles *half* y de la tasa de flujo que requieren los 9 módulos SATD 4×4 de la siguiente etapa, que en este diseño es de 32 sub-pixeles *quarter* por ciclo y por lo tanto 32 filtros bilineales.

Los resultados del análisis anterior permiten concretar el diseño del interpolador, como una arquitectura que genera los sub-pixeles de 8 bloques *half* 4×4 candidatos usando un esquema 1-D separable on-demand con 16 filtros FIR de 6-taps y un *buffer* de interpolación y los sub-pixeles de 8 bloques *quarter* 4×4 candidatos a partir de un esquema 2-D on-demand con 32 filtros bilineales (Figura 5.28).

Para la operación de la unidad de interpolación 4×4, el selector de pixeles proporciona 10 muestras enteras por ciclo de reloj. Algunas de ellas fluyen directamente por el *buffer* de interpolación. Las otras son procesadas inmediatamente por 5 filtros FIR_H, para generar los sub-pixeles *half* en la vecindad horizontal de los pixeles enteros. Después de 6 ciclos, se tienen los datos para calcular los sub-pixeles *half* en la vecindad vertical de los pixeles enteros y sub-pixeles *half*, utilizando 6 filtros FIR_{V1} y 5 FIR_{V2}. Todas las interpolaciones *half* son calculadas en sitio con el apoyo del *buffer* de

interpolación, que almacena los pixeles y sub-pixeles que alimentan a las unidades de ejecución. Cuando la unidad de interpolación recibe la mejor posición *half*, habilita el cálculo de los sub-pixeles *quarter* en su alrededor, seleccionando la información a procesar de 5 renglones de pixeles enteros y *half* almacenados en el *buffer*. El módulo utiliza 32 filtros bilineales para computar los sub-pixeles de 8 bloques *quarter* 4×4 en 4 ciclos de reloj.

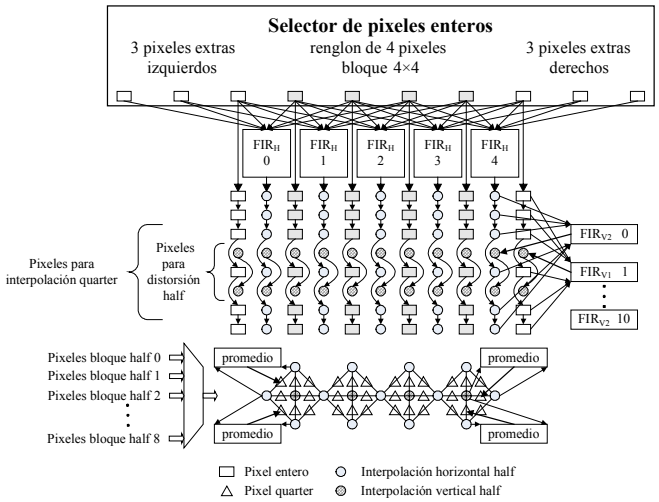


Figura 5.28. Arquitectura de la unidad de interpolación 4×4.

5.3.1.3 Módulo SATD 4×4.

El módulo SATD calcula el coste de distorsión con el método de la suma de diferencias transformadas absolutas, el cual optimiza la estimación de movimiento fraccionario con una transformada Hadamard 2-D en el lazo de decisión. El objetivo del diseño es calcular la distorsión de un bloque 4×4, con un componente que sea compatible con el flujo de datos que proporciona la unidad de interpolación. La arquitectura propuesta se replicara 9 veces, para paralelizar el lazo SATD del algoritmo FME, por lo que el área del diseño es critica.

El procesamiento SATD inicia cuando el componente recibe los bloques 4×4 actual c(i,j) y candidato r(i,j) y calcula la matriz de diferencias d(i,j) = c(i,j) − r(i,j), la cual es transformada usando la HT 2-D 4×4, para obtener la matriz td(i,j) (5.5).

$$td(i,j) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} d(0,0) & d(0,1) & d(0,2) & d(0,3) \\ d(1,0) & d(1,1) & d(1,2) & d(1,3) \\ d(2,0) & d(2,1) & d(2,2) & d(2,3) \\ d(3,0) & d(3,1) & d(3,2) & d(3,3) \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \quad (5.5)$$

Posteriormente, se suman los valores absolutos de los elementos de td(i,j) y el resultado se divide entre 2 con redondeo hacia el entero superior (5.6).

$$SATD_{4\times4} = \left(\sum_{i=0}^{3}\sum_{j=0}^{3} |td(i,j)| + 1\right) >> 1 \quad (5.6)$$

La arquitectura hardware SATD que satisface los requerimientos de flujo de datos, alta velocidad y mínima área se basa en el diseño propuesto en [Wang03], el cual consiste en una estructura HT paralela 2-D renglón-columna que procesa 4 pares de datos de entrada simultáneos, con 4 restadores, dos HT 1-D rápidas, un *buffer* de transposición 4×4 y la lógica de las operaciones de valor absoluto, suma, acumulación y división entre 2 (Figura 5.29).

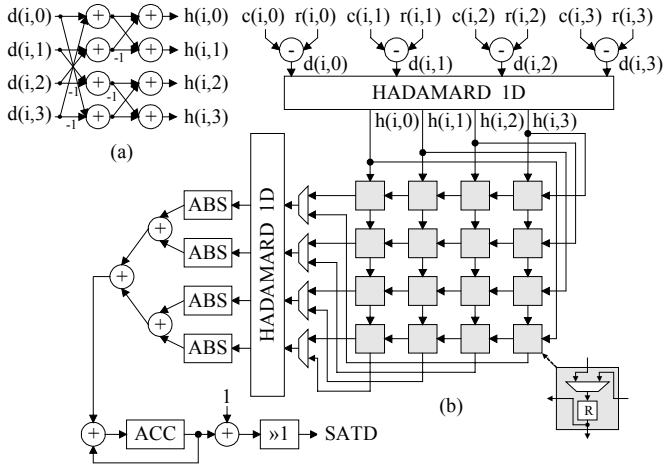


Figura 5.29. Módulo SATD 4×4, con una estructura Hadamard paralela 2-D renglón-columna.

El módulo SATD recibe los bloques 4×4 actual y candidato a la tasa en que los envía la unidad de interpolación (1 renglón de 4 pixeles/ciclo por bloque). Los renglones de datos son procesados por 4 restadores, para obtener la matriz de diferencias d(i,j) que alimenta, renglón por renglón, al módulo que ejecuta la transformada rápida de mariposa derecha (5.7).

$$\begin{bmatrix} h(0,0) & h(0,1) & h(0,2) & h(0,3) \\ h(1,0) & h(1,1) & h(1,2) & h(1,3) \\ h(2,0) & h(2,1) & h(2,2) & h(2,3) \\ h(3,0) & h(3,1) & h(3,2) & h(3,3) \end{bmatrix} = \begin{bmatrix} d(0,0) & d(0,1) & d(0,2) & d(0,3) \\ d(1,0) & d(1,1) & d(1,2) & d(1,3) \\ d(2,0) & d(2,1) & d(2,2) & d(2,3) \\ d(3,0) & d(3,1) & d(3,2) & d(3,3) \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \quad (5.7)$$

A la par de su procesamiento en 4 ciclos de reloj, la matriz h(i,j) se transfiere al *buffer* de transposición, con un flujo de datos de arriba hacia abajo. Al final de esta primera fase, se modifica la dirección del flujo de datos en el *buffer*, para procesar la matriz h(i,j) en la segunda HT 1-D rápida de mariposa, con un movimiento de columnas de derecha a izquierda (5.8).

$$td(i,j) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} h(0,0) & h(0,1) & h(0,2) & h(0,3) \\ h(1,0) & h(1,1) & h(1,2) & h(1,3) \\ h(2,0) & h(2,1) & h(2,2) & h(2,3) \\ h(3,0) & h(3,1) & h(3,2) & h(3,3) \end{bmatrix} \quad (5.8)$$

La dirección del flujo de datos en el *buffer* de transposición se define con la posición del multiplexor en la entrada de sus registros, controlado por el bit más significativo de un contador binario ascendente de carrera libre de 3 bits, que se

incrementa a partir de 0 cuando el módulo SATD recibe la señal de datos válidos y que permanece habilitado 4 ciclos después de que se invalidan los datos de entrada. Mientras la señal de validación permanezca activa, el módulo procesara adecuadamente los datos de entrada, sin importar la cantidad de bloques 4×4 que se reciban.

El resultado es un módulo con muy pocos componentes, para un bajo coste en área, que calcula la distorsión SATD de los bloques 4×4 actual y candidato en 4 ciclos de reloj más 4 ciclos de latencia, y con capacidad de procesamiento continuo de los bloques de entrada, sin acumular el retardo inicial.

5.3.1.4 Componentes de coste y comparación R/D fraccionarios.

El componente de coste fraccionario es el módulo que calcula el coste R/D de los 9 bloques candidatos *half* o *quarter* alrededor de cada mejor bloque entero. El componente de comparación es el que determina cual de los 9 candidatos tiene el menor coste. Si el proceso se realiza en los bloques *half*, el número del bloque resultante se retroalimenta a la unidad de interpolación para iniciar el ciclo *quarter* en su vecindad. El objetivo de diseño del primer módulo es optimizar el cálculo de la distorsión del bloque completo, a partir del valor SATD de sus descomposiciones 4×4 y optimizar el cálculo del coste de transmitir el MV, con los parámetros que recibe del estimador IME y la posición fraccionaria de cada bloque candidato. El comparador se diseña con el objetivo de obtener un alto rendimiento, ya que necesita comparar los 9 costes R/D a la mayor velocidad y en el menor tiempo.

El cálculo del coste R/D se realiza con la técnica que utiliza el software de referencia JM: la función Lagrangiana de coste (5.9), donde MVf es el vector de movimiento del bloque fraccionario candidato, MVp el vector de predicción, λ_{SAD} el multiplicador de Lagrange para estimación de movimiento y SATD y R el coste de distorsión y tasa respectivamente. El procedimiento es igual al que se aplicó en la estimación IME del capítulo anterior, con los mismos parámetros de coste, por lo que en esta sección se documentan tópicos específicos de la estimación fraccionaria, sin repetir lo expuesto previamente.

$$J_{MOVIMIENTO}(MVf \mid \lambda_{SAD}, MVp) = SATD(MVf) + \lambda_{SAD}\, R(MVf - MVp) \quad (5.9)$$

El coste de la distorsión SATD de los bloques candidatos se calcula acumulando el SATD de sus bloques de descomposición 4×4 (5.19), en una secuencia en donde se reciben 9 valores parciales de SATD y se acumulan para cada nivel de estimación fraccionaria y cada uno de los 41 mejores bloques enteros.

$$SATD_{bloque} = SATD_{descomposición\ 0} + SATD_{descomposición\ 1} + ... + SATD_{descomposición\ n-1} \quad (5.10)$$

La definición del coste de la tasa inicia con el cálculo de la diferencia "MVD = MVf – MVp", utilizando los siguientes parámetros que se reciben del estimador IME, en un proceso que se observa gráficamente en la figura 5.30:

* Las coordenadas del bloque actual (pic_pix_x, pic_pix_y) en unidades de pixeles enteros con respecto al origen del cuadro de video (enteros sin signo).

* Las coordenadas de la SW (search_x, search_y) en unidades de pixeles enteros con respecto al origen del cuadro de video (enteros sin signo).

* Los componentes *x* y *y* de la predicción del vector de movimiento MVp (pred_mv_x, pred_mv_y) en unidades de ¼ de píxel (enteros con signo)

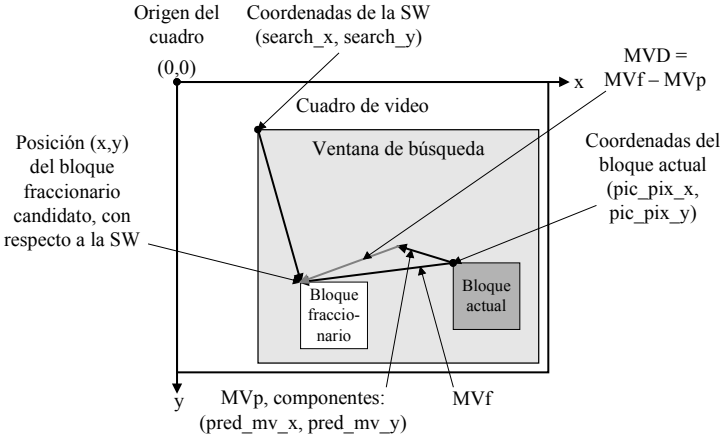


Figura 5.30. Cálculo de la diferencia de vectores MVD a partir de los parámetros de coste.

El MVf se define de acuerdo a la posición (x,y) del bloque fraccionario candidato, con respecto a las coordenadas de la SW. La posición fraccionaria se calcula con la posición del bloque entero candidato más el desplazamiento fraccionario del bloque *half* o *quarter* a su alrededor (Figura 5.31).

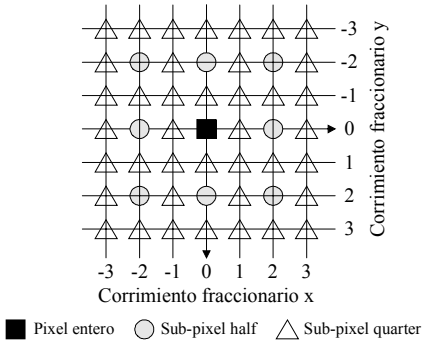


Figura 5.31. Posición fraccionaria de los candidatos *half* y *quarter* con respecto al bloque entero.

Una vez calculada la diferencia MVD en unidades de ¼ de pixel, el valor absoluto de sus componentes (x,y) se utiliza para indexar la tabla de longitud de código variable universal (UVLC). La suma de las salidas de la tabla da como resultado el número de bits que se necesitan para transmitir MVD. Finalmente, el coste de la tasa se determina con el número de bits anterior multiplicado por la versión en punto fijo de λ_{SAD}.

El módulo de coste R/D usa un esquema de control básico, basado en la señal de validación de los datos de entrada (valores SATD 4×4), con la cual se inicia del flujo de procesamiento y se genera la señal de validación de los datos de salida.

El comparador recibe 9 costes R/D y mediante un árbol de comparación (Figura 5.32), procesa la información de entrada y define el bloque de menor coste. Si los costes de entrada son de los bloques *half*, el resultado se envía a la unidad de interpolación para iniciar el ciclo *quarter*. Si los costes son de los bloques *quarter*, el MV del mejor bloque se respalda en un *buffer* para su posterior procesamiento en la etapa de decisión de modo inter. La operación del comparador inicia con la señal de validación de los datos de entrada y genera con la misma, por medio de una cadena de retardo, la señal de validación de salida, sin requerir otros elementos para su control.

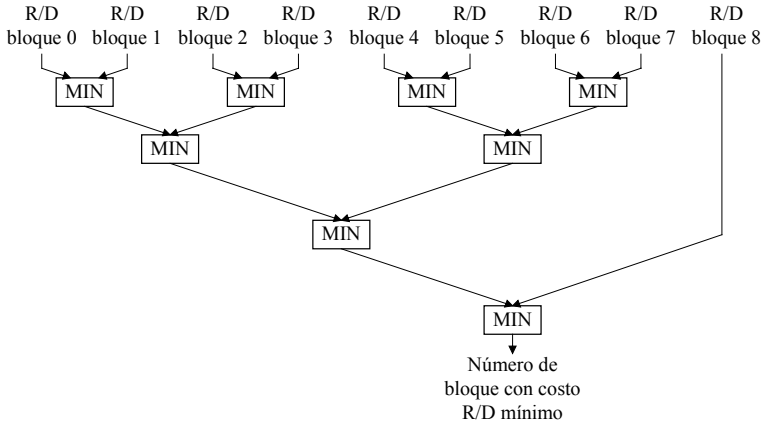


Figura 5.32. Arquitectura en árbol del comparador R/D.

5.3.1.5 Unidades fraccionarias generadoras de direcciones.

El sistema de estimación de movimiento incluye dos memorias para la ventana de búsqueda (RAM_{SW} #0 y RAM_{SW} #1) y una memoria para el MB actual (RAM_{MBA}). Diseñadas como periféricos para los procesadores local y del sistema respectivamente, las tres memorias son compartidas por los sub-sistemas IME y FME, de acuerdo a la lógica global de estimación. Cada sub-sistema define sus propias unidades de manejo de memoria, en función de sus requerimientos de tasa y datos a procesar. Su diseño se documenta a continuación, iniciando con una breve referencia a las memorias, que fueron desarrolladas en el capítulo anterior.

5.3.1.5.1 AGU fraccionaria de la RAM_{MBA}.

El MB actual se almacena en una memoria RAM local (RAM_{MBA}) de dos puertos, el primero configurado como periférico del procesador del sistema y el segundo, conectado a la sub-sistemas IME y FME, en operación de solo lectura. Para la transferencia de los pixeles desde la memoria externa del cuadro de video actual, la memoria local utiliza un puerto de 32 bits (bus del microprocesador de 32 bits), lo que permite una tasa de 4 pixeles por ciclo de reloj. Para la conexión a la matriz de procesamiento (IME) y a la unidad de interpolación (FME) se precisa un puerto que proporcione una tasa de 16 y de 4 pixeles/ciclo respectivamente. Para satisfacer estos requerimientos, se propone utilizar 4 bloques SRAM de 2 puertos, cada uno con una organización de 128×8 en el puerto A y de 32×32 en el puerto B. La memoria completa tiene una organización de 128×32 en el puerto A y de 32×128 en el puerto B, para almacenar 512 pixeles de 8 bits equivalentes a 2 MBs actuales.

La AGU fraccionaria de la memoria del MB actual (AGU_{MBAf}) calcula la dirección de los renglones de los bloques 4×4, busca los datos en memoria y los alimenta a la unidad de interpolación. Ya que la secuencia de direcciones es fija, la AGU_{MBAf} se define como una arquitectura de secuencia de direcciones predefinida con realización incremental [Kuhn99], alrededor de una máquina FSM integrada en la unidad de control del estimador FME y un proceso combinacional. En el diseño de la lógica combinacional se considera que la unidad de control accede a los datos de la RAM_{MBA} señalando el número de renglón (0 a 3), del bloque 4×4 (0 a 15), del MB (0 ó 1) deseado, en el formato que se muestra en la figura 5.33.

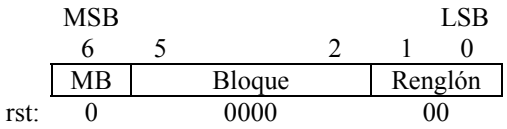


Figura 5.33. Formato de la palabra de control de la lógica combinacional de la AGU_{MBAf}.

El número de bloque se separa en dos grupos de 2 bits, donde el grupo menos significativo selecciona la SRAM donde se encuentra el bloque 4×4 y el más significativo define la dirección de su primer renglón, de acuerdo a la organización de la RAM_{MBA} (Tabla 5.7). La dirección final de la SRAM se forma con la suma del número de renglón, más la posición inicial del bloque, más el número del MB, como se observa en la figura 5.34.

Tabla 5.7. Organización de la memoria RAM_{MBA} (lado del puerto B), donde B es el número del bloque actual.

Dirección	Datos SRAM 3	Datos SRAM 2	Datos SRAM 1	Datos SRAM 0
(16×B)+0	P(0,15), P(0,14), P(0,13), P(0,12)	P(0,11), P(0,10), P(0,9), P(0,8)	P(0,7), P(0,6), P(0,5), P(0,4)	P(0,3), P(0,2), P(0,1), P(0,0)
(16×B)+1	P(1,15), P(1,14), P(1,13), P(1,12)	P(1,11), P(1,10), P(1,9), P(1,8)	P(1,7), P(1,6), P(1,5), P(1,4)	P(1,3), P(1,2), P(1,1), P(1,0)
(16×B)+2	P(2,15), P(2,14), P(2,13), P(2,12)	P(2,11), P(2,10), P(2,9), P(2,8)	P(2,7), P(2,6), P(2,5), P(2,4)	P(2,3), P(2,2), P(2,1), P(2,0)
(16×B)+3	P(3,15), P(3,14), P(3,13), P(3,12)	P(3,11), P(3,10), P(3,9), P(3,8)	P(3,7), P(3,6), P(3,5), P(3,4)	P(3,3), P(3,2), P(3,1), P(3,0)
(16×B)+4	P(4,15), P(4,14), P(4,13), P(4,12)	P(4,11), P(4,10), P(4,9), P(4,8)	P(4,7), P(4,6), P(4,5), P(4,4)	P(4,3), P(4,2), P(4,1), P(4,0)
(16×B)+5	P(5,15), P(5,14), P(5,13), P(5,12)	P(5,11), P(5,10), P(5,9), P(5,8)	P(5,7), P(5,6), P(5,5), P(5,4)	P(5,3), P(5,2), P(5,1), P(5,0)
(16×B)+6	P(6,15), P(6,14), P(6,13), P(6,12)	P(6,11), P(6,10), P(6,9), P(6,8)	P(6,7), P(6,6), P(6,5), P(6,4)	P(6,3), P(6,2), P(6,1), P(6,0)
(16×B)+7	P(7,15), P(7,14), P(7,13), P(7,12)	P(7,11), P(7,10), P(7,9), P(7,8)	P(7,7), P(7,6), P(7,5), P(7,4)	P(7,3), P(7,2), P(7,1), P(7,0)
(16×B)+8	P(8,15), P(8,14), P(8,13), P(8,12)	P(8,11), P(8,10), P(8,9), P(8,8)	P(8,7), P(8,6), P(8,5), P(8,4)	P(8,3), P(8,2), P(8,1), P(8,0)
(16×B)+9	P(9,15), P(9,14), P(9,13), P(9,12)	P(9,11), P(9,10), P(9,9), P(9,8)	P(9,7), P(9,6), P(9,5), P(9,4)	P(9,3), P(9,2), P(9,1), P(9,0)
(16×B)+10	P(10,15), P10(,14), P(10,13), P(10,12)	P(10,11), P(10,10), P(10,9), P(10,8)	P(10,7), P(10,6), P(10,5), P(10,4)	P(10,3), P(10,2), P(10,1), P(10,0)
(16×B)+11	P(11,15), P(11,14), P(11,13), P(11,12)	P(11,11), P(11,10), P(11,9), P(11,8)	P(11,7), P(11,6), P(11,5), P(11,4)	P(11,3), P(11,2), P(11,1), P(11,0)
(16×B)+12	P(12,15), P(12,14), P(12,13), P(12,12)	P(12,11), P(12,10), P(12,9), P(12,8)	P(12,7), P(12,6), P(12,5), P(12,4)	P(12,3), P(12,2), P(12,1), P(12,0)
(16×B)+13	P(13,15), P(13,14), P(13,13), P(13,12)	P(13,11), P(13,10), P(13,9), P(13,8)	P(13,7), P(13,6), P(13,5), P(13,4)	P(13,3), P(13,2), P(13,1), P(13,0)
(16×B)+14	P(14,15), P(14,14), P(14,13), P(14,12)	P(14,11), P(14,10), P(14,9), P(14,8)	P(14,7), P(14,6), P(14,5), P(14,4)	P(14,3), P(14,2), P(14,1), P(14,0)
(16×B)+15	P(15,15), P(15,14), P(15,13), P(15,12)	P(15,11), P(15,10), P(15,9), P(15,8)	P(15,7), P(15,6), P(15,5), P(15,4)	P(15,3), P(15,2), P(15,1), P(15,0)

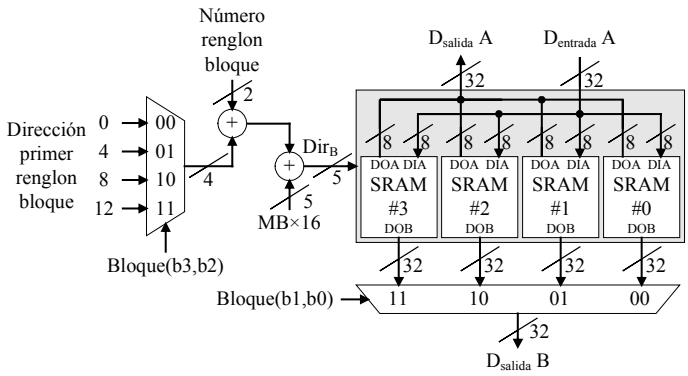


Figura 5.34. Arquitectura de la lógica combinacional de la AGU_{MBAf}, para el direccionamiento de los renglones de los bloques 4×4 actuales en la memoria RAM_{MBA}.

5.3.1.5.2 AGU fraccionaria de las RAM_{SW}.

Las memorias de búsqueda RAM_{SW} son los componentes que almacenan los pixeles de la SW para la estimación de movimiento entera y fraccionaria. La arquitectura de dos puertos que presentan, permite su configuración como periféricos del procesador local y simultáneamente conectarse a los estimadores IME y FME. Cada memoria se compone de 4 bloques SRAM de puertos A y B, los dos con una organización de ((2*Ph+16)(2*Pv+16)/16)×32, en función de los desplazamientos de exploración (Ph, Pv). La organización de la memoria completa es de ((2*Ph+16)(2*Pv+16)/4)×32 en el puerto A y de ((2*Ph+16)(2*Pv+16)/16)×128 en el puerto B. Su diseño está enfocado a la obtención del máximo ancho en bytes, conectando el puerto A al bus local de 32 ó 64 bits y administrando ambos puertos A y B para transferir 32 pixeles candidatos hacia los selectores de pixeles, vía los multiplexores de selección de memorias.

La unidad generadora de direcciones fraccionaria de la memoria de búsqueda (AGU_{SWf}), tiene como objetivo calcular la dirección en RAM_{SW} del píxel inicial de cada renglón de los bloques 10×10 (ó de bloques del mismo tipo integrados verticalmente), formados por el bloque de descomposición 4×4 y una franja de 3 pixeles a su alrededor. La información de entrada es la posición del bloque 4×4, en unidades de pixeles enteros (pos_x_4×4, pos_y_4×4), con respecto al origen de la SW (Figura 5.35).

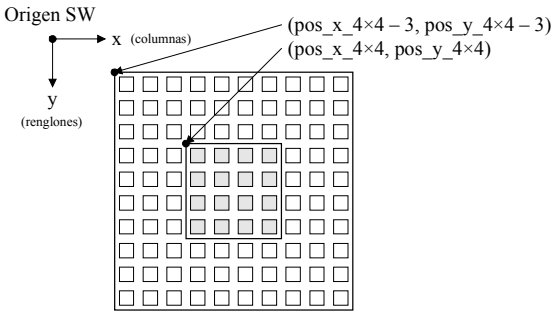


Figura 5.35. Posición del bloque 4×4 y del bloque 10×10 con respecto al origen de la SW.

El cálculo de la dirección inicia con la determinación de la posición inicial (x,y) del bloque 10×10, en función de la posición del bloque 4×4 (5.11).

$$pos\_x\_10\times10 = pos\_x\_4\times4 - 3 \qquad (5.11)$$

$$pos\_y\_10\times10 = pos\_y\_4\times4 - 3$$

El primer píxel del bloque 10×10 se ubica junto con 31 pixeles en la dirección definida por su posición (x,y) y por el número de columnas de bloques 16×16 en la SW (5.12).

$$dir\_inicio\_10\times10 = (num\_col\_bloques * pos\_y\_10\times10) + (pos\_x\_10\times10 >> 4) \qquad (5.12)$$

donde "num_col_bloques = (2 * Ph + 16) / 16 = (Ph/8) + 1 = (Ph >> 3) + 1". La generalización de la expresión (5.12), para calcular la dirección de cualquier renglón del

bloque 10×10, no solo del primero, define la ecuación de direcciones deseada (5.13), con la que se leen 32 pixeles de la memoria RAM_{SW} activa.

dir_renglón_10×10 = dir_inicio_10×10 + (renglón_10×10 * num_col_bloques) (5.13)

La posición exacta del inicio del renglón se define aplicando el resultado de la expresión (5.14) en la tabla 5.8, siendo el desplazamiento el valor que se envía al selector de pixeles, para seleccionar el renglón de 10 pixeles que procesa la unidad de interpolación por ciclo de reloj.

Desplazamiento = (pos_x_10×10) rem 16 (5.14)

Tabla 5.8. Posición exacta del inicio del primer renglón del bloque 10×10.

Desplazamiento	Posición del primer pixel en la RAM_{SW}
0	dato(7, downto 0) SRAM 0
1	dato(15 downto 8) SRAM 0
2	dato(23 downto 16) SRAM 0
3	dato(31 downto 24) SRAM 0
4	dato(7 downto 0) SRAM 1
5	dato(15 downto 8) SRAM 1
6	dato(23 downto 16) SRAM 1
7	dato(31 downto 24) SRAM 1
8	dato(7 downto 0) SRAM 2
9	dato(15 downto 8) SRAM 2
10	dato(23 downto 16) SRAM 2
11	dato(31 downto 24) SRAM 2
12	dato(7 downto 0) SRAM 3
13	dato(15 downto 8) SRAM 3
14	dato(23 downto 16) SRAM 3
15	dato(31 downto 24) SRAM 3

Como un ejemplo de la aplicación de las ecuaciones anteriores, se desea obtener la dirección en RAM_{SW} y la posición exacta del renglón 2, para una descomposición 4×4 con posición (41, 4) en SW, considerando rangos de desplazamiento Ph = Pv = 32. La tabla 5.9 muestra los datos de las primeras 20 localidades contenidos en la RAM_{SW} para los rangos indicados. La posición del bloque 10×10 en unidades de pixeles, aplicando (5.11) es:

pos_x_10×10 = pos_x_4×4 – 3 = 41 – 3 = 38
pos_y_10×10 = pos_y_4×4 – 3 = 4 – 3 = 1

Calculando la dirección del primer renglón con la ecuación (5.12):

dir_inicio_10×10 = (((Ph >> 3) + 1) * pos_y_10×10) + (pos_x_10×10 >> 4)
= (((32 >> 3) + 1) * 1) + (38 >> 4)
= 5 + 2 = 7

La dirección del renglón 2, de acuerdo a la ecuación (5.13) es:

dir_renglón_10×10 = dir_inicio_10×10 + (renglón_10×10 * ((Ph >> 3) + 1))
= 7 + 2*5 = 17

Tabla 5.9. Datos de las primeras 20 localidades de la RAM_{SW}, para un rango de desplazamiento Ph = 32, en el lado del puerto A o del puerto B.

Dirección	Datos SRAM 3	Datos SRAM 2	Datos SRAM 1	Datos SRAM 0
0	P(0,15), P(0,14), P(0,13), P(0,12)	P(0,11), P(0,10), P(0,9), P(0,8)	P(0,7), P(0,6), P(0,5), P(0,4)	P(0,3), P(0,2), P(0,1), P(0,0)
1	P(0,31), P(0,30), P(0,29), P(0,28)	P(0,27), P(0,26), P(0,25), P(0,24)	P(0,23), P(0,22), P(0,21), P(0,20)	P(0,19), P(0,18), P(0,17), P(0,16)
2	P(0,47), P(0,46), P(0,45), P(0,44)	P(0,43), P(0,42), P(0,41), P(0,40)	P(0,39), P(0,38), P(0,37), P(0,36)	P(0,35), P(0,34), P(0,33), P(0,32)
3	P(0,63), P(0,62), P(0,61), P(0,60)	P(0,59), P(0,58), P(0,57), P(0,56)	P(0,55), P(0,54), P(0,53), P(0,52)	P(0,51), P(0,50), P(0,49), P(0,48)
4	P(0,79), P(0,78), P(0,77), P(0,76)	P(0,75), P(0,74), P(0,73), P(0,72)	P(0,71), P(0,70), P(0,69), P(0,68)	P(0,67), P(0,66), P(0,65), P(0,64)
5	P(1,15), P(1,14), P(1,13), P(1,12)	P(1,11), P(1,10), P(1,9), P(1,8)	P(1,7), P(1,6), P(1,5), P(1,4)	P(1,3), P(1,2), P(1,1), P(1,0)
6	P(1,31), P(1,30), P(1,29), P(1,28)	P(1,27), P(1,26), P(1,25), P(1,24)	P(1,23), P(1,22), P(1,21), P(1,20)	P(1,19), P(1,18), P(1,17), P(1,16)
7	P(1,47), P(1,46), P(1,45), P(1,44)	P(1,43), P(1,42), P(1,41), P(1,40)	P(1,39), P(1,38), P(1,37), P(1,36)	P(1,35), P(1,34), P(1,33), P(1,32)
8	P(1,63), P(1,62), P(1,61), P(1,60)	P(1,59), P(1,58), P(1,57), P(1,56)	P(1,55), P(1,54), P(1,53), P(1,52)	P(1,51), P(1,50), P(1,49), P(1,48)
9	P(1,79), P(1,78), P(1,77), P(1,76)	P(1,75), P(1,74), P(1,73), P(1,72)	P(1,71), P(1,70), P(1,69), P(1,68)	P(1,67), P(1,66), P(1,65), P(1,64)
10	P(2,15), P(2,14), P(2,13), P(2,12)	P(2,11), P(2,10), P(2,9), P(2,8)	P(2,7), P(2,6), P(2,5), P(2,4)	P(2,3), P(2,2), P(2,1), P(2,0)
11	P(2,31), P(2,30), P(2,29), P(2,28)	P(2,27), P(2,26), P(2,25), P(2,24)	P(2,23), P(2,22), P(2,21), P(2,20)	P(2,19), P(2,18), P(2,17), P(2,16)
12	P(2,47), P(2,46), P(2,45), P(2,44)	P(2,43), P(2,42), P(2,41), P(2,40)	P(2,39), P(2,38), P(2,37), P(2,36)	P(2,35), P(2,34), P(2,33), P(2,32)
13	P(2,63), P(2,62), P(2,61), P(2,60)	P(2,59), P(2,58), P(2,57), P(2,56)	P(2,55), P(2,54), P(2,53), P(2,52)	P(2,51), P(2,50), P(2,49), P(2,48)
14	P(2,79), P(2,78), P(2,77), P(2,76)	P(2,75), P(2,74), P(2,73), P(2,72)	P(2,71), P(2,70), P(2,69), P(2,68)	P(2,67), P(2,66), P(2,65), P(2,64)
15	P(3,15), P(3,14), P(3,13), P(3,12)	P(3,11), P(3,10), P(3,9), P(3,8)	P(3,7), P(3,6), P(3,5), P(3,4)	P(3,3), P(3,2), P(3,1), P(3,0)
16	P(3,31), P(3,30), P(3,29), P(3,28)	P(3,27), P(3,26), P(3,25), P(3,24)	P(3,23), P(3,22), P(3,21), P(3,20)	P(3,19), P(3,18), P(3,17), P(3,16)
17	P(3,47), P(3,46), P(3,45), P(3,44)	P(3,43), P(3,42), P(3,41), P(3,40)	P(3,39), P(3,38), P(3,37), P(3,36)	P(3,35), P(3,34), P(3,33), P(3,32)
18	P(3,63), P(3,62), P(3,61), P(3,60)	P(3,59), P(3,58), P(3,57), P(3,56)	P(3,55), P(3,54), P(3,53), P(3,52)	P(3,51), P(3,50), P(3,49), P(3,48)
19	P(3,79), P(3,78), P(3,77), P(3,76)	P(3,75), P(3,74), P(3,73), P(3,72)	P(3,71), P(3,0), P(3,69), P(3,68)	P(3,67), P(3,66), P(3,65), P(3,64)
⋮	⋮	⋮	⋮	⋮

El desplazamiento (5.14) toma un valor 6, con el cual la tabla 5.8 da como resultado el dato (23 downto 16) en la SRAM 1, el cual corresponde al píxel P(3,38), que es el primer píxel del segundo renglón del bloque 10×10, derivado del bloque 4×4.

5.3.1.6 Selector de memorias y selector de pixeles.

El selector de memorias es el componente que elige la memoria RAM_{SW} que usó el acelerador IME en su último ciclo de operación y que contiene los 41 bloques próximos a estimar por el acelerador FME. El selector de pixeles accede al nivel de bytes a los datos de la memoria de búsqueda, para seleccionar los 10 pixeles/ciclo que procesa la unidad de interpolación en el cálculo de los sub-pixeles *half* y *quarter* (Figura 5.36).

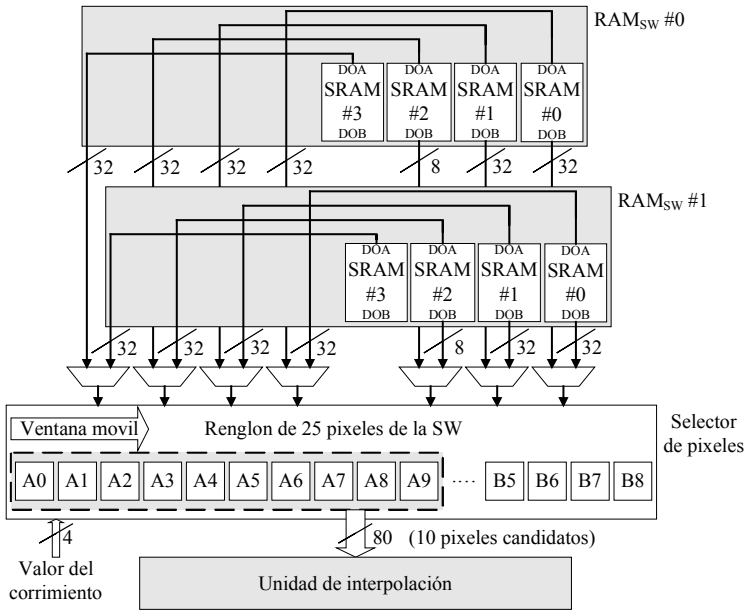


Figura 5.36. Arquitectura del selector de memorias y selector de pixeles.

Los 10 pixeles candidatos se seleccionan de la concatenación de los 16 pixeles del puerto A y de los 9 pixeles menos significativos del puerto B de la RAM_{SW} activa, en una operación tipo *barrel-shifter*, con el desplazamiento de una ventana móvil de 0 a 15 posiciones definido por la AGU_{SWf}. El acceso al puerto A se realiza con la dirección generada por la unidad de direccionamiento y al puerto B con la misma dirección incrementada una unidad. Esto permite disponer de dos renglones diferentes y sucesivos de la RAM_{SW} para efectuar la selección, con una lógica de control básica.

Para la selección de los 25 pixeles de las memorias RAM_{SW} se utilizan 6 multiplexores de 32 bits y uno de 8 bits, controlados por la unidad de control, de acuerdo a la información que se recibe del estimador IME.

5.3.1.7 Evaluación del diseño FME.

La evaluación del diseño FME se basa en la forma en que la aplicación de las técnicas y características estructurales y de funcionamiento, que se observaron en las arquitecturas del estado del arte, impactan en las variables de operación y rendimiento del diseño propuesto (nivel de reutilización de datos, porcentaje de uso de las unidades de procesamiento, área, velocidad de procesamiento, etc.).

La reutilización de datos mediante las técnicas de descomposición e integración vertical disminuye los requerimientos de ancho de banda y cómputo del estimador FME, al minimizar la redundancia en la manipulación y procesamiento de datos de la unidad de interpolación. La técnica de integración vertical de las descomposiciones 4×4 aplicada en este diseño reduce 25% el número total de accesos a la RAM_{SW}, gracias al descenso en un 30% de los pixeles que se procesan en el interpolador, cuando se integran 2 bloques 4×4 y de 45% en la integración de 4 bloques (Figura 5.20).

La aplicación de la técnica de descomposición 4×4, reduce ampliamente el área del estimador FME, al utilizar unidades de interpolación y distorsión SATD para los bloques de menor tamaño.

El diseño de la unidad de interpolación, con 16 filtros *half* (FIR 6-taps, 1-D separable *on-demand*) con realización en árbol sumador y 32 filtros *quarter* (bilineal, 2-D *on-demand*) paralelos, permite una utilización del 100%, a un bajo coste en área.

Gracias a la estructura de los módulos SATD, a partir de 2 HTs 4×4 de arquitectura paralela renglón-columna y una matriz de transposición, se tiene un diseño de bajo coste, que no incrementa excesivamente el área del estimador, cuando se replica 9 veces para paralelizar el cálculo de la distorsión.

El diseño presenta un flujo de datos continuo a la entrada de la sección de procesamiento, debido a la lectura de las memorias RAM_{SW} al nivel de palabra y de la manipulación de los datos al nivel de bytes, por el uso del módulo selector de pixeles.

El uso de una unidad de control global, logró reducir el número de ciclos falsos, al sincronizar a detalle todos los componentes de direccionamiento y procesamiento del estimador.

La paralelización de las etapas de interpolación y distorsión SATD, para procesar 9 bloques 4×4 simultáneamente, así como su operación continua sin acumulación de la latencia inicial, permite obtener una velocidad de procesamiento (Tabla 5.10), que mejora los resultados de arquitecturas con niveles equivalentes de complejidad [Chen04b].

Tabla 5.10. Duración en ciclos de reloj del procesamiento fraccionario de los 7 tamaños de bloque. Se incluyen los ciclos correspondientes a cada módulo de la ruta de datos.

	Número y tamaño de bloques	Ciclos de procesamiento			
		Interpolación	SATD	R/D	Total por bloques
	16 – 4×4	256	80	65	401
	8 – 4×8	146	96	35	277
	8 – 8×4	240	112	34	386
	4 – 8×8	156	80	17	253
	2 – 8×16	104	80	9	193
	2 – 16×8	144	88	9	241
	1 – 16×16	108	80	8	196
Total	**41**	**1154**	**616**	**177**	**1947**

Aun cuando la arquitectura propuesta presenta altos niveles de eficiencia y utilización, con un buen equilibrio área-velocidad, en su diseño se observan varias oportunidades para incrementar la velocidad de procesamiento, sin aumentar excesivamente el área utilizada. Puntos como la alta latencia de la unidad de interpolación (7 ciclos), el uso de la técnica de descomposición únicamente en 5 tamaños de bloques, el tiempo muerto en espera de la retroalimentación del número del mejor bloque *half* para iniciar el procesamiento *quarter* (10 ciclos) y el nivel de reutilización limitado por el tamaño de los bloques 4×4, pueden ser considerados como elementos de estudio para la optimización del acelerador FME. Es por ello que para complementar el diseño propuesto, al final del capítulo se presenta otra versión de la arquitectura FME, con enfoque hacia mayor velocidad, por medio de la modificación de la secuencia de procesamiento y el incremento de la paralelización de las etapas de interpolación y distorsión SATD.

5.3.2 Realización del diseño FME sobre FPGA.

La metodología de diseño del estimador FME-FSBM no enfoca su realización en una tecnología particular de sistemas digitales, por lo que puede utilizarse la que mejor se adapte a los recursos disponibles y al cumplimiento de los objetivos de tiempo y coste (ASIC, ASSP, FPGA, etc.). Con el propósito de integrar un sistema de estimación de movimiento, la arquitectura FSBM propuesta se realiza sobre la misma base tecnológica que el diseño IME del capítulo anterior, dispositivos FPGA de la familia Virtex-4 de Xilinx, con programación en VHDL y procesadores MicroBlaze y PowerPC, por lo que se siguen considerando las técnicas de diseño para comportamiento aplicadas previamente, el uso de las herramientas de Xilinx para síntesis (ISE 8.2i, sintetizador XST), diseño embebido (EDK 8.2i) y simulación (ModelSim Xilinx edition-III v6.1e) y el análisis elaborado para justificar el empleo de un dispositivo específico.

En las siguientes secciones se documenta la codificación RTL y la síntesis y realización sobre la FPGA Virtex-4 XC4VFX60-10-FF1152 de todos los componentes del acelerador FME. Se presentan algunas partes críticas del código VHDL y las

estadísticas de tiempo y área después de *Place & Route*, aplicando los valores por defecto de las propiedades de síntesis y realización del ISE 8.2i, sin definir restricciones de tiempo (si no se indica otra cosa). Las figuras que se anexan muestran la interacción de cada componente con su entorno, en función de sus señales de entrada y salida.

5.3.2.1 Unidad de control fraccionaria.

La unidad de control es el elemento que interactúa con el *buffer* de salida del estimador IME, el procesador local y los componentes de direccionamiento y procesamiento, para administrar la entrada de datos y parámetros de operación y sincronizar la operación global del acelerador FME, por medio de una máquina Moore y tres contadores binarios.

Diseñado como un elemento periférico, la unidad de control del acelerador FME puede conectarse a los procesadores embebidos de alto nivel que ofrece Xilinx, para crear sistemas hardware/software alrededor de sus FPGA: el procesador soft MicroBlaze y el procesador hard PowerPC 405. La conexión se realiza a través de los buses de propósito general que soportan ambos procesadores: el PLB y el OPB. La selección del procesador y bus para el estimador FME se realiza a la par con el acelerador IME, eligiendo los elementos que soporten la tasa de transferencia de los pixeles candidatos, desde la RAM externa de cuadros de referencia a las dos $RAM_{SW,}$ comunes a ambos estimadores. Ya que la tasa de transferencia depende de los parámetros de operación del codificador de video, se consideran las opciones de menor y mayor capacidad del par bus-procesadores (Figura 5.37).

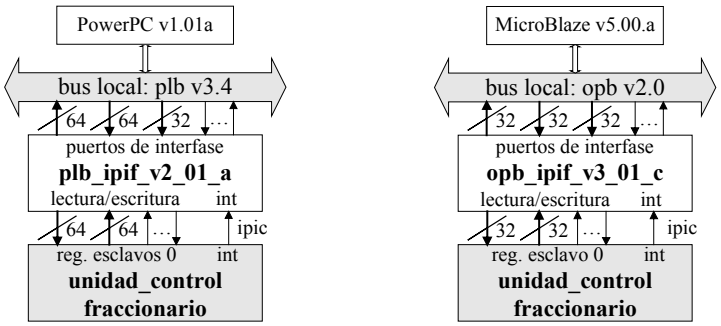


Figura 5.37. Interconexión de la unidad de control fraccionario con el procesador y bus local (MicroBlaze-OPB y PowerPC-PLB).

Los componentes que incluye la arquitectura FME, para conectar su unidad de control como periférico de los procesadores Xilinx embebidos son los puertos de interfase al bus, la interfase de propiedad intelectual (IPIF) para periféricos OPB o PLB y la lógica del usuario, que se conecta a la IPIF mediante un conjunto de puertos llamados interconexión de propiedad intelectual (IPIC).

El diseño de la interfase se realiza con la herramienta EDK 8.2i de Xilinx, en función de las necesidades de la lógica del usuario. Para el acelerador FME, se configura una interrupción por borde para indicar al procesador local el fin de la estimación fraccionaria y se especifica un registro esclavo de 32 bits para el bus OPB y de 64 bits para el bus PLB (Figura 5.38), con el que se desarrolla la transferencia de información entre el procesador y el acelerador hardware.

MSB 31 (63) ... 3	2 1	LSB 0
auxiliar	EEF	IEF
rst: 000000000000000.....00000000000000	11	1

IEF 0 Aprobación de inicio de estimación fraccionaria.
1 No aprobación de inicio de estimación fraccionaria.
EEF 00 Fin de estimación fraccionaria.
01 Estimación fraccionaria en proceso.
1x Estimación fraccionaria en espera de iniciar.

Figura 5.38. Especificación del registro esclavo de estado y supervisión de la estimación fraccionaria, bus local.

El ciclo fraccionario inicia cuando el acelerador IME valida la información en su *buffer* de salida, con la señal "datos_IME_val". La información se lee en forma paralela, excepto la posición en SW de los 41 bloques enteros de menor coste, la cual se recibe en forma serie, en el orden en que los bloques enteros son procesados, con el flujo controlado por la señal "desplazamiento_mejor_pos_IME" (Figura 5.39).

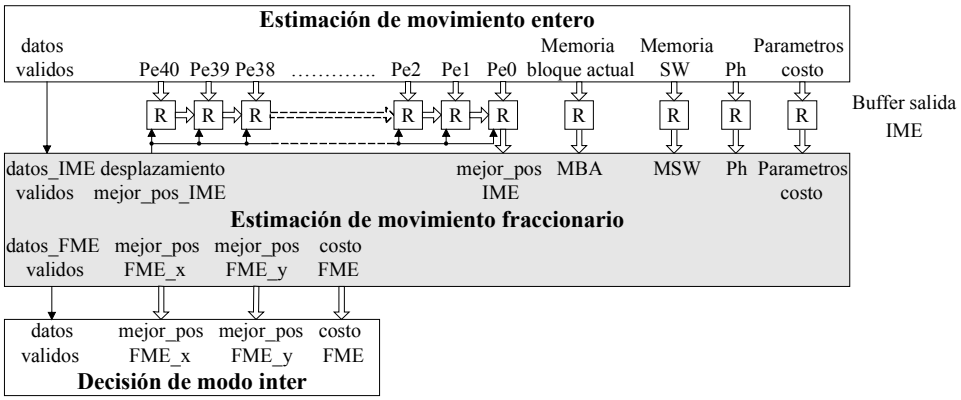


Figura 5.39. Detalle de la transferencia de información entre el acelerador IME y el estimador FME, a través del *buffer* de salida IME.

Otro requisito para el inicio del procesamiento fraccionario la define el procesador local, con el bit IEF del registro esclavo de supervisión y control. El procesador activa IEF si en el ámbito global del codificador de video no se presentan inconvenientes para iniciar el ciclo FME. El acelerador responde escribiendo su fase actual (en espera de inicio, en proceso o fin de ciclo), en los bits de estado de la estimación fraccionaria (EEF). El estado de fin del ciclo va acompañado con la interrupción al procesador local, para señalar la culminación correcta de su operación y la validez de los resultados obtenidos.

La unidad de control interactúa con los componentes de direccionamiento y procesamiento del acelerador hardware (Figura 5.40) para enviar y recibir los parámetros de configuración y las señales de control y estado, las cuales se describen en la tabla 5.11, para una conexión con el procesador MicroBlaze.

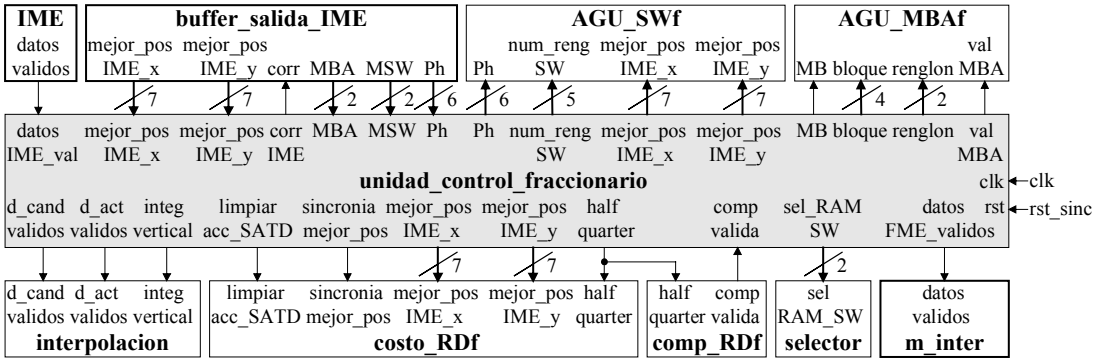


Figura 5.40. Señales de la unidad de control fraccionario y su interconexión con los componentes del sistema de estimación.

Tabla 5.11. Descripción de las señales de E/S de la unidad de control fraccionaria, para una aplicación con el procesador MicroBlaze.

Señal	Tipo	Descripción
clk	E	Reloj, con operación en su borde de subida
rst	E	Reset síncrono, activo en bajo
Bus2IP_Data(0 to 31)	E	Datos del bus a la *IPIF* (*OPB* local)
Bus2IP_BE(0 to 3)	E	Habilitación de bytes del bus a la *IPIF* (*OPB* local)
Bus2IP_RdCE(0 to 0)	E	Habilitación de lectura, del bus a la *IPIF* (*OPB* local)
Bus2IP_WrCE(0 to 0)	E	Habilitación de escritura, del bus a la *IPIF* (*OPB* local)
datos_IME_val	E	Validación de los datos del buffer de salida IME
mejor_pos_IME_x(6 downto 0)	E	Coordenada x del mejor bloque entero en unidades de pixeles
mejor_pos_IME_y(6 downto 0)	E	Coordenada y del mejor bloque entero en unidades de pixeles
MBA(1 downto 0)	E	Posición del MB actual en la RAM_{MBA} (0-posición #0, 1-posición #1, 2 y 3-posición no valida)
MSW(1 downto 0)	E	Número de la RAM_{SW} (0- #0, 1-#1, 2,3-no valida)
Ph(5 downto 0)	E	Desplazamiento horizontal en unidades de pixeles
hab_MBA	E	Habilitación de la lectura de la RAM_{MBA}
comp_valida	E	Resultados de comparación R/D valida
IP2Bus_IntrEvent(0 to 0)	S	Evento de interrupción de la *IPIF* al bus (*OPB* local)
IP2Bus_Data(0 to 31)	S	Datos de la *IPIF* al bus (*OPB* local)
IP2Bus_Ack	S	Reconocimiento de la *IPIF* al bus (*OPB* local)
IP2Bus_Retry	S	Respuesta de recuperación de la *IPIF* al bus (*OPB* local)
IP2Bus_ToutSup	S	Supresión de tiempo fuera de la *IPIF* al bus (*OPB* local)
corr_IME	S	Control del registro de desplazamiento del buffer de salida IME
Ph(5 downto 0)	S	Desplazamiento horizontal en unidades de pixeles
num_renglon_SW(4 downto 0)	S	Renglon que se lee del bloque candidato 10×10 en RAM_{SW}
mejor_pos_IME_x(6 downto 0)	S	Coodenada x del mejor bloque entero en unidades de pixeles
mejor_pos_IME_y(6 downto 0)	S	Coodenada y del mejor bloque entero en unidades de pixeles
MB	S	Número de MB actual (0 ó 1)
bloque(3 downto 0)	S	Número bloque 4×4 del MB actual a evaluar (15 a 0)
renglón(1 downto 0)	S	Número de renglón del bloque 4×4 a leer (3 a 0)
d_cand_validos	S	Validación de los pixeles candidatos
d_act_validos	S	Validación de lo pixeles actuales
integ_vertical	S	Integración vertical (0-si integración, 1-no)
limpiar_acc_SATD	S	Limpiar acumulación valores SATD de bloques 4×4
sincronia_mejor_pos	S	Sincronia de los datos de entrada de mejor posición
half_quarter	S	Ciclo de procesamiento actual (0-half, 1-quarter)
sel_RAM_SW(1 downto 0)	S	Selección de la RAM_{SW} (00-#0, 01#1, 1x-ninguna)
datos_FME_validos	S	Validación de resultados de estimación quarter
comp_valida	S	Validación resultado de comparación R/D (half o quarter)

La máquina de estados que define las señales de control se realiza como una FSM Moore, utilizando 151 estados de tipo enumerado y tres procesos VHDL (decodificador del siguiente estado, actualización del estado presente y decodificador de salidas), con un proceso final para registrar las salidas y limitar la concatenación de niveles lógicos. Los contadores binarios evalúan la secuencia de bloques y renglones actuales y candidatos para definir las condiciones de cambio de estados. A manera de ejemplo, la figura 5.41 muestra la secuencia de decodificación del siguiente estado, para el procesamiento de los bloques 4×8, donde se observa la aplicación de la técnica de integración vertical de 2 descomposiciones 4×4 y la ejecución de la secuencia de estados, hasta procesar en resolución *half* y *quarter* los 8 bloques 4×8.

Como un resultado preliminar de su realización, la tabla 5.12 muestra las estadísticas post-P&R de la codificación de la unidad control, considerando la interfase IPIF y los puertos IPIC que lo conectan al bus OPB, para su operación con el procesador MicroBlaze.

5.3.2.2 Unidad de interpolación.

La unidad de interpolación genera los sub-pixeles *half* y *quarter* de los 8 bloques alrededor del mejor bloque 4×4 de resolución previa y sincroniza los resultados para que los módulos de distorsión SATD los reciban junto con los pixeles del bloque actual. Su realización se deriva directamente de la arquitectura propuesta (Figura 5.28), donde los pixeles enteros y los sub-pixeles *half* y *quarter* representan los registros del *buffer* de interpolación, en el que fluyen los datos hacia los elementos de procesamiento.

Los filtros de interpolación *half* FIR_H, FIR_{V1} y FIR_{V2} se desarrollan usando las siguientes funciones en VHDL:

*half*_intermedio = pel_iii − 5 pel_ii + 20 pel_i + 20 pel_d − 5 pel_dd + pel_ddd + 16

*half*_end = Clip1((*half*_intermedio) >> 5)

half_half = Clip1((pel_iii_intermed − 5 pel_ii_intermed + 20 pel_i_intermed
+ 20 pel_d_intermed - 5 pel_dd_intermed + pel_ddd_intermed
) >> 10)

donde la función Clip1 recorta su argumento entre 0 y 255 y los datos pel_iii, pel_ii, pel_i y pel_d, pel_dd, pel_ddd son los pixeles a la izquierda (ó arriba) y a la derecha (ó abajo) del sub-píxel a generar. La función "*half*_intermedio" (Figura 5.42) es la base para derivar los filtros FIR_H y FIR_{V1}, con el apoyo de la función "*half*_end". El filtro FIR_{V2} se realiza con la función "*half_half*", utilizando los valores intermedios de los sub-pixeles *half* verticales.

Los bloques *quarter* se generan a partir de 5 renglones de pixeles enteros y sub-pixeles *half*, donde una fábrica de conmutación dirige los datos adecuados a los filtros bilineales, de acuerdo al número del mejor bloque *half* (0 a 8), determinado por el módulo de comparación R/D. En la conexión del interpolador con otros componentes (Figura 5.43), se observa el flujo de datos definido por las señales de E/S (Tabla 5.13), desde los elementos que proveen los pixeles actuales y candidatos, hasta los componentes que reciben los sub-pixeles *half* y *quarter*.

```
when inic_half_4x8              =>  -- Inicio de procesamiento fraccionario 4x8.
   next_state <= half_4x8;
when half_4x8                   =>  -- Ciclo de procesamiento half.
   next_state <= cargar_half_00_4x8;
when cargar_half_00_4x8         =>  -- Cargar bloque actual 4x4 inferior y 1a parte bloque candidato.
   if (cont_reng_cand < "01010") then
      next_state <= cargar_half_01_4x8;
   end if;
when cargar_half_01_4x8         =>  -- Cargar segunda parte bloque candidato.
   if (cont_reng_cand = "00011") then
      next_state <= esperar_half_00_4x8;
   end if;
when esperar_half_00_4x8        =>  -- Espera para sincronía de datos.
   if (cont_reng_cand = "00001") then
      next_state <= esperar_half_01_4x8;
   end if;
when esperar_half_01_4x8        =>  -- Espera para sincronía de señales.
   next_state <= cargar_half_10_4x8;
when cargar_half_10_4x8         =>   -- Cargar bloque actual 4x4 superior y bloque candidato, en el
   if (cont_reng_cand = "11111") then  -- proceso de integración vertical.
      next_state <= esperar_half_quarter_4x8;
   end if;
when esperar_half_quarter_4x8 =>  -- Esperar para sincronía entre half y quarter.
   if (cont_reng_cand <= "11010") then
      next_state <= inic_quarter_4x8;
   end if;
when inic_quarter_4x8           =>   -- Inicio de procesamiento quarter 4x8.
   next_state <= load_quarter_00_4x8;
when load_quarter_00_4x8        =>   -- Cargar bloque actual 4x4 inferior y 1a parte bloque candidato.
   if (cont_reng_cand < "01010") then
      next_state <= cargar_quarter_01_4x8;
   end if;
when cargar_quarter_01_4x8    =>   -- Cargar segunda parte bloque candidato.
   if (cont_reng_cand = "00011") then
      next_state <= esperar_quarter_00_4x8;
   end if;
when esperar_quarter_00_4x8  =>   -- Espera para sincronía de datos.
   if (cont_reng_cand = "00001") then
      next_state <= esperar_quarter_01_4x8;
   end if;
when esperar_quarter_01_4x8  =>   -- Espera para sincronía de señales.
   next_state <= cargar_quarter_10_4x8;
when cargar_quarter_10_4x8   =>    -- Cargar bloque actual 4x4 superior y bloque candidato, en el
   if (cont_reng_cand = "11111") then   --  proceso de integración vertical.
      next_state <= esperar_quarter_1_4x8;
   end if;
when esperar_quarter_1_4x8   =>    -- Esperar fin procesamiento quarter.
   if (cont_reng_cand < "11010") then
      next_state <= ciclo_bloque_4x8;
   end if;
when ciclo_bloque_4x8          =>    -- Verificar fin de procesamiento bloques 4x8.
   if (cont_bloques = "1100") then
      next_state <= inic_half_8x4;    -- Fin 4x8, iniciar procesamiento bloques 8x4.
   else
      next_state <= half_4x8;          -- No, continuar con el siguiente bloque 4x8.
   end if;
```

Figura 5.41. Sección del programa en VHDL de la unidad de control, donde se muestra la decodificación del siguiente estado para el procesamiento de los bloques 4×8.

Tabla 5.12. Estadísticas de síntesis y realización post-P&R de la unidad de control.

Recurso/especificación	Valor
Slices	256
Flip-flops	186
4-input LUTs	468
Puertas equivalentes	4657
Frecuencia (MHz)	121

```
-- half_intermedio = pel_iii – 5 pel_ii + 20 pel_i + 20 pel_d – 5 pel_dd + pel_ddd + 16

function interpol_half_intermed( pel_iii, pel_ii, pel_i,
   pel_d, pel_dd, pel_ddd: std_logic_vector(7 downto 0) ) return std_logic_vector is

   variable suma_0, suma_1, suma_2 : std_logic_vector(8 downto 0);
   variable suma_00 : std_logic_vector(9 downto 0);
   variable suma_3   : std_logic_vector(11 downto 0);
   variable suma_4   : std_logic_vector(12 downto 0);
   variable resultado_intermed : std_logic_vector(14 downto 0);

   begin
      suma_0  := ('0' & pel_iii) + ('0' & pel_ddd);
      suma_00:= ('0' & suma_0) + "0000010000";
      suma_1  := ('0' & pel_ii) + ('0' & pel_dd);
      suma_2  := ('0' & pel_i) + ('0' & pel_d);
      suma_3  := ('0' & suma_2 & "00") - ( "000" & suma_1);
      suma_4  := ("000" & suma_00) + (suma_3(11) & suma_3);
      resultado_intermed := (suma_4(12) & suma_4(12) & suma_4) +
                            (suma_3(11) & suma_3 & "00");
      return (resultado_intermed);
   end interpol_half_intermed;
```

Figura 5.42. Código del filtro FIR de 6-taps intermedio, programado como una función VHDL.

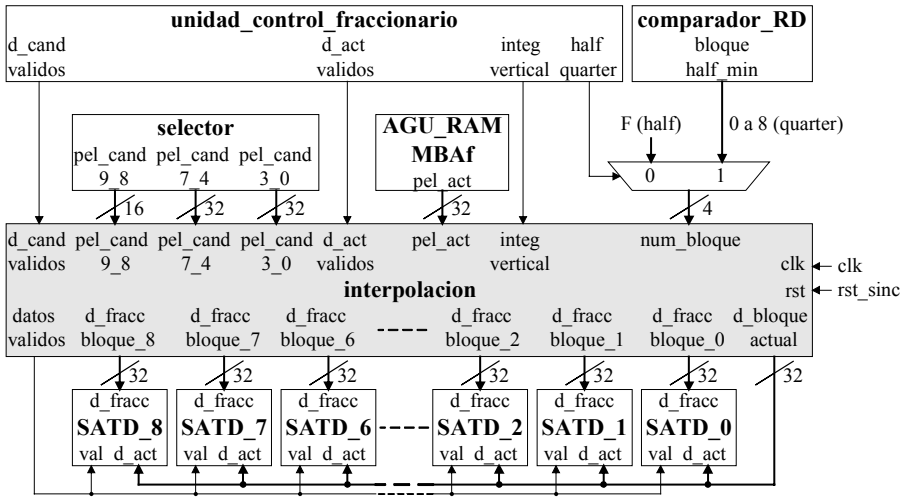


Figura 5.43. Unidad de interpolación y su interacción con los otros bloques del estimador FME.

Tabla 5.13. Descripción de las señales de E/S de la unidad de interpolación.

Señal	Tipo	Descripción
clk	E	Reloj, con operación en su borde de subida
rst	E	Reset síncrono, activo en bajo
d_cand_validos	E	Validación de los datos del bloque candidato 10×10
pel_cand_3_0(31 downto 0)	E	Pixeles 3, 2, 1 y 0 de un renglón del bloque candidato
pel_cand_7_4(31 downto 0)	E	Pixeles 7, 6, 5 y 4 de un renglón del bloque candidato
pel_cand_8_9(7 downto 0)	E	Pixeles 8 y 9 de un renglón del bloque candidato
d_act_validos	E	Validación de los datos del bloque actual 4×4
pel_act(31 downto 0)	E	Pixeles de un renglón del bloque actual
integ_vertical	E	Integración vertical (0-si integración, 1-no)
num_bloque(3 downto 0)	E	Número de bloque a procesar (F-half, 0 a 8-quarter)
datos_validos	S	Validación de los resultados de interpolación
d_bloque_actual(31 downto 0)	S	Pixeles de un renglón del bloque actual 4×4
d_fracc_bloque_0(31 downto 0)	S	Sub-pixeles de un renglón del bloque candidato 0
:	:	:
d_fracc_bloque_7(31 downto 0)	S	Sub-pixeles de un renglón del bloque candidato 7
d_fracc_bloque_8(31 downto 0)	S	Sub-pixeles de un renglón del bloque candidato 8

Las estadísticas post-P&R (Tabla 5.14), establecidas con los valores por defecto de las propiedades de síntesis y realización del ISE 8.2i y sin definir restricciones de tiempo, muestran un componente de gran densidad y baja frecuencia de operación, debido principalmente a la cantidad de filtros FIR con realización en un solo bloque combinacional, que evita el incremento y concatenación de latencias. Para cumplir con la meta de velocidad, se necesitará aplicar en el sistema completo las propiedades de optimización de la herramienta de síntesis y realización y la estipulación de restricciones de tiempo locales y globales.

Tabla 5.14. Estadísticas de síntesis y realización post-P&R de la unidad de interpolación.

Recurso/especificación	Valor
Slices	3072
Flip-flops	2239
4-input LUTs	4248
Puertas equivalentes	60098
Frecuencia (MHz)	86.89

5.3.2.3 Módulo de distorsión SATD.

El módulo SATD calcula el valor de la distorsión entre un bloque 4×4 de pixeles enteros y un bloque de sub-pixeles del mismo tamaño, en una ruta de datos que inicia con el cálculo de diferencias y continúa con la transformada Hadamard 2D y la suma de los valores transformados absolutos, para finalizar con la división entre 2 del resultado.

La realización hardware del diseño SATD busca optimizar el binomio área-velocidad por medio de un buen estilo de codificación VHDL y una reducción en el número de niveles lógicos, sin incrementar significativamente la latencia. Como un ejemplo, la figura 5.44 muestra el código de cálculo de diferencias y primera HT rápida, con algoritmo de mariposa. Este es un proceso puramente combinacional donde se entra con señales, se ejecutan las operaciones iniciales e intermedias usando variables y las

operaciones finales se asignan a señales. En este caso no se segmentó la lógica porque el número de niveles se mantuvo dentro del margen para obtener una alta frecuencia de operación.

```
process(datos_act, datos_fracc)
   variable diff_0, diff_1, diff_2, diff_3 : std_logic_vector(8 downto 0);
   variable m_0, m_1, m_2, m_3 : std_logic_vector(9 downto 0);
   begin
      diff_0 := ('0' & datos_act(7 downto 0))     – ('0' & datos_fracc(7 downto 0));
      diff_1 := ('0' & datos_act(15 downto 8))   – ('0' & datos_fracc(15 downto 8));
      diff_2 := ('0' & datos_act(23 downto 16)) – ('0' & datos_fracc(23 downto 16));
      diff_3 := ('0' & datos_act(31 downto 24)) – ('0' & datos_fracc(31 downto 24));

      m_0   := (diff_0(8) & diff_0) + (diff_3(8) & diff_3);
      m_1   := (diff_1(8) & diff_1) + (diff_2(8) & diff_2);
      m_2   := (diff_1(8) & diff_1) – (diff_2(8) & diff_2);
      m_3   := (diff_0(8) & diff_0) – (diff_3(8) & diff_3);

      had_1D_der_0 <= (m_0(9) & m_0) + (m_1(9) & m_1);
      had_1D_der_1 <= (m_2(9) & m_2) + (m_3(9) & m_3);
      had_1D_der_2 <= (m_0(9) & m_0) – (m_1(9) & m_1);
      had_1D_der_3 <= (m_3(9) & m_3) – (m_2(9) & m_2);
   end process;
```

Figura 5.44. Código VHDL del cálculo de diferencias y de la transformada Hadamard 1D derecha.

Otras secciones del diseño si fueron segmentadas, agregando registros para limitar el número de niveles lógicos y evitar concatenar la lógica combinacional del módulo SATD con los componentes posteriores (Figura 5.45), lo que permite cumplir con la frecuencia de operación deseada, al coste de aumentar la latencia de 4 a 6 ciclos de reloj.

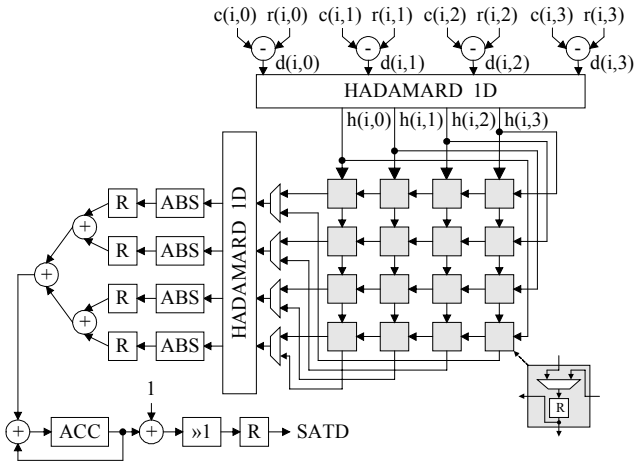


Figura 5.45. Arquitectura del módulo SATD con los registros de segmentación para reducir el número de niveles lógicos.

Al ubicarse en la cadena *pipeline* de procesamiento, el módulo SATD interactúa con los módulos de interpolación y costo_RD (Figura 5.46), sincronizando sus operaciones con las señales de validación de los datos.

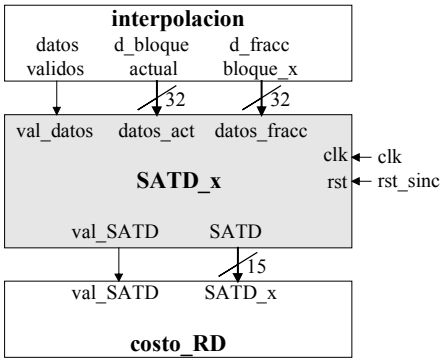


Figura 5.46. Módulo SATD y su conexión con los componentes de procesamiento del acelerador FME.

La tabla 5.15 muestra la descripción de las señales de E/S y la 5.16 sus estadísticas post-P&R, donde se observa que aún cuando se optimizó el área, esta es significativa, tomando en cuenta que el componente SATD se instancia 9 veces en el código de la arquitectura.

Tabla 5.15. Descripción de las señales de E/S de la unidad del módulo SATD.

Señal	Tipo	Descripción
clk	E	Reloj, con operación en su borde de subida
rst	E	Reset síncrono, activo en bajo
val_datos	E	Validación de los pixeles actuales y fraccionarios
datos_act(31 downto 0)	E	Pixeles de un renglón del bloque 4×4 actual
datos_fracc(31 downto 0)	E	Pixeles de un renglón del bloque 4×4 fraccionario
val_SATD	S	Validación del resultado SATD
SATD	S	Resultado de distorsión SATD del bloque 4×4

Tabla 5.16. Estadísticas de síntesis y realización post-P&R del módulo de distorsión SATD.

Recurso/especificación	Valor
Slices	316
Flip-flops	261
4-input LUTs	576
Puertas equivalentes	7375
Frecuencia (MHz)	119

5.3.2.4 Componentes fraccionarios de coste y comparación R/D.

El bloque con resolución fraccionaria de menor coste, alrededor del mejor bloque entero o *half*, se define en un procesamiento *pipeline* que incluye un componente de coste que acumula los valores SATD de las descomposiciones 4×4 de los bloques candidatos y calcula el coste de transmitir el MV fraccionario y de otro componente que compara el coste R/D de los 9 bloques aspirantes para determinar el que más se asemeje al bloque original.

La realización de los elementos de coste inicia con el estudio de su interacción con los componentes del acelerador hardware (Figura 5.47) y la determinación de sus

puertos de entrada y salida, que definen las señales de datos y control que ambos módulos necesitan para su operación (Tablas 5.17 y 5.18).

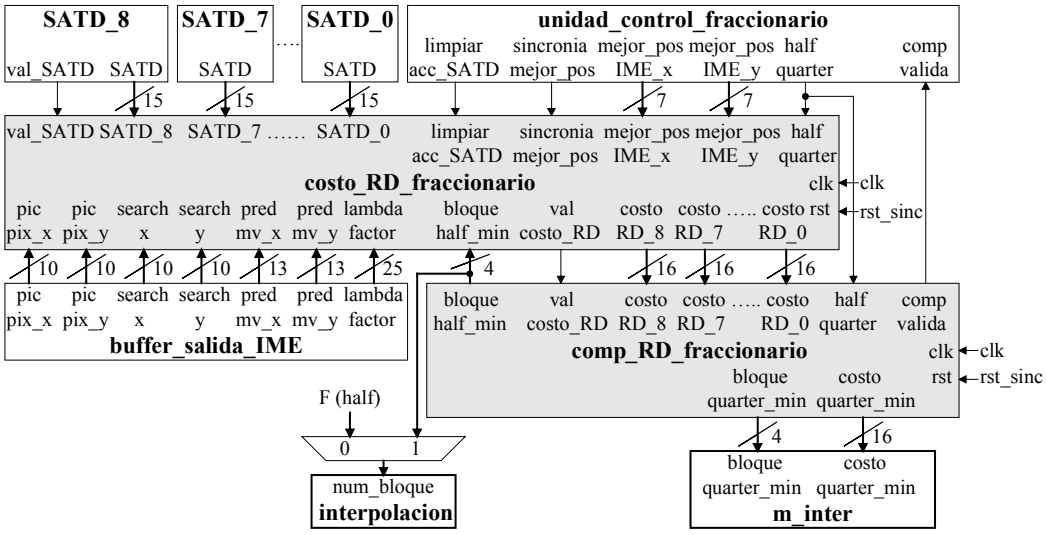


Figura 5.47. Componentes fraccionarios de coste y comparación R/D y su interacción con los otros elementos del acelerador FME y con el módulo de decisión de inter-predicción (m_inter).

Tabla 5.17. Descripción de las señales de E/S del componente de coste R/D fraccionario.

Señal	Tipo	Descripción
clk	E	Reloj, con operación en su borde de subida
rst	E	Reset síncrono, activo en bajo
val_SATD	E	Validación de los valores de distorsión SATD
SATD_0(14 downto 0)	E	Distorsión SATD descomposición 4×4 0
:	:	:
SATD_7(14 downto 0)	E	Distorsión SATD descomposición 4×4 7
SATD_8(14 downto 0)	E	Distorsión SATD descomposición 4×4 8
limpiar_acc_SATD	E	Limpiar acumulación valores SATD de bloques 4×4
sincronia_mejor_pos	E	Sincronia de los datos de entrada de mejor posición
mejor_pos_IME_x(6 downto 0)	E	Coordenada x del mejor bloque entero en unidades de pixeles
mejor_pos_IME_y(6 downto 0)	E	Coordenada y del mejor bloque entero en unidades de pixeles
half_quarter	E	Ciclo de procesamiento actual (0-half, 1-quarter)
pic_pix_x(10 downto 0)	E	Coordenada x del MB actual en unidades de pixeles
pic_pix_y(9 downto 0)	E	Coordenada y del MB actual en unidades de pixeles
search_x(10 downto 0)	E	Coordenada x de la SW en unidades de pixeles
search_y(9 downto 0)	E	Coordenada y de la SW en unidades de pixeles
pred_mv_x(12 downto 0)	E	Valor x de la predicción del MV, en unidades de ¼ de pixel
pred_mv_y(12 downto 0)	E	Valor y de la predicción del MV, en unidades de ¼ de píxel
lambda_factor(23 downto 0)	E	Equivalente entero de λ_{SAD}
bloque_half_min(3 downto 0)	E	Número del bloque half de coste mínimo
val_costo_RD	S	Validación de los resultados de coste R/D
costo_RD_0(15 downto 0)	S	Coste R/D bloque fraccionario candidato 0
:	:	:
costo_RD_7(15 downto 0)	S	Coste R/D bloque fraccionario candidato 7
costo_RD_8(15 downto 0)	S	Coste R/D bloque fraccionario candidato 8

Tabla 5.18. Descripción de las señales de E/S del componente de comparación R/D fraccionario.

Señal	Tipo	Descripción
clk	E	Reloj, con operación en su borde de subida
rst	E	Reset síncrono, activo en bajo
val_costo_RD	E	Validación de los resultados de coste R/D
costo_RD_0(15 downto 0)	E	Coste R/D bloque fraccionario candidato 0
:	:	:
costo_RD_7(15 downto 0)	E	Coste R/D bloque fraccionario candidato 7
costo_RD_8(15 downto 0)	E	Coste R/D bloque fraccionario candidato 8
half_quarter	E	Ciclo de procesamiento actual (0-half, 1-quarter)
comp_valida	S	Resultado de comparación half/quarter valido
bloque_half_min(3 downto 0)	S	Número del bloque half de coste mínimo
bloque_quarter_min(3 downto 0)	S	Número del bloque quarter de coste mínimo
costo_quarter_min(15 downto 0)	S	Coste R/D del mejor bloque quarter

La documentación de la ubicación de los componentes R/D en el entorno del estimador FME, facilita la organización de su codificación VHDL, al conjuntar todas las señales que intervienen en la lógica deseada, por lo que se pueden planear la aplicación de técnicas de comportamiento, principalmente en el módulo de coste, por la gran cantidad de información que procesa.

La operación de los módulos inicia con la activación de las señales "sincronia_mejor_pos" y "val_SATD", las cuales indican la validez de los parámetros de posición y coste y de los valores de distorsión SATD de las descomposiciones 4×4 respectivamente. Al terminar de recibir la distorsión del bloque completo, el primer módulo transmite al comparador los 9 resultados de coste, validados por la señal "val_costo_RD", con una latencia de 2 ciclos. El comparador determina el número de bloque candidato de menor coste y dispone sus resultados en sus puertos de salida, validados por la señal "comp_valida", después de 2 ciclos de latencia. Si los resultados corresponden al mejor bloque *half*, el número de bloque es enviado indirectamente a la unidad de interpolación. Si los resultados son del mejor bloque *quarter*, la unidad de control activa la señal "datos_FME_validos", para indicar al módulo de inter-predicción (m_inter) que respalde la información de número y coste del mejor bloque *quarter* y la posición (x,y) del bloque entero del cual proviene.

Finalizada la programación, las herramientas de simulación auxilian en la depuración del código para que cumpla con la funcionalidad esperada. Las estadísticas post-P&R (Tablas 5.19 y 5.20) muestran resultados satisfactorios de área y velocidad, que todavía pueden ser mejorados, al aplicar las propiedades de síntesis y realización y las restricciones de tiempo en el sistema completo,

Tabla 5.19. Estadísticas de síntesis y realización post-P&R del módulo de coste R/D.

Recurso/especificación	Valor
Slices	774
Flip-flops	243
4-input LUTs	1508
DSP48s	18
Puertas equivalentes	16574
Frecuencia (MHz)	93.1

Tabla 5.20. Estadísticas de síntesis y realización post-P&R del módulo de comparación R/D.

Recurso/especificación	Valor
Slices	135
Flip-flops	38
4-input LUTs	262
Puertas equivalentes	2738
Frecuencia (MHz)	93.64

5.3.2.5 AGU fraccionaria de la RAM_{MBA}.

En el diseño de la unidad de manejo fraccionaria de la memoria del MB actual (AGU_{MBAf}), se especificó una lógica combinacional que apoya a la FSM de la unidad de control, para facilitar la lectura de cada renglón de 4 pixeles, de uno de los 16 bloques 4×4, de cualquiera de las dos posiciones del MB actual en la $RAM_{MBA.}$ La lógica consiste en la definición de la ecuación de direcciones a partir de la información que envía y valida la unidad de control y del multiplexor de BRAMs, que selecciona la columna de la RAM_{MBA} donde se ubica el bloque 4×4 que se transmite a la unidad de interpolación (Figura 5.48).

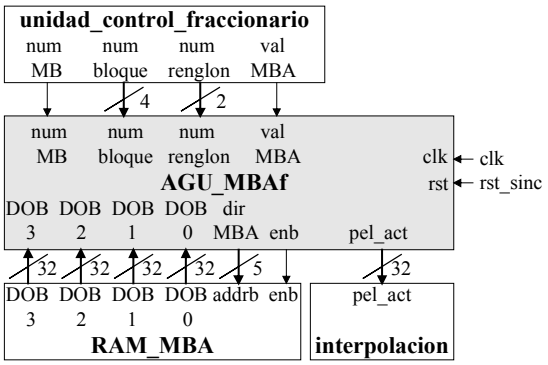


Figura 5.48. Unidad AGU_{MBAf} y su conexión con los módulos del acelerador FME y la memoria RAM_{MBA}.

La ecuación de direcciones, compuesta por un multiplexor y dos sumadores (Figura 5.33), equivale a una concatenación del número de MB, bloque y renglón (5.15). El multiplexor de datos se codifica en VHDL en un proceso combinacional con la sentencia *case*. Las salidas del módulo son registradas para truncar la sucesión de niveles lógicos, produciendo una latencia en la lectura de la RAM_{MBA} de 1 ciclo de reloj.

dir_MBA <= num_MB & num_bloque(3 downto 2) & num_renglón; (5.15)

La tabla 5.21 muestra la descripción de las señales de entrada y salida, y la 5.22 las estadísticas post-P&R de la síntesis y realización bajo el software ISE 8.2i, donde se observa un consumo mínimo de los recursos de la FPGA y una frecuencia no definida debido a la naturaleza combinacional del componente.

Tabla 5.21. Descripción de las señales de E/S del componente de la AGU_{MBAf}.

Señal	Tipo	Descripción
clk	E	Reloj, con operación en su borde de subida
rst	E	Reset síncrono, activo en bajo
val_MBA	E	Validación de datos de entrada
num_MB	E	Posición del MB actual en la RAM_{MBA} (0 ó 1)
num_bloque(3 downto 0)	E	Número bloque 4×4 a procesar (0 a 15).
num_renglon(1 downto 0)	E	Número del renglón (0 a 3) del bloque 4×4 a procesar
DOB_0(31 downto 0)	E	Puerto B de datos, BRAM 0, memoria RAM_{MBA}
DOB_1(31 downto 0)	E	Puerto B de datos, BRAM 1, memoria RAM_{MBA}
DOB_2(31 downto 0)	E	Puerto B de datos, BRAM 2, memoria RAM_{MBA}
DOB_3(31 downto 0)	E	Puerto B de datos, BRAM 3, memoria RAM_{MBA}
enb	S	Habilitación de la lectura del puerto B, RAM_{MBA}
dir_MBA	S	Direccion de lectura de la RAM_{MBA}
pel_act	S	Renglon de pixeles actuales del bloque 4×4 a procesar

Tabla 5.22. Estadísticas de síntesis y realización post-P&R del componente AGU_{MBAf}.

Recurso/especificación	Valor
Slices	32
Flip-flops	37
4-input LUTs	64
Puertas equivalentes	776
Frecuencia (MHz)	-

5.3.2.6 AGU fraccionaria de la RAM_{SW}.

La unidad fraccionaria generadora de direcciones de la memoria RAM_{SW} (AGU_{SWf}), calcula la dirección de un grupo de 32 pixeles de la línea de la SW donde se ubica un renglón del bloque 10×10 o de los bloques integrados verticalmente y determina la posición del primer píxel del renglón en la línea direccionada. Su realización parte del estudio de su ubicación en el acelerador FME (Figura 5.49) y la descripción de sus señales de E/S (Tabla 5.23).

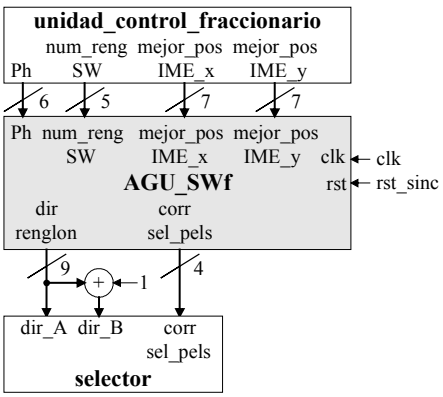


Figura 5.49. Unidad de direcciones AGU_{SWf} y su conexión con los restantes módulos del acelerador FME.

Su codificación VHDL consiste en dos procesos combinacionales, donde se programan las ecuaciones de dirección y desplazamiento que se desarrollaron en el

apartado de diseño. También incluye un proceso secuencial para el registro de salidas, que limita el encadenamiento de los niveles lógicos entre componentes, pero que produce una latencia de 1 ciclo en la lectura de la memoria de la SW.

Tabla 5.23. Descripción de las señales de E/S del componente AGU_{SWf}.

Señal	Tipo	Descripción
clk	E	Reloj, con operación en su borde de subida
rst	E	Reset síncrono, activo en bajo
Ph(5 downto 0)	E	Desplazamiento de búsqueda horizontal
num_renglón_SW(4 downto 0)	E	Número de renglón de los bloques verticales en la SW
mejor_pos_IME_x(6 downto 0)	E	Posición x del mejor bloque entero
mejor_pos_IME_y(6 downto 0)	E	Posición y del mejor bloque entero
dir_renglon(8 downto 0)	S	Dirección en RAM_{SWF} del renglón indicado
corr_sel_pels(3 downto 0)	S	Desplazamiento para el selector de pixeles

Las estadísticas post-P&R (Tabla 5.24) muestran un bajo consumo de los *slices* del dispositivo Virtex-4, con una frecuencia no definida. Esto se debe a que solo se incluye la sección combinacional de la AGU, ya que las señales de control de la memoria se especifican en la FSM de la unidad de control del estimador fraccionario.

Tabla 5.24. Estadísticas de síntesis y realización post-P&R del componente AGU_{SWf}.

Recurso/especificación	Valor
Slices	59
Flip-flops	13
4-input LUTs	113
Puertas equivalentes	1082
Frecuencia (MHz)	-

5.3.2.7 Selector de memorias y pixeles.

El módulo selector es el componente que incluye la ruta de datos para seleccionar la memoria RAM_{SW} donde se ubican los mejores bloques enteros, y en la cual se leen los 10 pixeles que procesa la unidad de interpolación por ciclo de reloj. En el inicio de operaciones, la unidad de control indica el número de RAM_{SW} valida y el selector lee 25 pixeles de la memoria elegida, de los cuales se toman 10, de acuerdo a la dirección y al valor de desplazamiento de la ventana de selección de pixeles que indica la AGU_{SWf}. Previamente se configura la memoria seleccionada para leer simultáneamente sus puertos A y B y disponer de los pixeles necesarios. En la figura 5.50 se observan los componentes de estimador FME que intervienen en el proceso.

La selección de memorias se realiza con 7 multiplexores 2 a 1, 6 de 32 bits y 1 de 8 bits, los cuales se codifican en VHDL con la sentencia algorítmica *case-when*. El selector de pixeles se realiza como una estructura tipo *barrel-shifter* al nivel de bytes, programado en VHDL, también con sentencias *case-when*. Los datos de salida son registrados, por lo que el componente opera con una latencia de 1 ciclo de reloj.

La tabla 5.25 muestra la descripción de las señales de E/S y la tabla 5.26 las estadísticas post-P&R del selector, donde se observa un componente con un consumo significativo de los recursos del dispositivo FPGA y la indefinición de la frecuencia de

operación, debido a su realización combinacional. En la síntesis y realización del sistema completo se vuelve a analizar su comportamiento, y en el caso de que no cumpla las características deseadas, optar por modificar su codificación o segmentar su operación.

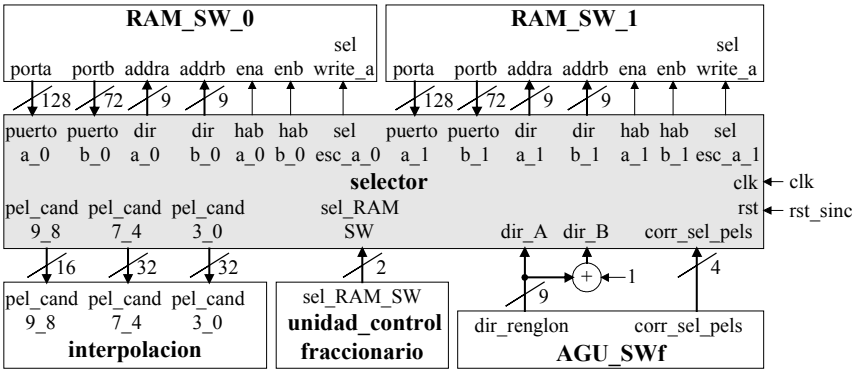


Figura 5.50. Componentes del estimador FME que intervienen en el proceso de selección de memorias y pixeles.

Tabla 5.25. Descripción de las señales de E/S del componente de selección de memorias y pixeles.

Señal	Tipo	Descripción
clk	E	Reloj, con operación en su borde de subida
rst	E	Reset síncrono, activo en bajo
sel_RAM_SW(1 downto 0)	E	Selección de la RAM_{SW} (00-#0, 01-#1, 1x-ninguna)
dir_a(8 downto 0)	E	Dirección del renglón de pixeles en el puerto A de la RAM_{SW}
dir_b(8 downto 0)	E	Dirección del renglón de pixeles en el puerto B de la RAM_{SW}
corr_sel_pels(3 downto 0)	E	Desplazamiento de la ventana del selector de pixeles
puerto_a_0(127 downto 0)	E	Puerto de datos A de los 4 bloques BRAM de la RAM_{SW} #0
puerto_b_0(71 downto 0)	E	Puerto de datos B de los bloques BRAM 0 y 1 y del primer byte del bloque 2 de la RAM_{SW} #0
puerto_a_1(127 downto 0)	E	Puerto de datos A de los 4 bloques BRAM de la RAM_{SW} #1
puerto_b_1(71 downto 0)	E	Puerto de datos B de los bloques BRAM 0 y 1 y del primer byte del bloque 2 de la RAM_{SW} #1
sel_esc_a_0	S	Selección de la configuración del puerto A de la RAM_{SW} #0 (0-puerto A lectura/escritura, conectado al procesador, 1-puerto A de solo lectura, conectado a los estimadores)
sel_esc_a_1	S	Selección de la configuración del puerto A de la RAM_{SW} #1 (0-puerto A lectura/escritura, conectado al procesador, 1-puerto A de solo lectura, conectado a los estimadores)
hab_a_0	S	Habilitación de la lectura del Puerto A RAM_{SW} #0
hab_b_0	S	Habilitación de la lectura del Puerto B RAM_{SW} #0
hab_a_1	S	Habilitación de la lectura del Puerto A RAM_{SW} #1
hab_b_1	S	Habilitación de la lectura del Puerto B RAM_{SW} #1
dir_a_0(8 downto 0)	S	Dirección del puerto A de la RAM_{SW} #0
dir_b_0(8 downto 0)	S	Dirección del puerto B de la RAM_{SW} #0
dir_a_1(8 downto 0)	S	Dirección del puerto A de la RAM_{SW} #1
dir_b_1(8 downto 0)	S	Dirección del puerto B de la RAM_{SW} #1
pel_cand_3_0(31 downto 0)	S	Pixeles 0 a 3 del renglón del bloque a procesar
pel_cand_7_4(31 downto 0)	S	Pixeles 4 a 7 del renglón del bloque a procesar
pel_cand_9_8(31 downto 0)	S	Pixeles 8 y 9 del renglón del bloque a procesar

Tabla 5.26. Estadísticas de síntesis y realización post-P&R del componente de selección de memorias y pixeles.

Recurso/especificación	Valor
Slices	393
Flip-flops	80
4-input LUTs	650
Puertas equivalentes	5476
Frecuencia (MHz)	-

5.3.2.8 Estadísticas del sistema FME completo.

El siguiente paso en la realización del acelerador FME es la instanciación de sus componentes en un módulo superior, donde se aplica la ejecución iterativa de los procesos de síntesis y realización del software ISE 8.2i y la depuración de la codificación del diseño, buscando obtener el mejor comportamiento área/velocidad.

La tabla 5.27 muestra las estadísticas post-P&R para varias configuraciones del ISE 8.2i, partiendo de la que utiliza las propiedades por defecto de la síntesis y realización sin restricciones de tiempo, hasta en la que se logran los mejores resultados. Se consideran restricciones globales de tiempo *period*, *pad to setup* y *clock to pad* de 10 ns, con un ciclo de trabajo del 50% en la señal de reloj. El estudio excluye los elementos externos al acelerador (procesadores, buses y memorias). Las primeras 5 iteraciones muestran un tiempo *clock to pad* excesivo. El análisis de tiempo estático post-P&R determinó un retardo significativo en la ruta de datos de la señal "corr_IME" (salida de la unidad de control), al no estar registrada. Las estadísticas de la iteración 6 muestran los resultados con la señal ya registrada, los cuales son los que marcan el mejor comportamiento en tiempo de la realización hardware.

Tabla 5.27. Estadísticas de síntesis y realización post-P&R del acelerador hardware FME.

	Iteración					
Recurso/especificación	1	2	3	4	5	6
Slices	7412	7412	7375	7403	7616	7678
Flip-flop	5581	5581	5641	6333	5548	6088
4-input LUTs	12253	12253	12022	12125	12084	11802
DSP48s	18	18	18	18	18	18
Puertas equivalentes	745047	745047	743910	751964	765333	751666
Periodo (ns)	10.6	10.384	10.166	9.97	11.876	10.119
Offset = in (ns) before clk	-	7.628	8.991	7.506	7.192	8.238
Offset = out (ns) after clk	-	14.55	15.011	13.733	13.669	10.004

Iteración #1: Propiedades de síntesis y realización por defecto, sin restricciones de tiempo.
Iteración #2: Propiedades de síntesis y realización por defecto, con restricciones de tiempo.
Iteración #3: Síntesis con esfuerzo de optimización alto, sin balance de registros, realización por defecto.
Iteración #4: Síntesis con esfuerzo de optimización alto, con balance de registros, realización por defecto.
Iteración #5: Síntesis por defecto, realización con esfuerzo de mapeo alto, optimización lógica y global, retiming y nivel de P&R alto.
Iteración #6: Síntesis con esfuerzo de optimización alto, con balance de registros, realización con esfuerzo de mapeo alto, optimización lógica y global, retiming y nivel de P&R alto.

La realización del acelerador también se puede avaluar en función de los efectos en el tiempo de procesamiento de la estimación fraccionaria, de las prácticas y técnicas de diseño de los dispositivos FPGA.

En la tabla 5.10 se documentó la duración en ciclos de reloj del procesamiento de cada grupo de bloques del mismo tamaño. Estos son resultados de diseño, sin considerar la tecnología de realización, por lo que la latencia y los ciclos extras son causados por la operación normal de cada uno de los elementos de procesamiento, direccionamiento y almacenamiento.

Los métodos de diseño sobre FPGA consideran el uso intensivo de la técnica *pipeline,* para alcanzar los objetivos de comportamiento en velocidad. Su aplicación limita el número de niveles lógicos y reduce el retardo en la ruta de datos, al coste de un mayor consumo de recursos y el incremento de la latencia. En la realización del acelerador FME sobre el dispositivo Virtex-4, este efecto puede observarse en el incremento de la latencia en cada uno de sus componentes (Tabla 5.28), lo cual incide directamente en la duración del procesamiento fraccionario (Tabla 5.29), debido a que la heterogeneidad en el tamaño de los bloques, en la resolución de las muestras y en la diversidad de la integración vertical de las descomposiciones 4×4, producen múltiples inicios de operación que concatenan la latencia. Un caso concreto es en el cambio del ciclo *half* al *quarter*. Con una mayor latencia, el ciclo *half* tarda más en retroalimentar al interpolador el número del mejor bloque, alrededor del cual inicia el procesamiento *quarter*.

Tabla 5.28. Latencia de cada componente del acelerador FME, en el diseño original y en la realización sobre FPGA.

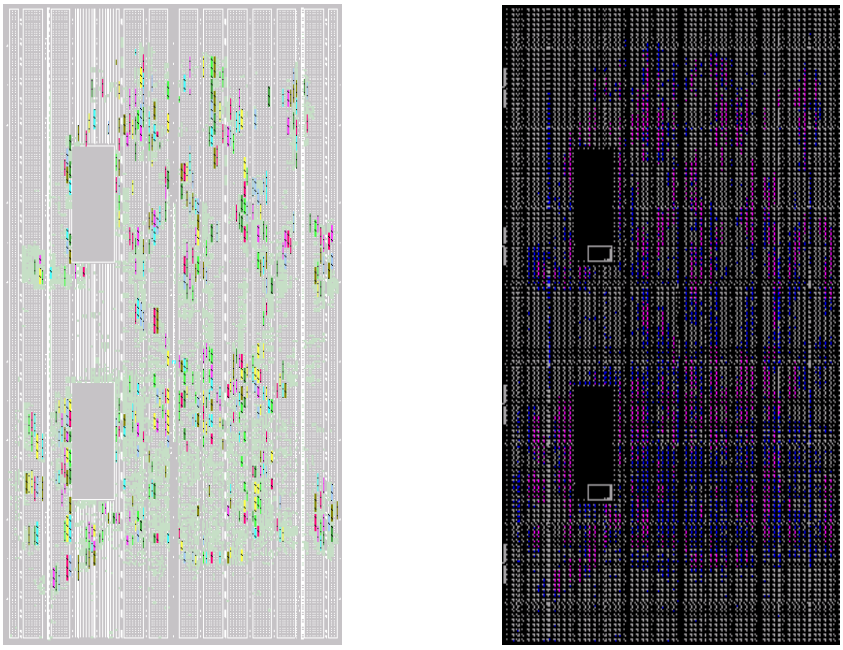
Componente	Latencia del diseño original (ciclos)	Latencia de la realización sobre FPGA (ciclos)
Control	0	1
Interpolación half	7	9
Interpolación half + quarter	8	10
SATD 4×4	4	6
coste R/D	0	2
Comparación R/D	0	2
AGU_{MBAf}	0	1
AGU_{SWf}	0	1
Selector	0	1

Tabla 5.29. Duración en ciclos de reloj del procesamiento fraccionario de los 7 tamaños de bloque, antes y después de la realización sobre FPGA.

	Número y tamaño de bloques	Ciclos de procesamiento, operación independiente de la tecnología	Ciclos de procesamiento, realización sobre FPGA
	16 – 4×4	401	419
	8 – 4×8	277	386
	8 – 8×4	386	537
	4 – 8×8	253	381
	2 – 8×16	193	306
	2 – 16×8	241	382
	1 – 16×16	196	302
Total	**41**	**1947**	**2713**

El incremento de un 39% en los ciclos de procesamiento de la arquitectura FME al desarrollarse sobre FPGA, motiva la continuación del estudio de su estructura y secuencia de operación, para buscar un comportamiento que satisfaga eficientemente los requerimientos del estándar H.264/AVC, lo cual se realiza en la siguiente sección de este capítulo.

La figura 5.51 muestra el emplazamiento y rutado de la arquitectura FME sobre el dispositivo XC4VFX60, en donde se observan los resultados gráficos de P&R del software ISE 8.2i. Cada componente y el sistema completo se simuló y depuró con la herramienta ModelSim XE III/Starter 6.1e, utilizando bancos de pruebas para comprobar la secuencia de operaciones ante todas las posibilidades de las señales de entrada (Figura 5.52). Al no considerarse los buses y procesadores, el programa del banco de pruebas incluyó las señales de interfase al bus para la operación del registro esclavo de supervisión y los pixeles de entrada se definieron como valores iniciales en las memorias RAM_{MBA} y RAM_{SW}.



(a) (b)

Figura 5.51. Emplazamiento (a) y rutado (b) del acelerador FME sobre la FPGA XC4VFX60.

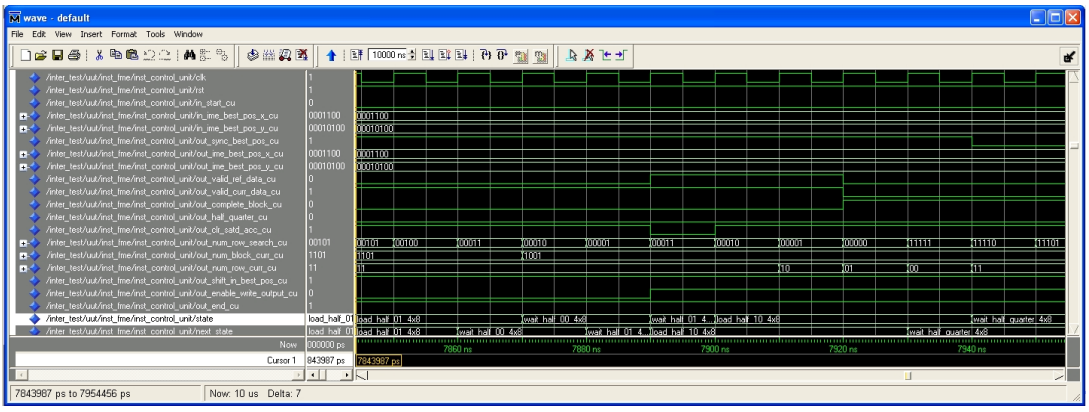


Figura 5.52. Señales de simulación de la unidad de control FME.

5.4 Arquitecturas FME-FSBM con enfoque hacia velocidad.

En la sección anterior se documentó el diseño de un acelerador FME-FSBM con altos niveles de eficiencia y utilización, que se ubica dentro del rango de velocidad de procesamiento de arquitecturas previas con niveles similares de área. Si bien el resultado obtenido es de interés desde el punto de vista del equilibrio área/velocidad, la duración en ciclos de reloj de la estimación fraccionaria puede ser muy alta para satisfacer las necesidades de codificación de algunas aplicaciones. Si a esto se le agrega el incremento significativo del número de ciclos cuando la arquitectura se realiza sobre FPGA, se llega a la conclusión de que se necesita una mayor optimización, para cumplir satisfactoriamente los requerimientos actuales. Es por ello que a continuación se presentan tres propuestas para reducir la duración del procesamiento fraccionario, la primera formulada a partir del análisis de la secuencia de operación del acelerador FME y las siguientes en función del aumento de los niveles de reutilización y paralelización.

5.4.1 Flujo de procesamiento en pares de bloques.

Dos de las características estructurales y de operación que se deben buscar en los nuevos diseños FME y que se determinaron con el estudio de las arquitecturas del estado del arte, es la ejecución de la secuencia de procesamiento con cero ciclos falsos y la alimentación continua de datos a la unidad de interpolación. Bajo esta perspectiva, en un primer análisis de la operación del acelerador propuesto, se detectó una discontinuidad en la secuencia de interpolación y distorsión de cada bloque candidato, entre el fin del ciclo *half* y el inicio del *quarter* (Figura 5.53). La pausa en el procesamiento se debe al retardo en la retroalimentación del número del mejor bloque *half* para iniciar el ciclo *quarter*, lo que agrega 4 ciclos extras por cada bloque procesado.

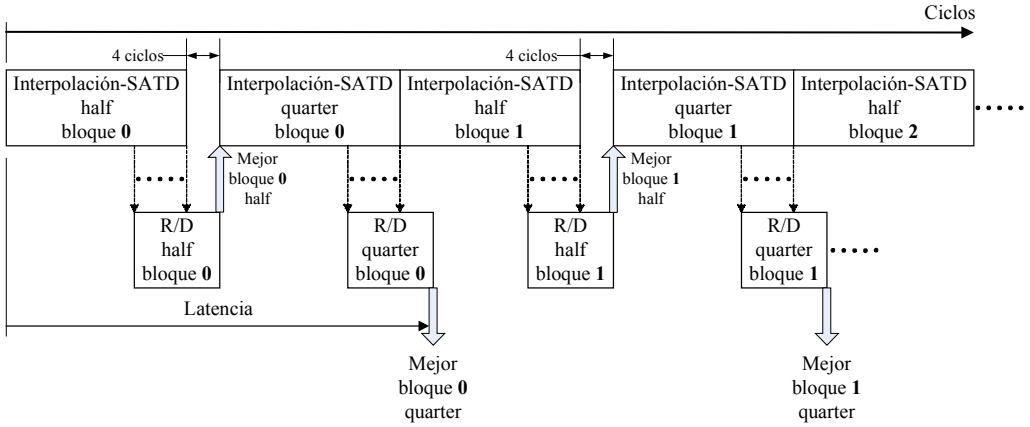


Figura 5.53. Secuencia original del procesamiento fraccionario de los bloques candidatos.

Aun cuando la solución a esta discontinuidad no es muy significativa, desde el punto de vista de la reducción en la duración de la estimación fraccionaria, ya que solo resta un máximo de 4×41 = 164 ciclos al total de ciclos de procesamiento, su supresión puede optimizar el diseño previo, sin incrementar el coste del área.

La eliminación de la discontinuidad se realiza en la unidad de control, alterando la secuencia de estimación de los bloques candidatos. Ahora el procesamiento se efectúa

como una sucesión de pares de bloques del mismo tamaño para la misma resolución (dos *half*, dos *quarter*, dos *half*,...), lo que separa la ejecución de los componentes de coste y comparación R/D, del flujo de interpolación y distorsión SATD (Figura 5.54). Con esta solución el número del mejor bloque *half* esta disponible antes de que inicie el procesamiento *quarter*, por lo que no se necesita un tiempo muerto entre ambos ciclos.

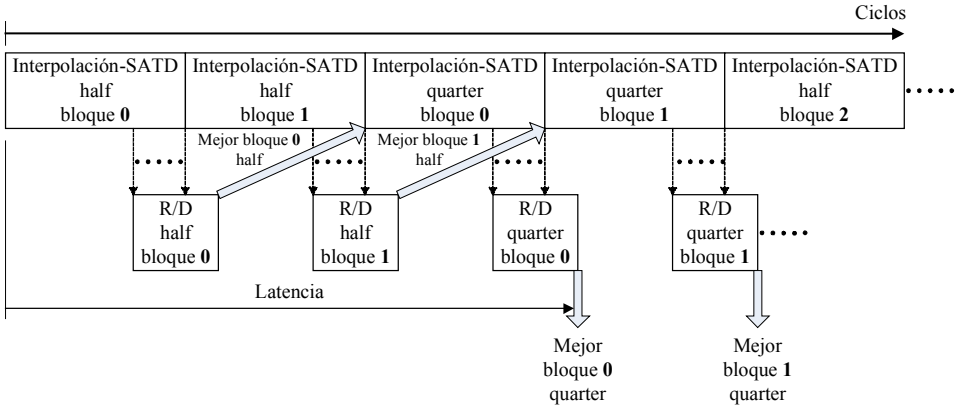


Figura 5.54. Optimización del acelerador FME a partir de una secuencia de procesamiento en pares de bloques.

La solución modifica la máquina de estados de la unidad de control, sin que esto incremente el área del acelerador. El único coste es el aumento en 7 ciclos de la latencia para la obtención del primer resultado *quarter*. La tabla 5.30 muestra la duración en ciclos de reloj del procesamiento para cada tamaño de bloque, con la propuesta en pares de bloques y operación independiente de la tecnología. Se observa que el total de ciclos de reloj es aún muy grande, por lo que se deben buscar acciones de optimización más drásticas.

Tabla 5.30. Duración en ciclos de reloj del procesamiento fraccionario de los 7 tamaños de bloque, antes y después de la secuencia en pares de bloques, operación independiente de la tecnología.

	Número y tamaño de bloques	Ciclos de procesamiento	
		Secuencia original	Secuencia en pares de bloques
	16 – 4×4	401	337
	8 – 4×8	277	245
	8 – 8×4	386	354
	4 – 8×8	253	237
	2 – 8×16	193	185
	2 – 16×8	241	233
	1 – 16×16	196	196
Total	**41**	**1947**	**1787**

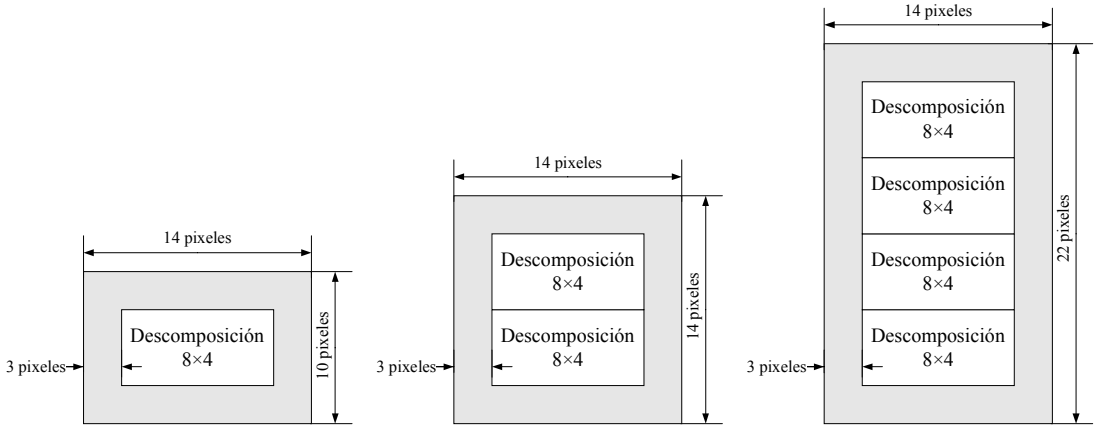
5.4.2 Procesamiento con descomposiciones 8×4.

Las técnicas de descomposición e integración vertical 4×4 contribuyen a la obtención de un acelerador FME con un alto equilibrio área-velocidad, pero sus procedimientos no se pueden aplicar a los tamaños de bloques 4×4 y 8×4 y limitan el nivel de reutilización de datos en la interpolación de los bloques 8×8, 8×16, 16×8 y 16×16.

Para mejorar las características de la arquitectura e incrementar la reutilización de datos, se propone aplicar las técnicas citadas utilizando descomposiciones 8×4, lo que reduce 33.65% los accesos a memoria y 20.19% el número de pixeles a procesar con respecto a las descomposiciones 4×4 (Tabla 5.31). La figura 5.55 muestra la técnica de integración vertical para descomposiciones 8×4, donde se observa una reducción del 30% en los pixeles a procesar en todos los casos de apilamiento de bloques. El precio es la sub-utilización de la sección de procesamiento para los tamaños 4×4 y 4×8 y el aumento del área de la arquitectura.

Tabla 5.31. Número total de accesos a memoria RAM_{SW} y pixeles a procesar en la aplicación de las estrategias de descomposición e integración vertical 4×4 y 8×4, para la estimación de movimiento *half* y *quarter*.

	Descomposición		% de
	4×4	8×4	reducción
Accesos a memoria	1664	1104	33.65
Pixeles a procesar	8320	6640	20.19



# descomposiciones verticales	Pixeles a procesar con integración 4×4	Pixeles a procesar con integración 8×4	% de reducción
1	200	140	30
2	280	196	30
4	440	308	30

Figura. 5.55. Integración vertical de las descomposiciones 8×4 y porcentaje de reducción del número de pixeles procesados en la unidad de interpolación, con respecto a la integración con descomposiciones 4×4.

La arquitectura FME para descomposiciones 8×4 se deriva de la estructura original (Figura 5.17), en la que se modifican el componente de selección de pixeles y las unidades de interpolación, distorsión SATD y control. El selector recibe 29 pixeles candidatos y selecciona los 14 que procesa la unidad de interpolación en cada ciclo de reloj (Figura 5.56). El interpolador genera los sub-pixeles *half* y *quarter* utilizando 28 filtros FIR de 6-taps y 64 filtros bilineales, además de los correspondientes registros del *buffer* de interpolación (Figura 5.57). El módulo SATD 8×4 calcula la distorsión por medio de la paralelización de dos componentes SATD 4×4 (Figura 5.58). La unidad de control reduce su número de estados en las secuencias de bloques con un ancho mayor a 4 pixeles.

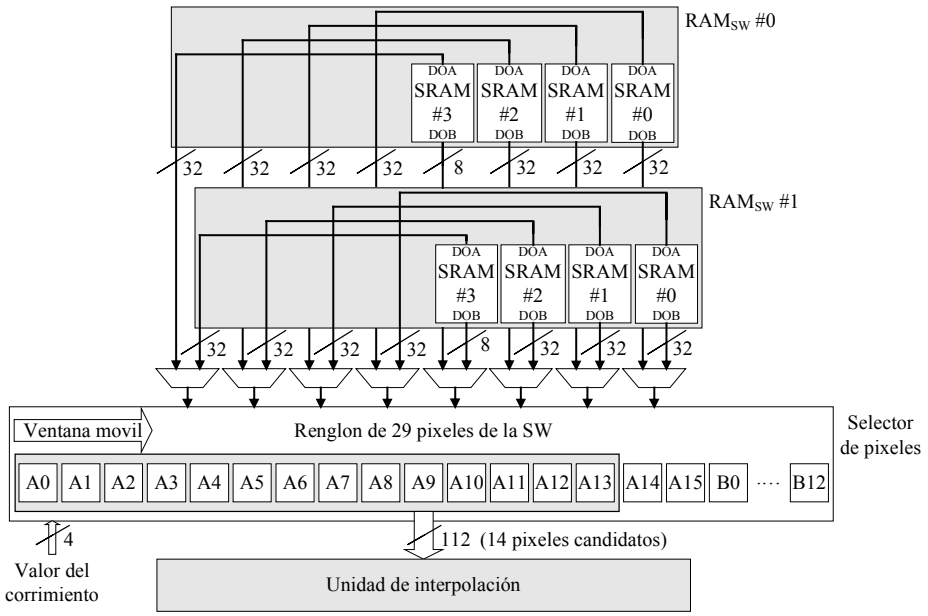


Figura 5.56. Esquema del selector de pixeles para la arquitectura FME, descomposiciones 8×4.

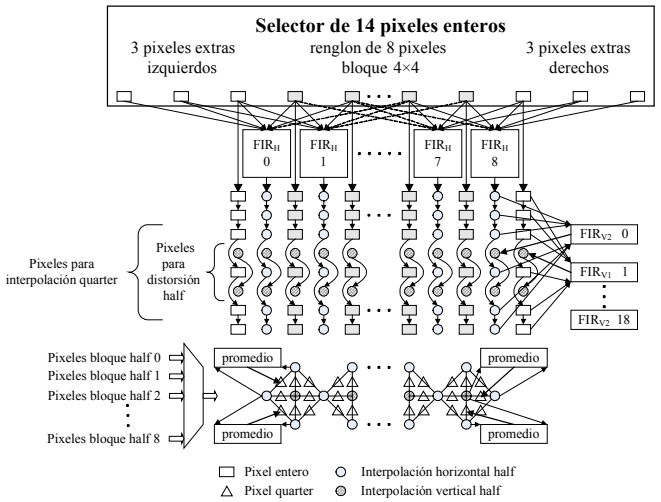


Figura 5.57. Estructura de la unidad de interpolación para el acelerador FME con descomposiciones 8×4.

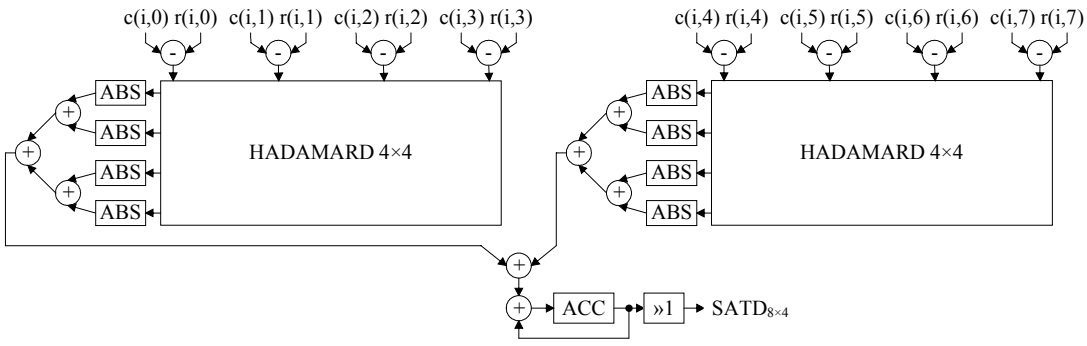


Figura 5.58. Diagrama del módulo SATD para el diseño FME con descomposiciones 8×4.

La aplicación del segundo método de optimización en una operación independiente de la tecnología de la arquitectura FME con enfoque en velocidad, mejora nuevamente la velocidad de procesamiento, al reducir 34 % la duración de la estimación fraccionaria (Tabla 5.32), con un aumento del 62 % en el coste del área, con respecto al acelerador de flujo de procesamiento en pares de bloques.

Tabla 5.32. Duración en ciclos de reloj del procesamiento fraccionario de los 7 tamaños de bloque, secuencia en pares de bloques, descomposiciones 4×4 y 8×4, operación independiente de la tecnología.

		Ciclos de procesamiento	
	Número y tamaño de bloques	Descomposición 4×4	Descomposición 8×4
	16 – 4×4	337	337
	8 – 4×8	245	245
	8 – 8×4	354	167
	4 – 8×8	237	123
	2 – 8×16	185	93
	2 – 16×8	233	117
	1 – 16×16	196	98
Total	**41**	**1787**	**1180**

5.4.3 Arquitectura con doble secuencia de procesamiento.

El flujo de ejecución en pares de bloques y la aplicación de las técnicas de descomposición e integración vertical 8×4, reducen 39.4% la duración de procesamiento de la arquitectura FME original en su realización sobre FPGA, al pasar de 1947 a 1180 ciclos de reloj. Si bien la optimización en velocidad es apreciable, puede ser insuficiente para aplicaciones de alto nivel, por lo que a continuación se presenta una versión altamente paralela del estimador, que mejora aun más las estadísticas de velocidad del acelerador hardware. La propuesta se soporta en las siguientes observaciones:

* Las arquitecturas FME típicas calculan el coste R/D de los bloques fraccionarios alrededor de los 41 mejores bloques enteros, en una secuencia bloque por bloque, lo que acumula la duración en ciclos de cada una de las operaciones.

* Las 41 posiciones dentro la SW de los bloques enteros de menor coste, son función del movimiento aleatorio de la imagen entre el cuadro actual y los cuadros de referencia, por lo que no existe correlación en la estimación fraccionaria de los bloques candidatos enteros.

* Los bloques 4×4 y 4×8 presentan la mayor duración en ciclos de la estimación con descomposiciones 8×4 (Tabla 5.32), además de que estos tamaños de bloque sub-utilizan las unidades de interpolación y distorsión SATD 8×4.

Bajo estas consideraciones, se propone realizar el procesamiento FME en dos secuencias, una para los bloques 4×4 y 4×8 y la otra para los bloques restantes (Figura 5.59). Los bloques 4×4 y 4×8 se procesan en la arquitectura original de una manera directa, sin necesidad de descomposición y con un procesamiento de integración vertical que ya está implícito en la operación continua de las unidades de interpolación y distorsión SATD 4×4. Los bloques 8×4, 8×8, 8×16, 16×8 y 16×16, se procesan en una

arquitectura ampliada, que utiliza una unidad de interpolación y módulos SATD para descomposiciones 8×4.

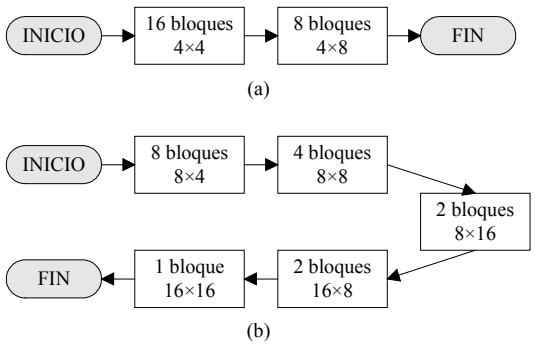


Figura 5.59. Procesamiento FME en dos secuencias, a) descomposición 4×4 y b) descomposición 8×4.

La arquitectura que realiza esta doble secuencia de procesamiento duplica las unidades de interpolación y los módulos de distorsión SATD e incluye dos *buffers* para respaldar los pixeles de los bloques actuales y candidatos (Figura 5.60), de tal forma que la lectura a las RAM_{MBA} y RAM_{SW} se ejecuta para las dos descomposiciones, lo que evita cuellos de botella en el acceso a memorias y permite la preparación de los datos en el orden que los necesitan las unidades de interpolación.

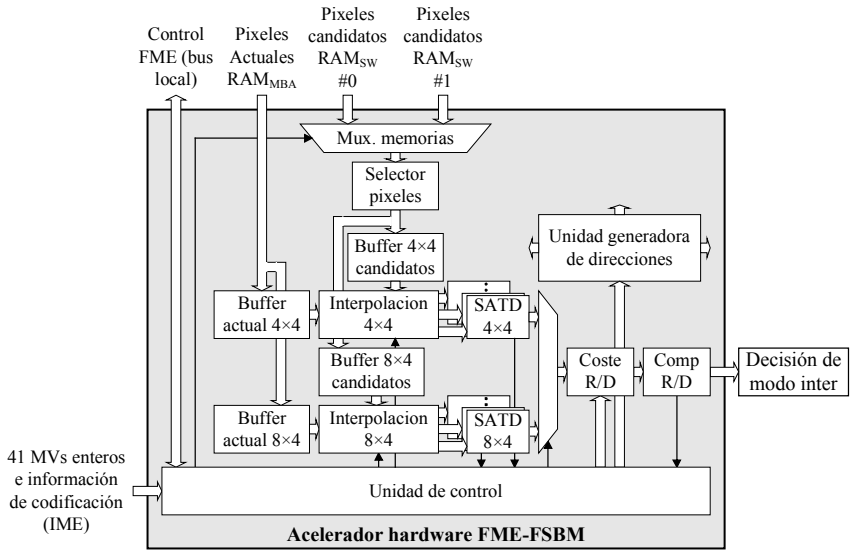


Figura 5.60. Arquitectura del acelerador hardware FME-FSBM, para las secuencias de procesamiento 4×4 y 8×4.

Los *buffers* se realizan con bloques BRAM de dos puertos, en forma similar al diseño de las memorias RAM. La distorsión de las descomposiciones 8×4 se acumula en los módulos SATD, en vez de sumarse en la unidad de coste R/D. La unidad de control sincroniza eficientemente la sección de procesamiento para evitar ciclos extras, al compartirse las unidades de coste y comparación R/D. De acuerdo a la tabla 5.32, la secuencia 4×4 tiene una duración de 337 + 245 = 582 ciclos y la secuencia 8×4 de 167 + 123 + 93 + 117 + 98 = 598 ciclos, por lo que la ruta critica y la duración del procesamiento fraccionario la define esta última. En suma, los ciclos totales de

procesamiento de la arquitectura de doble secuencia, con respecto a la primera propuesta, se reducen 69% (de 1947 a 598) y el área se incrementa 150%, por lo esta estructura representa una opción valida para la realización del algoritmo FME-FSBM sobre FPGA.

5.5 Integración de las arquitecturas IME y FME.

Una vez descrito el diseño y realización de las arquitecturas FME-FSBM, en esta sección se analiza su integración con las arquitecturas IME-FSBM propuestas. A manera de ejemplo, se utilizan las arquitecturas IME con forma de SW variable y la FME con enfoque hacia velocidad, con doble secuencia de procesamiento. Ambas arquitecturas fueron diseñadas para trabajar en forma cooperativa, con el objetivo de optimizar la velocidad de estimación de movimiento, a partir de una operación segmentada-paralela, por lo que su interconexión es directa y natural, con mínimos elementos externos (Figura 5.61).

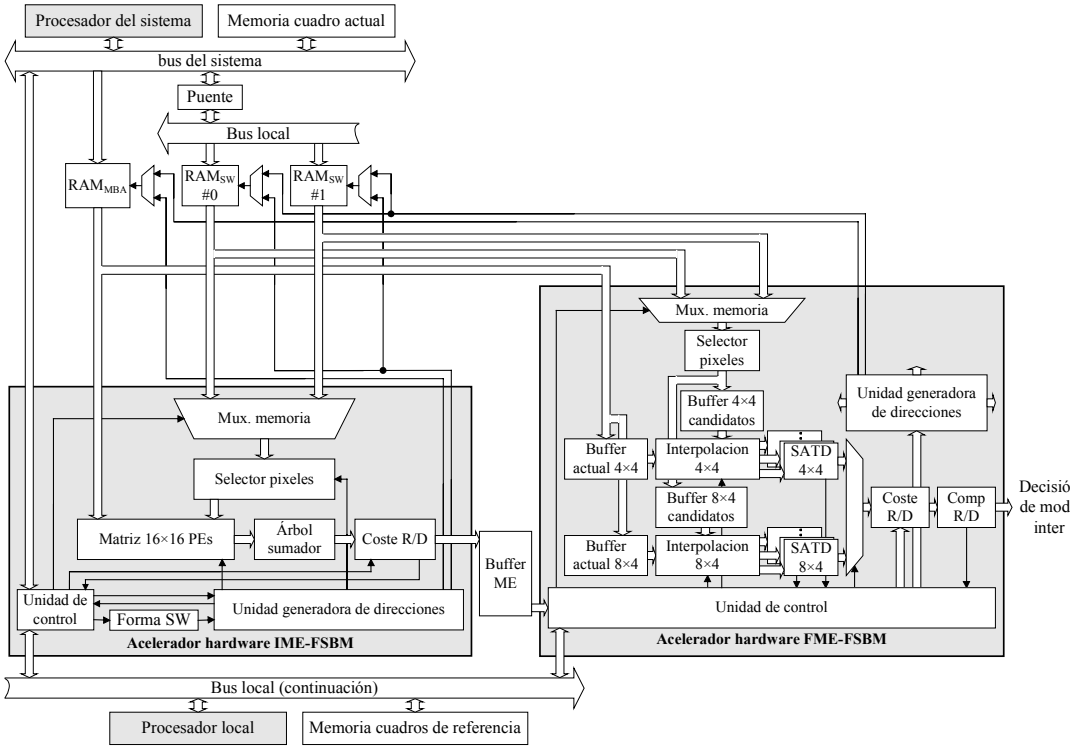


Figura 5.61. Arquitectura completa del sistema de estimación de movimiento propuesto, con resolución de pixeles enteros (IME-FSBM) y fraccionarios (FME-FSBM).

5.5.1 Operación global.

Las arquitecturas fueron desarrolladas como aceleradores hardware que funcionan como periféricos de dos procesadores, el del sistema para todo el codificador y el local para la etapa de estimación de movimiento. El módulo de estimación cuenta con tres memorias locales (RAM_{MBA}, RAM_{SW} #0 y RAM_{SW} #1), compartidas y direccionadas por ambos aceleradores. La memoria RAM_{MBA} almacena 2 MBs actuales y cada memoria RAM_{SW} tiene capacidad para almacenar los pixeles candidatos de una SW. Los pixeles de los cuadros actual y de referencia se ubican en memorias externas,

por lo que los procesadores son los encargados de realizar la transferencia de datos a las memorias locales.

El ciclo de estimación inicia con la transferencia de los datos y parámetros que necesita el estimador completo. El procesador del sistema escribe en la memoria RAM_{MBA} los pixeles del MB actual y en los registros de control del IME los siguientes parámetros de ejecución del proceso de ajuste de bloques:

* Los desplazamientos de exploración horizontal y vertical de la SW.

* La forma de la SW.

* La posición en RAM local del MB actual (posición #0 ó #1).

* La información para el cálculo del índice tasa/distorsión.

Al mismo tiempo, con la información que previamente recibió del procesador del sistema, el procesador local controla la escritura de los pixeles de la SW del MB actual en la RAM_{SW} #0. Para evitar cuellos de botella en ciclos consecutivos, se tiene la capacidad de almacenar otro MB actual en la RAM_{MBA} y otra SW en la RAM_{SW} #1, para que mientras el estimador utiliza los primeros datos, los procesadores almacenen los pixeles del siguiente MB actual y su SW.

A continuación, el procesador local da la orden de arranque de la estimación, indicando el número de memoria RAM_{SW} donde se encuentran los pixeles de la SW para el MB actual. El módulo IME lee los pixeles de referencia de la memoria RAM_{SW} y los pixeles del MB actual de la memoria RAM_{MBA} e inicia su proceso de estimación entera.

Al final del ciclo IME, su unidad de control interrumpe al procesador local para indicar su disponibilidad de iniciar otro ciclo. Simultáneamente, envia al *buffer* ME la información que necesita la etapa FME para realizar la estimación fraccionaria de cada uno de los 41 bloques y sub-bloques con resolución entera resultantes:

* Los pares de coordenadas (x,y) de los 41 bloques enteros de menor coste, con respecto al origen de la SW.

* El número de memoria RAM_{SW} valida para el procesamiento fraccionario (#0 ó #1).

* La posición en RAM_{MBA} del MB actual (posición #0 ó #1).

* El desplazamiento de búsqueda Ph y los parámetros de coste R/D.

El número de RAM_{SW} define la memoria que usó el acelerador IME en su último ciclo de operación y que contiene los bloques candidatos de la SW para los que son validas las 41coordenadas de los mejores bloques enteros. Se recordara que se tienen 2 RAM_{SW} compartidas por los aceleradores IME y FME, y que mientras el primero lee una de ellas, la otra se actualiza o es leída por el otro módulo, en una alternancia controlada por el procesador local, de tal forma que se maximice el nivel de ocupación del sistema de estimación.

La posición en RAM_{MBA} del MB actual apunta hacia el MB que ya se estimó en forma entera y que ahora se procesara en estimación fraccionaria. Mientras tanto, la otra posición se actualiza con el nuevo MB actual, para su posterior lectura y carga en la matriz de PEs del estimador IME.

Los parámetros de desplazamiento y cálculo del coste R/D son los mismos que se utilizaron en la etapa IME y que se recibieron para este ciclo de estimación desde el procesador del sistema. La idea de que el acelerador IME transfiera directamente estos parámetros, es para asegurar que el ciclo FME se ejecute bajo las mismas condiciones de operación.

El ciclo FME inicia con la recepción de la señal de visto bueno del procesador local a través de un registro de control y la indicación del acelerador IME de la validez de la información en su *buffer* de salida. Al detectar estas señales, el bloque FME lee la información del *buffer* ME y empieza su operación, la cual tiene una duración constante.

Al final de la estimación fraccionaria, el acelerador FME interrumpe al procesador local para señalar la validez de sus resultados y su disponibilidad de iniciar otro ciclo. Los resultados son los costes R/D y coordenadas con respecto al origen de la SW, en ¼ de píxel de resolución, de los mejores 41 bloques *quarter*, información que se respalda en un *buffer* para su transferencia a la etapa de modo de decisión inter, donde se determina la combinación de particiones y sub-particiones que mejor estima el movimiento de los pixeles del MB actual.

5.5.2 Secuencia de datos, parámetros y operaciones.

Para observar la operación segmentada-paralela del sistema, a continuación se define gráficamente la secuencia de datos, parámetros y operaciones indicadas en la sección anterior. Como referencia para el análisis, la tabla 5.33 muestra las velocidades de procesamiento de cada una de las arquitecturas propuestas, donde se observa que la velocidad de procesamiento de los diseños IME es función de los desplazamientos de exploración Ph y Pv, mientras que la velocidad de los diseños FME es constante.

Tabla 5.33. Velocidad de procesamiento en ciclos/MB de las arquitecturas propuestas, en función de los desplazamientos de exploración horizontal (Ph) y vertical (Pv). Operación independiente de la tecnología.

	Arquitectura	Velocidad de procesamiento (Ciclos/MB)
IME	forma SW:	
	rectangular/cuadrada	$(2Ph+1)(2Pv+1)+N-1$
	circular	$3.1Ph^2+N-1$
	rombica	$(2Ph+1)^2/2+N-1$
	eliptica	$1.55(Ph)^2+N-1$
	cruz	$(Ph/4+1)(15/4\ Ph+1)+N-1$
FME	versión:	
	original	1947
	pares de bloques	1787
	descomposición 8×4	1180
	descomposición 4×4 y 8×4	598

Debido a la diversidad de velocidades de las arquitecturas IME, se consideran dos posibilidades para la secuencia de operaciones: primera, de que el tiempo del ciclo IME sea mayor al tiempo del ciclo FME (Figura 5.62) y segunda, el caso contrario (Figura 5.63). En ambas situaciones, los procesamientos IME y FME inician cuando el procesador local confirma que se cuenta con todos los datos, parámetros y resultados necesarios para su operación.

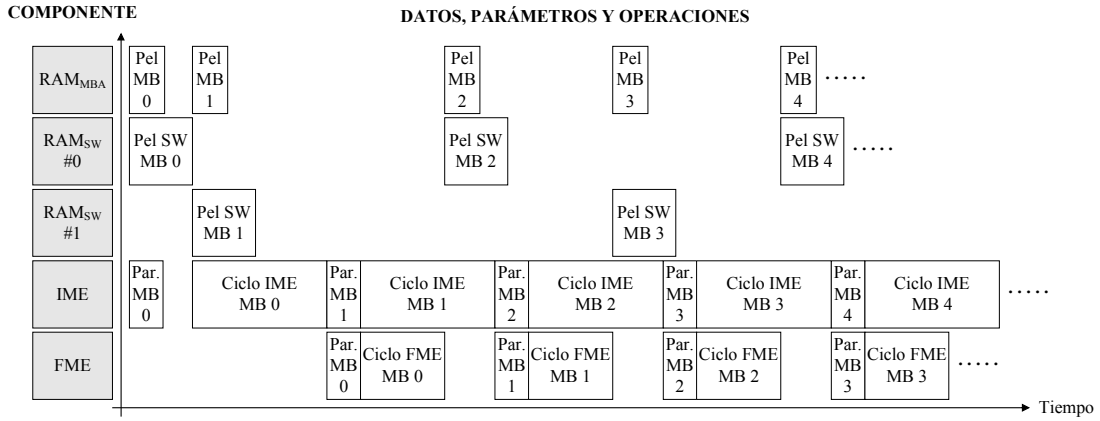


Figura 5.62. Secuencia de datos, parámetros y operaciones en los componentes del sistema de estimación de movimiento propuesto. Tiempo ciclo IME > tiempo ciclo FME.

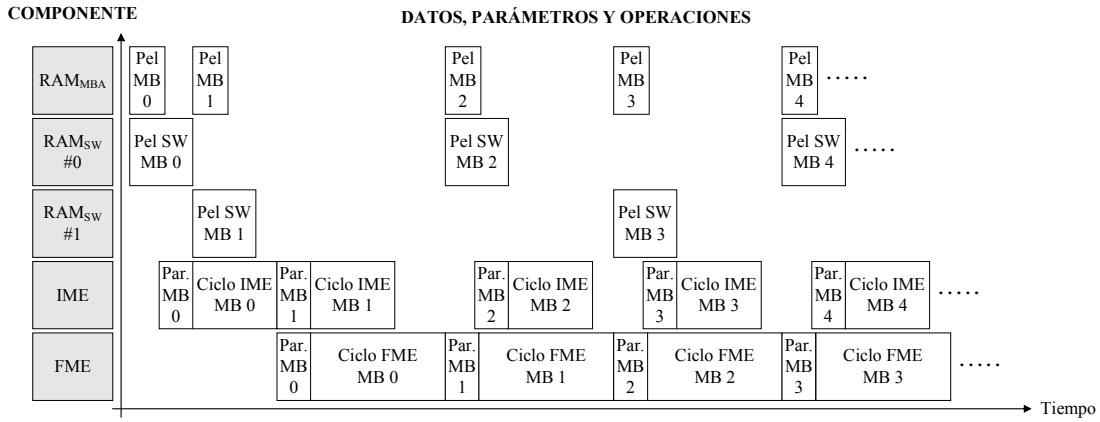


Figura 5.63. Secuencia de datos, parámetros y operaciones en los componentes del sistema de estimación de movimiento propuesto. Tiempo ciclo IME < tiempo ciclo FME.

En las figuras se muestra que el módulo IME inicia un ciclo cuando recibe los parámetros de operación por parte del procesador del sistema, considerando que previa o simultáneamente los pixeles del MB actual y su SW fueron cargados en las memorias indicadas. El inicio de un ciclo FME depende del termino de su operación anterior y del fin del ciclo IME del MB actual a procesar. Es en este momento cuando recibe los resultados de la estimación entera y los parámetros utilizados en su obtención. Los pixeles del MB actual y su SW ya se encuentran en las memorias locales indicadas, al ser los mismos que utilizó el acelerador IME.

El respaldo de los pixeles de 2 MBs y de 2 SW en las memorias locales permite que en todo momento se tengan los datos para la operación simultánea de los módulos IME y FME. Solo se debe tener cuidado con la actualización de las memorias, ya que

esta función se debe sincronizar con el fin del ciclo de estimación fraccionaria, al ser el acelerador FME el que libera el contenido de las mismas.

En conclusión, la integración de los módulos IME y FME forma un sistema con operación segmentada-paralela, donde básicamente, el tiempo crítico lo define el ciclo de estimación de mayor duración.

5.6 Conclusiones.

El capítulo finaliza con un resumen de las temáticas más importante del diseño, realización y optimización del acelerador hardware FME-FSBM sobre FPGA. Se hace referencia a las ideas, conceptos, procedimientos y técnicas que se observaron en las arquitecturas del estado del arte y la forma en que se aplicaron para obtener una versión inicial del estimador fraccionario y para definir las modificaciones en su estructura que incrementan su velocidad de procesamiento. Los resultados obtenidos se evalúan con la comparación de las arquitecturas propuestas, sobre la base de las estadísticas área-velocidad.

5.6.1 Diseño FME-FSBM.

Se ha diseñado una arquitectura hardware realizada sobre FPGA, para la estimación de movimiento fraccionario con el algoritmo FSBM, que satisface los requerimientos del estándar H.264/AVC, con una estructura que incluye en su *data-path* un selector de memorias y pixeles, una unidad de interpolación, un grupo de módulos de distorsión SATD y los elementos de coste y comparación R/D y en su *control-path*, las unidades generadoras de direcciones y la unidad de control. Todos los componentes están diseñados con el objetivo de ejecutar la estimación fraccionaria en el menor tiempo posible y al mínimo coste en área.

Antes de iniciar el diseño de la arquitectura FME, se estudió el software de referencia [JM1106] y se analizaron las arquitecturas del estado del arte, para documentar las características generales sobre las cuales deben partir los nuevos diseños, y que se listan a continuación:

* Equilibrio área-velocidad, con la paralelización de la mayor cantidad de lazos *for* del algoritmo FME y la optimización del área y velocidad de todos los componentes.

* Alta velocidad de procesamiento, con un flujo de datos continuo, cero ciclos falsos y latencia mínima de los elementos.

* Ancho de banda y coste computacional mínimos, con una máxima reutilización de datos en la unidad de interpolación.

La arquitectura que se presenta busca cumplir estos requerimientos generales con las siguientes especificaciones:

* Sincronización global de la arquitectura FME.

* Alta regularidad en el acceso a los datos.

* Paralelización de la sección de procesamiento, con un bajo coste en área.

* Unidad de interpolación de alto rendimiento.

* Diseño eficiente del módulo de distorsión SATD.

La unidad de control, diseñada a partir de una FSM Moore de 151 estados y 3 contadores binarios, centraliza el control del acelerador FME, lo que permite sincronizar la operación de todos los componentes, para evitar tiempos muertos y maximizar el nivel de utilización de cada uno de ellos.

La lectura de las RAM_{SW} al nivel de palabras y el uso de un selector que elige los pixeles que procesa el interpolador en cada ciclo de reloj, garantizan la regularidad en el acceso a los datos candidatos, con una lógica eficiente de direccionamiento de memorias.

La sección de procesamiento, con una unidad de interpolación, 9 módulos SATD y los componentes de coste y comparación R/D, paraleliza la estimación de movimiento *half* y *quarter*, al procesar simultáneamente 9 bloques con elementos de bajo coste en área, gracias al uso de la técnica de descomposición de los bloques mayores en bloques 4×4.

La unidad de interpolación 4×4, desarrollada con filtros FIR de 6-taps horizontales y verticales, con realización en árbol sumador y filtros bilineales paralelos, procesa en forma continua los pixeles de entrada y reutiliza los datos de las descomposiciones 4×4, en los bloques donde se aplica la técnica de integración vertical.

El uso de 9 módulos SATD, diseñados con transformadas Hadamard 4×4 de arquitectura paralela renglón-columna, permite soportar el flujo continuo de los sub-pixeles que genera la unidad de interpolación, con una latencia de 4 ciclos no acumulable y una lógica de control básica.

5.6.2 Realización FME-FSBM.

La arquitectura FME-FSBM se codificó en VHDL para su realización física sobre la FPGA Virtex-4 XC4VFX60-10-FF1152 de Xilinx, utilizando las herramientas de software ModelSim Xilinx edition-III v6.1, ISE 8.2i y EDK 8.2i, para los procesos de simulación, síntesis y realización.

En la codificación VHDL se emplearon técnicas de diseño para reutilización y velocidad. En el diseño para la reutilización destacan las técnicas de diseño lógico síncrono, la no-utilización de candados, el registro de las salidas de las unidades de diseño, el uso de procesos en vez de la instanciación de componentes pequeños o primitivas, etc. En el diseño para velocidad, se enfatizan la segmentación del diseño, el balance y duplicación de registros, la limitación del tamaño de los componentes, el tipo de codificación de los estados de las máquinas FSM, la preferencia de código estructural al RTL, el manejo eficiente de estructuras *if-else* y *case*, el uso de señales de habilitación de reloj, etc. Algunas de estas técnicas se aplican con el apoyo de la herramienta ISE 8.2i, la cual las contempla para su ejecución automática, en las propiedades de síntesis y realización.

Con el propósito de evaluar la funcionalidad del acelerador FME, se simuló la operación de cada componente y del total del estimador, depurando su código hasta observar el comportamiento deseado. Las pruebas se efectuaron sobre la herramienta ModelSim, con la ejecución de bancos de prueba (*testbench*) en diferentes situaciones de operación.

La síntesis y realización de los componentes se realizó con los valores por defecto de las propiedades de síntesis y realización del ISE 8.2i, sin definir restricciones de tiempo. En el sistema completo se fijaron en 10 ns las restricciones globales *period*, *pad to setup* y *clock to pad*. Los mejores resultados se obtienen con las propiedades de síntesis con esfuerzo de optimización alto y balance de registros y las propiedades de realización con esfuerzo de mapeo alto, optimización lógica y global, *retiming* y nivel P&R alto.

Los resultados obtenidos en la aplicación del flujo de diseño del ISE 8.2i, muestran una frecuencia de operación de 98.8 MHz y un consumo de 7678 *slices* de los recursos del dispositivo Virtex-4, lo cual es correspondiente a la complejidad de la arquitectura. El punto crítico es que la aplicación de las técnicas de codificación aumenta sustancialmente la duración del procesamiento, al pasar de 1947 a 2713 ciclos de reloj, con respecto a la duración en la arquitectura que no considera su realización sobre FPGA. Las técnicas de segmentación, registro de entradas y salidas y limitación con registros del tamaño de los componentes, son básicas en el diseño para comportamiento sobre FPGA, pero su empleo altera los retardos, latencias y ciclos falsos del diseño inicial. Lo anterior justifica un nuevo estudio de la arquitectura FME en su fase de diseño independiente de la tecnología, para determinar las modificaciones estructurales que permitan aumentar la velocidad de procesamiento, al nivel que lo requieren las aplicaciones modernas.

5.6.3 Optimización con enfoque hacia velocidad.

La baja velocidad de procesamiento obtenida en la realización sobre FPGA de la arquitectura FME, motivó la necesidad de analizar su operación, nivel de reutilización y grado de paralelización, para determinar las variantes en su estructura que puedan acelerar su ejecución. Los resultados del estudio generaron las siguientes propuestas:

* Modificación del flujo de procesamiento *half* y *quarter*.

* Incremento de la reutilización de datos con descomposiciones 8×4.

* Aumento del grado de paralelismo con dos secuencias de procesamiento.

En la primera opción se propone la ejecución de la secuencia de procesamiento en pares de bloques, para eliminar el tiempo de espera por la retroalimentación del número del mejor bloque *half* que inicia el ciclo *quarter*. La segunda propuesta toma en cuenta la premisa que se definió en la evaluación de las arquitecturas previas, de que el mayor nivel de reutilización de datos se consigue cuando la unidad de interpolación tiene capacidad para procesar los bloques de mayor tamaño, por lo que para este fin se sustituyó la descomposición 4×4 original por una 8×4, lo que mejora la velocidad de procesamiento de los bloques 8×4, 8×8, 8×16, 16×8 y 16×16. La tercera opción se deriva del hecho de que la aplicación independiente de cualquiera de las

descomposiciones 4×4 y 8×4 no es óptima para procesar todos los tamaños de bloque, por lo que se propone usar una secuencia de procesamiento con descomposición 4×4 para procesar los bloques 4×4 y 4×8 y otra secuencia 8×4 para procesar los bloques restantes.

La tabla 5.34 muestra los resultados normalizados de área y velocidad de las arquitecturas propuestas, con respecto al diseño FME inicial, donde se observa la mayor optimización en velocidad sin el mismo incremento en área, en la arquitectura con secuencia de procesamiento en pares de bloques y descomposiciones 4×4 y 8×4.

Tabla 5.34. Resultados normalizados área-velocidad de las propuestas de optimización, con respecto a la arquitectura FME inicial, operación independiente de la tecnología.

Arquitectura FME	Área	Velocidad de procesamiento
I (a)	1	1
II (a + b)	1	1.089
III (b + c)	1.62	1.65
IV (a + b + c)	2.5	3.255

a. Arquitectura con descomposiciones 4×4.
b. Ejecución de la secuencia de procesamiento en pares de bloques.
c. Arquitectura con descomposiciones 8×4.

Con respecto a los objetivos de diseño (pixeles de 8 bits, formato de video de hasta 1080 HD, velocidad de 30 fps, 1 cuadro de referencia y desplazamiento de exploración máximo de 56 pixeles), la tabla 5.35 muestra la frecuencia a la que deben operar las arquitecturas propuestas para su cumplimiento, por lo que las arquitecturas representan cuatro alternativas para satisfacer los requerimientos de diseño con menor o mayor área, en función de la tecnología de realización aplicada.

Tabla 5.35. Frecuencia de operación en MHz de las arquitecturas FME-FSBM propuestas para cumplir los objetivos de diseño, operación independiente de la tecnología.

	Arquitectura FME-FSBM			
Formato	I	II	III	IV
525 SD	78.853	72.373	47.790	24.219
625 SD	94.62	86.848	57.348	29.063
720p HD	210.276	192.996	127.440	64.584
1080 HD	476.626	437.458	288.864	146.390

Capítulo **6**

Conclusiones y Futuras Líneas

6.1 Conclusiones y principales aportaciones.

En esta tesis doctoral se presenta el diseño y realización de arquitecturas VLSI para la etapa de predicción de la compensación de movimiento del estándar de codificación de video H.264/AVC, utilizando el algoritmo FSBM en sus modalidades de resolución de pixeles enteros y fraccionarios. Los diseños propuestos son estructuras de procesamiento segmentadas-paralelas altamente optimizadas, que combinan las características de las arquitecturas de diversos autores, junto con la aplicación de nuevos esquemas y algoritmos hardware. Como ejemplo de realización, las arquitecturas se han descrito en VHDL, para obtener co-procesadores de aceleración hardware para sistemas de codificación de video sobre FPGA.

Con el objetivo de considerar todas las variables, el trabajo de tesis abarca aspectos de suma importancia para el desarrollo de arquitecturas hardware de estimación de movimiento. Temas como técnicas de reutilización, paralelización, comportamiento en velocidad, arquitecturas sistólicas, medidas hardware, multiprocesamiento, etc., son cubiertos en sus aspectos teóricos, para su aplicación en el diseño de las estructuras propuestas. El resultado es un conjunto de arquitecturas que satisfacen los requisitos funcionales del estándar.

Las aportaciones de mayor importancia realizadas en este proyecto de investigación y documentadas en el presente trabajo, son las siguientes:

1. Se han definido las características estructurales y de operación que deben presentar las arquitecturas de estimación de movimiento, para satisfacer los requerimientos de los codificadores modernos como el H.264/AVC. Su descripción se ha realizado a partir del estudio y evaluación de las arquitecturas hardware consideradas como el estado del arte.

2. Se presenta el diseño de una arquitectura hardware de estimación de movimiento con algoritmo FSBM y resolución de pixeles enteros, con medidas casi ideales de velocidad de procesamiento, latencia y eficiencia. El diseño descrito posee una alta regularidad en el acceso a los datos, obtenida a partir de un grupo de contadores binarios y una ecuación de direcciones simple. Tambien se incluye en el mismo la estructura y administración de las memorias, utilizando bloques básicos de RAM.

3. Se ha propuesto una arquitectura que reduce hasta un 66 % la complejidad computacional de la estimación de movimiento con algoritmo FSBM y resolución

entera, mediante ventanas de búsqueda con formas no cuadradas (circular, rómbica, cruz y elíptica). El diseño parte de la capacidad de modificar en línea el avance de la exploración vertical de la primera arquitectura FSBM fraccionaria. Su evaluación con secuencias de video QCIF sobre el modelo JM, muestra un coste señal/ruido promedio de 0.173 dB en el peor de los casos.

4. Se han determinado las características de diseño de las arquitecturas de estimación de movimiento fraccional de los autores más importantes sobre este tema. También se consideran sus componentes más críticos, la unidad de interpolación y el módulo de transformada Hadamard. Las arquitecturas de las unidades de interpolación son realizaciones directas de los esquemas de la clasificación, por lo que su análisis se hace de acuerdo a la categoría a la que pertenecen. El estudio de los módulos de transformada Hadamard se realiza en función de su clasificación y de las mediciones de comportamiento de las arquitecturas modelo.

5. Se presenta una arquitectura hardware con algoritmo FSBM para la estimación del movimiento fraccionario del componente de video luma. El diseño satisface los requerimientos de tamaño de bloque variable del algoritmo de inter-predicción del estándar H.264/AVC. La arquitectura propuesta tiene un excelente equilibrio área-velocidad, gracias al flujo continuo de datos a la entrada de la unidad de interpolación y a los altos niveles de concurrencia, reutilización de datos y uso de todos sus componentes.

6. Se propone otra versión de la arquitectura de estimación de movimiento fraccionaria, que incrementa 225% la velocidad de procesamiento, con un aumento en área de 150% con respecto al diseño original. Estos resultados se obtienen a partir de la modificación del flujo de procesamiento, el incremento de la reutilización de datos y el aumento del grado de paralelismo, al utilizar dos secuencias de ejecución simultaneas.

7. Como ejemplo de realización, los diseños propuestos se han descrito en lenguaje VHDL para su programación sobre FPGA. Para el apoyo de esta actividad, en el presente trabajo se documentan las técnicas de diseño para reutilización y diseño para comportamiento más importantes. Se incluyen otras recomendaciones al nivel de prácticas de codificación HDL y herramientas de diseño.

6.2 Líneas de investigación futuras.

De acuerdo a la temática que se trata en la presente tesis, las líneas de investigación futuras se pueden ubicar en las siguientes áreas:

* Estándar H.264/AVC.

* Arquitecturas hardware de algoritmos rápidos de estimación de movimiento.

* Estudio del movimiento y procesamiento digital de señales.

La tesis doctoral es parte de un proyecto global, encaminado al desarrollo de un sistema hardware de codificación/decodificación de video de la recomendación

H.264/AVC. Bajo este objetivo, el equipo de trabajo ha terminado el diseño y realización de las etapas de transformación directa e inversa y la de predicción de la compensación de movimiento. Las futuras líneas de investigación se centraran en completar el proyecto, por medio del estudio, propuesta de arquitecturas optimas y realización, de las etapas y componentes de predicción intra, decisión de modo, cuantización, reorden, entropía, filtro antibloqueo, control general, etc.

Las arquitecturas propuestas utilizan el algoritmo FSBM, por lo que una línea de investigación futura puede ser el diseño de arquitecturas hardware para los algoritmos rápidos de estimación de movimiento. La investigación se puede delimitar a los algoritmos que incluye el modelo de referencia JM (*Uneven Multi-Hexagon Search, Simplified Hexagon Search*, etc.). Tambien se puede incluir la optimización y desarrollo completo de la arquitectura IME-FSBM para diferentes formas de SWs. Con esto se tendrá una gama de arquitecturas que cubrirán un amplio abanico de aplicaciones, desde las que requieren la mayor calidad de codificación, hasta las que necesitan área mínima y restringen el consumo de energía. Se puede incluir su realización, aplicando el estilo de diseño VLSI sobre ASIC, completando el proceso hasta la fabricación del dispositivo.

En el desarrollo del presente trabajo, se han encontraron un gran número de referencias a la relación entre el movimiento y el procesamiento digital de señales. Conceptos como movimiento real, modelos de movimiento, actividad de movimiento, velocidad de movimiento, trayectorias de movimiento, etc. son comunes en la bibliografía utilizada. Esto muestra que el estudio del movimiento es un tema actual que puede manejarse como una línea de investigación, no solamente en codificación de video. El filtrado digital, la metrología, el registro de imágenes y la visión artificial, son ejemplos de otras áreas tecnológicas donde la evaluación del movimiento también tiene un gran impacto.

Acrónimos

Las abreviaturas utilizadas en este trabajo son:

AD	Absolute Difference
AGU	Address Generate Unit
ASSP	Application Specific Signal Processor
AVC	Advanced Video Coding
AVS	Audio Video Standard
BFM	Bus Functional Model
BM	Block Matching
BSB	Base System Builder
CAD	Computer Aided Design
CCD	Charge-Coupled Device
CID	Charge-Injection Device
CIF	Common Intermediate Format
CLB	Configurable Logic Block
CMOS	Complementary Metal-Oxide Semiconductor
CPLD	Complex Programmable Logic Device
CRT	Cathode Ray Tube
DC	Direct Current
DCT	Discrete Cosine Transform
DMA	Direct Memory Access
DPCM	Differential Pulse Coding Modulation
DRC	Design Rule Check
DSP	Digital Signal Processor
DVD	Digital Video Disk
DWT	Discrete Wavelet Transform
EDA	Electronic Design Automation
EDIF	Electronic Data Interchange Format
EDK	Embedded Design Kit
ESL	Electronic System Level Design
FF	Flip-Flop
FIFO	First Input First Output
FIR	Finite Impulse Response
FME	Fractional Motion Estimation
FPGA	Field Programmable Gate Array
FPS	Frames Per Second
FSBM	Full Search Block Matching

FSL	Fast Simplex Link
FSM	Finite State Machine
GPP	General Purpose Processor
HDL	Hardware Description Language
HDTV	High Definition Television
HVL	Hardware Verification Language
HVS	Human Visual System
ICT	Integer Cosine Transform
IEC	International Electro-Technical Committee
IME	Integer Motion Estimation
IP	Intellectual Property
ISE	Integrated Software Environment
ISO	International Standard Organization
ITU-T	International Telecommunication Union - Telecommunication
JPEG	Joint Picture Experts Group
JVT	Joint Video Team
LC	Logic Cell
LCD	Liquid Crystal Display
LE	Logic Element
LMB	Local Memory Bus
LUT	Look-Up Table
MAC	Multiply and Accumulate
MB	Macro-Bloque
MBA	Macro-Bloque Actual
MC	Motion Compensation
ME	Motion Estimation
MMU	Management Memory Unit
MPC	Multi-Processor Chip
MPEG	Moving Picture Experts Group
MSE	Medium Square Error
MV	Motion Vector
MVD	Motion Vector Difference
NAL	Network Abstraction Layer
NTSC	National Television System Committee
OCM	On-Chip Memory
OPB	On-Chip Peripheral Bus
PAL	Phase Alternate Line
PCM	Pulse Coding Modulation
PE	Processor Element
PLB	Processor Local Bus

PSNR	Peek Signal Noise Rate
PU	Processor Unit
QCIF	Quarter Common Intermediate Format
R/D	Rate/Distortion
RGB	Red Green Blue
RLC	Run Length Coding
RTL	Register Transfer Level
RTOS	Real Time Operating System
RTS	Real Time System
SAD	Sum of Absolute Differences
SATD	Sum of Absolute Transformed Differences
SDK	Software Debug Kit
SDTV	Standard Definition Television
SoC	System on a Chip
SQCIF	Sub-Quarter Common Intermediate Format
SSE	Sum of Square Errors
SW	Search Window
VCEG	Video Coding Experts Group
VCL	Video Coding layer
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VITAL	VHDL Iniciative Towars ASIC Libraries
VLC	Variable Length Coding
VLSI	Very large Scale Integration
XCF	Xilinx Constrains File
XCL	Xilinx Cache Link
XMD	Xilinx Microprocessor Debug
XPS	Xilinx Platform Studio
XST	Xilinx Synthesis Technology
YCbCr	Luma Croma blue Croma red

Referencias

[Actel00] Actel Corporation, "Actel HDL Coding Style Guide," *Actel Corporation*, USA, 2000, URL: http://www.actel.com/hdl.

[Adams02] L. Adams, "Choosing the Right Architecture for Real-Time Signal Processing Design," *Texas Instrument White Paper*, SPRA879, Nov. 2002.

[Aldec98] Aldec Inc., "EVITA, Enhanced Verilog Tutorial with Applications," *Aldec Inc.*, rev. 1.0, USA 1998, URL: http://www.aldec.com.

[Ambarella07] A2 single chip H.264 codec, *Ambarella Inc.*, January 2007, URL: http://www.ambarella.com/technology/compression.htm.

[Amer04] I. Amer, W. Badawy, and G. Jullien, "Hardware Prototyping for the H.264 4×4 Transformation," *in Proc. IEEE ICASSP 2004*, pp V-77-V-80.

[Anafocus07] Vision Systems on-Chip, *Anafocus Inc.*, 2007, URL: http://www. anafocus.com.

[Asokan07] V. Asokan, "Dual Processor Reference Design Guide," *Xilinx Inc*, XAPP996, v1.1, USA, Nov. 2007.

[Bailo04] G. Bailo, M. Bariani, I. Barbieri, and M. Raggio, "Search Window Size Decision for Motion Estimation Algorithm in H.264 Video Coder," *in Proc. IEEE ICIP 2004*, pp. 1453-1456.

[Bixler05] K. Bixler, "Physical Synthesis and Optimization with ISE Software," *Xilinx Xcell Journal*, pp. 14-16, fourth quarter 2005.

[Chen02] J. Chen, U. V. Koc, and K. J. Ray Liu, "Design of Digital Video Coding Systems," *Marcel Dekker Inc.*, USA, 2002.

[Chen04a] T. C. Chen, Y. W. Huang, and L. G. Chen, "Analysis and Design of Macroblock Pipelining for H.264/AVC VLSI Architecture," *in Proc. IEEE ISCAS 2004*, vol. II, pp. 273-274.

[Chen04b] T. C. Chen, Y. W. Huang, and L. G. Chen, "Fully Utilized and Reusable Architecture for Fractional Motion Estimation of H.264/AVC," *in Proc. IEEE ICASSP 2004*, pp. V-9-V-12.

[Chen06a] C. Y. Chen, S. Y. Chien, Y. W. Huang, T. C. Chen, T. C. Wang, and L. G. Chen, "Analysis and Architecture Design of Variable Block-Size Motion Estimation for H.264/AVC," *IEEE Trans. Circuits Syst.*, vol. 53, no. 2, pp. 578-593, Feb. 2006.

[Chen06b] T. C. Chen, S. Y. Chien, Y. W. Huang, C. H. Tsai, C. Y. Chen, T. W. Chen, and L. G. Chen, "Analysis and Architecture Design of an HDTV720p 30 Frames/s H.264/AVC Encoder," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16 no. 6, pp. 673-688, June 2006.

[Chen07] T. C. Chen, C. Y. Tsai, Y. W. Huang, and L. G. Chen, "Single Reference Frame Multiple Current Macroblocks Scheme for Multiple Reference Frame Motion Estimation in H.264/AVC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17 no. 2, pp. 242-247, Feb. 2007.

[Cheng04] Z. Y. Cheng, C. H. Chen, B. D. Liu, and J. F. Yang, "High Throughput 2-D Transform Architectures for H.264 Advanced Video Coders," *in Proc. IEEE Asia-Pacific Conference on Circuits and Systems 2004*, pp 1141-1144.

[Fernandez06] M. Fernandez, "Maximizing Design Performance for Virtex-5 FPGAs," *Xilinx Xcell Journal*, issue 59, pp. 28-30, fourth quarter 2006.

[Flierl04] M. Flierl and B. Girod, "Video Coding with Superimposed Motion-Compensated Signals," *Kluwer Academic Publishers*, USA, 2004.

[Frank02a] N. Frank, "Frank´s Designing for Speed Tips," 2002, URL: http:// www.brunham.net/kalen/fspeed.html.

[Frank02b] N. Frank, "Frank´s Finite State Machine Design Tips," 2002, URL: http://www.brunham.net/kalen/ffsm.html.

[Freescale07] i.MX27 Multimedia Applications Processor, *Freescale Inc.*, document number IMX27FS, rev. 3, 2007, URL:http://www. freescale.com.

[Fujitsu07] H.264 Format Video-Processing LSI Chip, *Fujitsu LTD.*, July 2007, URL: http://www.fujitsu.com/global/news/pr/archives/month/2006/ 200 61130-01.html.

[Garrault05] P. Garrault and B. Philofsky, "HDL Coding Practices to Accelerate Design Performance," *Xilinx Xcell Journal*, issue 55, pp. 31-35, fourth Quarter 2005.

[GDT07] H.264 IP core encoder, *Global Digital Technologies S.A.*, 2007, URL: http://www.gdt.gr/productsipcores.htm?ipcores/h264.htm.

[Greene07] P. Greene, "The Basics of HD H.264 and Next-generation Encoding," *NVISION and Shubha Tuljapurkar*, Telairity, URL: http://www.embedded.com/columns/ technicalinsights/ 197801034?pgno=1.

[Guan01] L. Guan, S. Y. Kung, and J. Larsen, "Multimedia Image and Video Processing," *CRC Press*, USA, 2001.

[Hallapuro02] A. Hallapuro, M. Karczewicz, and H. Malvar, "Low Complexity Transform and Quantization," *in Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG,* Docs. JVT-B038 and JVT-B039, Jan. 2002.

[Hawkins06] D.W. Hawkins, "High-Speed Signal Processing with FPGAs, CPUs, and DSP," *California Instutute of Technology*, doc. ESC-442.pdf, Dec. 2006.

[He00] Z. L. He, C. Y. Tsui, K. K. Chan, and M. L. Liou, "Low-Power VLSI Design for Motion Estimation Using Adaptive Pixel Truncation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 10, no. 5, pp. 669–678, Aug. 2000.

[He05] Z. He and S. K. Mitra, "From Rate-Distortion Analysis to Resource-Distortion Analysis," *IEEE Circuits and Systems Magazine*, vol. 5, no. 3, pp. 6-18, third quarter 2005.

[Hoffman07] J. Hoffman, "Multimedia on Any Device: Any Network: Seekink the Holy Grail," *Intel Corporation*, USA, 2007, URL: http://www.arc. com/configcon/.

[Huang03] Y. W. Huang, T. C. Wang, B. Y. Hsieh, and L. G. Chen, "Hardware Architecture Design for Variable Block Size Motion Estimation in MPEG-4 AVC/JVT/ITU-T H.264," *in Proc. IEEE ISCAS 2003*, vol. II, pp. 796–799.

[Huang06] Y. W Huang, C. Y. Chen, C. H. Tsai, C. F. Shen, and L. G. Chen, "Survey on Block Matching Motion Estimation Algorithms and Architectures with New

results", *Springer Science Journal on VLSI Signal Processing*, no. 42, pp. 297-320, 2006.

[H.264.05] ITU-T Recommendation H.264 | ISO/IEC International Standard 14496-10 video coding, version 4, March 2005.

[Jack05] K. Jack, "Video Demystified," *Elsevier Newnes*, 4th ed., USA, 2005.

[Jain81] J. R. Jain and A. K. Jain, "Displacement Measurement and its Application in Interframe Image Coding," *IEEE Trans. on Communications*, vol. 29, no. 12, pp. 1799-1808, Dec. 1981.

[Jentz06] B. J. Jentz, "Video and Image processing Using FPGAs," *Video Imaging Design Line, Altera Corp.*, July 2006, URL: http://www.embedded.com/190300989?_requestid=898594.

[JM1106] H.264 Reference Software Version JM11.0, Aug. 2006, URL: http:// bs.hhi.de/~suehring/html.

[Komarek89] T.Komarek and P. Pirsch, "Array Architectures for Block Matching Algorithms," *IEEE Trans. Circuits Syst.*, vol. 36, no. 10, pp. 1301-1308, Oct. 1989.

[Kuhn99] P. Kuhn, "Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation,", *Kluwer Academic Publishers*, 1st ed., Dordrecht, The Netherlands, 1999.

[Kuo05] Y. T. Kuo, T. J. Lin, C. W. Liu, and C. W. Jen, "Architecture for Area-Efficient 2-D Transform in H.264/AVC," *in Proc. IEEE ICME 2005*.

[Lappalainen03] V. Lappalainen, A. Hallapuro, and T. D. Hämäläinen, "Complexity of Optimized H.26L Video Decoder Implementation," *IEEE Trans. Circ. Sys. Video Techn.*, vol. II N. 7, pp. 717-725, July 2003.

[Lass06] S. Lass, "ESL Tools Make FPGAs Nearly Invisible to Designers," *Xilinx Xcell Journal*, issue 58, pp. 6-8, third quarter 2006.

[Li05] M. Li, R. Wang and W. Wu, "The High Throughput and Low Memory Access Design of Sub-pixel Interpolation for H.264/AVC HDTV Decoder," *in Proc. IEEE SIPS 2005*, pp. 296-30.

[Liu93] B. Liu and A. Zaccarin, "New Fast Algorithms for the Estimation of Block Motion Vectors," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 3, no. 2, pp. 148–157, Apr. 1993.

[LSI03] VLE4000 Real-Time H.264/MPEG4-AVC Encoder, *LSI Logic Corporation*, USA 2003.

[Malvar01] H.S. Malvar, "Low-complexity Lengh-4 Transform and Quantization with 16-bit Arithmetic," *in ITU-T SG16*, VCEG-N44, Sept. 2001.

[Man02] Man. Ng. mcn99, "High Level Design for High Speed FPGA Devices," *Department of Computing, Imperial College*, UK, 2002, URL: http://www.doc.ic.ac.uk/~ajf/Teaching/Proyects/Distinguished02/ManNg.pdf.

[ManualEDK8.2i] Xilinx Inc. "User Manual Embedded Development Kit, EDK 8.2i," *Xilinx Inc.*, UG111, v6.0, june 2006.

[ManualISE8.2i] Xilinx Inc. "User Manual Integrated Software Environment, ISE 8.2i," *Xilinx Inc.*, 2006.

[MathStar07] MPEG-4/H.264 Encoder for Arrix FPOAs, part number MIP-H2E02-P12, rev. 1.4, *MathStar Inc.*, April. 2007, URL: http://www. mathstar.com.

[Maxfield04] C. Maxfield, "The Design Warrior´s Guide to FPGAs," *Elsevier Newnes*, USA, 2004.

[Morris06] K. Morris, "Join the Cool Club," *Xilinx Xcell Journal*, issue 58, pp. 9-10, third quarter 2006.

[OL06] OL_H264MCE MultiChanel HDTV H.264/AVC Video Encoder, *Ocean-Logic*, Pty. Ltd., rev. 1.2, 2006, URL: http://www.ocean-logic.com.

[On2_07] Hantro 6280 Video Encoder, Multi-format Configurable Hardware (RTL) Design for Wireless SoCs, *On2*, 2007, URL: http://www. on2.com/index.php?292.

[Oshana07] R. Oshana, "Embedded DSP Software design Using Multicore a System-on-a-chip (SoC) Architecture: Part 2 Software Architecture for a Media SoC," *Texas Instruments*, Nov. 07, URL: http://www. embedded.com.

[Pardo00] F. Pardo y J. A. Boluda, "VHDL Lenguaje para Síntesis y Modelado de Circuitos," *Alfaomega ra-ma*, España, 2000.

[Pereira05] S. Pereira, "Síntesis Toos Strategies," *Xilinx Xcell Journal*, issue 55, pp. 22-23, fourth quarter 2005.

[Philofsky06] B. Philofsky, "HDL Coding and Design Practices for Improving Virtex-5 Utilization, Performance, and Power," *Xilinx Xcell Journal*, issue 59, pp. 19-22, fourth quarter 2006.

[Porto05] M. S. Porto, T. L. Da Silva, R. E. C. Porto, L. V. Agostini, I. S. Da Silva, and S. Bampi, "Design Space Exploration on the H.264 4×4 Hadamard Transform," *in Proc. Norchip 2005*.

[Puri04] A. Puri, X. Chen, and A. Luthra, "Video Coding Using the H.264/MPEG-4 AVC Compression Standard," *Elsevier Trans. Signal Processing Image Communication*, vol. 19, pp. 793-849, 2004.

[Rahman05] C. A. Rahman and W. Badawy, "A Quarter Pel Full Search Block Motion Estimation Architecture for H.264/AVC," *in Proc. IEEE ICME 2005*.

[Reed04] T. R. Reed, "Digital Image Sequence Processing, Compression, and Analysis," *CRC Press*, USA, 2004.

[Richarson03] I. E. G. Richarson, "H.264 and MPEG-4 Video Compresion," *Wiley*, Chichester, 2003.

[Roma02] N. Roma and L. Sousa, "Efficient and Configurable Full-Search Block-Matching Processors," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 12, no.12, pp. 1160-1167, Dec. 2002.

[Saponara04] S. Saponara and L. Fanucci, "Data-Adaptive Motion Estimation Algorithm and VLSI Architecture Design for Low-Power Video Systems," *in Proc. IEE Comput. Digit. Tech 2004*, vol. 151, No. 1, pp. 51-59.

[Schäfer03] R. Schäfer03, T. Wuegan, and H. Schwarz, "The emerging H.264/AVC," *EBU Technical Review*, January 2003.

[Shi99] Y. Q. Shi and H. Sun, "Image and Video Compression for Multimedia Engineering," *CRC Press*, USA, 1999.

[Siewert07] S. Siewert, "SoC Drawer: SoCs and the Digital Content Revolution," *IBM Developer Works*, Jan 2007, URL: http://www.ibm.com/ developerWorks/ powerarchitecturetechnology.

[Song05] Y. Song, Z. Liu, S. Goto, and T. Ikenaga, "A VLSI Architecture for Motion Compensation Interpolation in H.264/AVC," *in Proc. IEEE ASIC 2005*, pp 262-265.

[Srinivasan85] R. Srinivasan and K. R. Rao, "Predictive Coding Based on Efficient Motion Estimation," *IEEE Trans. on Communications*, vol. 33, no 8, pp. 888-896, Aug. 1985.

[Sullivan98] G. J. Sullivan and T. Wiegand, "Rate-Distortion Optimization for Video Compression," *IEEE Signal Processing Magazine*, pp. 74-90, Nov. 1998.

[Symes01] P. Symes, "Video Compression Demystified," *McGraw-Hill*, USA, 2001.

[Tessier02] T. Tessier, "Rethinking Your Verification Strategies for Multimillion-Gate FPGAs," *Xilinx Inc.*, application note XAPP408, v.1.2, Feb. 2002.

[TI05] Digital Media System-on-Chip TMS320DM6446, *Texas Instruments DaVinci Technology*, Dec. 2005, URL: http://www.ti.com.

[Tuan 02] J. C. Tuan, T. S. Chang, and C. W. Jen, "On the Data Reuse and Memory Bandwidth Analysis for Full-Search Block-Matching VLSI Architecture," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 12 no. 1, pp. 61-72, February 2002.

[Tusch06] M. Tusch, "High-Performance Image Processing on FPGAs," *Xilinx Xcell journal*, issue 57, pp. 42-44, second quarter 2006, USA.

[Vitecmm07] Multi-DSP Architecture, Advanced AVC/H.264 Encoder, *Vitecommunication*, 2007, URL: http://www.vitecmm.com/ productv2.php?id=52.

[Vos89] L. De Vos and M. Stegherr, "Parameterizable VLSI Architectures for the Full-Search Block-Matching Algorithm," *IEEE Trans. Circuits Syst.*, vol. 36, no. 10, pp. 1309–1316, Oct. 1989.

[Wang03] T. C. Wang, Y. W. Huang, H. C. Fang, and L. G. Chen, "Parallel 4×4 2D Transform and Inverse Transform Architecture for MPEG-4 AVC/H.264," *in Proc. IEEE ISCAS 2003*, pp. II-800-II-803.

[Wang05] S. Z. Wang, T. A. Lin, T. M. Liu, and C. Y. Lee, "A New Motion Compensation Design for H.264/AVC Decoder," *in Proc. IEEE ISCAS 2005*, pp 4558-4561.

[Wiegand03a] T. Wiegand, G. J. Sullivan, G Bjontegaard, and A. Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE Trans. Circuits Syst. Video technol.*, vol. 13, no 7, pp 560-576, July 2003.

[Wiegand03b] T. Wiegand, H. Schwarz, A. Joch, F. Kossentini, and G.J. Sullivan, "Rate-constrained coder control and comparison of video coding standards," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no 7 pp 688-703, July 2003.

[WikipediaES] Embedded Systems, URL: http://en.wikipedia.org/wiki/ Embedded_sistems.

[WikipediaH264] H.264/MPEG-4 AVC, URL: http://en.wikipedia.org/wiki/H.264.

[W&W06] High Definition H.264 Encoder WW10000BA, *W & W Communications*, 2006. URL: http://www.wwcoms.com/.

[Xcell01] Xilinx Inc. “Special Edition-Virtex-II Platform FPGA,” *Xilinx Xcell Journal*, USA, Summer 2001.

[XilinxDK03] Xilinx Inc., “CoolRunner-II RealDigital CPLD,” *Xilinx Design Kit*, USA, Nov. 2003.

[XilinxEDK8.2i] Xilinx Inc., “Embedded Development Kit (EDK) 8.2i,” *Xilinx Inc*., 2006.

[XilinxISE8.2i] Xilinx Inc., “Integrated Software Environment (ISE) 8.2i,” *Xilinx Inc*., 2006.

[XilinxMEE07] H.264 Motion Estimation Engine, v1.0, DS648, *Xilinx Inc.*, August 2007, URL: http://www.xilinx.com.

[XilinxReuse00] Xilinx Inc., “FPGA Reuse Field Guide,” *Qualis Design Corporatión*, Xilinx, rev. 03_2000_r1, USA 2000, URL: http://www.xilinx.com/ ipcenter/designreuse.

[XilinxReuse02] Xilinx Inc., “Xilinx FPGA Reuse Methodology Manual,” *Xilinx Inc.*, 2nd Ed., 2002, URL: http://www.xilinx.com/ipcenter/designreuse/ docs/ch15_fpga_reuse_ manual .pdf.

[XilinxUG018] Xilinx Inc., “PowerPC 405 Processor Block Reference Guide,” *Xilinx Inc.*, UG018, v. 2.2, USA 2007.

[XilinxUG081] Xilinx Inc., “MicroBlaze Processor Reference Guide,” *Xilinx Inc.*, UG081, v. 8.0, USA 2007.

[XilinxWP231] Xilinx Inc., “HDL Coding Practices to Accelerate Design Performance,” *Xilinx Inc.,* white paper 231, v.1.1, Jan 2006.

[XilinxWP262] Xilinx Inc., “Designing Multiprocessor Systems in Platform Studio,” *Xilinx Inc.,* white paper 262, v.2.0, Nov. 2007.

[XilinxWP272] Xilinx Inc., “Get Smart About Reset: Think Local, not Global,” *Xilinx Inc.,* white paper 272, v.1.0, Jan 2008.

[Yang06] C. Yang, S. Goto, and T Ikenaga, “High Performance VLSI Architecture of Fractional Motion Estimation in H.264 for HDTV,” *in Proc. IEEE ISCAS 2006*, pp. 2605-2608.

[YUVqcif] YUV 4:2:0 Video QCIF Sequences, URL: http://trace.eas.asu. edu/ yuv/index.html.

[Zhang05] L. Zhang and W. Gao, “Improved FFSBM Algorithm and its VLSI Architecture for Variable Block Size Motion Estimation of H.264,” *in Proc. IEEE ISPCS 2005*, pp. 445-448.

[Zhao06] Z. Zhao and P. Liang, “A Highly Efficient Parallel Algorithm for H.264 Video Encoder,” *in Proc. IEEE ICASSP 2006*, vol. V, pp. 489-492.

[Zuloaga98] A. Zuloaga y J.L. Martín, “Visión Artificial Dinámica, Determinación de Movimiento a Partir de Secuencias de Imágenes,” *Universidad del País Vasco*, Bilbao 1998, URL: http://www.geocities.com/aitzol.geo/docu003.pdf.

[Zurawski06] R. Zurawski, “Embedded Systems Handbook,” *CRC Press*, Taylor & Francis Group, LLC, USA 2006.

[4i2i05] H.264 Encoder, Baseline, *4i2i Communications Ltd.*, August 2005.
URL: http://www.4i21.com.

Anexo - Artículos Publicados

a) Artículos en congresos.

A. Mora, F. J. Ballester, M. A. Martínez, and J. A. Canals, **"High Parallel Pipeline Integer-pel and Fractional-pel Motion Estimation VLSI Architecture for H.264/AVC,"** *in Proc. Microtechnologies for the New Millennium SPIE Europe*, vol. 6590, Gran Canaria, Spain, May 2007, URL: http://spiedigitallibrary.aip.org/dbt/dbt.jsp?KEY=PSISDG&Volume=6590&Issue=1&bproc=symp&scode=EMT07#MAJOR8.

J. A. Canals, F. J. Ballester, M. A. Martínez, and A. Mora, **"New FPSoC-based Architecture for Efficient FSBM Motion Estimation Processing in Video Standards,"** *in Proc. Microtechnologies for the New Millennium SPIE Europe*, vol. 6590, Gran Canaria, Spain, May 2007, URL: http://spiedigitallibrary.aip.org/dbt/dbt.jsp?KEY=PSISDG&Volume=6590&Issue=1&bproc=symp&scode=EMT07#MAJOR8.

A. Mora, F. J. Ballester, M. A. Martínez, and J. A. Canals, **"Integer-pixel motion estimation H.264/AVC hardware accelerator,"** *in Proc. DCIS'06 XXI edition*, ISBN 978-84-690-4144-4, Barcelona, Spain, Nov. 2006.

A. Mora, F. J. Ballester, M. A. Martínez, and J. A. Canals, **"Fractional-pixel motion estimation H.264/AVC hardware accelerator,"** *in Proc. DCIS'06 XXI edition*, ISBN 978-84-690-4144-4, Barcelona, Spain, Nov. 2006.

J. A. Canals, F. J. Ballester, M. A. Martínez, and A. Mora, **"Efficient FSBM Algorithm Motion Estimation implementation over FPSoC based Architecture,"** *in Proc. DCIS'06 XXI edition*, ISBN 978-84-690-4144-4, Barcelona, Spain, Nov. 2006.

A. Mora, F. J. Ballester, M. A. Martínez, G. Payá, **"1-Gigabit Ethernet Media Access Controller Core,"** *in Proc. DCIS'03 XVIII edition*, ISBN 84-87087-40-X, pp. 353-356, Ciudad Real, Spain, Nov. 2003.

b) Artículo en revista.

A. Mora, F. J. Ballester, M. A. Martínez, and J. A. Canals, **"Integer-pixel motion estimation H.264/AVC accelerator architecture with optimal memory management,"** *Elsevier Microprocessors and Microsystems*, vol. 32, issue 2, pp. 68-78, March 2008, URL: http://www.sciencedirect.com/science/journal/01419331.