UNIVERSIDAD POLITÉCNICA DE MADRID

Escuela Universitaria de Ingeniería Técnica de Telecomunicación

CONTRIBUCIÓN A LAS METODOLOGÍAS DE OPTIMIZACIÓN DEL TIEMPO DE EJECUCIÓN DE ALGORITMOS DE DESCODIFICACIÓN DE VÍDEO SOBRE DSPs.

TESIS DOCTORAL

Autor: Fernando Pescador del Oso

Ingeniero de Telecomunicación

Directores:

César Sanz Álvaro

Doctor Ingeniero de Telecomunicación

Matías J. Garrido González

Doctor Ingeniero de Telecomunicación

Julio 2011

TÍTULO: Contribución a las metodologías de optimización del tiempo de ejecución de algoritmos de descodificación de vídeo sobre DSPs.

AUTOR: D. Fernando Pescador del Oso.

DIRECTORES: D. César Sanz Álvaro.

D. Matías J. Garrido González.

El Tribunal nombrado con fecha 1 de Julio de 2011 por el Mgfco. y Excmo. Sr. Rector de la Universidad Politécnica de Madrid, compuesto por los doctores:

Presidente:	D. Narciso García Santos
Vocal:	D. Antonio Núñez Ordóñez
Vocal:	D. Francisco José Ballester Merelo
Vocal:	D. Javier Morán Carrera
Secretario:	D. Juan M. Meneses Chaus
Suplente:	D. Gustavo Marrero Callicó
Suplente:	D. José Manuel Fernández Fernández

realizado el acto de lectura y defensa de la tesis en la Escuela Universitaria de Ingeniería Técnica de Telecomunicación de la UPM, acuerda otorgar la calificación de

_____

Madrid, a 20 de Julio de 2011

EL PRESIDENTE EL SECRETARIO

LOS VOCALES

"Queda prohibido llorar sin aprender,
levantarte un día sin saber que hacer,
tener miedo a tus recuerdos.
Queda prohibido no sonreír a los problemas,
no luchar por lo que quieres,
abandonarlo todo por miedo,
no convertir en realidad tus sueños.
Queda prohibido no demostrar tu amor,
hacer que alguien pague tus deudas y el mal humor.
Queda prohibido dejar a tus amigos,
no intentar comprender lo que vivieron juntos,
llamarles solo cuando los necesitas.
Queda prohibido no ser tú ante la gente,
fingir ante las personas que no te importan,
hacerte el gracioso con tal de que te recuerden,
olvidar a toda la gente que te quiere.
Queda prohibido no hacer las cosas por ti mismo,
tener miedo a la vida y a sus compromisos,
no vivir cada día como si fuera un ultimo suspiro.
Queda prohibido echar a alguien de menos sin
alegrarte, olvidar sus ojos, su risa,
todo porque sus caminos han dejado de abrazarse,
olvidar su pasado y pagarlo con su presente.
Queda prohibido no intentar comprender a las personas,
pensar que sus vidas valen mas que la tuya,
no saber que cada uno tiene su camino y su dicha.
Queda prohibido no crear tu historia,
no tener un momento para la gente que te necesita,
no comprender que lo que la vida te da, también te lo quita.
Queda prohibido no buscar tu felicidad,
no vivir tu vida con una actitud positiva,
no pensar en que podemos ser mejores,
no sentir que sin ti este mundo no sería igual".

Poema atribuido a Pablo Neruda

A la memoria de Javier,
gracias por regalarnos tu alegría,
tu sonrisa, tu esfuerzo, tu ilusión….
Siempre estaremos en deuda contigo.

AGRADECIMIENTOS.

Esta tesis sólo ha sido posible gracias al apoyo de muchas personas que de una forma más o menos directa me han ayudado. Con estas líneas quiero agradecerles su apoyo a lo largo de esta travesía del desierto.

A Maricruz, porque si he sido capaz de llegar hasta aquí ha sido gracias a ti y a tantas veces que te he robado tu tiempo para poder hacer un experimento o escribir un artículo. Tu apoyo durante todo este tiempo ha sido imprescindible, gracias por hacerme cada día un poco mejor. Te quiero.

A mi familia, a mis padres, a mi tía, a mis hermanos, a mi cuñada y a mi sobrina. Sois un ejemplo de lucha y superación, un espejo en el que siempre conviene mirarse. Juntos seremos capaces de superar todas las dificultades.

A los miembros del Grupo de Diseño Electrónico y Microelectrónico con los que siempre ha sido un placer trabajar. Todos los proyectos que hemos afrontado juntos me han hecho aprender y crecer como ingeniero. Espero que siga siendo así mucho tiempo.

A mis compañeros del Dpto. de Sistemas Electrónicos y de Control por haber generado un ambiente de trabajo envidiable en el que uno se siente entre amigos. Todos esos buenos ratos durante las comidas y los cafés no tienen precio.

A la compañía SIDSA y en especial a Carlos Santos, por creer en nosotros y financiar parcialmente el desarrollo de esta tesis. Lástima que la industria española no sepa valorar una labor de ingeniería como la que habeis realizado durante tantos años. Seguro que el futuro os deparará la suerte que mereceis.

A César y a Matías, directores y sobre todo amigos, por creer en esta tesis más que yo mismo. Vuestros consejos han sido imprescindibles para llegar hasta aquí y espero poder seguir disfrutando de ellos durante mucho más tiempo.

A Jesús que me dio la oportunidad de comenzar en esta profesión y me mostró lo apasionante que es enseñar a los demás. Tu motivación por la enseñanza ha sido y sigue siendo una referencia para mí.

A Deivid, ingeniero perfeccionista e incansable por haberme ayudado durante todo este tiempo en este trabajo tan ingrato de reducir ciclo a ciclo el tiempo de ejecución de todos y cada uno de los descodificadores. Ojalá tengas mucha suerte en tu vida profesional, te lo mereces.

A todos los alumnos con los que he tenido la suerte de trabajar durante estos años (Gonzalo, Ernesto, Rafa, Esther, Fran, Miguel, Oscar, Javier, Alejandro, Christian, David Sanz, Álvaro). Espero no haberme olvidado de ninguno porque de todos he aprendido algo y todos sois partícipes de este resultado.

A mis amigos, Bea, Isabel, Jesús, Juana, Juanra, Juan Ignacio, Mª José, Noe, Rafa, Sergio y Vicente, por tantos años llenos de buenos momentos y otros no tan buenos en los que siempre me habéis demostrado lo mucho que me queréis. Espero poder disfrutar de vuestra amistad durante mucho tiempo.

A todos y cada uno, gracias de corazón.

ÍNDICE GENERAL

RELACIÓN DE FIGURAS

RELACIÓN DE TABLAS

RESUMEN

La presente tesis se enmarca dentro de las líneas de investigación que desarrolla el Grupo de Diseño Electrónico y Microelectrónico de la Universidad Politécnica de Madrid centradas en la codificación y descodificación de vídeo. Dentro de esta línea de actividad se han desarrollado previamente tesis en las que se ha investigado en arquitecturas *hardware* orientadas a la codificación de vídeo digital. Sin embargo, la constante aparición de nuevos estándares de codificación de vídeo y el tiempo requerido para poder realizar implementaciones empleando arquitecturas *hardware* específicas, hacen que sea necesario plantearse otras soluciones tecnológicas más flexibles, en las que además se reduzca el tiempo de desarrollo (*time to market*) de las aplicaciones.

En este sentido, la aparición en el mercado de los denominados procesadores multimedia, compuestos de un procesador digital de señal (DSP) y una serie de periféricos orientados a las aplicaciones de vídeo, supone una alternativa tecnológica interesante debido fundamentalmente a su flexibilidad. En la mayoría de los casos, el diseño de codificadores y descodificadores basados en DSP toma como punto de partida un código de referencia, pensado para ser ejecutado en un ordenador personal. Este código se porta al DSP y se optimiza en velocidad hasta alcanzar el funcionamiento en tiempo real. Aunque en los últimos años se han publicado gran cantidad de trabajos en los que se describen técnicas de optimización en velocidad para codificadores y descodificadores sobre tecnología DSP, no se han encontrado publicaciones en las que se describan metodologías de trabajo que tengan en cuenta el proceso completo y que ayuden a llevarlo a cabo de manera más eficiente.

Investigar en una metodología que permita abordar todas las etapas del diseño y desarrollo de un sistema completo de codificación/descodificación de televisión digital sobre procesadores digitales de señal ha sido el principal objetivo de esta tesis. Con objeto de darle la mayor generalidad posible, la metodología se ha elaborado a partir de los datos recopilados en varias implementaciones de descodificadores compatibles con un conjunto de estándares y utilizando diferentes DSPs. Aunque en los experimentos sólo se han implementado descodificadores, la metodología puede ser también de utilidad en el diseño de codificadores, dado que en gran medida se utilizan algoritmos similares. La investigación realizada para proponer esta metodología se ha llevado a cabo en las cuatro fases que se resumen a continuación.

En primer lugar, se ha llevado a cabo un estudio de los estándares de codificación MPEG-2, MPEG-4 y H.264 desde el punto de vista de las herramientas que emplean. Este estudio ha permitido constatar las similitudes existentes entre ellos, lo que facilita que las metodologías de optimización definidas en esta tesis, sean aplicables a todos ellos.

En segundo lugar, se ha realizado un análisis exhaustivo del estado del arte en dos campos clave para el desarrollo de la tesis: los DSPs de última generación y las técnicas de optimización de codificadores y descodificadores de vídeo basados en DSP. Como resultado de este análisis se ha comprobado que la arquitectura interna de todos los DSPs existentes actualmente en el mercado es similar, lo que facilita que los métodos de optimización que se han validado para alguno de ellos sean aplicables para el resto. Por otro lado, se han recopilado gran cantidad de trabajos en los que se describen técnicas de optimización en velocidad para codificadores y descodificadores de vídeo sobre tecnología DSP. Sin embargo, como se ha mencionado antes, no se han encontrado publicaciones en las que se describan metodologías generales de trabajo.

En tercer lugar, se han implementado tres descodificadores de vídeo compatibles con los estándares MPEG-2, MPEG-4 y H.264 empleando los procesadores de señal TMS320DM642 y TMS320DM6437. En cada una de estas implementaciones, se han utilizado una serie de técnicas de optimización para reducir el tiempo de ejecución. Con ellas se ha logrado en todos los casos el funcionamiento en tiempo real para los tres estándares empleando secuencias de vídeo de definición estándar (SD); mejorando en muchos casos las prestaciones de los descodificadores que pueden encontrarse en la literatura científica. Estás técnicas se han clasificado en tres grupos: las relacionadas con la gestión del código y los datos en los diferentes niveles de memoria, las relativas al movimiento de datos entre memoria interna y memoria externa y las que permiten aprovechar la arquitectura SIMD de los DSPs. Para cada técnica de optimización empleada en cada implementación se ha generado una ficha en la que se describe su aplicación y se razona su posible generalización en la optimización de descodificadores compatibles con otros estándares o en implementaciones con diferentes DSPs.

En cuarto lugar, con objeto de realizar pruebas de funcionamiento con emisiones de televisión reales, se ha desarrollado íntegramente dentro del marco de esta tesis un sistema completo de recepción de televisión digital vía IP (*Set Top Box IP*). Esta plataforma ha permitido completar la metodología de optimización con algunas recomendaciones que afectan a la realización de un sistema completo. Para llevar a cabo este *Set Top Box IP* se ha diseñado una tarjeta de prototipado basada en el procesador TMS320DM642 y se ha empleado otra tarjeta comercial basada en el TMS320DM6437.

Como conclusión, a partir de la información recopilada en los experimentos antes mencionados, se ha sintetizado una metodología de optimización de algoritmos de codificación/descodificación de vídeo para procesadores digitales de señal. Esta metodología se basa en una serie de recomendaciones que deben aplicarse de forma secuencial para mejorar las prestaciones de los codificadores/descodificadores. Si bien algunas de las técnicas de optimización que se han utilizado en la tesis aparecen de forma dispersa en diferentes publicaciones, hasta el momento no se ha encontrado en la literatura científica una metodología de diseño que unifique la aplicación de esas técnicas, desde el portado del código de referencia al DSP, hasta la implementación de un sistema completo. La adopción de esta metodología en futuros diseños permitirá reducir de forma sustancial el tiempo necesario para implementar codificadores/descodificadores basados en DSPs.

ABSTRACT

This Ph. D. work is integrated in the research topics carried out by the Electronic and Microelectronic Research Group of the Universidad Politécnica de Madrid. These topics are focused on implementing video coding applications. Several theses have been developed previously in this line of activity. In these theses the research has been focused on specific hardware architectures oriented to the implementation of digital video encoders. However, the constant emergence of new video coding standards and the time required to develop specific implementations using hardware architectures, make it necessary to consider other flexible technology solutions. These solutions must also reduce the development time (time to market) of this kind of applications.

The emergence of the so-called media processors that consist of a digital signal processor (DSP) and some peripherals targeted at video applications is an interesting technological alternative due to its flexibility. The design of encoders and decoders using DSPs starts with the selection of a reference software implementation, designed to run on a personal computer. This code is migrated to the DSP environment and optimized to achieve real time operation.

During the last years several papers have been published focused on the optimization of video encoders and decoders using DSP technology. However, no publications were found describing a working method that take into account the complete process and help to carry out more efficiently.

The main objective of this thesis is the research on a methodology to design and develop a complete system of encoding/decoding of digital television on digital signal processors. In order to provide a methodology as general as possible, it has been developed using the data obtained after implementing decoders compatible with a set of standards and using different DSPs. The experiments were performed for implementations of decoders, but the methodology can be also useful in the design of encoders, largely because similar algorithms are used. The research to achieve the proposed methodology has been carried out in four phases that are summarized below.

First, a study of coding standards MPEG 2, MPEG 4 and H.264 from the point of view of the tools they use is presented. This study shows their similarities, which has facilitated the application of optimization methodologies defined to all of them.

Secondly, an analysis of the state of the art in two key areas for the development of the thesis has been done: the latest generation DSPs and the optimization techniques of video encoders and decoders based on DSP. This analysis has shown that the internal architecture of all DSPs is similar. This enables that the optimization methods validated for some of them are relevant to the rest. On the other hand, a lot of papers describe optimization techniques for video encoders and decoders on DSP technology. However, as mentioned before, no publications were found describing the overall working methods.

Thirdly, the implementations of MPEG-2, MPEG-4 and H.264 using TMS320DM642 and TMS320DM6437 signal processors are described. Some optimizations techniques have been used to reduce the execution time. Real time performance has been achieved for the three standard video decoders for standard definition (SD) sequences. These implementations have a better performance than that of others described in the literature.

These techniques have been classified in three groups: those related to the management of code and data at different levels of internal DSP memory, those related to the data movement between internal and external memories and those who take

advantage of the SIMD architecture. Each optimization technique has been documented and a summary has been created. This summary describes the technique implementation and its possible generalization to the optimization of decoders compatible with other standards or implementations with different DSPs.

Fourthly, a complete digital TV reception via IP (Set Top Box IP or STB-IP), developed entirely within the framework of this thesis, is described. This STB-IP has been developed in order to perform tests with real TV broadcasts. This platform has allowed completing the optimization methodology with some recommendations that affect to complex systems. Two prototypes boards have been used to implement the STB-IP: one designed within the framework of this thesis based on the TMS320DM642 DSP and a commercial board based on the TMS320DM6437 DSP.

A methodology for optimizing algorithms for encoding or decoding digital video using digital signal processors has been synthesized using the information obtained in previous experiments. This methodology is based on some recommendations to be applied sequentially to improve the performance of the encoders or decoders.

Some of the optimization techniques that have been used in the thesis are described in different publications but, a general methodology that unifies the application of these techniques from the selection of the reference software to the development of a complete system has not been presented so far. The application of this methodology in future designs will substantially reduce the time needed to implement encoders or decoders based on DSPs.

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Introducción

La digitalización de la señal de televisión, que comenzó a realizarse en la década de los 80, abrió un abanico de aplicaciones que a día de hoy resultan cotidianas, tales como la televisión de alta definición, el vídeo bajo demanda, la televisión sobre IP, el vídeo en terminales móviles o la televisión tridimensional.

Todas estas aplicaciones están basadas en la captura, transmisión, difusión, almacenamiento y/o reproducción de vídeo digital y en todas ellas es necesario emplear técnicas de compresión que reduzcan de forma drástica el régimen binario de la señal de vídeo codificada. Estas técnicas aprovechan las características del ojo humano para obtener relaciones de compresión superiores a 200:1 [RH96]. Aunque existen diversas técnicas para la compresión de imágenes estáticas o en movimiento, en todas las aplicaciones descritas anteriormente se emplea un esquema de codificación híbrido basado en la transformación de los píxeles de la imagen al dominio de la frecuencia, la estimación de movimiento y la codificación de entropía. Empleando estas técnicas se reduce la redundancia espacial y temporal existente en cualquier secuencia de vídeo natural con la consiguiente reducción en el régimen binario necesario para su codificación.

El desarrollo de estas técnicas de compresión de la señal de vídeo digital ha venido de la mano de organismos internacionales de estandarización, los cuales han ido generando a lo largo de las últimas tres décadas normas que definen herramientas para reducir el ancho de banda de la señal de vídeo.

Ya en 1984, el CCIR (actualmente la ITU-R) crea un grupo especializado en videotelefonía, cuyos trabajos dan como resultado el desarrollo de la Recomendación H.261 [RH96] que tiene como objetivo la codificación de audio y vídeo en telefonía para regímenes binarios múltiplos de 64 kbps. De forma paralela, desde 1988, la ISO crea el grupo de trabajo MPEG (*Moving Pictures Expert Group*) que desarrolla el estándar MPEG-1 para codificación de video y audio a un régimen binario de hasta 1.5 Mbps. Estos dos estándares ya empleaban el esquema de codificación híbrido mencionado anteriormente.

Ambos grupos de trabajo continuaron desarrollando nuevos estándares en la década de los 90. Así, en 1994, la ISO establece la primera versión del estándar MPEG-2 para la codificación de TV digital con regímenes binarios de hasta 10 Mbps que sigue siendo ampliamente utilizado. Este estándar ha sido actualizado en numerosas ocasiones mediante adendas y correcciones, siendo las últimas de este mismo año (2011).

En 1998, la ITU-T establece la Recomendación H.263 [ITU98] para la transmisión de vídeo a través de canales de baja velocidad. Un año más tarde, en 1999, la ISO establece el estándar MPEG-4 para la codificación, tanto de vídeo natural, como de vídeo sintético.

Posteriormente ambos organismos crean un grupo de trabajo conjunto (*Joint Video Team - JVT*) para desarrollar un nuevo estándar que acabó denominándose MPEG-4 Parte 10 (2003) en la terminología ISO y H.264 en la ITU [ITU03]. Más recientemente ambos organismos han seguido colaborando en su ampliación del estándar H.264 para incluir la codificación escalable (*Scalable Video Coding* o SVC) que ha sido publicada en 2007 [ITU07] y la codificación multivista (*Multiview Video Coding* o MVC) estandarizada en 2009 [ITU09]. Finalmente, en el año 2011 este grupo de trabajo ha presentado los primeros borradores de un nuevo estándar denominado HEVC (*High Efficient Video Coding*) dentro de la ISO y H.265 en la terminología ITU [ITU11].

El panorama de los estándares de codificación de vídeo se ha complicado aún más durante los últimos años con la aparición de estándares desarrollados por el SMPTE (*Society of Motion Picture and Television Engineers*), más concretamente los denominados VC-1 [SMP06] y VC-3 [SMP08] que han sido adoptados por empresas de gran relevancia a nivel mundial.

1.2 Justificación y Objetivos de la tesis

La constante evolución de los estándares de codificación de vídeo y las sucesivas modificaciones que se efectúan sobre los mismos, hace que los tiempos de desarrollo de los codificadores y descodificadores tengan que ser más cortos. También es cada vez más frecuente la realización de modificaciones sobre las implementaciones existentes.

La codificación y la descodificación de vídeo son aplicaciones muy exigentes que requieren soportes tecnológicos con una capacidad de cómputo elevada. En la actualidad, las diferentes alternativas tecnológicas que se utilizan para implementar este tipo de aplicaciones pueden clasificarse en tres grandes grupos: las basadas en procesadores de propósito general, las basadas en arquitecturas específicas y las basadas en procesadores digitales de señal.

Desde 1996, el Grupo de Diseño Electrónico y Microelectrónico (GDEM) de la E.U.I.T. de Telecomunicación perteneciente a la Universidad Politécnica de Madrid ha continuado con los trabajos comenzados en la E.T.S.I. de Telecomunicación de la misma universidad en los que se ha investigado en arquitecturas *hardware* orientadas a la codificación de vídeo digital. En el marco de esta investigación se desarrollaron tres tesis doctorales [Fer98] [San98] [Gar04] en las que se propusieron arquitecturas específicas avanzadas para la realización de codificadores de vídeo.

En paralelo con el desarrollo de estas tesis, algunas compañías sacaron al mercado procesadores digitales de señal que poseen una capacidad de cómputo elevada, lo que permite implementar aplicaciones de codificación y descodificación de vídeo sobre ellos. Adicionalmente, estos procesadores incorporan una serie de periféricos (puertos de vídeo de entrada y salida, convertidores digitales-analógicos o

acceso a la red local) que facilitan las interfaces propias de este tipo de aplicaciones, permitiendo implementar sistemas completos en un solo chip (*System on Chip* o SoC).

Las ventajas de estos procesadores respecto a las arquitecturas *hardware* específicas se encuentran en la facilidad para modificar el estándar que implementan, ya que sólo supone una sustitución del *software* que ejecuta el procesador. Esta versatilidad permite disponer de sistemas de codificación/descodificación con tiempos de desarrollo menores que los obtenidos con implementaciones basadas en ASICs. Como contrapartida, tanto el precio, como el consumo de los DSPs son superiores a los de los sistemas diseñados específicamente para implementar un estándar.

En un panorama tan cambiante como el de la codificación de vídeo, la solución tecnológica basada en DSP constituye una alternativa interesante para el desarrollo de ciertos tipos de aplicaciones. Por este motivo, el GDEM comienza en el año 2004 a investigar en los métodos de trabajo que permiten la implementación de codificadores/descodificadores de vídeo sobre estas plataformas. La presente tesis se enmarca en esta línea de trabajo, siendo la primera en finalizarse.

Generalmente, el punto de partida utilizado en el diseño de codificadores/descodificadores con DSP es un *software* de referencia que implementa el codificador o descodificador objetivo. Este código puede ser el *software* de referencia desarrollado dentro del proceso de estandarización o cualquier otro, habitualmente ejecutable sobre ordenador personal. El diseño de codificadores/descodificadores para DSP pasa, pues, por realizar una migración del código al DSP y optimizarlo en velocidad, siendo el objetivo principal el funcionamiento en tiempo real.

La implementación de codificadores/descodificadores sobre DSP requiere seleccionar un *software* de referencia, portarlo al DSP, adaptar el código a la arquitectura del DSP para explotar mejor sus características, integrar el codificador/descodificador en un sistema completo y realizar las pruebas de funcionamiento. Como en cualquier proceso complejo, en este caso resulta de mucha utilidad seguir una metodología de trabajo eficiente. En los últimos años se han publicado gran cantidad de trabajos en los que se describen técnicas de optimización en velocidad para codificadores y descodificadores sobre tecnología DSP. Sin embargo, no se han encontrado publicaciones en las que se describan metodologías de trabajo que tengan en cuenta el proceso completo y que ayuden a llevarlo a cabo de manera más sistemática y eficiente.

Por razones de oportunidad, además de como forma de acotar el trabajo, la tesis doctoral se ha centrado en los descodificadores de vídeo, aunque muchas de las conclusiones alcanzadas son igualmente aplicables a los codificadores. El principal objetivo de la tesis puede enunciarse de la siguiente manera:

Elaboración de una metodología de optimización del tiempo de ejecución de descodificadores de vídeo con tecnología DSP que tenga en cuenta todas las fases del desarrollo de este tipo de aplicaciones.

La síntesis de la metodología de optimización se ha basado en la realización de varias implementaciones de diferentes descodificadores (MPEG-2, MPEG-4 y H.264) sobre diferentes DSPs (TMS320DM642 y TMS320DM6437). Las diferentes técnicas y procedimientos empleados en cada uno de estos desarrollos han sido evaluados y documentados. En la fase de síntesis de la metodología, se han seleccionado las técnicas y procedimientos que mejoran el rendimiento y que son susceptibles de ser generalizados a otras implementaciones. La identificación de las similitudes que poseen tanto los estándares de codificación de vídeo MPEG-2, MPEG-4 y H.264, como los procesadores digitales de señal que existen actualmente en el mercado, ha permitido generalizar la metodología extraída, tanto a la implementación de

descodificadores de otros estándares, como a la utilización de diferentes procesadores.

Si bien el objetivo principal de la tesis es definir una metodología de optimización en velocidad para descodificadores basados en DSPs, para poder alcanzarlo se han definido dos objetivos secundarios. Por un lado, la implementación de sendos descodificadores compatibles con MPEG-2, MPEG-4 y H.264 funcionando en tiempo real para resolución estándar (*Standard Definition*). La implementación de descodificadores con estos requisitos de funcionamiento permite identificar algunos problemas de optimización que no aparecen con resoluciones menores o con un menor número de imágenes por segundo. Por otro lado, la implementación de un sistema completo de descodificación de televisión IP (*Set Top Box* IP o STB-IP) que posibilite la realización de pruebas con emisiones de televisión reales. Para realizarlo es necesario implementar todas las tareas que lo componen: análisis de la trama de transporte MPEG-2, descodificación de audio y presentación sincronizada de audio y vídeo. Este STB-IP permite analizar la influencia que tiene en las prestaciones de los descodificadores su integración en un sistema complejo, con lo cual se pueden obtener conclusiones metodológicas a nivel de sistema.

1.3 Organización de la memoria

La memoria se ha estructurado en 6 capítulos y 3 anexos. El primer capítulo es esta introducción donde se enmarca la tesis dentro de la línea de investigación en sistemas de codificación y descodificación de vídeo digital que se lleva a cabo en el GDEM de la Universidad Politécnica de Madrid y se explican sus objetivos.

En el capítulo 2 se incluyen algunos fundamentos de codificación de vídeo digital y se analizan los tres estándares de codificación de vídeo que se han empleado a lo largo del desarrollo de la tesis (MPEG-2, MPEG-4 y H.264). Este estudio pone de manifiesto las similitudes entre estos estándares lo que facilitará la tarea de optimización de estos algoritmos sobre los procesadores digitales de señal. Además, a lo largo del capítulo se establece la terminología básica que se emplea posteriormente en la memoria.

En el capítulo 3 se realiza una revisión del estado del arte en tres aspectos clave para el desarrollo de esta tesis: análisis de las soluciones tecnológicas que se vienen empleando en los últimos años para el desarrollo de codificadores y descodificadores de vídeo; estudio de las arquitecturas de los procesadores digitales de señal existentes actualmente en el mercado; y, revisión de las publicaciones científicas en las que se presentan técnicas de optimización de codificadores y descodificadores de vídeo para procesadores digitales de señal.

En el capítulo 4 se explican con detalle las técnicas de optimización que se han aplicado para reducir el tiempo de ejecución de los algoritmos de descodificación de video sobre dos procesadores digitales de señal concretos. Estas técnicas son la base experimental que ha permitido proponer una metodología de optimización de este tipo de aplicaciones. A lo largo del capítulo, mediante una serie de fichas, se describe brevemente cada técnica y se indica su aplicabilidad, tanto a otros estándares de codificación, como a otros procesadores digitales de señal. Para cada una de las técnicas y de los estándares de codificación, se presentan los resultados que se han obtenido como consecuencia de su aplicación empleando tres bancos de pruebas: el descodificador aislado, el descodificador integrado en un sistema completo de recepción de televisión digital con entrada de datos IP y el descodificador integrado en un entorno real de difusión de televisión digital. De esta forma se obtienen técnicas de optimización aplicables no sólo al descodificador propiamente dicho, sino que también son de aplicación cuando éste se integra en un sistema complejo.

En el capítulo 5, a partir de la información obtenida en los experimentos que se detallan en el capítulo 4, se sintetiza la metodología de optimización para la reducción del tiempo de ejecución de descodificadores de vídeo sobre procesadores digitales de señal que constituye la principal aportación de esta tesis. Esta metodología cubre todas las fases del desarrollo de este tipo de aplicaciones, desde la elección de un *software* que sirva como punto de partida, hasta la integración del descodificador optimizado en un sistema completo de recepción de televisión digital.

El capítulo 6 contiene un resumen de las aportaciones realizadas en la tesis: la implementación de descodificadores de vídeo compatibles con los estándares MPEG-2, MPEG-4 y H.264 funcionando en tiempo real, el desarrollo de un banco de pruebas completo para la recepción de televisión digital a través de la red IP y la síntesis de una metodología de optimización de algoritmos para procesadores digitales de señal. Además, se mencionan otros resultados de la tesis como las publicaciones y comunicaciones a congresos que se han llevado a cabo durante su desarrollo. Finalmente, se proponen algunas ideas que permitirían continuar con la línea de trabajo iniciada con esta tesis.

En el Anexo A se muestran las tres tarjetas de prototipado que se han empleado para aplicar las técnicas de optimización. Una de ellas (DESCOS) se ha diseñado y fabricado dentro del desarrollo de esta tesis, mientras que las otras dos (EVMDM642 y DM6437EVM) son tarjetas de prototipado comerciales.

El Anexo B describe el sistema de descodificación completo que se ha desarrollado, empleando las tarjetas de prototipado descritas en el Anexo A, para realizar pruebas de los descodificadores optimizados en entornos reales de difusión de televisión digital. En este anexo se describen todas las tareas que se han diseñado para poder disponer de este sistema completo (además del descodificador): el análisis de la trama de entrada, la descodificación del audio en diferentes formatos, la interfaz de usuario y la presentación del audio y el vídeo al usuario de forma sincronizada.

Finalmente, el Anexo C recopila todas las fichas que resumen las técnicas de optimización empleadas en las diferentes implementaciones explicadas en el capítulo 4.

La memoria concluye con la relación de referencias bibliográficas que se han consultado durante la elaboración de la tesis, junto con una lista de los acrónimos que aparecen en la memoria.

2 CODIFICACIÓN DE VÍDEO DIGITAL

Con objeto de hacer esta memoria autocontenida, en este capítulo se resumen los aspectos más relevantes relacionados con la codificación de vídeo digital incluyendo tanto las técnicas básicas que permiten la compresión de la información como los estándares que normalizan la trama de salida de los codificadores y el proceso de descodificación.

Para evitar duplicidades, en el apartado 2.1 se introducen los conceptos básicos que son comunes a todos los estándares y se establece parte de la nomenclatura que se emplea tanto en este, como en el resto de los capítulos. Seguidamente, en el apartado 2.2, se resumen los estándares de codificación que se han empleado durante el desarrollo de la tesis, explicando las herramientas que se utilizan en cada uno de ellos[1]. En el apartado 2.3 se comparan las herramientas empleadas en los diferentes estándares con objeto de poner de manifiesto las similitudes entre ellos. Finalmente, en el apartado 2.4 se resumen los aspectos más destacados que se han presentado en este capítulo.

2.1 Fundamentos de la codificación de vídeo digital

En este apartado se introducen una serie de conceptos empleados comúnmente en los estándares de codificación utilizados durante la elaboración de esta tesis: MPEG-2, MPEG-4 y H.264. Además, se establece parte de la nomenclatura que se emplea en el resto de capítulos que componen esta memoria.

2.1.1 Introducción: digitalización de la señal de vídeo

Desde que a finales del siglo XIX se realizara la primera proyección pública de imágenes en movimiento, la evolución que han sufrido las técnicas audiovisuales ha

[1] El objetivo es introducir algunos conceptos que se referenciarán en posteriores capítulos de la memoria. En ningún caso se pretende realizar un estudio exhaustivo puesto que en la actualidad existen numerosas publicaciones [PE02] [RR04], además de los propios estándares, en las que se explican los detalles de cada uno de ellos.

sido de tal magnitud que actualmente se han convertido en uno de los principales campos de investigación y desarrollo, tanto en el entorno universitario, como en el empresarial.

Los avances logrados han dado lugar a la aparición de múltiples campos de aplicación, tales como la videotelefonía, la videoconferencia, la televisión digital, el vídeo bajo demanda (VoD), la televisión móvil o la televisión a través de Internet (IPTV). Estos avances continúan en la actualidad, facilitando el nacimiento de nuevas aplicaciones como la televisión tridimensional o el vídeo escalable que probablemente se incorporarán, en un breve espacio de tiempo, a las que ya son ampliamente utilizadas.

Buena parte de estos logros han venido asociados a la digitalización de la señal de vídeo analógica y al desarrollo de mecanismos para su compresión que permiten su almacenamiento en los soportes actuales y su transmisión a través de los canales de comunicación existentes.

El formato de digitalización de la señal de televisión está estandarizado desde 1982 por parte de la Unión Internacional de Telecomunicaciones (UIT) mediante la Recomendación UIT-R BT.601-5 [ITU95], donde se define la estructura de muestreo que permite digitalizar la señal de vídeo en formatos PAL o NTSC. Este estándar establece para ambos sistemas una frecuencia de muestreo de 13.5 MHz[2], lo que en sistemas PAL (625 líneas, 25 imágenes/s) equivaldría a 864 muestras por línea, y en NTSC (525 líneas, 30 imágenes/s) a 858 muestras por línea. No obstante, la presencia de líneas ocultas y la existencia de impulsos de sincronismo horizontal dentro de cada línea, reducen la imagen de los formatos PAL y NTSC a un área útil de 720×576 y 720×480 píxeles, respectivamente. Con esta frecuencia de muestreo se obtiene una calidad de imagen que resulta aceptable respecto a la imagen original.

En este estándar se establece una representación de la señal basada en la utilización de tres componentes, una para la luminancia (Y) y dos para las diferencias de color o crominancias (C_R y C_B)[3]. Considerando que cada una de las componentes de la imagen se codifica con 8 bits, se obtiene un régimen binario de 248.8 Mbps[4].

La exploración de las líneas de una imagen fija puede llevarse a cabo de dos formas, lo que da lugar a dos tipos de barrido: el barrido progresivo, en el que todas las líneas de la imagen se barren consecutivamente, y el barrido entrelazado, en el que primero se barren las líneas impares de la imagen y a continuación se barren las líneas pares.

Actualmente, los diferentes estándares de codificación de vídeo utilizan, además del formato de muestreo UIT-R BT.601-5, otros formatos como los descritos en UIT BT.709-5 [ITU02] que se diferencian fundamentalmente en la resolución espacial de la imagen y en el número de imágenes por segundo. En la Tabla 2-1 se relacionan algunos de los formatos más utilizados, indicando el régimen binario que se requeriría para almacenar/transmitir una secuencia sin codificar con cada uno de ellos en función del número de imágenes por segundo.

---

[2] Se utiliza una frecuencia múltiplo entero de 2.25 MHz que es el mínimo común múltiplo de las que se emplean en los sistemas de 525 y 625 líneas.

[3] Existen otros formatos para representar los píxeles de la señal muestreada además del formado por la luminancia y las crominancias. El más conocido es el formato RGB en el que para cada muestra se almacena la información de los 3 colores (rojo, verde y azul).

[4] Este régimen binario se obtiene considerando el área útil de la imagen y viene dado por la expresión: (720×576) píxel/imagen × 25 imagen/segundo × 8 bits/píxel = 248.8 Mbps.

Formato	Resolución (píxeles)	Nº imágenes/segundo	Régimen Binario
UIT-T 709	1920×1080 (1080i)	60	1493 Mbps
UIT-T 709	1280×720 (720p)	30	663 Mbps
UIT-T 601	720×576 / 720×480	25 / 30	249 Mbps
16CIF	1408×1152	30	1167 Mbps
4CIF	704×576	30	292 Mbps
CIF	352×288	30	73 Mbps
QCIF	176×144	30	18 Mbps
Sub-QCIF	128×96	30	8.8 Mbps

Tabla 2-1. Formatos más utilizados en los sistemas de codificación de vídeo.

La Tabla 2-2 muestra una relación de los soportes de almacenamiento y los medios de transmisión disponibles en la actualidad indicando su capacidad aproximada. Como se puede comprobar el régimen binario que se obtiene cuando se digitaliza una secuencia de vídeo no es compatible, ni con los soportes de almacenamiento disponibles en la actualidad, ni con los medios de transmisión existentes. Este hecho es aún más notorio si se tienen en cuenta los nuevos formatos de imagen con los que se trabaja en la televisión de alta definición (1280×720 progresivo con 30 imágenes por segundo o 1920×1080 entrelazado con 60 imágenes por segundo). Para estas resoluciones se requieren regímenes binarios aún mayores si se desea disponer de las secuencias de vídeo sin comprimir.

Almacenamiento/Transmisión	Medio empleado	Capacidad
Almacenamiento	CD-ROM	720 *Mbytes*
	DVD	4.7 *Gbytes*
	Blu-Ray	50 *Gbytes*
Transmisión	Canal de TV digital	hasta 10 Mbps
	Red Local	hasta 1 Gbps
	ADSL	hasta 50 Mbps

Tabla 2-2. Capacidad de almacenamiento o transmisión de los medios disponibles en la actualidad.

Por tanto, se hace imprescindible reducir el régimen binario de la secuencia de vídeo digitalizada empleando algún proceso de compresión de la información. Actualmente las relaciones de compresión que se requieren sólo pueden ser obtenidas utilizando algoritmos en los que se asumen ciertas pérdidas.

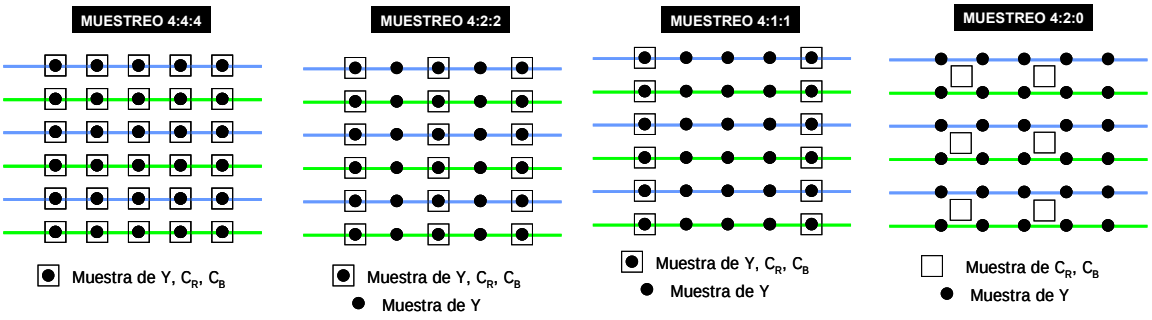
2.1.2 Fundamentos de la compresión de vídeo

En este subapartado se presentan los principios básicos en los que se apoyan las técnicas de compresión de vídeo que se emplean en los estándares de codificación actuales.

2.1.2.1 Muestreo de la señal de televisión

Las imágenes de una secuencia de vídeo se pueden digitalizar empleando diferentes esquemas de muestreo y diferente precisión en la representación de las muestras, en función de la aplicación a la que se vayan a destinar. En el esquema de muestreo 4:4:4 cada muestra o píxel de la imagen se representa con una componente de luminancia (Y) y dos de crominancia (C_R y C_B), como se muestra en la Figura 2-1-a. Sin embargo, y aprovechando que la visión humana es menos sensible a los matices de color que a la luz, es muy habitual que se realice un submuestreo de las crominancias con objeto de reducir la cantidad de información que hay que manejar sin que se produzca una pérdida apreciable de calidad.

Existen varios esquemas de muestreo en los que se reduce la cantidad de información de color; así, en el esquema 4:2:2 (Figura 2-1-b), las dos crominancias se submuestrean horizontalmente de forma que su resolución es la mitad que la de la luminancia. En los esquemas 4:1:1 (Figura 2-1-c) y 4:2:0 (Figura 2-1-d) las crominancias se submuestrean de forma que su resolución es una cuarta parte que la de la luminancia. La diferencia entre ambos esquemas es que, en el 4:1:1, se produce un submuestreo exclusivamente en el eje horizontal, manteniendo el mismo número de muestras en el eje vertical, mientras que, en el 4:2:0, cada crominancia representa la información de color correspondiente a la posición de las cuatro luminancias adyacentes por lo que el submuestreo se realiza tanto en el eje horizontal como en el vertical.



a) Muestreo 4:4:4 b) Muestreo 4:2:2 c) Muestreo 4:1:1 d) Muestreo 4:2:0

Figura 2-1. Esquemas de muestreo.

Estos esquemas de muestreo se utilizan en diferentes aplicaciones y con precisión de 8 ó 10 bits en la representación de Y, C_R y C_B. Finalmente, en los estándares MPEG-2, MPEG-4 y H.264 se utiliza la variante del esquema de muestreo 4:2:0 que se muestra en la Figura 2-2 con precisión de 8 bits.

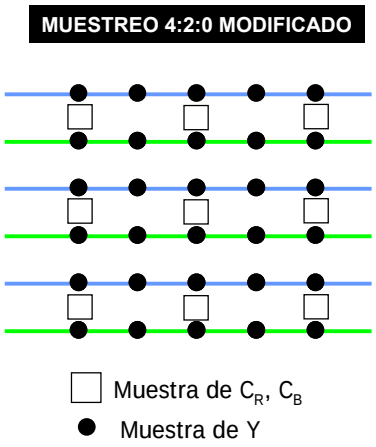


Figura 2-2. Esquema de muestreo 4:2:0 utilizado en los estándares MPEG-2, MPEG-4 y H.264.

2.1.2.2 Estructura de las imágenes

Con objeto de facilitar el procesamiento de cada imagen, en los estándares MPEG-2, MPEG-4 y H.264 no se trabaja con la información de la imagen a nivel de píxel. En estos estándares las imágenes son divididas en áreas cuadradas de tamaño 16×16 para la información de luminancia (Y) y de tamaño variable para la información de crominancia (C_B y C_R) dependiendo del esquema de muestreo que se haya empleado para la digitalización de las imágenes.

Las áreas de 16×16 píxeles se denominan macrobloques (MB). A su vez, cada macrobloque está dividido en áreas de 8×8 píxeles a las que se denomina bloques[5]. El número de bloques de luminancia siempre es 4, mientras que el número de bloques de las crominancias puede ser 8, 4 ó 2 en función del esquema de muestreo que se utilice (4:4:4, 4:2:2 ó 4:2:0, respectivamente). La Figura 2-3 muestra la estructura de un macrobloque, según el esquema de muestreo utilizado.

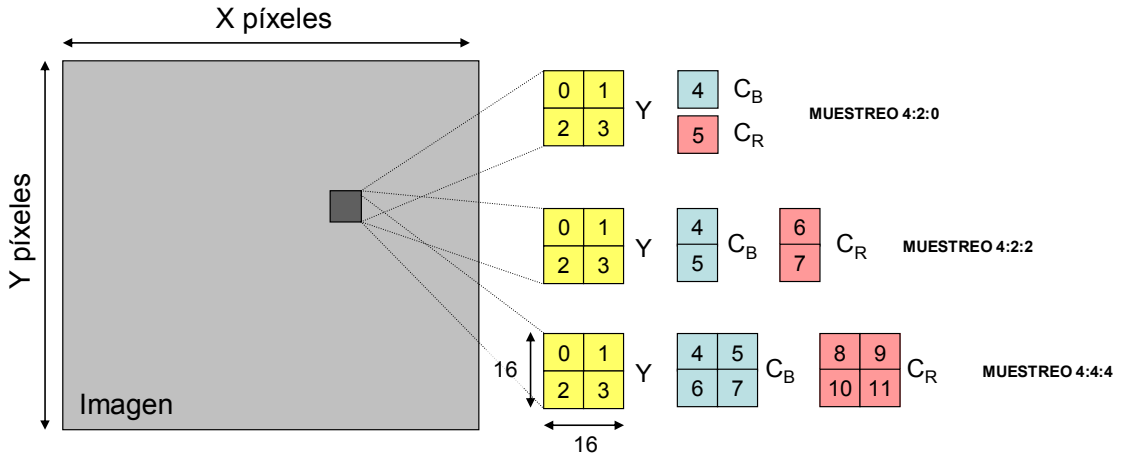


Figura 2-3. Estructura de un macrobloque para diferentes esquemas de muestreo.

[5] Como se verá en el apartado 2.2.4.2.4.1, en el estándar H.264 los bloques de 8×8 píxeles pueden estar a su vez divididos en sub-bloques de menor tamaño.

2.1.2.3 Redundancia estadística y de percepción

Las técnicas de compresión de vídeo aprovechan, tanto las características estadísticas inherentes a las secuencias de vídeo natural (redundancia estadística), como algunas limitaciones de visión que posee el ojo humano (redundancia de percepción) para reducir la información que tiene una secuencia de vídeo sin que se produzca una pérdida de calidad apreciable.

Analizando estadísticamente una secuencia de vídeo se observa una correlación, tanto entre los píxeles próximos entre sí de una imagen (redundancia espacial), como entre los píxeles que ocupan la misma posición o posiciones cercanas en imágenes consecutivas (redundancia temporal). El objetivo de las técnicas de compresión es reducir ambas redundancias.

La redundancia espacial conlleva que los píxeles que componen un bloque de una imagen suelen ser parecidos. Esta circunstancia es aprovechada realizando una transformación al dominio de la frecuencia, que permite compactar la información de la imagen en el entorno de las bajas frecuencias espaciales para que, posteriormente, los procesos de cuantificación y codificación de entropía, puedan reducir la cantidad de información que es necesario transmitir/almacenar tal y como se explica en el apartado 2.1.2.4.2.

Por otra parte, la redundancia temporal debida a la correlación temporal entre las imágenes de la secuencia de vídeo se reduce utilizando las técnicas de predicción o estimación de movimiento como se muestra en el apartado 2.1.2.4.5.

Además, el ojo humano tiene una serie de limitaciones que pueden ser aprovechadas para eliminar parte de la información de la secuencia de vídeo que no puede ser diferenciada por el espectador. Más concretamente, el ojo tiene una menor sensibilidad para distinguir matices de color, así como mayor dificultad para diferenciar píxeles próximos de valores muy diferentes dentro de una misma imagen. Para reducir la cantidad de información que posee una secuencia, se pueden eliminar, tanto algunos de los detalles de una imagen, como parte de la información de color empleando algunos de los esquemas de muestreo mostrados en el apartado 2.1.2.1.

2.1.2.4 Esquema de codificación híbrido

En los estándares de codificación MPEG-2, MPEG-4 y H.264 se emplea el llamado “esquema de codificación híbrido” para realizar la compresión de la señal de vídeo reduciendo la redundancia estadística y de percepción. Se trata de un esquema básico al que los diferentes estándares que han ido apareciendo han añadido variantes que permiten incrementar su capacidad de compresión.

2.1.2.4.1 Diagrama de bloques de un codificador híbrido

La Figura 2-4 muestra la parte del codificador híbrido que permite reducir la redundancia espacial. En este esquema, los bloques de píxeles en los que se encuentra dividida cada imagen de una secuencia de vídeo se transforman al dominio de la frecuencia obteniéndose una serie de coeficientes que representan las componentes de las diferentes frecuencias que contienen los píxeles del bloque.

Los coeficientes obtenidos son cuantificados con un doble objetivo, por un lado reducir su rango de variación para que puedan ser codificados con un menor número de bits y, por otro, eliminar los coeficientes que contienen la información de alta frecuencia que el ojo humano no es capaz de apreciar. Finalmente, los coeficientes cuantificados son procesados por un codificador de entropía que aprovecha las propiedades estadísticas de estos y asigna códigos de menor longitud a los elementos de la trama que son más probables, reduciendo de este modo el régimen binario de

salida. Este proceso lleva asociadas unas pérdidas debidas a la cuantificación ya que se elimina información de la señal transformada que posteriormente no puede ser recuperada.

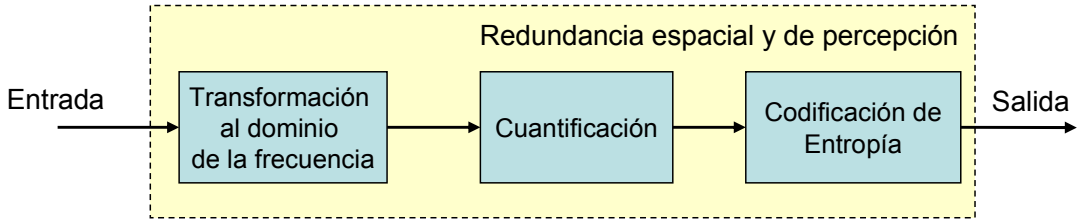


Figura 2-4. Reducción de la redundancia espacial y de percepción.

Para completar el codificador híbrido se añade al esquema anterior un compensador de movimiento que permite reducir la redundancia temporal. Para ello se almacenan imágenes previamente codificadas para buscar en ellas macrobloques lo más parecidos posible al de la imagen actual que se desea codificar. A partir de la calidad de la predicción, el codificador decide para cada macrobloque si emplea la predicción obtenida o si, por el contrario, codifica los píxeles directamente. Por tanto, el codificador posee en su entrada un selector que permite tomar esta decisión para cada macrobloque de la imagen de entrada

Si se ha localizado una buena predicción, se procede a codificar la diferencia entre dicho macrobloque y su predicción (error de predicción). Las muestras de error tienen valores pequeños por lo que al transformarlas al dominio de la frecuencia dan lugar a una matriz de coeficientes en la que muchos de ellos son cero y el resto suelen estar concentrados en la zona de las bajas frecuencias espaciales.

La Figura 2-5 muestra el diagrama de bloques simplificado de un codificador híbrido completo en el que, respecto al diagrama presentado en la Figura 2-4, se han añadido un cuantificador inverso, una transformación inversa para recuperar la imagen codificada y almacenarla en una memoria, así como un bloque de compensación y estimación de movimiento que realiza la búsqueda de las predicciones de los macrobloques entre las imágenes codificadas anteriormente. Estos bloques se añaden en un lazo de realimentación para que el estimador de movimiento realice las predicciones empleando las imágenes almacenadas en la memoria. Estas imágenes son las mismas que recuperará el descodificador, por lo que el compensador de movimiento que éste posee obtendrá los mismos píxeles que los empleados en la estimación de movimiento que se realiza en el codificador, evitando de este modo que el descodificador diverja.

Aquellos macrobloques que se codifican de forma independiente reciben el nombre de macrobloques *intra* (o macrobloques tipo I) mientras que los que emplean predicción de imágenes anteriores y/o posteriores son denominados *inter*. Los macrobloques de tipo *inter* se dividen a su vez en dos tipos: los que poseen una única referencia a una imagen anterior (o macrobloques tipo P) y los que poseen dos referencias, una a una imagen anterior y otra a una posterior (o macrobloques tipo B).

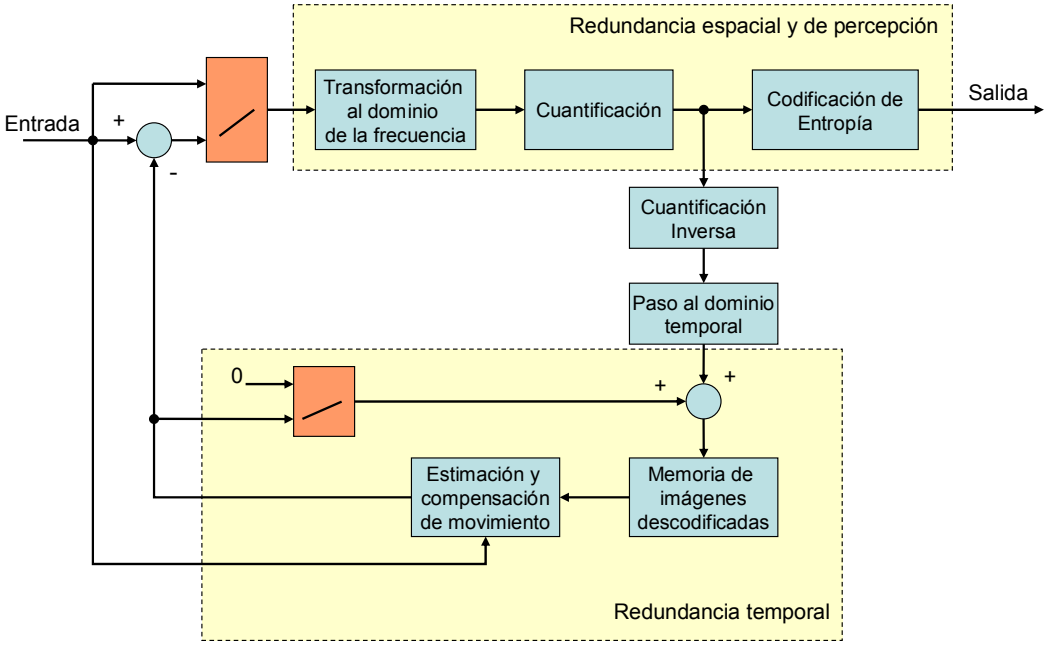


Figura 2-5. Esquema de codificación híbrido.

Se definen tres tipos de imagen en función del tipo de macrobloques que incluyen:

- Las imágenes tipo I, en las que todos sus macrobloques están codificados sin predicción (modo *intra*) por lo que no dependen de otras imágenes para su codificación. Estas imágenes suelen enviarse cada cierto tiempo dentro de una secuencia y facilitan el acceso a puntos intermedios de la misma. Por otra parte, la inclusión de imágenes tipo I permite evitar que los errores asociados al proceso de codificación se vayan acumulando y provoquen pérdidas significativas en la calidad de la secuencia.
- Las imágenes tipo P, en las que sus macrobloques pueden ser de tipo I, si no se ha localizado una predicción adecuada, o de tipo P si en una imagen de tipo I o P anterior se ha localizado una buena predicción. Este tipo de imágenes dependen exclusivamente de la imagen I o P inmediatamente anterior. La Figura 2-6 muestra un ejemplo de secuencia en la que hay imágenes de tipo I y de tipo P así como las relaciones entre ellas. Para que una imagen sea de tipo P es necesario que al menos uno de los macrobloques que la componen esté codificado como de tipo P y que ninguno sea de tipo B.

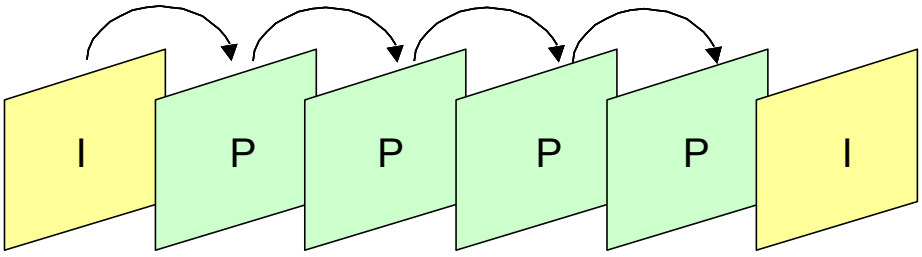


Figura 2-6. Secuencia que incluye imágenes de tipo I y P.

- Las imágenes tipo B, en las que sus macrobloques pueden ser de tipo I, P o B dependiendo de si el codificador ha localizado alguna predicción y si ésta se encuentra en las imágenes anterior (referencia *backward*) y/o posterior (referencia *forward*). Cada imagen de tipo B, independientemente del tipo de macrobloques de que esté compuesta, depende de imágenes I y/o P anteriores y/o posteriores tal y como se muestra en la Figura 2-7. Para que una imagen sea de tipo B es necesario que al menos uno de sus macrobloques esté codificado como de tipo B.

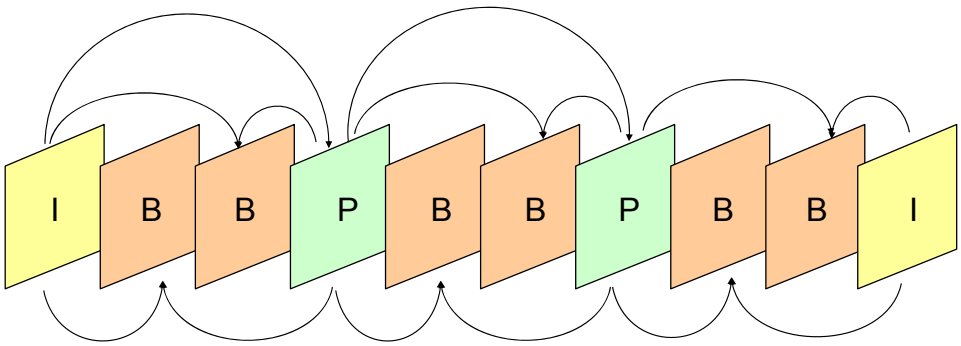


Figura 2-7. Secuencia que incluye imágenes de tipo I, P y B.

Para que sea posible codificar un macrobloque a partir de otros que se encuentran en imágenes posteriores (codificación bidireccional), es necesario que el orden en el que se codifican las imágenes no sea el mismo que el orden natural de las mismas. Esto implica que antes de codificar cualquier imagen de tipo B es necesario haber codificado las imágenes I y/o P de las que depende. La Figura 2-8 muestra una secuencia que ha sido codificada empleando imágenes de los tres tipos. En la parte superior se muestra el orden natural de la secuencia en la que la primera imagen se codifica como tipo I, la segunda y tercera como tipo B y la cuarta como tipo P. Para que esta codificación sea posible las imágenes tipo B requieren que la imagen P se haya codificado previamente. Por tanto, y como se muestra en la parte inferior de la figura, tras codificar la primera imagen (tipo I) se procede a codificar la cuarta (tipo P) y finalmente la segunda y tercera (tipo B).

Como se aprecia, el codificador debe guardar las dos imágenes que desea codificar como tipo B hasta que codifica la imagen tipo P. Por tanto, la necesidad de almacenamiento del codificador crece en función del número de imágenes tipo B consecutivas que se desee incluir en la trama de salida.

Orden natural	1	2	3	4	5	6	7	8	9	10
	I	B	B	P	B	B	P	B	B	I
Orden codificación	1	4	2	3	7	5	6	10	8	9
	I	P	B	B	P	B	B	I	B	B

Figura 2-8. Orden natural y de codificación de una secuencia que incluye imágenes tipo B.

Desde el punto de vista del descodificador, éste debe ir reconstruyendo las imágenes en el orden en el que las va recibiendo y reordenarlas para obtener de nuevo la secuencia original.

2.1.2.4.2 Transformación al dominio de la frecuencia

Las transformaciones que se emplean en los estándares de codificación de vídeo están basadas en transformadas del coseno bidimensionales que descomponen la señal de entrada en sumas ponderadas de cosenos de diferentes frecuencias espaciales.

La Transformada Discreta del Coseno bidimensional (*Discrete Cosine Transform* o DCT) aplicada a bloques de 8×8 píxeles es la que habitualmente se ha empleado en los estándares de codificación. La expresión de esta transformada normalizada para este tamaño de bloque se muestra en la Ec. 2-1 [RH96].

$$f(x,y)=\frac{C(u)C(v)}{4}\sum_{u=0}^{7}\sum_{v=0}^{7}F(x,y)\cos\left[\pi(2x+1)\frac{u}{16}\right]\cos\left[\pi(2y+1)\frac{v}{16}\right] \qquad \text{Ec. 2-1}$$

Donde:

x, *y*	son las coordenadas en el dominio espacial.
F(x, y)	es el valor que toma en (*x, y*) el píxel del bloque que se va a transformar.
u, *v*	son las coordenadas en el dominio de las frecuencias espaciales.
C(*u*)	$1/\sqrt{2}$ para *u* = 0, en el resto de casos 1.
C(*v*)	$1/\sqrt{2}$ para *v* = 0, en el resto de casos 1.

El resultado de aplicar esta transformación es un bloque de 64 coeficientes; el primero (coordenada 0, 0) es el coeficiente de continua o coeficiente DC y el resto son coeficientes correspondientes a frecuencias espaciales crecientes (coeficientes AC). La DCT no produce por tanto una compresión de la información, dado que genera el mismo número de coeficientes que píxeles tiene el bloque de entrada; es más, generalmente se produce un aumento en el número de bits con el que se codifica cada bloque debido a la mayor precisión con la que hay que representar los coeficientes en el dominio de la frecuencia. La ventaja que aporta es la concentración de la energía de cada bloque en el dominio espacial alrededor de las bajas frecuencias del dominio transformado[6], lo que es aprovechado por las siguientes etapas del proceso de codificación.

Idealmente se trata de un proceso sin pérdidas, sin embargo, esto no es cierto en la práctica debido a la precisión finita con la que se realizan los cálculos. Los diversos estándares indican la precisión con la que deben realizarse las operaciones para que las pérdidas no sean apreciables.

La expresión matricial de la DCT, para un bloque de 4×4[7], se muestra en la Ec. 2-2.

[6] Esto es tanto más cierto, cuanto más elevada sea la correlación entre los píxeles del bloque de entrada. Llevado al extremo, en el caso de un bloque en el que todos sus píxeles son iguales, el resultado de la DCT es un único coeficiente diferente de cero que se correspondería con la componente continua. El resto de los coeficientes en este caso serían cero.

[7] Se ha empleado este tamaño de bloque en lugar de un bloque de 8×8 para simplificar las expresiones de las ecuaciones.

$$Y = AXA^T = \begin{pmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & c \end{pmatrix} \cdot \begin{pmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{pmatrix} \cdot \begin{pmatrix} a & b & a & c \\ a & c & -a & -b \\ a & -c & -a & b \\ a & -b & a & c \end{pmatrix} \quad \text{Ec. 2-2}$$

Donde:

Y: matriz de los coeficientes tras aplicar la DCT a los píxeles de entrada.

$a = \frac{1}{2}$, $b = \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) = -0.653$ y $c = \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) = 0.271$

x_{ij}*:* son los píxeles en el dominio espacial.

Finalmente la transformada inversa del coseno (IDCT) se obtiene a partir de la expresión mostrada en la Ec. 2-3 para su representación espacial y en la Ec. 2-4 para su expresión matricial. Los parámetros que aparecen en la expresión de la IDCT tienen el mismo significado que los presentados para la DCT.

$$F(x,y) = \frac{1}{4}\sum_{u=0}^{7}\sum_{v=0}^{7} C(u)C(v)f(u,v)\cos\left[\pi(2x+1)\frac{u}{16}\right]\cos\left[\pi(2y+1)\frac{v}{16}\right] \quad \text{Ec. 2-3}$$

$$X = A^T Y A \quad \text{Ec. 2-4}$$

2.1.2.4.3 Cuantificación

El proceso de cuantificación permite al codificador reducir el número de niveles con los que se representan los coeficientes transformados en el dominio de la frecuencia. Para ello se emplea una función de cuantificación que asigna un determinado código a cada dato que se encuentra en un rango de entrada. La Figura 2-9 representa la función de transferencia de un cuantificador lineal uniforme en el que el rango de la entrada para el que se asigna un determinado código de salida es constante en todo el margen dinámico de la entrada. Así, a cualquier valor de la entrada que se encuentre entre i_1 e i_2 se le asignará el código de salida O_1. Se denomina "paso de cuantificación" a la distancia $|i_{n+1} - i_n|$.

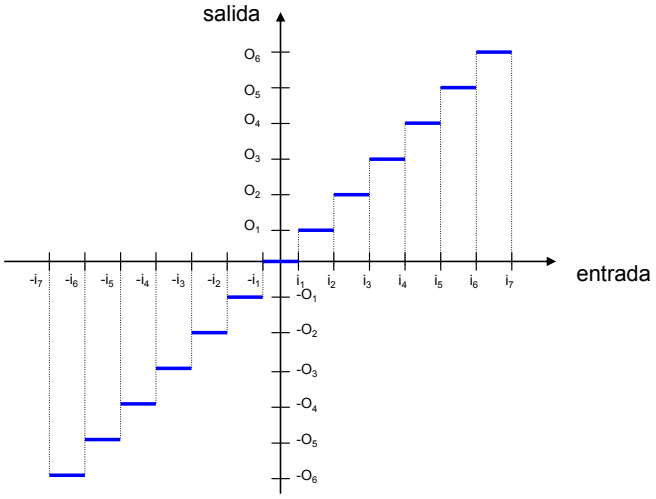


Figura 2-9. Función de transferencia de un cuantificador lineal.

Cuanto mayor sea el número de niveles, menor será la pérdida de información que se producirá en el proceso de cuantificación y mayor el número de bits necesarios para realizar su codificación. Por el contrario, si se emplean pocos códigos de salida, el ahorro de bits será mayor pero las pérdidas serán también mayores.

Es muy habitual que los coeficientes de alta frecuencia tengan valores que se encuentran en el primer escalón de cuantificación (el que se encuentra entre $-i_1$ e i_1 en la Figura 2-9) por lo que se les asigna el código 0, facilitando su transmisión o almacenamiento, como se verá en el apartado siguiente.

2.1.2.4.4 Codificación de entropía

Sobre los coeficientes de salida que entrega el cuantificador, así como sobre algunos elementos sintácticos de la trama, se aplica una codificación de entropía. Los diversos estándares emplean codificación de longitud variable (*Variable Length Coding* o VLC), en la que se asigna un menor número de bits a aquellos códigos más probables, reduciendo de este modo el régimen binario de salida del codificador.

En cada estándar se definen las tablas que deben emplearse para asignar los códigos de longitud variable a cada uno de los elementos sintácticos que componen la trama de bits de salida del codificador. Estas tablas pueden ser estáticas o modificarse dinámicamente durante el proceso de codificación (codificación adaptativa).

Los estándares MPEG-2, MPEG-4 y H.264[8] emplean variantes de la codificación VLC. A modo de ejemplo, en MPEG-2 se utiliza un algoritmo RLC (*Run Length Coding*) para codificar los coeficientes obtenidos a la salida del cuantificador. Este algoritmo se realiza en tres pasos:

1. Ordenación de los coeficientes. Los coeficientes cuantificados son ordenados de menor a mayor frecuencia siguiendo un determinado patrón. En la mayoría de los casos se emplea el denominado *Zig-Zag Scan*[9].
2. Codificación *last, run, level*. Los coeficientes son leídos de menor a mayor frecuencia empleando el patrón seleccionado. En este barrido, a cada coeficiente distinto de cero se le asigna un *level* igual a su valor y un *run* igual al número de ceros que le preceden. Este proceso se repite para llegar al último coeficiente distinto de cero. A este último coeficiente distinto de cero se le asigna un valor de *last* igual a 1 mientras que para el resto de coeficientes *last* permanece siempre a 0.
3. Asignación de un código. Dependiendo de la probabilidad de cada combinación[10], se asigna un código binario a cada terna de valores *last, run, level*.

2.1.2.4.5 Estimación y compensación de movimiento

El objetivo de este proceso es realizar una predicción de los macrobloques de una imagen (macrobloque de referencia) a partir de imágenes anteriores y/o posteriores. Para cada macrobloque que se desea codificar se realiza una búsqueda en posiciones próximas a la posición que ocupa ese bloque en imágenes previamente

[8] En el caso de estándar H.264 es posible emplear una codificación aritmética como se explicará en el apartado 2.2.4.2.3.2.

[9] Los estándares de codificación de vídeo definen otros patrones para la lectura de los coeficientes cuantificados como puede ser el *Alternate Scan* o el *Alternate Horizontal Scan*.

[10] Estas probabilidades son estáticas y se han obtenido experimentalmente tras el análisis de numerosas secuencias de vídeo. Cada estándar incluye una relación de tablas de asignación de códigos a las diferentes combinaciones de los elementos sintácticos a codificar.

codificadas (área de búsqueda[11]). En la Figura 2-10 se muestra un ejemplo de la búsqueda de un macrobloque de la "imagen n" en un área próxima a la ubicación de dicho macrobloque en una "imagen anterior (n-m)[12]". En el caso de que se emplee una imagen posterior para realizar la estimación, el esquema es el mismo sustituyendo la "imagen n-m" por la "imagen n+m".

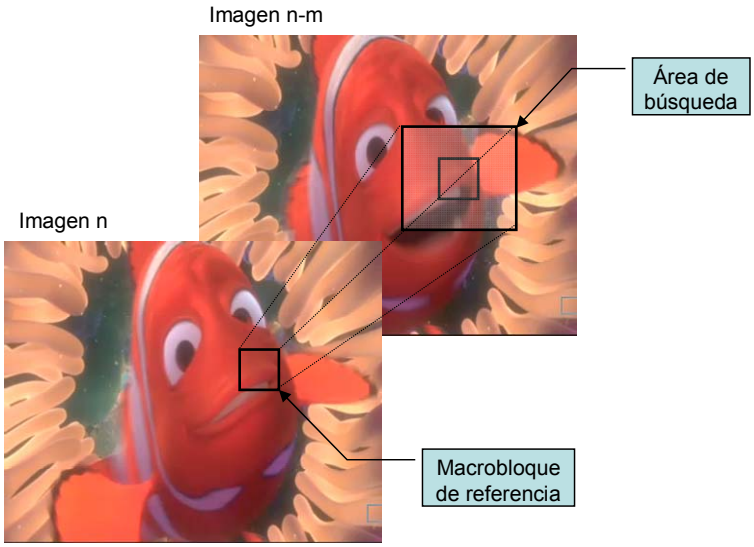


Figura 2-10. Estimación de movimiento basada en una imagen anterior.

El objetivo de esta búsqueda es localizar el macrobloque dentro de esta área que minimiza una función de error. Normalmente esta función de error es la Suma de Diferencias Absolutas (*Sum of Absolute Differences* o SAD) que se presenta en la Ec. 2-5.

$$SAD(i,j) = \sum_{m=1}^{N}\sum_{n=1}^{N}\left|x(m,n) - x_A(m+i,n+j)\right| \qquad \text{Ec. 2-5}$$

$$p_1 \le i \le p_2 \qquad p_1 \le j \le p_2$$

Donde p_1 y p_2 son los límites del área de búsqueda, $x(m,n)$ son los píxeles del macrobloque que se desea codificar y $x_A(m+i,n+j)$ son los píxeles del área de búsqueda de una imagen anterior o posterior.

La predicción obtenida en el proceso de estimación de movimiento puede ser utilizada por el codificador de forma que éste codifica la diferencia entre el macrobloque de referencia y la predicción. El codificador obtiene además un vector de movimiento que indica, dentro del fotograma n-m, el desplazamiento del macrobloque que minimiza la función SAD con respecto al que ocupa una posición homóloga al macrobloque que se está codificando en el fotograma n. Este desplazamiento se puede obtener con precisión entera (1 píxel) o fraccionaria (1/N píxel) dependiendo del estándar de codificación empleado, como se indicará en los apartados 2.2.2 para MPEG-2, 2.2.3 para MPEG-4 y 2.2.4 para H.264.

[11] El tamaño máximo del área de búsqueda se encuentra definido en cada estándar en función del perfil y del nivel seleccionados.

[12] Para MPEG-2 y para MPEG-4, m=1; sin embargo, para H.264, este valor puede ser mayor (ver apartado 2.2.4.2.4.4).

El vector de movimiento se incluye en la trama de bits de salida del codificador y es empleado por el descodificador para obtener, a partir de las imágenes previamente descodificadas, la predicción del macrobloque a descodificar.

2.1.2.5 Estructura del descodificador

La secuencia de bits generada por un codificador de vídeo debe ser procesada por un descodificador para obtener las imágenes reconstruidas con las pérdidas que haya provocado el proceso de codificación. El diagrama de bloques simplificado de un descodificador se presenta en la Figura 2-11. Para cada macrobloque se realiza la descodificación de entropía, la cuantificación inversa y la transformación al dominio espacial. Si se trata de un macrobloque codificado en modo *intra*, los datos obtenidos son directamente los píxeles que deben almacenarse en la memoria de imágenes descodificadas. Si por el contrario se está procesando un macrobloque de tipo *inter*, los datos recuperados se corresponden con el vector de movimiento y el error de predicción por lo que el bloque de compensación de movimiento debe obtener la predicción a partir de las imágenes descodificadas anteriormente para ser sumada con el error antes de almacenar el resultado en la memoria.

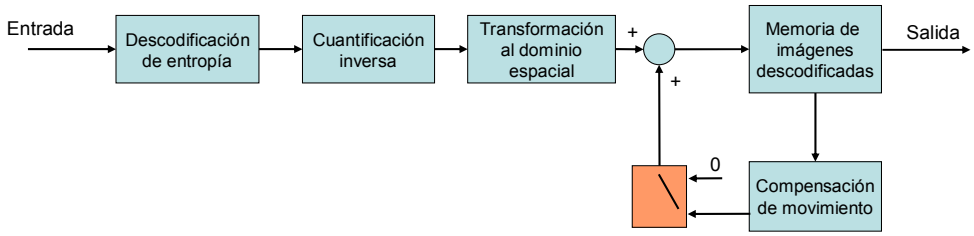


Figura 2-11. Diagrama de bloques del descodificador.

2.2 Estándares de codificación de vídeo

En este apartado se realiza una exposición de los aspectos más relevantes de los estándares de codificación de vídeo empleados en esta tesis (MPEG-2, MPEG-4 y H.264). El objetivo es mostrar las similitudes que permiten reutilizar herramientas comunes a varios de ellos, facilitando que las técnicas de reducción del tiempo de ejecución que se apliquen a alguno de ellos sean válidas para el resto.

2.2.1 Introducción a los estándares de codificación de vídeo

Durante los últimos 25 años, dos organismos internacionales de estandarización como son la ITU (*International Telecommunication Union*) y la ISO (*International Organization for Standardization*) han elaborado sucesivamente estándares basados en el codificador híbrido que tienen como objetivo reducir el régimen binario de salida manteniendo la calidad de la imagen. Así la ITU ha desarrollado sucesivamente los estándares H.261 (1984[13]) y H.263 (1998) mientras que la ISO ha creado en paralelo los estándares MPEG-1 (1990), MPEG-2 (1995) y MPEG-4 (1999). Posteriormente ambos organismos crearon un grupo de trabajo conjunto para desarrollar un nuevo estándar denominado MPEG-4 Parte 10 (2003) en la terminología ISO y H.264 en la terminología ITU.

[13] Las fechas que se han asociado a cada estándar indican el momento en el que se presentó la primera versión del mismo. Es muy habitual que con posterioridad se añadan anexos y se realicen enmiendas (*amendment*) o correcciones (*corrigenda*).

Recientemente ambos organismos han seguido colaborando en la ampliación del estándar H.264 para incluir la codificación escalable (*Scalable Video Coding* o SVC) que ha sido publicada en 2007 [ITU07] y la codificación multivista (*Multiview Video Coding* o MVC) estandarizada en 2009 [ITU09].

El número de estándares de codificación de vídeo se ha incrementado durante los últimos años con la aparición de los estándares desarrollados por el SMPTE (*Society of Motion Picture and Television Engineers*), más concretamente los denominados VC-1, VC-2 y VC-3 que han sido adoptados por empresas de gran relevancia a nivel mundial como *Microsoft*. El desarrollo temporal de todos estos estándares se muestra en la Figura 2-12.

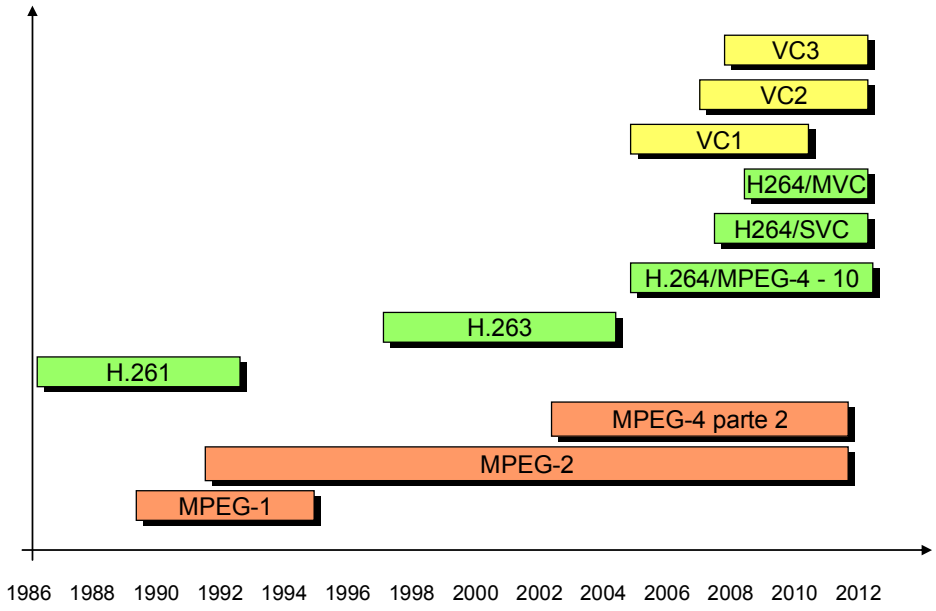


Figura 2-12. Evolución histórica de los estándares de codificación de vídeo.

A la vista de la anterior figura, se puede comprobar que el panorama actual de la descodificación de vídeo digital es complejo puesto que conviven en el mercado numerosas normas. Además, es más que previsible que este hecho se acentúe en los próximos años ya que los organismos internacionales de estandarización tardan cada vez menos tiempo en generar nuevos estándares.

La mayoría de los estándares de codificación de vídeo especifican cómo debe ser la trama de salida de un codificador, pero no cómo debe realizarse el codificador puesto que este aspecto queda sujeto a la competencia de las empresas para lograr un codificador que obtenga la mayor calidad posible reduciendo el régimen binario. El hecho de que la trama de salida esté definida implica, de algún modo, que se está especificando el descodificador puesto que éste debe ser capaz de interpretar adecuadamente cualquier trama que reciba.

Los estándares definen una serie de perfiles y niveles que permiten diseñar codificadores y descodificadores compatibles con un subconjunto de la especificación y adaptados al tipo de aplicación en el que se desean usar. Los perfiles representan subconjuntos de la sintaxis de la trama mientras que los niveles indican los márgenes de valores de algunos parámetros utilizados en el proceso de codificación.

En los apartados posteriores se resumen los estándares analizados durante el desarrollo de esta tesis mostrando las diferencias que presentan respecto al codificador híbrido explicado en el apartado 2.1.2.4.1.

2.2.2 El estándar MPEG-2

MPEG-2 es el nombre por el que popularmente se conoce al estándar ISO/IEC 13818 [ISO00], aprobado en su primera versión en 1995 y modificado posteriormente en varias ocasiones hasta el año 2007 en el que se publicó su última versión[14].

2.2.2.1 Introducción al estándar MPEG-2

Aunque se suele hablar de MPEG-2 como estándar de codificación de vídeo, la norma está dividida en 10 partes[15] de modo que en ella no sólo se especifica la codificación del vídeo sino que, entre otros aspectos, también se normaliza la codificación del audio, el sistema de transporte y la sincronización del audio y el vídeo. Las partes más relevantes que componen la norma así como una breve descripción del contenido de las mismas se relacionan a continuación:

- Parte 1: Sistema. Describe cómo combinar varias tramas elementales (*Elementary Streams* o ES) de audio, vídeo u otros datos en una única trama en la que además se añade información de control y sincronización entre las tramas elementales.
- Parte 2: Vídeo. Describe la sintaxis de la trama de salida de un codificador de vídeo y el modo en el que debe realizarse el descodificador.
- Parte 3: Audio. Describe la sintaxis de la trama de salida de un codificador de audio y el modo en el que debe realizarse el descodificador.
- Parte 4: Test de conformidad. Define un conjunto de pruebas que deben realizarse sobre un descodificador de vídeo para garantizar que es compatible con la norma. Esta parte incluye numerosas secuencias de test que deben ser probadas en los descodificadores.
- Parte 5: *Software* de referencia. Incluye una implementación, tanto del codificador, como del descodificador de vídeo en lenguaje C que puede servir como punto de partida para el desarrollo de aplicaciones.
- Parte 6: Extensión para la gestión de contenidos (*Digital Storage Media Command and Control* o DSM-CC). Especifica un conjunto de protocolos que incluyen funciones de control para gestionar tramas MPEG-2.
- Parte 7: Codificación de audio avanzada (*Advanced Audio Coding* o AAC). Define un algoritmo de codificación de audio más eficiente que el descrito en la parte 3 del estándar y que posteriormente fue mejorado en MPEG-4.
- Parte 9: Extensión para funcionamiento en tiempo real (*Real Time Interface* o RTI). Describe la interfaz que deben implementar los descodificadores para aceptar tramas de transporte recibidas a través de diferentes redes.
- Parte 10: Extensión de conformidad para DSM-CC. Define un conjunto de pruebas para garantizar la compatibilidad de un descodificador con la parte 6 del estándar.
- Parte 11: Gestión y protección de la propiedad intelectual (*Intellectual Property Management and Protection* o IPMP) en sistemas MPEG-2. Define

[14] Con posterioridad se han seguido publicado correcciones y adendas.

[15] La parte 8 de la norma se planificó para la codificación de vídeo empleando 10 bits para representar cada muestra, sin embargo, la falta de interés industrial hizo que no se finalizara el proceso de estandarización.

una extensión de la norma basada en MPEG-4 para incluir información en las tramas sobre los derechos de uso de las mismas así como protección de los contenidos.

La parte 2 de la norma especifica la sintaxis de la trama de vídeo y en ella se definen los perfiles y niveles cuyas características principales son las que se muestran en la Tabla 2-3.

Perfil	Nivel	Anchura	Altura	Imágenes/ segundo	Régimen binario máximo (Mbps)
Simple	Principal	720	576	30	15
Principal	Bajo	352	288	30	4
	Principal	720	576	30	15
	Alto 1440	1440	1088	60	60
	Alto	1920	1088	60	80
Escalable SNR	Bajo	352	288	30	4
	Principal	720	576	30	15
Escalable espacial	Alto 1440	720 (1440)	576 (1088)	30 (60)	60
Alto	Principal	352 (720)	288 (576)	30	20
	Alto 1440	720 (1440)	576 (1088)	30 (60)	80
	Alto	960 (1920)	576 (1088)	30 (60)	100
4:2:2	Principal	720	608	30	-
	Alto	1920	1088	60	-
Multivisión	Bajo	352 (352)	288 (288)	30 (30)	8
	Principal	720 (720)	576 (576)	30 (30)	25
	Alto 1440	1440 (1440)	1088 (1088)	60 (60)	100
	Alto	1920 (1920)	1088 (1088)	60 (60)	130

Tabla 2-3. Perfiles y niveles definidos en el estándar MPEG-2 vídeo.

2.2.2.2 MPEG-2 Vídeo (ISO/IEC 13818-2)

De los perfiles y niveles disponibles en la norma, el binomio que se emplea típicamente en la difusión de televisión digital es el formado por el nivel principal y el perfil principal (lo que se suele denominar MP@ML). Por tanto, los trabajos de optimización que se han realizado durante el desarrollo de la tesis para este estándar se han centrado en este perfil y nivel.

La estructura de un codificador MPEG-2 conforme al perfil principal y al nivel principal se basa en el esquema de codificación híbrido mostrado en la Figura 2-5. A continuación se resumen las particularidades de los bloques de dicho esquema.

2.2.2.2.1 Características generales

Para el perfil y nivel seleccionados es posible trabajar con imágenes, tanto en formato progresivo, como en formato entrelazado empleando en todos los casos el esquema de muestreo 4:2:0 que se presentó en la Figura 2-2. El codificador puede emplear imágenes tipo I, P y B en el proceso de compresión, no estando limitada la distancia entre imágenes I, ni el número de imágenes B entre imágenes I o P.

Las imágenes se dividen en rebanadas (*slices*) compuestas por un número variable de macrobloques consecutivos con la única limitación de que todos ellos tienen que pertenecer a una misma fila de macrobloques de la imagen.

2.2.2.2.2 Transformación al dominio de la frecuencia

En MPEG-2 se define la transformada inversa a utilizar como la transformada discreta del coseno inversa (IDCT) expresada en la Ec. 2-3. Los bloques de 8×8 coeficientes de entrada de la IDCT están codificados con 12 bits y el resultado debe poder representarse con 9 bits. Aunque no se define explícitamente la transformada directa a emplear en el codificador, ésta debe ser la transformada discreta del coseno (DCT) que se presentó en la Ec. 2-1, en la que las muestras de entrada se representan con 9 bits y los coeficientes de salida con 12 bits [IEEE98].

Para imágenes con estructura de campo, la DCT siempre se realiza empleando exclusivamente píxeles de uno de los campos; mientras que para imágenes con estructura de cuadro, la DCT puede realizarse sobre bloques con formato de campo o de cuadro.

2.2.2.2.3 Cuantificación

El cuantificador que se utiliza es diferente para la componente continua de los bloques *intra*, para el resto de coeficientes *intra* y para los coeficientes *inter*. Además, existe la posibilidad de emplear una de las dos matrices de cuantificación incluidas en el estándar, que definen un factor de cuantificación diferenciado para cada coeficiente.

2.2.2.2.4 Codificación de entropía

La codificación de entropía se realiza empleando códigos de longitud variable para codificar todos los elementos sintácticos de la trama de salida del codificador. En el caso particular de los coeficientes cuantificados de la DCT, antes de asignarles un código, se realiza una reordenación de los mismos empleando el patrón *Zig-Zag Scan* para bloques de 8×8 píxeles o usando el denominado *Alternate Scan*. La codificación de la componente DC de los macrobloques *intra* se puede realizar diferencialmente (con predicción) respecto a la componente DC del último macrobloque codificado. De igual forma, los vectores de movimiento se codifican con predicción respecto a los vectores del macrobloque codificado anteriormente.

2.2.2.2.5 Estimación y compensación de movimiento

La compensación de movimiento se realiza sobre macrobloques de 16×16 píxeles de luminancia que pueden tener estructura de campo o de cuadro[16]. Los vectores de movimiento que se obtienen para un macrobloque de luminancia se emplean también para ambas crominancias.

Los vectores de movimiento que obtiene el estimador pueden tener resolución de ½ píxel. Esto implica que los vectores apuntan a píxeles intermedios entre los píxeles de la imagen. Dado que estos píxeles intermedios no existen en las imágenes de referencia, es necesario calcularlos mediante interpolación. En la Figura 2-13 se muestran en color oscuro los píxeles con resolución entera que se obtienen de la imagen de referencia y, con color claro, los que hay que obtener para poder disponer de un vector de movimiento con resolución de ½ píxel.

En el caso del estándar MPEG-2, los puntos intermedios se calculan promediando los píxeles con resolución entera; así, el píxel ‘b’ que se muestra en la Figura 2-13 se obtiene promediando los píxeles ‘B’ y ‘E’ y el píxel ‘a’ promediando los píxeles ‘A’, ‘B’, ‘D’ y ‘E’.

[16] La existencia de macrobloques de tipo P y B, junto con la posibilidad de realizar la predicción con macrobloques con estructura de cuadro o de campo provoca que a cada macrobloque con predicción se le pueda asociar, desde un único vector de movimiento (macrobloque tipo P con formato de cuadro), hasta cuatro (macrobloque tipo B con formato de campo).

El procedimiento de cálculo del vector de movimiento no está estandarizado aunque lo más habitual es que se realice una búsqueda en ubicaciones que emplean píxeles con resolución entera para posteriormente realizar un refinamiento de dicha búsqueda alrededor del vector de movimiento extraído previamente. Si en la Figura 2-13 se supone que el estimador de movimiento ha obtenido un vector que apunta al píxel 'E', es necesario interpolar todos los puntos intermedios de las ubicaciones alrededor de la obtenida anteriormente ('a', 'b', 'c', 'd', 'e', 'f', 'g' y 'h' que representan la esquina superior izquierda de cada ubicación) con resolución de ½ píxel y realizar de nuevo el cálculo de la SAD para estas 8 ubicaciones. Si la SAD obtenida para alguna de estas ubicaciones es menor que la calculada para la referencia con resolución entera, dicha ubicación se convierte en el bloque de referencia a partir del que se extrae el vector de movimiento.

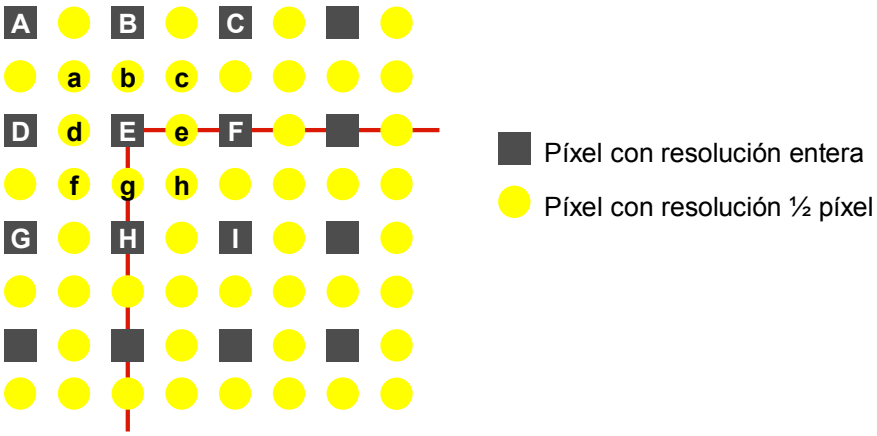


Figura 2-13. Píxeles con precisión de ½ píxel calculados a partir de píxeles con resolución entera.

2.2.2.2.6 La trama MPEG-2

Como se explicó anteriormente, el estándar MPEG-2 define la trama de bits de salida que debe generar un codificador y cuál es el proceso de descodificación. La trama de bits que proporciona un codificador incluye las imágenes comprimidas e información de sincronización y señalización para el descodificador. La trama posee una estructura jerárquica con diversas cabeceras que deben ser procesadas antes de descodificar cada imagen, tal y como se muestra en la Figura 2-14.

La trama comienza con una cabecera de secuencia que es idéntica a la que produciría un codificador MPEG-1. Seguidamente se añade una cabecera de extensión de secuencia que es la que diferencia a una trama MPEG-2 de otra MPEG-1. Tras la extensión de secuencia es posible que aparezcan datos asociados a dicha extensión y/o datos de usuario. Ambos tipos de datos son opcionales. A continuación puede aparecer una cabecera de grupo de imágenes (GOP) que permite agrupar conjuntos de imágenes en un nivel jerárquico superior que puede ser útil para la edición de vídeo y cuya inclusión no es obligatoria por parte del codificador. Tras esta, y también opcionalmente, se puede añadir una serie de datos de usuario.

Finalmente, para cada imagen el codificador incluye una cabecera de imagen, una extensión de codificación de imagen, opcionalmente datos de extensión y usuario y los datos propiamente dichos organizados en rebanadas. A su vez, cada rebanada está compuesta por macrobloques con su correspondiente cabecera y los datos asociados entre los que se encuentran los vectores de movimiento o los coeficientes de la DCT.

Después de la primera imagen se suceden secuencialmente las siguientes hasta llegar a una cabecera de fin de secuencia.

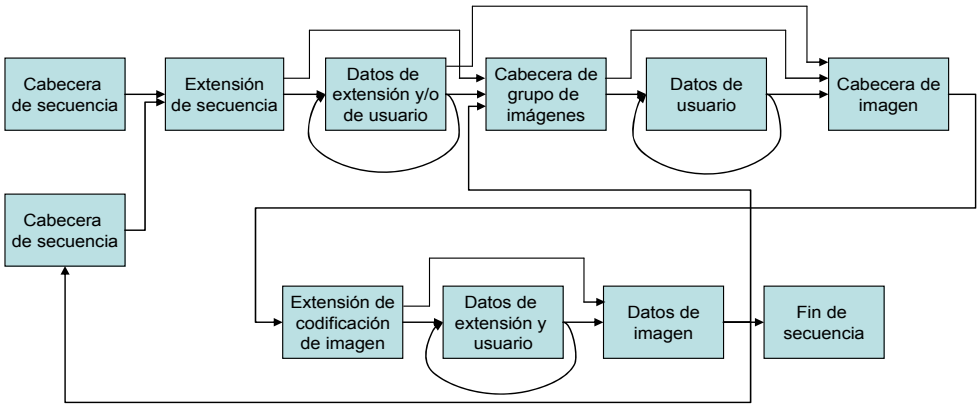


Figura 2-14. Organización de alto nivel de la trama generada por un codificador MPEG-2.

La Figura 2-15 muestra la estructura jerárquica de la trama MPEG-2 en la que se aprecia la información contenida en la trama desde el nivel más alto de la secuencia, hasta la información asociada a cada uno de los macrobloques que componen una imagen[17].

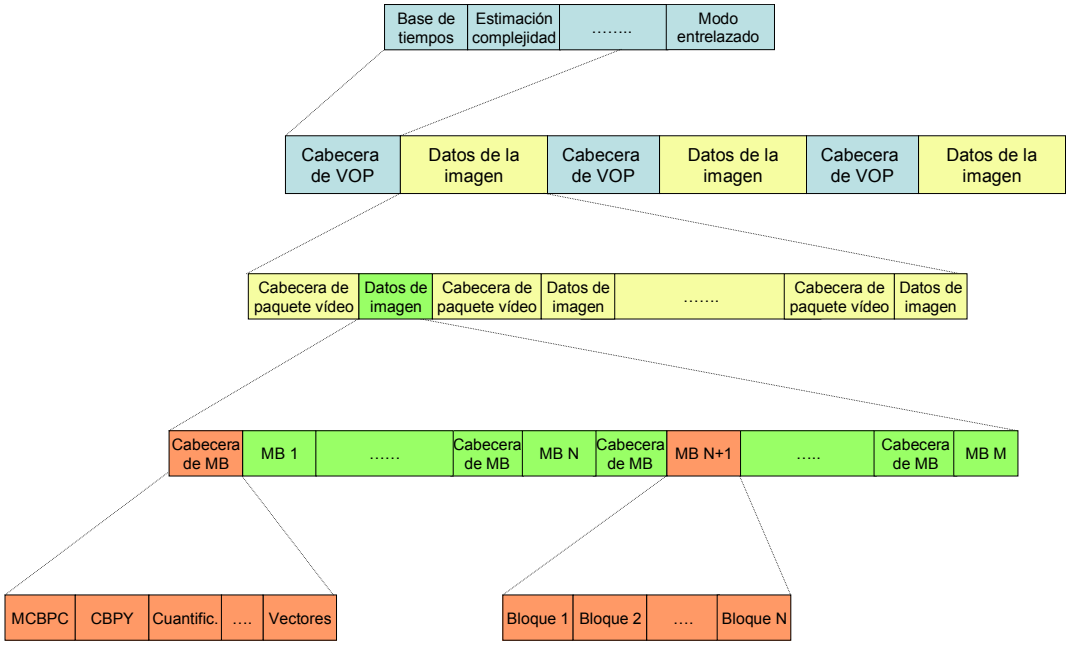


Figura 2-15. Estructura jerárquica de la trama generada por un codificador MPEG-2.

2.2.3 El estándar MPEG-4

MPEG-4 es el nombre convencional que se da al estándar de codificación ISO/IEC 14496 [ISO04b] que fue aprobado en su primera versión en 1999. El estándar está orientado al diseño de aplicaciones multimedia basadas en objetos, de tal modo que cada uno de los elementos que componen la secuencia es tratado como un objeto independiente. El ámbito de utilización de esta norma es muy amplio, permitiendo todo tipo de aplicaciones en las que, por ejemplo, es posible mezclar tramas de vídeo natural con imágenes sintéticas en una misma secuencia de vídeo.

[17] Para facilitar la lectura de la figura sólo se han incluido aquellos campos de las cabeceras que resultan más relevantes.

2.2.3.1 Introducción al estándar MPEG-4

En su primera versión (1999), el estándar estaba compuesto de 9 partes a las que con posterioridad se han ido añadiendo otras 16. A continuación se describen brevemente las partes que se consideran más significativas[18]:

- Parte 1. Sistema. Describe los mecanismos de sincronización y multiplexación de las secuencias de audio y vídeo.
- Parte 2. Vídeo. Describe la codificación de datos de vídeo incluyendo vídeo natural, vídeo sintético, vídeo de tamaño irregular, etc.
- Parte 3. Audio. Caracteriza las herramientas de codificación de audio incluyendo algunas variaciones sobre la codificación AAC descrita en MPEG-2. Además, se añade la codificación de audio sintético y la síntesis de voz.
- Parte 4. Conformidad. Define un conjunto de pruebas para verificar la compatibilidad de los descodificadores con la norma. Al igual que en MPEG-2 esta parte de la norma incluye numerosas secuencias de test que deben ser probadas en los descodificadores.
- Parte 5. *Software* de referencia. Incluye el código en lenguaje C que implementa, tanto un codificador, como un descodificador y que puede ser el punto de partida para el desarrollo de aplicaciones.
- Parte 10. Codificación de vídeo avanzado (*Advanced Video Coding* o AVC). Describe un mecanismo avanzado de codificación de vídeo natural rectangular con gran eficiencia de codificación (una descripción más amplia de esta parte del estándar se puede encontrar en el apartado 2.2.4[19]).
- Parte 11. Descripción de escenas (*BInary Format for Scenes* o BIFS). Presenta el mecanismo de descripción de los contenidos multimedia que se pueden incluir en una escena o secuencia de vídeo.
- Parte 13. Gestión y protección de la propiedad intelectual (*Intellectual Property Management and Protection* o IPMP). Proporciona mecanismos para proteger los contenidos descritos en una escena multimedia MPEG-4.

2.2.3.2 MPEG-4 Vídeo (ISO/IEC 14496-2)

La parte 2 de la norma MPEG-4 define cómo debe ser la trama de salida de un codificador de vídeo. Aunque la norma permite codificar diferentes tipos de secuencias de vídeo, desde la perspectiva del desarrollo de esta tesis, sólo resulta de interés la codificación de secuencias rectangulares de vídeo natural; quedando por tanto fuera de los objetivos de este trabajo, el tratamiento de cualquier otro tipo de secuencias de vídeo definidas en el estándar.

La primera versión de esta parte data de 1999 y fue completada con correcciones y añadidos hasta obtener la última versión de la misma que se publicó en el año 2008. Al igual que ocurre en MPEG-2, dentro de la norma se definen una serie de perfiles y niveles que permiten diseñar codificadores (y descodificadores) compatibles con un subconjunto de la especificación.

---

[18] En [PE02] puede encontrarse información detallada de las partes no descritas en este apartado.

[19] El tratamiento diferenciado que se da a la parte 10 de la norma se justifica tanto por las importantes diferencias que posee respecto a la parte 2 de la norma MPEG-4, como por la relevancia que ha alcanzado desde un punto de vista industrial.

Las herramientas que soporta cada perfil se agrupan en lo que se denominan objetos audiovisuales de modo que cada objeto audiovisual implica el uso de determinadas herramientas de codificación. Normalmente el perfil toma el nombre del objeto audiovisual más complejo que admite. La Tabla 2-4 muestra los 18 perfiles que se encuentran definidos en la última versión de la norma[20] y los objetos audiovisuales que incluye cada uno de ellos.

Tipos de objeto / Perfiles	*Simple*	*Core*	*Main*	*Simple Scalable*	*N-bit*	*Animated 2D Mesh*	*Basic Animated Texture*	*Scalable Texture*	*Simple Face*	*Core Scalable*	*Advanced R.T. Simple*	*Adv. Coding Efficiency*	*Adv. Scalable Texture*	*Simple FBA*	*Advanced Simple*	*Fine Granul. Scalable*	*Simple Studio*	*Core Studio*
Simple	X																	
Simple Scalable	X			X														
Core	X	X																
Main	X	X	X					X										
N-bit	X	X			X													
Hybrid	X	X				X	X	X	X									
Basic Animate Texture							X	X	X									
Scalable Texture								X										
Simple FA									X									
Adv. RT Simple	X										X							
Core Scalable	X	X		X						X								
Advanced Coding Efficiency	X	X										X						
Advanced Core	X	X											X					
Advanced Scalable Texture													X					
Simple FBA														X				
Advanced Simple	X														X			
Fine Granul. Scalable	X														X	X		
Simple Studio																	X	
Core Studio																	X	X

Tabla 2-4. Perfiles definidos en MPEG-4 parte 2.

[20] Debido a la dificultad que tiene la traducción del nombre de alguno de los perfiles al castellano, se ha optado por emplear el término inglés que aparece en la norma para todos ellos.

De todos los perfiles disponibles, tan sólo uno de ellos ha sido de interés durante el desarrollo de esta tesis. Se trata del perfil Simple Avanzado (*Advanced Simple Profile* o ASP). Las herramientas empleadas por los objetos audiovisuales incluidos en este perfil se resumen en la Tabla 2-5.

Perfil	Herramientas de codificación utilizadas	Características de las imágenes
Simple Avanzado	Codificador híbrido basado en DCT, cuantificación y estimación de movimiento con imágenes I, P y B en formato entrelazado o progresivo 4:2:0, estimación de movimiento con predicción con resolución de ¼ píxel y compensación global de movimiento (*Global Motion Compensation* o GMC). También puede emplearse la herramienta, *Short Header*, que proporciona una codificación compatible con H.263.	Vídeo natural, con imágenes rectangulares de tamaño arbitrario.

Tabla 2-5. Herramientas definidas para el perfil Simple Avanzado.

En la Tabla 2-6 se presentan los parámetros más representativos de los niveles asociados al perfil de interés. De todos ellos, y debido al tamaño de la imagen que soporta, el más interesante es el nivel 5 (ASP@L5). Este binomio de perfil y nivel ha sido el que se ha empleado para el desarrollo de esta tesis por lo que, a partir de este instante, todas las referencias que se hagan a MPEG-4 parte 2 se referirán a esta combinación.

Perfil	Nivel	Tamaño Imagen (píxeles)	Número máximo de objetos	Imágenes/ segundo	Régimen binario máximo (kbps)
Simple Avanzado	L0	176 × 144	1	15	128
	L1	176 × 144	4	15	128
	L2	352 × 288	4	15	384
	L3	352 × 288	4	30	768
	L4	352 × 576	4	30	3000
	L5	720 × 576	4	30	8000

Tabla 2-6. Niveles asociados al perfil Simple Avanzado.

2.2.3.3 Perfil Simple Avanzado

Un codificador MPEG-4 ASP emplea un esquema de codificación híbrido similar al mostrado en la Figura 2-5 para su implementación. A continuación se resumen las herramientas de codificación que emplea este estándar, indicando las particularidades que introduce la norma a la hora de implementar los diversos bloques funcionales del codificador.

2.2.3.3.1 Características generales

Para el perfil y nivel seleccionado es posible trabajar con imágenes, tanto en formato progresivo, como en formato entrelazado; empleando en todos los casos el esquema de muestreo 4:2:0 que se presentó en la Figura 2-2. El codificador puede emplear imágenes tipo I, P, B o S-(GMC) siendo estas últimas imágenes con GMC. MPEG-4 incluye herramientas de protección contra errores, como los códigos VLC reversibles (RVLC), la resincronización y/o el particionado de datos (*data partitioning*).

2.2.3.3.2 Transformación al dominio de la frecuencia

La transformación al dominio de la frecuencia se realiza empleando la DCT cuya expresión fue mostrada en Ec. 2-1 en la que los bloques de 8×8 píxeles de entrada están codificados con 9 bits, mientras que los de salida se obtienen con 12 bits [IEEE98]. Al igual que en el caso de MPEG-2, es posible realizar una transformación de campo o de cuadro para imágenes en formato progresivo.

2.2.3.3.3 Cuantificación

La cuantificación puede realizarse empleando dos formas diferentes denominadas primer método de cuantificación y segundo método de cuantificación; siendo el segundo de ellos compatible con el estándar H.263 [ITU98]. En ambos casos se aplica un factor de cuantificación diferente al coeficiente de continua de los macrobloques *intra*, al resto de coeficientes *intra* y a los coeficientes *inter*. Cuando se emplea el primer método de cuantificación, se utiliza una de las dos matrices de cuantificación definidas en el estándar que aplica un factor diferente a cada coeficiente.

2.2.3.3.4 Codificador de entropía

Para codificar los coeficientes cuantificados de salida de la DCT y el resto de los elementos sintácticos que genera el codificador se emplean códigos de longitud variable similares a los empleados en MPEG-2 y descritos en 2.2.2.2.4[21]. La codificación de los diversos tipos de coeficientes en el dominio de la frecuencia se realiza del siguiente modo:

- El coeficiente DC de los macrobloques tipo *intra* se codifica con predicción empleando como referencia para esta predicción, bien el bloque situado a la izquierda (predicción horizontal), bien el bloque situado encima (predicción vertical). Una vez obtenido el predictor, se codifica la diferencia y el número de bits necesario empleando códigos de longitud variable.
- Los coeficientes de la primera fila y/o columna de los macrobloques tipo *intra* pueden codificarse con o sin predicción. Si no se emplea predicción, la codificación se realiza como en MPEG-2 mediante tablas VLC. Si se utiliza predicción, ésta se calcula empleando la fila o columna correspondiente al bloque que contiene el coeficiente DC que ha sido utilizado como predictor en la descodificación del coeficiente de continua.
- Coeficientes de los macrobloques *inter* y resto de los coeficientes de los macrobloques *intra*. Para estos coeficientes el modo de codificación es idéntico al empleado en MPEG-2, con la salvedad de que el barrido de los coeficientes no tiene porqué seguir el patrón *Zig-Zag Scan*, si se emplea la predicción de los coeficientes AC de los macrobloques *intra*. En este caso se definen el barrido denominado *Alternate-Horizontal-Scan*, que se emplea si los coeficientes de la primera fila de los macrobloques *intra* están codificados con predicción horizontal, y el barrido *Alternate-Vertical-Scan*, si se ha usado la predicción vertical.

[21] En MPEG-4 ASP es posible emplear códigos VLC reversibles para codificar los coeficientes de la DCT. Este tipo de códigos tienen la particularidad de que pueden descodificarse en ambos sentidos por lo que en caso de que se produzca un error en la trama de bits es posible localizar el siguiente elemento de resincronización en la trama y continuar descodificando "hacia atrás" el resto de los coeficientes. Este tipo de codificación forma parte de las herramientas que aporta MPEG-4 para la recuperación frente a errores.

Los vectores de movimiento se codifican con predicción, empleando como predictor la mediana de los vectores de movimiento de sus vecinos más próximos. Así, en la Figura 2-16, para obtener el predictor del vector de movimiento del macrobloque X, se emplearán los vectores de los macrobloques A, B y C[22]. Si alguno de los vectores no está disponible, la mediana se calculará empleando el resto de vectores. Si ninguno de los vectores está disponible, el vector se codificará sin predicción.

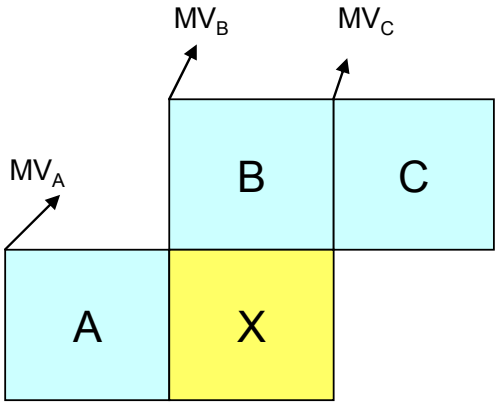


Figura 2-16. Predicción del vector de movimiento a nivel de macrobloque.

2.2.3.3.5 Estimación y compensación de movimiento

La predicción de movimiento se realiza con la información de luminancia de macrobloques de 16×16 píxeles (imágenes P, B y S-(GMC)) o de bloques de 8×8[23] píxeles (sólo para imágenes P y S-(GMC)). Las predicciones pueden realizarse con precisión de ½ o de ¼ de píxel.

Para calcular los píxeles intermedios que permiten obtener un vector de movimiento con resolución de ½ píxel, se emplea un filtro FIR (*Finite Impulse Response*) de 8 etapas. El procedimiento para obtener las muestras intermedias consiste en añadir una orla de 1 píxel de los macrobloques vecinos al macrobloque que se está procesando, con lo que se obtiene una referencia de 18×18 píxeles. A esta referencia se le añade una banda de 3 píxeles por cada lateral, obtenidos como réplicas simétricas de los píxeles de cada uno de sus bordes, tal y como se muestra en la Figura 2-17.

Si el píxel a interpolar se encuentra entre dos píxeles enteros tanto vertical como horizontalmente, éste se obtiene empleando los 8 píxeles que le rodean. En la Figura 2-18, el píxel ‘a’ se obtiene empleando los píxeles ‘A’, ‘B’, ‘C’, ‘D’ y ‘E’, análogamente para calcular el píxel ‘m’ se emplean los píxeles ‘F’, ‘B’, ‘G’, ‘H’ e ‘I’. Finalmente, los píxeles que no se encuentran entre píxeles enteros, como por ejemplo el píxel ‘aa’, son calculados empleando los píxeles con resolución de ½ píxel obtenidos previamente aplicando un filtro en la dimensión horizontal y, posteriormente, otro en la dimensión vertical.

[22] Si en lugar de realizar la predicción a nivel de macrobloque, ésta se realiza a nivel de bloque, el criterio es el mismo pero empleando los vectores de los bloques vecinos en lugar de los de los macrobloques.

[23] En este caso se obtienen cuatro vectores de movimiento por macrobloque para luminancia y uno para crominancia que se calcula como la media de los vectores de luminancia.

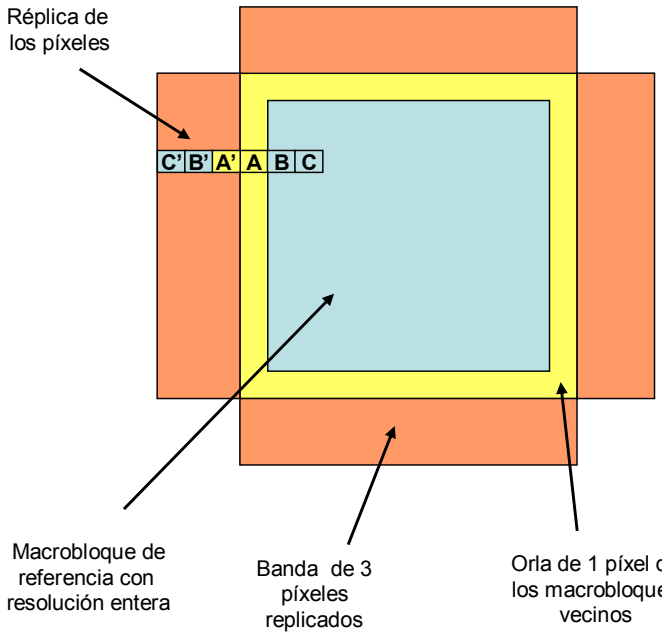


Figura 2-17. Estructura empleada para calcular los puntos intermedios con precisión de ½ píxel en MPEG-4.

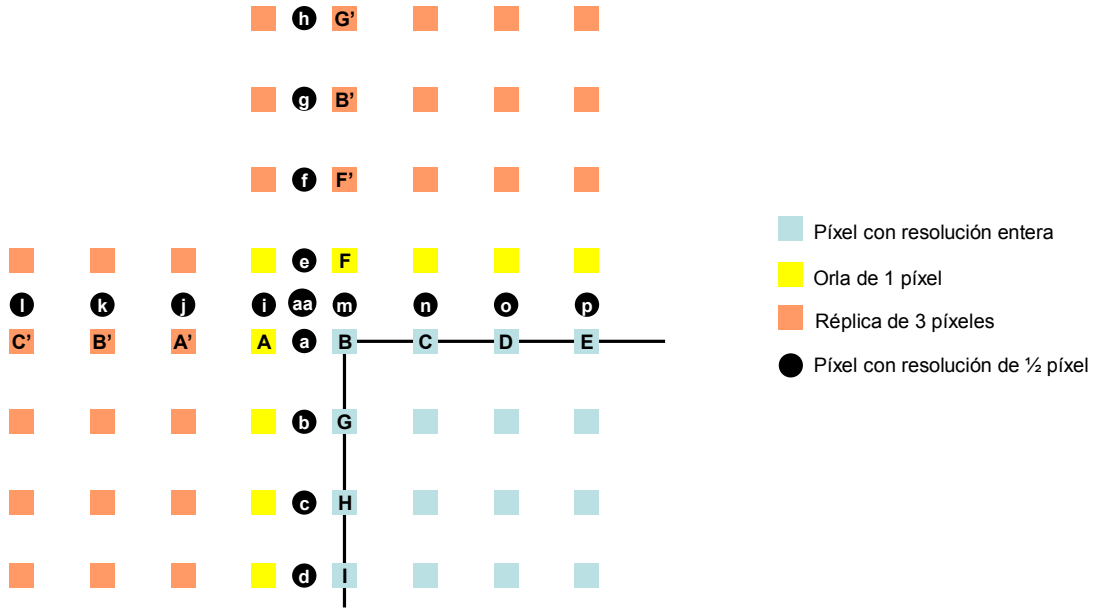


Figura 2-18. Obtención de los píxeles intermedios con precisión de ½ píxel en MPEG-4

Al igual que es posible obtener puntos intermedios entre los píxeles enteros, es posible repetir esta estrategia para obtener puntos intermedios entre los píxeles con resolución de ½ píxel que tienen, por tanto, una precisión de ¼ píxel. El cálculo de los píxeles intermedios se realiza promediando los píxeles más próximos. En la Figura 2-19 se muestran los píxeles desde el 'a' al 'h' que representan los píxeles intermedios con resolución de ¼ píxel. Así, el píxel 'd' se obtiene promediando 'D' y 'E', el píxel 'g' promediando 'AA' y 'E' y, finalmente, 'f' se calcula a partir de 'AA', 'D', 'E' y 'G'.

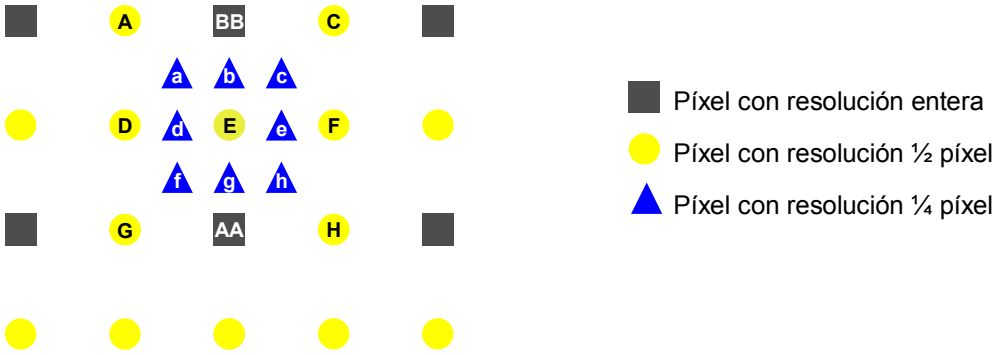


Figura 2-19. Interpolación empleada para obtener los píxeles intermedios con resolución de ¼ píxel en MPEG-4.

Habitualmente, un codificador obtiene un vector de movimiento con precisión de ¼ píxel localizando la mejor de las ubicaciones dentro del área de búsqueda con resolución entera, afinando la búsqueda para obtener un vector con resolución de ½ píxel y, finalmente, realizando una nueva búsqueda alrededor de la anterior en las ubicaciones que poseen precisión de ¼ píxel.

A diferencia de MPEG-2, en el proceso de estimación de movimiento los vectores pueden apuntar fuera de la imagen. Los píxeles que se encuentran fuera de la imagen son réplicas de los píxeles más próximos de los bordes de dicha imagen.

Se define un nuevo modo de compensación denominada compensación de movimiento directa para imágenes tipo B con predicción bidireccional. En la compensación directa, los vectores de movimiento hacia delante (MV_F) y hacia atrás (MV_B), correspondientes a un macrobloque de la imagen tipo B que se está codificando, se obtienen a partir de un único vector de movimiento (MV_D) y del vector de movimiento ya calculado para el macrobloque que ocupa una posición homóloga en la imagen de referencia posterior.

Finalmente, es posible emplear compensación global de movimiento para las imágenes de tipo S-(GMC). Este tipo de compensación es especialmente interesante para secuencias con movimiento de cámara, en las que se obtienen uno o varios vectores de movimiento que afectan a toda la imagen. El codificador puede, para cada macrobloque de este tipo de imagen, indicar si se codifica sin predicción, con predicción convencional o con predicción global.

En la compensación de movimiento global, para cada imagen puede haber varios vectores de movimiento globales que representan el desplazamiento de sendos puntos de referencia desde la imagen anterior hasta la que se está codificando. En el compensador, cada píxel del macrobloque codificado se obtiene realizando una interpolación a partir de varios píxeles de la imagen de referencia y la ubicación de estos píxeles se obtiene utilizando los vectores de movimiento globales.

2.2.3.3.6 La trama MPEG-4

La trama de salida que debe generar un codificador conforme con el estándar MPEG-4 Vídeo se muestra en la Figura 2-20[24]. La trama comienza con una cabecera de secuencia de objeto visual (VS) y una cabecera de objeto visual (VO) que representan, respectivamente, el perfil y nivel de la presentación y el tipo de objeto cuyos datos se envían a continuación.

[24] La estructura de la trama que se presenta es la que corresponde a una trama elemental MPEG-4 Parte 2 en la que no se están empleando las herramientas de multiplexación definidas en la parte 1 (MPEG-4 Sistema) de la norma.

Para los objetos de vídeo, la trama contiene un código de inicio de objeto de vídeo y una cabecera para cada capa de objeto de vídeo (VOL). La cabecera de VOL contiene información y algunas opciones utilizadas en la codificación. Tras ésta se inserta una cabecera de grupo de VOP (GOV) y, para cada imagen, una cabecera de imagen (VOP) seguida de los datos de la misma. Al final de cada imagen puede insertarse otra, o bien un final de secuencia de objeto visual.

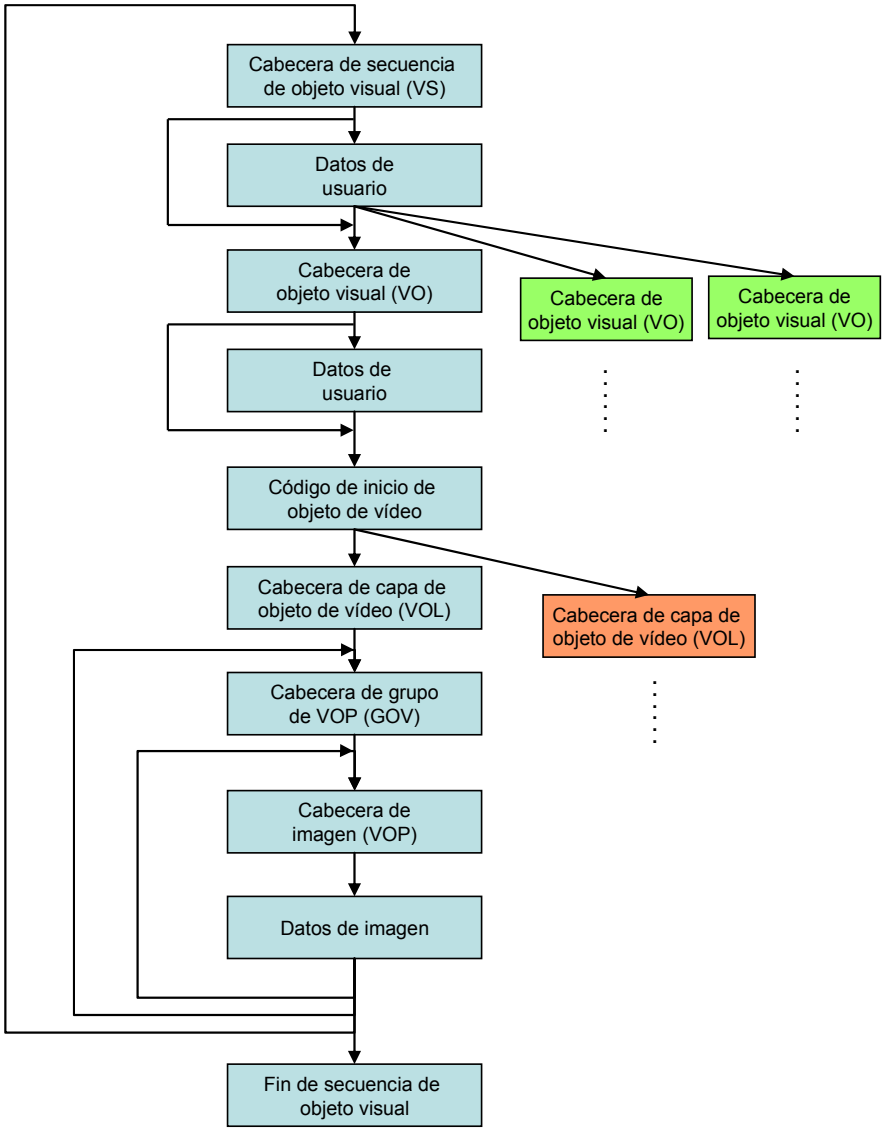


Figura 2-20. Trama de salida de un codificador MPEG-4.

Dentro de cada VOL, cada imagen se codifica con una cabecera de VOP seguida de los datos de la imagen, como se muestra en la Figura 2-21. Opcionalmente, los datos de la imagen pueden ir encapsulados en paquetes de vídeo (*video packets*) que cumplen una función parecida a las rebanadas (*slices*) de MPEG-2, facilitando la resincronización del descodificador si en algún momento se produce un error en la trama. Los datos de la imagen se dividen en macrobloques para ser incluidos en la trama, de modo que cada uno de ellos está formado por una cabecera y los coeficientes asociados al mismo.

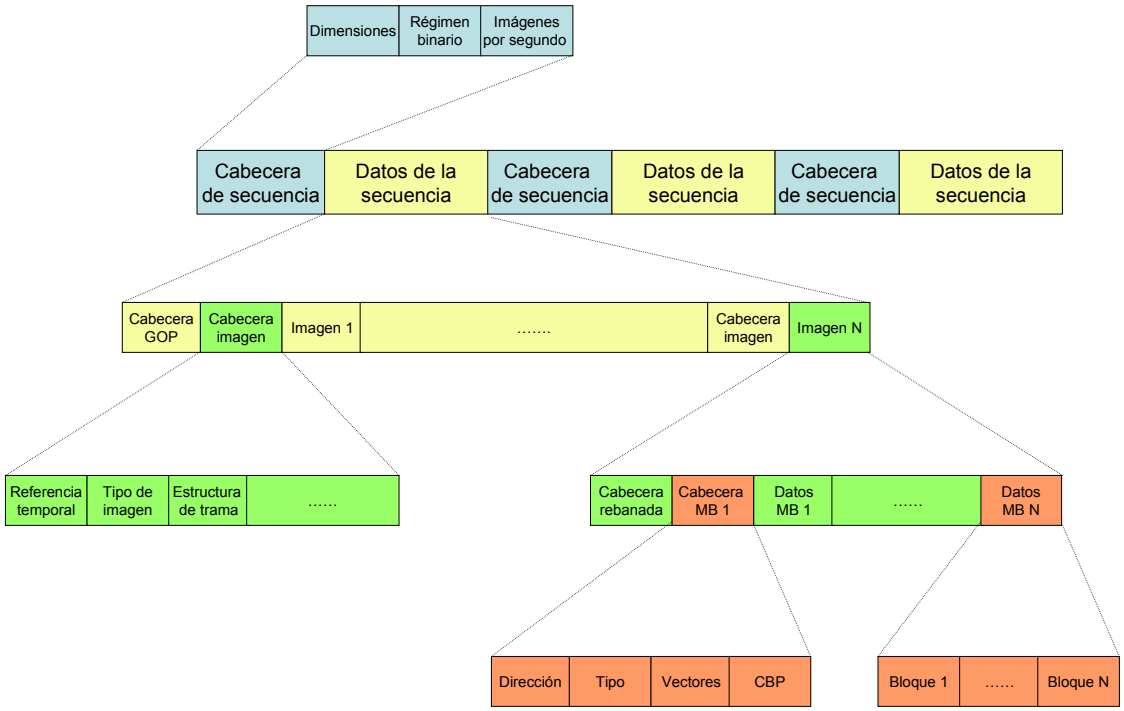


Figura 2-21. Estructura jerárquica de la trama de salida de un codificador MPEG-4.

2.2.4 El estándar H.264/AVC

Este estándar nace en 2003 como consecuencia del esfuerzo conjunto del *Motion Picture Expert Group* (MPEG) de ISO y el *Video Coding Expert Group* (VCEG) de ITU para obtener un estándar de codificación eficiente para imágenes rectangulares de vídeo natural, como las empleadas en la difusión de la televisión digital.

2.2.4.1 Introducción al estándar H.264/AVC

El estándar es publicado simultáneamente por ISO, como la parte 10 de la norma ISO/IEC 14496 (*Advanced Video Coding* o AVC) y con el nombre H.264, por parte de la ITU [ITU03]. Posteriormente, se han publicado cuatro nuevas versiones: una en el año 2005 y otras dos en el año 2007.

Al igual que todos los estándares mostrados anteriormente, la norma define una serie de perfiles y niveles. La relación de perfiles y niveles se ha ido modificando a lo largo de las versiones del estándar que se han publicado. La Tabla 2-7 muestra los 3 perfiles definidos en la primera versión del estándar (*Baseline, Main y Extended*), así como las herramientas de codificación que soportan cada uno de ellos.

Posteriormente, en la segunda versión de la norma, se añadieron 4 nuevos perfiles orientados fundamentalmente al desarrollo de aplicaciones profesionales, incluyendo características no soportadas por los perfiles anteriores[25] [STL04]. La Tabla 2-8 muestra las herramientas que añaden estos nuevos perfiles.

[25] Todos estos nuevos perfiles se basan en el perfil *Main* por lo que heredan todas sus herramientas.

Herramienta	*Baseline*	*Main*	*Extended*
Imágenes I y P	x	x	x
Imágenes B		x	x
CAVLC	x	x	x
CABAC		x	x
Codificación entrelazada		x	x
Corrección de errores	x		x
Imágenes SP y SI			x

Tabla 2-7. Perfiles definidos en la primera versión del estándar H.264 y herramientas empleadas.

Herramienta	*High*	*High 10*	*High 4:2:2*	*High 4:4:4*
Transformaciones de 8×8 y 4×4	x	x	x	x
Matrices escalables de cuantificación	x	x	x	x
Cuantificación diferente para crominancias	x	x	x	x
Formato monocromo	x	x	x	x
Muestreo de crominancias 4:2:2			x	x
Formato de muestreo 4:4:4				x
Muestras de 9 ó 10 bits		x	x	x
Muestras de 11 ó 12 bits				x
Trasformación residual del color				x
Codificación sin pérdidas				x

Tabla 2-8. Perfiles definidos en la segunda versión del estándar H.264.

La tercera versión del estándar ha incluido cinco nuevos perfiles orientados a aplicaciones profesionales (*High 10 Intra, High 4:2:2 Intra, High 4:4:4 Intra, CAVLC 4:4:4 Intra y High 4:4:4 Predictive*). La cuarta versión incluyó en el anexo G la codificación de vídeo escalable SVC dentro de la cual se definen tres nuevos perfiles (*Scalable Baseline*, *Scalable High* y *Scalable High Intra*). Finalmente, la última versión publicada ha incluido el anexo H para la codificación multivista en el que se definen dos nuevos perfiles (*Stereo High Profile y Multiview High Profile*).

De todos los perfiles definidos en el estándar, los que comercialmente se están empleando para la difusión de televisión digital son el perfil *Baseline* (BP) y el *Main* (MP). Por este motivo los trabajos de tesis relacionados con este estándar se han centrado en estos dos perfiles.

Además de los perfiles, el estándar define una serie de niveles que se caracterizan fundamentalmente por el tamaño de la imagen, el número de imágenes por segundo, el régimen binario y el número de imágenes de referencia que se deben almacenar. La Tabla 2-9 presenta los 16 niveles definidos en la última versión de la norma. De todos ellos el que ha sido de interés para el desarrollo de esta tesis es el nivel 3.

Nivel	Tamaño de la imagen	Imágenes por segundo	Régimen binario	Máximo número de imágenes de referencia
1	QCIF	15	64 kbps	4
1b	QCIF	15	128 kbps	4
1.1	QCIF/ CIF	7.5 (CIF) / 30(QCIF)	192 kbps	2 (CIF) / 9 (QCIF)
1.2	CIF	15	384 kbps	6
1.3	CIF	30	768 kbps	6
2	CIF	30	2 Mbps	6
2.1	HHR (480i/576i)	30/25	4 Mbps	6
2.2	SD	15	4 Mbps	5
3	SD	30/25	10 Mbps	5
3.1	1280×720p	30	14 Mbps	5
3.2	1280×720p	60	20 Mbps	4
4	HD (720p o 1080i)	60p / 30i	20 Mbps	4
4.1	HD (720p o 1080i)	60p / 30i	50 Mbps	4
4.2	1920×1080p	60p	50 Mbps	4
5	2k × 1k	72	135 Mbps	5
5.1	2k, 1k o 4k × 2k	120/30	240 Mbps	5

Tabla 2-9. Niveles definidos en la última versión del estándar H.264.

2.2.4.2 Características de los perfiles *Baseline* y *Main*

La estructura de un codificador H.264 se basa en el esquema híbrido presentado en la Figura 2-5 al que se han añadido algunas modificaciones, tal y como se muestra en la Figura 2-22. Realmente no se trata de grandes novedades sino que la suma de pequeñas mejoras es la que produce una mayor eficacia en la codificación a costa de incrementar sustancialmente la complejidad del algoritmo [KA03].

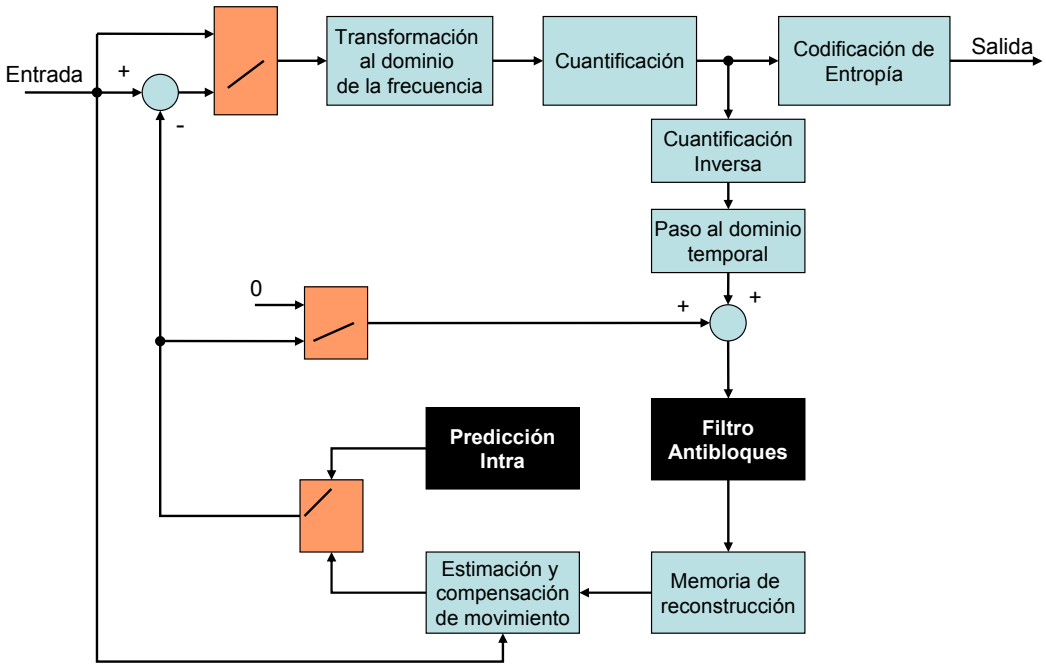


Figura 2-22. Esquema de codificación híbrido empleado en H.264.

Seguidamente se describen brevemente algunas características generales del sistema así como las novedades que aparecen en cada uno de los bloques que componen el codificador [WSBL03].

2.2.4.2.1 Características Generales

Los macrobloques que forman una imagen se agrupan en una estructura de nivel superior denominada rebanada (*slice*). A diferencia de lo que ocurre en MPEG-2, en el que las rebanadas deben estar formadas por macrobloques contiguos y formar parte de una única tira horizontal, en H.264 existe una mayor libertad para agrupar los macrobloques de una rebanada pudiendo hacerlo de forma arbitraria (*Arbitrary Slice Order*[26] o ASO) y extenderse a más de una línea horizontal de macrobloques. La Figura 2-23 muestra dos ejemplos de organización de los macrobloques de una imagen.

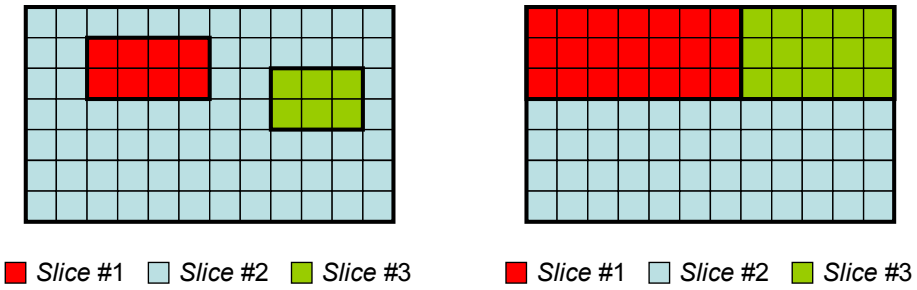


Figura 2-23. Ejemplos de agrupaciones arbitrarias de macrobloques en rebanadas.

En relación con la codificación de secuencias entrelazadas, se definen tres opciones para que el codificador elija la más apropiada en cada ocasión. La primera consiste en juntar ambos campos y realizar la codificación como si se tratara de una secuencia progresiva (modo *frame*). En la segunda se codifica cada uno de los campos independientemente (modo *field*). Finalmente, la tercera opción consiste en combinar ambos campos pero, a la hora de codificar cada pareja de macrobloques verticales, se decide si se hace como si fueran dos macrobloques de una imagen progresiva o dos campos de una imagen entrelazada.

La decisión entre las dos primeras opciones se puede realizar dinámicamente imagen a imagen empleando el procedimiento denominado *Picture-Adaptive Frame/Field* (PAFF). Esta posibilidad permite que las imágenes con más movimiento se codifiquen en formato de campo, mientras que las que tienen menos se codifican con formato de cuadro [WSBL03].

Por otro lado, la tercera opción se emplea cuando las imágenes tienen partes con movimiento y otras estáticas puesto que, en estos casos, es mejor realizar una decisión campo/cuadro para cada pareja de macrobloques aprovechando de este modo las ventajas de ambas opciones. A este método se le denomina *Macroblock-Adaptive Frame/Field* (MBAFF). La Figura 2-24 muestra como, a partir de una pareja de macrobloques pertenecientes a una imagen con formato de cuadro, se obtienen dos nuevos macrobloques de modo que cada uno de ellos contiene los píxeles de uno de los campos. Una vez realizada la separación, éstos se codifican independientemente.

[26] La herramienta ASO sólo está disponible en el perfil *Baseline*.

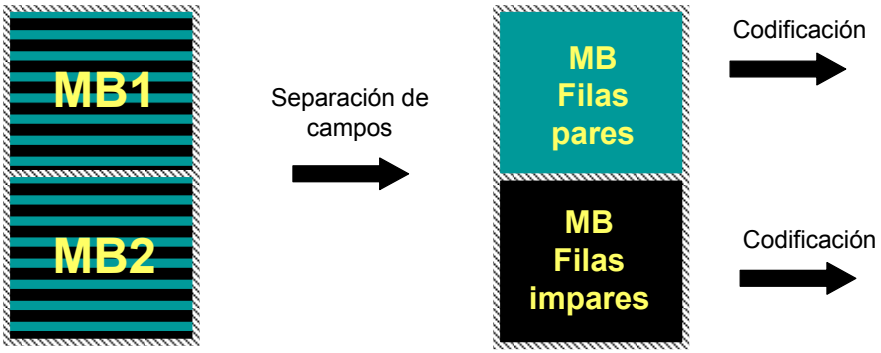


Figura 2-24. Separación de una pareja de macrobloques de una imagen en formato cuadro en dos macrobloques que poseen un campo cada uno de ellos.

2.2.4.2.2 Transformación al dominio de la frecuencia y cuantificación

En el estándar H.264 la transformación al dominio de la frecuencia y la cuantificación se realizan conjuntamente, puesto que la transformada que se emplea es la Transformada Entera del Coseno (*Integer Cosine Transform* o ICT) en la que resulta ventajoso integrar el proceso de cuantificación. Esta transformada es una variación de la transformada discreta del coseno [Cha89] en la que todas las operaciones se realizan exclusivamente con números enteros de 16 bits, eliminando de este modo los problemas de resolución en la representación de los coeficientes. Este hecho permite eliminar las pérdidas asociadas al proceso de transformación al dominio de la frecuencia.

La expresión matricial que permite calcular la ICT es análoga a la de la DCT presentada en la Ec. 2-2 pero modificando algunos coeficientes de la matriz *A* para que contenga exclusivamente números enteros. La ICT se aplica a bloques de 4×4 píxeles empleando la expresión que se muestra en la Ec. 2-6. En esta expresión se observa que el núcleo de la ICT (CXC^T) se realiza exclusivamente con operaciones de suma, resta y desplazamiento lo que facilita su materialización en plataformas *hardware*. Por otro lado, la matriz *E* supone hacer un escalado puesto que multiplica cada uno de los coeficientes obtenidos en la operación anterior por un factor constante.

$$Y = \left(CXC^T\right) \otimes E = \left[\begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{pmatrix} \cdot \begin{pmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{pmatrix} \right] \otimes \begin{pmatrix} a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \\ a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \end{pmatrix}$$

Ec. 2-6

Donde:

$$a = \frac{1}{2} \text{ y } b = \sqrt{\frac{2}{5}}$$

$\otimes$ representa el producto escalar.

X es la matriz de píxeles en el dominio del tiempo.

Y es la matriz de coeficientes resultado.

Dado que la cuantificación multiplica a su vez por una constante los coeficientes de la ICT del mismo modo que se hace con la matriz de escalado (*E*), es posible realizar ambas operaciones conjuntamente reduciendo de este modo el número de multiplicaciones.

La operación de transformación inversa (IICT) se puede calcular matricialmente a partir de la expresión mostrada en la Ec. 2-7. Donde *a* y *b* tienen los mismos valores que para la transformación directa, *Y* representa la matriz de coeficientes en el dominio de la frecuencia y *X'* la matriz de píxeles reconstruidos.

$$X' = C_i^T (Y \otimes E_i) C_i = \begin{pmatrix} 1 & 1 & 1 & 1/2 \\ 1 & 1/2 & -1 & -1 \\ 1 & -1/2 & -1 & 1 \\ 1 & -1 & 1 & -1/2 \end{pmatrix} \cdot \left[\begin{pmatrix} y_{00} & y_{01} & y_{02} & y_{03} \\ y_{10} & y_{11} & y_{12} & y_{13} \\ y_{20} & y_{21} & y_{22} & y_{23} \\ y_{30} & y_{31} & y_{32} & y_{33} \end{pmatrix} \otimes \begin{pmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{pmatrix} \right] \cdot \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1/2 & -1/2 & -1 \\ 1 & -1 & -1 & 1 \\ 1/2 & -1 & 1 & -1/2 \end{pmatrix}$$

Ec. 2-7

2.2.4.2.3 Codificación de entropía

En el estándar H.264 se emplean dos tipos de codificación de entropía siendo ambos adaptativos con el contexto. Este tipo de codificación se basa en que los códigos que se van asignando a los diferentes elementos sintácticos dependen de los elementos que se han codificado previamente.

En el caso del perfil *Baseline* se utilizan códigos de longitud variable adaptados al contexto (*Context-Adaptive Variable-Length Coding* o CAVLC) para codificar los coeficientes cuantificados de la ICT y códigos *Exp-Golomb*[27] [Gol06] para el resto de los elementos sintácticos. Por otro lado, para el perfil *Main*, además de la codificación CAVLC, es posible emplear una codificación binaria aritmética adaptada al contexto (*Context-Adaptive Binary Arithmetic Coding* o CABAC)[28] para codificar tanto los coeficientes cuantificados como el resto de elementos que componen la trama.

2.2.4.2.3.1 Codificación CAVLC

La codificación CAVLC se basa en que la asignación de códigos a los diferentes elementos que se van a codificar no se realiza de forma estática, sino que varía dinámicamente en función del tipo de elemento a codificar y de los valores que hayan tenido algunos elementos que se han codificado previamente (contexto).

Para realizar la codificación de cada bloque de coeficientes cuantificados, éstos se ordenan siguiendo un patrón *Zig-Zag Scan* inverso, tal y como se muestra en el ejemplo de la Figura 2-25.

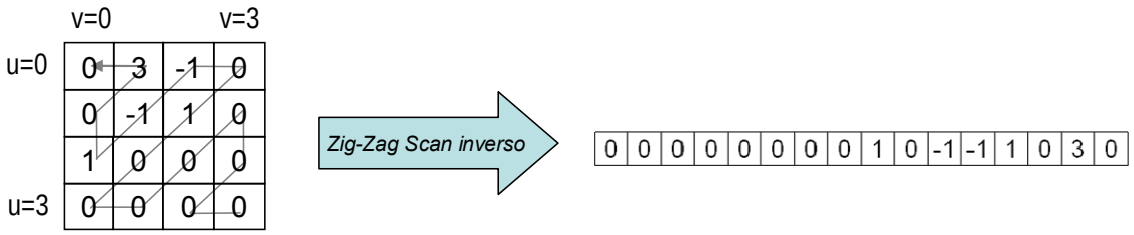


Figura 2-25. Ordenación de los coeficientes de un bloque de 4×4 siguiendo el patrón de *Zig-Zag Scan* inverso.

[27] Cada código *Exp-Golomb* se construye como una secuencia de ceros seguida por un uno y un código que representa el dato que se quiere codificar expresado por M bits de modo que la secuencia de ceros también está compuesta de M bits. La eficiencia de este tipo de códigos se basa en asociar códigos *Exp-Golomb* de pocos bits a los elementos sintácticos más probables.

[28] La eficiencia en la codificación CABAC respecto a la codificación con CAVLC está estimada en torno al 10%; sin embargo, la complejidad en la implementación también es considerablemente mayor tal y como se demuestra en [MSW03].

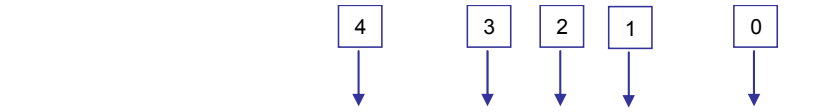
Para cada bloque de 4×4 coeficientes se codifican los siguientes parámetros:

- *Coeff_token*: representa la combinación del número de coeficientes distintos de cero (*TotalCoeff*) y el número de unos al principio de la secuencia (*TrailingOnes*). Si el número de unos es mayor que 3, el valor de *TrailingOnes* queda fijado en 3. Para el ejemplo de la figura, *CoefToken* sería la combinación (*TotalCoeff*=5, *TrailingOnes*=3). Existen varias tablas VLC para codificar estas combinaciones; la tabla que se emplea en cada caso se escoge en función del número de coeficientes distintos de cero que posean los bloques vecinos del que se desea codificar.
- *Trailing_ones_sign_flag*: los bits de signo de los unos del principio de la secuencia (hasta 3) se codifican con un bit; 0 si es positivo y 1 si es negativo.
- *Level*: El resto de coeficientes se codifican con un conjunto de tablas VLC. Cada coeficiente se codifica con un prefijo (*prefix*) y un sufijo (*suffix*). Después de codificar cada coeficiente, la tabla para la codificación del siguiente se escoge en función del valor del coeficiente recién codificado.
- *Total_zeros*: representa el número de ceros después del primer coeficiente no cero. En el caso del ejemplo serían 3. Este parámetro se codifica empleando una serie de tablas VLC; la tabla que se utiliza se escoge en función del valor de *TotalCoeff*.
- *Run_before*: indica el número de ceros que sigue a cada coeficiente distinto de cero, hasta el siguiente coeficiente distinto de cero. Se asigna un valor de *Run_before* por cada coeficiente, a excepción del último, que no es necesario codificar. En el ejemplo de la figura este parámetro sería 1 para el primer coeficiente, 0 para el segundo y el tercero y 1 para el cuarto. *Run_before* se codifica con un conjunto de tablas VLC; para cada caso se escoge una en función del número total de ceros que hay a la derecha del último coeficiente codificado.

En la Figura 2-26 se muestra la codificación del ejemplo de la figura anterior. En la parte superior de la figura se presenta la lista de coeficientes reordenada eliminado los ceros iniciales e indicando el número de orden que ocupa cada uno de los que son distintos de cero. Este número de orden se emplea posteriormente para asignar los bits que componen el código de salida.

La tabla muestra en la columna “elemento” cada uno de los parámetros que se han relacionado anteriormente; junto al nombre del elemento se muestra, entre paréntesis, el coeficiente al que se refiere; en la columna “valor” se indica el valor que posee cada uno de ellos y, finalmente, en la columna “código” se muestran los códigos utilizados para cada uno de los elementos sintácticos antes mencionados[29]. Recopilando todos los códigos asignados se aprecia que para codificar todos los coeficientes del bloque, sólo es necesario emplear 24 bits.

[29] La obtención de los códigos se ha realizado consultando las tablas correspondientes para cada uno de los casos. Por simplicidad, en la figura no se ha incluido la tabla de la que se extrae cada uno de los códigos.



TotalCoeffs = 5

Totalzeros = 3

TrailingOnes = *3*

Elemento	Valor	Código
Coeff_token	*TotalCoeff* = 5 *TrailingOnes* =3	0000100
TrailingOnesSign (4)	+	0
TrailingOnesSign (3)	-	1
TrailingOnesSign (2)	-	1
Level (1)	1	1 (*prefix*)
Level (0)	3	001 (*prefix*) 0 (*suffix*)
Total_zeros	3	111
Run_before (4)	1	10
Run_before (3)	0	1
Run_before (2)	0	1
Run_before (1)	1	01
Run_before (0)	1	-

Figura 2-26. Codificación CAVLC del bloque presentado en la Figura 2-25.

2.2.4.2.3.2 Codificación CABAC

a) Codificación aritmética con números reales

En la codificación aritmética se asigna cada uno de los símbolos del alfabeto que se va a utilizar para la codificación a un sub-intervalo dentro del intervalo semiabierto [0,1) de la recta real. El tamaño de cada uno de los sub-intervalos es proporcional a la probabilidad de ocurrencia del símbolo. Por ejemplo, en la Tabla 2-10 se relacionan los símbolos necesarios para codificar el mensaje “ALGORITMOS”, junto con su probabilidad de ocurrencia y el sub-intervalo asignado. La asignación de cada símbolo a un sub-intervalo puede ser cualquiera, aunque ha de ser conocida por el receptor.

En la parte superior de la Figura 2-27 se observa una representación gráfica de la asignación de estos símbolos a sub-intervalos dentro del intervalo [0,1). Para realizar la codificación aritmética del mensaje se parte del intervalo [0,1) codificado con los siguientes parámetros:

- H0: Límite superior del intervalo. Inicialmente 1.0.
- L0: Límite inferior del intervalo. Inicialmente 0.0.
- R0: Rango del intervalo. Inicialmente 1.0.

La codificación de cada símbolo del mensaje supone el cálculo de un nuevo intervalo, es decir, de nuevos valores para H, L y R. La codificación se realiza en dos pasos. En primer lugar se obtiene el nuevo intervalo a partir del símbolo que se va a codificar. En la figura, el primer símbolo a codificar es la ‘A’, con lo cual el nuevo intervalo será [0.2 , 0.3), con L1=0.2, H1=0.3 y R=0.1. En segundo lugar, cada uno de los símbolos del alfabeto se reasigna al nuevo intervalo. En la figura puede verse la asignación realizada para el intervalo [0.2 , 0.3) donde, por ejemplo, el símbolo ‘I’ ha quedado en el sub-intervalo [0.21 , 0.22).

Símbolo	Probabilidad (p_S)	Sub-intervalo
M	0.1	0.0 - 0.1
I	0.1	0.1 - 0.2
A	0.1	0.2 - 0.3
R	0.1	0.3 - 0.4
O	0.2	0.4 - 0.6
L	0.1	0.6 - 0.7
G	0.1	0.7 - 0.8
S	0.1	0.8 - 0.9
T	0.1	0.9 - 1.0

Tabla 2-10. Probabilidad asociada a los símbolos incluidos en el mensaje “ALGORITMOS”.

Este procedimiento debe repetirse para cada símbolo del mensaje. En la Figura 2-27 se han codificado sólo los 4 primeros símbolos. El resultado de la codificación es un conjunto de intervalos encajados, de manera que el último de ellos representa al mensaje completo. De hecho, conociendo únicamente un número que pertenezca a este último intervalo y mediante el procedimiento inverso al aquí descrito, es posible recuperar el mensaje completo.

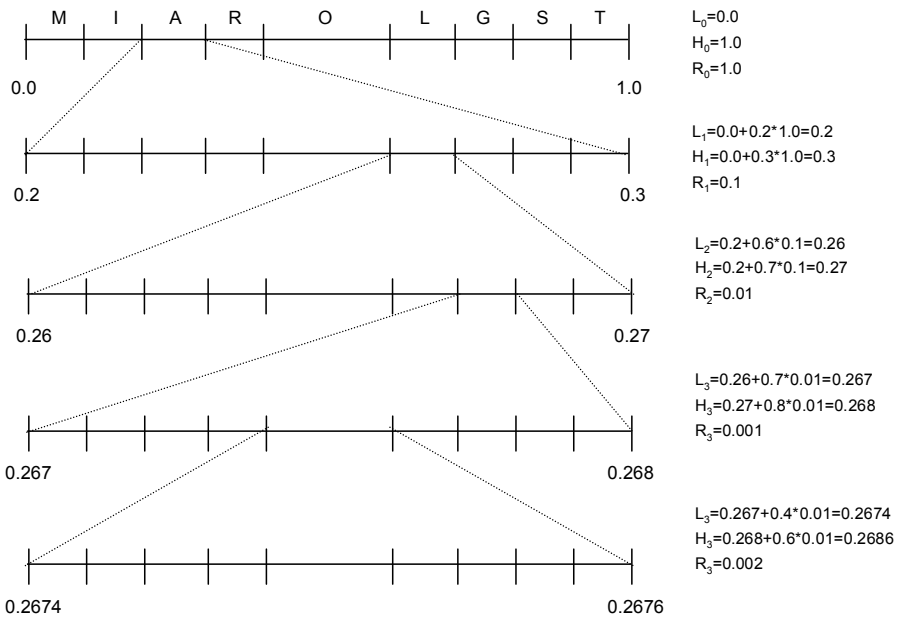


Figura 2-27. Procedimiento de codificación aritmética de un mensaje.

b) Codificación aritmética binaria

Cualquier mensaje se caracteriza por un número real perteneciente a un intervalo delimitado por un límite superior y otro inferior. La representación de este tipo de números es costosa cuando se desea implementar un sistema que emplee este tipo de codificación. Por este motivo se emplean estrategias que permiten trabajar con

rangos y probabilidades expresadas con números enteros, más fáciles de manipular en implementaciones tanto *hardware* como *software*.

Para ello las probabilidades de cada símbolo se representan como la frecuencia de aparición de dicho símbolo normalizada respecto al número de símbolos. De este modo el rango inferior asociado a cada símbolo se representa como la suma de las frecuencias de aparición de todos los símbolos previamente codificados; mientras que el límite superior de cada intervalo se expresa como el límite inferior más la frecuencia de aparición de este símbolo, tal y como se muestra en la Ec. 2-8. En dicha ecuación *CumCount[i]* es la frecuencia de aparición del símbolo i, *low_count* es el límite inferior del intervalo y *high_count* el límite superior.

$$low\_count = \sum_{i=0}^{símbolos-1} CumCount[i]$$

$$high\_count = low\_count + CumCount[i] \qquad \text{Ec. 2-8}$$

Para cada símbolo que se desea codificar es necesario calcular el nuevo intervalo teniendo en cuenta el intervalo actual y la frecuencia de aparición de cada símbolo. Para ello se calcula el paso (*step*) como el rango actual dividido entre la suma de las frecuencias de aparición de todos los símbolos como se aprecia en la Ec. 2-9.

$$step = (high\_count - low\_count + 1)/total \qquad \text{Ec. 2-9}$$

Partiendo del rango actual y del paso calculado anteriormente se recalculan los valores asociados al nuevo rango de representación (*low* o límite inferior y *high* o límite superior) empleando las ecuaciones representadas en la Ec. 2-10.

$$high\_new = (low + step \cdot high\_count - 1)$$

$$low\_new = (low + step \cdot low\_count) \qquad \text{Ec. 2-10}$$

Así, si se dispone de un alfabeto de 4 símbolos (a, b, c y d) con las frecuencias de aparición de símbolo que se muestran en la Tabla 2-11-a es posible calcular los valores de *high_count* y *low_count* de cada símbolo. La Tabla 2-11-b presenta el proceso de codificación de la cadena “abd”, empleando 15 bits. Por tanto, cualquier valor comprendido entre 0x2E00 y 0x2FFF representa unívocamente a la secuencia “abd”.

Símbolo	Frecuencia	*Low_count*	*High_count*
a	4	0	4
b	2	4	6
c	1	6	7
d	1	7	8

a) Frecuencias de aparición de los 4 símbolos que componen el alfabeto.

Símbolo	*Step*	*Low*	*High*
Valor inicial	-	0x0000	0x7FFF
a	0x1000	0x0000	0x3FFF
b	0x0800	0x2000	0x2FFF
d	0x0200	0x2E00	0x2FFF

b) Codificación de la secuencia “abd”.

Tabla 2-11. Codificación binaria con 15 bits de una secuencia de símbolos.

Si el proceso de extracción del segmento que representa a la secuencia se repite iterativamente, el rango va decreciendo progresivamente hasta llegar un momento en el que los límites inferior y superior coinciden, lo que impide continuar con el proceso. Para solventar este problema se emplean técnicas de escalado y normalización de los

rangos de modo que, en el momento en el que los dos valores que caracterizan a un segmento se encuentran en la misma mitad del rango, se puede asegurar que este segmento nunca va a abandonar dicha mitad. Por lo tanto, la información relacionada con la mitad que no se va a emplear (el bit de mayor peso) no es necesaria para los siguientes pasos en el proceso de codificación y es eliminada para el siguiente paso. En este momento se almacena en la secuencia de salida el bit más significativo y se amplia el rango multiplicando por 2, tanto el extremo superior, como el inferior [BCK07].

c) Codificación aritmética binaria adaptativa con el contexto (CABAC)

La codificación aritmética que se utiliza en H.264 se basa en la misma técnica de división de intervalos, aunque en este caso el alfabeto queda reducido a dos símbolos, el 0 y el 1, y las probabilidades de ocurrencia de cada símbolo no son fijas sino que van evolucionando en función del contexto durante el proceso de codificación. Como se muestra en la Figura 2-28 la codificación de cada sub-intervalo se realiza mediante dos parámetros:

- MPS: bit de información que identifica al símbolo más probable. Puede ser el 0 ó el 1. El otro símbolo será por tanto el menos probable, identificado como LPS.
- P_{LPS}: es la probabilidad del símbolo menos probable, y se corresponde siempre con la parte inferior del intervalo. Su valor está entre 0 y 0.5. El resto del intervalo se corresponde con la probabilidad del símbolo más probable, es decir, 1-P_{LPS}.

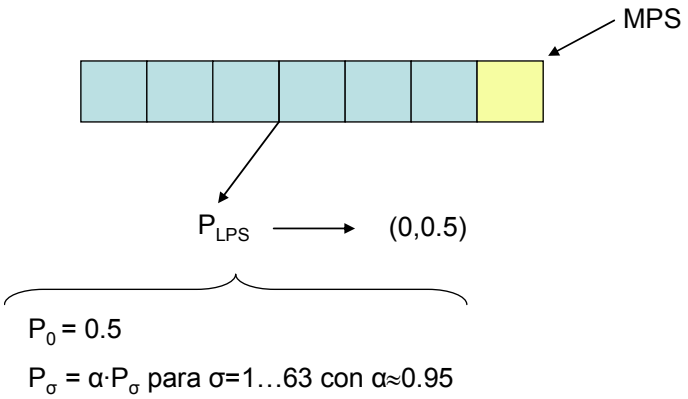


Figura 2-28. Asignación de probabilidades a MPS y LPS.

El valor de P_{LPS} se codifica con 6 bits, lo que permite un total de 64 valores, indexados desde $\sigma = 0$ hasta $\sigma = 63$. De este modo σ representa un índice que permite obtener la probabilidad del LPS y que se va actualizando cada vez que se codifica un símbolo que emplea ese modelo de probabilidad. El primer valor que se asigna a P_{LPS} es $P_0 = 0{,}5$ y el resto de posibles valores pueden calcularse de manera recursiva como se muestra en la Figura 2-29. Para codificar cada bit se parte de un estado en el cual se definen los valores iniciales de P_{LPS}, MPS, L (límite inferior del intervalo) y R (tamaño del intervalo). El límite superior del intervalo está implícito (H = L + R).

Dependiendo de si el símbolo a codificar es el MPS o el LPS, se realiza una de las transiciones definidas en la Figura 2-29. En el caso de que el símbolo sea el MPS, la P_{LPS} se decrementa, evolucionando de P_σ a $P_{\sigma+1}$. Si es el LPS, la P_{LPS} se incrementa, evolucionando como se indica en la figura. Si $\sigma = 0$ y el siguiente símbolo a codificar es el LPS, entonces se cambia el MPS (si era 0, pasa a 1 y viceversa).

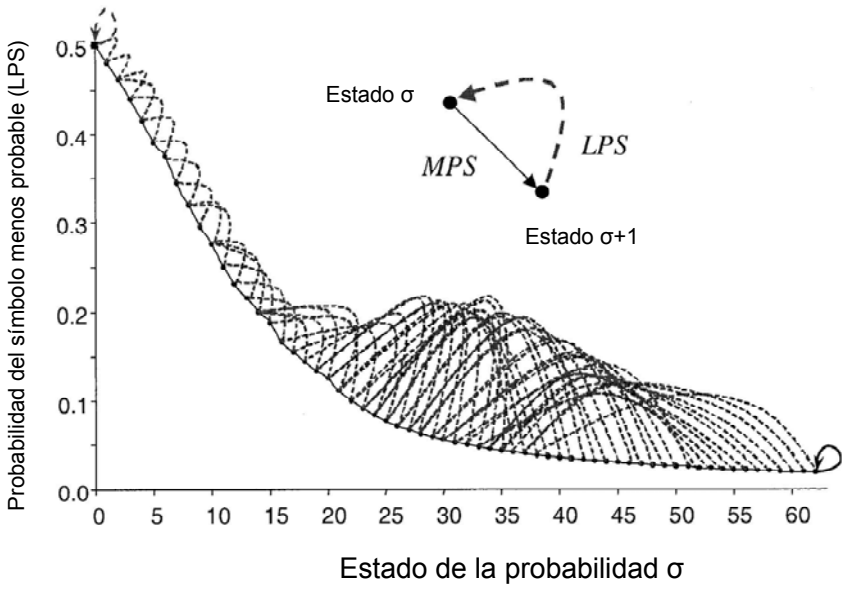


Figura 2-29. Curva de actualización del modelo de probabilidad extraída de [MSW03].

Los nuevos valores de L y R se actualizan como se muestra en la Figura 2-30. Para ello, únicamente es necesario realizar un producto, $R \cdot P_{LPS}$, que se encuentra además tabulado en el estándar para simplificar la implementación.

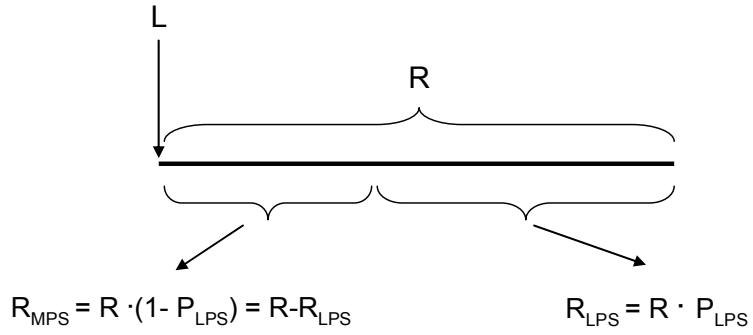


Figura 2-30. Actualización de los valores de R_{MPS} y R_{LPS}.

El valor de P_{LPS}, MPS y la curva de la figura forman un modelo de probabilidad. Durante el proceso de codificación de H.264 se mantienen 400 modelos de modo que cada elemento sintáctico de la trama puede utilizar varios de ellos para su codificación; la selección de uno u otro depende del contexto.

Finalmente, en la Figura 2-31 se muestra un diagrama de bloques simplificado del codificador aritmético adaptativo empleado en H.264. Cada uno de los elementos sintácticos de la trama se codifica inicialmente en binario (si no lo está). El proceso normal de codificación de cada bit pasa por seleccionar el modelo de probabilidad que se va a utilizar, en función del contexto, y la obtención de los nuevos valores de P_{LPS} y MPS (modo *regular*). Cuando P_{LPS} es cercana a 0.5 se utiliza un procedimiento de codificación simplificado (modo *bypass*), dado que el procedimiento normal es demasiado complejo para la escasa compresión que puede obtenerse en este caso.

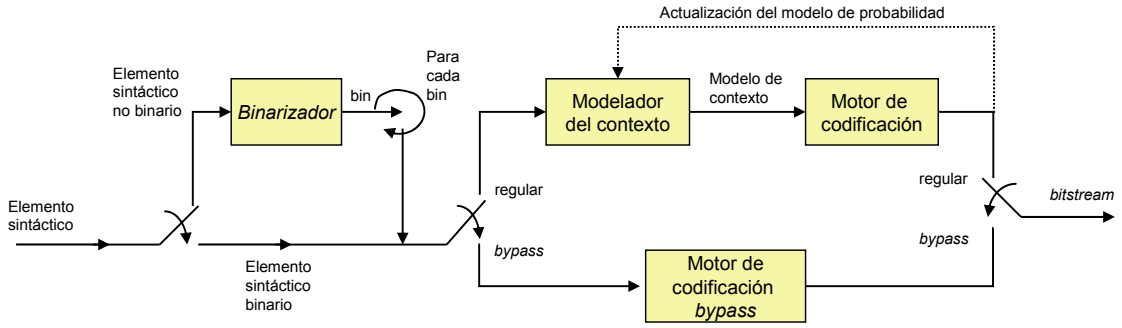


Figura 2-31. Diagrama de bloques del codificador aritmético adaptativo con el contexto empleado en H.264 [MSW03].

2.2.4.2.4 Estimación y Compensación de movimiento

Las mejoras relacionadas con la compensación de movimiento que se han incluido en H.264 son bastante significativas. A continuación se resumen las más relevantes.

2.2.4.2.4.1 Tamaño de bloque variable

El tamaño de los bloques para los que se pueden calcular las predicciones es variable. La Figura 2-32 muestra cómo un macrobloque de 16×16 píxeles de luminancia puede descomponerse en bloques de 8×16, 16×8 y 8×8 píxeles respectivamente. Posteriormente, cada bloque de 8×8 píxeles puede a su vez ser dividido en sub-bloques de 8×4, 4×8 ó 4×4 píxeles. A la hora de obtener una predicción, el codificador debe seleccionar el particionado más adecuado para cada macrobloque a codificar. Cada macrobloque, bloque o sub-bloque puede tener una predicción independiente y por tanto uno/s vector/es de movimiento diferente/s.

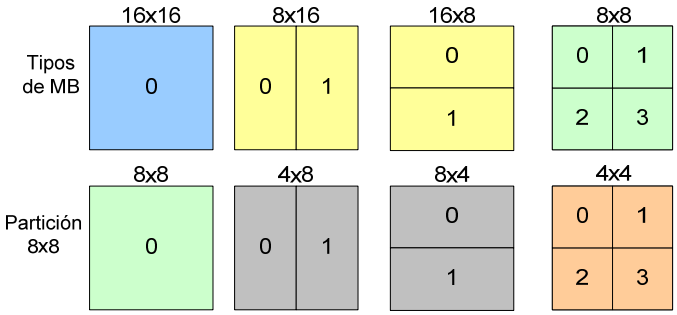


Figura 2-32. División de un macrobloque de 16×16 píxeles en bloques de menor tamaño y separación de los bloques de 8×8 píxeles en particiones menores.

La posibilidad que tiene el codificador de adaptar el tamaño del bloque permite que, para las zonas de la imagen que poseen un mayor detalle, se seleccione una partición basada en sub-bloques de 4×4 píxeles; mientras que para zonas más sencillas, se realiza una partición en macrobloques. La Figura 2-33 muestra el ejemplo de una imagen compleja en la que se aprecia la división de los macrobloques que ha realizado un codificador. En dicha imagen se observa que la zona en la que se encuentra el jugador posee un mayor detalle por lo que se codifica con bloques más pequeños; sin embargo, para las zonas más uniformes de la imagen se emplean bloques de mayor tamaño.

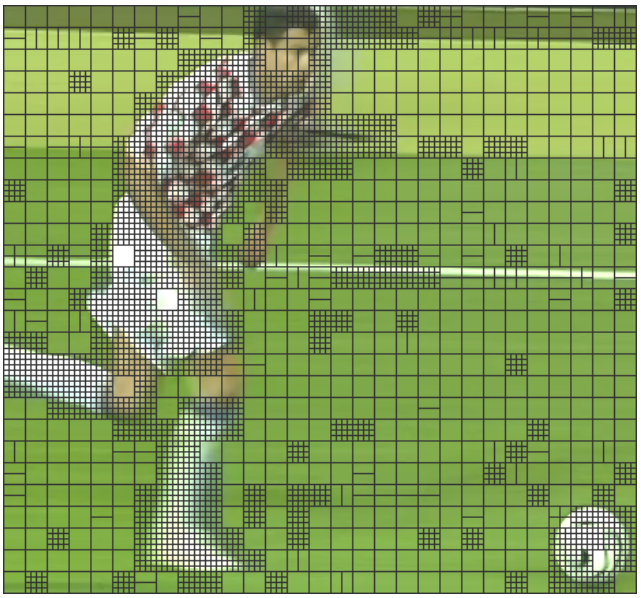


Figura 2-33. División de una imagen en diferentes tipos de bloques.

2.2.4.2.4.2 Predicción con precisión de ½ y ¼ píxel

Los vectores de movimiento se pueden calcular con precisión de hasta ¼ píxel. Para obtener los píxeles intermedios con resolución de ½ píxel se emplea un filtro FIR de 6 etapas[30], que usa los píxeles próximos al que se desea calcular. La Figura 2-34 ilustra cómo se calculan estos píxeles. Los píxeles 'b' y 'h' se obtienen como suma ponderada de los píxeles 'E', 'F', 'G', 'H', 'I' y 'J'[31], y 'A', 'C', 'G', 'M', 'R' y 'T'[32], respectivamente.

Los píxeles que ocupan posiciones intermedias entre 4 píxeles enteros, como por ejemplo el píxel 'j', se obtiene promediando en primer lugar los píxeles intermedios horizontales ('cc', 'dd', 'h', 'm', 'ee' y 'ff') y posteriormente, los píxeles intermedios verticales 'aa', 'bb', 'b', 's', 'gg' y 'hh'.

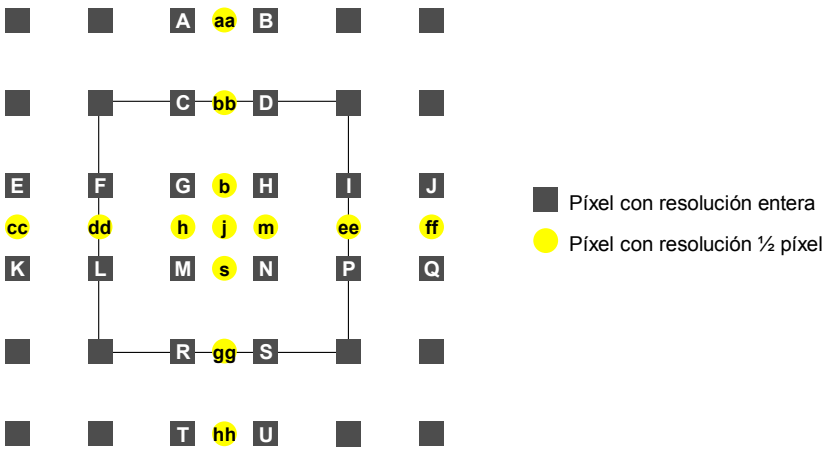


Figura 2-34. Píxeles con precisión de ½ píxel calculados empleando un filtro FIR.

[30] Se trata de un filtro simétrico con los siguientes coeficientes: 1/32, -5/32 y 20/32.

[31] Análogamente se obtienen los píxeles 'aa', 'bb', 'b', 's', 'gg' y 'hh'.

[32] Análogamente se obtienen los píxeles 'cc, 'dd', 'h', 'm', 'ee' y 'ff'.

Para obtener los píxeles con resolución de ¼ píxel se realiza un proceso de interpolación como el descrito en el apartado 2.2.3.3.5 para los píxeles de la misma resolución en MPEG-4.

2.2.4.2.4.3 Transformación de los coeficientes DC

Respecto a la transformación al dominio de la frecuencia hay que indicar que en el caso en el que se codifique un macrobloque I con predicción *intra* (ver apartado 2.2.4.2.5.1), los coeficientes DC de los 16 bloques de 4×4 que lo componen son de nuevo transformados empleando una transformada de Hadamard cuya expresión matricial se muestra en la Ec. 2-11.

$$Y_D = \left(\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix} \cdot [W_D] \cdot \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix} \right) / 2 \qquad \text{Ec. 2-11}$$

Donde Y_D representa la matriz transformada y W_D es la matriz que contiene los coeficientes DC de los 16 bloques de 4×4 que posee el macrobloque. Después de la transformación se procede a realizar una cuantificación antes de incluir los coeficientes en la trama de salida del codificador. El proceso se muestra gráficamente en la Figura 2-35. Para los coeficientes DC de crominancia se realiza un proceso análogo pero en este caso las matrices de coeficientes son de 2×2 debido al submuestreo de las crominancias. En el descodificador se realiza el proceso inverso para recuperar los coeficientes DC.

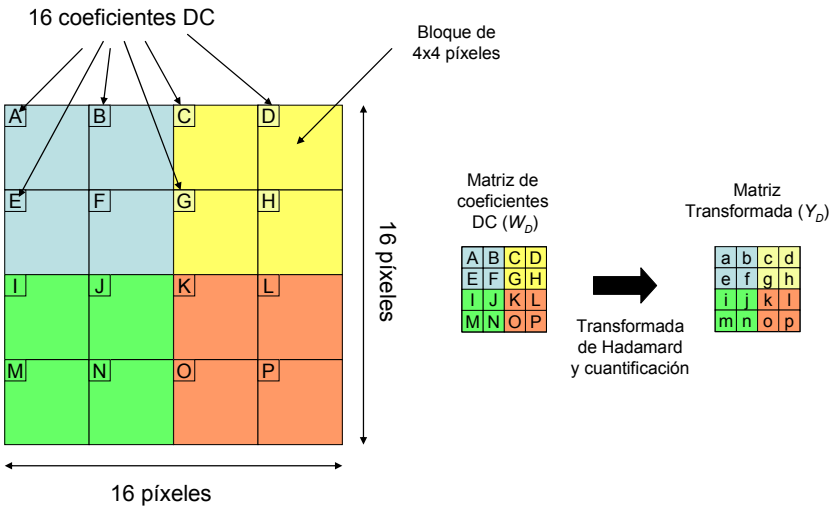


Figura 2-35. Transformación de los coeficientes DC de los bloques que componen un macrobloque I.

2.2.4.2.4.4 Múltiples imágenes de referencia

En los estándares MPEG-2 y MPEG-4 los bloques con predicción de tipo P pertenecientes a una imagen de tipo P o B debían referirse exclusivamente a la imagen tipo P o I inmediatamente anterior. Sin embargo, en el estándar H.264 esta referencia no tiene porqué ser de la imagen inmediatamente anterior sino que puede estar referida a imágenes codificadas anteriormente (aunque temporalmente sean posteriores) y que se encuentran almacenadas en el codificador (y en el

descodificador). La única limitación que impone el estándar es que todas las particiones de un macrobloque siempre estén referidas a una misma imagen de referencia. Además, y dependiendo del nivel, se define un número máximo de imágenes de referencia que pueden almacenarse (ver Tabla 2-9). La Figura 2-36 muestra como la imagen i tiene macrobloques cuyas referencias se encuentran en las imágenes i-1, i-2 e i-3.

Lo mismo que ocurre para los macrobloques tipo P de imágenes tipo P o B, se puede aplicar a macrobloques de tipo B pertenecientes a imágenes de tipo B pero, en este caso, existirán dos referencias[33] a alguna de las imágenes que se encuentran almacenadas en el codificador (y descodificador).

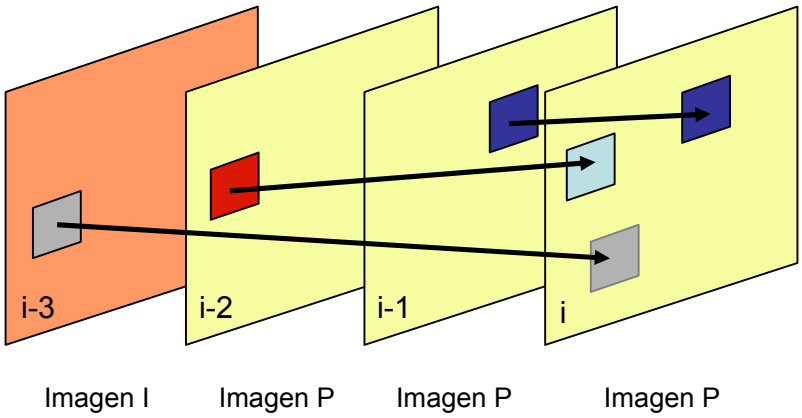


Figura 2-36. Múltiples imágenes de referencia en la predicción de una imagen tipo P.

2.2.4.2.4.5 Métodos para inferir el movimiento

El estándar H.264 incluye algunos modos de predicción que permiten incluir en la trama de salida del codificador información del movimiento dependiente del contexto. Estos modos de predicción (modos *skip* y *direct*) permiten obtener la referencia de un bloque a partir de las predicciones de bloques próximos evitando tener que transmitir los vectores de movimiento correspondientes (*direct*) o los coeficientes asociados al error de predicción (*skip*).

2.2.4.2.4.6 Predicción ponderada (weighted prediction)

Esta herramienta, sólo disponible para el perfil *Main,* permite que cada píxel de referencia de una predicción, ya sea de un macrobloque o bloque P o B, sea escalado por un factor[34] independiente del empleado para el resto de los píxeles. El factor de escala puede transmitirse dentro de la trama de bits (ponderación explícita o *explicit weight*) o depender de la distancia entre la imagen que se está descodificando y la imagen de la que procede la referencia (ponderación implícita o *implicit weight*).

2.2.4.2.5 Otras mejoras

Bajo este epígrafe se resumen otras herramientas incluidas en H.264 que no existen en los estándares MPEG-2 y MPEG-4.

[33] Puede darse el caso de que un macrobloque tipo B sólo disponga de una referencia de una imagen anterior o posterior aunque no suele ser lo habitual.

[34] En el caso de los macrobloques o bloques de tipo B existen dos factores de escala, uno para cada referencia. Esto permite realizar predicciones en las que una de las referencias tiene más “peso” que la otra a la hora de obtener los píxeles del macrobloque o bloque reconstruido.

2.2.4.2.5.1 Predicción intra en el dominio espacial

Habitualmente existe una elevada correlación entre los píxeles próximos de los bloques de 4×4 pertenecientes a macrobloques codificados en modo *intra*. Este hecho permite realizar en algunos casos una predicción en el dominio espacial de un bloque a partir de bloques vecinos.

El estándar H.264 realiza la predicción *intra* con bloques de 4×4 ó 16×16 píxeles. La predicción *intra* es más sencilla que la predicción *inter* puesto que no se realiza un proceso de estimación de movimiento, sino que la predicción se realiza empleando exclusivamente píxeles de la fila inmediatamente superior y/o de la primera columna del vecino de la izquierda[35]. La Figura 2-37 ilustra los píxeles empleados para la predicción *intra* de un bloque de 4×4 pertenecientes a los 4 bloques vecinos, el de la izquierda, el superior-izquierdo, el superior y el superior-derecho.

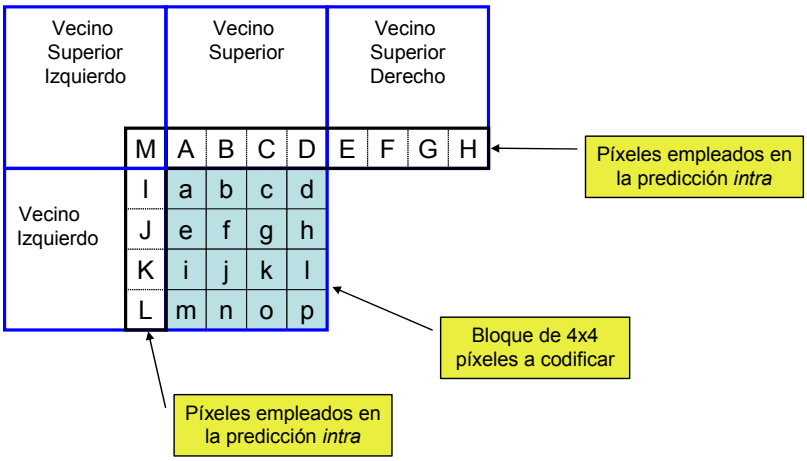


Figura 2-37. Píxeles involucrados en la predicción *intra* de un bloque de 4×4.

La predicción se puede realizar empleando uno de los 9 modos que están definidos para bloques de 4×4 o uno de los 4 modos existentes para bloques de 16×16 píxeles[36]. En el caso de emplear bloques de 4×4, la predicción hace uso de los píxeles del 'A' al 'M' mostrados en la Figura 2-37. El codificador puede evaluar, para cada bloque de 4×4 píxeles de tipo *intra*, cuál de los 9 modos de predicción es el que minimiza una función de error, eligiendo dicho modo para la codificación de ese bloque. La Figura 2-38 muestra los 9 modos disponibles para bloques de 4×4. Así, si el codificador selecciona el modo 1, los píxeles 'I', 'J', 'K' y 'L' se replican respectivamente en todas las filas del bloque[37].

Cuando el codificador selecciona uno de los modos de predicción *intra,* se calcula el residuo como la diferencia entre el bloque predicho y el bloque a codificar.

---

[35] Los píxeles de los bloques vecinos que se emplean no son los que se encuentran en la memoria de reconstrucción puesto que éstos han sido filtrados con el filtro antibloques (ver apartado 2.2.4.2.5.2) mientras que los que se utilizan para la predicción *intra* no deben haber sido filtrados todavía. La necesidad de disponer de estos píxeles provoca que antes de filtrar cualquier bloque sea necesario preservar los píxeles de la línea inferior (píxeles del 'A' al 'H' y el 'M' de la Figura 2-37) y la columna derecha (píxeles del 'I' al 'L' de la Figura 2-37).

[36] Los cuatro modos de predicción para macrobloques de 16×16 píxeles son similares a los modos 0, 1, 2 y 3 descritos para bloques de 4×4 píxeles.

[37] El cálculo de las predicciones para el resto de los modos se realiza de forma similar.

Este residuo es el que se transforma al dominio de la frecuencia y se codifica para ser incluido, junto con el modo de predicción, en la trama de salida[38].

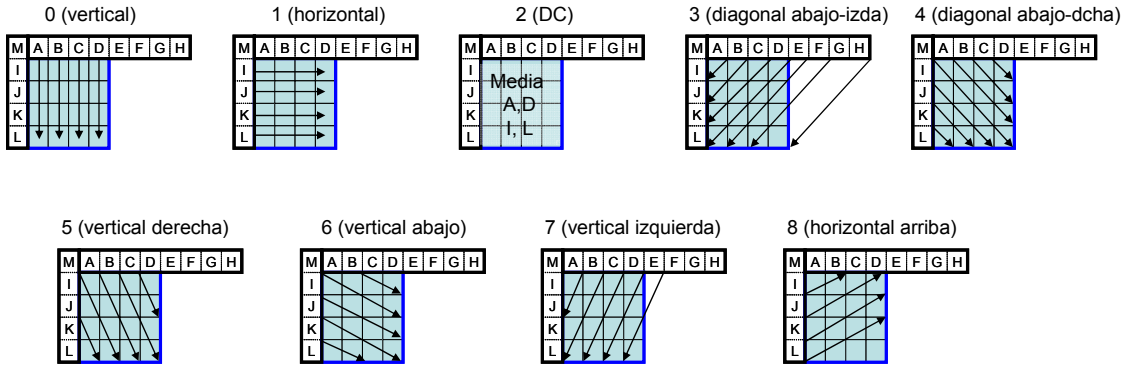


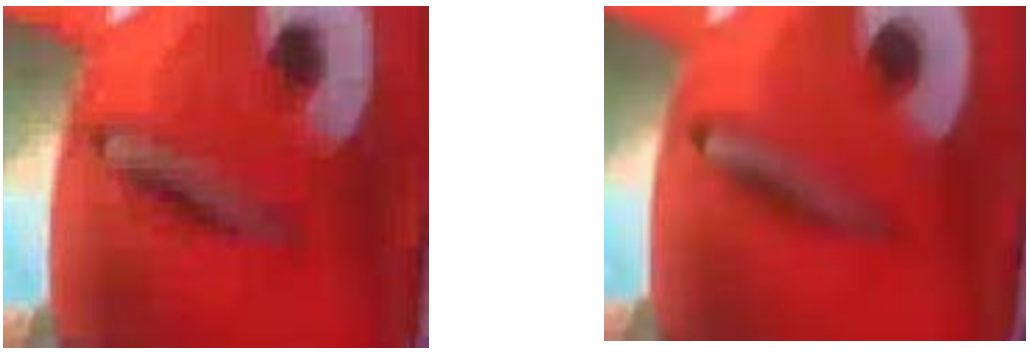
Figura 2-38. Modos de predicción *intra* definidos en H.264 para bloques de 4×4 píxeles.

2.2.4.2.5.2 Filtro antibloques

El uso de bloques por parte del codificador híbrido para procesar las imágenes provoca que aparezca un efecto de borde en los extremos de los bloques. Este efecto es aún más acusado en el estándar H.264 debido, tanto a que existen diferentes tamaños de bloques, como a que la partición en bloques más pequeños de un macrobloque es variable.

Además, este efecto es más apreciable cuando se utilizan factores de cuantificación elevados o cuando se emplean factores de cuantificación diferentes para macrobloques vecinos. En el primero de los casos se suprimen componentes de alta frecuencia, eliminando de este modo parte de los detalles; mientras que en el segundo, se aprecia la diferencia de calidad con la que se han codificado los bloques vecinos.

La Figura 2-39-a muestra un fragmento de una imagen descodificada en la que se aprecia el efecto de bloque comentado anteriormente. Para reducir este problema se aplica un filtro FIR sobre los bordes vertical y horizontal de los bloques de 4×4 píxeles en los que se divide la imagen. En la Figura 2-39-b se muestra el mismo fragmento pero, en este caso, procesado por un descodificador que sí ha realizado un filtrado de los bordes.



a) Imagen sin filtrado antibloques. b) Imagen con filtrado antibloques.

Figura 2-39. Comparativa de una misma imagen empleando o no el filtro antibloques.

[38] El procedimiento cuando se emplean macrobloques de 16×16 píxeles para realizar la predicción es idéntico pero empleando alguno de los 4 modos de predicción que se definen en la norma.

El filtro antibloques definido en H.264 puede aplicarse a todos los bordes de los bloques en los que se descompone un macrobloque cuando se realiza la transformación al dominio de la frecuencia. Dado que la transformada al dominio de la frecuencia se realiza sobre bloques de 4×4 píxeles, en cada macrobloque puede ser necesario aplicar el filtro sobre 4 bordes verticales y 4 horizontales de luminancia y sobre 2 horizontales y 2 verticales de crominancia, tal como muestra la Figura 2-40.

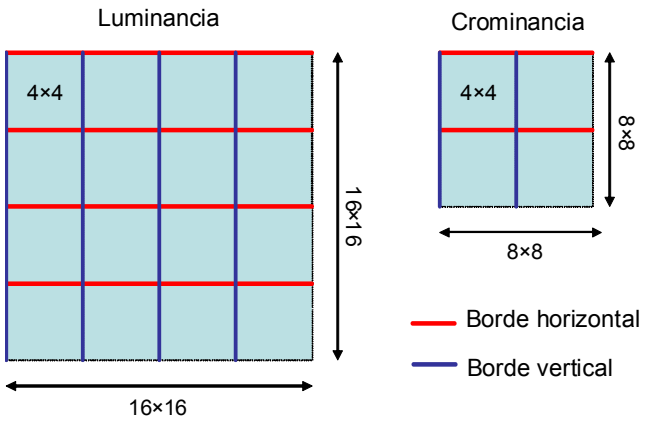
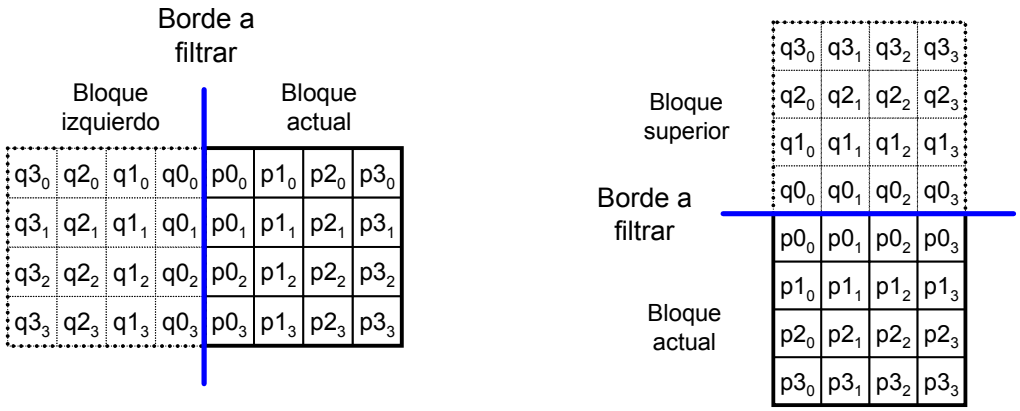


Figura 2-40. Bordes asociados a un macrobloque en los que se aplica el filtro antibloques.

El filtro que se define en el estándar es adaptativo puesto que su aplicación y los coeficientes que se emplean dependen de varias características. El filtro es adaptativo a varios niveles:

a) Adaptativo respecto de las muestras próximas al borde a filtrar. Antes de aplicar el filtro a uno de los bordes es necesario diferenciar si se trata de un borde real de la imagen o un borde debido al efecto de bloque. Para diferenciar estas dos situaciones se analiza el gradiente de las muestras próximas al borde. Para el filtrado vertical de un bloque se tienen en cuenta las muestras del bloque izquierdo mientras que para el filtrado horizontal se emplean las muestras del bloque superior. En ambos casos, y como se muestra en la Figura 2-41, se tienen en cuenta 4 muestras a cada uno de los lados del bloque a filtrar (‘$p0_i$’, ‘$p1_i$’, ‘$p2_i$’, ‘$p3_i$’, ‘$q0_i$’, ‘$q1_i$’, ‘$q2_i$’ y ‘$q3_i$’). Donde i representa cada una de 4 las filas o columnas que forman el borde.



a) Filtrado vertical. b) Filtrado horizontal.

Figura 2-41. Píxeles empleados para realizar el filtrado de un bloque de 4×4 píxeles.

Para calcular el gradiente en cada uno de los bordes se comparan las muestras '$p0_i$' y '$q0_i$', '$p1_i$' y '$p0_i$' y, finalmente, '$q1_i$' y '$q0_i$'. En caso de que esas comparaciones sean menores que unos umbrales, α y β, que dependen del factor de cuantificación (QP) del bloque actual (el cálculo de estos umbrales puede consultarse en [LJL+03]), se procede a realizar el filtrado puesto que se considera que no se trata de un borde real, sino de un borde debido al efecto de bloque.

b) Adaptativo respecto de los tipos de bloques que componen el borde a filtrar. La fuerza con la que se aplica el filtro (*Boundary Strength* o BS) es un valor entre 4 y 0 de modo que, cuanto mayor es este valor, más efecto tiene el filtro sobre las muestras próximas al borde. El cálculo de BS depende de los tipos de bloques que hay en el borde a filtrar tal como muestra la Tabla 2-12[39]. Los coeficientes y número de etapas del filtro que se deben aplicar son diferentes en función del valor de BS que se obtiene y afectan a diferentes píxeles del borde que se está filtrando.

Modo de cada bloque y condiciones	BS
Uno de los bloques es *intra* y se trata de un bloque en el borde de un macrobloque	4
Uno de los bloques es *intra*	3
Uno de los bloques tiene asociados coeficientes en el dominio de la frecuencia	2
La compensación de movimiento proviene de más de una imagen de referencia	1
La diferencia entre los vectores de movimiento de los dos bloques es mayor que una muestra entera	1
Resto de situaciones	0

Tabla 2-12. Asignación de BS en función del tipo de bloques existentes alrededor del borde a filtrar.

c) Adaptativo respecto del tipo de imagen. El codificador puede asignar unos valores de *offset*, tanto para α, como para β que permiten modificar el comportamiento del filtro. Las imágenes con un factor de cuantificación elevado, en las que el efecto de borde es más acusado, requieren valores positivos para estos *offsets*; mientras que las secuencias con gran calidad (factor de cuantificación bajo), en las que no es muy necesario aplicar filtrado de los bloques puesto que éste puede provocar un efecto de difuminación de los bordes reales, se aplican valores negativos para los *offsets*.

Como resultado de aplicar el filtro antibloques se produce una reducción del régimen binario para una relación señal/ruido de pico (*Peak Signal Noise Ratio* o PSNR) constante. En [LJL+03] se demuestra que la reducción del régimen binario se encuentra en torno al 9%.

[39] A la hora de calcular el parámetro BS se analizan las condiciones en el orden mostrado en la tabla. Cuando se cumple alguna de las condiciones, se asigna BS y termina el proceso.

2.2.4.2.6 La trama H.264

La estructura sintáctica de la trama H.264 está formada por un conjunto de paquetes denominados unidades NAL (*Network Abstraction Layer*), compuestos por una cabecera y una zona de datos (*Raw Byte Sequence Payload* o RBSP) tal como muestra la Figura 2-42. La zona de datos de las unidades NAL puede contener dos tipos de información: unidades VCL (*Video Coding Layer*), que transportan la secuencia de vídeo codificada, o unidades no VCL, que transportan información adicional necesaria en el proceso de descodificación.

Por tanto, una secuencia de vídeo codificada está formada por una colección de unidades NAL que pueden ser enviadas a través de una red orientada a la transmisión de paquetes o almacenadas en un fichero. La separación entre VCL y NAL permite diferenciar la información relativa a la codificación del vídeo, de la información específica del transporte de la trama.

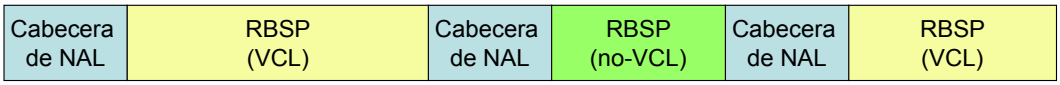


Figura 2-42. Trama H.264 formada por tres unidades NAL.

Las unidades no VCL imprescindibles para poder descodificar la secuencia son la SPS (*Sequence Parameter Set*) y la PPS (*Picture Parameter Set*). La primera transporta información válida para toda la secuencia de imágenes como puede ser el perfil, el nivel, las dimensiones, el formato de muestreo, etc. Por otra parte, la segunda incluye datos relativos a una o varias imágenes individuales como puede ser la codificación de entropía empleada o el uso del modo de transformación 8×8, entre otros.

Las unidades VCL son las encargadas de transportar la secuencia de vídeo codificada organizada en rebanadas. En la zona útil de datos de cada unidad VCL se encuentra la información de los macrobloques. Para cada uno de ellos se da información del tipo de macrobloque, el parámetro de cuantificación (Q), el modo de predicción (*Coded Block Pattern* o CBP), los vectores de movimiento asociados (MV) y la información residual resultado de su codificación. La Figura 2-43 muestra la estructura jerárquica de una NAL que contiene una unidad VCL.

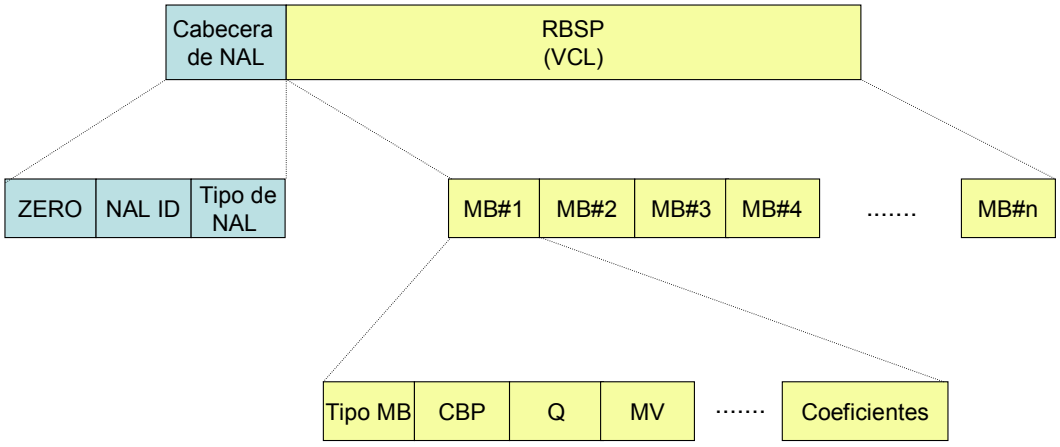


Figura 2-43. Estructura jerárquica de una trama H.264.

2.3 Comparativa entre los estándares de codificación de vídeo

En este subapartado se presenta una comparación entre los tres estándares que se han manejado durante los trabajos realizados a lo largo de esta tesis. Esta comparación se realiza desde un doble enfoque: por un lado, se buscan las similitudes entre las herramientas de codificación que emplea cada uno de ellos, que permitirán reutilizar la implementación de algunos módulos; y por otro, se presenta la eficiencia de los codificadores que implementan cada uno de los estándares en términos de reducción del régimen binario para una determinada calidad de codificación.

2.3.1 Herramientas de codificación

La Tabla 2-13 recoge todas las herramientas de codificación que utilizan los estándares presentados en este capítulo. Analizando las herramientas empleadas por cada uno de los estándares, se aprecia que algunas de ellas son compartidas por más de uno, lo que justifica el interés de investigar en metodologías de optimización que se puedan aplicar a todos ellos.

A modo de resumen se deduce que MPEG-2 MP@ML, MPEG-4 ASP, H.264 BP y H.264 MP tienen, desde un punto de vista general, las siguientes similitudes:

- La codificación está basada en macrobloques y bloques en todos los estándares. Además, H.264 incluye un particionado de los bloques en estructuras de menor tamaño.
- En todos los casos se emplea un esquema de muestreo 4:2:0 (en los binomios perfil/nivel de interés para la tesis).
- La trama de bits que genera cualquiera de los codificadores está compuesta por una serie de cabeceras que hay que analizar a nivel de bit y los datos que componen la imagen codificada.
- Todos los estándares emplean un esquema de codificación híbrido basado en transformación al dominio de la frecuencia, cuantificación, estimación y compensación de movimiento y codificación de longitud variable, en el cual:
    - MPEG-2 y MPEG-4 emplean la DCT mientras que H.264 utiliza la ICT. A pesar de ello ambas transformaciones son similares.
    - La estimación y compensación de movimiento se realiza de forma similar con algunas particularidades, como el modo en que se obtienen los píxeles intermedios para la estimación de ½ y de ¼ píxel.
    - El proceso de cuantificación es muy similar en los tres estándares.
    - Los tipos de imágenes que se emplean en los tres estándares son los mismos, I, P y B[40].

Sin embargo, a pesar de las similitudes se aprecian algunas diferencias significativas entre las que cabe destacar las siguientes:

- La codificación de entropía en MPEG-2 y MPEG-4 está basada en códigos VLC[41]; en H.264 BP se emplea codificación de longitud variable adaptativa

[40] H.264 en su perfil *Baseline* no permite el uso de imágenes tipo B. Por otro lado, en MPEG-4 se define el tipo de imagen S-(GMC) aunque no es ampliamente utilizado.

[41] A pesar de que MPEG-2 y MPEG-4 utilizan códigos VLC, las tablas que se emplean son diferentes por lo que el código que se emplea para uno de los estándares, no es válido para el otro.

con el contexto, mientras que para H.264 MP se utiliza codificación aritmética adaptativa.

- El filtro antibloques sólo aparece en el estándar H.264.
- La predicción *intra* para los macrobloques de tipo I sólo está disponible en el estándar H.264.

Elemento	Herramienta	MPEG-2	MPEG-4 ASP	H.264 BP	H.264 MP
Codificación de Entropía	VLC	X	X		
	RVLC		X		
	CAVLC			X	X
	CABAC				X
	Exp Golomb			X	X
Transformación al dominio de la frecuencia	DCT	X	X		
	ICT			X	X
Cuantificación	Matrices de cuantificación con pesos			X	X
Compensación de Movimiento	Precisión ½ píxel	X	X	X	X
	Precisión ¼ píxel		X	X	X
	Tamaño de los bloques variable			X	X
	Predicción *Intra*			X	X
	Compensación Global de Movimiento		X		
	Imágenes tipo B	X	X		X
	Múltiples imágenes de referencia			X	X
	Uso de imágenes B como referencia				X
Otros	Filtro antibloques			X	X
	Codificación sin pérdidas			X	X

Tabla 2-13. Herramientas de codificación empleadas en cada uno de los estándares.

2.3.2 Calidad *versus* régimen binario

Para poder comparar el rendimiento de los diferentes estándares se puede emplear el trabajo publicado en [WSJ+03], en el que tomando como referencia el estándar H.264/AVC (perfil *Main*) se comparan las características de éste con MPEG-2, H.263 y MPEG-4.

En la parte izquierda de la Figura 2-44 se representa el PSNR de la luminancia (Y-PSNR) en función del régimen binario de la secuencia de pruebas empleada[42]. Se observa que, para un régimen binario dado, el PSNR para H.264 se encuentra, en todo momento, al menos 2 dB por encima del resto de los estándares. En la parte derecha se representa la reducción del régimen binario, en relación al estándar MPEG-2 y en función del Y-PSNR. Se observa que para una calidad en torno a 32 dB se produce una reducción del régimen binario del 20% para H.263, del 45% para MPEG-4 ASP y del 70% para H.264.

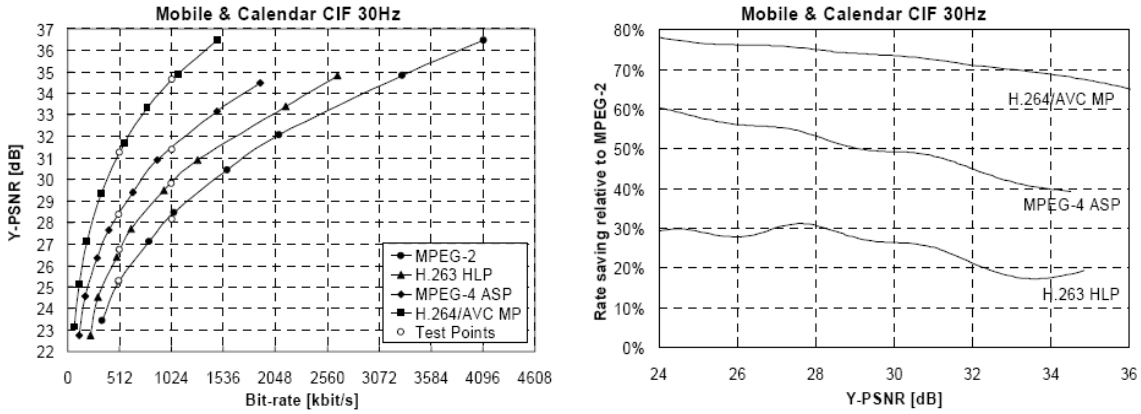


Figura 2-44. Relación entre el régimen binario y el PSNR de la luminancia para diversos estándares de codificación (extraídas de [WSJ+03]).

En la Tabla 2-14 se muestra la reducción media, en tanto por ciento, del régimen binario de salida de los diversos codificadores. Para realizar esta medida se ha codificado una misma secuencia, con todos los estándares, con una calidad fija en términos de Y-PSNR. Analizando el régimen binario necesario en cada caso se extraen los resultados de la tabla.

Estándar	MPEG-4 ASP	H.263	MPEG-2
H.264	38.6%	48.8%	64.5%
MPEG-4 ASP		16.6%	42.9%
H.263			30.6%

Tabla 2-14. Reducción media del régimen binario empleando los diferentes estándares [WSJ+03].

2.4 Resumen

Para el desarrollo de esta tesis ha sido necesario realizar un estudio exhaustivo de los estándares de codificación de vídeo MPEG-2, MPEG-4 y H.264 tratando de identificar aquellos elementos comunes que permiten plantear metodologías de optimización válidas para todos ellos.

[42] Para la realización de las medidas se ha empleado la secuencia *"Mobile & Calendar"* con tamaño CIF codificada a 30 imágenes por segundo. En el artículo se presentan resultados similares para otras secuencias con resoluciones espaciales y temporales diferentes.

A lo largo de la explicación que se ha realizado de cada uno de los estándares se han expuesto aquellos aspectos que se han considerado más interesantes desde el punto de vista de la optimización del tiempo de ejecución de los mismos. Sobre estos aspectos se incidirá en el capítulo 4 a la hora de plantear las técnicas de optimización que se muestran y que han permitido sintetizar la metodología propuesta en el capítulo 5.

Como consecuencia del estudio realizado, en la Tabla 2-13, se establece una comparación entre todos los estándares analizados en la que se aprecia que existen grandes similitudes entre ellos, lo que garantiza que las técnicas de optimización ensayadas con éxito en un estándar, serán directamente aplicables al resto, permitiendo establecer una metodología de carácter general que redunde en una reducción significativa del tiempo invertido en su optimización.

Adicionalmente, y con el objetivo de que la memoria sea autocontenida, se ha introducido buena parte de la terminología que se empleará en el resto de capítulos.

3 ANÁLISIS DEL ESTADO DEL ARTE EN LA OPTIMIZACIÓN DE ALGORITMOS CODIFICACIÓN DE VÍDEO

En este capítulo se realiza una revisión del estado del arte en dos campos clave para el desarrollo de la tesis: los DSPs de última generación y las metodologías de optimización de codificadores y descodificadores de vídeo basados en DSPs.

Para hacer este capítulo más autocontenido, en el apartado 3.1 se hace una revisión de las diferentes alternativas tecnológicas que se han venido utilizando en los últimos años para la implementación de codificadores y descodificadores de vídeo para diferentes estándares. El soporte tecnológico elegido para las implementaciones que se llevan a cabo en la tesis, los DSPs, se enmarca en este apartado dentro de las alternativas tecnológicas existentes, y se caracteriza como una opción consolidada y ventajosa, sobre todo desde el punto de vista de la flexibilidad.

En el apartado 3.2 se analiza un conjunto de DSPs de última generación, de diferentes fabricantes, con los cuales se han realizado en los últimos años una gran cantidad de implementaciones de codificadores y descodificadores de diferentes estándares. Las similitudes que se ponen de manifiesto en este apartado entre las diferentes arquitecturas permiten pensar que las metodologías de optimización que se proponen en la tesis se podrían aplicar al diseño con un conjunto amplio de procesadores.

En el apartado 3.3 se revisan las implementaciones de codificadores y descodificadores de vídeo para diferentes estándares publicadas en la literatura científica por diferentes autores en los últimos años, que utilizan arquitectura DSP. En este análisis se hace hincapié en las metodologías de optimización y en los resultados (prestaciones) que se han alcanzado utilizando dichas metodologías.

Finalmente, en el apartado 3.4, se realiza un resumen de los contenidos del capítulo.

3.1 Alternativas tecnológicas para la realización de sistemas de codificación y descodificación de vídeo

La codificación y la descodificación de vídeo son aplicaciones muy exigentes que requieren soportes tecnológicos con una capacidad de cómputo elevada. En la actualidad, las diferentes alternativas tecnológicas que pueden utilizarse para implementar este tipo de aplicaciones pueden clasificarse en tres grandes grupos: las basadas en procesadores de propósito general, las basadas en arquitecturas específicas y las basadas en DSPs. En los apartados 3.1.1, 3.1.2 y 3.1.3 se resumen algunas de las propuestas más representativas pertenecientes a cada grupo que se han publicado en la literatura científica durante los últimos años.

3.1.1 Alternativas basadas en procesadores de propósito general

Las primeras implementaciones de los diversos estándares de codificación suelen desarrollarse sobre procesadores de propósito general, estando normalmente programadas en lenguaje C o C++. Ejemplos claros son las implementaciones de codificadores y descodificadores que se realizan dentro de los propios estándares [ISO05], [ISO01]. Estas implementaciones, que no se encuentran optimizadas para ninguna arquitectura concreta, suelen ser el punto de partida para el desarrollo de las primeras versiones de codificadores o descodificadores.

Los ordenadores personales son un claro ejemplo de sistemas basados en procesadores de propósito general. Estos sistemas presentan una alta capacidad de cómputo y elevados recursos de memoria, características que les permiten codificar o descodificar secuencias de video en tiempo real a pesar de que el *software* no se encuentre optimizado para la arquitectura que poseen. Por el contrario, tanto el coste económico como el consumo de energía que requieren, hacen que no sean apropiados para trabajar, por ejemplo, con sistemas móviles en los que ambos factores deben ser minimizados.

Por otra parte, existen otros procesadores de propósito general, como por ejemplo los ARM, que son capaces de implementar codificadores y descodificadores de vídeo. A diferencia de los procesadores que contienen los ordenadores personales, estos procesadores poseen una capacidad de cálculo y una cantidad de memoria muy limitada por lo que no son capaces de codificar o descodificar en tiempo real secuencias de elevada resolución. Por tanto, este tipo de procesadores no son adecuados para desarrollar sistemas con elevadas resoluciones de pantalla, como por ejemplo, un *Set Top Box* para televisión digital terrestre.

En los siguientes subapartados se resumen algunos ejemplos de las propuestas presentadas en publicaciones científicas sobre implementaciones de descodificadores compatibles con los diferentes estándares empleando procesadores de propósito general. Inicialmente se muestran desarrollos sobre procesadores Intel, posteriormente sobre procesadores ARM y se finaliza describiendo las que emplean procesadores CELL.

3.1.1.1 Alternativas basadas en procesadores Intel

En [LMS04] investigadores de la Universidad Nacional de Seúl presentan un descodificador H.264 implementado sobre un Pentium 4 en el que se optimiza el código para hacer uso de las instrucciones MMX (*MultiMedia eXtension*). Estas instrucciones permiten explotar el paralelismo a nivel de datos que poseen los algoritmos de procesamiento de vídeo, de modo que con una única instrucción es posible realizar varias operaciones en paralelo con diferentes datos. Empleando estas instrucciones, los autores han optimizado los módulos de interpolación de luminancia y crominancia, la ICT inversa y el filtro antibloques del código de referencia del

descodificador (versión JM-7.2). Con estas optimizaciones, el tiempo de ejecución de algunos módulos se ha reducido desde un 70%, para la interpolación, hasta un 50%, para el filtro antibloques. Los autores sólo facilitan información de la mejora de los módulos optimizados, pero no del número de imágenes por segundo que es capaz de descodificar. Para los módulos optimizados (interpolación, transformada inversa y filtro antibloques) se indica que la versión sin optimizar requiere $130 \cdot 10^6$ ciclos de reloj para una imagen con definición estándar, mientras que la versión optimizada emplea $55 \cdot 10^6$.

En 2005, Quan y otros investigadores de la Universidad Hangzhou en China [QJSJ05] presentan algunas optimizaciones realizadas sobre el descodificador de referencia H.264 (versión JM-6.1) para el perfil *Baseline* que permiten reducir el tiempo de descodificación medio de las secuencias a la mitad empleando un procesador Intel Pentium a 2 GHz. Las optimizaciones que se han realizado afectan a la extracción de los coeficientes CAVLC, la compensación de movimiento y el filtro antibloques. Como resultado se ha pasado de descodificar 20 imágenes por segundo de resolución CIF con la versión original, a 65 con la versión optimizada.

Posteriormente, la compañía Emuzed presenta en [PMK+07] una implementación optimizada de un descodificador multiformato (MPEG-2, VC-1 y H.264) para un procesador de Intel denominado Bulverde [BUL04], orientado al desarrollo de aplicaciones inalámbricas. Se trata de un procesador de bajo consumo basado en la arquitectura XScale de Intel al que se le han añadido instrucciones SIMD que permiten operar con varios datos en paralelo. Aprovechando estas instrucciones se han optimizado módulos comunes a todos los estándares de codificación como son la transformada inversa y la predicción *inter,* y otros exclusivos de H.264 como son el filtro antibloques y la predicción *intra*. Como resultado de estas optimizaciones el número de imágenes descodificadas por segundo se ha incrementado en un 75% para MPEG-4, un 70% para VC-1 y un 100% para H.264. Los autores no indican el número de imágenes por segundo que es posible descodificar para cada estándar, pero sí que con el estándar VC-1 se descodifican un 50% más de imágenes que con H.264 y que con MPEG-4 es posible procesar el doble de imágenes que con H.264.

3.1.1.2 Alternativas basadas en procesadores ARM

En [KHCL07] investigadores de la Universidad Nacional Cheng Kung de Taiwán proponen algunas modificaciones en el *software* de referencia del descodificador H.264 BP para optimizar los accesos a memoria externa. Para ello emplean *buffers* intermedios en los que se almacena información que será reutilizada en fases posteriores del proceso de descodificación de cada macrobloque. El código optimizado ha sido ejecutado en un procesador ARM incrementando el número de imágenes descodificadas por segundo en un factor de 3.6. Con estos resultados los autores indican que, para lograr el funcionamiento en tiempo real con secuencias en formato QCIF, es necesario disponer de un procesador que proporcione 45 MIPS para la versión original y 12 MIPS para la versión optimizada.

Por otro lado, investigadores del Instituto de Tecnología Industrial de Taiwán muestran en [HJC+08] varias optimizaciones del *software* de referencia de H.264 para reducir el tiempo de descodificación. Las principales mejoras están orientadas a minimizar el número de accesos a memoria, sustituir operaciones aritméticas por desplazamientos, detectar de forma rápida bloques sin codificar y optimizar el cálculo del *Boundary Strength*. El descodificador optimizado ha sido ejecutado en un procesador ARM926EJS logrando el funcionamiento en tiempo real para secuencias de tamaño QCIF, con una frecuencia de reloj de 120 MHz.

3.1.1.3 Alternativas basadas en el procesador CELL

CELL es un procesador heterogéneo multinúcleo [Pha05] compuesto por un procesador de propósito general denominado PPE (*Power Processor Element*) y ocho procesadores especializados o SPEs (*Synergistic Processor Element*) que contienen una unidad de procesamiento VLIW (*Very Long Instruction Word*) con 256 kB de memoria local y un controlador de memoria (MFC). La Figura 3-1 muestra el diagrama de bloques del procesador CELL.

La empresa Toshiba Corporation muestra en [KTM+06] la implementación de un sistema de descodificación de televisión digital con vídeo multiformato (MPEG-2, MPEG-4 y H.264) empleando un procesador CELL. La carga computacional del sistema ha sido distribuida entre los diversos procesadores logrando el funcionamiento en tiempo real[43] para definición estándar con un régimen binario de 2 Mbps en formato H.264 MP.

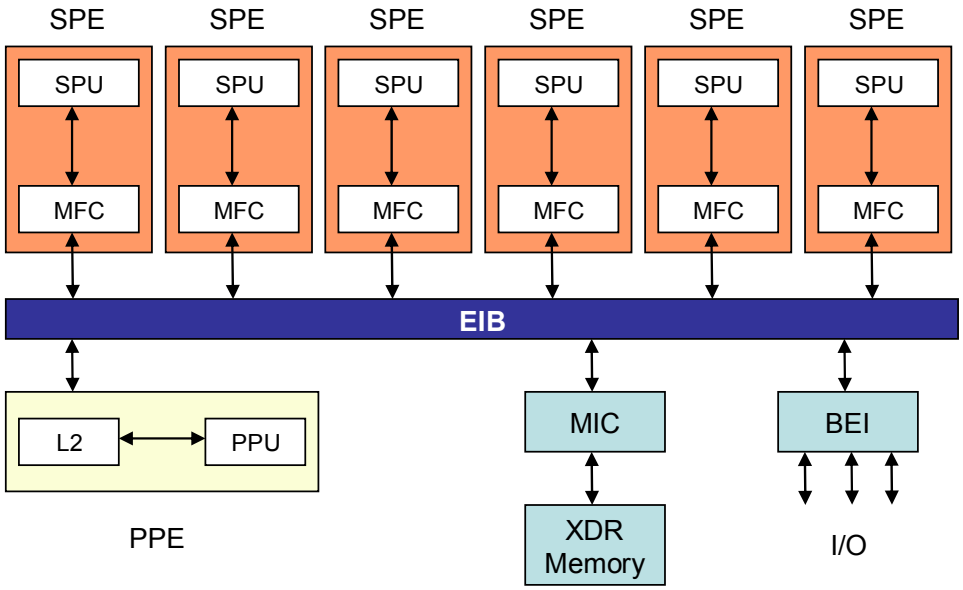


Figura 3-1 Diagrama de bloques del procesador CELL [KTM+06].

En [MYN+06] investigadores de la Universidad de Edimburgo presentan los resultados obtenidos tras migrar la librería *ffmpeg* [FFMPEG] que implementa un descodificador H.264 a un procesador CELL. El resultado de la migración es comparado en términos de tiempo de ejecución y consumo de energía con implementaciones basadas en ARM [TIVR], FPGA [4i2i05] o ASIC [KJB+04]. La Tabla 3-1 muestra los resultados de la comparación en la que se aprecia que el consumo del descodificador basado en CELL es inferior al del resto; sin embargo, el tiempo de descodificación sólo es menor que el de la solución basada en ARM, siendo muy superior al de la implementación basada en ASIC. El descodificador optimizado es capaz de procesar entre 12 y 21 imágenes por segundo con definición estándar dependiendo de las características de las secuencias.

[43] Los autores no indican en el artículo la frecuencia de reloj a la que funciona el procesador CELL.

Plataforma	Rendimiento Macrobloques/s	Energía µJ/Macrobloque
ARM [TIVR]	7500	7.33
FPGA [4i2i05]	79200	15.64
ASIC [KJB+04]	243000	2.28
[MYN+06]	1688	1.49

Tabla 3-1. Comparación del descodificador implementado en [MYN+06] con el procesador CELL y otros codificadores basados en diferentes tecnologías.

3.1.2 Alternativas basadas en arquitecturas específicas

A lo largo de los últimos años se han presentado multitud de implementaciones de descodificadores conformes con los diferentes estándares empleando arquitecturas especialmente diseñadas para minimizar el tiempo de descodificación y en algunos casos el consumo. La mayor parte de ellas se han prototipado sobre FPGAs, realizando en algunos de los casos una síntesis con tecnología ASIC.

La ventaja fundamental que presentan estas arquitecturas es su elevada eficiencia puesto que son capaces de descodificar secuencias de vídeo en tiempo real empleando una frecuencia de reloj baja y con un reducido consumo. Sin embargo, como contrapartida, estas implementaciones son poco flexibles puesto que una vez fabricadas son difícilmente adaptables a modificaciones que se produzcan en el estándar que implementan o a la inclusión de los nuevos estándares que van apareciendo.

En los siguientes subapartados se presentan algunos ejemplos que se han considerado representativos de este tipo de arquitecturas específicas[44].

3.1.2.1 Procesador MCP (*Multimedia Communication Processor*) de *Mitsubishi*

La compañía *Mitsubishi* presenta en [MMT+01] el procesador de señal MCP que incorpora un núcleo que realiza tareas de codificación/descodificación de vídeo. Dicho núcleo, tal y como se muestra en la Figura 3-2, se compone de un procesador RISC, un coprocesador para la codificación/descodificación de longitud variable, una unidad de operaciones matriciales especializada en realizar operaciones a nivel de bloque de píxeles y un controlador de DMA. Según se estima en [KOS00] este procesador es capaz de codificar/descodificar simultáneamente 30 imágenes/segundo en formato CIF[45] empleando el estándar MPEG-4.

[44] Además de las implementaciones que se muestran en este apartado, existen otras que están siendo comercializadas por diferentes fabricantes pero de las que no existe una documentación detallada sobre la arquitectura interna del circuito. Por este motivo se ha optado por no incluirlas en este apartado.

[45] Los autores no indican en este trabajo la frecuencia del procesador.

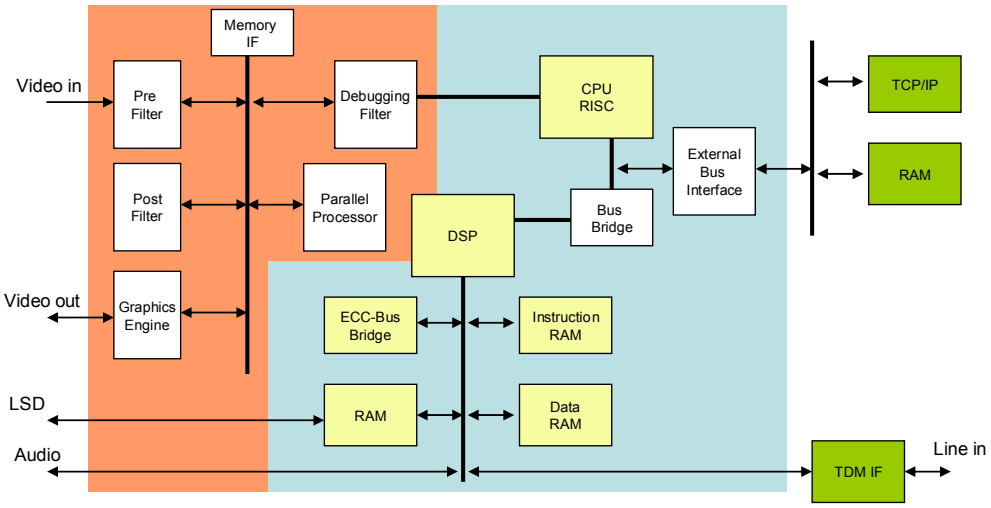


Figura 3-2 Diagrama de bloques del procesador MCP [MMT+01].

3.1.2.2 NEC uPD77210

En [HMK02] desarrolladores de NEC presentan un descodificador de bajo consumo para MPEG-4 SP que es capaz de descodificar 15 imágenes por segundo en formato QCIF trabajando a una frecuencia de 160 MHz y con un consumo de 80 mW. El procesador empleado para este desarrollo, cuyo diagrama de bloques se presenta en la Figura 3-3, no está orientado a aplicaciones multimedia por lo que no incluye ninguna funcionalidad específica para el proceso de codificación/descodificación.

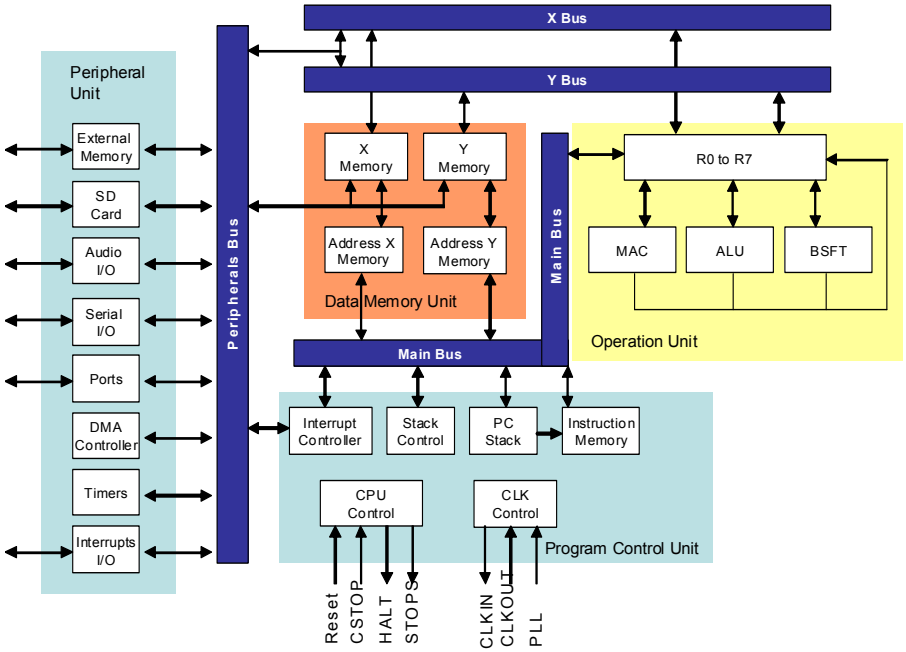


Figura 3-3 Diagrama de bloques del procesador uPD77210 presentado en [HMK02].

3.1.2.3 HiBRID-Soc

Moch y otros investigadores de la Universidad alemana de Hannover proponen en [MBS+03] un procesador que contiene tres núcleos de procesado orientados a la codificación/descodificación de vídeo, comunicados por un bus AMBA (*Advanced Microcontroller Bus Architecture*). De los tres procesadores, uno está dedicado al

procesado de macrobloques, otro al tratamiento del *stream* de entrada/salida y el último a operaciones habituales como filtrado, DCT/IDCT, etc. Sobre este dispositivo, cuya arquitectura se presenta en la Figura 3-4, se ha desarrollado un descodificador MPEG-4 ASP que es capaz de procesar en tiempo real[46] 25 imágenes por segundo de 720×576 píxeles.

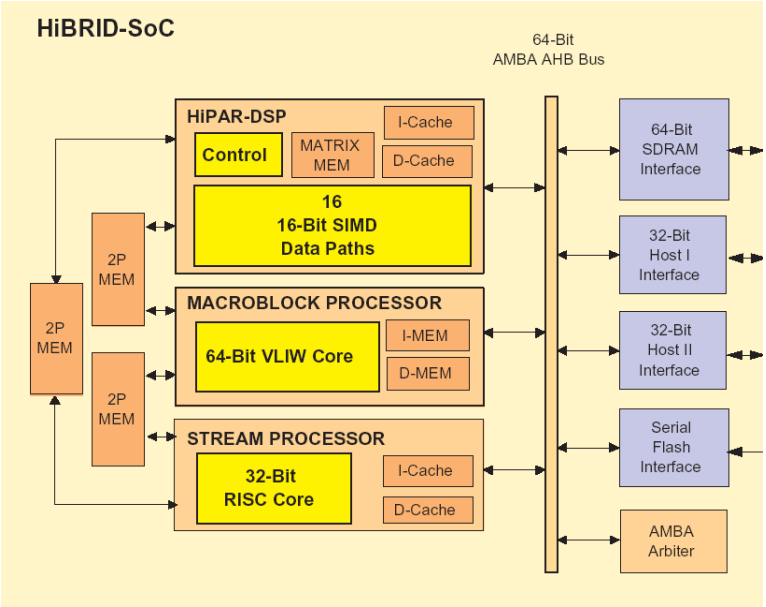


Figura 3-4 Diagrama de bloques del procesador HiBRIC-Soc [MBS+03].

3.1.2.4 MPEG-4 *Multimedia decoder chip*

En [SBPR01] investigadores de la Universidad de Hannover en colaboración con ingenieros de la empresa Robert Bosch presentan un descodificador conforme con MPEG-4 ASP basado en un procesador RISC de propósito general (ARM) y un DSP con arquitectura VLIW diseñado específicamente para el procesamiento de macrobloques (MBE). Ambos sistemas de procesamiento se comunican mediante una memoria compartida como se muestra en la Figura 3-5. El sistema sintetizado ocupa 25 mm^2 empleando tecnología de 0.13 µm y funciona en tiempo real con resolución estándar para una frecuencia de reloj de 200 MHz.

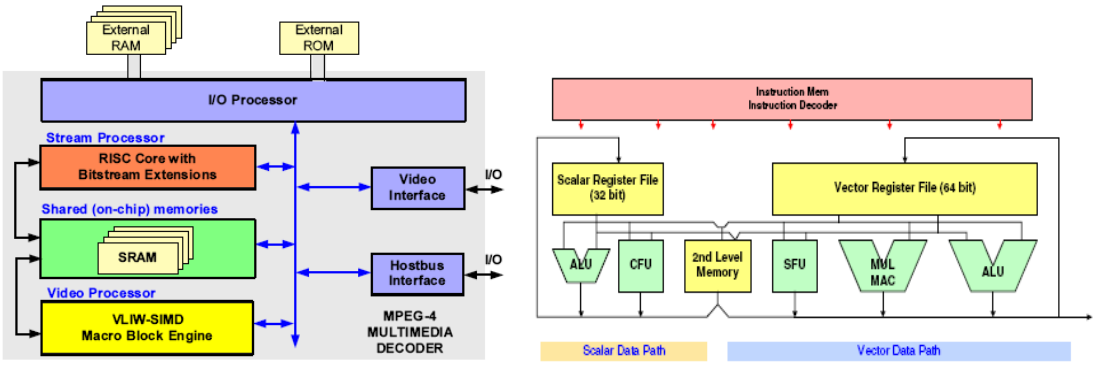


Figura 3-5 Diagrama de bloques del descodificador MPEG-4 y del procesador de macrobloques MBE diseñado con arquitectura VLIW [SBPR01].

[46] Los autores de este trabajo no indican la frecuencia a la que trabaja el procesador.

3.1.2.5 Descodificadores H.264 basados en procesador de propósito general y arquitectura *hardware* específica

En este subapartado se presentan 6 arquitecturas similares que implementan descodificadores H.264 de bajo consumo empleando un procesador RISC de propósito general y un *hardware* específico para acelerar el proceso de descodificación. A continuación se describen someramente, y en orden cronológico, las arquitecturas propuestas por los diversos grupos de investigación:

a) En [KJB+04] investigadores del *Semiconductor Laboratory* en Corea proponen la arquitectura que se muestra en la Figura 3-6 basada en un procesador de tipo ARM conectado mediante un bus AMBA a una serie de procesadores específicos que implementan los diversos módulos funcionales de un descodificador H.264 (CAVLC, IDCT, predicción INTER/INTRA, filtro antibloques, etc.). La arquitectura ha sido optimizada para minimizar el ancho de banda en el acceso a la memoria empleando memorias de doble puerto. La arquitectura se encuentra segmentada en cuatro etapas en las que la unidad de procesamiento de cada una de ellas es el macrobloque y una quinta etapa de filtrado antibloques en la que se procesa toda la imagen. En la primera etapa se analiza la trama (VLD y MVP), en la segunda se calcula la IDCT y se lee la referencia (IDCT/Q), en la tercera se realiza la predicción *inter* o *intra* (INTRA/INTER); y, en la cuarta y última (WR), se escribe el resultado de la memoria de imágenes reconstruidas. Una vez realizado todo el procesamiento para una imagen completa se aplica el filtro antibloques (DF).

La descodificador soporta el perfil *Baseline*, para una resolución de pantalla 1080HD funcionando a 130 MHz. El diseño ha sido sintetizado empleando tecnología de 0.13 µm ocupando un área de 13.7 mm^2 y con un consumo de 554 mW para secuencias en formato 1080HD.

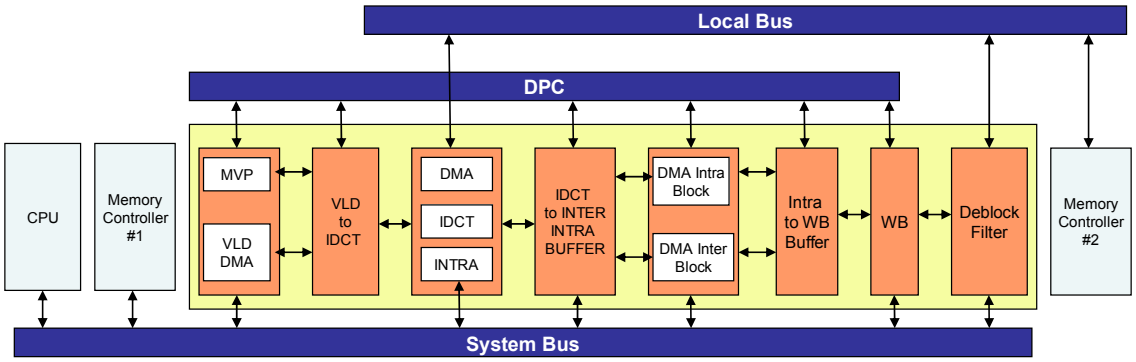


Figura 3-6 Diagrama de bloques del procesador propuesto en [KJB+04].

b) Hu y otros investigadores de la compañía Conexant muestran en [HSMC04] una arquitectura para descodificar en tiempo real secuencias de alta definición en formato H.264 MP. En esta propuesta se emplea un procesador RISC que gobierna un bus de tipo AMBA al que se han conectado una serie de módulos diseñados específicamente para el procesamiento de una secuencia en formato H.264. Los módulos diseñados, mostrados en la Figura 3-7, permiten la descodificación de entropía y el cálculo de la IICT (*Byte Stream Decoder*), la predicción *intra* (*Pixel Decode Module*), la predicción *inter* (*Inter Prediction*) y el filtro antibloques (*Deblocking Filter*). Estos bloques gestionan dos *buffers* de memoria para almacenar las imágenes previamente descodificadas (*Decoded Picture Buffer* o DPB) e información de macrobloques vecinos (*Row Store Buffer* o RSB).

La gestión de los *buffers* de memoria ha sido optimizada para reducir el ancho de banda en los accesos a memoria reutilizando datos previamente leídos que se almacenan en *buffers* en memoria interna y realizando lecturas de píxeles y coeficientes alineados en memoria a 32 ó 64 bits.

El descodificador ha sido sintetizado con tecnología de 130 nm requiriendo un total de 300 K puertas equivalentes, 74 kB de memoria interna y 24 MB de memoria externa. El consumo, descodificando secuencias de alta definición, es de 160 mW para una frecuencia de reloj de 200 MHz.

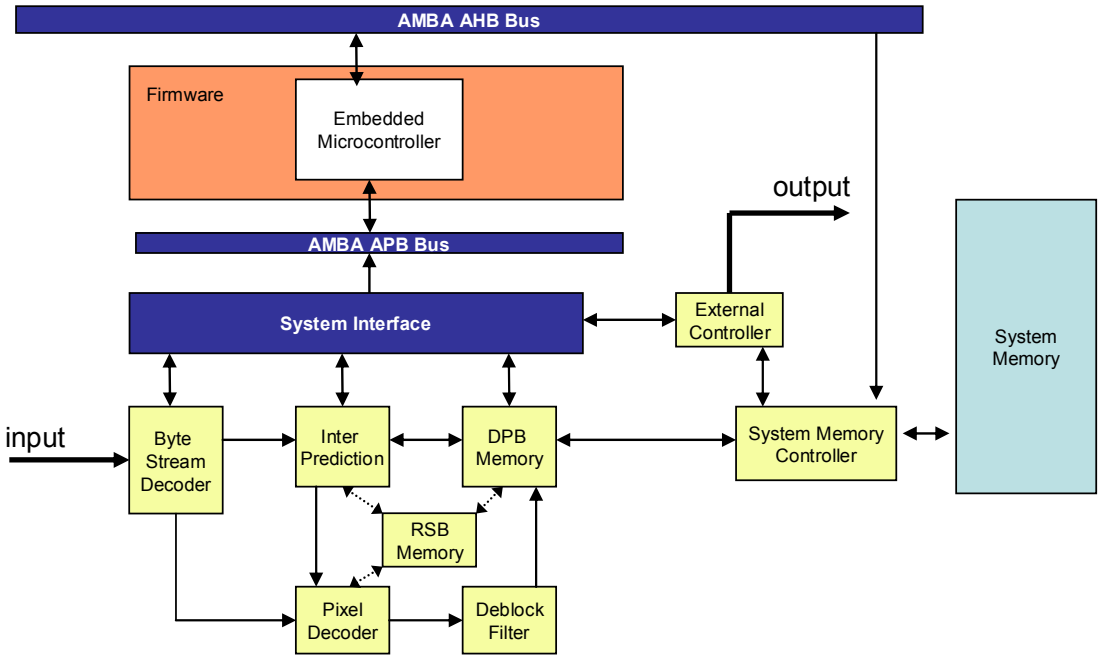


Figura 3-7 Diagrama de bloques del descodificador H.264 propuesto en [HSMC04].

c) En [LLWL06] varios autores de la Universidad Nacional Chiao Tung de Taiwán presentan un descodificador de muy bajo consumo para aplicaciones móviles que es capaz de descodificar secuencias H.264 BP en formato QCIF con una frecuencia de reloj de 1.2 MHz y un consumo de 865 µW.

El descodificador está basado en la arquitectura que se muestra en la Figura 3-8 y realiza el procesamiento a nivel de bloques de 4×4 píxeles. Al igual que las arquitecturas mostradas en apartados anteriores se emplea un procesador de propósito general conectado mediante un bus AMBA a un conjunto de periféricos específicos para el procesamiento de la trama H.264. En este caso se dispone de bloques para el análisis de la trama, la predicción *intra*, la compensación de movimiento, la descodificación de entropía CAVLC, la IICT y el filtro antibloques.

El resultado de la síntesis de este descodificador con tecnología de 0.18 µm requiere un área de 46.7 mm^2 empleando 457 K puertas equivalentes, 12 kB de memoria interna y 149 kB de memoria externa para una resolución de imagen QCIF.

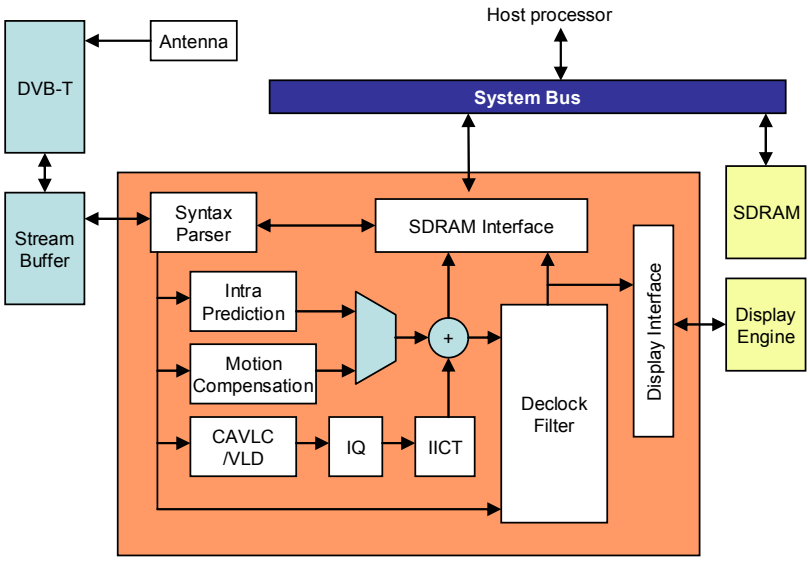


Figura 3-8 Diagrama de bloques del procesador multiestándar descrito en [LLWL06].

d) En [CHC+05] investigadores del Instituto de Ingeniería Electrónica de la Universidad Nacional de Taiwán muestran una arquitectura basada en un procesador RISC y una serie de módulos desarrollados en *hardware* que implementan los bloques funcionales de un descodificador conforme con H.264 BP (ver Figura 3-9). La arquitectura se encuentra segmentada a nivel de bloque, macrobloque e imagen. En el primero de los niveles de segmentación se han diseñado tres módulos que permiten la extracción de la información de la trama (*Parser Engine*), la cuantificación inversa e IICT (*IQ/IT Engine*) y la predicción *intra* (*Intra_Pred Engine*), en el segundo de los niveles se encuentra el bloque que permite realizar la predicción *inter* (*Inter_pred Engine*) y, finalmente, en el nivel de imagen se ha diseñado el módulo que calcula el filtro antibloques (*Deblock Engine*). Todos los módulos diseñados han sido optimizados para minimizar el ancho de banda en el acceso a la memoria externa.

El descodificador ha sido sintetizado empleando tecnología de 180 nm (217 K puertas equivalentes, 10 kB de memoria interna y 16 MB de memoria externa) logrando el funcionamiento en tiempo real para secuencias H.264 BP con una resolución máxima de 2048×1024 a una frecuencia de reloj de 120 MHz.

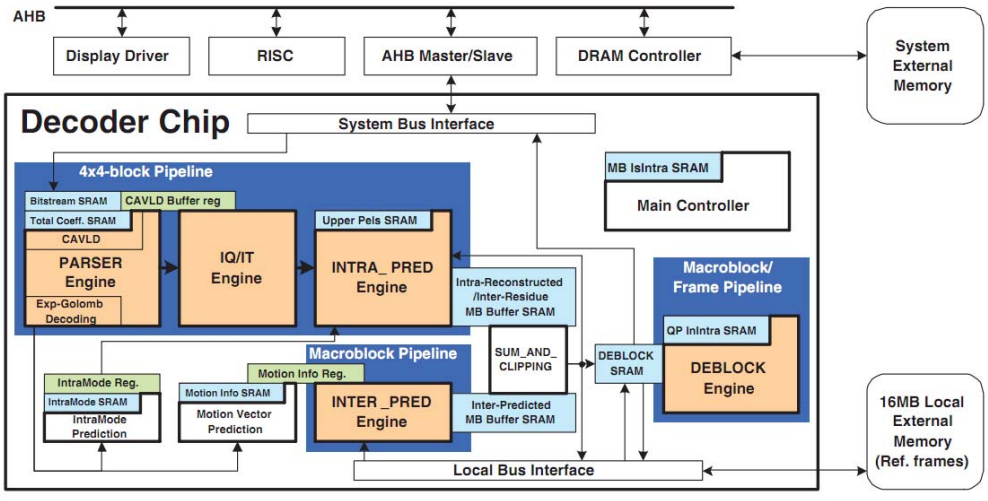


Figura 3-9 Diagrama de bloques del descodificador propuesto en [CHC+05].

e) Lee y otros investigadores de la Universidad de Corea presentan en [LPK+06] un descodificador H.264 orientado al desarrollo de aplicaciones móviles. Una vez más, se trata de una arquitectura basada en un procesador de propósito general tipo RISC (ARM), conectado mediante un bus AMBA a una serie de módulos desarrollados en una FPGA para el procesamiento de la trama H.264. La Figura 3-10 muestra el diagrama de bloques del descodificador en el que se aprecian los bloques para realizar las siguientes tareas: CAVLD, cuantificación inversa e IICT, predicción *inter* e *intra* y filtro antibloques.

El diseño se ha especificado en Verilog y el resultado de la síntesis ha sido probado en una Virtex_II de Xilinx requiriendo 71 K puertas equivalentes, 8 kB de memoria interna y 8 MB de memoria externa. Con este diseño se ha logrado el funcionamiento en tiempo real para secuencias de tamaño CIF a una frecuencia de reloj de 30 MHz.

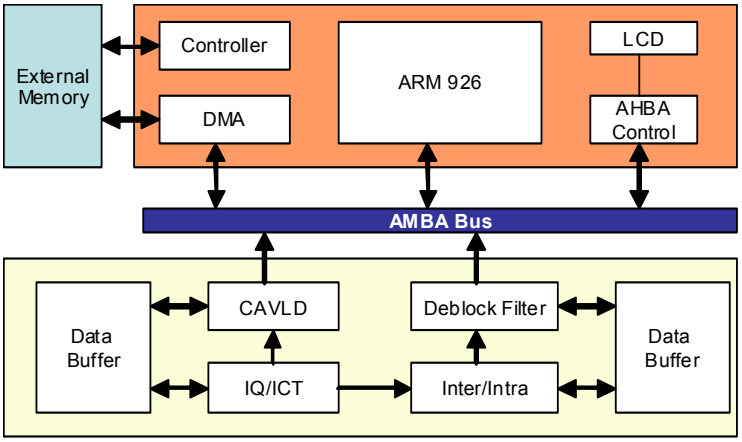


Figura 3-10 Diagrama de bloques del procesador desarrollado por Lee en la Universidad de Corea [LPK+06].

f) Finalmente, en [CCC+09] investigadores de la Universidad Nacional Chung Cheng proponen una arquitectura de bajo consumo para la implementación de un descodificador multiformato que soporta los estándares MPEG-2, MPEG-4 y H.264 BP. La arquitectura propuesta, mostrada en la Figura 3-11, se basa en un procesador RISC de propósito general conectado mediante un bus AHB a seis módulos diseñados específicamente para implementar los diversos elementos comunes a todos los descodificadores. La funcionalidad de cada uno de estos módulos es la siguiente:

- *Bit-Stream Analyzer* (BSA). Realiza el análisis de la trama.
- *Hybrid Texture Decoder* (HTD). Materializa la predicción AC/DC y la cuantificación y transformación inversas.
- *Hybrid Pixel Compensation* (HPC). Realiza la predicción *inter/intra* y la compensación de movimiento.
- *In-Loop Filter* (ILF). Implementa el filtrado que elimina el efecto de borde que aparece en los bloques descodificados.
- *Reusable Data Manager* (RDM). Reduce el número de accesos a memoria externa almacenando información que puede ser reutilizada en la descodificación de macrobloques posteriores.
- *System Controller*. Sincroniza el resto de los bloques que componen el descodificador.

El diseño ha sido sintetizado con tecnología CMOS de 0.13 µm, obteniendo un área de 5.0 mm^2, empleando 252 K puertas equivalentes y 4.9 kB de memoria SRAM interna. El consumo requerido es del 71.1 mW a 120 MHz para descodificar secuencias H.264 en formato 1080HD.

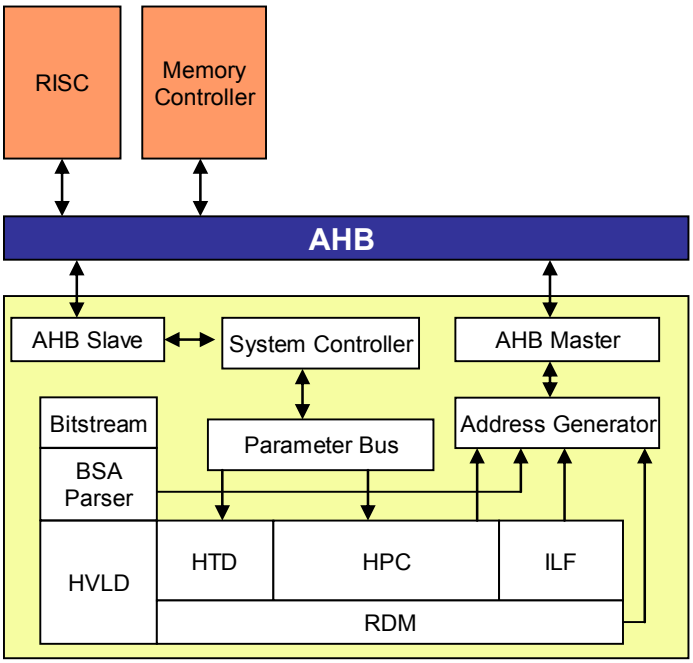


Figura 3-11 Diagrama de bloques del procesador multiestándar descrito en [CCC+09].

Comparando los resultados alcanzados en los trabajos presentados anteriormente se ha elaborado la Tabla 3-2 en la que se muestra un resumen de las características de todas las propuestas presentadas en este apartado.

Propuesta	[KJB+04]	[HSMC04]	[LLWL06]	[CHC+05]	[LPK+06]	[CCC+09]
Estándar	H.264 BP	H.264 MP	H.264 BP	H.264 BP	H.264 BP	MPEG-2 MPEG-4 SP H.264 BP
Tamaño Imagen	1920×1088	2048×1024	176×144	2048×1024	352×288	1920×1088
Puertas	910 K	300 K	457 K	217 K	71 K	252 K
Memoria Interna	4.9 kB	74 kB	12 kB	10 kB	8 kB	4.9 kB
Frecuencia	130 MHz	200 MHz	1.2 MHz	120 MHz	30 MHz	120 MHz
Tecnología	130 nm	130 nm	180 nm	180 nm	FPGA	130 nm
Consumo	554 mW	160 mW	0.865 mW	N/A	N/A	71.1 mW

Tabla 3-2. Comparativa de las características de las propuestas de descodificadores basados en un procesador de propósito general y módulos específicos para el procesamiento de la trama.

3.1.2.6 Descodificador H.264 basado en arquitectura de flujos de datos

En [KS05] investigadores de la Universidad de North Carolina muestran una metodología que facilita el diseño de procesadores orientados a la implementación de descodificadores de vídeo. Empleando la metodología se ha diseñado un procesador con una arquitectura orientada a datos WaveScalar desarrollada en la Universidad de Washington [SMSO03]. Este tipo de procesadores están formados por multitud de procesadores elementales (*grid* de procesadores) con un repertorio de instrucciones muy reducido que cooperan para realizar las operaciones.

La asignación de las instrucciones a los procesadores elementales se ha realizado empleando la información de las transferencias de datos entre los bloques funcionales, obtenida al ejecutar el descodificador sobre un procesador de tipo ARM. El procesador desarrollado emplea 1024 procesadores, ejecutando cada uno de ellos 8 instrucciones.

Los autores comparan (ver Tabla 3-3) el número de ciclos de reloj necesarios para descodificar 15 imágenes de resolución CIF y el área empleada en tres implementaciones: una basada en un ARM, otra basada en un procesador configurable y finalmente una tercera con el procesador basado en flujo de datos. Como se puede apreciar la tercera tiene un área 2.8 veces mayor que la requerida por un ARM pero, sin embargo, reduce en aproximadamente 5 veces el tiempo de ejecución[47].

H.264	ARM9TDMI	Procesador Configurable	[KS05]
Ciclos de Reloj/imagen	728.12 Mciclos	278.34 Mciclos	144.49 Mciclos
Área Estimada	1.16 mm^2	2.47 mm^2	3.2 mm^2

Tabla 3-3. Comparativa de las características de la implementación propuesta en [KS05].

3.1.3 Alternativas basadas en procesadores digitales de señal

En la última década han aparecido en el mercado procesadores digitales de señal orientados a la implementación de codificadores o descodificadores de vídeo. Estos procesadores poseen una arquitectura que facilita la implementación de este tipo de aplicaciones e incluyen periféricos (puertos de vídeo, interfaces con redes de área local, puertos de audio, etc.) que son de gran interés para el diseño de sistemas de codificación o descodificación de vídeo.

Esta alternativa aúna la ventaja de la flexibilidad de los procesadores de propósito general, en cuanto a poder adaptarse a nuevos estándares modificando el programa que ejecutan, junto con una adecuada eficiencia en la ejecución de este tipo de algoritmos que, sin llegar a alcanzar las prestaciones de las arquitecturas específicas, las hacen competitivas frente a la rigidez de los ASIC. Adicionalmente, los procesadores digitales de señal tienen un coste económico y un consumo de energía menor que los procesadores de propósito general, siendo en algunos casos comparables a los ofrecidos por los ASIC.

[47] Los autores no indican la frecuencia máxima de funcionamiento para ninguna de las tres implementaciones presentadas por lo que no es posible extrapolar la resolución máxima que podrían tener las imágenes manteniendo el funcionamiento en tiempo real.

Se trata por tanto de una tecnología intermedia entre las implementaciones basadas en procesadores de propósito general, en las que para un mismo consumo de energía se obtiene un rendimiento menor, y las que se apoyan en el diseño de ASIC que poseen un mayor rendimiento para un determinado consumo.

En el estudio del estado del arte de implementaciones de descodificadores sobre tecnología DSP se han localizado numerosas referencias. Seguidamente se presentan sólo algunos ejemplos que se han considerado representativos de la implementación de sistemas completos de descodificación sobre esta tecnología.

Ya en 2002, Cacopardi y otros investigadores de la Universidad de Perugia, presentan en [CCFS02] un descodificador MPEG-2 multicanal implementado en un procesador TMS320DM642 [TI10b] de Texas Instruments a 600 MHz (para más detalle ver 3.2.1). En este artículo se recogen algunas técnicas de optimización que permiten mejorar el rendimiento de alguno de los bloques funcionales. Como resultado del proceso de optimización se concluye que es posible lograr la descodificación de 3.3 secuencias con definición estándar en paralelo lo que equivale aproximadamente a una secuencia de televisión en alta definición con una resolución de 1280×720 píxeles.

En [Sie05], P. Siebert perteneciente a la compañía Siemens, presenta un STB-IP multiformato que soporta H.264 en su perfil *Main* y nivel 3.0. El STB-IP se basa en un procesador TMS320DM642 de Texas Instruments a 600 MHz y un SoC de IBM denominado STB04x [STB04]. Este SoC realiza todas las tareas de recepción de datos por la red, extracción de las tramas elementales, descodificación de audio, sincronización e interfaz de usuario; mientras que el DSP sólo realiza la descodificación del vídeo H.264. El diagrama de bloques del sistema desarrollado se muestra en la Figura 3-12. La carga computacional del DSP descodificando una secuencia H.264 MP de 2 Mbps es del 80% en media.

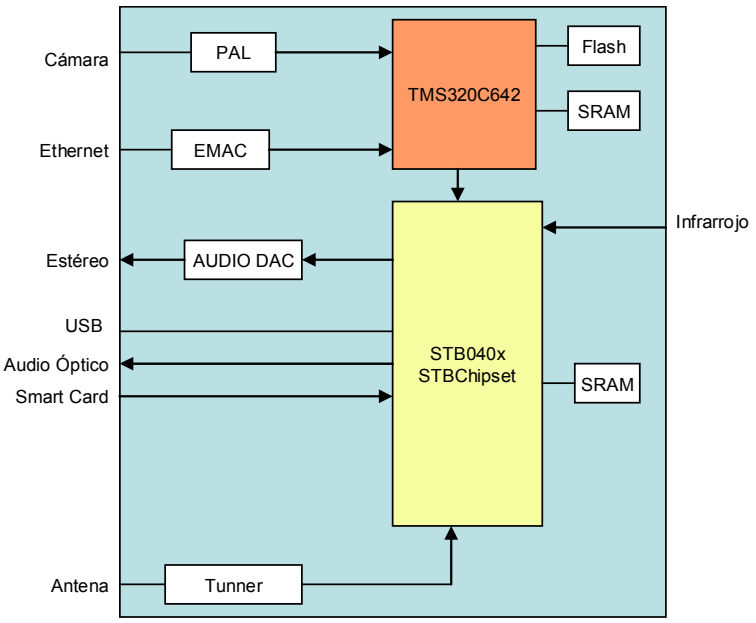


Figura 3-12 Diagrama de bloques del STB-IP presentado en [Sie05].

Por otra parte, Y. S. Tung y otros investigadores de la Universidad Nacional de Taiwán presentan en [TWT+05] un sistema de descodificación de televisión digital con entrada de datos IP basado en el procesador TriMedia [TRIM] de NXP (para más detalles ver 3.2.4). El sistema desarrollado soporta los estándares MPEG-2, MPEG-4, VC-1 y H.264 perfiles *Baseline* y *Main*. En este trabajo se analizan los módulos comunes a todos los estándares para posteriormente prestarles una atención especial

en el proceso de optimización. Con el sistema desarrollado se logra el funcionamiento en tiempo real para secuencias en formato SD codificadas conformes al perfil *Main* de H.264 con 1.5 Mbps empleando una frecuencia de reloj de 300 MHz. La Figura 3-13 muestra el prototipo desarrollado.

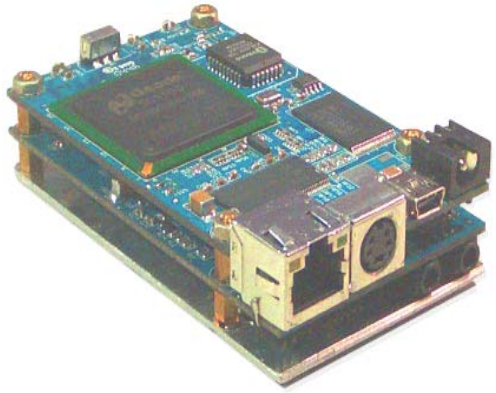


Figura 3-13 Aspecto del sistema de descodificación basado en TriMedia propuesto en [TWT+05].

3.2 Estudio de la arquitectura de los Procesadores Digitales de Señal

Dentro del mercado de los DSPs existen en la actualidad varios fabricantes que comercializan este tipo de procesadores. Entre los más destacados se encuentran Texas Instruments, Analog Devices y NXP (anteriormente Philips). Cada uno de estos fabricantes dispone de varias familias de DSPs orientadas al desarrollo de aplicaciones de vídeo. Estas familias crecen a una gran velocidad, por lo que constantemente están apareciendo nuevos dispositivos con mejores prestaciones lo que hace que el diseñador disponga de numerosas opciones a la hora de seleccionar el procesador más adecuado para su aplicación.

A pesar del elevado número de dispositivos disponibles en cada familia, el núcleo del procesador es el mismo y las principales diferencias se encuentran en la cantidad de memoria interna que poseen y los periféricos que integran.

De todos los fabricantes existentes en la actualidad se han seleccionado las cuatro familias que se han considerado más representativas tanto por volumen de ventas, como por la existencia de publicaciones científico-técnicas sobre la optimización de aplicaciones de vídeo en las que se emplean estos procesadores. En los siguientes apartados se presentan estas familias de procesadores y se describe brevemente su arquitectura.

3.2.1 Procesadores de la familia TMS320DM64x

Se trata de una familia de DSPs de la compañía Texas Instruments presentada en 2003 y compuesta actualmente por 11 dispositivos [DM64x]. La Figura 3-14 muestra la estructura interna del núcleo que poseen todos los procesadores de esta familia, formada por una CPU con arquitectura VLIW, dos niveles de memoria interna, un controlador de DMA, una interfaz con memoria externa (*External Memory InterFace* o EMIF) y una serie de periféricos [TI01].

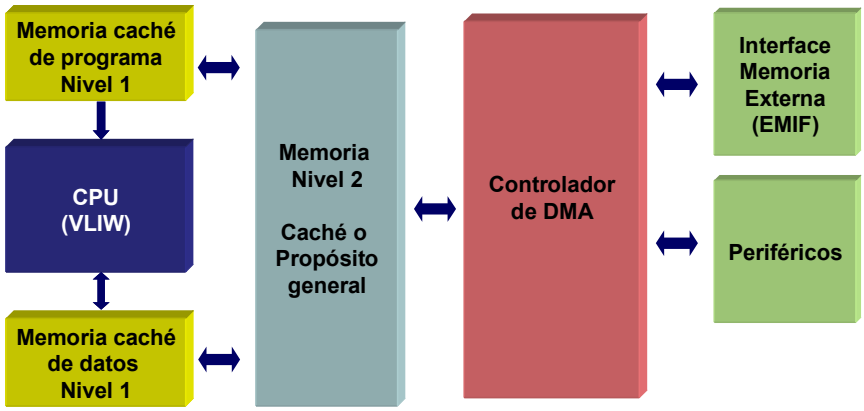


Figura 3-14. Diagrama de bloques de los procesadores de la familia TMS320DM64x.

El núcleo de la CPU denominado C64x [TI10] está formado por una máquina VLIW segmentada que es capaz de ejecutar hasta 8 instrucciones por ciclo de reloj con datos de 32 bits. Para poder conseguir este rendimiento dispone de 8 unidades funcionales y 64 registros internos de 32 bits. Las unidades funcionales se encuentran agrupadas en dos rutas de datos que se reparten a partes iguales los registros disponibles. Cada una de ellas dispone de 4 unidades funcionales idénticas que operan principalmente con los registros de la ruta de datos a la que pertenecen, aunque tienen una capacidad limitada para acceder a registros de la otra ruta.

El procesador posee una arquitectura SIMD (*Single Instruction, Multiple Data*) que permite que una unidad funcional realice simultáneamente varias operaciones con datos de un tamaño inferior a 32 bits (4 datos de 8 bits ó 2 datos de 16 bits) que se encuentran almacenados en alguno de sus registros internos.

La memoria del procesador está organizada jerárquicamente en tres niveles tal y como se muestra en la Figura 3-15. Los dos primeros niveles (Nivel 1 o L1 y Nivel 2 o L2) son internos al procesador, mientras que el tercero es externo. El primero de los niveles es de tipo caché y siempre se encuentra activo puesto que es el único con el que la CPU es capaz de trabajar directamente. Dicha memoria está a su vez dividida en dos, una para datos (L1D) y otra para programa (L1P), siendo ambas de 16 kB. Por otra parte, el nivel L2 posee 256 kB y puede ser configurado como memoria de propósito general, memoria caché de nivel 2 o con una distribución mixta, según las necesidades de la aplicación. Finalmente, el nivel 3 de la jerarquía es externo al DSP y se conecta a éste a través de la interfaz de memoria externa EMIF pudiendo disponer de un máximo de 1 GB.

El tiempo de acceso a las memorias depende del nivel de la jerarquía en el que se encuentren. Así, el acceso a las memorias L1 se realiza a la frecuencia del procesador, el acceso a L2 se produce a la mitad de dicha frecuencia y el acceso a la memoria externa se efectúa a una frecuencia programable entre 100 MHz y 133 MHz.

El controlador de DMA gestiona todas las transferencias entre la memoria interna (L1 y L2) y la memoria externa o los periféricos tal y como se aprecia en la Figura 3-14. Las situaciones que provocan una transferencia de datos mediante el controlador de DMA son las siguientes:

- La CPU accede a un fragmento de código o a datos que no se encuentran almacenados en alguna de las memorias internas, por lo que es necesario acceder a la memoria externa.
- Los periféricos realizan alguna transferencia de datos hacia/desde la memoria externa.

- El usuario programa el controlador de DMA para realizar una transferencia explícita de datos entre la memoria interna de propósito general y la memoria externa, bien mediante el mecanismo denominado QDMA (*Quick* DMA), bien mediante el EDMA[48] (*Enhanced* DMA). Dichas transferencias pueden realizarse con bloques de datos unidimensionales o bidimensionales (transferencias 1D o 2D).

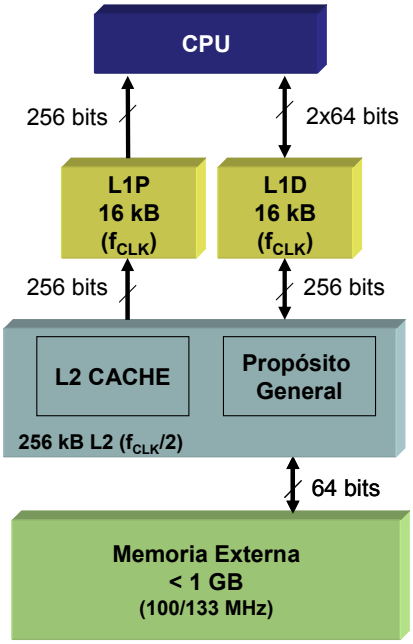


Figura 3-15. Jerarquía de memoria del procesador TMS320DM642.

Todas las transferencias que se solicitan son almacenadas en una de las cuatro colas de petición que posee el controlador (ver Figura 3-16). Cada cola dispone de 16 posiciones por lo que es posible acumular 64 peticiones en total. Las colas de petición están priorizadas lo que implica que, en caso de tener pendientes transferencias en varias de ellas, éstas se atienden en un orden preestablecido; además, es posible configurar el número de peticiones máximas que puede generar cada uno de los elementos que puede acceder al controlador del DMA. Así por ejemplo, en la Figura 3-16 se aprecia que las peticiones provenientes de los periféricos, representadas en color amarillo, pueden acumularse en cualquiera de las 4 colas de prioridad pero sólo disponen de 4 posiciones en la cola de prioridad urgente, 7 en la de prioridad alta y 5 en las de prioridad media y baja. Lo mismo se puede indicar para las peticiones generadas por el controlador de EDMA, representadas en color rojo, y por la CPU y/o el usuario a través del controlador QDMA, en color azul.

El rendimiento de las transferencias de DMA depende de parámetros como el tipo de recurso al que se desea acceder, el número de datos que posee el *buffer* a transferir y su alineamiento en la memoria. En el caso de una memoria externa conectada al DSP a través de una interfaz EMIF, se pueden llegar a transferir ráfagas

[48] La diferencia entre emplear el controlador QDMA o utilizar el EDMA estriba en que el tiempo de programación de las transferencias del primero es menor ya que se accede directamente a los registros del controlador. Por el contrario, la configuración del segundo es más flexible puesto que incluye funcionalidades que el primero no soporta. El fabricante recomienda emplear el QDMA si es necesario realizar muchas transferencias consecutivas, mientras que el controlador EDMA se suele emplear para transferencias en las que se debe acceder a periféricos periódicamente.

de datos de 32 bits a un régimen de un dato por ciclo de reloj del procesador. Sin embargo, si sólo se transfiere un dato de 1 *byte* puede ser necesario emplear hasta 18 ciclos de reloj.

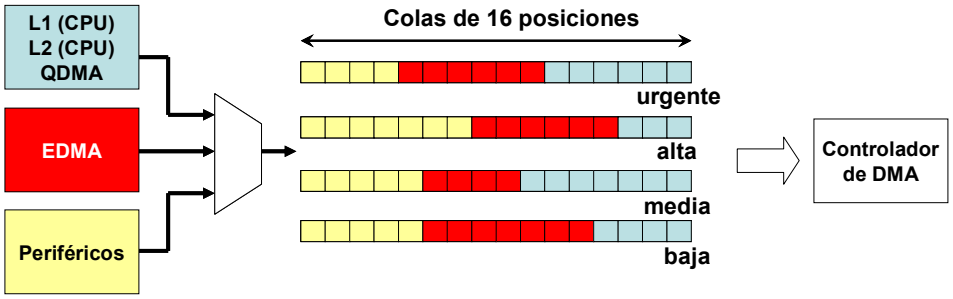


Figura 3-16. Estructura interna del controlador de DMA disponible en el TMS320DM642.

El DSP integra varios periféricos que pueden ser de interés para el desarrollo de aplicaciones de descodificación de vídeo. Así, se dispone de tres puertos bidireccionales de vídeo (*videoport*s) que generan la señal de video digital[49] o capturan la señal de televisión analógica, un puerto Ethernet (*Ethernet Medium Access Controller* o EMAC), un puerto serie multicanal de audio (McASP0), tres temporizadores de 32 bits, 16 pines de entrada/salida de propósito general (GP0) y un controlador del bus I2C.

3.2.2 Procesadores de la familia DaVinci

Esta familia de DSPs ha sido desarrollada también por la compañía Texas Instruments y fue presentada en 2007. Las principales novedades de esta familia respecto a la mostrada en el apartado 3.2.1 son el menor tamaño, pero mayor flexibilidad de configuración de la memoria interna, la mejora en la eficiencia del controlador de las transferencias por DMA, la ampliación del repertorio de instrucciones y la existencia, en algunos elementos de la familia, de un Procesador de Propósito General (*General Purpose Processor* o GPP) tipo ARM[50]. Para presentar la arquitectura del DSP que integran los elementos de esta familia se ha seleccionado el procesador TMS320DM6437 [TI06b].

El núcleo de este procesador, denominado C64x+ [TI10], posee una arquitectura VLIW con dos niveles de memoria interna (L1 y L2) y un controlador de DMA para transferencias entre elementos internos (*Internal* DMA o IDMA). El repertorio de instrucciones ha sido mejorado respecto al del núcleo C64x ya que se han incluido algunas operaciones aritméticas nuevas de suma, resta, multiplicación y empaquetado de datos. Además, se ha reducido el tamaño de algunas instrucciones para generar un código más compacto sin reducir el rendimiento y se ha flexibilizado la configuración de la memoria interna de nivel 1 (L1D y L1P).

El núcleo se conecta mediante una matriz de conmutación con un conjunto de periféricos internos y un subsistema de procesado de vídeo [TI08]. El diagrama de bloques se muestra en la Figura 3-17.

[49] Los puertos de vídeo soportan los siguientes formatos de salida de vídeo digital: CCIR601, ITU-BT.656, BT.1120, SMPTE 125M, 260M, 274M y 296M.

[50] Desde el punto de vista de los trabajos de optimización de descodificadores de vídeo realizados en esta tesis, la existencia de un GPP no supone ninguna ventaja, por lo que la explicación del procesador que se realiza en este apartado se centra en el DSP que integran todos los elementos de esta familia.

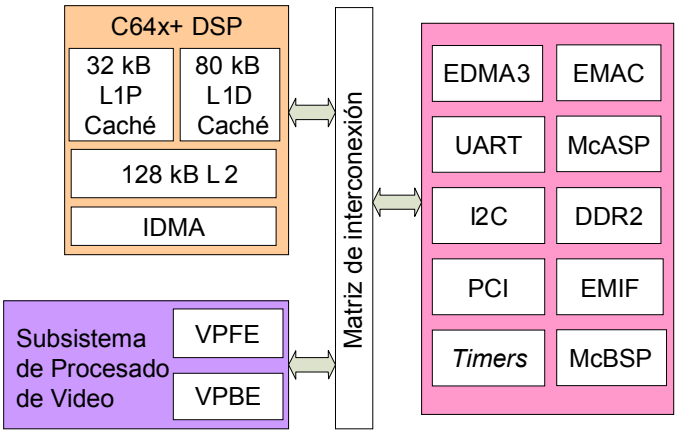


Figura 3-17. Diagrama de bloques de la arquitectura del procesador TMS320DM6437.

El tamaño total de la memoria interna es de 240 kB (48 kB menos que el TMS320DM642) pero se ha flexibilizado su configuración. La caché de nivel 1 de programa (L1P) consiste en 32 kB que pueden ser configurados como memoria de propósito general, cache de nivel 1 para código o una combinación de ambas. La cache de nivel 1 de datos (L1D) posee 80 kB y también puede configurarse como memoria de datos de propósito general, cache de nivel 1 de datos, con un tamaño máximo de 32 kB, o una combinación de ambas. Finalmente, la memoria L2 posee 128 kB que pueden ser compartidos entre memoria caché de nivel 2 y memoria de propósito general para datos y código.

La funcionalidad del DMA ha sido modificada sustancialmente respecto a las familias anteriores de este mismo fabricante. En esta nueva arquitectura aparecen dos controladores independientes: el IDMA, para realizar transferencias más eficientes entre las memorias internas y los periféricos; y el EDMA3, para las transferencias entre memoria externa y memoria interna. El empleo combinado de ambos controladores mejora significativamente el rendimiento puesto que se reduce la intervención de la CPU en todo el proceso. La Figura 3-18 presenta la relación existente entre los dos controladores de DMA y los recursos a los que puede acceder cada uno de ellos. La inclusión de los dos controladores de DMA permite realizar transferencias de bloques de datos tridimensionales así como enlazar automáticamente transferencias sin intervención de la CPU.

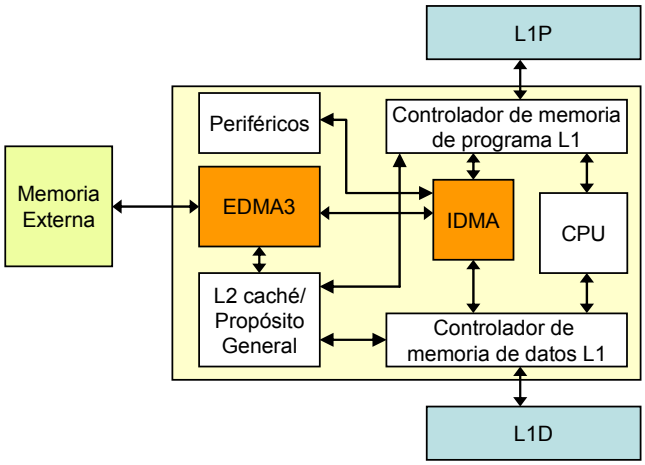


Figura 3-18. Arquitectura de los controladores de IDMA y EDMA3 de la familia DaVinci.

La Tabla 3-4 muestra las diferencias más significativas entre los procesadores TMS320DM642 y TMS320DM6437. De todas las características que diferencian a ambos procesadores, las que mayor incidencia tienen a la hora de aplicar metodologías de optimización de descodificadores de vídeo, son la existencia de una arquitectura de DMA mejorada y la flexibilidad de configuración de los diferentes niveles de memoria interna.

Características	TMS320DM642	TMS320DM6437
Núcleo de la CPU	C64x	C64x+
Repertorio de instrucciones	Básico	Básico + instrucciones compactas + nuevas operaciones aritméticas
L1D Propósito General / Caché (máximo)	0 kB / 16 kB	80 kB / 32 kB
L1P Propósito General / Caché (máximo)	0 kB / 16 kB	32 kB / 32 kB
L2 Propósito General / Caché (máximo)	256 kB / 256 kB	128 kB / 128 kB
Transferencias DMA	EDMA	IDMA + EDMA

Tabla 3-4. Diferencias más significativas entre los procesadores TMS320DM642 y TMS320DM6437 de Texas Instruments.

3.2.3 Procesadores de la familia Blackfin

Esta familia de procesadores ha sido desarrollada por Analog Devices y está formada por 24 dispositivos. Entre los elementos disponibles se ha seleccionado el ADSP-BF549 [BF549] para presentar las características más destacadas de su arquitectura puesto que, en el momento de redactar esta tesis, es el más completo de la familia. Estos procesadores poseen una arquitectura basada en la *Micro Signal Architecture* (MSA) desarrollada por Analog Devices en cooperación con Intel [WK07].

Todos los elementos de la familia poseen una estructura como la mostrada en la Figura 3-19. El procesador está formado por una CPU con arquitectura RISC que es capaz de ejecutar instrucciones empaquetadas de 16 y 32 bits, varios módulos de memoria interna, una interfaz con diversos tipos de memoria externa y numerosos periféricos internos.

La CPU incluye las siguientes unidades funcionales: dos multiplicadores de 16 bits, dos acumuladores de 40 bits, seis ALUs de 40 bits que son capaces de ejecutar instrucciones específicas para el procesado de vídeo y un desplazador de 40 bits. Todas las unidades funcionales pueden trabajar con datos de un número de bits diferente a su tamaño. Las unidades funcionales pueden emplear alguno de los 8 registros de 32 bits que posee su única ruta de datos, no existiendo restricciones en el acceso de cualquier unidad funcional a cualquier registro.

La arquitectura de memoria del procesador está organizada en tres niveles como se muestra en la Figura 3-20. En dicha figura se aprecia que la CPU está conectada a cuatro memorias de nivel 1, a las que se accede a la frecuencia del reloj de la CPU. Estas cuatro memorias tienen funcionalidades diferentes:

- L1I: Este bloque de 64 kB está orientado a alojar las instrucciones que serán ejecutadas por la CPU. De los 64 kB un máximo de 16 kB se pueden configurar como caché de instrucciones de nivel 1.
- L1D: Esta memoria de 64 kB permite alojar datos de la aplicación. De su tamaño total, un máximo de 32 kB pueden ser configurados como memoria caché de datos de nivel 1.

- L1S: Se trata de un bloque de 4 kB que permite almacenar temporalmente datos que emplea la aplicación pero que pueden ser sustituidos por otros a lo largo de la ejecución del programa.
- L1ROM: Esta memoria de 64 kB no es accesible por las aplicaciones de usuario y se emplea durante el arranque del sistema.

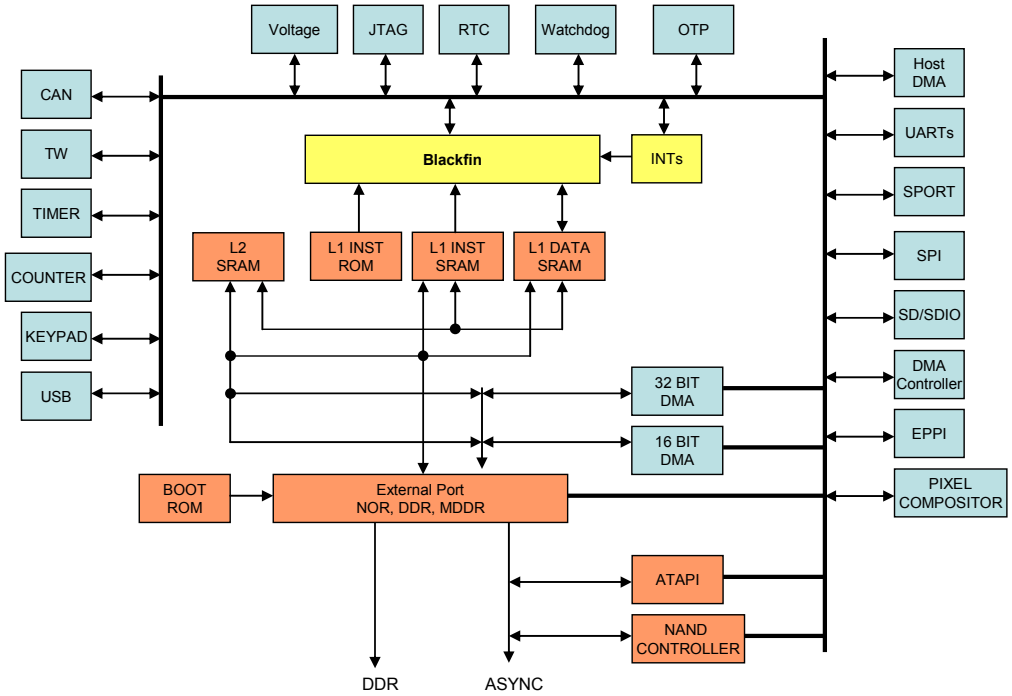


Figura 3-19. Arquitectura interna de los procesadores de Analog Devices.

El segundo nivel de memoria (L2) está compuesto por un bloque de 128 kB que puede ser configurado como memoria interna, memoria caché de nivel 2 o una combinación de ambas. El tiempo de acceso a esta memoria es el doble que el periodo de la señal de reloj del procesador. Finalmente, el tercer nivel de memoria está formado por un módulo interno denominado L3 de 64 kB de memoria ROM orientado al arranque del sistema y por toda la memoria externa que se conecte a través de alguno de sus interfaces hasta un máximo de 512 MB. Las interfaces de acceso a la memoria externa se pueden configurar para acceder a memorias de tipo DDR, SRAM, EPROM o FLASH.

En relación con las transferencias de datos entre los diversos niveles de memoria, y tal y como se aprecia en la Figura 3-19, existen dos controladores de DMA; uno de 16 bits y otro de 32. Cada uno de estos controladores puede gestionar hasta un máximo de 12 peticiones priorizadas estáticamente, de modo que cada periférico o la CPU deben solicitar las transferencias por el canal adecuado dependiendo de la prioridad que tenga dicha petición. Las transferencias que son capaces de realizar ambos controladores de DMA pueden ser de una o dos dimensiones (1D y 2D), tanto en los datos de origen, como en los de destino, pudiendo enlazarse transferencias lo que implica que tras la finalización de una, se comienza automáticamente la siguiente sin intervención de la CPU.

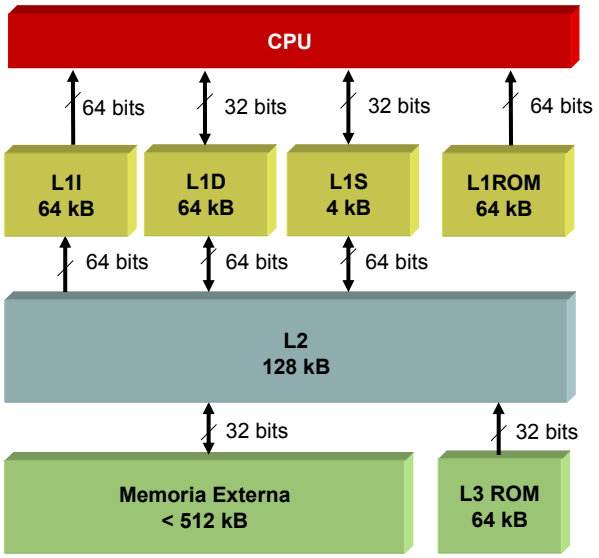


Figura 3-20. Arquitectura de memoria del procesador ADSP-BF549.

Finalmente, en relación con los periféricos disponibles para la realización de sistemas de descodificación de vídeo, cabe mencionar el controlador de red local (*Controller Access Network* o CAN), el controlador de bus I2C, el sistema de procesado de vídeo (*Pixel Compositor* o PIXC) y los puertos serie síncronos (SPORTs).

3.2.4 Procesadores de la familia TriMedia

Los primeros procesadores de la familia TriMedia fueron desarrollados por Philips a finales de la década de los 90. Posteriormente se segregó la división de procesadores, pasando a llamarse NXP. Esta compañía ha continuado desarrollando DSPs de la familia TriMedia [TRIM], de modo que actualmente existen 12 elementos disponibles. Todos los dispositivos tienen una denominación del tipo PNXx e integran un núcleo de procesamiento basado en las diversas versiones del procesador digital de señal TriMedia. Uno de los últimos elementos que ha aparecido en el mercado es el denominado PNX1700 que integra el DSP TM5250 [PNX17] y que será el empleado para ejemplificar las características más relevantes de la arquitectura de esta familia de procesadores[51].

El diagrama de bloques del DSP se muestra en la Figura 3-21. En él se aprecian, además de los periféricos que se comentarán posteriormente, el núcleo de procesamiento TM5250. Dicho núcleo posee 29 unidades funcionales: 4 para gestión de constantes, 4 para la gestión de números enteros de 32 bits, 3 ALUs de propósito general, 3 registros de desplazamiento, 3 ALUs especializadas para el procesamiento de vídeo, 4 multiplicadores, 4 unidades para operación con números en punto flotante, 3 unidades para cálculo de saltos y 1 unidad de carga y almacenamiento. Estas unidades funcionales trabajan con los 128 registros de propósito general que posee la ruta de datos del procesador.

La arquitectura SIMD que posee el procesador permite a las unidades funcionales trabajar en paralelo con datos de un tamaño menor al de los registros que posee la ruta de datos.

[51] El fabricante de los procesadores TriMedia no ofrece información detallada del funcionamiento del núcleo de sus procesadores sin haber firmado un acuerdo de confidencialidad. Por este motivo, la información que se describe en este apartado no es tan completa como la incluida para otros DSPs.

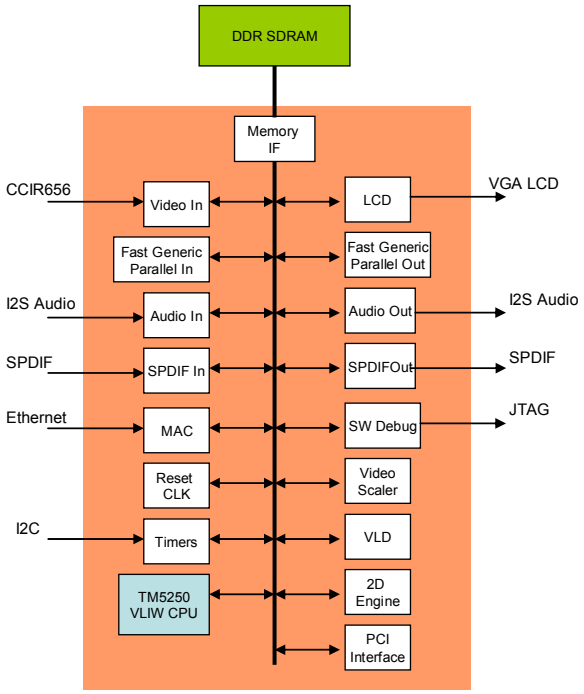


Figura 3-21. Arquitectura interna de los procesadores Trimedia.

La jerarquía de memoria del procesador está formada por los tres niveles que se muestran en la Figura 3-22[52]. En primer lugar se encuentran las memorias caché de nivel 1 tanto de datos como de instrucciones. La de instrucciones (*Level 1 ICache*) es de 8 vías asociativas con un tamaño de 64 kB y la de datos (*Level 1 DCache*) es de 4 vías asociativas con un tamaño de 16 kB. El segundo nivel de memoria caché (*Level 2 DCache*) está compuesto de 128 kB de memoria de 8 vías asociativas. En el tercer y último nivel, se encuentra la memoria externa cuyo tamaño máximo es de 64 GB. El acceso a este último nivel de memoria se puede realizar mediante el controlador de DMA que posee el procesador.

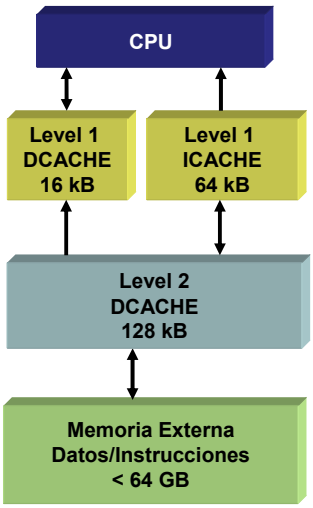


Figura 3-22. Jerarquía de memoria disponible en el procesador TM5250.

[52] En la figura no se ha incluido la anchura de los buses debido a que el fabricante no facilita esta información en la documentación disponible.

En relación con los periféricos integrados en el dispositivo, y desde el punto de vista del diseño de un sistema de descodificación de televisión digital, caben destacar los siguientes: interfaz con memoria externa (MMI), procesadores de vídeo de entrada (VIP) y de salida (QVCP), procesadores de audio de entrada (AI y SPDI) y de salida (AO y SPDO), controlador de bus I2C, controlador de acceso a Ethernet, descodificador de códigos de longitud variable para MPEG-2 y MPEG-4 (VLD) y procesador para realizar escalado de imágenes en tiempo real (MSB).

3.2.5 Comparación de los procesadores analizados

Para finalizar este apartado en el que se ha presentado el estado del arte en los procesadores digitales de señal, se ha elaborado la Tabla 3-5 en la que se muestra una comparativa de las características de los DSP analizados.

Características	TMS320DM642	TMS320DM6437	ADSP-BF549	PNX1700
Arquitectura CPU	VLIW+SIMD	VLIW+SIMD	RISC	VLIW+SIMD
Unidades Funcionales	8	8	11	29
Registros internos	64	64	8	128
Rutas de datos	2	2	1	1
Memoria L1 Programa	16 kB caché	32 kB caché o propósito general	64 kB caché o propósito general	64 kB caché
Memoria L1 Datos	16 kB caché	80 kB caché o propósito general	64 kB + 4 kB caché	16 kB caché
Memoria L2	256 kB caché o propósito general	128 kB caché o propósito general	128 kB caché o propósito general	128 kB caché o propósito general
Memoria externa	1 GB	256 GB	512 MB	64 GB
Controlador de DMA	1 Controlador DMA	2 Controladores DMA	2 Controladores DMA	1 Controlador DMA

Tabla 3-5. Comparación de las características de los DSPs analizados.

La principal conclusión que se extrae de esta comparación es la similitud que poseen las arquitecturas de estos procesadores ya que todos ellos disponen de varias unidades funcionales que gestionan la información almacenada en los registros internos, poseen varios niveles de memoria caché interna en los que se diferencian los empleados para almacenar datos y programa dentro del primer nivel de la caché, poseen controladores de DMA para acelerar las transferencias en los accesos a memoria externa y en todos los casos, a excepción del DSP ADSP-BF549 poseen una arquitectura de tipo VLIW.

Todas estas similitudes posibilitan definir técnicas de optimización de algoritmos de codificación o descodificación de vídeo que pueden ser aplicadas fácilmente a todos ellos.

3.3 Metodologías de optimización: implementaciones de codificadores/descodificadores basados en DSP

El análisis exhaustivo de las publicaciones relacionadas con la optimización del tiempo de ejecución de descodificadores de vídeo sobre procesadores digitales de señal ha demostrado que, hasta la fecha, no se ha propuesto una metodología de optimización sistemática que pueda ser aplicada al diseño de descodificadores implementados sobre tecnología DSP. Sí se han encontrado numerosas publicaciones en las que se describen, con mayor o menor detalle, las técnicas de optimización que han sido empleadas por diferentes grupos de trabajo para reducir el tiempo de ejecución de codificadores[53] o descodificadores de vídeo con esta tecnología. Por tanto, en este apartado se presenta una recopilación de las diversas técnicas de optimización de codificadores y descodificadores de vídeo que se han publicado recientemente utilizando los procesadores digitales de señal. Si bien en la recopilación se incluyen técnicas aplicadas para diferentes procesadores, se ha prestado una especial atención a las empleadas con procesadores de la compañía Texas Instruments puesto que éstos han sido el soporte utilizado para la realización de los experimentos que han permitido la extracción de la metodología de optimización que se presenta como resultado principal de esta tesis.

La información de este apartado se ha organizado por estándares, puesto que el nivel de complejidad de cada uno de ellos conlleva que las técnicas empleadas para la optimización de los más antiguos sean más sencillas que las utilizadas para los descodificadores más recientes. Por tanto, en 3.3.1 se analizan las técnicas empleadas para la optimización de codificadores y descodificadores que implementan el estándar MPEG-2, seguidamente, en 3.3.2, las utilizadas para el estándar MPEG-4 y, finalmente, en 3.3.3 las aplicadas para el estándar H.264.

3.3.1 Técnicas aplicadas en la optimización en velocidad de codificadores/descodificadores MPEG-2

Sriram y otros investigadores del *DSPs R&D Center* de Texas Instruments presentan en [SH98] algunas técnicas de optimización para un descodificador MPEG-2 y los resultados obtenidos tras su aplicación. Las técnicas empleadas se basan en el uso de varias *look-up tables* para obtener los coeficientes de la VLD, la descomposición del cálculo de la IDCT en filas y columnas [HL97] y la sustitución del código C que implementa la obtención de los píxeles interpolados en el caso de la compensación de ½ píxel, por una función en ensamblador que emplea instrucciones específicas para realizar la carga y almacenamiento de los píxeles. El número medio de ciclos de reloj necesarios para descodificar una imagen de una secuencia de vídeo[54] es de $7.6 \cdot 10^6$. A partir de este dato se puede extrapolar que, con un TMS320DM642 trabajando a 600 MHz, sería posible descodificar unas 92 imágenes por segundo en formato NTSC o, lo que es lo mismo, 3 secuencias en dicho formato simultáneamente.

[53] La búsqueda de referencias no se ha limitado a las implementaciones de descodificadores, sino que también se han analizado publicaciones en las que se trata de optimizar codificadores de los diferentes estándares puesto que las técnicas empleadas con éstos pueden ser trasladadas a los descodificadores de forma sencilla.

[54] La secuencia empleada tiene un tamaño de 720×480 píxeles, posee 30 imágenes por segundo y está compuesta de 2 imágenes de tipo I, 8 de tipo P y 20 de tipo B. En el artículo no se indica el régimen binario que posee la secuencia.

S. Arora y otros investigadores de Texas Instruments presentan en [ARG03] el desarrollo de un descodificador optimizado MPEG-2 para el TMS320DM642 explicando el procedimiento que se ha llevado a cabo[55]. Los métodos de optimización empleados en esta publicación afectan a diferentes módulos del descodificador. La IDCT se realiza mediante el algoritmo de filas y columnas [HL97]. Para el cálculo de la referencia de un macrobloque es posible diferenciar hasta 16 posibles combinaciones[56] si se trata de una imagen con predicción bidireccional; a partir de estas 16 combinaciones los autores definen un conjunto de 7 funciones que son capaces de implementar todas las posibles combinaciones, dada la similitud entre algunas de ellas. Para la carga, promediado y almacenamiento de los píxeles se emplean las instrucciones SIMD que permiten trabajar en paralelo con varios datos. Los píxeles de referencia que se encuentran en la memoria externa se transfieren a memoria interna empleando el controlador de DMA, aunque no se detalla cómo se realizan dichas transferencias. Finalmente, y dado que el bucle principal de descodificación posee un tamaño mayor que la cache de nivel 1 de programa, se ha separado el bucle principal de descodificación de macrobloques en dos partes de modo que, en primer lugar, se realiza la VLD, la IDCT y las peticiones por DMA de las referencias de varios macrobloques para, posteriormente, realizar la compensación de movimiento de todos los macrobloques con la información obtenida con anterioridad. Con esta estrategia cada una de las dos partes que componen el bucle principal pueden ser alojadas en memoria caché de nivel 1 de programa.

Las medidas realizadas emplean como secuencia de referencia una de las definidas en el estándar, más concretamente la denominada gi_9.m2v[57]. Se ha medido el número de ciclos de reloj que invierte la CPU para la descodificación de cada una de las primeras 5 imágenes mostrándose los resultados en la Tabla 3-6. A la vista de estos resultados, con un procesador trabajando a 600 MHz, sería posible descodificar más de 4 secuencias extraídas de un DVD comercial si no se realiza ninguna otra tarea.

Imagen	Ciclos de Reloj
I	$5.5 \cdot 10^6$
P	$4.6 \cdot 10^6$
B1	$4.8 \cdot 10^6$
B2	$5.3 \cdot 10^6$
B3	$5.3 \cdot 10^6$

Tabla 3-6. Rendimiento del descodificador MPEG-2, en ciclos de reloj por imagen, propuesto en [ARG03].

Cacopardi y otros investigadores de la Universidad de Perugia presentan en [CCFS02] un descodificador MPEG-2 multicanal optimizado para el procesador TMS320DM642 que también podría ser válido para televisión de alta definición (HDTV)

[55] Este mismo grupo de trabajo presenta en [JRGS02] el proceso de optimización de un codificador MPEG-2 empleando las mismas técnicas que se describen en este artículo.

[56] Las 16 combinaciones se corresponden con las siguientes posibilidades: que no haya que realizar interpolación en ninguna de las direcciones (1 opción), que haya exclusivamente una predicción horizontal o vertical en alguna de las dos referencias (4 opciones), que haya dos predicciones en total en ambas direcciones (6 opciones), que existan tres predicciones (4 opciones) o que sea necesario realizar las cuatro interpolaciones (1 opción).

[57] Esta secuencia está caracterizada por los siguientes parámetros: 720×480 píxeles con 15 Mbps de ancho de banda, compensación de movimiento de ½ píxel, una imagen I, tres P y doce B.

con una resolución de 1280×720 píxeles. En este artículo se recogen algunas técnicas de optimización que permiten mejorar el rendimiento de alguno de los bloques funcionales. Las técnicas más destacadas son la escritura del código que implementa la IDCT y la VLD en lenguaje ensamblador para facilitar su paralelización, el empleo de las instrucciones SIMD para la lectura, procesado y almacenamiento de los píxeles involucrados en la compensación de movimiento y el empleo del controlador de DMA para todas las transferencias entre memoria interna y externa.

La Tabla 3-7 muestra el número de ciclos de reloj empleados por los diversos módulos funcionales que componen un descodificador MPEG-2 empleando una secuencia de un segundo con 2 imágenes tipo I, 8 tipo P y 20 tipo B y un 25% de coeficientes de la DCT distintos de cero. Los autores indican que a estos datos hay que añadirles una sobrecarga de 1.5 para funciones auxiliares por lo que realmente, para una secuencia de un segundo, se requieren 185 Mciclos/s lo que permite que a una frecuencia de 600 MHz se puedan descodificar 3.3 secuencias con definición estándar en paralelo.

	Análisis de la trama	IDCT	VLD y Cuantificación	Compensación de movimiento
[CCFS02]	$25.4 \cdot 10^6$	$31.6 \cdot 10^6$	$39.0 \cdot 10^6$	$27.5 \cdot 10^6$

Tabla 3-7. Número de ciclos de reloj por segundo invertidos por los módulos que componen el descodificador MPEG-2 optimizado propuesto en [CCFS02].

Finalmente, Anurag y otros investigadores del departamento de vídeo e imagen de Texas Instruments presentan en [JRGS02] el análisis del rendimiento de un codificador MPEG-2 optimizado sobre el procesador TMS320DM642. Las principales aportaciones de este trabajo a las metodologías de optimización son, por un lado, el empleo del controlador de QDMA en lugar del EDMA que se venía usando en los trabajos previos puesto que su programación requiere un menor tiempo; y, por otro, la utilización de la técnica de doble *buffer* que permite procesar un macrobloque en memoria interna, mientras que otro está siendo transferido por el QDMA hacia/desde memoria externa evitando de este modo que la CPU tenga que realizar esperas innecesarias. Los resultados del rendimiento que se facilitan indican el número de ciclos de reloj medios invertidos por la CPU para codificar una secuencia con 15 Mbps de régimen binario y una resolución de 720×480 píxeles. La Tabla 3-8 muestra el número de ciclos de reloj para diferentes configuraciones del codificador. Empleando un procesador a 600 MHz es posible codificar entre 62.5 y 27.3 imágenes por segundo logrando por tanto el funcionamiento en tiempo real.

Progresivo/Entrelazado	GOP	Patrón	Ciclos de Reloj
Progresivo	1	IPIPIP….	$237.36 \cdot 10^6$
Entrelazado	1	IPIPIP…	$249.75 \cdot 10^6$
Entrelazado	15	IPPP…	$455.16 \cdot 10^6$
Entrelazado	14	IBPBPBP...	$474.54 \cdot 10^6$
Entrelazado	15	IBBPBBP...	$479.16 \cdot 10^6$
Progresivo	15	IPPP…	$484.60 \cdot 10^6$
Progresivo	14	IBPBPBP...	$548.18 \cdot 10^6$
Progresivo	15	IBBPBBP...	$564.12 \cdot 10^6$

Tabla 3-8. Ciclos de reloj empleados por el codificador empleando diferentes configuraciones.

3.3.2 Técnicas aplicadas en la optimización en velocidad de codificadores/descodificadores MPEG-4

Las publicaciones relacionadas con la optimización de codificadores o descodificadores MPEG-4 para DSPs no son abundantes, siendo las referencias más significativas las que se detallan a continuación.

Byeong-Doo Choi y otros investigadores de las Universidades de Corea y Pensilvania presentaron en [CCKM03] un sistema de descodificación de audio AAC y vídeo en formato MPEG-4 SP en tiempo real empleando el DSP TMS320C5510 y un procesador ARM9 de propósito general. El ARM se encarga de procesar la trama de entrada mientras que el DSP realiza las tareas de descodificación, tanto del audio, como del vídeo. La comunicación de datos entre ambos procesadores se realiza mediante el uso de memoria compartida.

En la publicación se describen dos técnicas básicas de optimización del algoritmo. En la primera de ellas se propone el uso de doble *buffers* tanto para la recepción de los datos de referencia de las imágenes descodificadas previamente como para el almacenamiento de los macrobloques ya descodificados. De este modo se pueden realizar transferencias entre memoria interna y externa empleando uno de los *buffers* mientras que la CPU procesa los datos almacenados en el otro. La segunda de las técnicas se basa en emplear las instrucciones SIMD para realizar el cálculo de los píxeles intermedios cuando se utiliza la compensación de ½ píxel.

La Tabla 3-9 muestra el número de imágenes por segundo que es capaz de descodificar el DSP[58] antes y después del proceso de optimización. A la vista de los resultados se comprueba que el sistema es capaz de trabajar en tiempo real con secuencias en definición estándar, aunque en el artículo no se especifican ni las características de las secuencias (régimen binario, tamaño, etc.) ni la frecuencia del reloj del procesador.

Procesador	*Tennis*	*Soccer*	*News*
DSP antes de la optimización	28.30	22.23	28.10
DSP después de la optimización	43.94	37.52	42.19

Tabla 3-9. Comparativa entre el número de imágenes por segundo en formato MPEG-4 SP que es capaz de descodificar el DSP antes y después del proceso de optimización.

Por otra parte en [GSS+06] varios autores de la universidad de Huazhong en China proponen algunas técnicas que permiten la optimización de los accesos a los diferentes niveles de memoria de un descodificador MPEG-4 SP ejecutándose en un TMS320DM642. En este trabajo se plantea el problema que supone la división de la memoria interna del procesador entre memoria de propósito general y memoria caché, así como los criterios que se deben aplicar para alojar datos/código en dicha memoria.

Los autores proponen cuatro posibles configuraciones para las que se evalúa el rendimiento descodificando varias secuencias. Los modos de funcionamiento son los siguientes:

1. Toda la memoria interna L2 se configura como memoria caché.
2. Toda la memoria interna L2 se configura como memoria de propósito general. En dicha memoria se almacenan las funciones ejecutadas más frecuentemente

[58] Los datos presentados se corresponden con una secuencia que sólo contiene una trama elemental de vídeo por lo que no se realiza la descodificación de audio.

como la IDCT o las relacionadas con la interpolación de ½ píxel. Además se alojan en esta memoria una serie de *buffers* de tipo ping-pong[59] que almacenan los píxeles del macrobloque a procesar. El movimiento de datos desde/a estos *buffers* se realiza mediante el controlador de DMA.

3. De la memoria L2 se asignan 64 kB a memoria caché de nivel 2. El resto de la memoria se emplea para almacenar los *buffers* ping-pong empleados para la compensación de movimiento. En el espacio restante (19 kB) se alojan algunas funciones de uso habitual.

4. De la memoria L2 se asignan 32 kB a memoria caché de nivel 2. En el resto de la memoria de propósito general se almacenan los *buffers* ping-pong junto con la mayor parte de las funciones que implementan el descodificador.

La Tabla 3-10 muestra el número medio de ciclos de reloj necesarios para descodificar secuencias en formato CIF empleando las cuatro configuraciones propuestas. Como se aprecia los mejores resultados se obtienen para la configuración con 64 kB de caché de nivel 2.

Secuencia	256 kB caché	64 kB caché	32 kB caché	Sin caché
Foreman	$3.67 \cdot 10^6$	$2.76 \cdot 10^6$	$3.28 \cdot 10^6$	$2.84 \cdot 10^6$
Akiyo	$2.95 \cdot 10^6$	$2.25 \cdot 10^6$	$2.39 \cdot 10^6$	$2.27 \cdot 10^6$
Tempete	$3.88 \cdot 10^6$	$2.90 \cdot 10^6$	$3.12 \cdot 10^6$	$2.97 \cdot 10^6$

Tabla 3-10. Rendimiento del descodificador MPEG-4, expresado en ciclos de reloj necesarios para descodificar una imagen en formato CIF, con diferentes configuraciones de la memoria interna de nivel 2.

Finalmente, en [KHH+03] investigadores de la Universidad Nacional de Taiwán muestran las técnicas de optimización que han empleado para implementar un descodificador MPEG-4 SP utilizando un procesador TriMedia. La metodología propone la modificación del código del descodificador de referencia en tres etapas: en la primera etapa se realizan mejoras a nivel del algoritmo (*algorithm level*) como puede ser la sustitución de la IDCT original por un algoritmo rápido o la organización del código de los bucles para reducir el número de saltos. En la segunda etapa se realizan las modificaciones a nivel de instrucciones (*instruction level*) de modo que algunos fragmentos del código[60] son reescritos empleando instrucciones próximas al ensamblador (*custom ops*) que aprovechan la arquitectura SIMD del procesador. Por último, en la tercera etapa (*assembly level*), se modifica manualmente el código generado por el compilador para mejorar el paralelismo a nivel de instrucción.

Como resultado de estas optimizaciones los autores han logrado el funcionamiento en tiempo real para secuencias de tamaños similares a 4CIF[61]. La Tabla 3-11 muestra el tiempo medio empleado por el descodificador en procesar cada imagen con las versiones optimizada y sin optimizar del descodificador. Como se

[59] Los *buffers* ping-pong que se declaran para poder realizar la compensación de movimiento en memoria interna requieren 173 kB por lo que sólo quedan disponibles 83 kB de memoria interna.

[60] En el artículo no se dan detalles de las funcionalidades que han sido rescritas empleando estas instrucciones.

[61] Los autores no indican en el artículo las características con las que se han configurado las secuencias ni el régimen binario que poseen.

aprecia, el tiempo de descodificación se ha visto reducido en torno a un 20% en media, si bien no se indica el efecto que ha tenido individualmente cada una de las etapas de optimización propuestas.

	Optimizado Imágenes/Segundo	Sin Optimizar Imágenes/Segundo
Outbreak (4CIF)	73.0	33.8
ComingBack (512×384)	44.0	33.3
Crash (512×384)	42.1	33.3
LastStop (512×384)	39.7	33.4
Radius (640×352)	60.6	33.4

Tabla 3-11. Número de imágenes por segundos descodificadas por ambas versiones.

3.3.3 Técnicas aplicadas en la optimización en velocidad de codificadores/descodificadores H.264

Existen numerosas publicaciones en las que se presentan técnicas de optimización de codificadores y descodificadores para el estándar H.264 empleando diversos DSPs. Seguidamente se presentan las aportaciones que se han considerado más relevantes

En [WC06] investigadores de la Universidad Huaqiao en China proponen varias técnicas para optimizar el codificador de referencia H.264 (JM-8.1) para el núcleo TMS320C64x[62]. Las principales aportaciones de este trabajo se encuentran en la codificación de algunos módulos empleando la arquitectura SIMD del procesador y la organización de los datos en los niveles de memoria para minimizar el número de fallos en los accesos a la memoria caché.

Aprovechando las instrucciones SIMD, y a modo de ejemplo, en el mencionado trabajo se presenta la recodificación, de lenguaje C a ensamblador del procesador, del módulo que calcula la interpolación de ¼ píxel, pasando de emplear 672 ciclos de reloj en la versión codificada en C a requerir 30 en la versión codificada en ensamblador.

Por otra parte, para reducir el tiempo de acceso a la memoria externa se emplea el controlador de DMA para transferir los datos, de modo que el procesamiento que realiza la CPU siempre se lleva a cabo con datos alojados en memoria interna. Dado que el espacio de memoria interna es limitado[63], los autores proponen almacenar en ella el área de búsqueda del siguiente macrobloque a codificar, varios macrobloques ya codificados y los *buffers* necesarios para realizar operaciones intermedias.

El codificador ha sido configurado para el perfil *Baseline*, con un área de búsqueda de 16 píxeles, con sólo dos imágenes de referencia, sin permitir bloques de 4×4 píxeles y con un factor de cuantificación constante de valor 28. Con esta configuración el procesador es capaz de codificar 23 imágenes de tamaño QCIF por segundo.

[62] En esta publicación no se especifica el elemento concreto de la familia C64x que se ha empleado.

[63] En el artículo no se indica cómo se ha distribuido la memoria interna L2 entre memoria de propósito general y memoria caché.

En 2006, Zhuo, junto con otros investigadores de la Universidad Tecnológica de Beijing, presentan en [ZWFS07] el proceso de optimización de un codificador H.264 para el procesador TMS320DM642. La principal novedad que presenta este trabajo, respecto a otros en los que se proponen optimizaciones de algoritmos de codificación de vídeo, se centra en la descripción que se realiza del uso de los *buffers* ping-pong para permitir que la CPU procese los datos almacenados en uno de los *buffers*, mientras el controlador de DMA transfiere los del otro. Adicionalmente, en este trabajo se propone emplear las instrucciones que aprovechen la arquitectura SIMD para codificar algunos módulos (estimación/compensación de movimiento e ICT).

En este trabajo, el codificador implementa el perfil *Baseline*, con un GOP de 30 imágenes y un factor de cuantificación constante de valor 25. La Tabla 3-12 muestra el número de imágenes por segundo en formato CIF que es capaz de codificar un TMS320DM642 a 600 MHz antes y después de la optimización.

Secuencia	Imágenes por segundo Sin optimizar	Imágenes por segundo Optimizada
Akiyo	3.66	53.44
News	3.46	46.17
Foreman	2.77	33.55
Football	2.58	26.79
Mobile	2.90	26.07

Tabla 3-12. Número de imágenes (CIF) por segundo procesadas por el codificador propuesto en [ZWFS07].

Wang y otros investigadores de la Universidad Shandong en China presentan en dos artículos [WHL06] [WHL07][64] la optimización del codificador x264 [X264] para la plataforma TMS320DM642. La optimización realizada se centra en tres aspectos:

- Configurar la memoria interna de nivel 2 del DSP para que 128 kB se dediquen a memoria de propósito general y otros 128 kB a memoria caché de nivel 2. En la memoria de propósito general se almacena tanto código como datos[65].
- Operar con datos que siempre se encuentren alojados en memoria interna. Para ello se definen *buffers* de tipo ping-pong de modo que la CPU procesa los datos almacenados en uno de los *buffers* mientras que el controlador de DMA transfiere los datos del otro *buffer*. La Figura 3-23 ilustra este proceso, en el que se aprecian las transferencias realizadas por el controlador de DMA en la línea superior y el procesamiento de la CPU en la línea inferior.
- Aprovechar la arquitectura SIMD que posee el procesador TMS320DM642 para recodificar algunos módulos que poseen una elevada carga computacional. Concretamente, en [WHL06] se explica la optimación del cálculo de las muestras intermedias para el filtro de ¼ píxel y de la función que calcula el SAD.

[64] En cada uno de los artículos los autores presentan parte de las técnicas empleadas por lo que se ha optado por describirlas de forma conjunta.

[65] Los autores del trabajo no indican exactamente qué funciones o datos se alojan en esta memoria.

En los trabajos mencionados, el rendimiento de la versión optimizada se ha medido empleando 5 secuencias de resolución QCIF, codificando todas ellas con un factor de cuantificación de 24[66]. La Tabla 3-13 resume el número de imágenes por segundo codificadas, así como la pérdida que se produce en la relación señal ruido de pico (PSNR) respecto a la versión original como resultado de alguna de las optimizaciones.

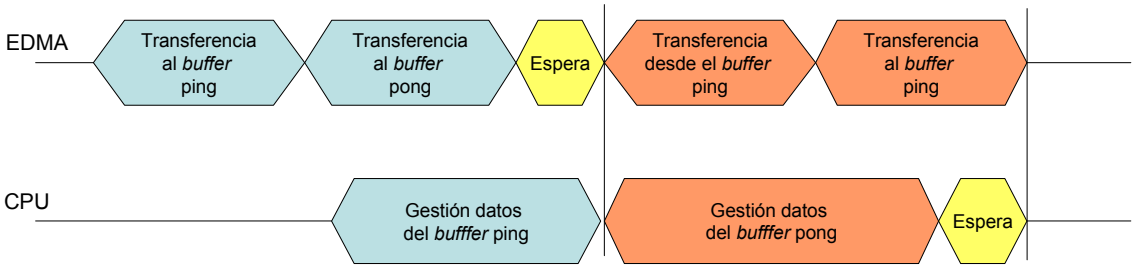


Figura 3-23. Paralelización de la transferencia de datos con el proceso de codificación.

Secuencia	Imágenes por segundo	ΔPSNR (dB)
Foreman	46.52	-0.24
Moth&Daughter	54.37	-0.19
Carphone	49.14	-0.08
News	58.93	-0.15
Highway	51.56	-0.14

Tabla 3-13. Número de imágenes por segundo y pérdida de calidad de la versión optimizada del codificador x264 propuesto por Wang en [WHL07].

En [YGLZ06] investigadores del instituto Harbin de Tecnología (China) explican de forma detallada la optimización del cálculo del filtro antibloques de un descodificador H.264 para una arquitectura C64x+. Como se indicó en el apartado 2.2.4.2.5.2, el filtro antibloques está dividido realmente en dos etapas, en la primera se evalúan los bordes que es necesario filtrar y en la segunda se realiza el filtrado propiamente dicho. La primera de las fases implica la evaluación de numerosas condiciones, lo que genera fallos en los accesos a la memoria caché. Para minimizar estos fallos, los autores proponen realizar los siguientes pasos:

1. Todos los píxeles necesarios para filtrar un bloque de 4×4 píxeles son siempre leídos empleando instrucciones de carga de varios píxeles en paralelo.

2. Se pre-calculan las seis condiciones usadas en el algoritmo. Para extraer estas condiciones se emplean instrucciones que realizan varias comparaciones simultáneamente. Finalmente las condiciones de 1 bit se convierten en máscaras de 8 bits empleando instrucciones de extensión de bits.

3. Se calculan todas las posibles salidas del filtro para cada píxel involucrado. Este proceso es muy eficiente ya que las unidades funcionales del procesador lo realizan en paralelo.

[66] Los autores no detallan en los trabajos los parámetros con los que se ha configurado el codiicador.

4. Las salidas del filtro son combinadas con las máscaras para asignar a los píxeles el valor adecuado en función de las condiciones que se pre-calcularon en el paso 2.

5. Las variables que contienen los 24 píxeles modificados se almacenan en el *buffer* que contiene la imagen reconstruida.

Los autores no presentan en este trabajo ni la mejora global del rendimiento del descodificador ni la que supone sobre el proceso de filtrado[67], ya que sólo detallan el número de ciclos de reloj que se invierten para cada bloque de 4×4 píxeles. En el caso de la luminancia son necesarios 268 ciclos para la componente vertical y 293 para la horizontal, mientras que para la crominancia se requieren 118 para la componente vertical y 128 para la horizontal. Como se puede apreciar, el número de ciclos que emplea la CPU en ejecutar el algoritmo optimizado es constante puesto que no hay condiciones que evaluar, a diferencia del algoritmo original que, en función de las condiciones que se cumplieran, requería diferentes tiempos de ejecución.

Otros autores como Werda [WKBM06] presentan la optimización del codificador UB Vídeo para el procesador TMS320DM642 en el que se pasa de codificar 55 imágenes por segundo a 65 empleando técnicas basadas en la transferencia de datos por DMA como las descritas en [WHL06].

En [VN07] investigadores de la Universidad Tecnológica de Singapur proponen la migración y optimización del codificador de referencia H.264 [JSVM] para una plataforma basada en TMS320DM642. La principal aportación de este trabajo se encuentra en la descripción de proceso de migración del código desarrollado para PC al entorno de desarrollo del DSP y el modo en el que se han resuelto los problemas que plantea este proceso. Desde el punto de vista de la optimización del código, los logros alcanzados no son significativos puesto que simplemente se han alojado las funciones con mayor carga computacional en la memoria interna de propósito general.

Li junto con investigadores de la Universidad Jiangsu de China, presenta en [LXZ08] la optimización del cálculo de los píxeles interpolados para realizar la estimación de movimiento en un codificador H.264 aprovechando la arquitectura SIMD del procesador TMS320DM642. El algoritmo propuesto es idéntico al descrito en [WHL06] por lo que no supone una aportación relevante desde el punto de vista de las técnicas de optimización del tiempo de ejecución.

Finalmente, Hu presenta en 2010 [HD10] una optimización del descodificador H.264 para el procesador Blackfin 533 a 600 MHz. En este trabajo los autores modifican el flujo de ejecución del descodificador de referencia [JSVM] para realizar el filtrado antibloques a nivel de macrobloque en lugar de a nivel de imagen[68]. En el descodificador modificado, se emplea el controlador de DMA para transferir los bloques de referencia necesarios para la compensación de movimiento que se encuentran en memoria externa a *buffers* ping-pong alojados en memoria interna. Adicionalmente se presenta una modificación del algoritmo de filtrado de bloques como la que se describe en el artículo publicado por el autor de esta tesis [PSG+08] y que se encuentra detallada en el apartado 4.4.2.3.2 de esta memoria. La Tabla 3-14

[67] A pesar de no disponer de datos sobre la optimización lograda por el algoritmo, se ha considerado interesante incluir esta referencia puesto que como se verá en el apartado 4.4.2.3.2 esta técnica se ha empleado dentro de los trabajos de optimización del descodificador H.264 desarrollado en esta tesis.

[68] El descodificador de referencia realiza todo el proceso de descodificación de entropía y de compensación de movimiento de toda la imagen para, posteriormente, aplicar el filtro antibloques a toda la imagen reconstruida.

muestra el número de imágenes por segundo de tamaño CIF[69] descodificadas con las versiones optimizada y sin optimizar del descodificador. Como puede verse[70] se logra el funcionamiento en tiempo real para este tamaño de imagen.

Secuencia	Sin optimizar (fps)	Optimizada (fps)	Mejora (%)
Akiyo	29.8	39.7	33.2
Vectra	17.8	26.5	48.9
Stefan	19.0	30.1	58.4
Mobile	16.2	27.9	72.2
Tempere	16.8	29.8	77.4

Tabla 3-14. Comparación de número de imágenes por segundo procesadas por las versiones optimizadas y sin optimizar del descodificador H.264 propuesto en [HD10].

3.4 Resumen

En este capítulo se ha realizado una revisión del estado del arte en tres aspectos importantes para el desarrollo de esta tesis. En primer lugar, se han analizado las soluciones tecnológicas que se vienen empleando en los últimos años para el desarrollo de codificadores y descodificadores de vídeo, con el objeto de poner en valor las ventajas y mostrar los inconvenientes que tienen los procesadores digitales de señal para la implementación de este tipo de aplicaciones. Como consecuencia de este análisis se ha comprobado que en la actualidad existen numerosos grupos de investigación que trabajan en la implementación de codificadores y descodificadores de vídeo empleando estos procesadores. Esta evidencia avala el interés que puede existir en la comunidad científica en disponer de una metodología general para reducir el tiempo de desarrollo de este tipo de sistemas sobre plataformas basadas en DSPs de última generación.

A continuación, se ha analizado el mercado de los procesadores digitales de señal para evaluar las similitudes y diferencias entre ellos. La principal conclusión extraída de este estudio se ha mostrado en la Tabla 3-5 y demuestra que la arquitectura interna de todos estos procesadores es muy similar, lo que garantiza que las metodologías de optimización que se sinteticen utilizando los datos de experimentos realizados con algunos de ellos, serán válidas para el resto sin demasiadas modificaciones.

Finalmente, se han revisado las técnicas de optimización de codificadores y descodificadores de vídeo que utilizan DSPs que han sido publicadas en la literatura científica por diferentes grupos de investigación a lo largo de los últimos años. En estas publicaciones se muestran diversas técnicas de optimización que han permitido a los autores desarrollar codificadores o descodificadores conformes con los diferentes estándares funcionando en tiempo real para diferentes tamaños de imagen.

En esta revisión no se ha localizado ninguna publicación en la que se presente una metodología general de optimización de este tipo de aplicaciones que se pueda aplicar desde los primeros pasos del proceso de desarrollo hasta llegar al funcionamiento en tiempo real, lo cual constituye el objetivo principal de esta tesis.

[69] Los autores no dan información respecto al perfil con el que se han codificado las secuencias. Sólo se indica que se codifican con una tasa binaria de 500 kbps y un GOP de 25 imágenes.

[70] Llama la atención la tremenda diferencia que se aprecia en la mejora del rendimiento entre las secuencias empleadas. Este detalle no se justifica en el trabajo presentado.

4 OPTIMIZACIÓN DEL TIEMPO DE EJECUCIÓN DE ALGORITMOS DE DESCODIFICACIÓN DE VÍDEO

En este capítulo se presentan los trabajos de optimización del tiempo de ejecución de los descodificadores de vídeo que implementan los estándares MPEG-2, MPEG-4 y H.264 que se han llevado a cabo en la tesis, utilizando diferentes DSPs. Las acciones realizadas, las decisiones tomadas y las soluciones adoptadas durante estos procesos de optimización han servido para sintetizar la metodología de optimización de descodificadores de vídeo sobre DSPs que se presenta en el capítulo 5.

La optimización de los diferentes descodificadores se ha llevado a cabo siguiendo un método de trabajo orientado a facilitar el posterior proceso de síntesis de la metodología de optimización. En el apartado 4.1 se explica este método de trabajo. En los apartados 4.2, 4.3 y 4.4 se explica el proceso de optimización de los descodificadores MPEG-2, MPEG-4 y H.264 respectivamente. Finalmente, en el apartado 4.5 se resumen los resultados obtenidos.

4.1 Método de trabajo

Para la síntesis de la metodología de optimización se ha empleado el siguiente método de trabajo:

- Optimización de los algoritmos de descodificación de diferentes estándares sobre diferentes DSPs.
- Evaluación de los resultados de la optimización utilizando simulaciones, pero también realizando medidas en tiempo real, con diferentes plataformas de desarrollo, e incluso en condiciones de funcionamiento reales.
- Documentación de las técnicas de optimización utilizadas en cada caso particular.
- Síntesis de la metodología de optimización mediante la selección de las técnicas y métodos susceptibles de ser aplicados con carácter general.

La optimización de diferentes algoritmos utilizando distintos DSPs permite una acumulación de experiencias que resulta fundamental para la extracción de la metodología. Durante este proceso se plantea una amplia variedad de problemas de optimización y se toma un conjunto amplio de decisiones para resolverlos. En este sentido, es deseable que el número de estándares y de DSPs con los que se trabaje sea grande para incrementar la cantidad y variedad de los problemas de optimización que se plantean y de las soluciones aportadas para resolverlos.

La medida de los resultados de la optimización es fundamental para evaluar la bondad de las decisiones que se toman en cada caso. Para que esta medida sea útil debe realizarse no sólo con el descodificador aislado, sino también cuando éste forma parte de un sistema completo, cuyas características en ocasiones tienen una influencia en las prestaciones de los algoritmos que no aparecen en las pruebas con los descodificadores aislados.

Finalmente, toda la experiencia acumulada durante los procesos de optimización y pruebas debe formalizarse de manera que resulte de utilidad para el proceso final de síntesis de la metodología. Los aspectos anteriores de la metodología de trabajo son abordados en mayor profundidad en los siguientes subapartados.

4.1.1 Optimización de diferentes algoritmos utilizando distintos DSPs

Debido a consideraciones prácticas relacionadas con la fuerza de trabajo y los sistemas de desarrollo disponibles, esta tesis se ha limitado a trabajar con tres estándares (MPEG-2, MPEG-4 y H.264, ver capítulo 2) y dos DSPs (TMS320DM642 y TMS320DM6437, ver apartados 3.2.1 y 3.2.2). Para realizar las pruebas se han utilizado tres tarjetas de prototipado (ver Anexo A) que integran los DSPs indicados anteriormente junto a las interfaces de entrada/salida necesarias. Por limitaciones de tiempo no ha sido posible optimizar todos los descodificadores empleando todas las tarjetas disponibles; en la Tabla 4-1 se muestran los que se han optimizado para cada una de ellas. Con todo, el conjunto es lo suficientemente amplio como para establecer las conclusiones metodológicas que se muestran en el capítulo 5.

Tarjeta / Estándar	DESCOS	EVMDM642	DM6437 EVM
MPEG-2	X	X	
MPEG-4 ASP[71]	X	X	
H.264 *Baseline Profile*	X	X	X
H.264 *Main Profile*	X	X	X

Tabla 4-1. Tarjetas de prototipado utilizadas y descodificadores de vídeo optimizados.

[71] A pesar de lo prometedor que parecía este estándar gracias a la reducción del ancho de banda que logra respecto a MPEG-2, durante el desarrollo de los trabajos de tesis orientados a optimizar su rendimiento, se detectó el escaso interés suscitado por dicho estándar en el entorno industrial. Este hecho lo corrobora que en la actualidad no existe ningún servicio de difusión de televisión digital que lo emplee, quedando su uso restringido a implementaciones parciales (formatos DivX [DIVX] o XviD [XVID]) de la norma orientadas al almacenamiento en ficheros. Por este motivo se decidió aplicar un esfuerzo limitado para la optimización de este descodificador y sólo se aplicaron las técnicas que ya se emplearon con MPEG-2.

El proceso de optimización del tiempo de ejecución de un algoritmo para un DSP comienza con la selección de un código de referencia adecuado para su implementación, continúa con la migración de ese código al DSP y termina con la optimización propiamente dicha.

4.1.1.1 Selección del código de referencia

Una opción que normalmente está disponible es el propio código de referencia del estándar que se vaya a implementar ([ISO05] [ISO01]). En ocasiones este código no resulta ser el más adecuado para ser utilizado como punto de partida para la optimización debido a varios motivos:

- El lenguaje de programación utilizado. Por ejemplo, los códigos implementados en C++ son más difíciles de portar, son compilados de forma menos eficiente por las herramientas de desarrollo de los DSPs y requieren esfuerzos de optimización extra.
- Suelen ser códigos complejos, voluminosos y poco optimizados en cuanto al consumo de memoria y de CPU.

Normalmente existen otras opciones, algunas de código abierto, que es conveniente evaluar ([FFMPEG], [DIVX], etc.). Para cada una de ellas es necesario realizar pruebas de conformidad que permitan garantizar que los códigos seleccionados sean compatibles con los perfiles y niveles del estándar que se desea implementar.

La elección de una u otra alternativa puede venir también determinada por otros factores, tales como el grado de documentación del código y la existencia de foros de desarrolladores que le den soporte.

4.1.1.2 Migración del código de referencia al DSP

Tanto los códigos de referencia de los diferentes estándares, como el resto de códigos que se han evaluado durante la tesis han sido desarrollados para entorno PC. Por esta razón, antes del proceso de optimización, es necesario realizar un proceso de migración al entorno de desarrollo del DSP. Este proceso tiene asociadas una serie de etapas para cada descodificador que se detallarán en el apartado correspondiente a la optimización de cada uno de ellos.

Al finalizar el proceso de migración se pueden realizar medidas del rendimiento del descodificador a partir del análisis del número de ciclos de reloj que invierte el DSP en la descodificación de diferentes secuencias. Estas primeras medidas se realizan con el código y los datos en memoria externa y permiten establecer una cota máxima del número de ciclos de reloj que invierte el procesador para descodificar cada una de las secuencias. Esta cota debe tomarse como referencia para las posteriores fases del proceso de optimización.

4.1.1.3 Proceso de optimización

El proceso de optimización tiene un doble objetivo, por un lado se trata de reducir el tiempo de ejecución de cada uno de los bloques funcionales del descodificador y, por otro, minimizar los efectos que tiene sobre el rendimiento del descodificador su integración en un entorno real.

Para lograr el primer objetivo, se analiza detalladamente la arquitectura del procesador sobre el que se desea realizar la implementación y se modifica el código para que se adapte de la mejor manera posible a esa arquitectura. Los elementos de

las arquitecturas de los DSPs utilizados en la tesis que tienen una mayor influencia[72] sobre el rendimiento de los descodificadores son los siguientes:

1. Arquitectura de la memoria. Estos procesadores disponen de varios niveles de memoria con tamaños y tiempos de acceso muy diferentes por lo que es necesario realizar una distribución adecuada tanto del código como de los datos en cada uno de los niveles.

2. Controlador de Acceso Directo a Memoria (DMA). El tiempo de acceso a la memoria externa es uno de los aspectos que más influye en la ejecución de los algoritmos de descodificación de vídeo puesto que estos deben realizar accesos masivos a datos que se encuentran en ella (como por ejemplo a las imágenes descodificadas previamente). Para solventar este problema, los DSPs analizados disponen de controladores de DMA que pueden ser configurados para que realicen transferencias de bloques de datos alojados en memoria externa, mientras la CPU procesa otros datos que se encuentran almacenados en sus registros o en la memoria interna.

3. Arquitectura SIMD (*Single Instruction Multiple Data*). A pesar de que las unidades funcionales que poseen los DSPs analizados son de 32 bits, éstas son capaces de realizar varias operaciones simultáneas con los datos de 8 ó 16 bits que se incluyen en cada palabra de 32 bits empleando instrucciones específicas. Esto permite procesar en paralelo varios datos (por ejemplo píxeles o coeficientes, en el caso de un descodificador) con una mejora sustancial en el rendimiento. A pesar de que los compiladores que se incluyen en los entornos de desarrollo de los DSPs son capaces de emplear estas instrucciones, en numerosas ocasiones el grado de optimización puede mejorarse sustancialmente escribiendo directamente el código en el lenguaje ensamblador del procesador.

Por otra parte es necesario integrar el descodificador una vez optimizado dentro de un sistema completo (como puede ser el STB-IP descrito en el Anexo B) en el que se incluyan otras tareas (descodificación de audio, análisis de trama, etc.) para valorar el efecto que éstas tienen sobre el rendimiento del descodificador. Una vez integrado se realizan una serie de optimizaciones a nivel de sistema que permiten minimizar la pérdida de prestaciones del descodificador.

4.1.2 Evaluación de los resultados de la optimización

La evaluación de los resultados de optimización se ha llevado a cabo mediante la medición del número de ciclos de reloj que requieren los DSPs para realizar la descodificación de diferentes secuencias de vídeo. Estas medidas se han realizado sobre el descodificador aislado, tanto en simulación, como en tiempo real, tal y como se detalla en el apartado 4.1.2.1.

Las medidas con el descodificador aislado proporcionan información rápida y precisa sobre la bondad de cada modificación dentro del proceso de optimización. No obstante, los descodificadores nunca se utilizan aisladamente, sino que forman parte de sistemas más complejos que incluyen interfaces de entrada y salida de datos, otros descodificadores (por ejemplo de audio) e interfaces de usuario. La utilización de un descodificador dentro del sistema completo tiene una influencia sobre el rendimiento global del sistema que es necesario evaluar. Para poder medir esta influencia, durante

[72] El buen uso de estos elementos de las arquitecturas de los DSPs permite reducir el tiempo empleado por el procesador para ejecutar los descodificadores compatibles con los diferentes estándares. Para el caso del descodificador MPEG-2, la versión optimizada emplea aproximadamente un 10% del tiempo de la versión original, para el descodificador MPEG-4 es en torno al 12% y para el descodificador H.264 alrededor del 8%. Estos resultados se muestran en los apartados 4.2.4.2, 4.3.4 y 4.4.4 respectivamente.

el desarrollo de la tesis, se ha diseñado el sistema completo de descodificación de televisión digital que se detalla en el Anexo B. Las pruebas realizadas y las optimizaciones que se han llevado a cabo para garantizar el correcto funcionamiento de todo el sistema se explican en el apartado 4.1.2.2.

Finalmente, se ha considerado de utilidad la realización de pruebas simulando situaciones reales de funcionamiento. En estas pruebas, más que evaluar los resultados de la optimización, se persigue comprobar la funcionalidad de los descodificadores optimizados de forma más exhaustiva. Las pruebas con situaciones reales de funcionamiento se describen en el apartado 4.1.2.3.

4.1.2.1 Pruebas con el descodificador aislado

En este caso el descodificador se integra en un banco de test que permite leer las secuencias codificadas desde un fichero, descodificarlas y almacenar el resultado en un nuevo fichero con formato YUV 4:2:0. La estructura del banco de pruebas es, en todos los casos, la que se muestra en la Figura 4-1.

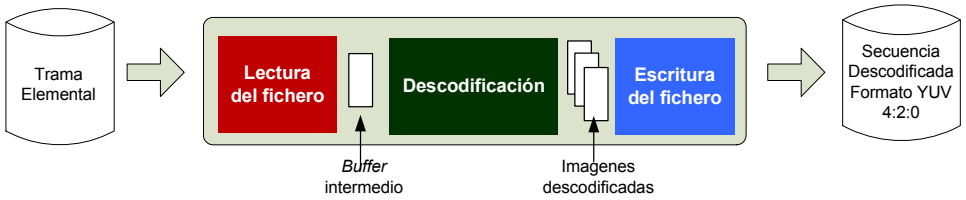


Figura 4-1 Banco de pruebas empleado para medir las prestaciones del descodificador aislado.

Las medidas del rendimiento se han realizado empleando la herramienta de análisis de ejecución del código en simulación que posee el entorno de desarrollo de los DSPs denominada *profiler* [TI11]. Dicha herramienta se configura para medir el número de ciclos de reloj que tarda en ejecutarse cada una de las funciones que componen el descodificador a lo largo de la descodificación de una secuencia. Los resultados obtenidos permiten evaluar la mejora que provoca la inclusión de cada una de las técnicas de optimización que se aplican sobre el descodificador.

Para realizar estas medidas se han empleado dos tipos de secuencias; por una parte se seleccionan algunas de las definidas en el propio estándar y, por otra, se generan secuencias a partir de emisiones de televisión reales y de películas almacenadas en DVD. Debido a la lentitud del proceso de medida, las pruebas no pueden realizarse con un número muy elevado de imágenes por lo que en todos los casos[73] se han descodificado 100 imágenes, como máximo.

Finalmente, para validar los resultados obtenidos en simulación, se ha ejecutado el banco de pruebas sobre las tarjetas de prototipado (ver Anexo A) y se han realizado nuevamente medidas de rendimiento. Dado que la herramienta de *profile* sólo puede utilizarse en simulación, es necesario emplear otro método para realizar la medida del rendimiento. El método está basado en la utilización de un temporizador de 32 bits disponible en el DSP. El valor del temporizador se lee tanto al comienzo del fragmento de código a medir, como al final; calculando la diferencia entre ambos valores y conociendo la frecuencia de reloj a la que funciona el temporizador, se obtiene el tiempo invertido en la ejecución del fragmento de código. Los resultados obtenidos en todos los casos corroboran las medidas realizadas en simulación puesto que el error es inferior al 1%.

[73] Algunas secuencias de las descritas en los estándares poseen menos de 100 imágenes por lo que en estos casos se ha descodificado la secuencia completa.

4.1.2.2 Pruebas con el descodificador integrado en un sistema de descodificación completo

Con objeto de validar los resultados obtenidos en simulación en un escenario en el que el descodificador se integra en un sistema completo de descodificación de televisión digital, se ha diseñado el STB-IP que se detalla en el Anexo B. La existencia de otras tareas en el sistema (descodificador de audio o el analizador de la trama de entrada) hacen que algunos recursos del procesador como la memoria interna, la memoria caché o el controlador de DMA sean compartidos entre todas las tareas del sistema, lo que influye en su rendimiento.

Empleando este STB-IP se ha diseñado un entorno de pruebas como el mostrado en la Figura 4-2 en el que un ordenador personal encapsula y transmite, empleando paquetes IP[74], ficheros que contienen tramas de transporte MPEG-2 (ver Anexo B, apartado 8.2.1.1) previamente grabadas. El STB-IP desencapsula las tramas elementales de audio y vídeo, las descodifica y las muestra en un televisor.

Para poder comparar los resultados obtenidos con este banco de test y el correspondiente al descodificador aislado (apartado 4.1.2.1.) se deben emplear las mismas secuencias y descodificar el mismo número de imágenes en ambos casos. Para ello es necesario incluir las secuencias de vídeo codificadas en una trama de transporte[75]. Para las secuencias procedentes del estándar no existe una trama de audio asociada por lo que la trama de transporte sólo incluye la secuencia de vídeo; sin embargo, para las tramas procedentes de un DVD o de una emisión de televisión digital, sí se ha incluido el audio asociado al fragmento de vídeo. Las tramas de transporte generadas se almacenan en ficheros que son difundidos a través de la red Ethernet empleando la herramienta VLC.

Las medidas de rendimiento en este caso se realizan empleando un temporizador interno del DSP, tal y como se ha descrito en el apartado 4.1.2.1.

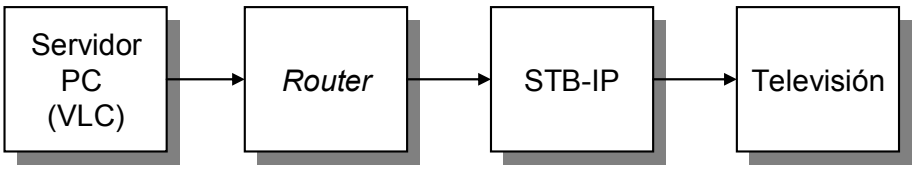


Figura 4-2 Banco de pruebas empleado para medir el rendimiento de los descodificadores de vídeo integrados en un STB-IP.

4.1.2.3 Pruebas en entornos reales de difusión de televisión digital

El tercer entorno de pruebas utilizado tiene como objeto analizar el funcionamiento del descodificador en situaciones reales, en las que el sistema completo en el que se encuentra integrado recibe tramas de larga duración procedentes de una transmisión vía satélite o de un codificador de vídeo en tiempo real. Con este entorno se pretende poner de manifiesto problemas que pueden surgir a nivel de sistema cuando se descodifican secuencias largas y que no aparecen cuando se realizan las pruebas con pequeños fragmentos de vídeo.

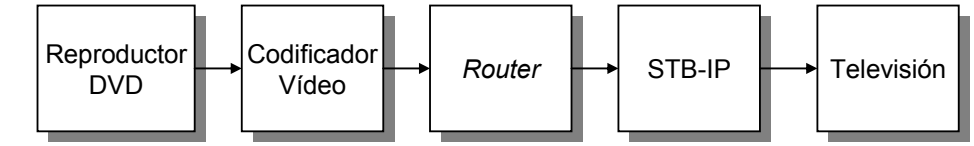
Para realizar pruebas de funcionamiento en este entorno se han diseñado dos configuraciones. La primera es similar a la empleada en el apartado anterior, y mostrada en la Figura 4-2, con la diferencia de que ahora la trama es generada a partir

[74] Para realizar el encapsulado en la trama de transporte se emplea la herramienta VLC [VLC].

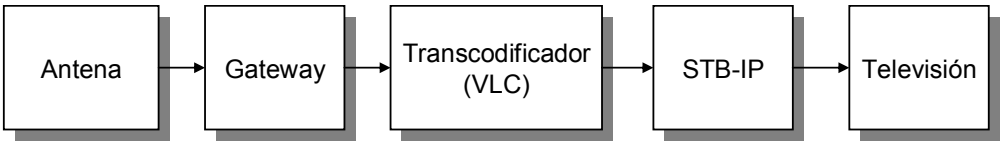
[75] El encapsulado de las tramas elementales de audio y vídeo se ha realizado empleando la herramienta Elecard XMuxer Pro [XMUX].

de los datos almacenados en un DVD en lugar de emplear ficheros pregrabados. Un reproductor de DVD genera la salida analógica que es conectada a un codificador que genera en tiempo real las tramas elementales de audio y vídeo que son encapsuladas inicialmente en una trama de transporte MPEG-2 y, posteriormente, en paquetes IP para ser distribuidos a través de la red Ethernet. El banco de pruebas se muestra en la Figura 4-3-a[76].

La segunda configuración emplea un *gateway* [ETHTV] comercial que sintoniza un transpondedor de satélite y encapsula cada uno de los programas disponibles en una trama de transporte MPEG-2 que es transmitida a través de una dirección IP *unicast*. La trama de transporte generada es recibida por un transcodificador [VLC], que recodifica la trama elemental de vídeo de acuerdo al estándar que se desee (si es necesario) y vuelve a encapsularla en una nueva trama de transporte MPEG-2, que es redistribuida de nuevo a través de la red local. La Figura 4-3-b muestra el esquema de esta segunda configuración[77].



a) Banco de pruebas empleando un reproductor de DVD.



b) Banco de pruebas empleando una trama recibida a través de una antena.

Figura 4-3 Bancos de pruebas empleados para medir el rendimiento en un entorno real.

4.1.3 Documentación de las técnicas de optimización utilizadas

Las técnicas utilizadas durante los procesos de optimización de los diferentes descodificadores empleando diferentes DSPs se documentan de una manera simple, mediante fichas, con el siguiente formato:

Referencia: A cada técnica empleada se le asigna un código numérico y una etiqueta para poder referenciarla posteriormente con facilidad (por ejemplo [ficha 000]).
Descripción: Breve descripción del procedimiento de optimización llevado a cabo.
Aplicabilidad: Características arquitecturales que deben tener los DSPs, o características que deben tener los algoritmos de descodificación, para que sea posible aplicar el procedimiento de optimización que se indica.

[76] En el caso del estándar MPEG-2 no es necesario recodificar los datos almacenados en el DVD puesto que estos ya se encuentran codificados conforme a dicho estándar. En este caso, en lugar de emplear un codificador MPEG-2, se encapsulan directamente las tramas elementales incluidas en el DVD en una trama de transporte que es distribuida a través de la red Ethernet.

[77] Para el caso del estándar MPEG-2 no es necesario transcodificar la trama de transporte que genera el *Gateway* puesto que las tramas elementales de vídeo que se encuentran encapsuladas en ella están directamente codificadas en MPEG-2.

La documentación de cada técnica se realiza en paralelo con los procesos de optimización y pruebas de rendimiento, de forma que para cada una de ellas se elabora una ficha. En la presente memoria, las fichas que documentan las diferentes técnicas aplicadas aparecen entremezcladas con la explicación del proceso de optimización de los diferentes descodificadores (apartados 4.2, 4.3 y 4.4). Adicionalmente, y para facilitar su consulta, también se han agrupado en el Anexo C.

Al finalizar los diferentes procesos de optimización, se dispone de una documentación que se toma como punto de partida para la síntesis de la metodología. Dicha síntesis se realiza posteriormente revisando esta documentación, identificando las técnicas que, por haber sido utilizadas con éxito en todas las ocasiones, o por otras razones, se considere que son susceptibles de ser generalizables.

4.2 Optimización del tiempo de ejecución de un descodificador MPEG-2

En este apartado se presenta el proceso seguido para optimizar el tiempo de ejecución de un descodificador acorde al perfil principal y nivel principal (*Main Profile-Main Level* o MP@ML) de la norma MPEG-2 (ISO/IEC 13818-3). En primer lugar se analiza el *software* de referencia que se ha empleado como punto de partida y se explican los pasos que se han dado para migrar el descodificador al entorno de desarrollo del DSP. A continuación se exponen las técnicas de optimización que se han empleado[78] y se analiza su posible aplicación, tanto a otros descodificadores de vídeo, como a otros procesadores. Para finalizar, se presentan los resultados obtenidos utilizando los tres entornos de pruebas descritos en el apartado 4.1.2.

4.2.1 Elección del *software* de referencia

Para materializar el descodificador MPEG-2 se tomó como punto de partida el *software* incluido en la parte 5 de la norma [ISO05] puesto que, en el momento en que comenzaron los trabajos de esta tesis, era el único descodificador compatible con el estándar a cuyo código fuente[79] se tuvo acceso. Este *software* soporta los perfiles Simple y Principal definidos en el estándar; sin embargo, en el caso del descodificador desarrollado, se ha optado por limitar la funcionalidad a la combinación perfil principal-nivel principal puesto que es la combinación que se emplea habitualmente tanto en las emisiones de televisión digital como en las películas almacenadas en DVD.

Este código de referencia está desarrollado en lenguaje C y se encuentra preparado para ejecutarse sobre plataforma PC. Los datos se manejan siempre a nivel de *byte* y los módulos que lo componen no están adaptados a la arquitectura de ningún procesador concreto. En los siguientes apartados se describe el modo en el que el *software* de referencia implementa el algoritmo y el proceso llevado a cabo para migrar el descodificador al entorno de desarrollo del DSP.

[78] El proceso de optimización se ha llevado a cabo sobre el procesador TMS320DM642 (ver apartado 3.2.1) empleando las tarjetas de prototipos DESCOS (ver apartado 7.1) y EVMDM642 (ver apartado 7.2).

[79] Puesto que se trata del *software* de referencia definido en el estándar no fue necesario realizar las pruebas de conformidad puesto que se da por hecho que es capaz de descodificar todas las secuencias definidas en dicho estándar.

4.2.1.1 Descripción del algoritmo

El descodificador implementado con el código de referencia de la norma MPEG-2 está integrado en un banco de test similar al mostrado en la Figura 4-1 en el que la secuencia es leída desde el fichero en fragmentos de tamaño constante que son almacenados en un *buffer* intermedio. A continuación se analiza la trama para extraer los elementos sintácticos de las cabeceras de secuencia, grupo de imágenes, rebanada (*slice*) e imagen propiamente dicha. Posteriormente se ejecuta un bucle en el que se descodifican los macrobloques de forma secuencial. Cada macrobloque descodificado se almacena en uno de los tres *buffers* de imágenes reconstruidas (I, P o B). Una vez que se dispone de una imagen completa, ésta se almacena en un fichero de salida. Cuando finaliza la descodificación del fragmento de secuencia leído, se realiza una nueva lectura y así sucesivamente hasta el final del fichero.

La Figura 4-4 muestra el ordinograma de ejecución del bucle de descodificación para cada rebanada. Inicialmente se extrae la información de la cabecera de rebanada para a continuación procesar el primero de los macrobloques que la componen. La información asociada a cada macrobloque depende de si se trata de un macrobloque *inter* o *intra*. En el primero de los casos es necesario obtener los vectores de movimiento[80] antes de extraer los coeficientes de la DCT, mientras que en el segundo, no existen dichos vectores por lo que se pasa directamente a extraer los coeficientes.

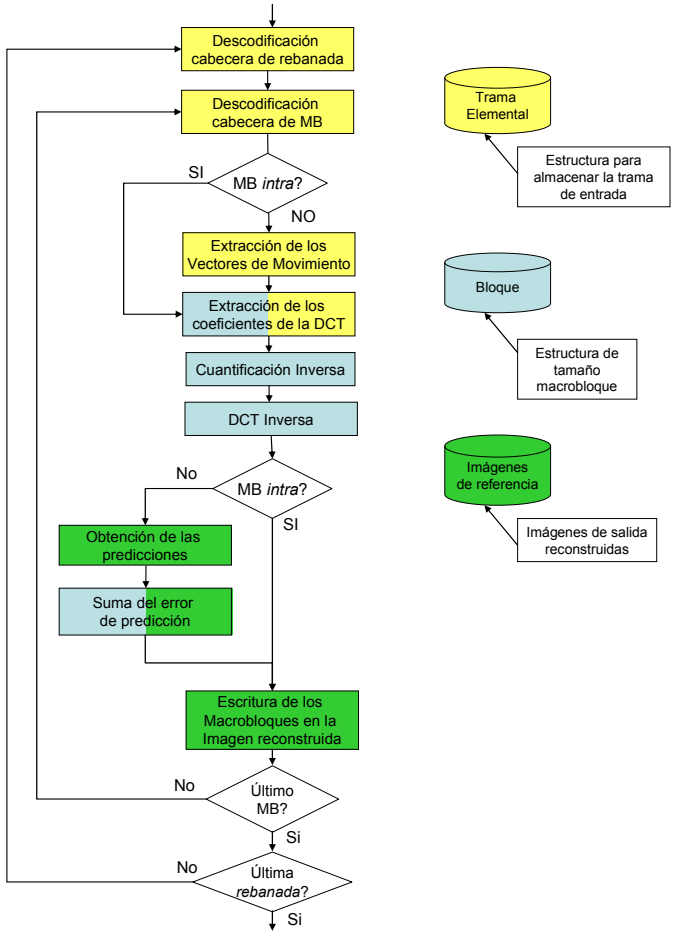


Figura 4-4. Ordinograma del bucle principal de descodificación de una rebanada en MPEG-2.

[80] Dependiendo del tipo de imagen y del tipo de macrobloque el número de vectores de movimiento que es necesario extraer varía entre 1 y 4.

Seguidamente se realiza la cuantificación inversa y la DCT inversa sobre los coeficientes extraídos para obtener el error de predicción. Si el macrobloque es de tipo *intra*, se obtiene el macrobloque reconstruido que es almacenado directamente en el correspondiente *buffer* de imágenes reconstruidas; si por el contrario se trata de un macrobloque de tipo *inter*, antes de almacenar el macrobloque reconstruido es necesario calcular la predicción a partir de los vectores de movimiento y sumarle el error de predicción obtenido previamente. El proceso descrito se repite para todos los macrobloques de una rebanada y para todas las rebanadas de una imagen.

Junto al ordinograma se han representado con diferentes colores las estructuras de datos principales que se emplean, de modo que cada parte del algoritmo se ha coloreado para relacionarlo con las que utiliza. La funcionalidad de cada una de ellas es la siguiente:

- Trama Elemental. Almacena los fragmentos de la trama elemental MPEG-2 que emplea el descodificador.
- Bloque. Contiene inicialmente los coeficientes de la DCT correspondientes a un macrobloque extraídos de la trama elemental sobre los que se aplican los procesos de cuantificación inversa y DCT inversa. El resultado de estos dos procesos se almacena sobre esta misma estructura. Su tamaño es constante e igual al tamaño de un macrobloque[81].
- Imágenes de referencia (o imágenes reconstruidas). Contiene las imágenes descodificadas. Se trata en realidad de tres *buffers* en los que se almacenan las imágenes descodificadas de salida. El tamaño de cada uno de estos *buffers* es igual al tamaño de la imagen que se esté descodificando[82].

Para facilitar la explicación del modo en que se paralelizan las diferentes etapas por las que pasa el proceso de descodificación de cada macrobloque con predicción, éste se ha dividido en las fases que se muestran en la Figura 4-5.

En primer lugar se lee del *buffer* denominado trama elemental el tipo de macrobloque a descodificar, se obtiene el vector de movimiento asociado y se calcula la dirección física donde se encuentra el macrobloque de referencia en alguna de las imágenes descodificadas anteriormente (fase A); seguidamente, se leen de la trama elemental los coeficientes de la DCT, se almacenan en el *buffer* denominado bloque y se aplica sobre ellos la IDCT para obtener el error de predicción que se guarda sobre el mismo *buffer* (fase B); a continuación, se leen del *buffer* denominado imágenes de referencia los píxeles necesarios para calcular el macrobloque de referencia (fase C), después se realiza la compensación del movimiento y la suma del error de predicción (fase D); y finalmente, se escribe el macrobloque reconstruido en el *buffer* de imágenes de referencia correspondiente (fase E).

Todas las fases que aparecen en el diagrama son ejecutadas por la CPU secuencialmente de modo que hasta que no finaliza una de ellas no puede comenzar la siguiente.

[81] Se trata de un *array* que contiene los 6 bloques de 8×8 coeficientes que componen un macrobloque. Cada dato es de 16 bits para poder almacenar los coeficientes de la DCT (ver apartado 2.1.2.4.2). El tamaño total del *buffer* es de 768 *bytes*.

[82] En el caso de las imágenes PAL cada una de ellas requiere 720×576 *bytes* para la luminancia y 360×288 *bytes* para cada una de las crominancias (si se emplea el submuestreo 4:2:0). El tamaño total de los tres *buffers* para esta resolución es de 1822.5 kB.

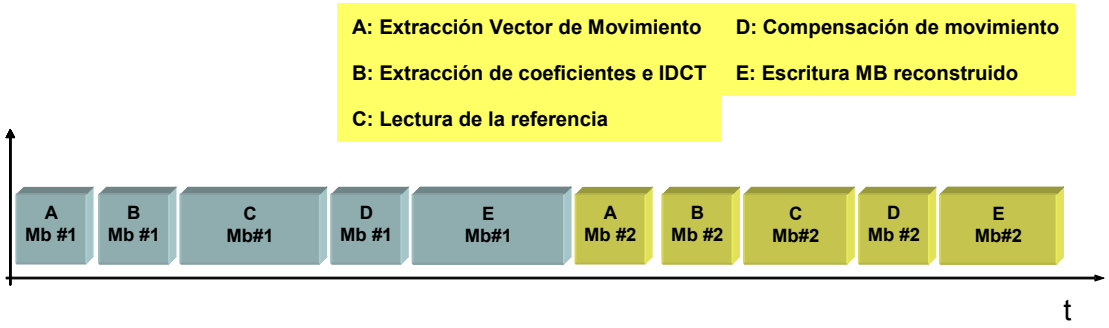


Figura 4-5. Fases por las que pasa la descodificación de un macrobloque con predicción.

4.2.1.2 Migración del descodificador MPEG-2 al entorno de desarrollo del procesador TMS320DM642

Para migrar el *software* de referencia al entorno de desarrollo del DSP [TI06], es necesario diseñar un proyecto que incluya los ficheros fuente que materializan el descodificador y la configuración del sistema operativo en tiempo real (*Real Time Operating System* o RTOS) que emplea el procesador (DSP/BIOS) [DSPBIOS]. Además es necesario definir el tamaño de la pila y la *heap*, que serán alojadas inicialmente en memoria externa. Para el caso del descodificador MPEG-2 se ha asignado inicialmente un tamaño a ambas secciones de 1 MB que será ajustado en posteriores etapas del proceso de optimización.

El descodificador de referencia está escrito en lenguaje C sin optimizaciones para ninguna arquitectura concreta, por lo que se ejecuta correctamente sobre el DSP sin necesidad de realizar cambios sobre el mismo. Para verificar su funcionamiento se descodificó un conjunto amplio de las secuencias descritas en la parte 4 del propio estándar [ISO04], siendo las secuencias descodificadas idénticas a las obtenidas en el PC[83].

Además, sobre esta versión se realizaron medidas de rendimiento del descodificador que permitieron determinar el número de imágenes por segundo procesables, así como las funciones con mayor coste computacional. Estas primeras medidas sirven de referencia en el posterior proceso de optimización.

Para realizar las medidas de rendimiento se emplearon dos tipos de secuencias; por una parte se seleccionaron algunas de las secuencias de test descritas en el estándar y por otra se grabaron fragmentos de diferentes canales de televisión digital, así como de algunas películas almacenadas en DVD.

Para presentar los resultados del proceso de optimización se han seleccionado dos secuencias que se han considerado representativas. De entre las primeras se ha elegido la denominada GI_9 puesto que se trata de una secuencia muy exigente para el descodificador por su elevado régimen binario. Por otro lado, entre las secuencias grabadas de emisiones de televisión digital, la que se ha considerado más significativa, debido a que presenta gran cantidad de eventos y continuos cambios de planos, es la que se ha extraído del canal de televisión por satélite “Arte TV” (de aquí en adelante esta secuencia será referida como “Arte”). Las características con las que está realizada la codificación de ambas secuencias son las siguientes:

[83] Para realizar la comparación entre las secuencias descodificadas por parte del DSP y del PC se empleó la herramienta *Elecard YUV Viewer* [ELECARD] que compara píxel a píxel los datos de ambas secuencias calculando el PSNR de cada una de las tres componentes (Y, C_R y C_B).

- Secuencia GI_9. Secuencia de 15 imágenes[84] de 720×480 píxeles con 15 Mbps de régimen binario y compensación de movimiento de ½ píxel.
- Emisión de TV digital (Arte). Secuencia con una resolución de 720×576 píxeles, un régimen binario medio de 3 Mbps, compensación de movimiento de ½ píxel y una proporción de una imagen de tipo I por cada tres imágenes de tipo P y una de tipo P por cada dos de tipo B.

Para medir el rendimiento se ha contabilizado el número medio de ciclos de reloj que emplea la CPU para la ejecución de cada una de las funciones. Posteriormente, éstas han sido agrupadas para conocer el tiempo de ejecución de cada uno de los bloques funcionales que componen el descodificador. La Tabla 4-2 presenta la media del número de ciclos de reloj empleados por el procesador para descodificar una imagen así como el porcentaje del tiempo total invertido en cada uno de los bloques funcionales que lo componen.

Bloque funcional	Arte		GI_9	
	Ciclos de reloj ($x10^6$)	% Total	Ciclos de reloj ($x10^6$)	% Total
Descodificación VLD y cuantificación inversa	193.82	11.85%	599.45	27.08%
IDCT	596.08	36.46%	521.75	23.57%
Compensación de movimiento	836.93	51.19%	1085.62	49.05%
Otras funciones	8.10	0.49%	6.44	0.29%
TOTAL	1634.93 10^6		2213.26 10^6	

Tabla 4-2. Número de ciclos de reloj empleados para descodificar una imagen usando las dos secuencias.

Para las secuencias seleccionadas, el descodificador emplea una media[85] entre 1600 y 2200 millones de ciclos de reloj en descodificar una imagen[86]. Esto equivale, trabajando a 600 MHz, a 0.37 imágenes por segundo para la secuencia 'Arte' y a 0.27 para la secuencia GI_9, suponiendo que ésta es la única tarea que realiza el DSP.

4.2.2 Optimización del tiempo de ejecución del descodificador MPEG-2

En los siguientes apartados se explica cómo se han explotado las características de la arquitectura del DSP TMS320DM642 para mejorar el rendimiento del descodificador MPEG-2. Aunque los diversos pasos se presenten de forma secuencial y aparentemente aislada, en realidad se trata de un proceso iterativo en el que algunas modificaciones influyen significativamente sobre el resto.

Tal y como se indicó en el apartado 4.1.3, para cada una de las técnicas de optimización se incluye una ficha en la que se le asigna un código, se realiza una

[84] La secuencia esta compuesta de una imagen I, tres P y once B.

[85] Esta medida se corresponde con la media aritmética obtenida como resultado de dividir el número total de ciclos de reloj necesarios para descodificar la secuencia completa entre el número de imágenes que posee dicha secuencia.

[86] La elevada diferencia entre el número de ciclos de reloj que emplea el procesador para descodificar ambas secuencias se debe a que la GI_9 tiene una tasa binaria de 15 Mbps, mientras que la obtenida de la emisión de televisión digital tiene una tasa de 3 Mbps. Esta diferencia provoca que el número de ciclos necesarios para realizar la VLD sea mucho mayor en la primera que en la segunda.

breve descripción de la misma y se indica su posible empleo, tanto en otros descodificadores, como con otras arquitecturas de procesadores.

Los siguientes apartados muestran las técnicas que se han aplicado agrupadas en los tres apartados que se mostraron al definir el método de trabajo: arquitectura de la memoria, controlador de acceso directo a memoria y arquitectura SIMD.

4.2.2.1 Arquitectura de la memoria

En relación con la gestión de la memoria del procesador TMS320DM642, hay dos factores que tienen una gran influencia en el rendimiento de los algoritmos que se ejecutan. Por una parte, la distribución que se realice de la memoria interna de nivel 2, entre memoria caché L2 y memoria de propósito general (ver apartado 3.2.1); y, por otra, la ubicación en la memoria interna o externa de los *buffers* y las funciones que se utilizan en el código.

4.2.2.1.1 Configuración de la memoria interna de nivel 2

En la fase inicial del trabajo de optimización no se dispone de ningún criterio para realizar el reparto de la memoria interna de nivel 2 entre memoria caché y memoria de propósito general. Para tener una referencia, en primer lugar, se configuró toda ella como caché y se realizaron medidas de rendimiento sobre un conjunto de secuencias. En la Tabla 4-3 se muestran los resultados del rendimiento para dos de esas secuencias[87]. Como era de esperar, la mejora producida al habilitar la memoria caché es muy grande en todos los bloques funcionales en los que se divide el algoritmo. Además de los datos incluidos en las tablas anteriores, para cada secuencia de prueba se obtuvieron y analizaron los datos de rendimiento de las diferentes funciones, lo que permitió identificar las que tienen un coste computacional mayor.

Bloque funcional	Arte		GI_9	
	Ciclos de Reloj ($x10^6$)	Mejora	Ciclos de Reloj ($x10^6$)	Mejora
VLD+IQ	10.35	95.66%	28.66	88.32%
IDCT	20.32	96.59%	18.70	96.41%
MC	25.87	96.90%	33.78	96.89%
Otras	1.06	86.91%	0.47	92.70%
TOTAL	57.06	96.51%	81.61	96.31%

Tabla 4-3. Ciclos de reloj de cada uno de los bloques funcionales tras el uso de la caché de nivel 2 en las secuencias TV digital y GI_9.

Los resultados anteriores deben tomarse como una referencia para ser comparados con posteriores repartos de la memoria interna con los que el rendimiento del descodificador debería ir mejorando. Utilizando los datos de rendimiento antes mencionados, se realizó una ordenación de las funciones teniendo en cuenta los potenciales beneficios que tendría su ubicación en la memoria interna de propósito general. Lo mismo se hizo con los *buffers* de datos. A falta de otros criterios, se priorizaron las funciones con un mayor coste computacional en la ejecución, así como los *buffers* utilizados por estas funciones. Esto permitió explorar otros repartos; en concreto, se realizaron medidas de rendimiento para repartos con 64 kB y 128 kB de memoria interna.

[87] Las secuencias son las mismas que las indicadas en el apartado 4.2.1.2.

Los resultados con memorias internas de 64 kB y 128 kB mejoraban en ambos casos los mostrados en las tablas anteriores. No obstante, conforme fue progresando el proceso de optimización, se observó que no en todas las ocasiones era mejor el mismo reparto. Esto resulta bastante lógico dado que, durante el proceso de optimización, las funciones cambian su tamaño y su velocidad de ejecución. También se crean nuevos *buffers* de datos mientras que otros desaparecen.

La conclusión es que el reparto de la memoria de nivel 2 debe ser un proceso iterativo, que debe revisarse cada vez que se realizan cambios sustanciales en la optimización del código.

Referencia: Configuración de la memoria interna [ficha 001].
Descripción: Inicialmente configurar la totalidad de la memoria interna como caché. Realizar pruebas de rendimiento global y por funciones y tomarlas como una referencia del rendimiento. Realizar una ordenación de las funciones y los *buffers* en función del interés de su ubicación en memoria interna. Priorizar las funciones con un mayor coste computacional y los *buffers* que sean utilizados por esas funciones. Realizar pruebas con diferentes repartos y almacenar los resultados. Repetir la ordenación y las pruebas cada vez que se produzcan cambios importantes en la estructura del código a lo largo del proceso de optimización.
Aplicabilidad: Procesadores en los que es posible configurar un reparto entre la memoria caché y la memoria interna de propósito general.

Una vez finalizado el proceso de optimización, la memoria interna de nivel 2 quedó dividida en 128 kB para la memoria caché y 128 kB para la memoria de propósito general. Esta configuración permitió alojar en esta memoria interna todas las funciones (73.3 kB[88]) y todos los *buffers* intermedios implicados en la descodificación de cada macrobloque.

Esta es una situación óptima, dado que el proceso de descodificación realiza las mismas tareas y utiliza los mismos *buffers* para descodificar cada uno de los macrobloques de la imagen, de manera que la inclusión de todas estas funciones y *buffers* en memoria interna produce una reducción drástica de los fallos en caché.

Referencia: Ubicación de funciones y *buffers* en memoria interna [ficha 002].
Descripción: Durante el proceso de optimización se procurará limitar el tamaño de las funciones y *buffers* que realizan la descodificación de los macrobloques de manera que quepan en la memoria interna disponible. La pertenencia al conjunto de funciones y *buffers* que realizan la descodificación del macrobloque se utilizará como criterio adicional para priorizar la inclusión de estos elementos en la memoria interna.
Aplicabilidad: Funciones y *buffers* que realizan el proceso de descodificación macrobloque a macrobloque.

[88] Este es el tamaño del código después de todas las optimizaciones presentadas en los apartados 4.2.2.2 y 4.2.2.3.

4.2.2.1.2 Ubicación del código en los diferentes niveles de memoria

Para maximizar el rendimiento de la memoria caché de programa es necesario minimizar el número de fallos en el acceso a la misma. Por ello, la adecuada ubicación del código en los diferentes niveles de la jerarquía de memoria del DSP resulta relevante.

La Figura 4-6 ilustra a través de un ejemplo cómo se ha optimizado el uso de estas memorias para el descodificador MPEG-2 empleando el procesador TMS320DM642 (16 kB de caché de programa de nivel 1) en un caso en el que se tienen dos funciones A y B que implementan parte de un bloque funcional del descodificador y que habitualmente se ejecutan secuencialmente.

Para mejorar su rendimiento ambas funciones se han ubicado en un fragmento de la memoria interna de propósito general que tiene un tamaño igual al de la caché de programa de nivel 1 (Bloque 1). De este modo, cuando se produce un fallo en el acceso a cualquiera de ellas se carga en la caché el segmento del programa que contiene ambas funciones.

Además, los fragmentos en los que se ha dividido el código se han ubicado en direcciones de memoria interna que son múltiplos del tamaño de la caché (alineados al tamaño de la caché) para que de este modo puedan transferirse más rápidamente desde la memoria de nivel superior ya que dichas transferencias se hacen en bloques de tamaño múltiplo al tamaño de las líneas de caché.

Las funciones que se encuentran ubicadas en la memoria externa del procesador y deben ser transferidas a la caché de programa de nivel 2, también se agrupan en fragmentos de tamaño igual al de la caché de nivel 1 y se alinean a direcciones múltiplo de dicho tamaño.

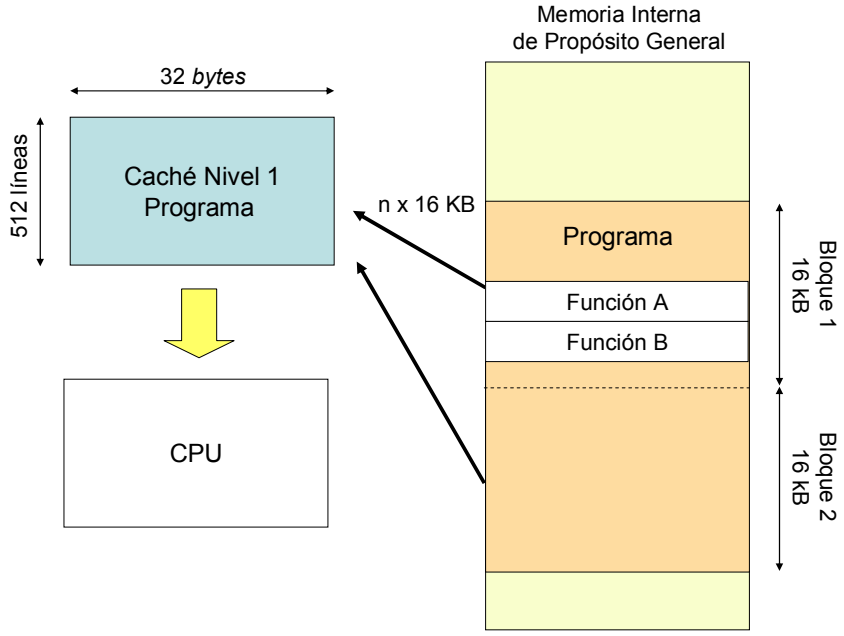


Figura 4-6. Ejemplo de organización de dos funciones en memoria interna para minimizar el número de fallos en los accesos de la CPU a la caché de nivel 1 de programa.

La tarea de agrupar adecuadamente las funciones y ubicarlas en fragmentos de memoria de tamaño igual al de la caché de nivel 1 de programa resulta bastante costosa. Además, debido al proceso de optimización, se producen algunos cambios estructurales en el código que hacen cambiar su tamaño, lo que obliga a reajustar la distribución del mismo en la memoria. Como conclusión se puede extraer que el

esfuerzo necesario para optimizar la ubicación del código en memoria debe distribuirse a lo largo de todo el proceso de optimización, no siendo necesario invertir demasiado tiempo en las fases tempranas, ni intermedias del proceso. Por este motivo no es posible hablar de la influencia individual que tiene la aplicación de esta técnica sin tener en cuenta el resto del proceso de optimización[89].

Referencia: Organización del código en memoria [ficha 003].
Descripción: Generar secciones de código de tamaño múltiplo del tamaño de la caché de programa de nivel 1. Agrupar en cada una de estas secciones aquellas funciones que se ejecuten de forma secuencial. Evaluar la influencia que tienen en el rendimiento del descodificador los diversos fragmentos del código que se generan para determinar si se alojan en memoria interna o externa. Ubicar cada sección a partir de direcciones de memoria que sean múltiplo del tamaño de dicha caché. Durante el proceso de optimización no debe aplicarse un gran esfuerzo a organizar el código en la memoria de programa puesto que, según se vayan realizando nuevas optimizaciones, el tamaño de las funciones va cambiando significativamente lo que obliga a tener que modificar continuamente el reparto.
Aplicabilidad: Aplicable a todos descodificadores que se ejecuten en procesadores en los que existan niveles de memoria caché de programa.

Para el descodificador MPEG-2, y tras parte del proceso de optimización que se describe en los apartados 4.2.2.2 y 4.2.2.3, se comprobó que el núcleo de procesamiento (VLD, IDCT y compensación de movimiento) tenía un tamaño muy próximo al de la caché de programa de nivel 1, estando además el rendimiento del descodificador por encima de los requisitos de funcionamiento en tiempo real. En esta situación se optó por optimizar algunas funciones para reducir su tamaño a pesar de que, a priori, se reducía su rendimiento ya que la mayor parte de las técnicas que emplean los compiladores para optimizar la velocidad de ejecución del código implican un incremento en el tamaño del mismo.

Como resultado final se obtuvo que el núcleo de procesamiento requiere 15.98 kB, por lo que pudo alojarse íntegramente en la caché de programa de nivel 1 minimizando los fallos de acceso a esta memoria. La inclusión de todo el código en memoria caché de nivel 1 compensó holgadamente la pérdida de rendimiento de las funciones que fueron optimizadas para reducir su tamaño.

La conclusión que se extrae de esta experiencia es que en las primeras fases del proceso de optimización no es posible tomar una decisión respecto a si es interesante reducir el tamaño del código sacrificando en parte el rendimiento del descodificador. Esta decisión debe tomarse una vez que se ha realizado parte del proceso de optimización valorando si el tamaño del código es cercano al de la memoria caché y por tanto es susceptible de reducirse para alojarlo íntegramente en dicha memoria. Si se logra el objetivo es probable que la pérdida de rendimiento que tienen las funciones que se optimizan para reducir su tamaño se vea compensado por la mejora global que se obtiene en el descodificador.

[89] Los resultados globales de aplicar este proceso junto al resto de optimizaciones se presentan en el apartado 4.2.4.

Referencia: Reducción del tamaño del código [ficha 004].
Descripción: Reducir el tamaño de las funciones que componen el bucle de descodificación de los macrobloques. Agrupar todas las funciones en una sección y comprobar si todas ellas pueden alojarse en la memoria caché de programa de nivel 1. Ubicar la sección en memoria interna de propósito general alineada al tamaño de la caché de nivel 1.
Aplicabilidad: Aplicable a todos los descodificadores que se ejecuten en procesadores en los que existan niveles de memoria caché de programa.

4.2.2.1.3 Ubicación de los datos en los diferentes niveles de memoria

En la memoria caché de datos de nivel 1 el problema es similar al presentado anteriormente para la memoria de programa, con la diferencia de que ahora los fallos en el acceso provocan que sea necesario realizar una actualización de los datos en la memoria de orden superior con la pérdida de tiempo que este proceso lleva asociado.

La optimización del uso de la memoria caché de datos del TMS320DM642 (caché de dos vías de 8 kB cada una) que se ha realizado en MPEG-2 se ejemplifica en la Figura 4-7. En dicha figura se muestra la ubicación en memoria interna de propósito general de dos *buffers* denominados A y B. Los criterios que se han empleado para alojar ambos *buffers* en la memoria son similares a los empleados para la caché de código (ver apartado 4.2.2.1.2); así, ambos *buffers* se definen con un tamaño múltiplo del tamaño de una línea de la cache de datos de nivel 1 y alineados a direcciones de memoria que también sean múltiplos de dicho tamaño. Además, ambos *buffers* se integran en una sección que se ubica en direcciones múltiplo del tamaño de las vías de la caché de datos de nivel 1.

Generalizando lo explicado anteriormente para un conjunto amplio de *buffers* se puede indicar, en base a los experimentos realizados, que la mejora del rendimiento del descodificador más significativa se logra haciendo que, si es posible, los *buffers* no superen el tamaño de una de las vías de la memoria caché de nivel 1 de datos, que se encuentren alineados en memoria a la anchura de línea de caché (64 *bytes* para el TMS320DM642) y que tengan un tamaño múltiplo del tamaño de la línea de dicha memoria (64 *bytes*). Además conviene que los *buffers* que se utilicen de forma más o menos consecutiva se ubiquen en zonas de memoria lo más próximas posible.

La misma estrategia que se ha empleado para distribuir los *buffers* en la memoria interna de propósito general se ha aplicado para ubicar los *buffers* de la memoria externa que, posteriormente, se alojarán en la caché de nivel 2. Así, tanto las imágenes de referencia como la trama de bits de entrada que emplea el descodificador MPEG-2 se han alineado al tamaño de una línea de la caché de nivel 2.

El esfuerzo empleado para ubicar los *buffers* del descodificador MPEG-2 en las diferentes secciones de memoria se distribuyó a lo largo del proceso de optimización puesto que los cambios que se fueron realizando afectaron de forma significativa a la organización de los datos en la memoria interna. Una vez que se dispuso de una versión final, se realizó un ajuste fino ubicando todos los *buffers* que emplea el descodificador en una única sección de la memoria interna de propósito general, con la excepción de los *buffers* empleados para almacenar las imágenes descodificadas y la trama de entrada que por su tamaño tienen que ser alojados en la memoria externa.

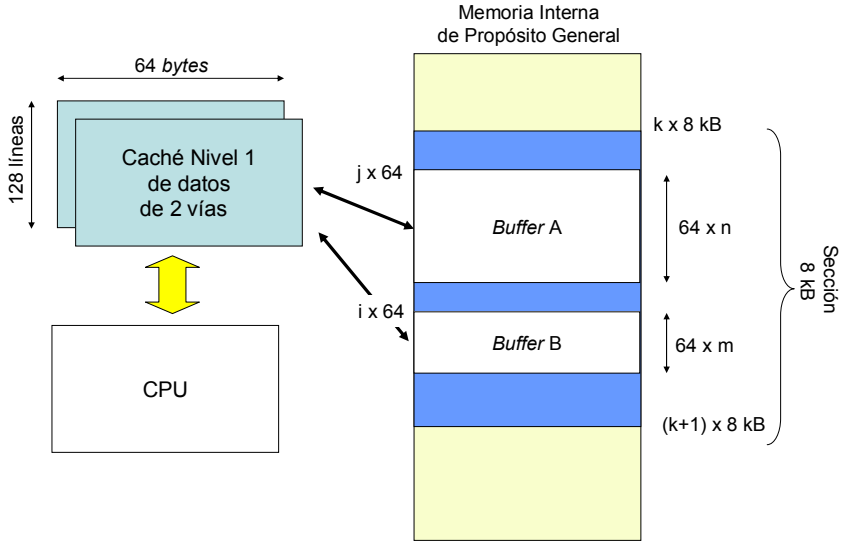


Figura 4-7. Organización de los *buffers* en memoria interna para optimizar el uso de la memoria caché de datos de nivel 1.

Referencia: Organización de los datos en memoria [ficha 005].
Descripción: Definir los *buffers* de datos con un tamaño múltiplo del tamaño de la línea de la caché de datos de nivel 1 y alinear su dirección de comienzo a una posición de memoria que también sea múltiplo de dicho tamaño. Agrupar los *buffers* que sean empleados por alguno de los módulos funcionales del descodificador en una sección de memoria y alinear la sección a una posición de memoria que sea múltiplo del tamaño total de una vía de la caché de datos de nivel 1. Los *buffers* que por su tamaño no puedan ubicarse en la memoria interna se alinearán al tamaño de la caché de nivel 2 en la memoria externa para facilitar el acceso a los mismos. El esfuerzo aplicado a distribuir los *buffers* de datos en la memoria debe realizarse de forma gradual a lo largo de todo el proceso de optimización puesto que los cambios en el código a menudo influyen, tanto en el tamaño de los datos que hay que manejar, como en su reparto en la memoria. Sólo cuando se finaliza el proceso de optimización se debe realizar un ajuste fino de la ubicación de los datos en las diferentes secciones de memoria.
Aplicabilidad: Aplicable a todos los descodificadores que se ejecuten en procesadores en los que existan niveles de memoria caché de datos.

4.2.2.2 Controlador de Acceso Directo a Memoria

El acceso al programa y los datos que se encuentran en memoria externa provoca una reducción significativa del rendimiento del descodificador. En el caso de los datos que maneja el descodificador MPEG-2, hay *buffers* que, por su tamaño, no pueden ubicarse en la memoria interna. Si la CPU accede directamente a estos datos, el número de fallos en el acceso a las memorias cachés de datos (nivel 1 y nivel 2) se incrementa sustancialmente con la correspondiente pérdida de prestaciones del descodificador.

El controlador de DMA permite reducir el número de fallos paralelizando la ejecución del algoritmo por parte de la CPU con el movimiento de datos entre la memoria interna y la externa. Para utilizar este controlador con el descodificador MPEG-2 se definen una serie de *buffers* intermedios alojados en la memoria interna a los que se transfieren los datos que deben procesarse haciendo uso de transferencias explícitas de DMA[90]; sobre ellos se realiza la descodificación y, finalmente, se envía el resultado a la memoria externa empleando de nuevo el controlador de DMA. De este modo mientras el controlador transfiere los datos, la CPU continúa ejecutando instrucciones de alguno de los bloques funcionales del descodificador.

A continuación se presentan las estrategias de optimización empleadas sobre el descodificador MPEG-2 en relación con el controlador de DMA. Aunque se han realizado de manera conjunta, se presentan de manera individual para una mejor sistematización.

4.2.2.2.1 Uso básico de las transferencias explícitas de DMA

Como se indicó anteriormente, los *buffers* que emplea el descodificador MPEG-2 para alojar las imágenes reconstruidas deben estar situados en la memoria externa debido a su tamaño. Esta situación penaliza el proceso de compensación de movimiento puesto que en él es preciso acceder de forma intensiva a memoria externa para obtener los macrobloques de referencia.

La Figura 4-8 muestra las etapas por las que pasa la reconstrucción de un macrobloque perteneciente a una imagen con formato de cuadro, un vector de movimiento con resolución de ½ píxel y una única referencia. La CPU debe acceder al *buffer* que almacena la trama de bits codificada para obtener el vector de movimiento y los coeficientes de la DCT, que se almacenan en el *buffer* denominado bloque. Seguidamente, se realiza la IDCT almacenando sus resultados en ese mismo *buffer*. Posteriormente, se obtienen los píxeles del macrobloque de referencia interpolando los de los macrobloques a los que apunta el vector de movimiento. El resultado de la interpolación se suma[91] con el error de predicción, y se guarda directamente sobre la imagen reconstruida.

Analizando el proceso de reconstrucción de un macrobloque se aprecia que la CPU debe acceder a la memoria externa en tres etapas (ver Figura 4-8):

- Etapa 1: Lectura de la trama de bits de entrada. El tamaño de la imagen codificada no es constante por lo que no es interesante realizar transferencias por DMA de estos datos a memoria interna[92]. Además, la imagen codificada

[90] Todos los accesos que el TMS320DM642 hace a la memoria externa se realizan mediante el controlador de DMA puesto que éste es el único mecanismo que permite acceder a dicha memoria (ver apartado 3.2.1). Sin embargo, estas transferencias pueden realizarse de forma automática cuando la CPU accede a datos ubicados en dicha memoria, o explícitamente cuando son transferencias ordenadas por el usuario mediante la programación de los registros del controlador de DMA. En el primer caso la CPU debe parar la ejecución del programa hasta que los datos estén disponibles; sin embargo, en el segundo caso, las transferencias pueden ir realizándose mientras la CPU continúa ejecutando otra parte del programa. Además, en el segundo caso es posible realizar el movimiento de bloques de datos con el consiguiente ahorro en el tiempo de la transferencia. En adelante, cuando se haga referencia a las transferencias de DMA, se tratará en todos los casos de transferencias explícitas.

[91] Para realizar esta suma no se emplea ningún *buffer* intermedio para almacenar el resultado.

[92] Cuando el descodificador se integra con el resto de elementos del sistema completo (ver apartado 4.2.4.2), la trama de entrada está alojada directamente en un *buffer* ubicado en memoria externa (ver apartado 8.1.1 del Anexo B). Dado que el tamaño de la imagen codificada no es constante, no es posible definir un *buffer* en memoria interna al que transferirla para acelerar el procesamiento de la trama.

suele tener un tamaño elevado por lo que es posible que no haya espacio disponible en la memoria interna para almacenarla.

- Etapa 2: Lectura de los píxeles necesarios para obtener el macrobloque de referencia. Cada uno de los macrobloques implicados tiene un tamaño fijo de 16×16 píxeles para luminancia y 8×8 para cada una de las crominancias. En este caso, resulta interesante emplear el controlador de DMA para transferir estos macrobloques a *buffers* en memoria interna.
- Etapa 3: Escritura de los píxeles del macrobloque reconstruido. Al igual que en el caso anterior, el tamaño del macrobloque reconstruido es constante lo que facilita su transferencia por DMA a la imagen descodificada alojada en memoria externa.

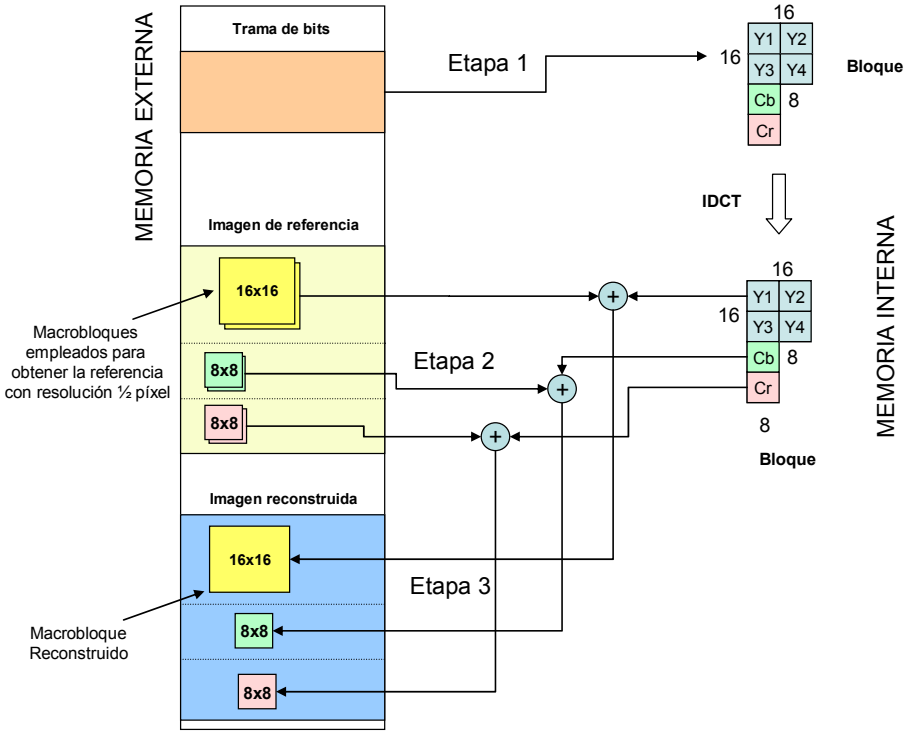


Figura 4-8. Proceso de obtención de un macrobloque reconstruido con predicción y una única referencia.

Por tanto el controlador de DMA se emplea para realizar la lectura de los píxeles necesarios para calcular el macrobloque de referencia (etapa 2) y la escritura de los píxeles del macrobloque reconstruido (etapa 3) mientras la CPU ejecuta otras partes del algoritmo.

Para poder emplear el controlador de DMA hay que realizar modificaciones en la estructura del descodificador ya que es necesario definir *buffers* intermedios del tamaño de un macrobloque, alojados en memoria interna en los que se almacenan temporalmente los datos que se transfieren. Todas las operaciones asociadas al proceso de reconstrucción de un macrobloque se realizan sobre estos *buffers* intermedios con la correspondiente mejora del rendimiento.

El uso del controlador de DMA obliga a modificar el bucle de descodificación para los macrobloques con predicción presentado en la Figura 4-4 para incluir en él las peticiones explícitas de transferencias de datos mediante el controlador de DMA. En el ordinograma modificado que se muestra en la Figura 4-9, se aprecia que, después de

extraer de la trama los vectores de movimiento, se configura el controlador de DMA para obtener los macrobloques necesarios. Seguidamente se calcula la predicción que se almacenará en los *buffers* intermedios ubicados en la memoria interna.

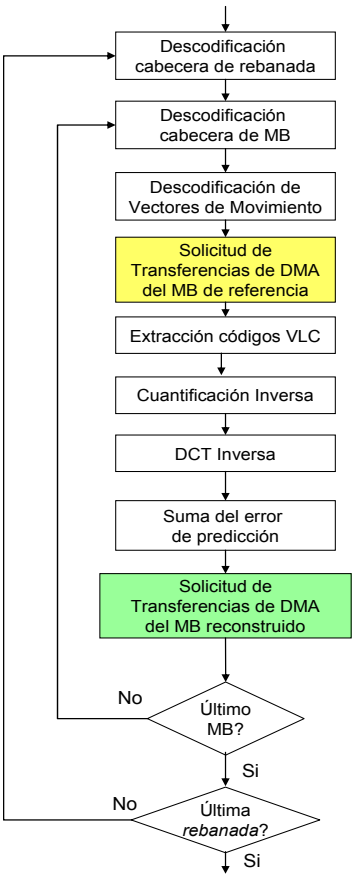


Figura 4-9. Ordinograma del bucle de descodificación de un macrobloque *inter* empleando transferencias explícitas de DMA.

La Figura 4-10 muestra los píxeles que son necesarios para el cálculo de píxeles intermedios en la compensación de movimiento cuando se tiene un vector de movimiento con resolución de ½ píxel. El código empleado como punto de partida realiza la interpolación leyendo directamente los píxeles con resolución entera desde la memoria externa. Así, para calcular el píxel 'a' son necesarias cuatro lecturas de memoria (píxeles 'A', 'B', 'D' y 'E'). Por tanto, para obtener todos los píxeles interpolados de un macrobloque asociado a un vector de movimiento que apunta al píxel 'a' es necesario leer de forma independiente todos los píxeles de los macrobloques enmarcados por las líneas azul y roja.

Por consiguiente, para disponer de todos los píxeles necesarios para realizar la interpolación del macrobloque de referencia es necesario configurar seis transferencias de DMA (dos para la luminancia y otras dos por cada componente de crominancia) que permiten obtener los dos macrobloques empleados para calcular los píxeles intermedios[93].

[93] La existencia de colas de petición de transferencias que posee el controlador de DMA del TMS320DM642 (ver apartado 3.2.1) posibilita lanzar varias transferencias de forma consecutiva de modo que el controlador las encola internamente y las va atendiendo secuencialmente dependiendo de la prioridad que se les haya asignado.

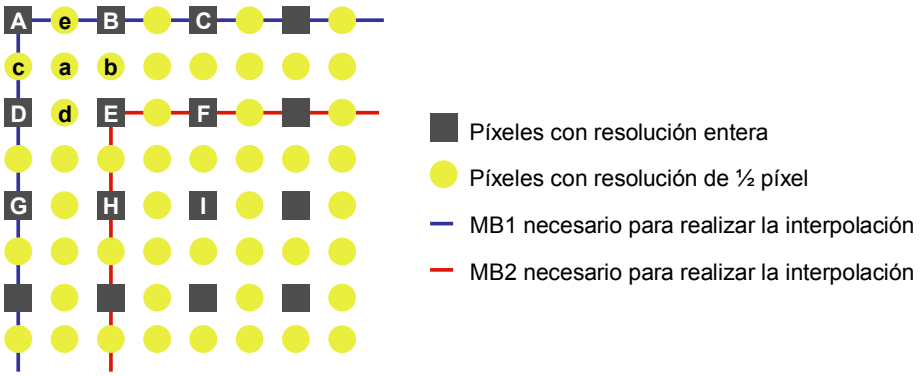


Figura 4-10. Cálculo de los píxeles interpolados a partir de los píxeles con resolución entera.

Mientras se realizan las transferencias a los *buffers* intermedios alojados en memoria interna, denominados MB_Referencia*i*, la CPU extrae los coeficientes de la DCT y calcula la IDCT. Una vez finalizadas las transferencias se realiza la interpolación de ½ píxel y la suma con el error de predicción, almacenando el resultado en un nuevo *buffer* intermedio ubicado en memoria interna y denominado MB_Reconstruido[94]. El macrobloque reconstruido es transferido a la imagen descodificada configurando de nuevo el controlador de DMA. Estas transferencias se realizan en paralelo con el comienzo del procesamiento del siguiente macrobloque (descodificación de cabecera de macrobloque y extracción, si es necesario, de los vectores de movimiento).

La Figura 4-11 muestra el movimiento de datos entre los *buffers* existentes para un macrobloque perteneciente a una imagen con formato de cuadro con una única referencia y un vector de movimiento con resolución de ½ píxel[95]. En ella se aprecia la existencia de *buffers* auxiliares de tamaño igual al de un macrobloque ubicados en memoria interna. Los *buffers* denominados MB_Referencia1 y MB_Referencia2 permiten alojar los píxeles de los dos macrobloques necesarios para realizar la interpolación de ½ píxel[96]; el otro *buffer* llamado MB_Reconstruido almacena los píxeles del macrobloque reconstruido. Como se puede observar son necesarias 9 transferencias entre la memoria interna y la externa[97] (marcadas en la Figura 4-11 desde DMA1 a DMA9).

---

[94] Para almacenar los resultados del macrobloque reconstruido se emplea un *buffer* (MB_Reconstruido) diferente que el empleado para guardar los resultados de la IDCT para poder paralelizar la transferencia de un macrobloque reconstruido con la extracción de los vectores de movimiento y la IDCT del macrobloque siguiente. La explicación de cómo se realiza esta paralelización se detalla posteriormente dentro de este mismo apartado de la memoria (ver apartado 4.2.2.3.4).

[95] El algoritmo que realiza la compensación de movimiento con resolución de ½ píxel requiere los píxeles de dos macrobloques por lo que es necesario acceder a todos los píxeles que los forman (ver apartado 4.2.2.2.2).

[96] En el caso de los macrobloques tipo B es necesario definir dos nuevos *buffers* intermedios (no incluidos en la Figura 4-11). Para diferenciar ambos *buffers,* a uno se le ha denominado MB_Referencia_backward*i* y al otro MB_Referencia_forward*i* (para i = 1 ó 2).

[97] En el peor de los casos se podrían llegar a tener 27 transferencias para un macrobloque. Este caso se corresponde con un macrobloque perteneciente a una imagen con formato de campo y con referencias hacia delante y hacia atrás que requeriría 24 transferencias (3 componentes × 2 vectores × 2 campos × 2 MB/interpolación) para obtener la referencia y 3 para almacenar el macrobloque reconstruido.

Dado que cada una de las líneas de un macrobloque se encuentra en una línea diferente de la imagen descodificada, los píxeles del macrobloque de referencia no están en posiciones de memoria consecutivas. Esto hecho implica que para pasar del último píxel de una línea del macrobloque al primero de la siguiente es necesario "saltar" el resto de los píxeles de una línea de la imagen. Por tanto, el controlador de DMA debe ser configurado para realizar transferencias de dos dimensiones a una dimensión[98] (2D-1D) para la obtención de los píxeles de referencia y de una dimensión a dos dimensiones (1D-2D) para la escritura del macrobloque reconstruido.

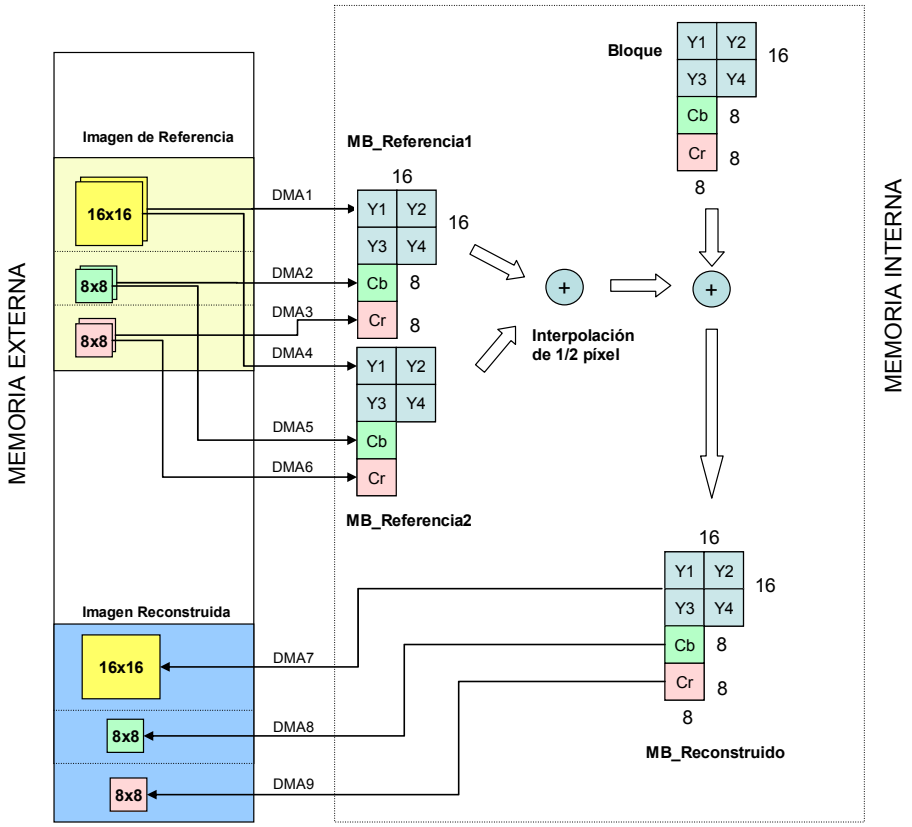


Figura 4-11. Movimiento de datos empleando transferencias explícitas de DMA para un macrobloque con una única referencia y un vector de movimiento con resolución de ½ píxel.

La secuencia de ejecución de las fases en las que se dividió el algoritmo de descodificación (ver Figura 4-5), se ve modificada al emplear el controlador de DMA para realizar el movimiento de datos entre la memoria interna y la externa. La Figura 4-12 muestra cómo las transferencias de datos (fases C y E) se paralelizan con la ejecución del algoritmo por parte de la CPU (fases A, B y D).

[98] Las imágenes son tratadas como un *buffer* bidimensional, y los *buffers* de memoria interna son utilizados como *buffers* unidimensionales de tamaño igual al número de píxeles de un macrobloque.

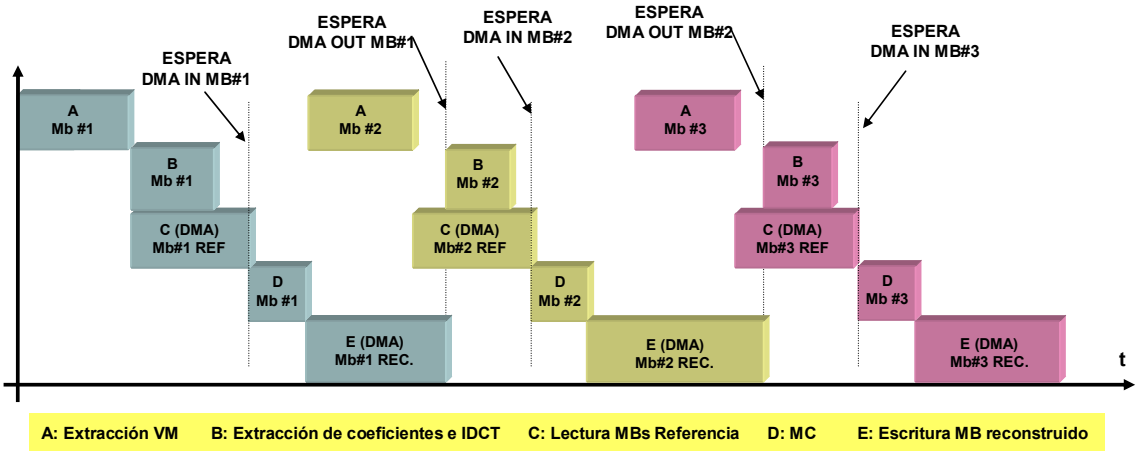


Figura 4-12. Diagrama temporal que muestra la paralelización del movimiento de datos y la descodificación de un macrobloque empleando el controlador de DMA.

Referencia: Paralelización de las transferencias de los macrobloques de referencia (fase C) con la IDCT (fase B) y del macrobloque reconstruido (fase E) con la extracción de los vectores de movimiento (fase A) [ficha 006].
Descripción: Definir en la memoria interna tantos *buffers* del tamaño de un macrobloque como sean necesarios para alojar todos los píxeles implicados en el proceso de reconstrucción de un macrobloque. Configurar el controlador de DMA inmediatamente después de obtener los vectores de movimiento, para transferir a los *buffers* intermedios los píxeles de las imágenes descodificadas anteriormente, mientras se ejecutan otras partes del algoritmo. Realizar todas las operaciones necesarias para descodificar cada macrobloque empleando exclusivamente estos *buffers* intermedios. Transferir el resultado a la imagen que se está descodificando en ese momento empleando de nuevo el controlador de DMA.
Aplicabilidad: Esta metodología puede aplicarse a cualquier otro descodificador de vídeo que procese las imágenes macrobloque a macrobloque en el que sea necesario el acceso frecuente a datos que se encuentren en memoria externa. En el caso del acceso a los píxeles de las imágenes de referencia es necesario que el controlador de DMA permita realizar transferencias bidimensionales puesto que los píxeles de la referencia no se encuentran en posiciones contiguas de memoria.

4.2.2.2.2 Reducción del número de transferencias de DMA

Para realizar la interpolación de ½ píxel son necesarios dos macrobloques que se solapan tal como muestra la Figura 4-13.

Se aprovecha dicho solape de forma que realizando una única transferencia de 17×17 *bytes* para luminancia y dos de 9×9 *bytes* para las crominancias, se dispone de todos los píxeles necesarios para realizar la interpolación reduciendo a la mitad el número de transferencias de DMA necesarias. Para realizar estas transferencias se

han declarado nuevos *buffers* (referencia_intermedia[99]) en memoria interna a los que transferir los bloques de 17×17 píxeles de luminancia y los dos de 9×9 píxeles de crominancia empleados para realizar la interpolación. El resultado de la interpolación se almacena en el *buffer* MB_referencia para posteriormente sumarle el error de predicción y guardarlo en el *buffer* MB_reconstruido. La Figura 4-14 muestra los *buffers* implicados en el proceso y el movimiento de datos realizado por parte del controlador de DMA para la obtención de un macrobloque reconstruido que posee una única referencia.

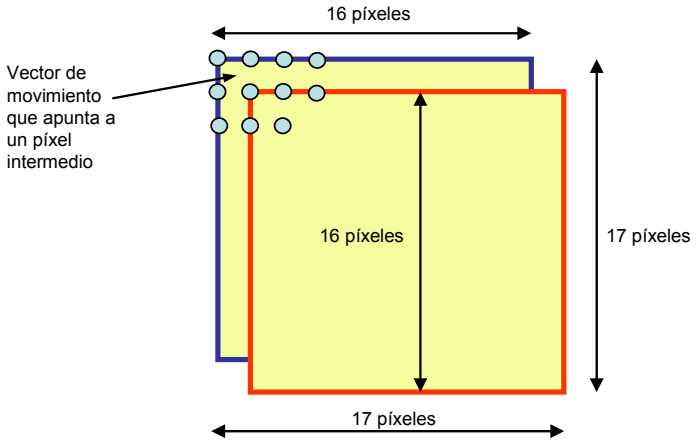


Figura 4-13. Macrobloques de luminancia empleados para interpolar los píxeles del macrobloque de referencia a partir de un vector de movimiento que apunta a un píxel intermedio.

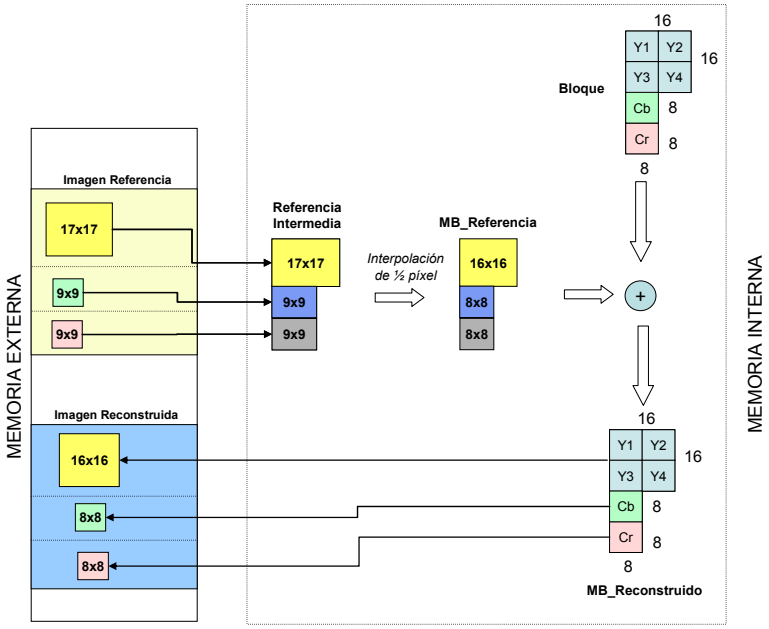


Figura 4-14. Compensación de movimiento con resolución de ½ píxel empleando el controlador de DMA para un macrobloque con una única referencia y un vector de movimiento con resolución de ½ píxel.

[99] En el caso de tratarse de macrobloques con predicción bidireccional es necesario definir dos nuevos *buffers* intermedios para transferir ambas referencias. A partir de dichas referencias se realiza la interpolación de los píxeles intermedios y el promedio para obtener el macrobloque de referencia.

Referencia: Reducción del número de transferencias de DMA necesarias para realizar la compensación de movimiento [ficha 007].
Descripción: Declarar *buffers* en memoria interna para alojar en ellos todos los píxeles que permitan realizar la compensación de movimiento con resolución fraccionaria. Transferir todos los píxeles necesarios para calcular la compensación de movimiento con los datos de estos *buffers*. Obtener el macrobloque de referencia, almacenarlo en otro *buffer* intermedio y sumarle el error de predicción. Una vez obtenido el macrobloque reconstruido se transfiere a la imagen descodificada empleando de nuevo el controlador de DMA.
Aplicabilidad: A todos los estándares de codificación que empleen predicción con resolución de ½ píxel.

4.2.2.2.3 Gestión de doble *buffer* o *buffer* ping-pong

Las transferencias de datos por parte del controlador de DMA no se realizan instantáneamente sino que, dependiendo de la ocupación de los buses y del número de transferencias acumuladas, pueden requerir un tiempo significativo. Por tanto, y como se aprecia en la Figura 4-12, la CPU debe esperar a que dichas transferencias finalicen en dos situaciones:

1. La compensación de movimiento (fase D) no puede comenzar hasta no disponer del macrobloque de referencia (fase C) puesto que los datos necesarios pueden no estar disponibles (instantes marcados en la Figura 4-12 como “Espera DMA IN”).

2. La extracción de los coeficientes de la DCT (fase B) de un macrobloque no puede comenzar hasta que no se haya finalizado la transferencia del macrobloque anterior (fase E) puesto que ambos procesos emplean el mismo *buffer* (instantes marcados como “Espera DMA OUT”).

Para eliminar la espera en el segundo caso se ha utilizado una técnica de doble *buffer* que permite que, mientras la CPU está procesando los datos que hay en uno de los *buffers*, el controlador de DMA pueda transferir datos a/desde el otro. En este caso, mientras se está realizando la transferencia de la fase E sobre un *buffer* (ping), se ejecuta la fase B del macrobloque siguiente sobre otro *buffer* (pong) idéntico. Los *buffers* se van intercambiando para que ambos procesos no accedan simultáneamente al mismo. Por tanto, los *buffers* implicados en este proceso son los mismos que los presentados en la Figura 4-14 a excepción del *buffer* MB_reconstruido que pasa a ser un doble *buffer* por lo que su tamaño se duplica.

La secuencia temporal de ejecución del algoritmo de descodificación dividido en las fases mostradas en la Figura 4-5 y el movimiento de datos entre memoria externa y memoria interna se muestra en la Figura 4-15. En este diagrama se aprecia que la transferencia para almacenar un macrobloque descodificado alojado en uno de los dos *buffers* ping-pong (fase E) se paraleliza con la extracción de los coeficientes de la DCT del siguiente macrobloque (fase B) y las transferencias por DMA para obtener sus referencias (fase C) que se realizan empleando el otro *buffer* ping-pong. Al finalizar el proceso de compensación (fase D), se intercambia la funcionalidad de los *buffers* ping-pong para el siguiente macrobloque a descodificar.

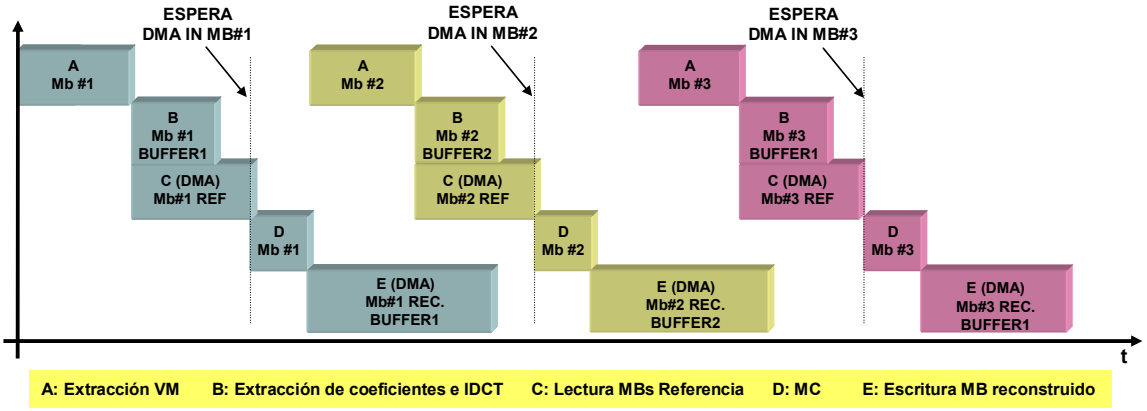


Figura 4-15. Diagrama temporal de la descodificación de un macrobloque empleando la técnica de doble *buffer* para la transferencia del macrobloque reconstruido a la imagen que se está descodificando.

Con esta técnica se eliminan las esperas por parte de la CPU en las transferencias de los macrobloques reconstruidos.

Referencia: Eliminación de esperas de la CPU para paralelizar la transferencia de un macrobloque reconstruido (fase E) con la extracción de los coeficientes de la DCT del siguiente macrobloque (fase B) [ficha 008].
Descripción: Comprobar si la CPU debe esperar a que finalicen las transferencias por DMA. En caso de que así sea, duplicar el tamaño de los *buffers* definidos en memoria interna de modo que el controlador de DMA realice transferencias a/desde uno de ellos mientras que la CPU procesa y almacena datos del siguiente macrobloque en el otro. Tras finalizar el procesamiento de un macrobloque se intercambian los *buffers* para el procesamiento del siguiente.
Aplicabilidad: A aquellos descodificadores en los que la CPU deba esperar a que finalice el movimiento de datos entre memoria interna y externa cuando se realicen transferencias explícitas por parte del controlador de DMA.

4.2.2.2.4 Modificación de la estructura del descodificador

Para eliminar las esperas asociadas a las transferencias de los macrobloques de referencia (“Espera DMA IN” de la Figura 4-15) es necesario realizar algunas modificaciones en el bucle de descodificación de macrobloques. El bucle principal de descodificación trabaja a nivel de macrobloque (ver Figura 4-4) lo que implica que hasta que no se finaliza la descodificación de uno de ellos, no puede comenzar la del siguiente. Este hecho provoca que la CPU no pueda realizar la compensación de movimiento (fase D) hasta que el controlador de DMA haya transferido los píxeles de la/s referencia/s (fase C).

Esta espera puede eliminarse reorganizando el bucle de descodificación. La Figura 4-16 muestra el nuevo bucle de descodificación en el que mientras se espera la finalización de las transferencias de los macrobloques de referencia (MB#X-1), se comienza con la descodificación del siguiente macrobloque (MB#X). Por tanto, en cada iteración del bucle se manejan datos correspondientes a dos macrobloques (MB#X-1 y MB#X). Debido a la reestructuración del bucle es necesario diferenciar el procesamiento del primer y del último macrobloque de cada rebanada tal y como se aprecia en la Figura 4-16.

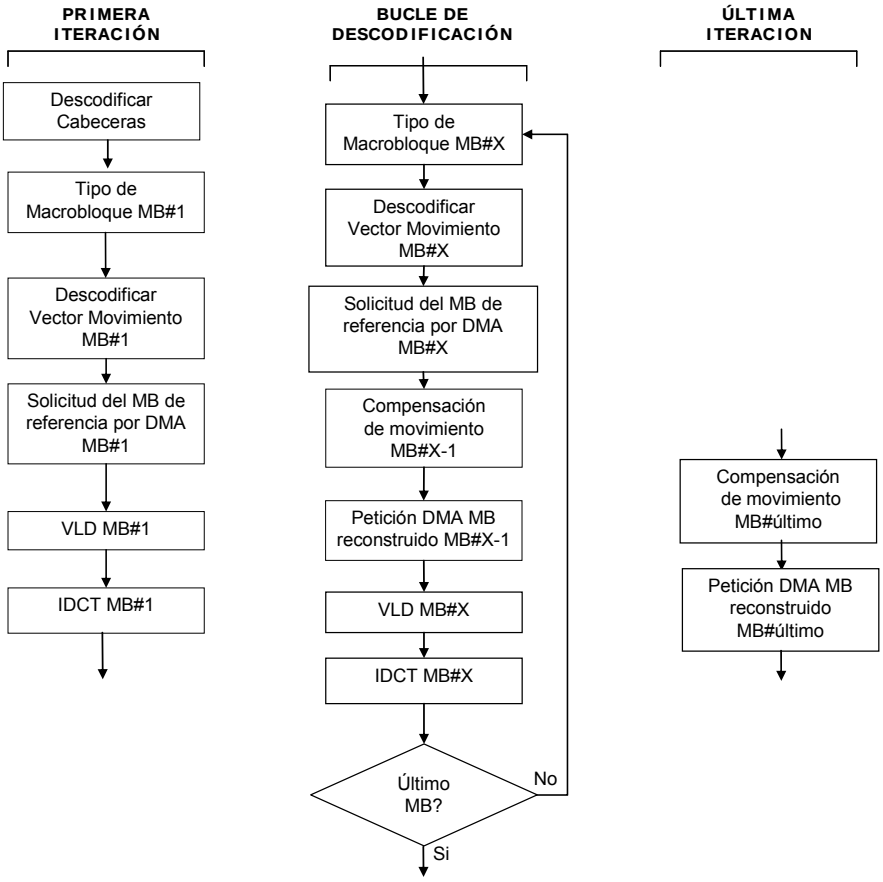


Figura 4-16. Ordinograma del bucle de descodificación de cada macrobloque que elimina las esperas en las transferencias de DMA.

Como consecuencia de la reorganización del bucle de descodificación, desde que la CPU dispone del vector de movimiento del MB#X (fase B), hasta que calcula la compensación de movimiento de dicho macrobloque (fase D), se intercalan la compensación de movimiento del macrobloque anterior MB#X-1 (fase D), el cálculo de la IDCT y VLD del macrobloque actual MB#X (fase B) y la extracción del vector de movimiento del macrobloque siguiente MB#X+1 (fase A).

El procesamiento de dos macrobloques dentro del bucle principal de descodificación requiere que algunos de los *buffers* intermedios alojados en memoria interna tengan que ser duplicados puesto que es necesario almacenar simultáneamente datos pertenecientes a dos macrobloques. Sin embargo, como estos *buffers* tienen el tamaño de un macrobloque, no implica un incremento importante de la memoria requerida.

El diagrama temporal que representa la ejecución del algoritmo por parte de la CPU y las transferencias de DMA se muestra en la Figura 4-17. En él se aprecia que la CPU está ejecutando instrucciones continuamente sin necesidad de realizar esperas puesto que mientras que el controlador de DMA realiza las transferencias para obtener las referencias del macrobloque actual MB#X (fase C), la CPU comienza a procesar el siguiente macrobloque MB#X+1 (fase A).

En las pruebas realizadas para las secuencias Arte y GI_9 se ha comprobado que, a pesar de retrasar el comienzo de la transferencia (fase C), la carga de procesamiento introducida (fases A y D) antes de necesitar esos datos es suficiente para que la CPU nunca realice esperas.

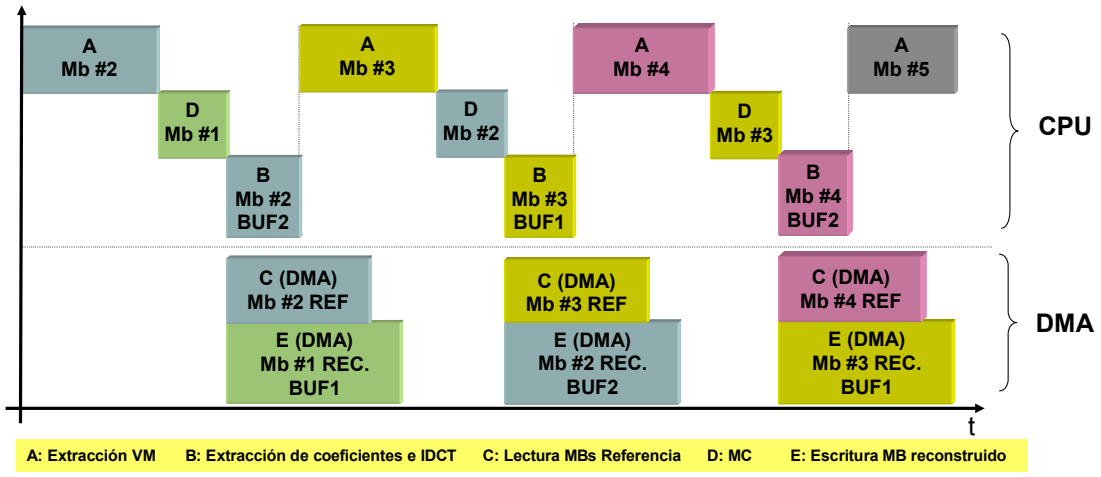


Figura 4-17. Diagrama temporal de la ejecución del bucle de descodificación modificado.

Referencia: Reorganización del bucle principal de descodificación [ficha 009].
Descripción: Rellenar los tiempos en los que el controlador está transfiriendo por DMA los píxeles necesarios para poder realizar la compensación de movimiento de un macrobloque con instrucciones que permitan que la CPU pueda continuar el procesamiento de otros datos. Dado que éstos no pueden pertenecer al mismo macrobloque, es necesario comenzar a descodificar el siguiente por lo que el bucle principal de descodificación se ve modificado para manejar simultáneamente datos de dos macrobloques diferentes. Aplicar esta técnica implica almacenar más *buffers* en memoria interna de propósito general lo puede conllevar tener que sacar de esta memoria otros *buffers* o funciones con la correspondiente pérdida de rendimiento. Por tanto, es necesario confirmar que la CPU realiza esperas de las transferencias de DMA antes de abordar la modificación del bucle de descodificación de macrobloques.
Aplicabilidad: Esta técnica, al igual que la [ficha 008], es aplicable para cualquier descodificador que se ejecute en un procesador en el que la CPU deba esperar la finalización de la transferencia por DMA de los píxeles necesarios para realizar la compensación de movimiento.

4.2.2.2.5 Alineamiento de las transferencias

El rendimiento de las transferencias de DMA en el procesador TMS320DM642 se incrementa notablemente si se realiza con *buffers* de tamaño múltiplo de 32 bits y con direcciones de memoria de origen y destino que sean también múltiplos de 32 bits (alineadas a 32 bits) ya que de este modo se aprovecha todo el ancho de banda de acceso a la memoria externa.

El descodificador MPEG-2 realiza dos tipos de transferencias: la de los macrobloques reconstruidos en memoria interna, hacia la imagen descodificada almacenada en memoria externa; y, la de los macrobloques de referencia de las imágenes previamente descodificadas, hacia los *buffers* intermedios ubicados en memoria interna.

En el primer tipo, deben transferirse 256 *bytes* para la luminancia y 64 *bytes* para cada crominancia; en ambos casos múltiplos de 32 bits. Además, las direcciones de origen y destino de las transferencias son múltiplo de 32 bits puesto que los *buffers*

origen (MB_reconstruido) y destino (Imagen_reconstruida) tienen su dirección de comienzo alineada[100] a este número de bits.

Sin embargo, para el segundo tipo de transferencias no es posible asegurar que la dirección origen del macrobloque de referencia esté alineada a 32 bits, puesto que el macrobloque de referencia puede estar ubicado en cualquier posición de las imágenes descodificadas anteriormente; además, tanto el tamaño de la transferencia para la luminancia (17×17) como el de las crominancias (9×9) no son múltiplos de 32 bits. Ambas circunstancias obligan a tener que realizar transferencias de datos de tamaño 8 bits con la correspondiente pérdida de ancho de banda en el acceso a la memoria externa.

Para solventar este problema se ha enmarcado el bloque de 17×17 píxeles de luminancia necesario para realizar la compensación de movimiento en otro de tamaño 20×17 píxeles[101] que cumple ambos requisitos. Este bloque de 20×17 incluye en todos los casos los píxeles que son necesarios, posee un tamaño múltiplo de 32 bits y puede alinearse también a 32 bits realizando una aritmética sencilla como la que se muestra en la Figura 4-18. En dicha figura se aprecia que a partir de un vector de movimiento que no apunta a un píxel entero (resolución de ½ píxel) se calcula el vector de movimiento con resolución entera[102] más próximo que sí esté alineado y se almacena el desplazamiento.

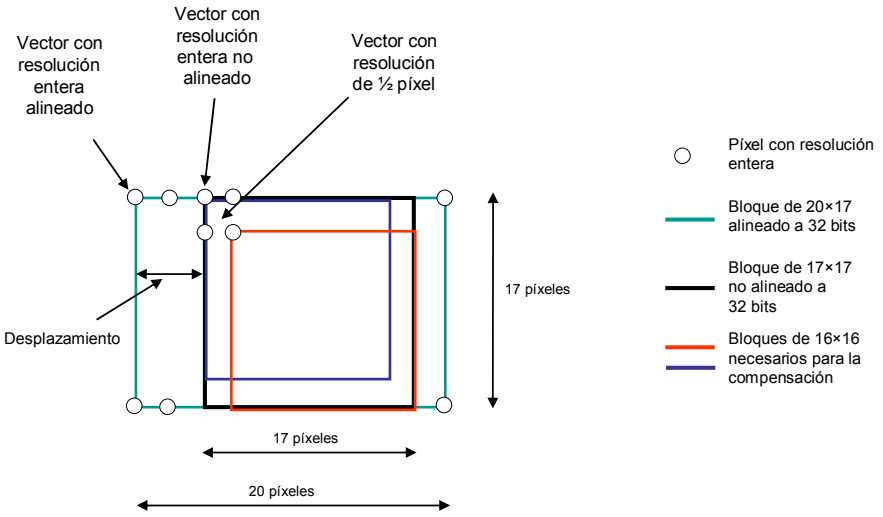


Figura 4-18. Cálculo de la dirección de memoria alineada para realizar las transferencias de bloques de 20×17 píxeles.

[100] El alineamiento de los *buffers* a direcciones de memoria es configurable dentro del código con algunas directivas del precompilador. En este caso, y para asegurar los alineamientos, se ha forzado a que las direcciones de origen de todos los *buffers* implicados en la descodificación sean múltiplos de 32 bits. Dado que los saltos que se realizan en las direcciones de destino de las transferencias son siempre múltiplos de 16 *bytes* (se trata siempre de saltos de tamaño macrobloque -16 *bytes*-), y que la anchura de la imagen también es múltiplo de 32 bits, se puede asegurar que la dirección de destino siempre está alineada a 32 bits. Por otro lado los saltos que se producen dentro del *buffer* que almacena el macrobloque reconstruido siempre son de tamaño 16×16 (luminancias) u 8×8 (crominancias) siendo en ambos casos múltiplos de 32 bits

[101] En el caso de las crominancias el procedimiento es el mismo salvo que los bloques de 9×9 píxeles se enmarcan en otros de tamaño 12×9 píxeles.

[102] Para obtener este vector de movimiento basta con truncar el bit de menor peso de las dos componentes del vector de movimiento con resolución de ½ píxel.

Una vez obtenida la dirección alineada se realiza la transferencia de tamaño 20×17, se calculan los píxeles interpolados y se emplea el desplazamiento para obtener los píxeles que realmente componen el macrobloque de referencia.

Referencia: Alineamiento de todas las transferencias de DMA empleadas en la compensación de movimiento [ficha 010].
Descripción: Forzar que todas las transferencias tengan un tamaño múltiplo de 32 bits y que sus direcciones de origen y destino también sean múltiplos de 32 bits. Si alguna transferencia no cumple estos requisitos, el bloque a transferir debe enmarcarse en otro de mayor tamaño que sí esté alineado y gestionar el desplazamiento del macrobloque de referencia respecto al bloque de píxeles transferidos. Al transferir más píxeles de los necesarios y tener que gestionar el desplazamiento de los datos útiles respecto a los transferidos es necesario evaluar si realmente se produce una mejora en el rendimiento global del descodificador
Aplicabilidad: A todos los procesadores en los que la eficiencia de las transferencias de DMA se incremente cuando el tamaño de los datos a transferir y las direcciones de origen y destino requieran algún requisito específico respecto a su tamaño y/o alineamiento en memoria.

4.2.2.2.6 Distribución de las transferencias entre las colas de petición del controlador de DMA

El controlador de DMA del procesador TMS320DM642 posee cuatro colas FIFO de solicitud de transferencias de DMA en las que se acumulan las peticiones procedentes de la CPU para acceder a la memoria externa debido a fallos en las memorias caché, las configuradas explícitamente por parte del usuario y las procedentes de los periféricos (ver apartado 3.2.1). Las solicitudes se deben distribuir entre todas las colas de petición de transferencias de forma homogénea puesto que si alguna de ellas se llena, la CPU realiza una parada en la ejecución (*stall*) hasta que ésta se vacía con la correspondiente pérdida de tiempo.

La cola más prioritaria (urgente) es empleada por parte de la CPU para transferir datos a sus memorias caché por lo que no debe ser empleada por el usuario. Las otras tres pueden ser empleadas por las aplicaciones de usuario y los periféricos.

Para la optimización del tiempo de ejecución del descodificador MPEG-2 se ha realizado un reparto en el que todas las colas tienen un mismo número de peticiones puesto que los píxeles de luminancia se solicitan por la cola de prioridad alta, los de crominancia del azul por la de prioridad media y los del rojo por la de prioridad baja.

Referencia: Gestión de las colas de petición de transferencias del controlador de DMA [ficha 011].
Descripción: Analizar el número de peticiones de transferencias que realiza el descodificador. Repartir homogéneamente entre las colas de petición del controlador de DMA las transferencias que se soliciten para evitar la sobrecarga de alguna.
Aplicabilidad: La técnica propuesta debe aplicarse a procesadores en los que existan colas de petición de transferencias que nunca deben desbordarse para evitar paradas (*stalls*) en el *pipeline* de la CPU.

4.2.2.2.7 Procesado por tiras de macrobloques

La eficiencia en las transferencias que ofrece el controlador de DMA del TMS320DM642 aumenta si se configuran pocas peticiones de elevado tamaño frente a la opción de realizar muchas de tamaño menor. Esta característica permite optimizar el rendimiento del descodificador MPEG-2 en el proceso de transferencia de los macrobloques reconstruidos a la memoria externa (fase D) puesto que en lugar de realizar una transferencia para cada macrobloque, éstos se almacenan en un *buffer* temporal alojado en memoria interna para, posteriormente, realizar una única transferencia con varios de ellos.

Esta estrategia aporta una segunda ventaja si en este *buffer* se almacenan todos los macrobloques de una tira de la imagen. En este caso se pasa de realizar transferencias 1D-2D (ver apartado 4.2.2.2.1) a transferencias unidimensionales (1D-1D) que son gestionadas de un modo más eficiente por parte del controlador de DMA.

La Figura 4-19 ilustra el proceso seguido para almacenar en la imagen reconstruida una tira completa de macrobloques. Cada macrobloque descodificado es copiado sobre la posición correspondiente del *buffer* temporal denominado Tira_reconstruida en lugar de usar el *buffer* MB_reconstruido. Una vez que se han copiado todos los macrobloques de una tira, se programa una única transferencia de DMA. Por tanto, y en relación con el diagrama temporal presentado en la Figura 4-17, se elimina la etapa E que pasa a realizarse una vez por tira de macrobloques en lugar de una vez por macrobloque.

Para poder paralelizar esta única transferencia con el procesamiento de los primeros macrobloques de la siguiente tira, el *buffer* Tira_reconstruida se ha definido como un doble *buffer* o *buffer* ping-pong. El empleo de esta técnica supone definir un doble *buffer* en memoria interna de propósito general de 33.75 kB[103] para una imagen en formato PAL.

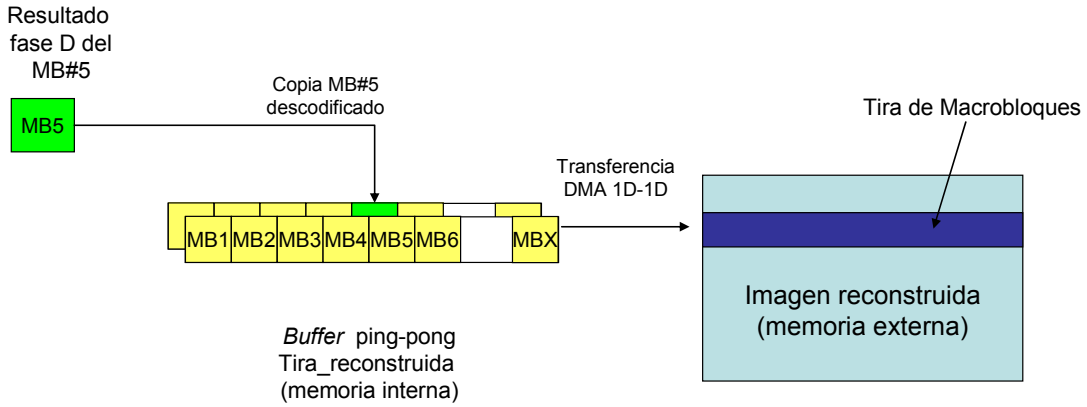


Figura 4-19. Transferencia de la imagen reconstruida por tiras completas de macrobloques.

[103] El tamaño del *buffer* viene dado por 720 píxeles/tira × 16 píxeles/macrobloque × (1 luminancia + 0.5 crominancias) = 17280 *bytes*. Este tamaño debe multiplicarse por 2 al tratarse de un doble *buffer* para obtener los 33.75 kB.

Referencia: Transferencias de macrobloques reconstruidos por tiras [ficha 012].
Descripción: Almacenar los macrobloques descodificados de la imagen en un doble *buffer* intermedio de modo que, cuando se dispone de todos los que componen una tira de la imagen, se realiza una única transferencia de DMA. En estos casos hay que tener en cuenta que su aplicación requiere definir un doble *buffer* en memoria interna de propósito general de elevado tamaño, lo que no siempre es posible. Si no hay memoria interna de propósito general disponible, es necesario trasladar algunas funciones/datos a memoria externa para poder alojar estos *buffers*, con la correspondiente pérdida de rendimiento. Valorar si la mejora que proporciona transferir mediante el controlador de DMA tiras de macrobloques compensa respecto a la pérdida de rendimiento que puede suponer tener que trasladar a memoria externa algunas funciones y/o datos.
Aplicabilidad: Esta técnica puede ser aplicada a los procesadores en los que la eficiencia de las transferencias de DMA se incrementa si éstas son de tamaño elevado.

4.2.2.3 Arquitectura SIMD

Analizando el número de ciclos que emplea el descodificador en cada uno de los módulos en los que se descompone, se han identificado algunas funciones cuya carga computacional es elevada. Para evaluar la carga computacional hay que analizar los datos de *profile* iniciales (con todo el código y los datos alojados en memoria externa) obtenidos tras la descodificación de varias secuencias. Las funciones candidatas a ser optimizadas son aquellas que emplean un mayor número de ciclos para ejecutarse sin tener en cuenta los fallos en las memorias cachés ya que al estar alojados los datos y las funciones en memoria externa, el número de fallos va a ser con toda probabilidad elevado. El problema de los fallos en acceso a la memoria caché se resolverá empleando por una parte el controlador de DMA (ver apartado 4.2.2.2) y alojando tanto las funciones como los datos en alguna de las memorias internas del procesador (ver apartado 4.2.2.1).

Las funciones con mayor carga computacional en el descodificador MPEG-2 son aquellas que realizan operaciones repetitivas sobre un conjunto de datos (píxeles, coeficientes de la DCT, etc.). Estas funciones pueden ser codificadas en lenguaje ensamblador para aprovechar mejor la arquitectura SIMD que posee el procesador. Se han identificado cuatro funcionalidades adecuadas para ser optimizadas empleando el lenguaje ensamblador: la DCT inversa, la VLD, la interpolación de píxeles de diferentes macrobloques[104] y la suma del error de predicción con el macrobloque de referencia.

Para las dos primeras funcionalidades (cálculo de la DCT inversa y de la VLD) se dispone de una librería optimizada de procesado de imágenes que proporciona el fabricante del DSP [TI02] en la que existen funciones idénticas a las que utiliza el descodificador. A pesar de ello, el proceso de sustitución de las funciones en C por las funciones optimizadas no es inmediato, puesto que es necesario realizar una adaptación de los parámetros de entrada y salida.

Para las otras dos funcionalidades (interpolación de píxeles de diferentes macrobloques y suma del error de predicción) ha sido necesario reescribir el código

[104] La interpolación de píxeles se emplea tanto en compensación de movimiento de ½ píxel como en el promediado de los macrobloque con predicción bidireccional (macrobloques tipo B).

correspondiente. Para optimizar estas funciones es imprescindible emplear las instrucciones del procesador que son capaces de operar con varios datos en paralelo. El compilador es capaz de emplear estas instrucciones automáticamente aunque no siempre genera un código suficientemente optimizado puesto que desconoce cómo están organizados los datos; por ello, es necesario recodificar estas funciones directamente en lenguaje ensamblador y paralelizar su ejecución manualmente[105].

4.2.2.3.1 IDCT

La función de librería que realiza la IDCT se denomina *IMG_idct_8x8*. Esta función permite calcular la IDCT de varios bloques de 8×8 píxeles conforme a la norma IEEE 1180-1990, incluyendo el redondeo y la saturación a 9 bits con signo.

Para ello debe recibir como argumentos una lista de bloques de 8×8 coeficientes y el número de bloques contenidos en esa lista. Además, el *buffer* que almacena los bloques de entrada debe estar alineado a una dirección de memoria múltiplo de 64 bits y el resultado se devuelve en posiciones consecutivas de memoria, a partir de la dirección de comienzo del *buffer* de entrada.

El número de ciclos de reloj que emplea esta función es de 92 × número de IDCTs + 62 lo que implicaría una mejora superior al 90% respecto a la versión escrita en C (la IDCT pasaría de 2300 a 160 ciclos de reloj por bloque de 8×8 píxeles).

Sin embargo, una vez efectuada la sustitución de la función, las medidas realizadas sobre el descodificador muestran que la mejora es menor de la esperada debido a que los datos de rendimiento que facilita el fabricante se corresponden a un caso óptimo en el que tanto los coeficientes de la DCT como el código se encuentran en memoria caché de nivel 1, cosa que no ocurre cuando esta función es invocada por el descodificador. Con estos condicionantes las medidas realizadas indican que el bloque que realiza la IDCT obtiene una mejora en torno al 60% al sustituir el código original por la función de librería.

Referencia: Empleo de funciones de librería [ficha 013].
Descripción: Analizar las librerías optimizadas que facilitan los fabricantes de los procesadores para tratar de localizar aquellas funciones del descodificador que pueden ser sustituidas por alguna de estas funciones optimizadas. Adaptar el código del descodificador para cumplir los requisitos que requieran las funciones de librería. Evaluar la mejora que supone en el rendimiento del descodificador ya que en algunas ocasiones puede que la inclusión de estas funciones no implique una mejora global.
Aplicabilidad: Esta técnica es aplicable de forma general.

[105] El entorno de desarrollo del DSP permite escribir funciones en ensamblador lineal sin asignar la ejecución de las instrucciones a las diferentes unidades funcionales del procesador de modo que es el compilador el que realiza esta tarea; sin embargo, el rendimiento del código que se obtiene es menor que el que se puede lograr paralelizando manualmente la ejecución de las instrucciones y la asignación de las mismas a las unidades funcionales. Por tanto, y dependiendo de la complejidad del algoritmo que se esté realizando, en algunos casos se ha optado por codificarlo en ensamblador lineal mientras que en otras ocasiones se ha realizado manualmente la paralelización.

4.2.2.3.2 VLD

Al igual que ocurre con la IDCT, existen dos funciones de librería que realizan la VLD, tanto para macrobloques codificados en modo *inter*, como para macrobloques codificados en modo *intra*. La función empleada para macrobloques *inter* requiere 10 × S + 37 × CB + 15 × NCB + 34 ciclos de reloj para su ejecución, siendo *S* el número de símbolos del macrobloque, *CB* el número de macrobloques codificados y *NCB* el número de macrobloques no codificados.

Por otra parte, la función empleada para macrobloques *intra* requiere 10 × (*S* -*CB*) + 55 × *CB* + 15 × *NCB* + 35 ciclos de reloj para su ejecución, donde *S*, *CB* y *NCB* tienen el mismo significado que en el caso anterior. En ambos casos es necesario que los datos de entrada se encuentren ubicados en direcciones de memoria alineadas a 64 bits lo que obliga a realizar algunas modificaciones en el código de partida para que se cumpla este requisito.

Realizadas las medidas de rendimiento con varias secuencias de prueba, se obtiene una mejora de este bloque, respecto a la versión codificada íntegramente en C, superior al 60%.

4.2.2.3.3 Interpolación de píxeles pertenecientes a macrobloques diferentes

Tal y como se indicó anteriormente, el descodificador MPEG-2 debe interpolar píxeles pertenecientes a diferentes macrobloques en dos etapas del proceso de descodificación: la compensación de movimiento de ½ píxel y la interpolación para obtener los píxeles de un macrobloque tipo B. Dado que ambos procesos son muy similares, en este apartado se detalla la técnica empleada para optimizar la interpolación de ½ píxel, siendo esta extrapolable al procesamiento de los macrobloques tipo B.

La compensación de movimiento con resolución de ½ píxel requiere el cálculo de los píxeles interpolados a partir de los píxeles enteros de la imagen de referencia. Para poder calcular estos píxeles intermedios es necesario disponer del bloque de 17×17 píxeles al que apunta el vector de movimiento con resolución entera asociado al macrobloque que se está descodificando (ver apartado 4.2.2.2.2).

La Figura 4-20 muestra el modo en que el código empleado como punto de partida realiza el cálculo de los píxeles interpolados para obtener el bloque de referencia cuando el vector de movimiento (MV_x y MV_y) apunta a un píxel intermedio tanto en la componente vertical como en la horizontal. Inicialmente, se obtiene el vector de movimiento con precisión entera (MV'_x y MV'_y) y se leen de la imagen de referencia los píxeles del macrobloque al que apunta dicho vector denominados B_{xy}, así como algunos píxeles extras pertenecientes a los macrobloques vecinos inferior, derecho y diagonal inferior derecho marcados como A_{1y}, C_{x1} y D_{11} respectivamente.

Para calcular cada uno de los píxeles interpolados se emplean cuatro píxeles con resolución entera; así, para el píxel R_{11} es necesario promediar los píxeles B_{11}, B_{12}, B_{21} y B_{22} tal y como se muestra en la parte derecha de la Figura 4-20. Este proceso se repite para los seis bloques de 8×8 píxeles que componen un macrobloque.

Analizando la carga computacional requerida para el cálculo de cada bloque de 8×8 píxeles, se obtiene que son necesarias 81 cargas, 256 operaciones de suma, 64 desplazamientos y 64 almacenamientos[106], todos ellos a nivel de *byte*[107].

[106] Las cargas se corresponden con la lectura de los 64 píxeles del bloque B, 8 píxeles de los bloques A y C y 1 píxel del bloque D. Las 256 operaciones de suma se corresponden con 4 sumas para cada píxel. Los 64 desplazamientos y los 64 almacenamientos son las operaciones que hay que realizar para cada píxel interpolado.

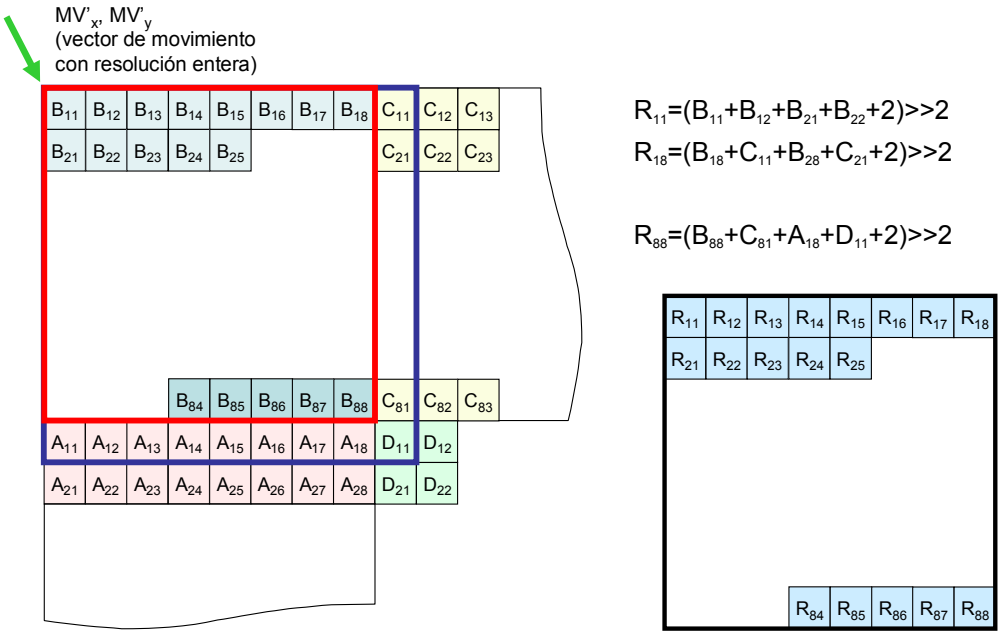


Figura 4-20. Cálculo de los píxeles intermedios del bloque reconstruido para un vector de movimiento con precisión de ½ píxel.

La arquitectura SIMD de 32 bits que posee el DSP[108] permite ejecutar instrucciones en las que los datos de 32 bits con los que se trabaja están formados por 4 datos de 8 bits ó 2 de 16 bits de modo que las unidades funciones realizan varias operaciones simultáneas con datos de menor tamaño. Además, el DSP dispone de instrucciones para la lectura y almacenamiento de 8 *bytes* que se encuentren contiguos en memoria (instrucciones LOAD y STORE) y para el cálculo de cuatro medias aritméticas con redondeo de los cuatro datos de 8 bits que se encuentran almacenados en dos registros de 32 bits (instrucción AVGU4).

La Figura 4-21 muestra cómo se realiza la compensación de movimiento de 8 píxeles interpolados (R_{11}-R_{18}) empleando estas instrucciones. El proceso se divide en la siguiente secuencia de operaciones:

1. Se obtienen los 18 píxeles (B_{11}-B_{18}, B_{21}-B_{28} y C_{11}-C_{12}) implicados con 4 instrucciones LOAD.
2. Se calcula la media entre los píxeles de la primera fila (P_{11}-P_{18}) de datos que aparecen en la figura con 2 instrucciones de promediado AVGU4.
3. Se repite el proceso anterior para los píxeles de la segunda fila (P_{21}-P_{28}).
4. Se promedian los resultados obtenidos tras las dos fases anteriores obteniendo los píxeles interpolados (R_{11}-R_{18}).
5. Se finaliza el proceso almacenando en memoria los datos calculados mediante una instrucción STORE.

[107] Se trata de un cálculo teórico en el que no se ha tenido en cuenta que el compilador es capaz de realizar algunas palalelizaciones entre los datos que permiten que se reduzca el número de ciclos de reloj que requiere su ejecución.

[108] A pesar de tratarse de una arquitectura de 32 bits, el repertorio de instrucciones que posee el procesador permite acceder en algunos casos a datos de 64 bits (por ejemplo operaciones de carga y almacenamiento).

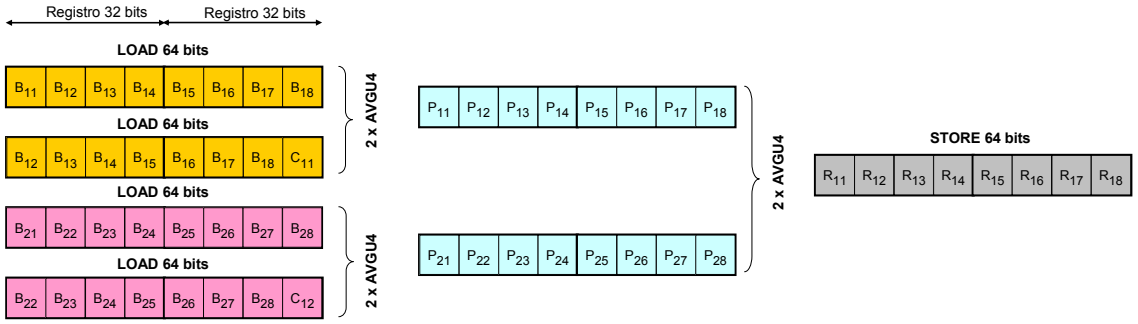


Figura 4-21. Procedimiento empleado para realizar el cálculo de los píxeles interpolados de una fila (R_{11}-R_{18}) perteneciente a un bloque de 8×8 píxeles.

Este proceso se repite para todas las filas del bloque teniendo en cuenta que es posible reutilizar datos previamente leídos y promediados. Así, para obtener los píxeles R_{21}-R_{28} es necesario disponer de los píxeles B_{21}-B_{28} y C_{21} y el promedio entre ellos (P_{21}-P_{28}) que ha sido calculado previamente. El algoritmo descrito se ha codificado directamente en lenguaje ensamblador y se ha paralelizado manualmente para obtener el mayor grado de optimización posible.

Las operaciones necesarias para calcular todos los píxeles intermedios de un bloque de 8×8 píxeles son 18 lecturas de 64 bits para obtener los 81 píxeles involucrados, 34 promedios[109] y 8 almacenamientos de 64 bits para preservar los resultados obtenidos. La suma total de operaciones es de 60 frente a las 320 que eran necesarias cuando se trabajaba a nivel de *byte* lo que supone una mejora teórica del 81%[110].

Referencia: Funciones en las que se realizan cargas, promedios y almacenamientos de varios píxeles [ficha 014].
Descripción: Recodificar las funciones en las que es posible emplear las instrucciones que permiten cargar/almacenar varios píxeles en los registros internos de la CPU y las instrucciones que permiten trabajar con ellos como si se tratara de varios datos de menor tamaño. Inicialmente se realiza una codificación de las funciones en lenguaje ensamblador lineal y en función del grado de paralelización que obtenga el compilador y la complejidad de la función, se valora la posibilidad de paralelizar las instrucciones manualmente.
Aplicabilidad: A todos los procesadores que posean una arquitectura SIMD. Para ello es necesario estudiar con detalle el repertorio de instrucciones del procesador. La función desarrollada es directamente aplicable a otros descodificadores que se desarrollen empleando procesadores que posean un repertorio de instrucciones compatible con el del TMS320DM642.

[109] Estas 34 operaciones provienen de los 2 promedios/fila × 9 filas + 2 promedios/intermedios × 8 filas.

[110] El cálculo del número de instrucciones necesarias es teórico puesto que no tiene en cuenta la optimización del código que es capaz de realizar el compilador. Dado que la versión inicial trabaja a nivel de *byte,* el compilador tiene más facilidad para paralelizarlo que el escrito directamente en ensamblador. La mejora real obtenida se encuentra en torno al 60%.

4.2.2.3.4 Suma del error de predicción

La suma del error de predicción se realiza empleando un proceso similar al descrito para la compensación de movimiento de ½ píxel. La principal diferencia se encuentra en que el número de bits que tienen algunos de los datos no es el mismo ya que los píxeles de la referencia poseen 8 bits mientras que los datos obtenidos tras calcular la IDCT son de 9 bits[111] (8 bits más el signo).

El DSP utilizado no posee instrucciones que permitan la suma en paralelo de datos cuando tienen diferente número de bits. Por tanto, para poder aprovechar la instrucción de suma en paralelo de dos datos almacenados en sendos registros de 32 bits es necesario que ambos datos posean 16 bits. El resultado de la IDCT ya está codificado con ese número de bits; sin embargo, los píxeles de referencia (8 bits) deben transformarse para cumplir este requisito[112]. Una vez transformados, los datos se suman para obtener los píxeles reconstruidos que se representan con 16 bits, aunque sólo son necesarios 8 bits. Finalmente, el procesador posee instrucciones específicas de empaquetado que permiten el paso de varios datos en paralelo de 16 a 8 bits[113].

La Figura 4-22 resume el proceso y las instrucciones utilizadas para realizar la suma. Inicialmente, cada 2 píxeles del bloque de referencia de 8 bits (por ejemplo B_{11} y B_{12}), se expanden (UNPACK) a datos de 16 bits (B'_{11} y B'_{12}) que se almacenan en un registro de 32 bits. Estos dos datos se suman en paralelo con los dos primeros valores del error de predicción (C_{11} y C_{12}) empleando una instrucción de suma en paralelo (ADD2) y almacenando el resultado (R'_{11} y R'_{12}) en un nuevo registro de 32 bits. Estos dos píxeles reconstruidos, junto con los obtenidos de forma análoga a partir de B_{13}, B_{14} C_{13} y C_{14}, se empaquetan como cuatro datos de 8 bits (R_{11}, R_{12}, R_{13} y R_{14}) en un registro de 32 bits. Finalmente, el resultado se transfiere al *buffer* de memoria en el que se almacena el macrobloque reconstruido.

Por tanto, para calcular los 64 píxeles de un bloque de 8×8 son necesarias las siguientes operaciones:

- 8 cargas de 64 bits para los píxeles del bloque de referencia y 16 para los valores obtenidos tras el cálculo de la IDCT.
- 32 instrucciones de desempaquetado de datos para almacenar los píxeles de referencia con un tamaño de 16 bits.
- 32 sumas de dos datos de 16 bits con signo en paralelo.
- 32 instrucciones de empaquetado de datos para que los píxeles almacenados con 16 bits se compacten en registros de 32 bits.
- 8 almacenamientos de 64 bits de los resultados en el *buffer* en el que se encuentra el bloque reconstruido.

Para un bloque de 8×8 píxeles son necesarias 112 instrucciones frente a las 256[114] que emplea el código de partida sin optimizar, lo que supone una mejora en torno al 50%. Este cálculo se refiere al número de operaciones que es necesario

[111] Para poder almacenar datos de 9 bits es necesario emplear variables/registros de 16 bits.

[112] Para ello el procesador tiene instrucciones específicas de desempaquetado (UNPACK) y empaquetado (PACK) cuya funcionalidad se mostrará posteriormente.

[113] Los datos de 16 bits sin signo se saturan a 255 cuando se empaquetan en datos de 8 bits.

[114] Las operaciones son 128 cargas, 64 sumas y 64 almacenamientos.

realizar en el que no se considera la paralelización en la ejecución de las instrucciones que realiza el compilador del DSP.

Comparando el número de ciclos medio de reloj que se requiere para obtener un macrobloque reconstruido entre la versión optimizada y la codificada íntegramente en lenguaje C, se obtiene una reducción del número de ciclos de reloj del 40%.

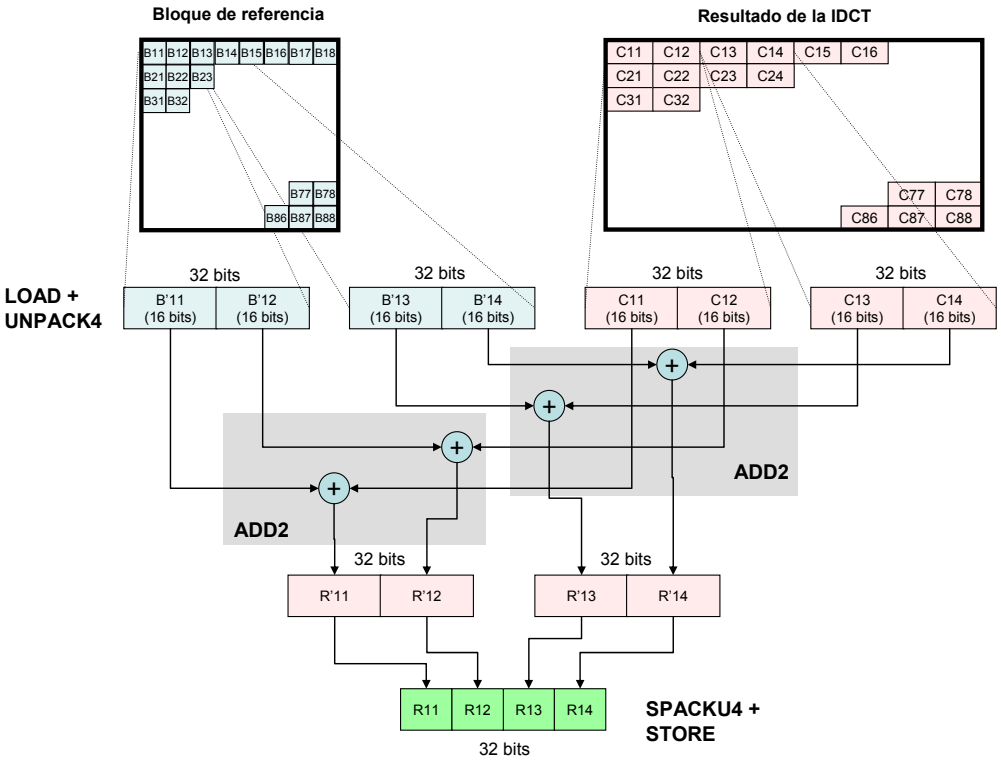


Figura 4-22. Proceso de suma de un bloque de referencia de 8×8 píxeles con el error de predicción.

Referencia: Emplear instrucciones de empaquetado y desempaquetado de datos [ficha 015].
Descripción: Emplear las instrucciones que permitan compactar/expandir los datos que se almacenan en los registros internos de la CPU cuando se recodifican funciones que manejen datos de diferentes tamaños.
Aplicabilidad: A todos los procesadores que posean una arquitectura SIMD y que incluyan instrucciones de empaquetado y desempaquetado de datos. La funcionalidad desarrollada (suma del error de predicción), con algunas pequeñas modificaciones, puede ser directamente aplicada a otros descodificadores que se desarrollen para los procesadores de la misma familia.

4.2.3 Optimización a nivel de sistema del descodificador MPEG-2

La integración del descodificador en un sistema complejo requiere algunas optimizaciones a nivel de sistema para asegurar que el rendimiento del descodificador no se ve afectado por la inclusión de otras tareas que debe realizar el procesador.

El STB-IP en el que se integra el descodificador se ha desarrollado empleando tareas del RTOS (ver Anexo B apartado 8.1.1) de modo que la configuración de las mismas influye de forma significativa en el rendimiento global del sistema. Concretamente, es necesario tener en cuenta la configuración de la prioridad asignada a cada una de ellas, el reparto de las peticiones de DMA entre las diferentes colas de petición que posee el controlador y la distribución del código y los datos en los diferentes niveles de memoria.

El sistema está compuesto por 6 tareas: análisis de la trama de entrada, interfaz de usuario, descodificación de audio y vídeo y presentación de ambos. El RTOS permite asignar prioridades a las tareas de modo que si una de ellas solicita su ejecución mientras la CPU está atendiendo a otra de menor prioridad, la primera pasa a ejecutarse y una vez que finalice (o que se quede bloqueada a la espera de algún evento) se devuelve el control a la segunda. Por tanto, si una tarea de alta prioridad se ejecuta durante un largo periodo de tiempo, impide que otras de menos prioridad puedan ser atendidas con la posible pérdida de datos por parte de alguna de ellas.

En el caso del STB-IP la tarea que necesita un mayor tiempo de ejecución es la de descodificación de vídeo por lo que se le asigna una prioridad baja, en el otro extremo se encuentra la tarea que procesa los datos que se reciben a través de la red Ethernet que posee un tiempo de ejecución reducido pero que necesita ser atendida rápidamente para que no se pierdan los datos que recibe.

Teniendo en cuenta estas dos cuestiones, la asignación de prioridades que se ha realizado es la siguiente: la tarea más prioritaria es la que analiza la trama por el motivo expuesto anteriormente, el siguiente nivel de prioridad lo ocupan las tareas de presentación (audio y vídeo) y la de descodificación de audio puesto que su tiempo de ejecución es reducido y no es admisible que se pierdan los datos que procesan. Seguidamente se encuentra la descodificación de vídeo y, finalmente, la interfaz de usuario debido a que su tiempo de respuesta es el menos crítico de todos.

Dado que hay cuatro tareas más prioritarias que la de descodificación de vídeo, es probable que alguna de ellas interrumpa al descodificador, lo que implica que algunos datos y/o código sean desalojados de las memorias caché con la correspondiente pérdida de rendimiento.

La Figura 4-23 muestra el orden de ejecución de las tareas para un ejemplo en el que periódicamente, y en los instantes marcados sobre el eje de tiempos, se reciben datos a través de la red Ethernet. En la figura se observa como la tarea de descodificación de vídeo (tarea amarilla) es interrumpida por la de recepción y análisis de un paquete de datos (tarea verde). El análisis de los datos provoca la activación sucesiva de las tareas de procesamiento y presentación de una trama de audio (tareas azul y negra). Durante la ejecución de esta última se recibe un nuevo paquete de datos y la tarea correspondiente se activa. Finalizada la presentación del audio, la tarea de descodificación de vídeo vuelve a ser atendida. Una vez finalizada ésta, se ejecuta la tarea de presentación de vídeo (tarea roja) que envía la imagen al monitor. Cuando termina, se pasa a ejecutar la tarea de reposo (tarea blanca) a la espera de la recepción de nuevos datos.

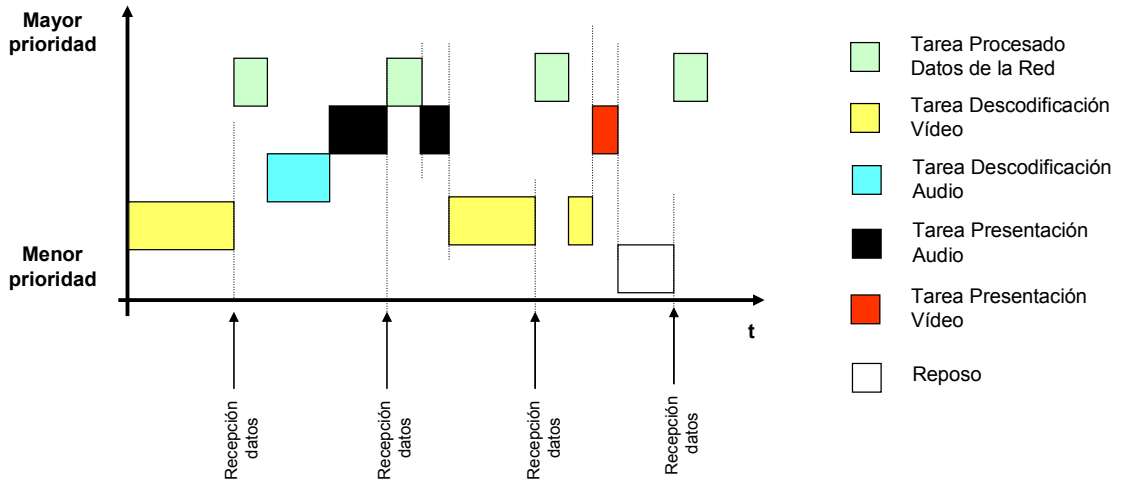


Figura 4-23 Distribución del tiempo de ejecución de las tareas por parte del RTOS.

Referencia: Asignación de prioridades a las tareas que forman el sistema [ficha 016].
Descripción: Evaluar el tiempo de ejecución de cada una de las tareas que forman el sistema y la posibilidad de pérdida de datos por parte de alguna de ellas si se demora su ejecución. Asignar mayor prioridad a aquellas tareas que requieran una atención más rápida y cuyo tiempo de ejecución sea bajo. Asignar menor prioridad a aquellas tareas que tengan un tiempo de ejecución elevado para que puedan ser interrumpidas por el resto.
Aplicabilidad: A cualquier sistema en el que exista un RTOS que permita asignar prioridades a las tareas que componen el sistema.

Una vez integrado el descodificador con el resto de tareas que componen el sistema completo es necesario replantear el reparto de las peticiones de transferencias por DMA. Para esta asignación se ha tenido en cuenta que la obtención de datos a través de la red local, la tarea de presentación de vídeo y el controlador del puerto de vídeo también emplean intensivamente el controlador de DMA. Sin embargo, el número de peticiones que acumulan estos tres procesos es menor[115] que las que realiza el descodificador de vídeo, si bien el tamaño de las mismas es considerablemente mayor sobre todo en los dos últimos casos, puesto que se transfieren imágenes completas.

Para el desarrollo del sistema se ha optado por asignar las peticiones de la red local a la cola de prioridad alta (que compartirán con las transferencias de luminancia), las asociadas a la tarea de presentación de vídeo a la de prioridad media y las del

[115] El puerto de red solicita una transferencia por cada paquete Ethernet que recibe (1322 *bytes*). Para una velocidad de transmisión de 2 Mbps, esto equivale aproximadamente a unas 190 solicitudes de transferencia por segundo. Por otro lado, tanto la tarea de presentación como el controlador de vídeo, solicitan tres transferencias por imagen lo que equivale a 75 transferencias por segundo. Finalmente, y por tener una referencia aproximada, una imagen tipo P de tamaño PAL en la que la mitad de sus macrobloques tengan predicción requiere un total de 10368 transferencias (1728 macrobloques × 6 transferencias/macrobloque) solicitadas por el descodificador de vídeo.

controlador de vídeo a la de baja prioridad. Con esta distribución se ha comprobado de forma experimental que en ningún caso se acumulan más peticiones de las que se puede encolar el controlador de DMA, por lo que nunca se producen paradas en la ejecución por parte de la CPU.

Referencia: Gestión de las peticiones de transferencias de DMA con el descodificador integrado en el sistema completo [ficha 017].
Descripción: Analizar las peticiones que realizan el resto de elementos que componen el sistema (resto de tareas y controladores de periféricos). Asignar sus peticiones a las diferentes colas teniendo en cuenta las que realiza el descodificador para lograr una distribución final uniforme entre todas las colas de petición. Comprobar experimentalmente que en ningún caso se encolan simultáneamente más solicitudes que el tamaño de cada una de las colas.
Aplicabilidad: A procesadores en los que el controlador de DMA disponga de colas de petición con prioridad.

Finalmente, es necesario asignar el código y los datos de las tareas a los diferentes niveles de memoria que posee el procesador. La Tabla 4-4 muestra el tamaño y ubicación de los diversos elementos que componen una tarea (código, variables globales y pila) para todas las que forman el sistema.

	Código	Variables Globales	Pila
Transporte	6 kB (Interna)	2 kB (Interna) 1 kB (Externa)	1 kB (Interna)
Descodificación Video	40 kB (Interna)	3 MB (Externa)	2 kB (Interna)
Descodificación Audio	4 kB (Interna) 16 kB (Externa)	45 kB (Externa)	4 kB (Interna)
Presentación Video	0.1 kB (Externa)	1 kB (Externa)	0.5 kB (Externa)
Presentación Audio	0.1 kB (Externa)	--	2 kB (Interna)
Interfaz Usuario	0.2 kB (Externa)	--	0.2 kB (Interna)

Tabla 4-4. Asignación del código y los datos de las tareas del sistema a los niveles de memoria.

Referencia: Ubicación en la jerarquía de memoria del código y los datos de las tareas que componen el sistema [ficha 018].
Descripción: Analizar el tamaño del código y los datos del sistema así como la pila asignada a cada una de las tareas. Ubicar en memoria interna la pila de las tareas así como el código y los datos del descodificador de vídeo siguiendo los criterios mostrados en el apartado 4.2.2.1.
Aplicabilidad: A los sistemas gestionados mediante tareas.

4.2.4 Resultados de la optimización

En el presente apartado se muestran los resultados obtenidos tras aplicar las técnicas de optimización que se han descrito en los apartados anteriores sobre el descodificador MPEG-2. Para obtener los resultados se han empleado los tres bancos de test descritos en el apartado 4.1.2.

4.2.4.1 Rendimiento del descodificador MPEG-2 aislado

Este apartado muestra la evolución en el rendimiento del descodificador MPEG-2 al aplicar las técnicas de optimización descritas anteriormente (apartados 4.2.2.1, 4.2.2.2 y 4.2.2.3). Las medidas se han obtenido empleando un banco de test como el que se muestra en la Figura 4-1. En dicho banco de test un módulo lee la trama elemental desde un fichero del PC, la descodifica y almacena el resultado en otro fichero del PC en formato YUV 4:2:0.

La aplicación de algunas de estas técnicas tiene influencia sobre el resto por lo que no es posible hacer un análisis de cada una de ellas por separado. Por este motivo los resultados que se presentan son acumulativos, lo que implica que para valorar la mejora obtenida al utilizar una de las técnicas, ésta se aplica sobre una versión del descodificador que incluye las anteriores.

El orden en el que se presentan los resultados se corresponde con la secuencia temporal en la que se han ido aplicando las optimizaciones durante el desarrollo de la tesis. Así, se presentan los resultados para 6 etapas del proceso de optimización:

1. C nativo. Se trata de la versión inicial en la que tanto el código como los datos se encuentran en memoria externa y la caché de nivel 2 está deshabilitada.

2. Caché L2. Se habilita la caché L2 que se configura con un tamaño de 256 kB.

3. IDCT+VLD+Memoria Interna. Se sustituyen las funciones en C por sus equivalentes en lenguaje ensamblador incluidas en las librerías que proporciona el fabricante. Además, se reduce el tamaño de la caché L2 a 128 kB y se introducen en la memoria interna de propósito general tanto las funciones en lenguaje ensamblador, como las que realizan la compensación de movimiento.

4. DMA. Se aplican las optimizaciones relacionadas con el controlador de DMA descritas en el apartado 4.2.2.2 a excepción del procesamiento de tiras de macrobloques (apartado 4.2.2.2.7). Además los *buffers* intermedios y las funciones que emplean DMA se ubican en memoria interna de propósito general.

5. Arquitectura SIMD. Las funciones que calculan la interpolación de píxeles de macrobloques diferentes (4.2.2.3.3) y la suma del error de predicción (4.2.2.3.4) se recodifican en lenguaje ensamblador y se mantienen en memoria interna de propósito general.

6. Versión Final. Se organizan los datos y las funciones en memoria interna, tal y como se describió en los apartados 4.2.2.1.2 y 4.2.2.1.3 y se emplea el controlador de DMA para realizar transferencias por tiras de macrobloques (apartado 4.2.2.2.7).

En la Figura 4-24 se observa el número de millones de ciclos de reloj invertidos en media en la descodificación de una imagen tras aplicar cada una de las etapas del proceso de optimización. Para la secuencia GI_9 se han descodificado las 15 imágenes que posee mientras que para la secuencia Arte se han procesado 100 imágenes.

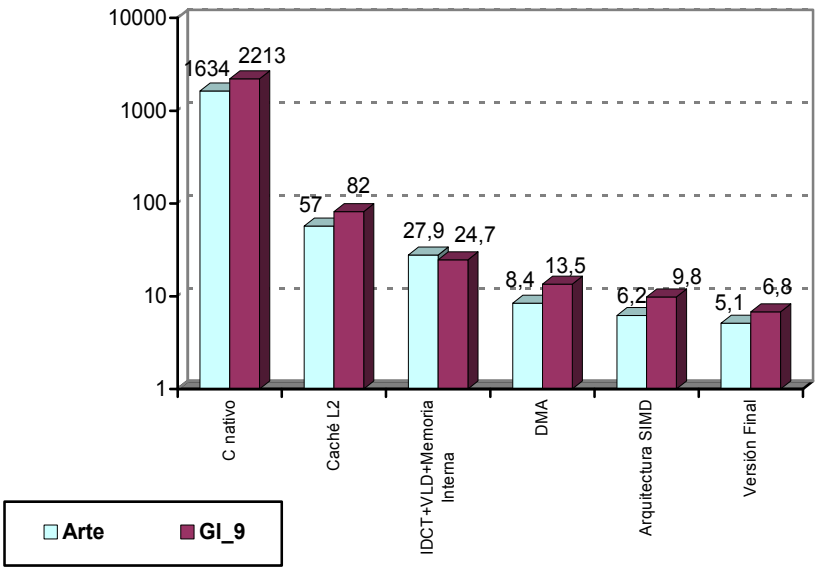


Figura 4-24. Evolución del número de millones de ciclos de reloj medios empleados para la descodificación de una imagen tras aplicar las etapas de optimización definidas.

Por otra parte, en la Figura 4-25 se aprecia el aumento del número de imágenes por segundo que es posible descodificar empleando un TMS320DM642 funcionando a 600 MHz según se han ido aplicando los diferentes pasos del proceso de optimización. Los resultados obtenidos corroboran que se ha logrado holgadamente el funcionamiento en tiempo real empleando entre un 21% y un 28% de CPU para descodificar las secuencias utilizadas.

Los resultados garantizan el funcionamiento en tiempo real del descodificador MPEG-2, dejando disponible más del 70% de la CPU para que en el sistema se incluyan otras tareas como pueden ser la descodificación de audio, la presentación del vídeo a través de un monitor, el análisis de la trama de entrada, etc.

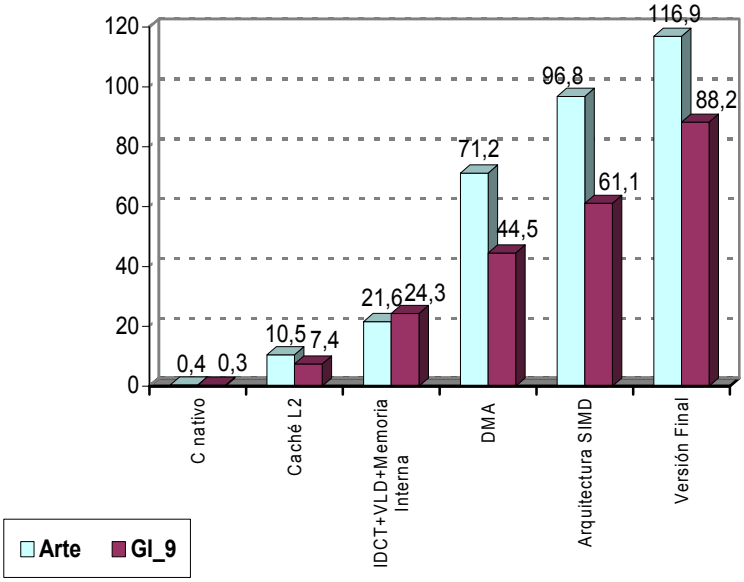


Figura 4-25. Evolución del número medio de imágenes por segundo que es posible descodificar en las 6 etapas del proceso de optimización definidas suponiendo una frecuencia de reloj de 600 MHz.

Además, sobre la versión final, se ha realizado un análisis de cada uno de los bloques funcionales que componen el lazo de descodificación. Con los resultados se ha elaborado la Tabla 4-5 en la que se presenta el número de ciclos de reloj que emplea cada uno de los bloques. En la Figura 4-26 se ha representado esta misma información de forma porcentual.

Bloque funcional	Arte	GI_9
VLD y cuantificación inversa	$0.67 \cdot 10^6$	$1.86 \cdot 10^6$
IDCT	$1.08 \cdot 10^6$	$0.82 \cdot 10^6$
Compensación del movimiento	$2.88 \cdot 10^6$	$3.72 \cdot 10^6$
Otras funciones	$0.50 \cdot 10^6$	$0.40 \cdot 10^6$
TOTAL	**$5.13 \cdot 10^6$**	**$6.80 \cdot 10^6$**

Tabla 4-5. Ciclos de reloj invertidos en cada uno de los bloques funcionales asociados a la descodificación de una imagen.

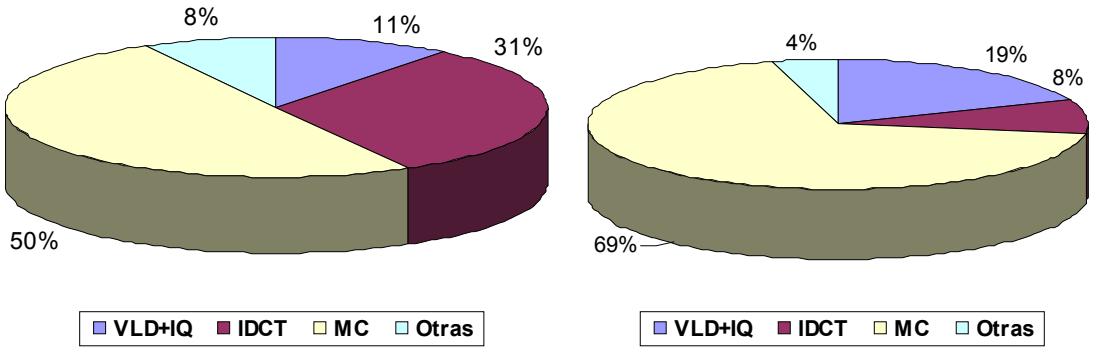


Figura 4-26 Reparto porcentual del tiempo empleado para descodificar una imagen entre los diferentes bloques funcionales (figura de la izquierda para la secuencia Arte y figura de la derecha para la secuencia GI_9).

4.2.4.2 Rendimiento del descodificador MPEG-2 integrado en un sistema de descodificación completo

En este banco de pruebas el descodificador de vídeo es integrado en el sistema de descodificación de televisión digital completo (STB-IP) que se describe en el Anexo B. Una vez que se dispuso del sistema completo de descodificación se realizaron medidas de rendimiento para comprobar la influencia del resto del sistema en el tiempo de ejecución del descodificador de vídeo utilizando el banco de test.

Para analizar el rendimiento del sistema se caracteriza el de cada una de las tareas que lo componen. Para ello se han utilizado las mismas tramas elementales empleadas para las medidas del descodificador aislado (apartado 4.2.4.1). Ambas tramas se han encapsulado en una trama de transporte MPEG-2 y se han difundido a través de la red Ethernet empleando la herramienta VLC [VLC]. En el caso de la secuencia obtenida de una emisión de televisión digital se ha incluido en la trama de transporte la información del audio correspondiente[116].

[116] Para la secuencia GI_9 no se ha incluido información de audio en las tramas de transporte.

La Tabla 4-6 muestra el porcentaje de uso de la CPU para cada una de las tareas que componen el sistema cuando se transmiten las dos secuencias indicadas anteriormente. La columna "Otros" indica el tanto por ciento de CPU que emplean algunos elementos auxiliares como la pila TCP/IP o la sobrecarga de las comunicaciones entre las tareas.

	Transporte	Desc. Vídeo	Present. Video	Desc. Audio	Present. Audio	Otros	Libre
Arte	4.1%	23.8%	0.1%	2.2%	0.5%	10.2%	52.1%
GI_9	15.8%	31.1%	0.1%	---	---	11.1%	41.9%

Tabla 4-6. Porcentaje de uso de la CPU empleado por cada una de las tareas para las dos secuencias de test.

A la vista de los resultados, es claro que se obtiene un funcionamiento en tiempo real holgado quedando más de un 40% de CPU disponible para otras tareas. Además, se aprecia que la influencia que tiene la integración del descodificador MPEG-2 con el resto de tareas es menor de un 12% de su carga computacional[117].

4.2.4.3 Descodificador MPEG-2 integrado en un entorno real

Para finalizar con las medidas de rendimiento se han empleado las dos configuraciones de pruebas descritas en el apartado 4.1.2.3 que permiten comprobar el funcionamiento del sistema en un entorno real con secuencias de larga duración. En ambas configuraciones se han realizado pruebas, tanto con varias películas en DVD, como con varios canales de televisión vía satélite.

Para realizar las medidas de rendimiento que se presentan en este apartado se han seleccionado las siguientes secuencias:

- Para la configuración en la que se emplea la secuencia MPEG-2 almacenada en un DVD se ha utilizado la película "La Guerra de las Galaxias. Episodio I" puesto que posee un elevado número de eventos en muchos fragmentos de la misma. Esta secuencia tiene un régimen binario medio de 5.6 Mbps, un 9% de imágenes I, un 32% de imágenes P y un 59% de imágenes B. El audio se encuentra codificado en formato AC3 [ATSC05] con un régimen binario de 128 kbps.
- Para la configuración que emplea un *gateway* comercial para encapsular las tramas de transporte en paquetes IP se ha utilizado el canal "Eurosport" debido a que las emisiones de este canal suelen incluir fragmentos con gran cantidad de eventos. Esta secuencia posee un régimen binario medio de 3.2 Mbps, un 5.6% de imágenes I, un 27.7% de imágenes P y un 66.7% de imágenes B. El audio se encuentra codificado en formato MPEG-1 Capa II con dos canales a 128 kbps. Este banco de pruebas se presenta en la Figura 4-27. El *gateway* es el chasis de la parte izquierda, en la parte central se encuentra la tarjeta DESCOS (ver apartado 7.1) en la que está implementado el STB-IP y en la parte superior se observa el monitor de televisión en el que se presenta tanto el audio como el vídeo.

[117] Para la secuencia GI_9 la sobrecarga es mayor que para la secuencia extraída de un canal de televisión. Esto se debe a que el régimen binario de la primera es considerablemente mayor que el de la segunda por lo que la tarea que analiza la trama de transporte demanda una mayor carga computacional lo que provoca un mayor número, tanto de transferencias de DMA, como de interrupciones en la ejecución del descodificador de vídeo.

Figura 4-27 Banco de pruebas empleando un *gateway* comercial.

La Tabla 4-7 muestra el porcentaje de uso de la CPU para cada una de las tareas que componen el sistema empleando las dos secuencias indicadas anteriormente ("La Guerra de las Galaxias. Episodio I" y "EuroSport"). En ambos casos los resultados se han obtenido tras descodificar aproximadamente una hora de vídeo (unas 90000 imágenes). Los resultados muestran el funcionamiento en tiempo real dejando más de un 45% de CPU disponible para otras tareas.

	Transporte	Desc. Vídeo	Desc. Audio	Presenta. Vídeo	Presenta. Audio	Otros	Libre
La Guerra de las Galaxias	4.9%	32.8%	2.8%	0.1%	0.7%	11.3%	47.4%
EuroSport	4.1%	28.5%	2.0%	0.1%	0.7%	11.3%	53.3%

Tabla 4-7. Porcentaje de uso de la CPU utilizado por cada una de las tareas para dos entornos de funcionamiento reales.

4.3 Optimización del tiempo de ejecución de un descodificador MPEG-4

En este apartado se presenta el trabajo desarrollado para optimizar un descodificador de vídeo compatible con la parte 2 de la norma ISO/IEC 14496.

Como se indicó en el apartado 4.1.1, durante el desarrollo de los trabajos de esta tesis, se decidió aplicar un esfuerzo limitado para la optimización de este descodificador por su escasa implantación industrial. No obstante, ha servido para comprobar cuáles de las técnicas de optimización identificadas en el trabajo con el estándar MPEG-2 (apartado 4.2) son aplicables también al estándar MPEG-4.

El trabajo desarrollado para este estándar sigue el método descrito en el apartado 4.1.

4.3.1 Elección del *software* de referencia

Como se presentó en el apartado 2.2.3.2, MPEG-4 vídeo define 19 perfiles y 5 niveles. Analizando todos ellos se observa que el perfil Simple Avanzado *(Advanced Simple Profile* o ASP) es el de mayor interés desde el punto de vista de la difusión de televisión y por tanto, la búsqueda de un *software* de referencia se acotó a los que implementan este perfil del estándar.

Para la realización de este descodificador, se han evaluado los diversos códigos fuente disponibles en el momento en el que comenzaron los trabajos de tesis relacionados con este estándar. En la Tabla 4-8 se presenta una breve descripción de los códigos encontrados y en la Tabla 4-9 una comparación de las herramientas soportadas por cada uno de ellos.

Para validar la conformidad de estos descodificadores con el estándar se han seleccionado algunas de las secuencias incluidas en el test de conformidad descrito en el estándar MPEG-4, parte 4, [ISO04c] para el perfil ASP. Las pruebas se han realizado con más de 30 secuencias y la conclusión más relevante es que el descodificador XviD no es capaz de procesar adecuadamente la mayor parte de ellas y por este motivo se ha descartado. Los otros dos descodificadores sí son capaces de procesar estas secuencias.

Tras realizar un análisis de la complejidad de ambos descodificadores se aprecia que el descodificador proporcionado por ISO es considerablemente más complejo puesto que implementa todos los perfiles y niveles definidos en el estándar. Por el contrario, el código de FFMPEG que implementa el descodificador MPEG-4 es más sencillo puesto que sólo soporta los perfiles SP (*Simple Profile*) y ASP[118]. Además este segundo código tiene algunos bloques funcionales optimizados para arquitecturas de 32 bits lo que facilita su adaptación a la arquitectura del procesador TMS320DM642. Por estos motivos se selecciona como punto de partida el *software* proporcionado por FFMPEG.

Software	Descripción	Procedencia
ISO/IEC 14496-5:2001	Software de referencia del estándar MPEG-4 [ISO01] para codificar y descodificar secuencias de vídeo. Esta versión implementa todas las funcionalidades y herramientas descritas en el estándar.	www.iso.org
FFMPEG	Librería de transcodificación de audio y vídeo multiestándar [FFMPEG]. Admite secuencias de entrada codificadas según los principales estándares de codificación de vídeo, entre los que cabe destacar H.261, H.263, MPEG-1, MPEG-2, MPEG-4 y H.264. Respecto a MPEG-4, sólo soporta vídeo natural con imágenes rectangulares.	www.ffmpeg.org
XviD	Codificador/descodificador de vídeo basado en el formato de compresión MPEG-4. Soporta, en función del perfil XviD utilizado[119], distintas herramientas de MPEG-4: compensación con precisión de ¼ píxel, compensación de movimiento global, video entrelazado, etc. Sólo admite secuencias de vídeo natural con imágenes rectangulares.	www.xvid.org

Tabla 4-8. Relación de los códigos de referencia evaluados para el desarrollo del descodificador MPEG-4.

[118] A pesar del que el código que implementa el descodificador MPEG-4 es más sencillo, el tamaño de la librería es elevado debido a la gran cantidad de codificadores y descodificadores que implementa lo que obligó a realizar un trabajo de selección del código asociado al estándar MPEG-4.

[119] Los cuatro perfiles definidos en XviD que admite este *software* son: *Simple Profile* (SP) o *DivX Portable, DivX Handheld*, *DivX Home Theatre* o *DivX High Definition* y *Advanced Simple Profile* (ASP).

Herramientas	ISO/IEC 14496-5:2001	FFMPEG	XviD
Predicción AC	Soportado	Soportado	Soportado
B-VOPs	Soportado	Soportado	Soportado
4 vectores de movimiento	Soportado	Soportado	Soportado
Compensación de ¼ píxel	Soportado	Soportado	Soportado
Compensación global	Soportado	Soportado	Soportado
Vídeo entrelazado	Soportado	Soportado	Soportado parcialmente
VOPs de baja resolución	Soportado	No soportado	Soportado
Rebanadas *(Slices)*	Soportado	Soportado	Soportado
Partición de datos	Soportado	Soportado	No soportado
VLC reversible	Soportado	Soportado	No soportado
Imágenes no rectangulares	Soportado	No soportado	No soportado

Tabla 4-9. Herramientas soportadas por los descodificadores MPEG-4 de los que se disponía del código fuente.

4.3.1.1 Descripción del algoritmo

FFMPEG es una librería que integra más de 20 codificadores y descodificadores, tanto de vídeo como de audio, compatibles con diferentes estándares, así como la capacidad de procesar diferentes formatos de ficheros. Su estructura permite añadir con relativa facilidad nuevos codificadores o descodificadores sin más que implementar un API (*Application Program Interface*) que permite a la aplicación que desea emplearlos comunicarse con ellos de forma estándar. La estructura descrita se muestra esquemáticamente en la Figura 4-28.

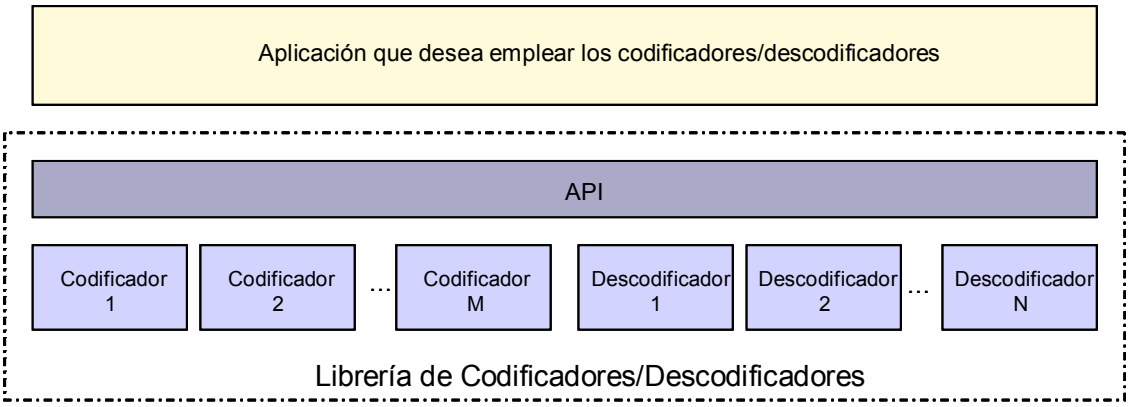


Figura 4-28 Estructura de la librería FFMPEG.

El descodificador seleccionado como punto de partida realiza la descodificación macrobloque a macrobloque. La Figura 4-29 muestra el ordinograma que implementa la descodificación de cada macrobloque. Junto al ordinograma se representan los *buffers* principales que se emplean de modo que cada parte del algoritmo se ha coloreado para relacionarla con los *buffers* que utiliza. La funcionalidad de cada uno de ellos es la siguiente:

- Trama elemental*: Buffer* de tamaño constante en el que se almacenan fragmentos de la trama codificada. El descodificador va analizando los datos almacenados en él hasta llegar a su finalización. Una vez terminado el análisis, se rellena de nuevo con el siguiente fragmento de la trama codificada.
- Bloque*:* contiene inicialmente los coeficientes de la DCT extraídos de la trama sobre los que se aplican los procesos de cuantificación inversa y DCT inversa. Su tamaño es constante e igual al tamaño de un macrobloque.
- *Buffers* Intermedios: Se trata de varios *buffers* intermedios en los que se almacenan las referencias que se emplean para la compensación de movimiento. Cada uno de ellos tiene el tamaño de un macrobloque.
- MB reconstruido: En este *buffer* se almacena el macrobloque reconstruido después de sumar la referencia almacenada en los *buffers* intermedios con el resultado de la IDCT. Su tamaño es el de un macrobloque.
- *Buffers* de Referencia: Estos tres *buffers* contienen las imágenes descodificadas por lo que el tamaño de cada uno de ellos es el de una imagen en formato 4:2:0.

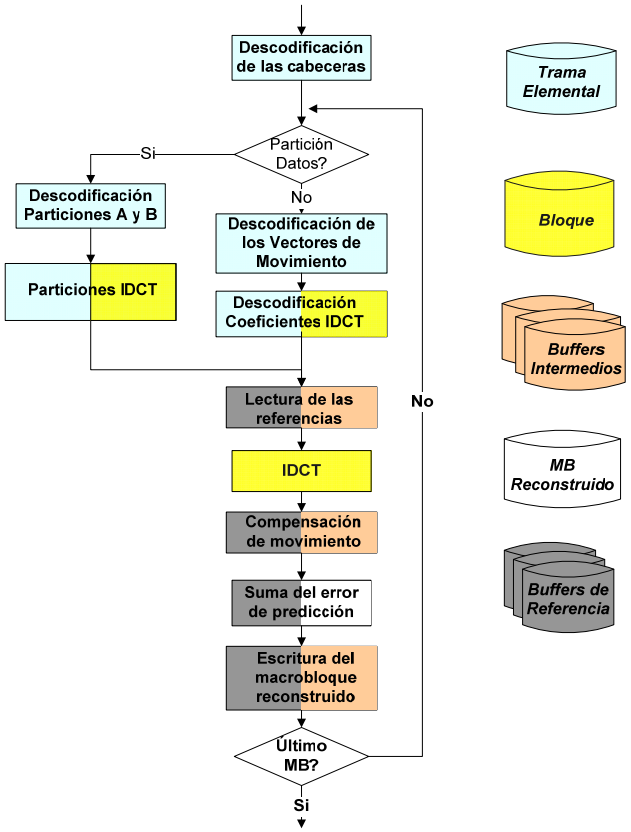


Figura 4-29 Ordinograma del bucle principal de descodificación para un macrobloque en el que se han incluido los *buffers* que se emplean en el proceso de descodificación.

4.3.1.2 Migración del descodificador MPEG-4 al entorno de desarrollo del procesador TMS320DM642

La librería FFMPEG está diseñada para ser ejecutada en diferentes plataformas *hardware* (Intel, Motorola, ARM, etc.). Por tanto, existen algunos módulos codificados para procesadores de 32 bits, lo que permite que el rendimiento inicial del descodificador en el DSP sea mayor que en el caso del descodificador MPEG-2.

El proceso de migración del descodificador al entorno del DSP es más complejo que el descrito en el apartado 4.2.1.2 para MPEG-2 ya que código de referencia seleccionado implementa varios descodificadores optimizados para diferentes arquitecturas. La adaptación a la arquitectura del DSP se ha realizado siguiendo las fases que se describen seguidamente:

1. Eliminación de ficheros y análisis de las opciones de compilación condicional. Todos los ficheros relacionados con otros estándares de codificación se han eliminado. Además, se han seleccionado las opciones de compilación condicional que mejor se adaptan a la arquitectura del DSP.

2. Creación de una librería para el entorno de desarrollo del DSP (*Code Composer Studio* o CCS) e importación del código de referencia. Una vez identificados los ficheros de interés, se han migrado al entorno de desarrollo y se han compilado. Durante este proceso ha sido necesario realizar algunas modificaciones puesto que el código de partida está escrito para el compilador *gcc* (GNU *Collection Compiler*), que admite algunas opciones no estandarizadas del lenguaje C y que no son soportadas por el compilador del DSP.

3. Creación de una aplicación de prueba. Para poder realizar pruebas con la librería migrada al entorno del DSP ha sido necesario desarrollar un banco de test como el mostrado en la Figura 4-1. Dicho banco de pruebas lee la trama elemental desde un fichero, realiza la descodificación y almacena en otro fichero las imágenes descodificadas en formato YUV. En el proceso de creación del banco de pruebas es necesario configurar el RTOS del procesador teniendo en cuenta las características de la tarjeta a emplear (mapa de memoria, frecuencia de reloj, periféricos disponibles, etc.).

4. Depuración del descodificador en tiempo de ejecución. Utilizando la aplicación desarrollada se han realizado pruebas que han permitido identificar un funcionamiento incorrecto del descodificador debido a que el código realiza lecturas y escrituras de datos de 16/32/64 bits ubicados en direcciones de memoria no alineadas al tamaño del dato, y el DSP no es capaz de realizar tales accesos desde el lenguaje C. Ha sido necesario identificar y reescribir los fragmentos de código en los que se realizan estas lecturas/escrituras empleando instrucciones en ensamblador que sí permiten este tipo de accesos.

5. Eliminación de código no utilizado. Parte del código incluido en el fichero que implementa el descodificador MPEG-4 se emplea realmente por parte de otros estándares por lo que se ha eliminado. Además, se han eliminado todas las funciones y variables relacionadas con la codificación, dejando exclusivamente el código asociado a la descodificación.

6. Creación de un banco de test automatizado. Dado el elevado número de secuencias para las que hay que comprobar su correcta descodificación, es necesario disponer de un banco de pruebas automático. El entorno de desarrollo permite elaborar *scripts* en Java que facilitan la ejecución del descodificador empleado en cada una de dichas ejecuciones una secuencia diferente. El resultado de la descodificación es comparado de forma también automática con el obtenido por parte del descodificador de referencia incluido en el estándar. Esta comparación se realiza en base al cálculo del PSNR.

Tras realizar todas las fases anteriores y repetir las pruebas de conformidad, se dispone de un descodificador MPEG-4 funcionalmente correcto compatible con el perfil Avanzado Simple. Esta versión del descodificador se ha utilizado como punto de partida en el proceso de optimización.

4.3.2 Optimización del tiempo de ejecución del descodificador MPEG-4

En los siguientes apartados se explica cómo se ha aplicado el proceso de optimización al descodificador MPEG-4. Dicho proceso es similar al descrito en el apartado 4.2.2 para el descodificador MPEG-2.

4.3.2.1 Arquitectura de la memoria

Siguiendo la técnica descrita en la [ficha 001] para el descodificador MPEG-2 (ver apartado 4.2.2.1) se ha realizado un reparto de la memoria interna entre caché de nivel 2 y memoria de propósito general idéntico al definido para el estándar anterior (128 kB de caché de nivel 2 y 128 kB de memoria de propósito general).

Teniendo en cuenta lo descrito en la [ficha 005], los *buffers* Trama Elemental y *Buffers* de Referencia (ver Figura 4-29), se almacenan en memoria externa mientras que el resto se ubican en la memoria interna de propósito general.

La memoria interna disponible, una vez que se alojan en ella los *buffers,* no es suficiente para almacenar todo el código que implementa el descodificador. Por este motivo se han de seleccionar aquellos fragmentos que deben ubicarse en dicha memoria. Los criterios empleados para hacer esta selección son dos: la carga computacional de las funciones y el número de fallos en el acceso a la caché de programa. Aquellas funciones que requieren una elevada carga computacional y provocan un gran número de fallos en el acceso a la caché se ubican en memoria interna, teniendo en consideración los criterios definidos en la [ficha 003].

Como consecuencia, el código que realiza la compensación de movimiento, tanto de ½ píxel, como de ¼ píxel, se ubica en memoria interna junto con la IDCT y algunas otras funciones pertenecientes al resto de los bloques funcionales.

Referencia: Criterios adicionales de selección de bloques funcionales que deben alojarse en memoria interna [ficha 019].
Descripción: Identificar las funciones del descodificador que comportan una mayor carga computacional así como aquellas que provocan más fallos en el acceso a memoria caché de programa. Ubicar estas funciones en memoria interna y evaluar la mejora que supone en el rendimiento del descodificador. Realizar varias distribuciones hasta obtener el mayor rendimiento. Esta técnica debe aplicarse a lo largo del proceso de optimización puesto que la aplicación de las otras técnicas de optimización puede provocar cambios significativos en el tamaño y rendimiento de las funciones, que harán que sea necesario modificar la distribución.
Aplicabilidad: A cualquier descodificador en el que el código no pueda alojarse completamente en la memoria interna de propósito general.

4.3.2.2 Controlador de Acceso Directo a Memoria

El empleo del controlador de DMA para el descodificador MPEG-2 se describió en el apartado 4.2.2.2. En el descodificador MPEG-4 se han empleado las técnicas

resumidas en las fichas [006], [007], [008], [010] y [011]. Por el contrario, no se han aplicado la modificación de la estructura del descodificador (ver [ficha 009], apartado 4.2.2.2.4) ni el procesado por tiras de macrobloques (ver [ficha 012], apartado 4.2.2.2.7) puesto que requieren la definición de *buffers* de gran tamaño en memoria interna, y ésta está ocupada por el código del descodificador.

La única diferencia en el empleo del controlador de DMA respecto al estándar MPEG-2 es el tamaño de las transferencias utilizadas para obtener los macrobloques de referencia cuando se emplean vectores de movimiento con resolución de ¼ píxel. En este caso, y como se expuso en el apartado 2.2.4.2.4.2, son necesarios píxeles pertenecientes a macrobloques vecinos para realizar el filtrado que permite obtener los píxeles intermedios. Por tanto, es necesario transferir bloques de 22×22 píxeles para la luminancia y de 14×14 para las crominancias, en lugar de los bloques de 17×17 y 9×9 respectivamente que se transferían en MPEG-2 (ver apartado 4.2.2.2.2).

Referencia: Tamaño de las transferencias de DMA para la compensación de movimiento con resolución de ¼ píxel [ficha 020].
Descripción: Analizar los píxeles implicados en la compensación de movimiento. Declarar *buffers* en memoria interna en los que se puedan almacenar todos los píxeles necesarios para realizar la compensación de movimiento con resolución de ¼ píxel. Transferir a estos *buffers* tanto el/los macrobloque/s de referencia como algunos píxeles de los macrobloques vecinos que permiten realizar la compensación de movimiento. Aplicar el filtrado sobre los datos transferidos almacenando el resultado en otro *buffer* intermedio alojado en memoria interna. El resto del proceso empleado para calcular el macrobloque reconstruido es idéntico al empleado en MPEG-2 (ver [ficha 007]).
Aplicabilidad: A todos los descodificadores que empleen vectores de movimiento con resolución de ¼ píxel.

Tras modificar el descodificador para hacer uso del controlador de DMA, el diagrama temporal que muestra la paralelización de los bloques funcionales en los que se descompuso el proceso de descodificación se muestra en la Figura 4-30. En dicha figura se aprecia como se paraleliza el cálculo de la IDCT (etapa B) con la transferencia de los macrobloques de referencia (etapa C); así como la transferencia de un macrobloque reconstruido (etapa E) con la extracción de los vectores de movimiento y los coeficientes de la DCT del siguiente (etapa A).

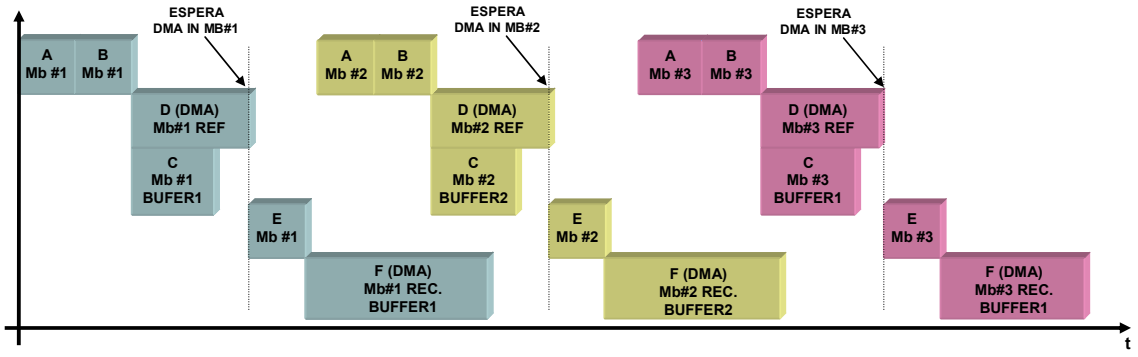


Figura 4-30 Paralelización de las etapas del proceso de descodificación de un macrobloque en MPEG-4.

4.3.2.3 Arquitectura SIMD

Varios módulos del descodificador MPEG-2 optimizados para aprovechar la arquitectura SIMD del procesador se han reutilizado en el descodificador MPEG-4 ASP. Los más destacados son la IDCT (ver apartado 4.2.2.3.1), la interpolación de píxeles pertenecientes a macrobloques diferentes (ver apartado 4.2.2.3.3) y la suma del error de predicción (ver apartado 4.2.2.3.4).

Además, se han identificado dos bloques funcionales que, debido a su elevada carga computacional, son susceptibles de ser optimizados aprovechando la arquitectura SIMD del DSP. Se trata de la descodificación VLD y el filtro de 8 etapas asociado a la compensación de movimiento con resolución de ¼ píxel. Sin embargo, estos bloques funcionales no han sido optimizados debido al esfuerzo limitado que se dedicó a este estándar (tal como se justificó en el apartado 4.1) y la imposibilidad de reutilizarlos posteriormente para optimizar otros estándares, ya que estas herramientas sólo se emplean en MPEG-4.

4.3.3 Optimización a nivel de sistema del descodificador MPEG-4

El descodificador optimizado se ha integrado en el sistema completo de descodificación de televisión digital. Para optimizar el funcionamiento de todas las tareas se siguieron los criterios descritos en las fichas [016] y [017].

4.3.4 Resultados de la optimización

En este apartado se presentan los resultados obtenidos tras aplicar las técnicas de optimización descritas en los apartados 4.3.2 y 4.3.3. Para extraer estos resultados se han empleado dos de los tres bancos de test descritos en el apartado 4.1.2: el primero para analizar el tiempo de ejecución del descodificador aisladamente en simulación (ver apartado 4.1.2.1) y el segundo para realizar pruebas de funcionamiento en un entorno real utilizando un codificador o un transcodificador para generar la secuencia a partir de una emisión vía satélite o de un DVD respectivamente (ver apartado 4.1.2.3).

4.3.4.1 Rendimiento del descodificador MPEG-4 aislado

En este banco de pruebas se ha desarrollado una aplicación que lee de un fichero la trama codificada y se la proporciona al descodificador. Este descodifica las imágenes dejando el resultado en un *buffer* que es finalmente almacenado en un fichero de salida (ver Figura 4-1).

Para realizar las medidas de rendimiento, tanto de la versión inicial del código como de las diversas versiones optimizadas, se han empleado secuencias definidas en el estándar [ISO04c]. En este apartado se han seleccionado 5 de ellas que se han considerado significativas para mostrar los resultados de rendimiento. La Tabla 4-10 recoge las secuencias seleccionadas y sus principales características. Estas secuencias emplean todas las herramientas definidas en el estándar para el perfil Simple Avanzado.

Secuencia	Perfil y Nivel	Tamaño [120]	Tipos de imágenes	Tipo de MC	Tipo de codificación	Otras herramientas
GE-16	ASP@L2	352×288	I + P	½ píxel	Progresiva	-
A1GE-10	ASP@L2	352×288	I + P	¼ píxel	Progresiva	-
A1GE-13	ASP@L1	352×288	I + P + B	¼ píxel	Progresiva	-
A1GE-11	ASP@L1	352×288	I + S	¼ píxel	Progresiva	GMC
er-3	SP@L3	352×288	I + P	½ píxel	Progresiva	Partición y RVLC

Tabla 4-10. Secuencias seleccionadas para realizar las medidas de rendimiento en simulación.

La Figura 4-31[121] muestra, para cada una de las secuencias seleccionadas, el número medio de ciclos de reloj que emplea el descodificador para procesar una imagen. Para cada secuencia se han realizado cuatro medidas que se corresponden con cuatro etapas del proceso de optimización; la columna "original" se corresponde con la versión inicial con la configuración de memoria caché descrita en 4.3.2.1; la columna "gestión de memoria", con una versión en la que se han distribuido los *buffers* y funciones entre los diversos niveles de memoria; la columna "funciones optimizadas" refleja la inclusión del código en ensamblador optimizado previamente para MPEG-2; por último, la columna "DMA" muestra el resultado final tras emplear el controlador de DMA para realizar las transferencias.

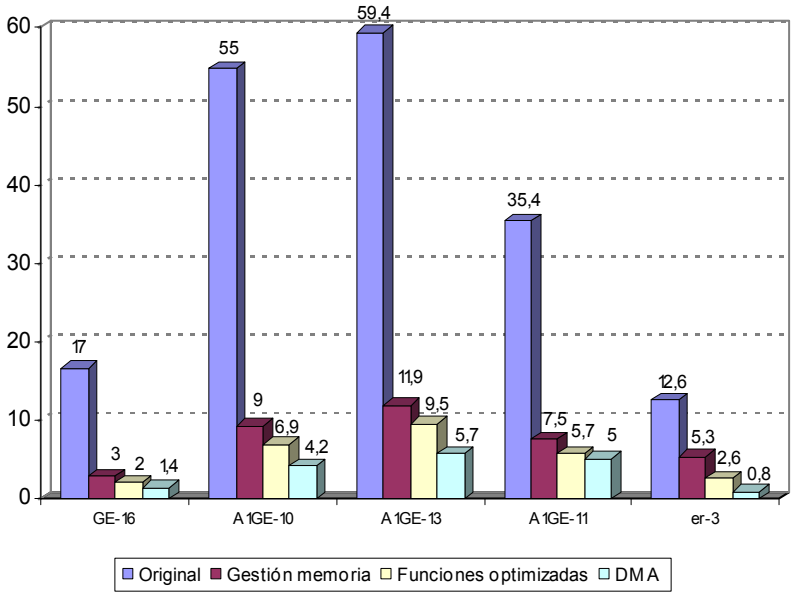


Figura 4-31 Evolución del rendimiento del descodificador MPEG-4 tras cada una de las etapas del proceso de optimización, expresado en millones de ciclos de reloj medios por imagen.

[120] La elección de secuencias de este tamaño viene justificada porque la mayor parte de las secuencias del estándar son de tamaño QCIF o CIF.

[121] En la figura se aprecia que el rendimiento de la versión empleada como punto de partida para el proceso de optimización del descodificador MPEG-4 es sustancialmente mejor que el obtenido para el descodificador MPEG-2 debido a que, como se indicó en el apartado 4.3.1.2, aquel ya incluía algunas optimizaciones que aprovechaban la arquitectura SIMD del DSP.

Por otra parte, la Figura 4-32 muestra el número medio de imágenes por segundo que es capaz de procesar un TMS320DM642 trabajando a 720 MHz. Analizando los resultados se aprecia que en todos los casos el descodificador emplea menos del 25% de la CPU lo que garantizaría su funcionamiento en tiempo real para imágenes con formato PAL[122].

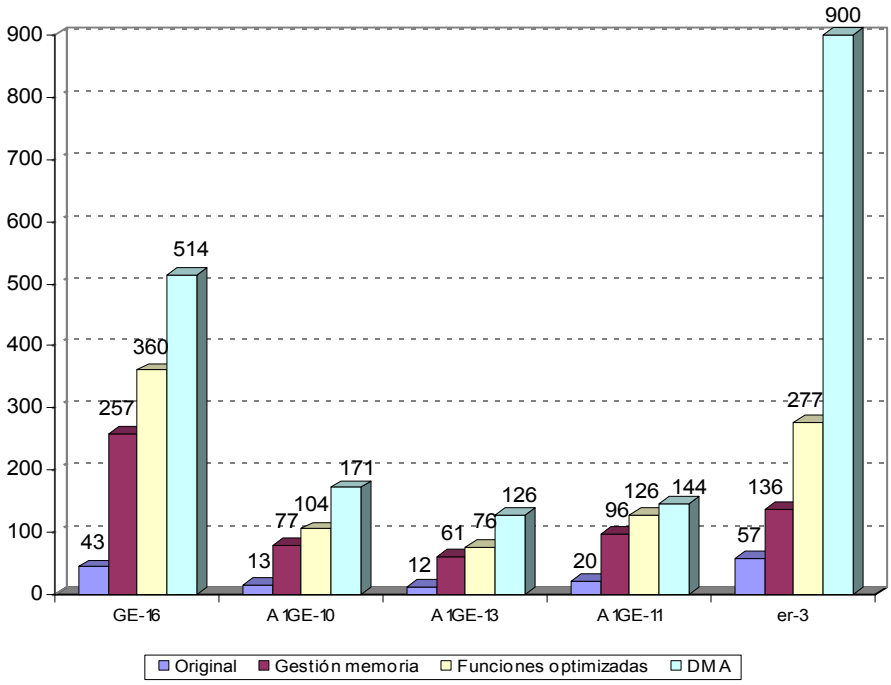


Figura 4-32 Evolución del rendimiento del descodificador MPEG-4 expresado en número medio de imágenes por segundo trabajando a 720 MHz.

Además de las tramas elementales definidas en el estándar, se han generado otras dos secuencias obtenidas a partir de un canal de televisión digital vía satélite (“*Arte*”) y de un DVD (*“Buscando a Nemo”*). Las imágenes de estas secuencias tienen tamaño PAL lo que ha permitido corroborar la aproximación teórica descrita en el párrafo anterior.

Las secuencias de prueba originales estaban codificadas en MPEG-2 y se han transcodificado a MPEG-4 con la herramienta VLC [VLC] empleando diferentes regímenes binarios[123]. La Tabla 4-11 muestra los resultados expresados en número medio de ciclos de reloj por imagen y el número de imágenes por segundo que es posible descodificar en cada caso con un procesador TMS320DM642 funcionando a 720 MHz. Los datos de rendimiento se han obtenido para una media de 2900 imágenes descodificadas y como se aprecia se ha logrado el funcionamiento en tiempo real para todas las secuencias.

[122] El formato de todas secuencias seleccionadas es CIF por lo que para saber si se lograría el funcionamiento en tiempo real se ha realizado una extrapolación al tamaño PAL. Para ello es necesario multiplicar el número de ciclos de reloj por 4.09 (=720×576/352×288).

[123] Las secuencias generadas tienen una distancia entre imágenes I de 16, 2 imágenes B por cada P, compensación de movimiento con resolución de ¼ píxel y sin compensación global de movimiento.

Secuencias		MPEG-4	
Nombre	**Régimen Binario**	**CPU CLK×10^6**	**Imágenes/segundo**
Buscando a Nemo	5.6 Mbps	19.81	36.4
	3 Mbps	18.32	39.3
	1 Mbps	14.34	50.3
Arte	4 Mbps	15.08	47.7
	3 Mbps	14.40	50.0
	1 Mbps	12.27	58.5

Tabla 4-11. Número de ciclos de reloj empleados en media e imágenes por segundo procesadas por el descodificador MPEG-4 para las secuencias de pruebas.

4.3.4.2 Rendimiento del descodificador MPEG-4 integrado en un entorno real

El descodificador optimizado se ha integrado en el sistema completo de descodificación de televisión digital descrito en el Anexo B. Para realizar las medidas se han empleado los dos bancos de test descritos en el apartado 4.1.2.3 y las secuencias presentadas en el apartado anterior (“Arte” y “Buscando a Nemo”).

El primer banco de test está formado por un PC que envía de forma continuada a través de la red Ethernet una trama de transporte MPEG-2 que incluye una secuencia de vídeo MPEG-4 y una trama de audio AC-3 (Buscando a Nemo). La trama es recibida por el STB-IP, descodificada y presentada en un monitor de televisión.

El segundo banco de pruebas está compuesto por un *gateway* [ETHTV] comercial que recibe un programa de televisión vía satélite en formato MPEG-2 (Arte) y lo encapsula en paquetes IP; estos paquetes son recibidos por un transcodificador [VLC] que los recodifica en tiempo real en formato MPEG-4 y los vuelve a enviar. Finalmente, los paquetes son recibidos, descodificados y presentados por parte del STB-IP. El formato de la trama de audio es MPEG-1 Capa II.

Los resultados de rendimiento obtenidos se presentan en dos tablas. La primera de ellas (Tabla 4-12) muestra el rendimiento del descodificador expresado en millones de ciclos de reloj por imagen en media para diferentes regímenes binarios de las secuencias con ambos bancos de test. Por otro lado, la segunda (Tabla 4-13) presenta el porcentaje de uso de la CPU de cada una de las tareas que componen el sistema cuando la trama elemental de vídeo MPEG-4 está codificada con 3 Mbps.

Banco de test	Régimen binario	MPEG-4 (CLK×10^6)
DVD (Buscando a Nemo)	5.6 Mbps	20.21
	3 Mbps	18.75
	1 Mbps	14.66
DTV (Arte)	4 Mbps	15.42
	3 Mbps	14.76
	1 Mbps	12.30

Tabla 4-12. Rendimiento del descodificador con diferentes regímenes binarios.

Tarea / %DSP	Transporte	Descod. de video	Descod. de audio	Present. de video	Present. de audio	Otros[124]	Libre
DVD (Nemo)	4.7%	78.1%	2.8%	0.1%	0.7%	2.1%	11.5%
DTV (Arte)	4.6%	61.5%	2.0%	0.1%	0.7%	2.1%	29.0%

Tabla 4-13. Rendimiento de las tareas que componen el STB-IP empleando una secuencias con 3 Mbps.

Como se aprecia a partir de los resultados obtenidos, una vez aplicadas las técnicas extraídas durante el proceso de optimización del descodificador MPEG-2, se logra el funcionamiento en tiempo real con secuencias MPEG-4 ASP en formato PAL.

4.4 Optimización del tiempo de ejecución de un descodificador H.264

En este apartado se presenta el proceso de optimización de un descodificador de vídeo acorde al estándar ISO/IEC 14496 parte 10 según la terminología de ISO o H.264 según la terminología ITU.T [ITU03].

La complejidad de este estándar en comparación con MPEG-2 y MPEG-4 hace que la carga computacional necesaria para implementar un descodificador sea mucho mayor que la requerida por sus predecesores [OBL+04]. Este hecho convierte al descodificador H.264 en un reto interesante para desarrollar metodologías de optimización de descodificadores de video sobre DSPs, puesto que para lograr el funcionamiento en tiempo real no es suficiente con aplicar las técnicas empleadas para MPEG-2 y MPEG-4 (apartados 4.2 y 4.3), sino que es necesario explorar otras técnicas de optimización.

Como se indicó en el apartado 4.1.1, este descodificador ha sido optimizado para dos procesadores (TMS320DM642 y TMS320DM6437) para confirmar que las técnicas de optimización son válidas para diferentes arquitecturas. Por este motivo, la información de este apartado se ha organizado del siguiente modo: en primer lugar se muestra el proceso de selección del *software* y el de migración al entorno de desarrollo de los DSPs (apartado 4.4.1), seguidamente se presenta el proceso de optimización y los resultados obtenidos para el procesador TMS320DM642 (apartados 4.4.2, 4.4.3 y 4.4.4) y, finalmente, se describen las técnicas aplicadas para el procesador TMS320DM6437 y los resultados alcanzados (apartados 4.4.5, 4.4.6 y 4.4.7).

4.4.1 Elección del *software* de referencia

El *software* empleado como punto de partida es el desarrollado dentro del proyecto FFMPEG por los motivos ya explicados para MPEG-4 en el apartado 4.3.1: simplicidad del código y la existencia de algunos bloques funcionales ya desarrollados para procesadores de 32 bits. Para comprobar la conformidad de este descodificador sobre entorno PC se ha seleccionado un conjunto amplio de las secuencias test incluidas en el estándar y los resultados obtenidos en todos los casos han sido satisfactorios, lo que valida la elección de este *software* para implementar el descodificador.

[124] La columna “otros” incluye otras funcionalidades del sistema como pueden ser la tarea de aplicación, la gestión del RTOS, la pila TCP/IP o las comunicaciones entre tareas.

4.4.1.1 Descripción del algoritmo

El descodificador seleccionado es conforme con los perfiles *Baseline* (BP) y *Main* (MP) a excepción de las herramientas ASO (*Arbitrary Slice Ordering*), MSG (*Multiple Slice Groups*) y el formato entrelazado MBAFF (*MacroBlock Adaptative Frame/Field*).

El ordinograma del bucle principal[125] de descodificación de cada NAL (*Network Abstraction Layer*) es el mostrado en la Figura 4-33[126]. El proceso de descodificación comienza identificando el tipo de NAL para proceder a su procesamiento. Si se trata de un SPS (*Sequence Parameter Set*) o de un PPS (*Picture Parameter Set*), se extraen los parámetros asociados a cada uno de ellos que, hasta que sean modificados, serán válidos para todo el proceso de descodificación de la secuencia. Si por el contrario se trata de un NAL que identifica a una rebanada (*slice*) que forma parte de una imagen, se entra en un bucle que se repite para todos los macrobloques que componen dicha rebanada.

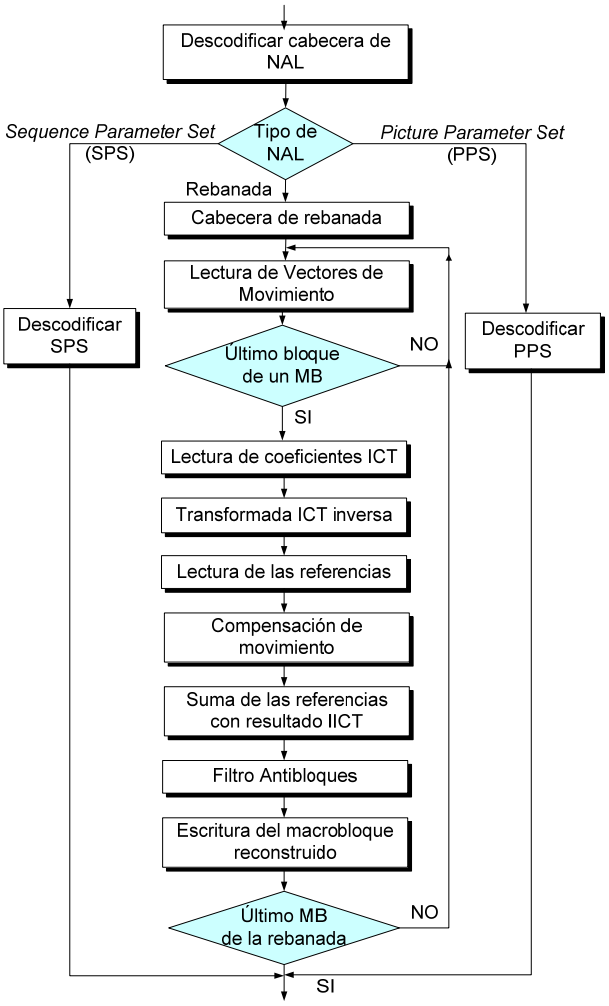


Figura 4-33 Ordinograma del bucle principal de descodificación de cada NAL.

[125] El ordinograma presentado es el empleado para descodificar macrobloques con predicción (macrobloques *inter*). Para macrobloques sin predicción (tipo *intra*) algunas de las etapas del proceso de descodificación (lectura de vectores de movimiento o compensación de movimiento) no se realizan.

[126] A diferencia de como se indicó para MPEG-2 (Figura 4-4) y MPEG-4 (Figura 4-29), en el ordinograma del bucle principal no se han incluido los *buffers* que se emplean puesto que son muy numerosos. En apartados posteriores se irán presentando los *buffers* existentes y su ubicación en memoria.

Dentro de este bucle, y tras analizar la cabecera de rebanada, se obtienen, si el macrobloque tiene predicción (tipo *inter*), los vectores de movimiento de todos los bloques o sub-bloques en los que se encuentre dividido. Posteriormente, se extraen los coeficientes de la ICT de cada bloque de 4×4 píxeles y se realiza la transformada ICT inversa. A continuación se leen los píxeles de referencia de las imágenes anteriores y/o posteriores y se calculan con ellos los bloques de referencia tras aplicar el algoritmo de compensación de movimiento correspondiente. A los píxeles de referencia obtenidos se les suma el resultado de la ICT inversa para obtener el macrobloque reconstruido. Finalmente, si es necesario, se aplica el filtro antibloques y se almacena el resultado en la imagen reconstruida.

Si el macrobloque a descodificar es de tipo *intra*, no existen vectores de movimiento por lo que no es necesario realizar la lectura de los bloques de referencia. Por tanto, tras leer los coeficientes de la ICT y calcular la ICT inversa, es necesario comprobar si se emplea predicción *intra* para obtener la referencia y sumarla al resultado de la IICT. Para finalizar, y al igual que ocurre con los bloques *inter,* se realiza el filtrado antibloques y la escritura del macrobloque reconstruido en la imagen descodificada.

Para facilitar la explicación del modo en que se paralelizan las diferentes etapas que componen el proceso de descodificación de cada macrobloque, éste se ha dividido en las fases que se muestran en la Figura 4-34.

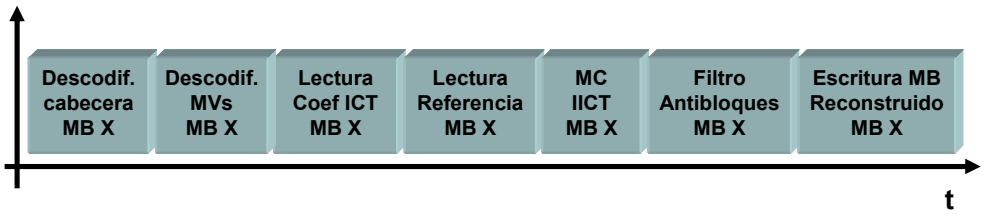


Figura 4-34. Fases en las que se ha dividido el proceso de descodificación de un macrobloque.

4.4.1.2 Migración del descodificador H.264 al entorno de desarrollo de los Procesadores Digitales de Señal

Dado que el descodificador seleccionado procede de la misma librería que la empleada para el estándar MPEG-4, el proceso de migración al entorno de desarrollo del DSP es idéntico al resumido en el apartado 4.3.1.2.

Una vez realizada la migración se ha comprobado la conformidad del código con el estándar empleando para ello el mismo conjunto de secuencias usadas para las pruebas de conformidad en entorno PC. La versión del descodificador obtenida tras la migración se ha empleado como punto de partida para el proceso de optimización para ambos procesadores.

4.4.2 Optimización del tiempo de ejecución del descodificador H.264 para el procesador TMS320DM642

Seguidamente se presentan las técnicas que se han aplicado para reducir el tiempo de ejecución del descodificador H.264 aprovechando las características de la arquitectura del procesador TMS320DM642.

4.4.2.1 Arquitectura de la memoria

Siguiendo las recomendaciones descritas en [ficha 001], se han realizado diferentes repartos de la memoria interna entre memoria de propósito general y memoria caché de nivel 2. Para cada uno de estos repartos se han priorizado las funciones que debían alojarse en la memoria interna siguiendo los criterios propuestos en [ficha 019] y se han realizado medidas de rendimiento. Este proceso se ha repetido cada vez que se han producido optimizaciones significativas en alguno de los módulos funcionales del descodificador H.264.

Finalmente, el reparto en el que se premia la inclusión de *buffers* y funciones en memoria interna es el que minimiza el tiempo medio de descodificación de las imágenes. Por tanto se ha realizado una distribución de la memoria en la que se asignan 64 kB para la caché de nivel 2 y 192 kB para la memoria de propósito general.

Los criterios para realizar el reparto de las funciones y *buffers* en la memoria interna de propósito general fueron expuestos en [ficha 003] y [ficha 005]. Teniendo en cuenta estas recomendaciones se han generado secciones de tamaño igual a la memoria caché de programa de nivel 1 en las que se han asignado tanto fragmentos de código como grupos de *buffers*. Las funciones se han agrupado en secciones teniendo en cuenta su funcionalidad. Las secciones generadas son las siguientes:

1. Funciones relacionadas con la descodificación de entropía. En este caso se han generado dos secciones, una para la descodificación CABAC y otra para la descodificación CAVLC. Dado que ambas no pueden ser empleadas simultáneamente, en memoria interna sólo se aloja una de ellas. Este código ocupa la sección 1 completa y un fragmento de la sección 2.

2. Funciones relacionadas con la algoritmia empleada para transferir los bloques de referencia. Estas funciones son ubicadas en el fragmento sobrante de la sección 2.

3. Funciones relacionadas con la compensación de movimiento. Existen numerosas funciones relacionadas con la compensación de movimiento por lo que es imposible ubicar todas ellas en memoria interna. Por este motivo se han seleccionado las relacionadas con la obtención de los píxeles con resolución de ½ y ¼ píxel. Estas funciones se alojan prácticamente en su totalidad en la sección 3. El resto de funciones se alojan en memoria externa.

4. Funciones relacionadas con el filtrado antibloques. Este filtro está compuesto realmente por tres tipos de funciones, por una parte las funciones relacionadas con la decisión del tipo de filtro, el filtrado de cada uno de los bordes y las copias de datos para dejar preparada la información para el siguiente macrobloque. Estas funciones se alojan en la sección 4 completa y en un fragmento de la 5.

5. Otras funciones de propósito general que tienen una elevada carga computacional se distribuyen entre el fragmento restante de la sección 5 y la sección 6.

Por otro lado, los *buffers* también se han agrupado por su funcionalidad. El criterio aplicado consiste en que los *buffers* que son empleados por las funciones que forman parte de un bloque funcional (descodificación de entropía, compensación de movimiento, filtrado, etc.) se agrupan en una misma sección. Junto a los *buffers* usados por el descodificador, se ha alojado en memoria interna la pila y la *heap*[127] de la tarea que lo integra. La Tabla 4-14 muestra el reparto final de funciones y *buffers* en la memoria interna siguiendo los criterios indicados anteriormente.

[127] La *heap* es una sección que genera el compilador para que la aplicación pueda realizar reservas dinámicas de memoria.

Código/Datos	Nombre	Tamaño	Secciones
Código	CABAC	25 kB	1 y 2
	CAVLC	23 kB	1 y 2
	Filtro Antibloques	16 kB	4
	Compensación de Movimiento + IICT	17 kB	3
	Solicitudes de las referencias	7 kB	2
	Otras funciones	22 kB	5 y 6
Datos	*Buffers* intermedios para compensación de movimiento	6 kB	6
	Buffers para los MBs reconstruidos	2 kB	6
	Buffers para filtrado	8 kB	7
	Buffers auxiliares	2 kB	6
	Pila	4 kB	7
	Heap	27 kB	7 y 8

Tabla 4-14. Reparto de funciones y *buffers* en las secciones en las que se ha dividido la memoria interna de propósito general.

Con este reparto la distribución de funciones y *buffers* en la memoria interna queda como muestra la Figura 4-35. A la vista de la figura se aprecia que el descodificador H.264 emplea 8 secciones de 16 kB cada una lo que da lugar a un total de 128 kB. El resto de la memoria interna (64 kB) no se ha empleado puesto que cuando el descodificador se integra en el sistema completo hay otros fragmentos de código y *buffers* que deben ubicarse en dicha memoria interna.

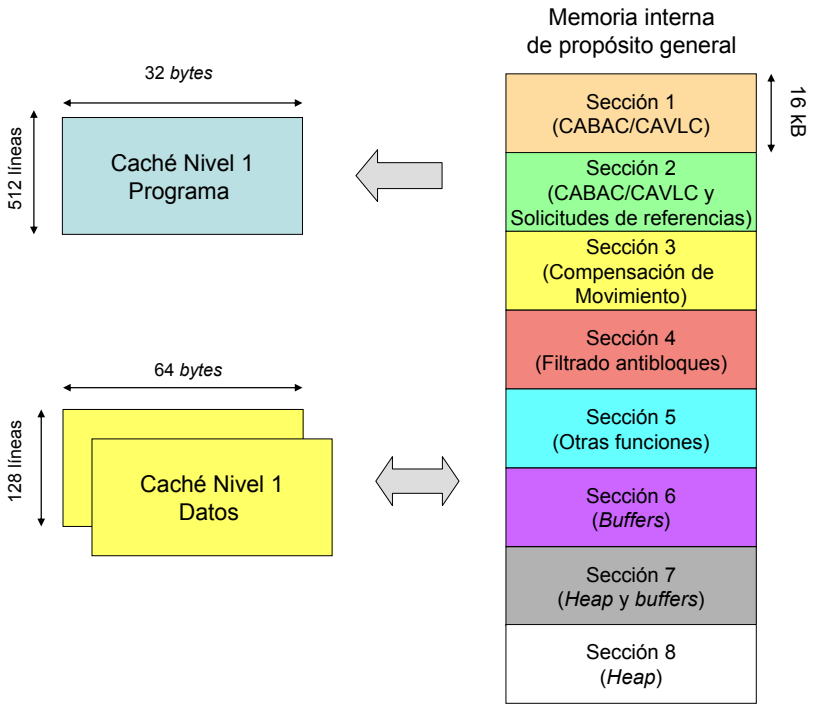


Figura 4-35 Reparto de funciones y *buffers* en la memoria interna de propósito general.

4.4.2.2 Controlador de Acceso Directo a Memoria

En el descodificador H.264 se utilizan las técnicas descritas previamente tanto para MPEG-2 (ver apartado 4.2.2.2), como para MPEG-4 (ver apartado 4.3.2.2) respecto al empleo del controlador de DMA para realizar transferencias de datos entre memoria interna y memoria externa. Todas estas técnicas fueron resumidas en [ficha 008], [ficha 010] y [ficha 011].

Sin embargo, las particularidades del descodificador H.264 respecto a sus predecesores permiten completar algunas de las recomendaciones realizadas para los estándares anteriores tal y como se muestra en los siguientes apartados.

4.4.2.2.1 Tamaño de los *buffers* necesarios

Para realizar las transferencias empleando el controlador de DMA es necesario modificar el bucle de descodificación de macrobloques que se presentó en la Figura 4-33, tal y como se indicó en [ficha 006].

Una vez realizadas las modificaciones, el ordinograma del bucle de descodificación de macrobloques se presenta en la Figura 4-36. A diferencia de lo que ocurría con MPEG-2 y MPEG-4, es posible que un macrobloque esté dividido en bloques de menor tamaño por lo que cada vez que se obtiene un vector de movimiento de alguno de estos bloques se realiza una petición de transferencia para obtener la referencia correspondiente. De este modo, y al igual que ocurría en MPEG-2 y MPEG-4, el movimiento de datos se paraleliza con la extracción de los coeficientes de la ICT y la ICT inversa. Una vez que se calcula la compensación de movimiento y se filtra el macrobloque, éste es almacenado en la imagen reconstruida en memoria externa empleando de nuevo el controlador de DMA.

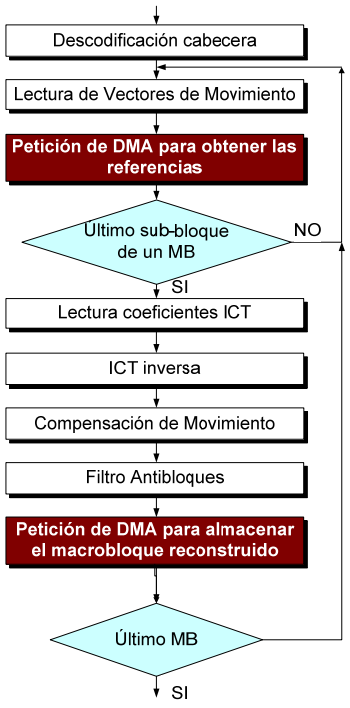
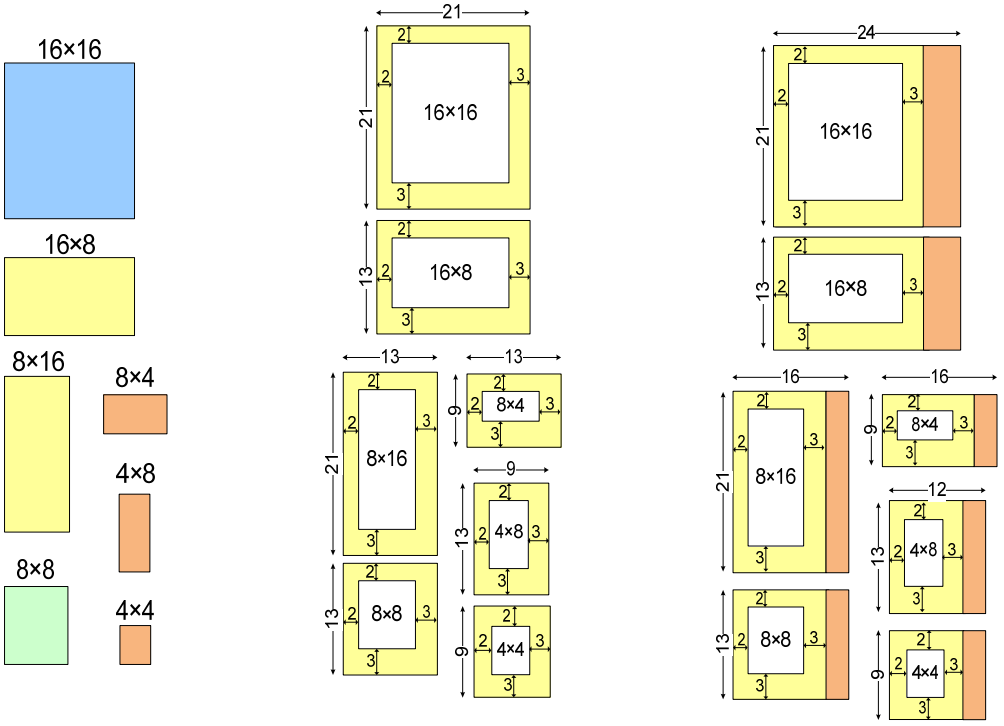


Figura 4-36 Ordinograma modificado para realizar transferencias por DMA en los bloques con predicción.

La Figura 4-37-a muestra los tamaños que pueden tener los bloques que componen un macrobloque. Dado que en H.264 existen macrobloques con compensación de movimiento con precisión de ¼ píxel, es necesario transferir algunos píxeles adicionales tal y como se mostró en la [ficha 020]. Por lo tanto, el tamaño de las transferencias que hay que realizar para cada posible partición es el que se

muestra en la Figura 4-37-b. Finalmente, la Figura 4-37-c presenta los tamaños de las transferencias que es necesario realizar para que éstas estén alineadas a direcciones de memoria múltiplos de 32 bits tal y como se indicó en [ficha 010].



a) Tamaños de los bloques que pueden formar un macrobloque.

b) Tamaño de los bloques incluyendo bordes para la compensación de movimiento.

c) Tamaño de los bloques para asegurar el alineamiento de las transferencias a 32 bits.

Figura 4-37 Tamaño de los diferentes bloques en los que se puede descomponer un macrobloque.

En el estándar H.264 un macrobloque puede dividirse en bloques de menor tamaño, existiendo una gran variedad de posibles combinaciones (ver apartado 2.2.4.2.4.1). Por tanto, si se siguen las indicaciones de [ficha 006], sería necesario definir tantos *buffers* en memoria interna como posibles combinaciones. Esto supondría tener que definir numerosos *buffers* además de complicar sustancialmente la gestión de las transferencias reduciendo el rendimiento del descodificador.

Alternativamente se define un único *buffer* en el que alojar todas las referencias de un macrobloque necesarias para realizar la compensación de movimiento anotando en cada situación el desplazamiento del fragmento transferido respecto al comienzo del *buffer*. Por tanto sólo es necesario definir tres *buffers* intermedios en memoria interna:

1. Ref_Y. Se trata del *buffer* en el que se almacenan las referencias de luminancia asociadas a un macrobloque independientemente de los bloques en los que se encuentre dividido. Análogamente se definen *buffers* equivalentes para las crominancias denominados Ref_Cr y Ref_Cb respectivamente.

2. ICT_COEF. Se trata de un *buffer* de tamaño macrobloque en el que se almacenan los coeficientes de la ICT extraídos de la trama de entrada.

3. REC_i: Se trata de un doble *buffer* en el que se almacenan los macrobloques reconstruidos tras realizar los procesos de ICT inversa y de compensación de movimiento.

El tamaño del *buffer* REF_Y es de 3.4 kB[128] puesto que en el peor caso un macrobloque está dividido en 16 bloques de 4×4 píxeles con predicción bidireccional. Para esta situación el tamaño del *buffer* se calcula empleando la expresión mostrada en la Ec.4-1.

$$\text{Tamaño} = 2\ (\text{referencias}) \cdot 12 \cdot 9\ (\text{tamaño\_bloque}) \cdot 16\ (\text{bloques/MB}) = 3456\ \text{bytes}$$

Ec.4-1

Los diferentes *buffers* implicados en el proceso de descodificación de la luminancia de un macrobloque con predicción y su ubicación en los diferentes niveles de memoria se muestran en la Figura 4-38. Como se aprecia en el *buffer* REF_Y se almacenan varias referencias por lo que es necesario almacenar el desplazamiento de cada uno de los bloques transferidos respecto al comienzo del *buffer*. La gestión de los desplazamientos es sencilla por lo que no afecta al rendimiento del descodificador.

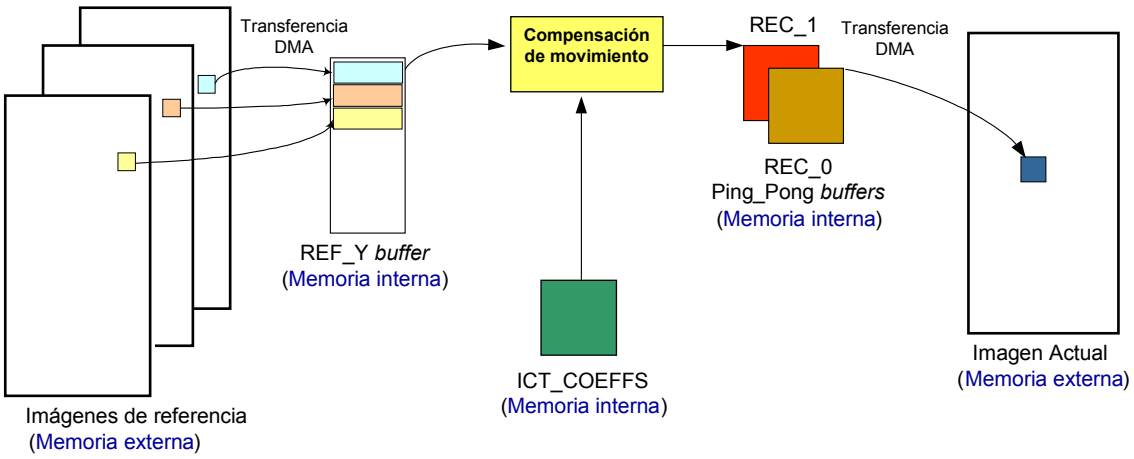


Figura 4-38 Proceso de reconstrucción de un macrobloque con predicción empleando transferencias de DMA para obtener las referencias y almacenar el resultado en la imagen descodificada.

Referencia: Tamaño de las transferencias de DMA para macrobloques en los que existen diferentes particiones [ficha 021].
Descripción: Definir un único *buffer* en memoria interna que permita alojar todas las posibles referencias que pueda tener un macrobloque en el caso en el que éste pueda estar dividido en otros de menor tamaño. Transferir a este *buffer* las referencias, incluyendo los píxeles adicionales necesarios para poder realizar la compensación de movimiento. Almacenar en una estructura interna el desplazamiento de cada uno de los bloques transferidos respecto al comienzo del *buffer*, para posteriormente poder realizar la compensación de movimiento. Guardar el resultado de la compensación de movimiento y la suma del error de predicción en uno de los doble *buffers* destinados a almacenar el macrobloque reconstruido. Finalmente, transferir el resultado a la imagen reconstruida.
Aplicabilidad: A descodificadores en los que un macrobloque pueda descomponerse en diversas combinaciones de bloques de menor tamaño.

[128] Los *buffers* de crominancia deben tener una cuarta parte del tamaño del de luminancia.

4.4.2.2.2 Modificación de la estructura del descodificador

Siguiendo las recomendaciones propuestas en [ficha 009] se ha reorganizando el bucle de descodificación para paralelizar las transferencias de los bloques de referencia de un determinado macrobloque con el comienzo del procesado del siguiente.

Sin embargo, debido al tamaño variable de los macrobloques, es posible que para obtener una referencia se requieran hasta 96[129] transferencias de DMA. Esto provoca que el tiempo que necesita el controlador de DMA para finalizar estas transferencias sea elevado por lo que sigue siendo necesario realizar esperas.

Para que esto no ocurra, la CPU debe seguir ejecutando instrucciones mientras el DMA termina de transferir los datos. Para lograr este objetivo se ha reorganizado el bucle principal de procesamiento para que procese dos macrobloques en paralelo. Por tanto, mientras finalizan las transferencias de DMA que permiten obtener las referencias del macrobloque que se está procesando (MB#X), la CPU realiza el filtrado antibloques del macrobloque anterior (MB#X-1) y su posterior transferencia a la imagen reconstruida. De este modo se eliminan todas las esperas por parte de la CPU. La Figura 4-39 muestra el ordinograma en el que se introduce esta reorganización del bucle.

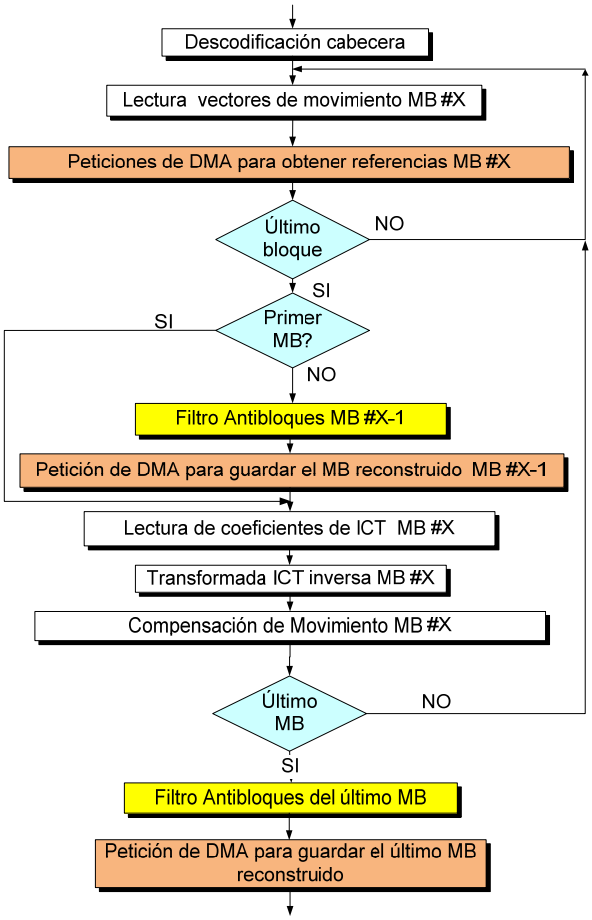


Figura 4-39 Ordinograma del bucle principal de descodificación para procesar dos macrobloques en paralelo y eliminar las esperas por parte de la CPU.

[129] Se trataría de un macrobloque compuesto exclusivamente por bloques de 4×4 píxeles y predicción bidireccional. Esta situación da lugar a 16 bloques/MB × 2 referencias/bloque × 3 componentes.

La Figura 4-40 presenta el diagrama temporal de las fases por las que pasa la descodificación de un macrobloque en el que se aprecia que han desaparecido completamente las esperas de la CPU[130].

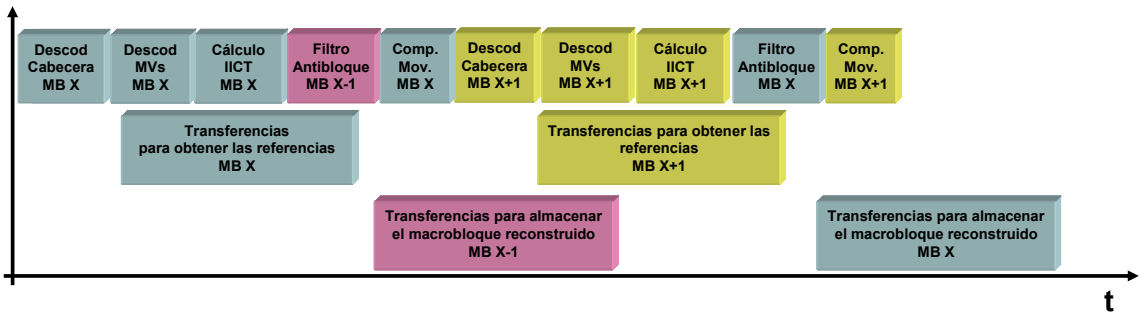


Figura 4-40. Paralelización de las fases por las que pasa la descodificación de un macrobloque empleando el controlador de DMA.

Referencia: Reorganización del bucle principal de descodificación en H.264 [ficha 022].
Descripción: Rellenar los tiempos en los que el controlador está transfiriendo por DMA los píxeles necesarios para poder realizar la compensación de movimiento de un macrobloque con instrucciones que permitan a la CPU continuar el procesamiento de otros datos. Una vez reorganizado el bucle de descodificación, evaluar si en alguna situación la CPU debe esperar a la finalización de alguna transferencia. En este caso se paralelizará la descodificación con alguno de los módulos funcionales del proceso de descodificación del siguiente macrobloque.
Aplicabilidad: Esta técnica es aplicable para cualquier descodificador que se ejecute en un procesador en el que la CPU deba esperar la finalización de la transferencia por DMA de los píxeles necesarios para realizar la compensación de movimiento.

4.4.2.3 Arquitectura SIMD

A diferencia de lo ocurrido con el estándar MPEG-4 en el que fue posible reutilizar algunos módulos optimizados del descodificador MPEG-2, en este caso no ha sido factible ya que ni la transformada al dominio de la frecuencia, ni la codificación de entropía, ni el método de cálculo de los píxeles intermedios cuando se realiza la compensación de movimiento con resolución de ½ o ¼ píxel son iguales. Por tanto todas las funciones que ha sido necesario optimizar han tenido que ser reescritas íntegramente.

Debido al elevado número de funciones que han sido recodificadas, en los siguientes apartados sólo se mostrarán los detalles de aquellas que se han considerado más significativas desde el punto de vista de extracción de metodologías de optimización del tiempo de ejecución.

[130] El único caso en el que es necesario comprobar si ha finalizado la transferencia es cuando se está descodificando un macrobloque de tipo “saltado” (*skipped*) en cuyo caso el procesamiento que hay que realizar no es muy complejo por lo que el controlador de DMA no tiene tiempo suficiente para finalizar las transferencias. En cualquier caso esta situación sólo ocurre en algunos de los macrobloques por lo que tiene un impacto pequeño en el rendimiento global.

4.4.2.3.1 Movimiento de datos asociado al filtro antibloques

El filtro antibloques es uno de los módulos más complejos en el proceso de descodificación llegando a necesitar aproximadamente ⅓ de la carga computacional total del descodificador [HJKH03]. Para realizar el filtrado de un macrobloque son necesarios píxeles pertenecientes a los macrobloques vecinos superior e izquierdo. Más concretamente, y como se muestra en la Figura 4-41, para cada macrobloque de luminancia se emplean cuatro filas del macrobloque superior y cuatro columnas del izquierdo.

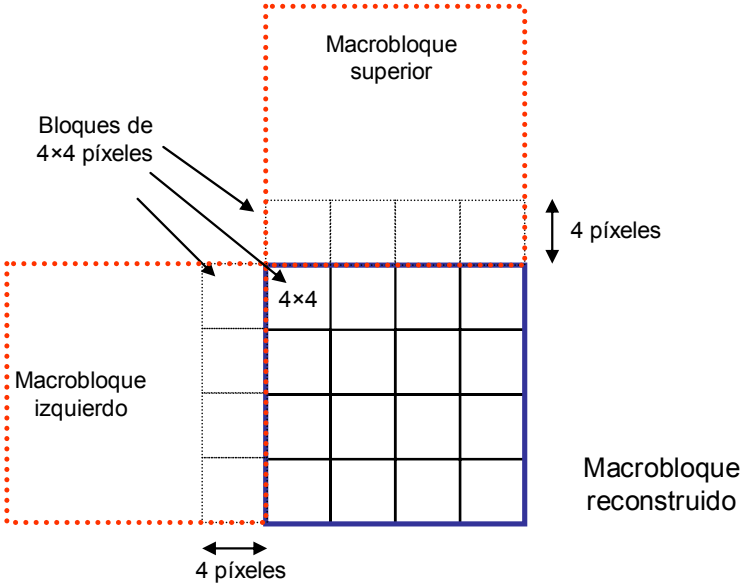


Figura 4-41 Píxeles necesarios para realizar el filtrado de la luminancia de un macrobloque.

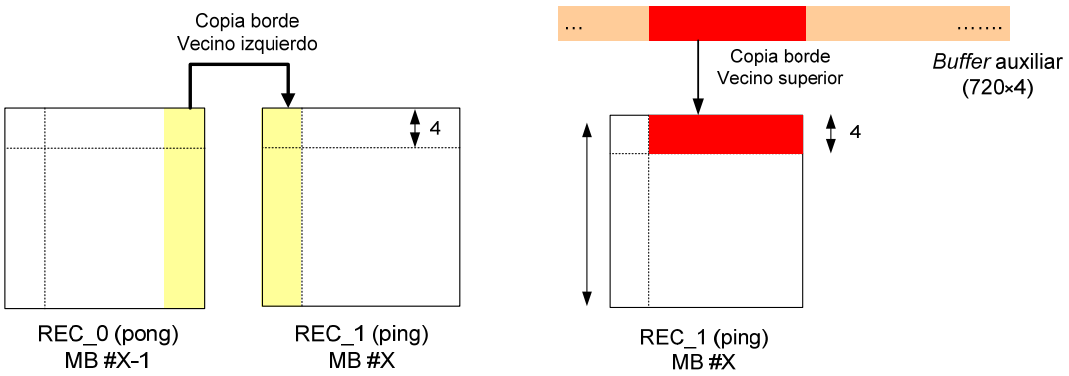
Dado que el macrobloque reconstruido se encuentra en un *buffer* en memoria interna (*buffer* REC_i ver 4.4.2.2), antes de filtrar es necesario acceder a los píxeles de los bloques vecinos de la imagen descodificada actual. Acceder a dichos píxeles reduce el rendimiento del descodificador puesto que la imagen se encuentra almacenada en la memoria externa. Una posible solución a este problema consiste en obtener los píxeles necesarios empleando transferencias de DMA; sin embargo, esta solución implica incrementar el número de peticiones al controlador de DMA con la correspondiente sobrecarga y pérdida de rendimiento.

La opción que se ha tomado consiste en sobredimensionar los *buffers* REC_i de modo que en lugar de tener el tamaño de un macrobloque se les han añadido cuatro filas y cuatro columnas en las que se almacenan los píxeles necesarios para el proceso de filtrado. De este modo, todos los píxeles involucrados se encuentran siempre en memoria interna. Cada *buffer* REC_i debe tener por tanto un tamaño de 20×20 píxeles para luminancia y de 12×12 píxeles para crominancia.

Para obtener la información que debe almacenarse en estas filas y columnas adicionales antes del filtrado del MB#X se ha implementado el siguiente algoritmo para una imagen de tamaño PAL (720×576 píxeles):

- Los píxeles del vecino de la izquierda (MB#X-1) se obtienen del *buffer* gemelo (pong) al de reconstrucción (se trata de un doble *buffer*) puesto que en dicho *buffer* está almacenado el macrobloque que se ha descodificado inmediatamente antes. Dado que es una copia entre datos que se encuentran en memoria interna no supone una penalización importante. La Figura 4-42-a muestra este proceso.

- Los píxeles del vecino superior se obtienen a partir de un *buffer* auxiliar almacenado en memoria interna y de tamaño igual a cuatro veces la anchura de la imagen en el que, para cada macrobloque, se van guardando los píxeles descodificados (y filtrados) de sus 4 filas inferiores. La copia no supone una penalización puesto que ambos *buffers* se encuentran en memoria interna. La Figura 4-42-b muestra el proceso.

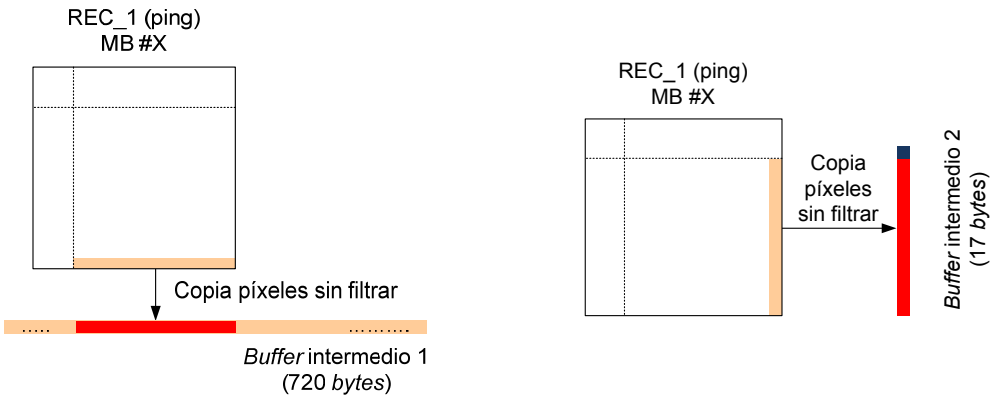


a) Copia entre *buffers* del borde del vecino izquierdo.

b) Copia del borde del vecino superior desde el *buffer* auxiliar.

Figura 4-42 Obtención de los píxeles necesarios para filtrar un macrobloque.

- Antes de realizar el filtrado hay que preservar una serie de píxeles por si los macrobloques vecinos de la derecha y/o de abajo son *intra*. En este caso los píxeles sin filtrar del macrobloque con el que se está trabajando pueden ser empleados para obtener la predicción. Dichos píxeles se preservan en *buffers* intermedios antes de que sean modificados por efecto del filtrado. Estos *buffers* intermedios se almacenan en memoria interna. La Figura 4-43-a y la Figura 4-43-b muestran este proceso de copia[131].



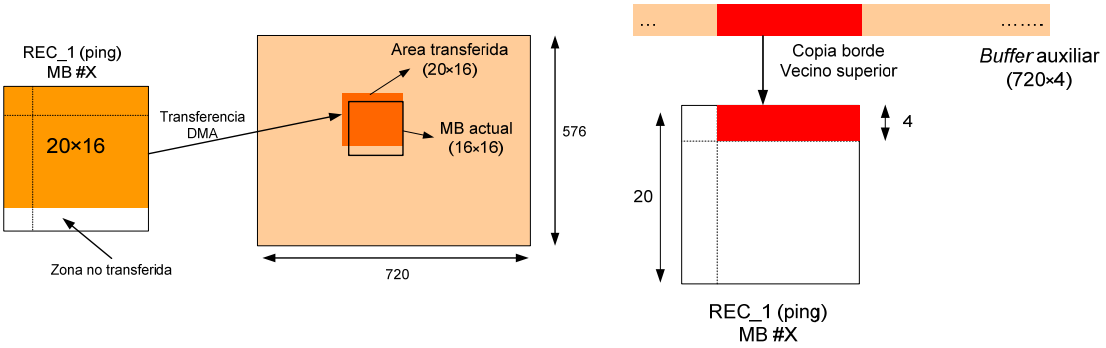
a) Copia del borde inferior del macrobloque sin filtrar a un *buffer* intermedio.

b) Copia del borde derecho del macrobloque sin filtrar a un *buffer* intermedio.

Figura 4-43 Píxeles sin filtrar preservados para la predicción *intra* de los macrobloques vecinos.

[131] El *byte* número 17 que aparece en la Figura 4-43-b no se copia desde el macrobloque que se está tratando si no que proviene del *buffer* intermedio de 720 píxeles que contiene los píxeles de la línea inferior sin filtrar.

- Finalmente se realiza el filtrado de todos los bordes de 4×4 píxeles para obtener el macrobloque reconstruido. Este proceso se muestra en la Figura 4-44-a en la que se aprecia que las 4 últimas filas del macrobloque no se transfieren a la imagen reconstruida puesto que serán modificadas por el macrobloque vecino inferior. Estas cuatro filas se almacenan en el *buffer* auxiliar como se muestra en la Figura 4-44-b. No es necesario almacenar todos los píxeles de esas cuatro últimas filas puesto que las cuatro últimas columnas de esa tira serán almacenadas al procesar el siguiente macrobloque de la tira[132].



a) Copia del macrobloque reconstruido sobre la imagen actual.

b) Copia del borde inferior sobre un *buffer* auxiliar.

Figura 4-44 Transferencia del macrobloque reconstruido y almacenamiento de los píxeles para la tira de macrobloques inferior.

Referencia: Gestión de *buffers* intermedios para almacenar la información necesaria para la descodificación de macrobloques vecinos [ficha 023].
Descripción: Definir *buffers* intermedios en los que poder almacenar temporalmente los píxeles de macrobloques vecinos que son necesarios para la descodificación. La inclusión de estos *buffers* en memoria interna agiliza las copias de datos a/desde ellos. En el caso de tener que definir *buffers* de elevado tamaño puede ser necesario desalojar algunas funciones de la memoria interna. Antes de consolidar estos cambios es necesario evaluar de nuevo el rendimiento del descodificador para confirmar que se produce una mejora.
Aplicabilidad: Aplicable a cualquier descodificador que realice el procesamiento de la imagen a nivel de macrobloque y en el que la descodificación de uno de ellos emplee píxeles pertenecientes a macrobloques vecinos.

4.4.2.3.2 Algoritmo del filtro antibloques

El proceso de filtrado se descompone en dos partes: en primer lugar se determina si es necesario realizar el filtrado y la fuerza con la que éste debe realizarse y, en segundo lugar, se realiza el filtrado propiamente dicho.

[132] El proceso descrito tiene varias excepciones que se corresponden con el primer macrobloque de una tira en el que no existe vecino de la izquierda, el último macrobloque de una tira puesto que no posee vecino derecho y la primera y última tiras de macrobloques de la imagen puesto que no tienen vecinos superiores e inferiores respectivamente.

Las modificaciones que se proponen en este apartado afectan a la segunda parte descrita en el párrafo anterior. La Figura 4-45 muestra los píxeles involucrados en el filtrado horizontal entre dos bloques de 4×4 píxeles de luminancia. Como se aprecia, este proceso involucra a todas las muestras tanto del bloque que se está procesando como del bloque vecino superior aunque como resultado, sólo los píxeles coloreados en azul y amarillo pueden ser alterados.

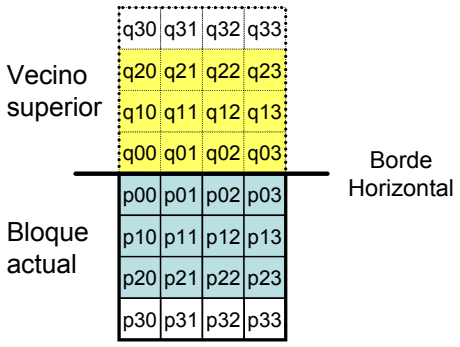


Figura 4-45 Píxeles involucrados en el filtrado de un borde horizontal.

El diagrama de flujo asociado al filtrado de un borde horizontal correspondiente a un bloque de luminancia[133] se muestra en la Figura 4-46. En este ordinograma se aprecia que el procesamiento no es muy regular puesto que es necesario evaluar varias condiciones, lo que dificulta la paralelización del código generado por parte del compilador. Las condiciones que aparecen en el ordinograma dependen de tres parámetros: la fuerza del filtro (*Boundary Strength* o BS) y los umbrales, α y β, que derivan fundamentalmente del parámetro de cuantificación de los dos bloques involucrados y que, indirectamente, indican las muestras que se modifican.

A partir de los tres parámetros involucrados se decide si se debe filtrar o no el borde, y en caso afirmativo, la expresión que debe emplearse para calcular las muestras modificadas. Por ejemplo, la muestra en la posición p0X (X=0…3) puede ser calculada empleando 4 expresiones diferentes indicadas como $P0X_0$, $P0X_1$, $P0X_2$ y $P0X_3$ dependiendo de las condiciones cond1, cond4 y cond5.

La existencia de tantas condiciones dificulta la tarea del compilador a la hora de generar el código en ensamblador. Para evitar estas condiciones se ha empleado el método de optimización propuesto en [YGLZ06] que se basa en realizar siempre todos los cálculos posibles para finalmente asignar a las variables de salida los valores adecuados en función de las condiciones que se cumplan en cada caso. Aplicado al filtro antibloques, este método se puede dividir en cinco pasos:

1. Todos los píxeles necesarios para filtrar el bloque de 4×4 completo son leídos de tal modo que con una única instrucción se leen 32 bits que se corresponden con 4 píxeles contiguos (por ejemplo q30, q31, q32 y q33). Empleando 8 variables se almacenan los 32 píxeles implicados en el proceso.

2. Se pre-calculan las seis condiciones usadas en el algoritmo. Para facilitar la extracción de estas condiciones se emplean instrucciones que son capaces de realizar 4 comparaciones y restas simultáneamente. Finalmente, las condiciones de 1 bit extraídas se convierten en máscaras de 8 bits empleando las instrucciones de extensión de bits.

[133] El algoritmo es análogo para el filtrado vertical y para el filtrado de las crominancias.

3. Se calculan todas las posibles salidas del filtro para cada píxel involucrado (por ejemplo, para el píxel p0X se calculan $P0X_0$, $P0X_1$, $P0X_2$ y $P0X_3$) aunque sólo uno de ellas será usada posteriormente. Aunque pueda parecer lo contrario, este proceso no es ineficiente puesto que todos los cálculos pueden paralelizarse. Dado que los resultados de las operaciones intermedias requieren 16 bits es necesario emplear instrucciones de empaquetado y desempaquetado para pasar datos de 8 bits a 16 bits y viceversa (ver ficha [015]).

4. Seguidamente las salidas del filtro son combinadas con las máscaras para asignar a los píxeles el valor adecuado en función de las condiciones que se pre-calcularon en el paso 2.

5. Finalmente, las 6 variables de 32 bits que contienen los 24 píxeles que han podido ser modificados se almacenan en el *buffer* del macrobloque descodificado que posteriormente será transferido a la imagen reconstruida.

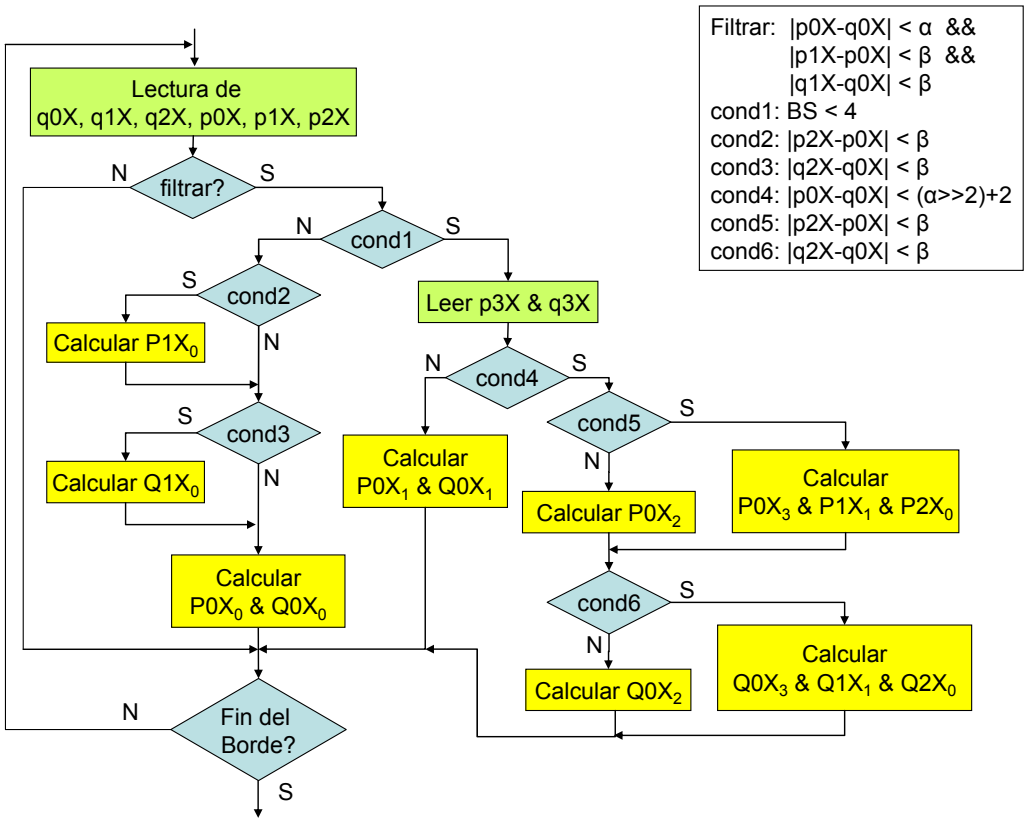


Figura 4-46 Ordinograma del filtrado horizontal de un bloque de 4×4 píxeles.

La Figura 4-47 muestra un diagrama de flujo que resume el algoritmo optimizado. Como se aprecia no existen saltos ni condiciones y todos los píxeles de un bloque de 4×4 píxeles son calculados en paralelo. Además, el número de ciclos que emplea la CPU en ejecutar este algoritmo es constante debido a que no hay condiciones que evaluar[134]. El mismo método se puede aplicar a los bordes verticales y los bloques de crominancia.

[134] Esta afirmación es cierta siempre y cuando los datos a procesar se encuentran en la memoria interna de propósito general. En el caso de que se encontraran en memoria caché de nivel 1, el tiempo de ejecución se reduciría puesto que la CPU no necesitaría transferirlos.

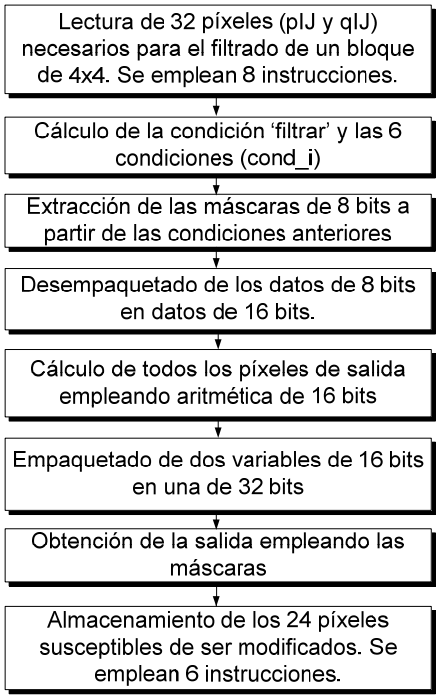


Figura 4-47 Algoritmo optimizado empleado para realizar el filtrado antibloques sin necesidad de realizar saltos condicionales asociados a las condiciones que se cumplan en cada ejecución.

Referencia: Eliminación de condiciones en algoritmos de cálculo de píxeles [ficha 024].
Descripción: Calcular todas aquellas condiciones que aparezcan en el algoritmo. A partir de las condiciones generar para cada una de ellas máscaras de bits. Realizar todos los posibles cálculos independientemente de las condiciones que se cumplan. Asignar a las variables de salida los datos correctos aplicando las máscaras que se han calculado previamente.
Aplicabilidad: A algoritmos en los que se realicen cálculos intensivos con píxeles en los que las operaciones dependen de una serie de condiciones que se van evaluando a lo largo de la función.

4.4.2.3.3 CABAC

La función que realiza el núcleo de la descodificación aritmética de entropía adaptativa con el contexto (CABAC) posee dos características que la hacen especialmente interesante desde el punto de vista de la optimización. Por una parte, el descodificador realiza un elevado número de llamadas a esta función, lo que supone que su carga computacional total sea muy elevada; y por otra, posee un tamaño muy reducido lo que permite plantearse su escritura directamente en lenguaje ensamblador haciendo la paralelización manual de las instrucciones.

Así, para un conjunto de secuencias en formato PAL y con una tasa binaria media de 2 Mbps, esta función se ejecuta aproximadamente 29000 veces por imagen[135] en media lo que supone en torno al 16% de la carga computacional

[135] Esta información se ha obtenido calculando la media de llamadas por imagen a la función que realiza la descodificación de entropía para varias secuencias de vídeo.

demandada por la CPU. El tiempo medio de ejecución de esta función sin optimizar en lenguaje C es de 167 ciclos de reloj. Tras recodificar la función se ha logrado reducir su tiempo de ejecución a 22 ciclos de reloj[136] lo que supone un 2% del total de la carga computacional.

Referencia: Codificación de funciones en ensamblador paralelizado manualmente [ficha 025].
Descripción: Identificar funciones de tamaño reducido y elevada carga computacional. Recodificar la función inicialmente en ensamblador lineal para comprobar su funcionalidad. Evaluar la mejora en el rendimiento y analizar el nivel de paralelización de las instrucciones que logra el compilador. En función de los resultados obtenidos, valorar la opción de paralelizar las instrucciones manualmente para mejorar el rendimiento. Durante todo el proceso se debe tener en cuenta que es posible que la sobrecarga asociada a la llamada a la función optimizada pueda reducir la mejora esperada.
Aplicabilidad: A funciones de tamaño reducido que sean ejecutadas en múltiples ocasiones y que, por tanto, supongan una elevada carga computacional.

4.4.2.3.4 Compensación de movimiento con precisión de ¼ píxel

Para obtener los píxeles intermedios para los bloques con predicción de ¼ píxel es necesario aplicar un filtro de 6 etapas (ver apartado 2.2.4.2.4.2) lo que implica tener que realizar 6 lecturas de píxeles, 6 productos, 6 sumas, un desplazamiento y un almacenamiento para cada píxel interpolado[137]. Por tanto, para un bloque de 8×8 píxeles, serían necesarias 384 lecturas de píxeles, 384 sumas y productos, 64 desplazamientos y 64 almacenamientos.

El número de operaciones se reduce puesto que el compilador emplea las instrucciones que aprovechan la arquitectura SIMD del procesador. A pesar de la mejora lograda por el compilador, ésta puede incrementarse sustancialmente si se tiene en cuenta la organización de los datos en la memoria y se recodifica el algoritmo utilizando instrucciones en ensamblador.

La Figura 4-48 muestra los píxeles necesarios para calcular los valores intermedios para un bloque de 8×8 píxeles en el que el vector de movimiento tiene una resolución fraccionaria en el eje horizontal[138]. Para calcular los píxeles intermedios de

[136] En esta situación se ha comprobado que el tiempo invertido en la llamada y retorno de la función por parte del descodificador es comparable a su tiempo de ejecución. Esta es una situación no deseable que, sin embargo, no puede ser resuelta puesto que las herramientas de desarrollo para el entorno del DSP impiden que las funciones escritas directamente en ensamblador sean convertidas en funciones *inline* (funciones que se incrustan directamente en el lugar en el que se producen las llamadas eliminando de este modo la sobrecarga asociada).

[137] Este número de operaciones se corresponde con el cálculo de los píxeles intermedios cuando el vector de movimiento tiene sólo una de sus dos componentes (horizontal o vertical) con resolución fraccionaria. En el caso de que ambas componentes tengan resolución fraccionaria el número de operaciones se incrementa puesto que es necesario realizar el filtrado en dos ocasiones, una para cada componente del vector de movimiento.

[138] Si la componente fraccionaria es la vertical el procesamiento es el mismo con la salvedad que los píxeles no pueden ser leídos directamente puesto que no se encuentran en posiciones consecutivas de la memoria.

cada una de las filas son necesarios 13 píxeles que, como se indicó en la [ficha 014], pueden ser leídos empleando 4 instrucciones de carga de 4 *bytes* simultáneos[139].

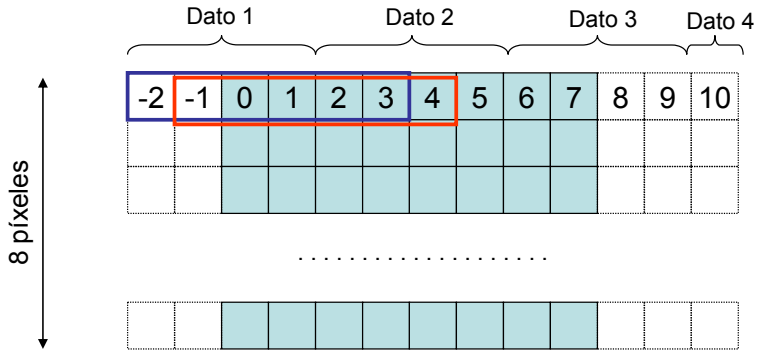


Figura 4-48 Algoritmo optimizado empleado para calcular los píxeles intermedios cuando el vector de movimiento tiene una componente horizontal con resolución fraccionaria.

Para calcular el primer píxel interpolado se emplean los que se encuentran enmarcados en un cuadro azul en la figura, y que han sido almacenados en dos variables de 32 bits denominadas Dato1 y Dato 2. Cada uno de los píxeles involucrados debe ser multiplicado por un coeficiente y el resultado de estos productos es acumulado y dividido por 32. Estas operaciones se pueden realizar de forma optimizada empleando las instrucciones de multiplicación acumulada que posee el procesador, de modo que con dos instrucciones de multiplicación, una de suma (del resultado de las dos multiplicaciones) y un desplazamiento se logra el píxel interpolado.

Para obtener el segundo de los píxeles interpolados se aplica el mismo procedimiento utilizando los píxeles que están enmarcados por la línea roja. Como se aprecia se emplean los mismos datos (Dato 1 y Dato 2) que para el primero de los píxeles por lo que no debe realizarse ninguna nueva lectura.

Los resultados de las multiplicaciones y acumulaciones de 4 *bytes* requieren 16 bits para su almacenamiento por lo que es posible, empleando las instrucciones de empaquetamiento tal y como se indicó en [ficha 015], realizar las operaciones de acumulación de los resultados de las multiplicaciones y desplazamiento con dos píxeles intermedios simultáneamente.

Finalmente, y empleando de nuevo instrucciones de empaquetado, es posible agrupar 4 píxeles intermedios para almacenarlos con una única instrucción de escritura tal y como se indicó en [ficha 014].

El algoritmo implementado queda por tanto como se muestra en la Figura 4-49 en la que se aprecia cómo, a partir de los datos Dato 1 y Dato 2, se calculan los productos/acumulaciones (_dotpsu4), empaquetamientos (_pack2), sumas (_add2) y desplazamientos (_shr2) necesarios para obtener los píxeles interpolados I0 e I1. Estos dos píxeles se empaquetan (_pack4) con I2 e I3 que han sido obtenidos de forma análoga para almacenarlos con una única instrucción de escritura. De este modo, el cálculo de dos píxeles interpolados requiere 8 instrucciones en lugar de las 26 empleadas antes de la modificación del algoritmo.

[139] Realmente son necesarias 3 lecturas de 4 *bytes* simultáneos y una de un solo *byte*. Sin embargo, por regularidad del algoritmo, se realizan 4 lecturas de 4 *bytes* a pesar de que se obtienen píxeles que no son necesarios.

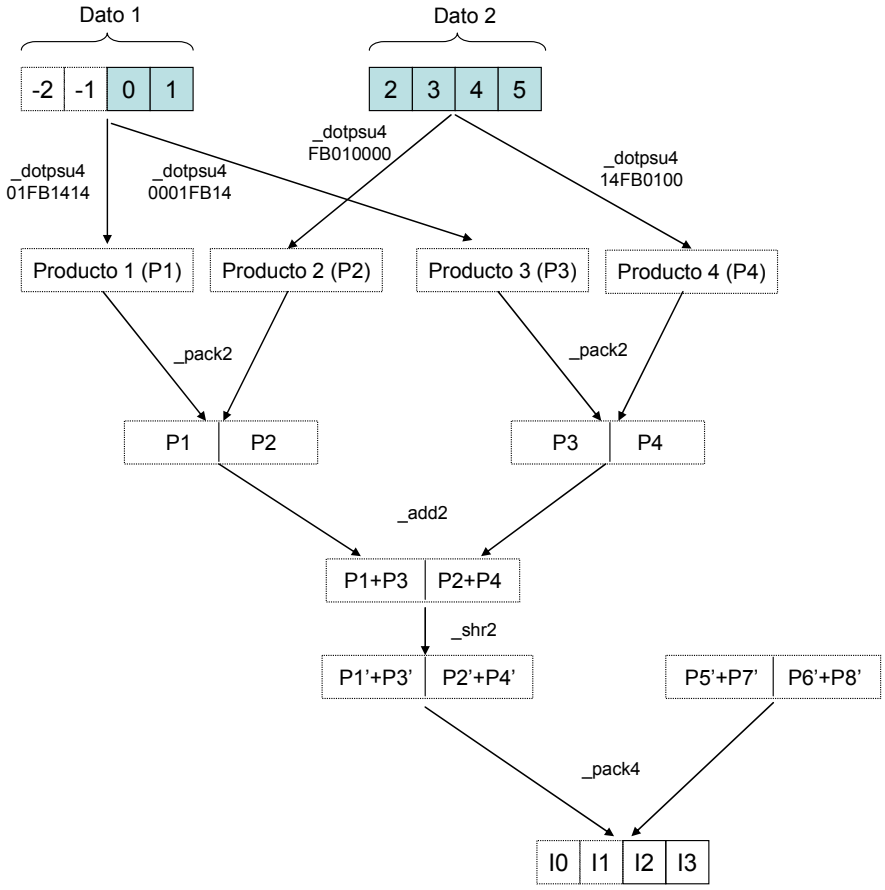


Figura 4-49 Cálculo de los píxeles intermedios de una fila para un vector de movimiento con resolución de ¼ pixel en la componente horizontal.

Referencia: Empleo de instrucciones de multiplicación/acumulación y desplazamiento para operaciones de interpolación [ficha 026].
Descripción: Identificar algoritmos en los que se realice el filtrado de varios píxeles. Obtener los píxeles necesarios para poder realizar el filtrado y almacenar varios en una variable de 32 bits (ver [ficha 014]). Emplear las instrucciones de multiplicación con acumulación para calcular la salida del filtro. Analizar si el número de bits empleados por los datos obtenidos previamente permite empaquetar varios en variables de 32 bits. Emplear instrucciones de suma y/o desplazamiento para operar con los datos almacenados en estas variables (ver [ficha 015]). Empaquetar los datos obtenidos antes de almacenarlos.
Aplicabilidad: A algoritmos en los que se realiza la interpolación de los píxeles de un bloque.

4.4.2.3.5 Otras funciones optimizadas dentro del código del descodificador

Durante el proceso de optimización del descodificador H.264 se han optimizado muchas otras funciones entre las que destacan la descodificación de entropía basada en CAVLC, la ICT inversa y la suma del error de predicción. Las técnicas aplicadas en todos los casos son similares a las propuestas en [ficha 014], [ficha 015] y [ficha 026] por lo que no se ha considerado necesario describirlas con más detalle.

4.4.3 Optimización a nivel de sistema del descodificador H.264

El elevado número de transferencias de DMA que realiza el descodificador H.264 influye en el rendimiento del resto de tareas que forman el sistema de descodificación de televisión digital (ver Anexo B), por este motivo es necesario realizar un análisis de las peticiones de transferencias de DMA que solicitan cada una de las tareas que componen el sistema para distribuirlas lo más equitativamente posible entre las colas de petición que posee el controlador de DMA (para mas detalles consultar la [ficha 017]).

El mayor rendimiento se ha obtenido cuando el descodificador emplea las colas de prioridad alta y media, el analizador de red y la presentación de vídeo la de prioridad baja, quedando la de prioridad urgente para las peticiones que se realizan por fallos en la caché.

Como se mostrará en el apartado 4.4.4.2, el rendimiento del descodificador H.264 es en media próximo al 100% de la CPU. Realizando un análisis del rendimiento imagen a imagen se observa que para descodificar algunas de ellas se emplea un tiempo mayor que el tiempo de presentación (40 ms). Esta situación provoca que cuando la tarea de presentación demanda una de esas imágenes, esta aún no se encuentra disponible por lo que no puede ser mostrada, siendo necesario repetir la imagen descodificada previamente.

La Figura 4-50 muestra un diagrama temporal en el que se pone de manifiesto el problema presentado. El controlador de vídeo del DSP (*videoport*) posee un doble *buffer* para que el descodificador pueda copiar una imagen descodificada en uno de ellos, mientras el controlador se encarga de mostrar la imagen almacenada en el otro. Las líneas verticales de la figura representan los instantes en los que una imagen debe ser presentada por parte del controlador de vídeo. Estas líneas se encuentran separadas 40 ms, que es el tiempo que posee el descodificador para procesar una imagen y copiarla en uno de los *buffers* del controlador. Así, durante el periodo de tiempo T1 se descodifica la imagen 1 que será copiada en el *buffer 1* para presentarse en el periodo T2. En el periodo T3, los procesos de descodificación y copia requieren más de 40 ms por lo que en T4 es necesario repetir la imagen descodificada en T2[140] en lugar de la imagen 3 como correspondería.

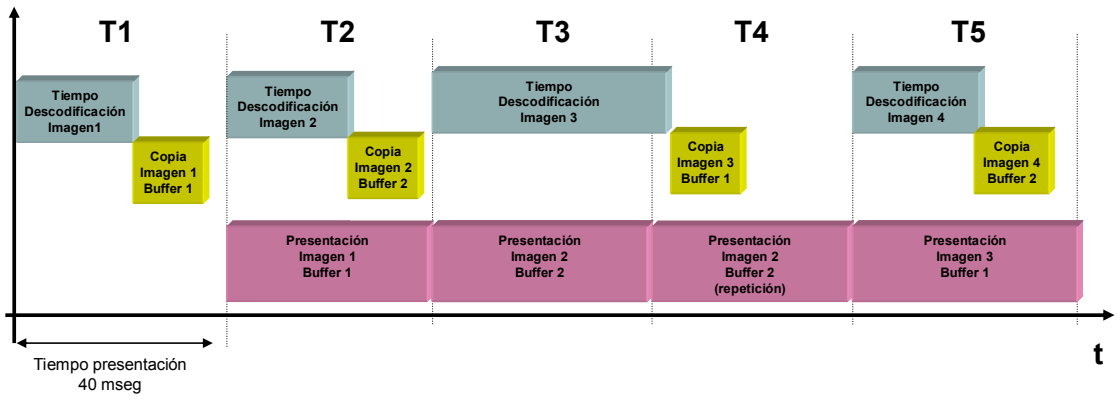


Figura 4-50 Diagrama de tiempos del proceso de descodificación y presentación de imágenes cuando el tiempo de procesamiento de alguna de ellas es superior al tiempo de presentación.

[140] Este problema podría haberse manifestado tanto en MPEG-2 como en MPEG-4. Sin embargo, debido a la carga computacional requerida por estos estándares para las secuencias de pruebas empleadas, no se detectó en ningún caso esta situación.

Para analizar el efecto que provoca a nivel de sistema esta situación es necesario recordar (ver apartado 8.2 del Anexo B) que el STB-IP almacena imágenes sin descodificar en un *buffer* intermedio para que el descodificador de vídeo las procese. Por cada imagen que es mostrada por la tarea de presentación, el descodificador procede a procesar la siguiente imagen que se encuentra en el *buffer* intermedio. Dado que se reciben 25 imágenes por segundo, el hecho de no poder presentar una de ellas provoca que se produzca el retardo de un tiempo de presentación y una imagen se acumule en el *buffer* intermedio. Si se van acumulando imágenes llega un momento en el que el *buffer* se desborda provocando el incorrecto funcionamiento del sistema.

Para solventar este problema se ha aumentado el número de *buffers* de imágenes descodificadas y se ha implementado un algoritmo de gestión de los mismos de forma que el descodificador va almacenando imágenes ya procesadas para que la tarea que se encarga de presentarlas las vaya mostrando con un determinado retraso. La Figura 4-51 muestra el caso en el que se almacenan tres imágenes. En esta situación, el descodificador dispone de dos tiempos de presentación para descodificar dos imágenes consecutivas, por lo que si una de ellas requiere más tiempo de descodificación que un tiempo de presentación (por ejemplo la imagen 3), puede compensarse con el tiempo empleado por la siguiente (imagen 4). De forma general, si se tienen *n* posiciones en el *buffer* de imágenes descodificadas, es necesario que el descodificador nunca emplee más de *n-1* veces el tiempo de presentación en procesar *n-1* imágenes consecutivas.

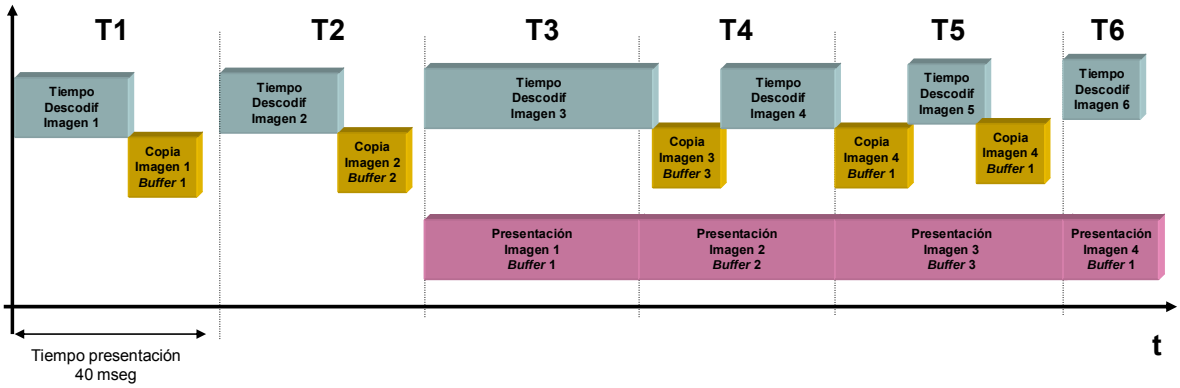


Figura 4-51 Diagrama de tiempos del proceso de descodificación y presentación empleando 3 *buffers* para almacenar imágenes ya descodificadas.

Experimentalmente se ha comprobado que para las secuencias empleadas para realizar las pruebas del descodificador integrado en un sistema de descodificación de vídeo (ver apartado 4.4.4.2), es suficiente con almacenar 4 imágenes descodificadas puesto que en ningún caso se invierten más de 100 ms (4 tiempos de presentación) en descodificar 4 imágenes consecutivas[141].

Cuantas más imágenes se almacenen, mayor será el retardo del descodificador y mayor la necesidad de memoria por lo que hay que dimensionar adecuadamente el número de *buffers* necesario.

[141] A pesar de los resultados experimentales obtenidos con las secuencias de test es posible que con otras sea necesario almacenar más imágenes descodificadas.

Referencia: Resolución de problemas de funcionamiento en tiempo real cuando el tiempo de descodificación de algunas imágenes supera el intervalo de presentación [ficha 027].
Descripción: Cuando la carga computacional media del descodificador es próxima al 100% de la CPU, no es suficiente con asegurar que es menor al 100%, sino que es necesario analizar el tiempo de descodificación de cada una de las imágenes. Si el tiempo de descodificación de alguna de ellas supera el intervalo de presentación de una imagen, es necesario definir un *buffer* de imágenes descodificadas en el que almacenar las imágenes ya procesadas. La inclusión de este *buffer* provoca una latencia en la presentación proporcional al número de imágenes almacenadas. El tamaño del *buffer* debe ajustarse para garantizar que el tiempo que se invierte en descodificar el número de imágenes que se pueden alojar en él sea siempre menor que el tiempo de presentación de todas ellas.
Aplicabilidad: Sistemas en los que el tiempo de descodificación de algunas imágenes sea superior al intervalo de presentación.

4.4.4 Resultados de la optimización para el procesador TMS320DM642

En este apartado se muestran los resultados obtenidos tras aplicar al descodificador H.264, las técnicas de optimización que se han descrito en los apartados anteriores. Para obtener los resultados se han empleado los tres bancos de test descritos en el apartado 4.1.2.

4.4.4.1 Rendimiento del descodificador H.264 aislado

Las medidas del rendimiento del descodificador se han realizado empleando el banco de pruebas mostrado en la Figura 4-1 y que para mayor claridad se replica en la Figura 4-52. Este banco está formado por una tarea que lee la trama codificada en H.264 de un fichero y se la facilita al descodificador mediante un *buffer* intermedio. La trama es descodificada y almacenada en un nuevo fichero.

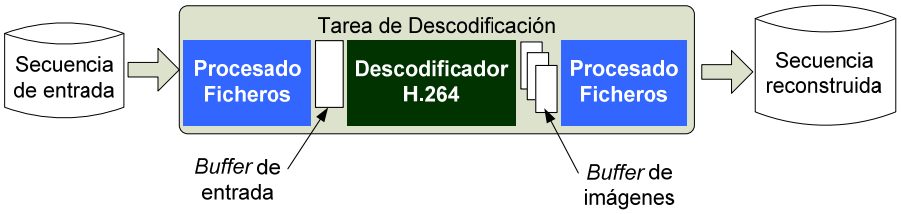


Figura 4-52. Banco de pruebas para las medidas de rendimiento del descodificador H.264 aislado.

Para las pruebas se han empleado secuencias previamente codificadas utilizando el *software* de referencia. Los datos que se presentan en este apartado se corresponden con dos secuencias provenientes de DVD (“Buscando a Nemo” y “La Guerra de las Galaxias. Episodio I”) y una emisión vía satélite de un partido de futbol codificado en formato H.264.

Estas secuencias poseen una resolución de 720×576 píxeles con 25 imágenes por segundo[142] y –las procedentes de DVD– se han codificado con los perfiles

[142] Para realizar las medidas se han descodificado 100 imágenes puesto que la herramienta de análisis de rendimiento del entorno de desarrollo del DSP impide obtener datos para simulaciones excesivamente largas.

Baseline y *Main* empleando la herramienta VLC [VLC]. La tasa binaria empleada es de 2 Mbps[143] (excepto la secuencia Nemo1M que está codificada con 1 Mbps).

Las medidas de rendimiento iniciales realizadas antes de la optimización muestran que, para secuencias de vídeo con perfil *Main* (MP) y nivel 3, con un régimen binario medio de 2 Mbps, el procesador emplea una media de 10^8 ciclos de reloj para descodificar cada imagen. Por tanto, funcionando a 600 MHz, el procesador es capaz de descodificar aproximadamente 0.25 imágenes por segundo.

Una vez aplicadas las técnicas de optimización descritas en los apartados anteriores se han realizado nuevas medidas de rendimiento. La Tabla 4-15 muestra en su primera fila los ciclos de reloj medios empleados para descodificar cada una de las imágenes que componen las secuencias. Las siguientes filas indican el número de ciclos de reloj invertidos en cada uno de los bloques funcionales en los que se descompone el descodificador. Finalmente, las dos últimas filas presentan el número de imágenes por segundo que se pueden descodificar con el procesador funcionando con una frecuencia de reloj de 600 y 720 MHz respectivamente. Analizando estos resultados se aprecia que se obtiene el funcionamiento en tiempo real para todas las secuencias con ambas frecuencias de funcionamiento.

# ciclos×10^6	Nemo		Guerra Galaxias		Fútbol		Nemo1M	
	BP	MP	BP	MP	BP	MP	BP	MP
Descodificador	16.6	22.1	16.0	21.7	16,2	21.5	13,2	20.4
CAVLC/CABAC	4.9	8.2	4.7	7.8	4,9	7.6	3.9	7.0
IICT+MC	5.1	6.7	5.1	6.8	5.1	6.7	4.3	6.0
Filtro Antibloques	4.2	4.4	3.9	4.2	3,9	4.2	3.2	4.6
Otras	2.4	2.8	2.3	2.9	2,3	2.9	1,8	2.8
fps @720 MHz	43.4	32.6	45.0	33.2	44.4	33.5	54.6	35.3
fps @600 MHz	36.1	27.1	37.5	27.7	37.0	27.9	45.5	29.4

Tabla 4-15. Ciclos de reloj invertidos por el descodificador H.264 optimizado y número de imágenes por segundo descodificadas con el TMS320DM642 funcionando a 600 MHz y a 720 MHz.

4.4.4.2 Rendimiento del descodificador H.264 integrado en un entorno de descodificación completo

El descodificador H.264 ha sido integrado en el STB-IP (ver Anexo B) para analizar sus prestaciones cuando interacciona con el resto de tareas que componen el sistema completo.

El banco de test fue mostrado en la Figura 4-2 y está formado por una aplicación de PC [VLC] que transmite a través de la red local tramas de transporte MPEG-2 que contienen tramas elementales H.264 y tramas de audio en formato MPEG-1 Capa II. El STB-IP extrae las tramas elementales, descodifica las secuencias de vídeo y audio y las presenta en un monitor de televisión.

Para realizar las medidas de rendimiento en tiempo real de las diversas tareas se han empleado las mismas secuencias que se han descrito en el apartado 4.4.4.1.

[143] Las secuencias con perfil *Baseline* se han generado con un 5% de imágenes tipo I y un 95% de tipo P. Las secuencias con el perfil *Main* se han creado con un 4% de imágenes I, un 48% de imágenes P y un 48% de imágenes B. Además, las secuencias del perfil *Main* se han codificado empleando CABAC.

El porcentaje de uso de la CPU invertido para cada una de las tareas que componen el sistema se muestra en la Tabla 4-16. Las 4 primeras columnas se corresponden con secuencias codificadas con el perfil *Baseline* y procesadas con la tarjeta DESCOS, las otras 4 columnas muestran los resultados para el perfil *Main* utilizando la tarjeta de desarrollo comercial[144]. Cada una de las filas representa el porcentaje de uso de la CPU para cada tarea de las que componen el sistema. La última fila indica el tanto por ciento total de CPU empleado. Como se aprecia en todos los casos se obtiene el funcionamiento en tiempo real con una carga computacional próxima al 100%.

	Baseline @ 600MHz				*Main* @ 720MHz			
	Nemo	Guerra Galaxias	Fútbol	Nemo 1M	Nemo	Guerra Galaxias	Fútbol	Nemo 1M
Descodificador de Vídeo	79.9%	77.5%	78.4%	65.8%	85.7%	84.3%	83.7%	79.8%
Descodificador de Audio	2.0%	2.0%	2.0%	2.0%	2.0%	2.0%	2.0%	2.0%
Análisis de la trama de transporte	3.2%	3.1%	3.2%	1.9%	3.2%	3.1%	3.2%	1.9%
Presentaciones de Vídeo y Audio	1.4%	1.6%	1.7%	2.7%	1.4%	1.5%	1.4%	2.4%
Otros	7.5%	7.4%	7.2%	7.5%	7.5%	7.5%	7.5%	7.8%
Total	94.0%	91.6%	92.5%	79.9%	99.8%	98.4%	97.8%	93.9%

Tabla 4-16. Carga computacional requerida por cada una de las tareas que componen el STB-IP.

Comparando los resultados mostrados en la Tabla 4-15 con los de la Tabla 4-16 se aprecia que hay una pérdida de rendimiento por parte del descodificador de aproximadamente un 8%. Esta pérdida se debe fundamentalmente a la sobrecarga de peticiones por parte del controlador de DMA puesto que, a las que realiza el descodificador, hay que añadir las que produce el analizador de la trama de red, la tarea de presentación del vídeo y los controladores de los periféricos.

4.4.4.3 Rendimiento del descodificador H.264 integrado en un entorno real

Para realizar medidas de rendimiento con secuencias de larga duración en entornos reales se ha empleado el banco de pruebas que se mostró en la Figura 4-3-a[145] y que por comodidad se repite en la Figura 4-53.

[144] Dado que para el perfil *Main* la carga computacional del descodificador es muy próxima al 100%, no es posible lograr el funcionamiento en tiempo real empleando el prototipo desarrollado dentro de los trabajos de tesis (tarjeta DESCOS apartado 7.1). Por tanto, y para poder realizar medidas de rendimiento para este perfil, se ha adaptado el STB-IP a la tarjeta de desarrollo comercial EVMDM642 (apartado 7.2) que incorpora el mismo procesador pero funcionando a 720 MHz. Con esta frecuencia de trabajo sí es posible alcanzar el funcionamiento en tiempo real.

[145] El banco de pruebas mostrado en la Figura 4-3-b no se ha implementado puesto que el transcodificador de vídeo basado en VLC no es capaz de realizar la codificación de secuencias H.264 en tiempo real.

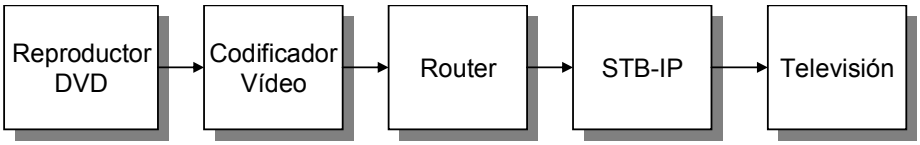


Figura 4-53 Bancos de pruebas empleados para medir el rendimiento en un entorno real.

En este banco de pruebas se ha empleando un codificador comercial [ATEME], que codifica el vídeo procedente de un DVD en formato H.264 y el audio en formato MPEG-1 Capa II. Ambas tramas se encapsulan en una trama de transporte MPEG-2 y se difunden empleando paquetes IP. En la Figura 4-54 se muestra el banco de pruebas en el que el ordenador de la derecha incluye la tarjeta que materializa el codificador comercial, el elemento que se encuentra en la parte inferior es el reproductor de DVD y la placa que se encuentra a la izquierda se corresponde con el prototipo desarrollado[146].

Figura 4-54. Banco de pruebas empleado para las medidas de rendimiento en un entorno real.

Las pruebas se han realizado empleando varios fragmentos de DVD que se han codificado con el perfil *Main* a 3 Mbps utilizando CABAC para la codificación de entropía, un 6% de imágenes I y un 47% de imágenes P y B. En todos los casos se ha medido el rendimiento descodificando unas 3000 imágenes. Los resultados obtenidos han sido muy similares a los presentados en la Tabla 4-16 logrando un funcionamiento en tiempo real con una carga computacional próxima al 100%.

4.4.5 Optimización del tiempo de ejecución del descodificador H.264 para el procesador TMS320DM6437

Seguidamente se presentan las técnicas que se han aplicado para reducir el tiempo de ejecución del descodificador H.264 aprovechando las características del procesador TMS320DM6437. Las principales diferencias entre ambos procesadores (ver apartado 3.2.2) y su influencia en la mejora del rendimiento del descodificador se resumen en los siguientes puntos:

[146] El banco de pruebas es el mismo tanto para la tarjeta DESCOS, como para la tarjeta de desarrollo comercial que integra el TMS320DM642 a 720 MHz.

1. La reducción en el tamaño total de la memoria interna se puede ver compensada con la mayor flexibilidad en su configuración respecto a la del TMS320DM642.

2. La nueva arquitectura del DMA basada en dos controladores, IDMA y EDMA, reduce el tiempo necesario para programar las transferencias lo que implica una mejora en el rendimiento del descodificador.

3. La inclusión de nuevas instrucciones en su repertorio no aporta una gran ventaja desde el punto de vista de la implementación de descodificadores de vídeo. Por tanto, ninguno de los módulos descritos en 4.4.2.3 se ha modificado para incluir estas nuevas instrucciones. A pesar de ello, su inclusión permite al compilador optimizar el código generado, mejorando ligeramente el rendimiento.

El punto de partida para el proceso de optimización empleando el TMS320DM6437 ha sido el código previamente optimizado para el procesador TMS320DM642. En los siguientes apartados se presentan las modificaciones realizadas en el descodificador a nivel de gestión de memoria y de transferencia de datos utilizando el controlador de DMA.

4.4.5.1 Arquitectura de la memoria

La configuración de la memoria interna empleada en el TMS320DM642 no es aplicable a este procesador debido a que los tamaños que poseen cada uno de los niveles existentes son diferentes. Inicialmente, para configurar el reparto óptimo de las memorias internas (entre memoria de propósito general y memoria caché) se han seguido las recomendaciones propuestas en [ficha 001]; en consecuencia se han configurado todas las memorias internas como memorias caché y se han realizado pruebas de rendimiento global. Los resultados obtenidos se han tomado como referencia para poder compararlos con los logrados con otros repartos. Seguidamente se han probado otras configuraciones en las que existe espacio disponible para almacenar funciones o datos de propósito general en memoria interna.

Los mejores resultados se logran configurando las memorias de nivel 1 con el mayor tamaño de caché posible (en ambos casos 32 kB) y repartiendo la memoria de nivel 2 entre memoria caché y memoria de propósito general (64 kB cada una de ellas). Con esta configuración quedan disponibles 48 kB para datos en la memoria de nivel 1 y 64 kB para funciones y datos en la memoria de nivel 2[147].

Por tanto, y dado que la memoria de propósito general disponible es menor que la existente en el TMS320DM642, ha sido necesario eliminar algunos fragmentos de código y datos de esta memoria. Como consecuencia, la distribución de los datos y el programa en la memoria queda del siguiente modo: los *buffers* empleados para la compensación de movimiento y la pila se ubican en memoria de nivel 1 de datos (L1D), las funciones que realizan la codificación de entropía (CAVLC y CABAC) se alojan en memoria de propósito general de nivel 2 (L2), las funciones de filtrado y compensación de movimiento, así como la *heap* se colocan en memoria externa. El resto de funciones se han distribuido entre L2 y memoria externa. La Tabla 4-17 muestra el reparto en los diferentes niveles de memoria de las funciones y *buffers* que en el caso del procesador TMS320DM642 estaban alojados en la memoria interna.

[147] Por tanto, y respecto a la configuración propuesta para el TMS320DM642, aparecen 48 kB de memoria de datos de nivel 1 y se reduce el tamaño de la memoria interna de propósito general que pasa de tener 192 kB a 64 kB.

Código/Datos	Nombre	Tamaño	Nivel de Memoria
Código	CABAC	25 kB	L2
	CAVLC	23 kB	L2
	Filtro Antibloques	16 kB	Externa
	Compensación de Movimiento + IICT	17 kB	Externa
	Solicitudes DMA de las referencias	7 kB	L2
	Otras funciones	22 kB	L2 / Externa
Datos	*Buffers* intermedios para la compensación de movimiento	6 kB	L1D
	Buffers para almacenar los MBs reconstruidos	2 kB	L1D
	Buffers para filtrado	8 kB	L1D
	Buffers auxiliares	2 kB	L1D
	Pila	4 kB	L1D
	Heap	27 kB	Externa

Tabla 4-17. Reparto de funciones y *buffers* en los diferentes niveles de memoria en el procesador TMS320DM6437.

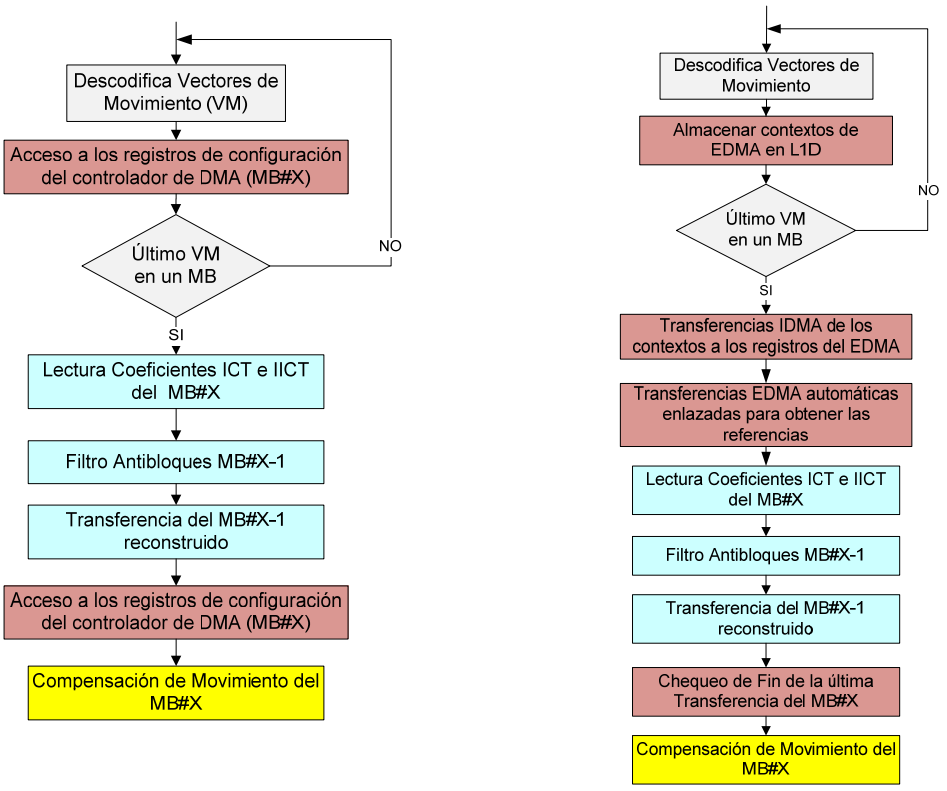
Referencia: Criterios adicionales para la configuración de la memoria interna [ficha 028].
Descripción: Analizar los posibles repartos de las memorias internas entre memoria caché y memoria de propósito general tal y como se indicó en [ficha 001]. En el nivel 1 de memoria de programa y de datos priorizar inicialmente su configuración como memoria caché realizando pruebas de rendimiento global con diferentes repartos de la memoria de nivel 2. Una vez fijado el tamaño de la memoria de nivel 2 repetir las pruebas para diferentes configuraciones de las memorias de nivel 1. En cada uno de estos pasos se ordenarán las funciones y los *buffers* en función del interés de su ubicación en memoria interna. El procedimiento debe repetirse cada vez que se produzcan cambios importantes en la estructura del código a lo largo del proceso de optimización.
Aplicabilidad: Procesadores en los que es posible configurar un reparto entre la memoria caché y la memoria interna de propósito general en los diferentes niveles de la arquitectura de memoria.

4.4.5.2 Controlador de Acceso Directo a Memoria

Cada una de las transferencias de DMA que realiza el TMS320DM642 requiere acceder a los registros de configuración del controlador de DMA (ver Figura 4-39). Este proceso, que se resume en la Figura 4-55-a, es especialmente lento debido a la latencia en las escrituras y lecturas en los registros de los periféricos.

La arquitectura del DMA del procesador TMS320DM6437 permite reducir el número de accesos por parte de la CPU a los registros de configuración de los periféricos aprovechando tanto su capacidad para realizar transferencias enlazadas, como la existencia de dos controladores (IDMA y EDMA).

El algoritmo de descodificación se ha modificado para aprovechar estas nuevas características tal y como se muestra en la Figura 4-55-b. Tras obtener los vectores de movimiento, estos se almacenan en un *buffer* alojado en L1D invirtiendo un ciclo de reloj por dato. Cuando se tienen todas las referencias, se configura el controlador IDMA para realizar una única transferencia que programe los registros del controlador EDMA (*parameter sets*). Esta transferencia se realiza de forma muy eficiente puesto que el controlador IDMA está diseñado específicamente para este tipo de movimiento de datos. Una vez finalizada la transferencia del controlador IDMA, y sin intervención por parte de la CPU, se lanzan de forma automática todas las transferencias de los bloques de referencia. Finalmente, y antes de realizar la compensación de movimiento del macrobloque actual, se comprueba que ha finalizado la última de las transferencias que se enlazaron. Con esta solución la CPU sólo almacena las transferencias a realizar en L1D, programa una transferencia empleando el controlador de IDMA y comprueba que ha finalizado la última de las transferencias del controlador EDMA.



a) TMS320DM642 b) TMS320DM6437

Figura 4-55. Diagrama de flujo del proceso de obtención de las referencias de un macrobloque para ambos DSPs.

El mismo algoritmo es empleado para enviar los macrobloques reconstruidos a la memoria externa en la que se encuentra la imagen reconstruida. Este proceso es más sencillo puesto que tan sólo es necesario programar 3 transferencias, una para la luminancia y dos para las crominancias.

En la Figura 4-56 se detallan las operaciones que se realizan para obtener un macrobloque reconstruido formado por 4 bloques de 8×8 píxeles de luminancia. La secuencia de operaciones es la siguiente:

- Paso 1. Después de descodificar cada vector de movimiento, la información necesaria para obtener cada bloque de referencia (direcciones de origen y destino, tamaño de la transferencia, etc.) se organiza en una estructura de datos (contexto de transferencia). Cada contexto de transferencia asociado a un bloque se almacena en un *buffer* ubicado en L1D. Este *buffer* tiene la misma estructura que los registros de configuración del controlador EDMA (denominados *params*) por lo que es llamado "parámetros imagen" (*shadow params*). El acceso a este *array* se realiza a un régimen de un dato por ciclo de reloj de la CPU.
- Paso 2. Cuando todos los vectores de movimiento se han descodificado, el controlador IDMA se configura para transferir los parámetros imagen a los registros de control del EDMA. Esta transferencia se realiza en paralelo con la ejecución de instrucciones por parte de la CPU sin necesidad de esperas.
- Paso 3. Una vez finalizada la transferencia IDMA, y de forma automática, comienza la primera transferencia de EDMA. En el ejemplo de la figura se transfieren 12 contextos de transferencia dado que hay 4 bloques y para cada bloque se deben transferir la luminancia y dos crominancias.
- Paso 4. El controlador EDMA va transfiriendo las referencias desde la memoria externa a un *buffer* intermedio alojado en memoria interna. Las transferencias se encuentran enlazadas por lo que tras finalizar una, la siguiente comienza de forma automática.
- Paso 5. Tras finalizar las transferencias de EDMA se realiza el proceso de compensación de movimiento para obtener el macrobloque reconstruido y se transfiere a la imagen descodificada.

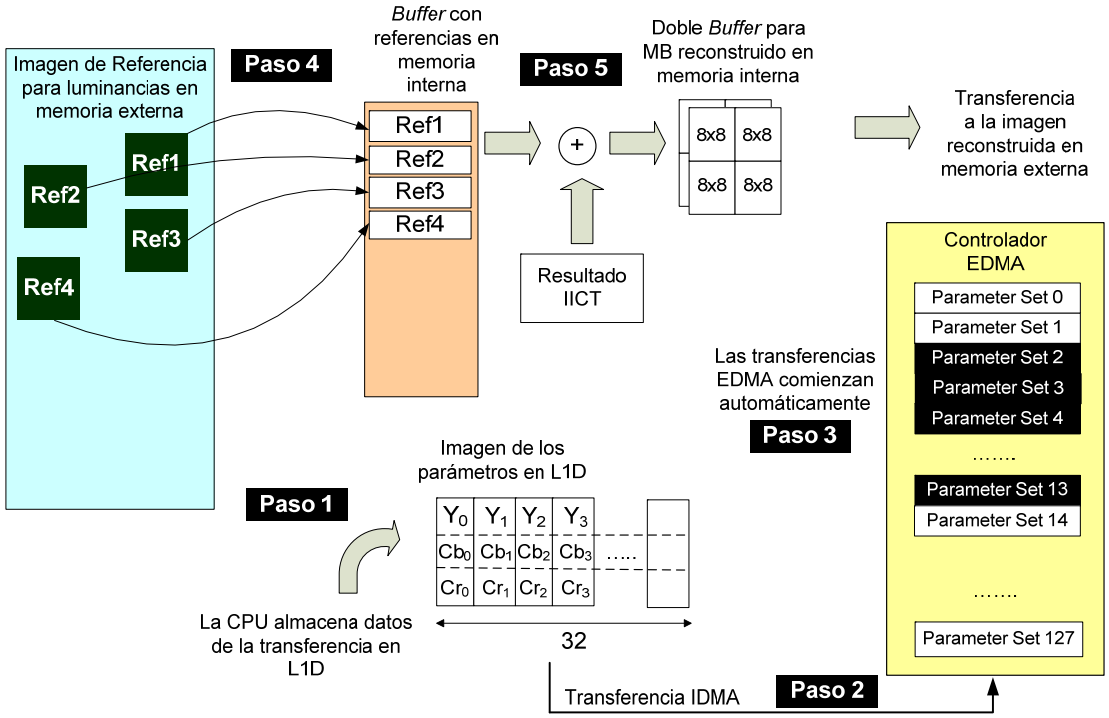


Figura 4-56. Proceso de reconstrucción de un macrobloque empleando los controladores IDMA y EDMA.

Referencia: Empleo de los controladores de DMA para optimizar las transferencias de datos [ficha 029].
Descripción: Almacenar en memoria interna de datos L1D los parámetros necesarios para realizar todas las transferencias mediante en controlador de EDMA. Transferir los parámetros almacenados a los registros de configuración del EDMA empleando el controlador de IDMA. Configurar el controlador EDMA para que enlace las transferencias configuradas. Comprobar que ha finalizado la última antes de comenzar a usar los datos transferidos.
Aplicabilidad: Procesadores en los que existen varios controladores de DMA que permiten optimizar las transferencias entre memoria interna y memoria externa.

4.4.6 Optimización a nivel de sistema del descodificador H.264 para el procesador TMS320DM6437

Tal y como se indicó en [ficha 017] es necesario realizar un análisis de las peticiones de transferencias de DMA que solicitan cada una de las tareas que componen el sistema para distribuirlas lo más equitativamente posible entre las colas de petición que posee el controlador de DMA. En el procesador TMS320DM6437 sólo existen dos colas de petición de transferencias por lo que, para realizar una distribución homogénea entre ellas, éstas se han asignado del siguiente modo: las que realiza el descodificador se solicitan a través de la cola de baja prioridad y las que realizan el resto de tareas, incluyendo las relativas a los fallos de caché, se encolan en la de alta prioridad.

4.4.7 Resultados de la optimización para el procesador TMS320DM6437

En los siguientes apartados se muestran los resultados del rendimiento del descodificador obtenidos empleando los tres bancos de pruebas descritos en el apartado 4.1.2. Además, se ha realizado una comparativa del rendimiento del descodificador optimizado para el procesador TMS320DM6437 con el obtenido para el TMS320DM642.

4.4.7.1 Rendimiento del descodificador H.264 aislado

El rendimiento se ha medido empleando el mismo banco de pruebas que el utilizado para el procesador TMS320DM642 y mostrado en la Figura 4-52. Para poder realizar una comparación del rendimiento del descodificador H.264 en ambos procesadores se han empleado las mismas secuencias que se describieron en el apartado 4.4.4.1.

El número medio de ciclos de reloj que invierte el procesador en descodificar las imágenes de cada secuencia se muestra en la primera fila de la Tabla 4-18. En el resto de filas se representa en número medio de ciclos de reloj empleado por los bloques funcionales más representativos. Finalmente, la última fila indica el porcentaje medio de uso de la CPU. Como se aprecia se obtiene el funcionamiento en tiempo real en todos los casos.

# ciclos×10^6	Nemo		Guerra Galaxias		Football		Nemo1M	
	BP	MP	BP	MP	BP	MP	BP	MP
Descodificador	14.02	19.48	13.48	19.68	13.69	19.53	11.72	17.92
CAVLC/CABAC	4.69	7.59	4.52	7.47	4.69	7.35	3.63	6.52
IICT+MC	4.19	5.24	4.12	5.49	4.12	5.61	3,55	5.14
Filtro Antibloques	4.28	4.83	3.95	4.89	3.99	4.71	3.75	4.51
Otras	0.87	1.82	0.90	1.84	0.90	1.86	0.79	1.75
CPU% @594 MHz	58.8	81.2	56.5	82.6	57.4	81.9	51.0	73.7

Tabla 4-18. Ciclos de reloj invertidos por el descodificador H.264 optimizado para el DSP DM642.

Estos resultados se han comparado con los obtenidos para el procesador TMS320DM642 que se presentaron en la Tabla 4-15. La Tabla 4-19 resume el rendimiento del descodificador para ambos procesadores. Las dos primeras columnas muestran en número medio de ciclos de reloj empleado por cada procesador para descodificar cada imagen, mientras que la tercera columna refleja la mejora expresada en tanto por ciento. Como se aprecia en todos los casos la mejora está por encima del 7% para el perfil *Main* y del 11% para el perfil *Baseline*.

Por tanto, y a pesar de disponer de menor memoria interna en el procesador TMS320DM6437, la técnica empleada para optimizar las transferencias de datos por DMA y la distribución que se ha realizado de los niveles de memoria interna, permiten obtener una mejora del rendimiento global.

		TMS320DM6437	TMS320DM642	Mejora %
NEMO	BP	14.0	16.6	15.6%
	MP	19.5	21.1	7.6%
GUERRA GALAXIAS	BP	13.5	16.0	15.6%
	MP	19.7	21.7	9.2%
FOOTBALL	BP	13.7	16.2	15.4%
	MP	19.5	21.5	9.3%
NEMO1M	BP	11.7	13.2	11.4%
	MP	17.9	20.4	12.2%

Tabla 4-19. Resultados comparados del rendimiento del descodificador H.264 para los procesadores TMS320DM642 y TMS320DM6437.

4.4.7.2 Rendimiento del descodificador H.264 integrado en un sistema de descodificación completo

Las medidas de rendimiento del descodificador optimizado para el procesador TMS320DM6437 se han llevado a cabo empleando la plataforma DM6437EVM descrita en el apartado 7.3 del Anexo A. Para ello se ha migrado el STB-IP a dicha plataforma empleando como punto de partida el sistema desarrollado para el TMS320DM642 sobre el que se han realizado los siguientes cambios:

- Actualización de los controladores de los dispositivos ya que los periféricos que integra el TMS320DM6437 son diferentes a los incluidos en el

TMS320DM642. Estos cambios han provocado que las tareas de presentación de audio y vídeo hayan sido prácticamente reescritas.

- Adaptación de todas las transferencias de DMA que se realizan tanto en la tarea de red como en la tarea de presentación de vídeo para que aprovechen la arquitectura de los controladores de DMA del procesador TMS320DM6437.
- Integración de la pila TCP/IP adaptada a este procesador.

Como resultado se dispone de un STB-IP completo en el que es posible realizar medidas de rendimiento de las diferentes tareas funcionando en tiempo real. La Tabla 4-20 muestra el porcentaje de uso de la CPU de cada una de las tareas que componen el sistema. La última fila representa el porcentaje total de uso de la CPU empleando una frecuencia de reloj de 594 MHz. Como se aprecia se obtiene el funcionamiento en tiempo real para todas las secuencias de pruebas empleadas.

	Perfil *Baseline*				Perfil *Main*			
	Nemo	Guerra Galaxia	Fútbol	Nemo 1M	Nemo	Guerra Galaxia	Fútbol	Nemo 1M
Descodificador de Vídeo	58.8%	56.5%	57.4%	51.0%	81.2%	82.6%	81.9%	73.7%
Descodificador de Audio	1.9%	2.1%	2.0%	2.0%	1.9%	2.1%	2.1%	2.0%
Trama de Transporte	3.0%	3.1%	3.1%	2.0%	3.0%	3.1%	3.2%	1.9%
Presentación de Vídeo[148] y de Audio	2.4%	2.3%	2.5%	2.8%	2.4%	2.5%	2.3%	2.7%
Otros	6.5%	6.4%	6.6%	6.4%	6.2%	6.5%	6.5%	6.2%
CPU% @594 MHz	72.6%	70.4%	71.6%	64.2%	94.7%	96.8%	96.0%	86.5%

Tabla 4-20. Carga computacional requerida por cada una de las tareas que componen el STB-IP.

4.4.7.3 Rendimiento del descodificador H.264 integrado en un entorno real

Para realizar medidas de rendimiento con secuencias de larga duración en entornos reales se ha empleado el mismo banco de pruebas que el utilizado para el procesador TMS320DM642 (ver Figura 4-53). En este banco de pruebas se emplea un codificador comercial [ATEME], tanto para generar la trama de vídeo codificada en H.264, como la de audio en formato MPEG-1 Capa II.

Al igual que se hizo para el procesador TMS320DM642, en las pruebas se han utilizado varios fragmentos de DVD que se han codificado con el perfil *Main* a 3 Mbps utilizando CABAC para la codificación de entropía, un 6% de imágenes I y un 47% de imágenes P y B.

[148] Como se puede apreciar el porcentaje de carga computacional de la tarea de presentación de vídeo se ha incrementado sustancialmente respecto a las versiones del STB-IP basadas en el procesador TMS320DM642. Esto se debe a que el controlador de vídeo que posee este procesador requiere que los píxeles de luminancia y crominancia se encuentren en formato UYVY a diferencia del procesador anterior en el que los datos de luminancia y crominancia se enviaban de forma separada. Por tanto, y debido a que en la imagen descodificada las componentes de luminancia y crominancia se encuentran separadas, la tarea de presentación vídeo necesita realizar un procesamiento de los píxeles antes de tranferir la imagen al controlador de vídeo.

En todos los casos se ha medido el rendimiento descodificando unas 3000 imágenes. Los resultados obtenidos ha sido muy similares a los presentados en la Tabla 4-20 logrando un funcionamiento en tiempo real con una carga computacional próxima al 70% para el perfil *Baseline* y al 90% para el perfil *Main*.

4.5 Resumen

En este capítulo se han presentado todas las técnicas que se han aplicado para la optimización del tiempo de ejecución de diferentes descodificadores de vídeo sobre procesadores digitales de señal. La generalización de estas técnicas ha permitido definir la metodología de optimización de este tipo de aplicaciones que se presenta en el capítulo 5.

Las técnicas aplicadas se han agrupado en tres bloques; las relacionadas con la organización del código y los datos en los diferentes niveles de la memoria interna del procesador, las relativas al movimiento de datos entre la memoria interna y la memoria externa empleando el/los controlador/es de DMA integrado/s en los procesadores y, finalmente, las relacionadas con la adaptación de las funciones que materializan el descodificador a la arquitectura SIMD de estos procesadores.

Para cada una de las técnicas aplicadas se ha generado una ficha en la que se le asigna un nombre, se realiza una breve descripción del modo en el que se ha aplicado a la optimización del descodificador correspondiente y se trata de generalizar su empleo tanto para otros procesadores como para otros estándares. Estas fichas servirán de base para la síntesis de la metodología que se realiza en el capítulo 5.

Para cada una de las técnicas, no sólo se realiza una descripción teórica de la misma, sino que se ha medido su repercusión en las prestaciones del descodificador en tres bancos de pruebas:

- Pruebas con el descodificador aislado tanto en simulación como sobre una plataforma *hardware* para caracterizar sus prestaciones.
- Pruebas con el descodificador integrado en un STB-IP completo desarrollado en el marco de esta tesis (ver Anexo B).
- Pruebas en entornos reales de difusión de televisión digital.

En cada uno de los casos se ha evaluado el tiempo que invierte el descodificador en procesar las secuencias de entrada con el fin de valorar el impacto que tiene sobre sus prestaciones la integración del descodificador con otros elementos habituales en estos sistemas. Estas pruebas han permitido extraer algunas conclusiones a nivel de sistema que se incluirán como parte de la metodología de optimización.

5 SINTESÍS DE LA METODOLOGÍA DE OPTIMIZACIÓN

5.1 Introducción

En este capítulo se explica la metodología de optimización en velocidad de descodificadores de vídeo para procesadores digitales de señal. Esta metodología se ha sintetizado a partir de las experiencias de optimización de descodificadores compatibles con diferentes estándares que se han mostrado en el capítulo 4.

La metodología propuesta se materializa en una serie de pasos pensados que se recomienda realizar secuencialmente. A cada uno de los pasos se dedica, a continuación, un apartado de este capítulo:

- Elección del código de referencia (apartado 5.2).
- Optimización de las funciones con una mayor carga computacional (apartado 5.3).
- Optimización del tiempo de acceso a los datos alojados en memoria externa (apartado 5.4).
- Optimización del tiempo de ejecución de las funciones alojadas en memoria externa (apartado 5.5).
- Distribución de la memoria interna entre memoria de propósito general y memoria caché (apartado 5.6).
- Optimización a nivel de sistema (apartado 5.7).

En cada uno de estos apartados se exponen las acciones que deben llevarse a cabo, y se justifican en base a la experiencia acumulada en los procesos de optimización que se han realizado. Para ello se hace referencia a las fichas que resumen las técnicas empleadas en cada proceso de optimización.

El apartado 5.8 contiene una versión resumida de la metodología propuesta, que recoge los pasos que deben llevarse a cabo, pero no los justifica como se hace en los apartados anteriores. Finalmente, en el apartado 5.9 se hace un resumen del capítulo.

5.2 Elección del código de referencia

La metodología de optimización propuesta descansa en la existencia de un *software* de referencia que materialice el descodificador de forma compatible con el estándar que se desea implementar. En todos los casos conocidos hasta el momento este *software* no solo existe, sino que se dispone de varias opciones de entre las cuales resulta necesario elegir una. La elección de un *software* que materialice el descodificador compatible con el estándar que se desea implementar es de una importancia capital para el proceso de optimización: una correcta elección del mismo y las primeras pruebas que deben realizarse con él, condicionan fuertemente todo el proceso posterior.

Por este motivo, aunque no entra dentro del proceso de optimización propiamente dicho, la elección del código de referencia se ha incluído como la primera fase de la metodología de optimización. Esta primera fase, a su vez, se puede dividir en las etapas que se detallan a continuación.

5.2.1 Evaluación de las alternativas disponibles

Para hacer una selección del *software* que se empleará como punto de partida es necesario evaluar las alternativas existentes. Una de las que siempre se encuentra disponible es la desarrollada dentro del proceso de estandarización. Sin embargo, esta no siempre es la opción más interesante puesto que:

- el código no se encuentra optimizado para ninguna plataforma lo que obliga a realizar un proceso de optimización desde cero de todos sus módulos.
- soporta todos los perfiles y niveles definidos en el estándar lo que implica que normalmente es necesario realizar una labor de selección, porque en las implementaciones habitualmente sólo es necesario soportar un conjunto de perfiles y niveles.
- en los estándares más recientes, el *software* de referencia se está codificando en C++, lenguaje que resulta poco adecuado para el desarrollo de sistemas embebidos.

Por tanto es interesante buscar otras implementaciones de *software* libre que puedan servir como punto de partida. Normalmente estas implementaciones se han desarrollado para entorno PC, por lo que las primeras pruebas deben realizarse sobre dicha plataforma.

Antes de tomar una decisión sobre el *software* a emplear se deben realizar pruebas de conformidad del mismo, puesto que en algunas ocasiones estos descodificadores pueden no implementar completamente los perfiles y niveles que se desean soportar. Para ello se utilizarán las secuencias de referencia creadas dentro del proceso de estandarización. Estas secuencias deben ser descodificadas tanto por el descodificador incluido en el estándar como por las implementaciones candidatas a ser empleadas como punto de partida para el proceso de optimización. Las secuencias descodificadas por ambos descodificadores se compararán para confirmar que los resultados obtenidos son idénticos en todos los casos. Los resultados de estas pruebas pueden servir para descartar definitivamente algunas de las implementaciones evaluadas.

Si alguno de los descodificadores seleccionados es conforme con los perfiles y niveles que se desean implementar, es candidato a ser empleado como *software* de referencia para el proceso de optimización. Para poder tomar una decisión es necesario valorar el rendimiento, en términos de tiempo empleado para la descodificación de las secuencias, de todos los descodificadores compatibles con el

estándar. Si bien la medida del rendimiento en el entorno PC no es directamente extrapolable al entorno de desarrollo de los procesadores digitales de señal, sí puede servir, en una primera aproximación, para descartar alguno de los descodificadores si sus prestaciones son muy inferiores al del resto de implementaciones disponibles.

Si tras este proceso sigue habiendo varias alternativas es necesario realizar la migración de todos los descodificadores al entorno de desarrollo del DSP seleccionado para poder evaluar su rendimiento y tomar una decisión sobre cual de ellos se empleará como punto de partida.

5.2.2 Proceso de migración al entorno de desarrollo del DSP

El proceso de migración de los descodificadores disponibles al entorno de desarrollo del DSP seleccionado implica la generación de un proyecto, la selección de las opciones de compilación y la inclusión de todos los ficheros fuente. En la configuración del proyecto hay que prestar especial atención a la definición de las constantes que permiten la compilación condicional de algunos fragmentos de código que pueden estar optimizados para arquitecturas de 32 bits y que, por tanto, mejorarán sustancialmente las prestaciones del descodificador. En cuanto a la configuración del proyecto, se debe prestar especial atención a las opciones de optimización que posee el compilador. En este sentido se deben emplear aquellas que permiten reducir el tiempo de ejecución del código.

En el proceso de migración es necesario configurar el RTOS que poseen los entornos de desarrollo de este tipo de procesadores. En esta configuración hay que definir el mapa de memoria del procesador y la ubicación de los diferentes fragmentos de código en los niveles de memoria disponibles. La memoria interna de los procesadores digitales de señal puede ser configurada como memoria de propósito general o memoria caché. Inicialmente (ver [ficha 001]) se debe configurar la totalidad de la memoria interna como caché. Esta configuración hace que no sea posible alojar en dicha memoria ningún fragmento de código, ni ningún dato de los empleados por el descodificador.

En la configuración del RTOS es necesario definir la pila y la *heap* del sistema. En este sentido, y como se refleja en el apartado 4.3.1.2, es recomendable asignar un tamaño elevado[149] a ambas secciones para evitar problemas de desbordamiento o falta de memoria respectivamente, que son difíciles de depurar en tiempo de ejecución. En etapas posteriores del proceso se ajustará el tamaño de ambas secciones para que se adecue a las necesidades reales.

Cuando se dispone de una versión operativa del descodificador, se deben repetir las pruebas de conformidad que se realizaron para el PC y comprobar que los resultados son correctos. Debido al elevado número de secuencias de test a emplear, y tal y como se indicó en el apartado 4.3.1.2.6, es necesario disponer de un banco de pruebas automatizado que permita comparar las secuencias descodificadas por el DSP con las obtenidas por parte del PC.

Si el descodificador ejecutado en el DSP no supera las pruebas de conformidad, la experiencia acumulada indica que es probable que los problemas en tiempo de ejecución se deban a que el código realiza lecturas y/o escrituras de datos de 16/32/64 bits ubicados en direcciones de memoria no alineadas al tamaño del dato. El DSP no es capaz de realizar tales accesos desde el lenguaje C por lo que es necesario

[149] En el caso de los descodificadores MPEG-2 y MPEG-4 el tamaño inicial que se asignó a la pila fue de 1 MB y a la *heap* de 0.5 MB, en el caso del descodificador H.264 el tamaño de la pila fue de 1.5 MB y el de la *heap* de 1 MB.

identificar y reescribir los fragmentos de código en los que se realizan estas lecturas/escrituras empleando instrucciones en ensamblador que sí permiten este tipo de accesos (ver apartado 4.3.1.2.4).

Una vez que se dispone de todos los descodificadores funcionando correctamente en entorno DSP, se deben realizar medidas de rendimiento que permitan seleccionar el *software* que se empleará como punto de partida. Para ello se emplearán las herramientas de *profile* que incluyen los entornos de desarrollo de estos procesadores. El descodificador que invierta menor tiempo en descodificar un conjunto de secuencias seleccionadas del estándar será el utilizado para realizar el proceso de optimización.

Las medidas de rendimiento del descodificador seleccionado se emplearán como referencia para el proceso de optimización puesto que serán los valores con los que se realizarán las comparaciones tras aplicar las recomendaciones que se indican en cada una de las etapas posteriores.

5.3 Optimización de las funciones que poseen una mayor carga computacional

El primer paso del proceso de optimización propiamente dicho consiste en la recodificación de las funciones que poseen una mayor carga computacional. A la hora de seleccionar qué funciones es necesario optimizar, hay que tener en cuenta el número de ciclos de reloj que invierte la CPU en ejecutar dicha función sin tener en consideración los fallos en el acceso a las memorias caché de programa y datos (ver apartado 4.2.2.3). La reducción de estos fallos se realizará en etapas posteriores del proceso de optimización (ver apartados 5.4.1, 5.4.2, 5.5 y 5.6)

Una vez identificadas las funciones con mayor carga computacional, es necesario recodificarlas empleando las instrucciones de ensamblador que aprovechan la arquitectura SIMD que poseen estos procesadores.

En una primera fase, y tal y como se indicó en la [ficha 013], es interesante evaluar si alguno de los módulos que componen el descodificador ha sido previamente optimizado por el fabricante del procesador y se encuentra en alguna de las librerías de procesamiento de vídeo. La inclusión de estas funciones de librería implica normalmente la modificación de los parámetros de entrada y salida por lo que una vez integradas es necesario evaluar de nuevo el rendimiento de la misma. Habitualmente las librerías del fabricante no incluyen funcionalidades de los descodificadores más recientes por lo que no siempre es posible localizar funciones que se puedan emplear directamente.

Para seleccionar las funciones que han de recodificarse hay que tener en cuenta el tipo de función de que se trata. De forma general las funciones se pueden agrupar en aquellas que realizan procesamiento de datos mediante bucles (compensación de movimiento, filtrado, interpolación de píxeles, etc.) y las que toman las decisiones del algoritmo en función de las condiciones que se vayan cumpliendo (análisis sintáctico de la trama, cálculo de coeficientes del filtro antibloques,…). Para las primeras es interesante aplicar las siguientes técnicas:

- Emplear instrucciones de carga, promedio y almacenamiento de varios píxeles [ficha 014]. Estas técnicas se pueden aplicar a aquellas funciones en las que se realizan operaciones con datos de tamaño inferior al de los registros del procesador y que se encuentren ubicados en posiciones contiguas de memoria. Varios de esos datos pueden ser cargados simultáneamente a los registros internos del procesador para realizar

operaciones con todos ellos simultáneamente. En los descodificadores de vídeo este tipo de técnicas se han aplicado en los siguientes módulos:

  - compensación de movimiento para realizar la interpolación y el promediado de los píxeles.
  - Almacenamiento de vectores de movimiento necesarios para la predicción de bloques posteriores.
  - cálculo de las transformadas inversas.
  - predicción *intra*.

- Emplear instrucciones de empaquetado y desempaquetado de datos [ficha 015]. Estas técnicas se deben aplicar en aquellos casos en los que sea necesario realizar operaciones en las que los datos posean diferentes tamaños. En estos casos es necesario empaquetar/desempaquetar los datos almacenados en los registros internos para que las operaciones se realicen con la resolución necesaria. Las funciones que se han recodificado aplicando estas técnicas son las que realizan la suma de la predicción de un bloque con el residuo obtenido tras realizar la transformada inversa.
- Emplear instrucciones de multiplicación/acumulación y desplazamiento para operaciones de interpolación [ficha 026]. El objetivo de esta técnica consiste en emplear las instrucciones de multiplicación de varios píxeles almacenados en un registro interno del procesador por una serie de coeficientes y acumular el resultado. Este tipo de operaciones son muy habituales en los procesos de filtrado que se aplican para la obtención de la predicción con ½ y ¼ píxel y en el filtrado antibloques.

Para el segundo tipo de funciones la estrategia de optimización pasa por aplicar la técnica descrita en la [ficha 024], consistente en eliminar los saltos condicionales que impiden la paralelización de las operaciones. Para ello se pre-calculan todas las condiciones, se realizan todos los posibles cálculos y, finalmente, se asignan los valores adecuados en función de las condiciones calculadas. Esta técnica se ha aplicado exclusivamente en el filtro antibloques empleado en el descodificador H.264.

La última etapa de esta fase del proceso de optimización se centra en la escritura en ensamblador y la paralelización manual (ver [ficha 025]) de alguna función. La dificultad de este proceso hace que la aplicación de esta opción se limite a funciones de tamaño muy reducido que se ejecutan multitud de veces por imagen. En el caso de los descodificadores de vídeo optimizados, esta técnica sólo se ha aplicado para la extracción de la trama de bits de los coeficientes codificados con CABAC.

5.4 Optimización del tiempo de acceso a los datos alojados en memoria externa

El acceso a los datos que se encuentran en la memoria externa es uno de los factores que más reduce las prestaciones de los descodificadores. Para minimizar el tiempo de acceso a estos datos hay que tener en consideración en tamaño de los *buffers* a los que se accede.

Si éstos son de tamaño reducido, como por ejemplo los que permiten almacenar los píxeles de un macrobloque, pueden alojarse directamente en memoria interna; si por el contrario se trata de *buffers* que no pueden alojarse en memoria interna debido a su elevado tamaño, como pueden ser los que almacenan las imágenes descodificadas, se empleará el controlador de DMA para transferir en cada momento el fragmento del *buffer* que requiere la CPU, mientras ésta realiza otras tareas. De

este modo se evita que la CPU acceda directamente a datos que se encuentran en memoria externa, lo que provoca fallos en las memorias caché y la correspondiente parada del *pipeline* del procesador hasta que dichos datos estén disponibles.

Para poder emplear este controlador es necesario declarar *buffers* intermedios de tamaño reducido en memoria interna que servirán de origen y destino de las transferencias de DMA. El acceso a estos *buffers* intermedios reduce las esperas por parte de la CPU a la hora de acceder a los datos.

En los siguientes subapartados se muestra cómo debe emplearse explícitamente[150] el controlador de DMA para reducir el tiempo de acceso a los datos y los criterios que se pueden aplicar para ubicar los diferentes *buffers* en memoria interna.

5.4.1 Transferencias de datos empleando el controlador de DMA

El proceso de compensación de movimiento requiere las imágenes previamente descodificadas que, debido a su elevado tamaño, deben almacenarse en memoria externa. Cuando la CPU accede a estos datos es probable que no se encuentren en ninguno de los niveles de la memoria caché, lo que provoca continuos fallos de acceso con las correspondientes paradas en el *pipeline* de la CPU. Para reducir el tiempo de acceso a estos datos se definen *buffers* de menor tamaño en memoria interna a los que se transfieren los datos necesarios para realizar la compensación de movimiento, mientras que la CPU continúa ejecutando otras fases del proceso de descodificación. Cuando se obtiene el macrobloque reconstruido, éste es transferido de nuevo a la memoria externa por parte del controlador de DMA mientras la CPU comienza el procesamiento del siguiente macrobloque de la imagen. El objetivo final es que la CPU nunca deba esperar a la finalización de las transferencias de datos entre memoria interna y memoria externa.

La inclusión de las transferencias de datos empleando el controlador de DMA en el código del descodificador es un proceso complejo que se ha descrito en varias fases como se explicó en los apartados 4.2.2.2, 4.3.2.2 y 4.4.2.2. Algunas de estas fases son suficientemente generales como para poder aplicarlas a cualquier descodificador de vídeo que se desee optimizar sobre plataformas basadas en DSPs, mientras que otras sólo tienen sentido en algunos casos particulares, que dependen del descodificador que se esté optimizando y del procesador que se emplee. A continuación se enumeran las recomendaciones generales que deben aplicarse para realizar las transferencias de datos mediante el controlador de DMA:

- Obtención de los píxeles empleados (fichas [006], [007] y [020]). Analizar los píxeles implicados en la compensación de movimiento. Definir en la memoria interna tantos *buffers* como sea necesario para alojar en ellos todos los píxeles que permitan realizar la compensación de movimiento con resolución fraccionaria. Configurar el controlador de DMA inmediatamente después de obtener los vectores de movimiento, para transferir a los *buffers* intermedios los píxeles de las imágenes descodificadas anteriormente, mientras la CPU ejecuta otras partes del algoritmo. Obtener los píxeles interpolados alojando los resultados en otros *buffers* intermedios ubicados en memoria interna. Realizar la suma con el residuo y transferir el resultado a la imagen descodificada empleando de nuevo el controlador de DMA.

[150] Se emplea el término “explícitamente” para diferenciar las transferencias que se programan dentro del código del descodificador para mover bloques de datos, de las que se realizan cuando la CPU accede a información que no se encuentra alojada en ninguno de los niveles de la memoria caché.

- Empleo de *buffers* ping-pong ([ficha 008]). Duplicar el tamaño de algunos *buffers* (*buffers* ping-pong) definidos en memoria interna de modo que el controlador de DMA realice transferencias a/desde uno de ellos mientras que la CPU procesa y almacena datos del siguiente macrobloque en el otro.
- Tamaño y alineamiento de las transferencias ([ficha 010]). Forzar que todas las transferencias tengan un tamaño múltiplo de 32 bits y que sus direcciones de origen y destino también sean múltiplos de 32 bits. Si alguna transferencia no cumple estos requisitos, el bloque a transferir debe enmarcarse en otro de mayor tamaño que sí esté alineado y gestionar el desplazamiento del macrobloque de referencia respecto al bloque de los píxeles transferidos.
- Distribución de las peticiones de transferencia ([ficha 011]). Analizar el número de peticiones de transferencias que realiza el descodificador. Repartir homogéneamente entre las colas de petición del controlador de DMA las transferencias que se soliciten para evitar la sobrecarga de alguna de ellas.

Una vez aplicadas las recomendaciones anteriores es necesario medir de nuevo las prestaciones del descodificador para evaluar si en algún momento la CPU debe esperar a las transferencias de datos por parte del controlador de DMA, con la correspondiente pérdida de prestaciones. Si no se realizan esperas, no es necesario aplicar más recomendaciones; sin embargo, si se detecta que la CPU debe esperar es posible aplicar dos recomendaciones adicionales:

- Procesar varios macrobloques en paralelo (fichas [009] y [022]). Rellenar los tiempos en los que el controlador está transfiriendo por DMA los píxeles necesarios para poder realizar la compensación de movimiento de un macrobloque con instrucciones que permitan que la CPU pueda continuar el procesamiento de otros datos. Dado que éstos no pueden pertenecer al mismo macrobloque, es necesario comenzar a descodificar el siguiente por lo que el bucle principal de descodificación se ve modificado para manejar simultáneamente datos de varios macrobloques simultáneamente. Dependiendo de la cantidad de tiempo que la CPU espera la finalización de las transferencias, será necesario alterar en mayor o menor medida el bucle de descodificación (ver apartados 4.2.2.2.4 y 4.4.2.2.2).
- Reducir el número de transferencias del controlador de DMA ([ficha 012]). Almacenar los macrobloques descodificados de la imagen en un doble *buffer* intermedio alojado en memoria interna de modo que, cuando se dispone de todos los que componen una tira de macrobloques de la imagen, se realiza una única transferencia mediante el controlador de DMA.

Existen otras recomendaciones que no pueden ser generalizables para todos los descodificadores, ni para todos los procesadores digitales de señal. En relación con los descodificadores es posible reducir del número de *buffers* internos ([ficha 021]). Si el estándar permite que los macrobloques tengan particiones en bloques de tamaño menor (como por ejemplo ocurre en H.264), es necesario definir muchos *buffers* intermedios y realizar una gestión compleja de las transferencias que reducirá las prestaciones del descodificador. En este caso se propone definir un único *buffer* en memoria interna que permita alojar todas las posibles referencias que pueda tener un macrobloque que se encuentre dividido en bloques de menor tamaño. A continuación transferir a este *buffer* las referencias, incluyendo los píxeles adicionales necesarios para poder realizar la compensación de movimiento con precisión no entera. Realizar la compensación de movimiento con los datos alojados en este *buffer*, almacenando el macrobloque reconstruido en otro *buffer* auxiliar alojado también en memoria interna. Este macrobloque será finalmente transferido a la imagen descodificada.

Por otra parte, si el procesador dispone de varios controladores de DMA optimizados ([ficha 029]) para diferentes tipos de transferencias, se emplearán estos controladores para minimizar el tiempo de acceso a los diversos niveles de memoria. En el caso de los procesadores de la familia C64x+, hay que almacenar en memoria interna de datos L1D los parámetros necesarios para realizar todas las transferencias mediante en controlador de EDMA; a continuación, transferir los parámetros almacenados a los registros de configuración del EDMA empleando el controlador de IDMA; y finalmente, configurar el controlador EDMA para que enlace las transferencias configuradas.

5.4.2 Criterios para alojar datos en memoria interna

El uso del controlador de DMA obliga a tener que alojar *buffers* de reducido tamaño en memoria interna de propósito general. Como se ha explicado en el apartado 5.2.2, inicialmente, toda la memoria interna se asigna a memoria caché ([ficha 001]) por lo que en esta fase del proceso es necesario modificar la configuración inicial.

Antes de modificar la configuración de la memoria interna es necesario evaluar el tamaño de los *buffers* empleados por parte del controlador de DMA puesto que éstos deben alojarse obligatoriamente en memoria interna. El tamaño de estos *buffers* condiciona el espacio mínimo de memoria de propósito general que se debe asignar en la configuración del sistema operativo. Para poder alojar estos *buffers* en memoria interna se realizarán varios repartos entre caché y memoria de propósito general midiendo las prestaciones del descodificador con cada uno de ellos y seleccionado el que mejores resultados produzca ([ficha 001]).

Junto a estos *buffers* empleados por el controlador de DMA es posible alojar otros que el descodificador emplee frecuentemente y que por su tamaño puedan ser ubicados en la memoria interna. Para identificar los *buffers* susceptibles de alojarse en esta memoria es necesario analizar el número de fallos en el acceso a la memoria caché de datos que genera cada una de las funciones. Este dato indica qué porcentaje de la carga computacional asociada a una determinada función se debe a que la CPU debe acceder a datos almacenados en memoria externa. A partir de esta información es posible identificar los *buffers* que maneja la función, responsables de los fallos en el acceso a la caché de datos. Dependiendo del tamaño de estos *buffers*, éstos podrán ser alojados en el espacio disponible de la memoria interna. Un ejemplo de esta técnica se mostró en la [ficha 023] en la que la función que realiza el filtro antibloques empleaba píxeles de bloques vecinos para calcular los píxeles filtrados. Estos píxeles se almacenan en *buffers* intermedios alojados en memoria interna para acelerar el acceso a los mismos.

Adicionalmente, y como se indicó en la [ficha 005], para reducir el tiempo de descodificación todos los *buffers* que se almacenan en memoria interna se deben dimensionar a un tamaño múltiplo del de la línea de la caché de datos de nivel 1 y alinear su dirección de comienzo a una posición de memoria que también sea múltiplo de dicho tamaño. Todos los *buffers* que se incluyan en memoria interna se agruparán en secciones alineadas a posiciones de memoria que sean múltiplo del tamaño de una vía de la caché de datos de nivel 1.

5.5 Optimización del tiempo de ejecución de las funciones alojadas en memoria externa

Como se ha explicado en el apartado 5.2.2, inicialmente, las funciones que componen el descodificador se alojan en memoria externa ([ficha 001]). Sin embargo,

algunas de ellas pueden ubicarse en memoria interna de propósito general para reducir su tiempo de ejecución. Para seleccionar las funciones que se pueden ubicar en esta memoria, el parámetro fundamental a tener en cuenta son los fallos en la memoria caché de programa que genera cada una de las funciones (ver [ficha 019]).

Alojando las funciones con una mayor carga computacional y un mayor número de fallos en caché de programa en memoria interna (se puede emplear el número de fallos en caché que se producen en cada acceso como criterio adicional para la selección) se producirá una mejora en las prestaciones del descodificador.

Además, y como se indicó en la [ficha 003], las funciones que se ubiquen en la memoria se agruparán generando secciones de código de tamaño múltiplo del tamaño de la caché de programa de nivel 1. En estas secciones se incluirán aquellas funciones que se ejecuten de forma secuencial. Cada sección se debe alojar a partir de direcciones de memoria que sean múltiplo del tamaño de la caché de nivel 1.

Finalmente, cuando el tamaño del código del bucle principal de descodificación sea cercano al tamaño de la memoria interna, se valorará la posibilidad de incluir todo el código en memoria interna de propósito general (ver [ficha 004]). En este caso se tratará de configurar el compilador para obtener el código del menor tamaño posible.

5.6 Distribución de las memorias internas entre propósito general y caché

El esfuerzo aplicado a distribuir los *buffers* de datos y las funciones en la memoria debe realizarse de forma gradual a lo largo de todo el proceso de optimización puesto que los cambios en el código a menudo influyen, tanto en el tamaño de los datos, como en el de las funciones (fichas [003] y [005]).

Cuando el proceso de optimización está avanzado se debe realizar un ajuste fino de la ubicación de los datos y el programa en los diferentes niveles de memoria. Por tanto es necesario analizar los posibles repartos de las memorias internas entre memoria caché y memoria de propósito general.

Como criterio general ([ficha 028]), en relación al nivel 1 de memoria de programa y de datos se priorizará inicialmente su configuración como memoria caché realizando pruebas de rendimiento global con diferentes repartos de la memoria de nivel 2. Una vez fijado el tamaño de la memoria de nivel 2 es necesario repetir las pruebas para diferentes configuraciones de las memorias de nivel 1. En cada uno de estos pasos se ordenarán las funciones y los *buffers* en función del interés de su ubicación en memoria interna.

5.7 Optimización a nivel de sistema

La integración del descodificador en un sistema completo implica definir una tarea en el sistema operativo y asignarle una prioridad. Para poder asignar la prioridad a las tareas del sistema ([ficha 016]) es necesario evaluar el tiempo de ejecución de cada una y la posibilidad de pérdida de datos si se demora su ejecución. El criterio a aplicar consiste en asignar mayor prioridad a aquellas tareas que requieran una atención más rápida y cuyo tiempo de ejecución sea bajo, mientras que las tareas que tengan un tiempo de ejecución elevado y puedan ser interrumpidas por el resto (como por ejemplo el descodificador de vídeo) tendrán una menor prioridad.

La inclusión del descodificador en el sistema completo afecta negativamente a sus prestaciones ya que tiene que competir con otras tareas por el empleo de la memoria interna (memoria caché y memoria de propósito general) y el controlador de

DMA. Para minimizar la influencia que esta integración tiene en su rendimiento, se deben tener en cuenta las siguientes consideraciones:

- Analizar las peticiones de transferencias de DMA que realizan el resto de elementos que componen el sistema (ver [ficha 017]). Asignar sus peticiones a las diferentes colas teniendo en cuenta las que realiza el descodificador para lograr una distribución final uniforme entre todas las colas de petición. Comprobar experimentalmente que en ningún caso se encolan simultáneamente más solicitudes que el tamaño de cada una de las colas.
- Analizar el tamaño del código y los datos del sistema, así como la pila asignada a cada una de las tareas (ver [ficha 018]). Favorecer que en la memoria interna se ubiquen las pilas de las tareas (normalmente tienen un tamaño reducido), junto con el código y los datos del descodificador de vídeo que se hayan determinado en las etapas anteriores.
- Cuando la carga computacional media del descodificador es próxima al 100% de la CPU, no es suficiente con obtener el valor medio del tiempo de descodificación de una secuencia, sino que es necesario analizar el tiempo de descodificación de cada una de las imágenes (ver [ficha 027]). Si este tiempo supera el intervalo de presentación de una imagen (40 ms), es necesario definir un *buffer* de imágenes descodificadas en el que se almacenan aquellas que ya han sido procesadas. El tamaño del *buffer* debe ajustarse para garantizar que el tiempo que se invierte en descodificar el número de imágenes que se pueden alojar en él, sea siempre menor que el tiempo de presentación de todas ellas.

5.8 Metodología propuesta

En este apartado se desarrolla una versión resumida de la metodología de optimización de descodificadores de video para procesadores digitales de señal que se ha explicado en los apartados anteriores. En esta versión se explican los siete pasos que se propone llevar a cabo de forma secuencial para realizar la optimización, pero no se justifican como se hace en los apartados anteriores.

5.8.1 Elección del código de referencia

Antes de comenzar el proceso de optimización es necesario disponer de una implementación funcionalmente correcta del algoritmo (en este caso descodificador de vídeo) que se desee implementar. Normalmente estas implementaciones se han desarrollado para entornos PC por lo que las primeras pruebas se realizarán en dicho entorno. En el caso de que existan varias opciones, se debe comprobar cuales de ellas soportan los perfiles y niveles que se desean implementar empleando para ello las secuencias definidas en el proceso de estandarización.

Los descodificadores disponibles deben portarse al entorno de desarrollo del DSP que se haya seleccionado como plataforma objetivo. Esta migración debe realizarse en varios pasos: importar todos los ficheros fuentes al entorno de desarrollo, definir las opciones de compilación y configurar el sistema operativo en tiempo real. Para configurar el sistema operativo, se recomienda definir toda la memoria interna como memoria caché por lo que todo el código y los datos serán ubicados, inicialmente, en memoria externa. En el proceso de configuración del RTOS es necesario asignar inicialmente un tamaño elevado tanto a la pila (1.5 MB para H.264) como a la *heap* del sistema (1 MB para H.264). En etapas posteriores se ajustarán estos tamaños.

Seguidamente se realizan de nuevo pruebas de conformidad en el entorno DSP para localizar posibles errores que aparezcan en tiempo de ejecución. Los errores más comunes tienen que ver con las lecturas/escrituras no alineadas en memoria, por lo que es necesario revisar el código en busca de este tipo de acceso a los datos.

Finalmente, para tomar una decisión respecto al código que debe elegirse como punto de partida hay que analizar las prestaciones de los candidatos en términos de tiempo invertido en la descodificación de las secuencias de referencia. El código que tenga unas mejores prestaciones será seleccionado para comenzar el proceso de optimización. Las medidas de rendimiento obtenidas serán tomadas como referencia para todo el proceso posterior.

5.8.2 Optimización de funciones con elevada carga computacional

El primer paso del proceso de optimización propiamente dicho consiste en la recodificación de las funciones que poseen una mayor carga computacional. El criterio principal para seleccionar las funciones a optimizar se basa en elegir aquellas que hayan empleado un mayor número de ciclos de reloj para ejecutarse tras la descodificación de una secuencia, sin tener en cuenta los fallos en los accesos a los diferentes niveles de memoria caché.

Antes de recodificar una función es necesario comprobar si su funcionalidad está implementada y optimizada en alguna función de librería. En este caso será necesario adaptar los parámetros de entrada y salida de la función para emplear dicha librería.

Para aprovechar la arquitectura SIMD se recodifican las funciones empleando instrucciones en ensamblador (*intrinsics*). De forma general, las funciones se pueden agrupar en dos categorías: aquellas en las que predomina el procesamiento de datos mediante bucles y en aquellas en las que predomina la toma de decisiones en función de las condiciones que se vayan cumpliendo.

Para el primer tipo de funciones las técnicas a aplicar se basan en: emplear instrucciones de carga, promedio y almacenamiento, operar con varios píxeles/coeficientes en paralelo, utilizar instrucciones de empaquetado y desempaquetado de datos para adaptar el número de bits que poseen los operandos y aprovechar las instrucciones de multiplicación y acumulación para realizar operaciones de interpolación y filtrado de píxeles. Para el segundo tipo, la técnica a aplicar se basa en pre-calcular las condiciones, obtener todos los posibles valores intermedios y asignar los valores de salida en función de las condiciones calculas anteriormente.

Finalmente, si alguna función tiene un tamaño reducido y se ejecuta intensivamente a lo largo del proceso de descodificación, esta puede ser reescrita directamente en ensamblador y paralelizada manualmente.

5.8.3 Optimización del tiempo de acceso a los datos alojados en memoria externa

Para reducir el tiempo de acceso a los *buffers* que se encuentran alojados en memoria externa se pueden emplear dos técnicas. La elección de la técnica a aplicar depende fundamentalmente del tamaño de los *buffers*. Si tienen un tamaño reducido (por ejemplo los que alojan macrobloques), éstos pueden alojarse directamente en memoria interna. Si por el contrario tienen un tamaño elevado (por ejemplo los que almacenan imágenes descodificadas) se debe emplear el controlador de DMA para transferir la información necesaria en cada momento a *buffers* alojados en memoria interna mientras la CPU ejecuta otras partes del algoritmo.

Para el empleo del controlador de DMA, se han identificado una serie de recomendaciones generales independientes tanto del descodificador como del DSP a emplear:

1. Analizar todos los píxeles implicados en el proceso de compensación de movimiento y transferirlos a *buffers* intermedios alojados en memoria interna configurando el controlador de DMA.

2. Definir los *buffers* intermedios como *buffers* ping-pong para paralelizar el procesamiento de unos datos por parte de la CPU con el movimiento de otros empleando el controlador de DMA.

3. Forzar que el tamaño de las transferencias sea múltiplo de 32 bits y alinear las direcciones de origen y destino a este mismo tamaño.

4. Distribuir las peticiones de transferencia entre las colas de petición del controlador de DMA para evitar la sobrecarga de alguna de ellas.

Adicionalmente, hay otras recomendaciones que no pueden generalizarse a todos los descodificadores o a todos los procesadores. Por una parte, si los macrobloques se pueden dividir en bloques más pequeños, como ocurre en H.264, es recomendable definir un único *buffer* en memoria interna en el que se almacenarán todos píxeles de todos los bloques empleados en la compensación de movimiento, evitando de este modo tener que definir multitud de *buffers* para todas las combinaciones posibles. Por otro lado, si el DSP posee dos controladores de DMA, se aprovecharán ambos haciendo que uno de ellos configure automáticamente las transferencias del otro, si esto es posible.

En relación con la gestión de la memoria interna, hay que tener en cuenta que es necesario almacenar en ella tanto los *buffers* intermedios empleados por el controlador de DMA como algunos datos o *buffers* empleados por el resto de funciones. El criterio que debe aplicarse para seleccionar los *buffers* de propósito general que se alojarán en memoria interna se basa en los fallos en el acceso a las memorias caché de datos que tengan las funciones que los emplean. A partir de esta información se genera una lista priorizada de los *buffers*, que se utilizará para incluirlos en la memoria interna hasta agotar su capacidad.

El tamaño de todos estos datos condiciona el espacio mínimo que hay que definir en la memoria interna para almacenar datos de propósito general. Antes de realizar una distribución definitiva de la memoria interna entre memoria caché y memoria de propósito general es necesario analizar el rendimiento del descodificador con diversas configuraciones. La que obtenga mejores resultados será seleccionada para continuar el proceso de optimización.

Para mejorar las prestaciones del descodificador, los *buffers* que se ubiquen en memoria interna deben tener un tamaño múltiplo del de la línea de caché de datos de nivel 1 y estar alineados a una dirección de memoria múltiplo de ese mismo tamaño.

5.8.4 Optimización del tiempo de acceso a las funciones alojadas en memoria externa

Algunas funciones pueden ubicarse en la memoria interna para reducir su tiempo de ejecución. Para ello se seleccionarán aquellas que tengan un mayor número de fallos en el acceso a la memoria caché de programa. Las funciones seleccionadas deben agruparse en secciones atendiendo a su funcionalidad. El tamaño de las secciones y su alineamiento debe ser múltiplo del tamaño de la caché de programa de nivel 1.

La inclusión de algunas funciones en la memoria interna implica, bien modificar la distribución de dicha memoria entre memoria de propósito general y memoria caché o bien extraer algunos *buffers* de datos de esa memoria para ubicar en su lugar las funciones. En ambos casos es necesario realizar nuevas medidas de rendimiento para seleccionar la configuración con la que se obtienen los mejores resultados.

5.8.5 Distribución del espacio de la memoria interna

En los dos pasos anteriores se trata de reducir el tiempo de acceso a los datos y al código que se encuentran en memoria externa a base de incluir los *buffers* o las funciones en memoria interna de propósito general.

Durante el proceso de optimización de las funciones y del acceso a datos y código que se encuentren en memoria externa, el tamaño de ambos (funciones y datos), se va modificando continuamente por lo que no hay que aplicar un gran esfuerzo a la distribución de las funciones/datos en la memoria durante la optimización, ya que la situación cambiará en numerosas ocasiones a lo largo del proceso. Una vez que se dispone de una versión optimizada del algoritmo es cuando se debe realizar un ajuste fino de la distribución de la memoria. En esta fase del proceso es necesario ajustar el tamaño de la pila y de la *heap* del sistema para valorar su posible inclusión en la memoria interna en función de su tamaño real.

Como criterio general, en el nivel 1 de memoria se priorizará su configuración como memoria caché realizando medidas de prestaciones con diferentes repartos de la memoria interna de nivel 2. Para cada reparto se hará una asignación de funciones y datos (incluyendo pila y *heap*) teniendo en cuenta los resultados de rendimiento que se hayan obtenido en las fases anteriores.

Una vez fijada la distribución de la memoria de nivel 2 se realizarán de nuevo pruebas para diferentes configuraciones de la memoria de nivel 1. De nuevo se ordenarán las funciones y los *buffers* en función del interés de su ubicación en memoria interna.

5.8.6 Optimización a nivel de sistema

La inclusión del descodificador en un sistema completo implica su integración en una tarea del sistema operativo. La prioridad que se asigna a cada tarea depende fundamentalmente del tiempo de ejecución de la misma, de modo que las tareas que demandan más tiempo para su ejecución deben tener una menor prioridad que las que emplean menos tiempo. Con este criterio las tareas menos prioritarias pueden ser interrumpidas por el resto, evitando de este modo la pérdida de datos de estas últimas. En el caso de los descodificadores de vídeo, la prioridad que se les debe asignar es la más baja de entre todas las tareas.

Por otra parte, cuando el descodificador se integra en un sistema completo surgen problemas a nivel de sistema que afectan a su rendimiento. Con objeto de reducir el impacto negativo en las prestaciones tras la integración del descodificador, es recomendable analizar las peticiones de transferencias de DMA de todas las tareas del sistema y realizar un reparto equitativo entre todas las colas de petición del controlador. Además, y en relación con el empleo de la memoria interna, se propone favorecer que dicha memoria se emplee para ubicar funciones y datos del descodificador.

Adicionalmente, si la carga computacional global del sistema es próxima en media al 100% aparecen algunos problemas que impiden el procesamiento en tiempo real. En estos casos es necesario almacenar varias imágenes descodificadas antes de presentarlas de modo que el tiempo de descodificación de todas ellas sea inferior al de su presentación.

5.9 Resumen

En este capítulo se ha explicado la metodología de optimización del tiempo de ejecución de descodificadores de vídeo para procesadores digitales de señal. Dicha metodología supone la principal aportación de esta tesis y se apoya en los resultados obtenidos en el proceso de optimización de diferentes descodificadores sobre varios procesadores digitales de señal que se ha mostrado en el capítulo 4.

La metodología se plantea como una serie de pasos que deben realizarse de forma secuencial para minimizar el tiempo que se invierte en la optimización de algoritmos con elevada carga computacional sobre procesadores digitales de señal.

A pesar de que la metodología se ha extraído a partir de la experiencia obtenida en la optimización de descodificadores de vídeo, su marco de aplicación es más amplio y puede emplearse para reducir el tiempo de ejecución de codificadores, pudiendo ser incluso de utilidad para optimizar cualquier algoritmo que se ejecute en un procesador digital de señal que posea una elevada carga computacional.

6 RESULTADOS, APORTACIONES Y TRABAJOS FUTUROS

En este capítulo se resumen los objetivos, los resultados obtenidos, las aportaciones realizadas y las líneas de trabajo futuro a las que puede dar lugar esta tesis.

6.1 Objetivos

En el capítulo 1 se ubica la tesis dentro de las líneas de investigación del Grupo de Diseño Electrónico y Microelectrónico de la Universidad Politécnica de Madrid y se justifican sus objetivos. A modo de recordatorio, el principal objetivo planteado al comienzo de la tesis es la elaboración de una metodología de optimización del tiempo de ejecución de descodificadores de vídeo sobre tecnología DSP. Esta metodología tiene en cuenta todas las fases del desarrollo de este tipo de aplicaciones, desde la elección del *software* de referencia hasta su integración en un sistema completo. En ella se proponen una serie de recomendaciones que sistematizan los pasos a dar en este proceso y se destacan las técnicas que tienen unos mejores resultados desde el punto de vista de reducción del tiempo de ejecución de los algoritmos.

Si bien el objetivo principal de la tesis es una metodología de optimización en velocidad para descodificadores basados en DSPs, para alcanzarlo se han definido otros dos objetivos secundarios:

1. Realización de implementaciones de descodificadores compatibles con MPEG-2, MPEG-4 y H.264 funcionando en tiempo real para resolución estándar (*Standard Definition*). La implementación de descodificadores con estos requisitos de funcionamiento ha permitido identificar algunos problemas de optimización que no aparecen con resoluciones menores o con un menor número de imágenes por segundo.

2. Desarrollo de un sistema completo de descodificación de televisión IP (STB-IP) que ha permitido realizar pruebas con emisiones de televisión reales. Para realizar este sistema se han implementado todas las tareas que lo componen: análisis de la trama de transporte MPEG-2, descodificación de audio y presentación sincronizada de audio y vídeo. Este sistema ha permitido analizar la influencia que tiene en las prestaciones de los descodificadores, su

integración en un sistema complejo, con lo cual se han podido obtener conclusiones metodológicas a nivel de sistema.

6.2 Resultados y aportaciones

En este apartado se resumen los resultados alcanzados para cada uno de los tres objetivos planteados al comienzo de la tesis, así como las aportaciones más destacadas asociadas a cada uno de ellos.

6.2.1 Metodología de optimización

El empleo de los procesadores digitales de señal como soporte tecnológico para el desarrollo de aplicaciones de vídeo digital se presenta como una alternativa consolidada y flexible para poder adaptarse a los continuos cambios que vienen de la mano de los estándares de codificación de vídeo.

La implementación de codificadores o descodificadores sobre esta tecnología requiere normalmente reducir su tiempo de ejecución para lograr el funcionamiento en tiempo real. En este sentido, y tras realizar un análisis exhaustivo del estado del arte, se han localizado numerosas publicaciones en las que se proponen técnicas para reducir el tiempo de ejecución de los algoritmos de codificación o descodificación de vídeo sobre DSPs. En este sentido destacan las siguientes aportaciones:

- En [CCFS02] se realiza la recodificación de las funciones de la IDCT y la VLD del descodificador MPEG-2 en ensamblador y se emplean las instrucciones SIMD para la lectura, procesado y almacenamiento de los píxeles involucrados en la compensación de movimiento.
- En [JRGS02] se emplea el acceso directo a memoria y la técnica de doble *buffer* que permite procesar un macrobloque alojado en memoria interna, mientras que otro está siendo transferido.
- En [GSS+06] se analiza la influencia que tiene en las prestaciones de un descodificador MPEG-4, el reparto de la memoria interna, entre memoria caché y memoria de propósito general.
- En [WHL06] se explica la optimización del cálculo de las muestras intermedias para el filtro de ¼ píxel y de la función que obtiene el SAD en un codificador H.264.
- En [YGLZ06] se describe la técnica que permite simplificar el cálculo del filtro antibloques precalculando las condiciones que permiten decidir los diferentes tipos de filtrado.

Sin embargo, y a pesar del elevado número de publicaciones analizadas, no se ha localizado ninguna en la que se proponga una metodología completa de optimización que cubra todas las fases del diseño, desde la selección del *software* que se empleará como punto de partida hasta la integración en un sistema completo de descodificación.

La elaboración de una metodología que facilite este proceso ha sido el principal resultado de esta tesis. La metodología que se ha elaborado recoge las técnicas de optimización descritas en otras publicaciones, propone otras novedosas y prioriza su orden de aplicación.

En esta metodología se proponen una serie de pasos que deben aplicarse secuencialmente para facilitar la optimización en velocidad de los algoritmos que se ejecutan en un DSP y se destacan las técnicas que obtienen unos mejores resultados

desde el punto de vista de las prestaciones del algoritmo. Además, incluye algunas técnicas relacionadas con la integración de los descodificadores en sistemas completos, que no se han encontrado en otros trabajos.

Debido a la ausencia de publicaciones en las que se cubra todo el proceso de optimización, no ha sido posible comparar la metodología propuesta con otras planteadas en trabajos similares.

6.2.2 Implementación de descodificadores con requisitos de tiempo real

La metodología propuesta se ha elaborado a partir de la experiencia adquirida tras la optimización de tres descodificadores de vídeo compatibles con los estándares MPEG-2, MPEG-4 y H.264, respectivamente. En todos los casos se ha seleccionado un *software* de referencia que, una vez portado al entorno de desarrollo del DSP, ha sido optimizado empleando las técnicas propuestas en la metodología.

Para valorar la reducción del número de ciclos de reloj que provoca la aplicación de cada una de las técnicas, se ha elaborado un banco de pruebas que permite analizar el comportamiento del descodificador aisladamente. Como se muestra en la Figura 6-1, el banco de test en el que se ha integrado cada descodificador permite leer las secuencias codificadas desde un fichero, descodificarlas y almacenar el resultado en un nuevo fichero. El objetivo de estas pruebas es analizar el rendimiento del descodificador, sin tener en cuenta el resto de elementos que componen un sistema completo de distribución de televisión digital.

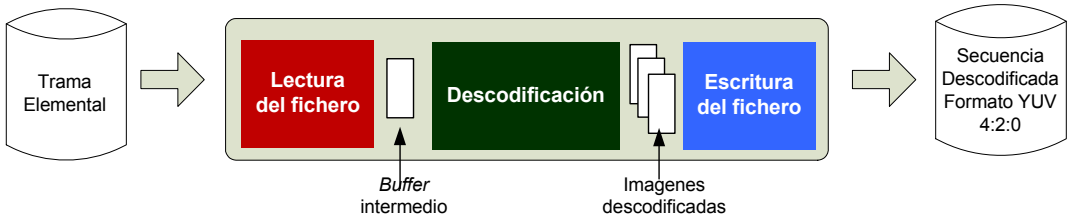


Figura 6-1 Banco de pruebas empleado para medir el rendimiento de los descodificadores aisladamente.

En los siguientes apartados se resumen los resultados obtenidos para cada uno de los descodificadores, comparándolos, en los casos en los que ha sido posible, con otras implementaciones presentadas en publicaciones científicas.

6.2.2.1 Descodificador compatible con el estándar MPEG-2

La optimización del descodificador MPEG-2 tiene como punto de partida el *software* de referencia desarrollado dentro del proceso de estandarización dado que en el momento de comenzar los trabajos relacionados con este estándar no existía otra implementación de la que se dispusiera del código fuente.

Las técnicas que se han aplicado para reducir su tiempo de ejecución han permitido pasar de una versión inicial en la que se descodificaban 10.5 imágenes por segundo de una secuencia extraída de un canal de televisión digital, a una versión optimizada capaz de procesar 116.9 imágenes por segundo. La Figura 6-2 muestra la evolución del número de imágenes por segundo procesadas por el descodificador MPEG-2 tras algunos de los pasos del proceso de optimización descritos en la metodología propuesta.

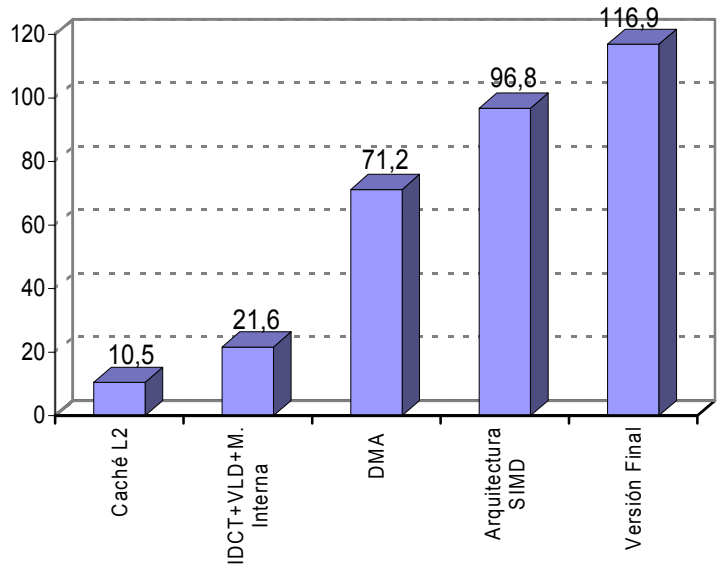


Figura 6-2. Número de imágenes por segundo procesadas por el descodificador MPEG-2 empleando el TMS320DM642 funcionando a 600 MHz tras aplicar los pasos del proceso de optimización.

En la Tabla 6-1 se comparan los resultados obtenidos por otras implementaciones publicadas. La primera fila de la tabla representa el número medio de ciclos reloj invertidos por el procesador para descodificar cada imagen de una secuencia; mientras que la segunda, muestra el número de imágenes por segundo descodificadas por cada implementación.

La comparación que se ha realizado es aproximada puesto que no se dispone de las secuencias empleadas por cada uno de los autores para obtener los datos presentados. Como se aprecia, el descodificador optimizado en el marco de esta tesis mejora los resultados alcanzados por otros autores.

	[PSG+06]	[SH98]	[ARG03]	[CCFS02]
Ciclos CLK por imagen	5.13×10^6	6.52×10^6	$> 6.00 \times 10^6$	7.27×10^6
Imágenes por segundo	116.9	92	>100[151]	82.5

Tabla 6-1. Comparación de las prestaciones (ciclos de reloj invertidos por la CPU e imágenes por segundo) del descodificador MPEG-2 optimizado desarrollado en esta tesis, con otros presentados en publicaciones científicas.

6.2.2.2 Descodificador compatible con el estándar MPEG-4

En el caso del descodificador compatible con el estándar MPEG-4, el punto de partida fue el *software* desarrollado dentro del proyecto FFMPEG. Durante el desarrollo de los trabajos de esta tesis, se decidió aplicar un esfuerzo limitado para la optimización de este descodificador por su escasa implantación industrial. No obstante, se han empleado buena parte de las técnicas descritas en la metodología para su optimización.

[151] En este caso se muestra una referencia aproximada puesto que en el artículo correspondiente no se dan más detalles.

Como resultado del proceso de optimización se ha logrado el funcionamiento en tiempo real para secuencias codificadas con el perfil ASP y un régimen binario de 3 Mbps procedentes de un DVD. La Figura 6-3 muestra la evolución en el número de imágenes por segundo procesadas por un TMS320DM642 tras varias etapas del proceso de optimización[152] para una secuencia de la película "La guerra de las galaxias". Como se aprecia, se ha obtenido el funcionamiento en tiempo real holgadamente, lo que permite incluir el descodificador en un sistema en el que se realicen otras tareas.

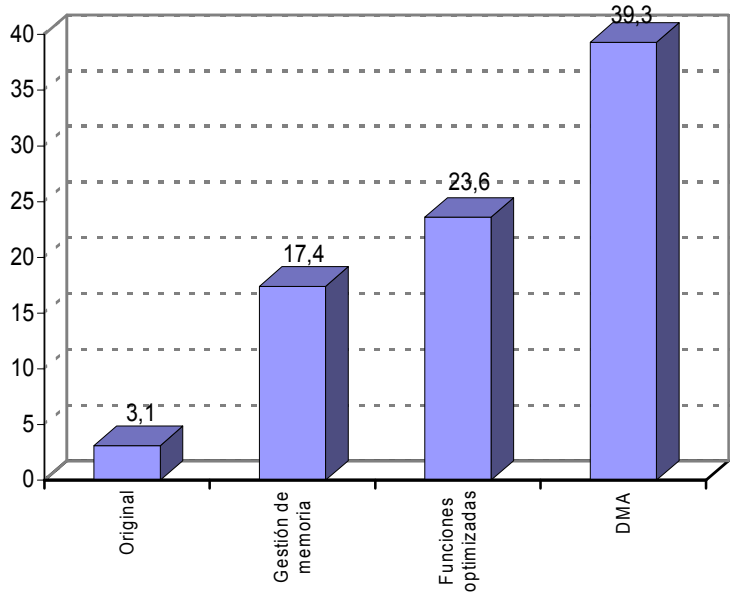


Figura 6-3. Número de imágenes por segundo procesadas por el descodificador MPEG-4 empleando el TMS320DM642 funcionando a 720 MHz.

En la Tabla 6-2 se tratan de comparar los resultados de la implementación realizada en esta tesis con otras dos encontradas en publicaciones científicas. Las dos primeras columnas se corresponden con las prestaciones del descodificador optimizado en el marco de esta tesis, empleando el procesador TMS320DM642 funcionando a 720 MHz para dos secuencias codificadas con el perfil ASP, 3 Mbps de régimen binario y resolución estándar. La tercera columna se corresponde con una implementación sobre un TMS320DM642 funcionando a 600 MHz que procesa secuencias codificadas con el perfil SP y de tamaño 4CIF[153]. Finalmente, la última columna muestra los resultados obtenidos con el procesador Trimedia descodificando secuencias con el perfil SP de tamaño 4CIF. Como se aprecia, las condiciones con las que se han obtenido los datos son muy diferentes ya que, tanto los perfiles, como el tamaño de las imágenes son distintos. Además, en los trabajos con los que se comparan las prestaciones del descodificador optimizado en esta tesis, no se indican las características de las secuencias que se han empleado.

[152] Estos datos no se encuentran disponibles en los experimentos que se presentan en el capítulo 4 por lo que se han obtenido interpolando la información que aparece en la Tabla 4-11 y en la Figura 4-32.

[153] En el artículo correspondiente a este trabajo los datos que se presentan se basan en la descodificación de secuencias CIF. Para poder realizar una comparación con el resto de resultados se ha realizado una extrapolación multiplicando el número de ciclos de reloj por 4.

	DM642@720MHz DVD (SD) [PGS+06]	DM642@720MHz Canal TV (SD) [PGS+06]	DM642@600MHz (4CIF) [GSS+06]	Trimedia (4CIF) [KHH+03]
Ciclos CLK por imagen	$18.3 \cdot 10^6$	$14.4 \cdot 10^6$	$9.0 \cdot 10^6$	--
Imágenes por segundo	39.3	50.0	66.6	73.0

Tabla 6-2. Número de ciclos de reloj empleados en media e imágenes por segundo procesadas por varias implementaciones de un descodificador MPEG-4.

Analizado los resultados mostrados en la tabla se aprecia que las prestaciones del descodificador optimizado en el marco de esta tesis son inferiores al resto; la justificación a este hecho se encuentra fundamentalmente en que el perfil empleado en esta implementación (ASP) es más exigente en términos de carga computacional que el perfil empleado en las otras implementaciones (SP) y que el esfuerzo de optimización de este descodificador ha sido menor que el aplicado al resto, como se explicó en el apartado 4.3.

6.2.2.3 Descodificador compatible con el estándar H.264

Como punto de partida para el proceso de optimización del descodificador H.264 se seleccionó como *software* de referencia el elaborado dentro del proyecto FFMPEG. Debido a la elevada carga computacional demandada por este descodificador, fue necesario aplicar algunas técnicas de optimización diferentes a las utilizadas para los descodificadores MPEG-2 y MPEG-4.

La Tabla 6-3 muestra el número de imágenes por segundo procesadas para secuencias codificadas conforme a los perfiles BP y MP con un régimen binario de 2 Mbps[154] y resolución estándar. Las dos primeras filas muestran los resultados para un TMS320DM642 funcionando a 600 MHz y 720 MHz respectivamente. Por otra parte, la última fila presenta los resultados obtenidos para el procesador TMS320DM6437 trabajando a 594 MHz. Como se aprecia, se ha logrado el funcionamiento en tiempo real para ambos perfiles con los dos procesadores utilizados.

En el caso del descodificador H.264, no se ha encontrado en la bibliografía ninguna referencia con la que comparar los resultados obtenidos en esta tesis.

	Nemo		Guerra Galaxias		Fútbol		Nemo1M	
	BP	MP	BP	MP	BP	MP	BP	MP
TMS320DM642 @720 MHz	43.4	32.6	45.0	33.2	44.4	33.5	54.6	35.3
TMS320DM642 @600 MHz	36.1	27.1	37.5	27.7	37.0	27.9	45.5	29.4
TMS320DM6437 @594 MHz	42.5	30.8	44.2	30.2	43.6	30.5	49.0	33.9

Tabla 6-3. Número de imágenes por segundo procesadas por el descodificador H.264 empleando tanto un TMS320DM642 funcionando a 600 MHz y a 720 MHz como un TMS320DM6437 a 594 MHz.

[154] A excepción de la secuencia indicada en la última columna (Nemo1M) que se ha codificado con 1 Mbps para analizar la influencia del régimen binario en las prestaciones del descodificador.

6.2.3 Sistema completo de descodificación IP

El otro objetivo secundario de la tesis era el desarrollo de un sistema completo de recepción de televisión digital IP (STB-IP) que pusiera de manifiesto los problemas de integración de los descodificadores implementados, con otros elementos del sistema. Como resultado se dispone de un sistema completo en el que es posible integrar diferentes descodificadores con el que realizar pruebas de larga duración. Para realizar este sistema se han implementado varios descodificadores de audio (MPEG-1 Capas II y III, AAC y AC3), un analizador de la trama de transporte MPEG-2 y un algoritmo de sincronización de audio y vídeo.

El STB-IP realizado se ha integrado en dos bancos de pruebas: el primero permite validar los resultados obtenidos con el descodificador aislado. La Figura 6-4 muestra el banco de pruebas en el que un ordenador personal transmite ficheros que contienen tramas de transporte MPEG-2 previamente grabadas. El STB-IP desencapsula las tramas elementales de audio y vídeo, las descodifica y las muestra en un televisor.

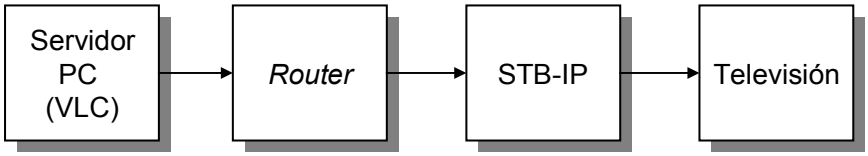


Figura 6-4 Banco de pruebas empleado para medir el rendimiento de los descodificadores de vídeo integrados en un STB-IP.

El segundo banco de pruebas permite trabajar con entornos reales de difusión de televisión digital y tiene como objeto analizar el funcionamiento del descodificador en situaciones reales, en las que el sistema completo en el que se encuentra integrado recibe tramas de larga duración procedentes de una transmisión vía satélite o de un codificador de vídeo en tiempo real. Con este entorno se pretende poner de manifiesto problemas que pueden surgir a nivel de sistema cuando se descodifican secuencias largas y que no aparecen cuando se realizan las pruebas con pequeños fragmentos de vídeo.

Se han diseñado dos configuraciones:

1. En la primera, un reproductor de DVD genera la salida analógica que es conectada a un codificador que genera, en tiempo real, una trama de transporte MPEG-2 que es distribuida a través de la red Ethernet mediante paquetes IP. La Figura 6-5 muestra el diagrama de bloques de esta configuración y la Figura 6-6 una imagen del banco de pruebas implementado.

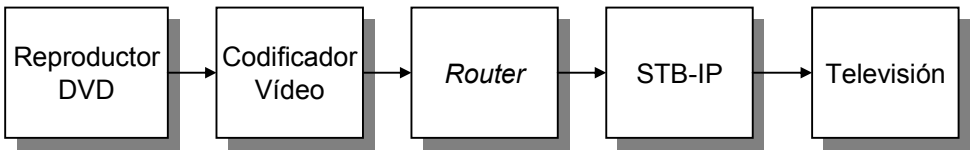


Figura 6-5 Banco de pruebas empleando un reproductor de DVD.

Figura 6-6 Imagen del banco de pruebas empleando un reproductor de DVD.

2. En la segunda, se emplea un *gateway* comercial que sintoniza un transpondedor de satélite y encapsula cada uno de los programas disponibles en una trama de transporte MPEG-2, que es transmitida a través de una dirección IP. La trama de transporte generada es recibida por un transcodificador que recodifica la trama elemental de vídeo de acuerdo al estándar que se desee y vuelve a encapsularla en una nueva trama de transporte MPEG-2, que es redistribuida a través de la red local. La Figura 6-7 presenta el diagrama de bloques de la configuración indicada y la Figura 6-8 una imagen del sistema implementado.

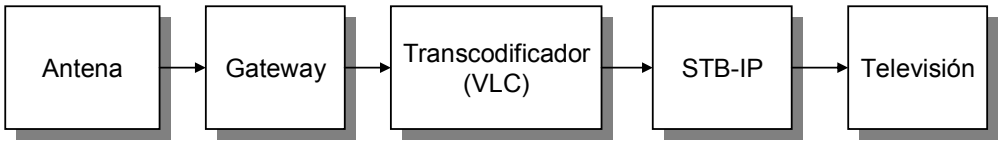


Figura 6-7 Banco de pruebas empleando una transmisión vía satélite.

Figura 6-8 Aspecto del banco de pruebas basado en un *gateway* comercial.

Para cada uno de los descodificadores que se han integrado en el STB-IP se han realizado medidas de la carga computacional asociada a las tareas que lo componen. La Tabla 6-4 muestra el tanto por ciento de CPU empleando por cada una de las 5 tareas principales del sistema. Además, se incluye una columna que indica el porcentaje de CPU empleado por otras tareas incluidas en el sistema (columna "Otros") y otra en la que se muestra la carga computacional libre disponible para realizar otras tareas (columna "Libre").

		MPEG TS	Desc. Vídeo	Desc. Audio	Presenta. Vídeo	Presenta. Audio	Otros	Libre
MPEG-2 DM642 600 MHz	**La Guerra de las Galaxias**	4.9%	32.8%	2.8%	0.1%	0.7%	11.3%	47.4%
	EuroSport	4.1%	28.5%	2.0%	0.1%	0.7%	11.3%	53.3%
MPEG-4 DM642 720 MHz	**Nemo**	4.7%	78.1%	2.8%	0.1%	0.7%	2.1%	11.5%
	Arte	4.6%	61.5%	2.0%	0.1%	0.7%	2.1%	29.0%
H.264 BP DM642 600 MHz	**Nemo**	3.2%	79.9%	2.0%	1.4%		7.5%	6.0%
	La Guerra de las Galaxias	3.1%	77.5%	2.0%	1.6%		7.4%	8.4%
H.264 MP DM642 720 MHz	**Nemo**	3.2%	85.7%	2.0%	1.4%		7.5%	0.2%
	La Guerra de las Galaxias	3.1%	84.3%	2.0%	1.6%		7.5%	1.6%
H.264 BP DM6437 594 MHz	**Nemo**	3.0%	58.8%	1.9%	2.4%		6.5%	27.4%
	La Guerra de las Galaxias	3.1%	56.5%	2.1%	2.3%		6.4%	29.6%
H.264 MP DM6437 594 MHz	**Nemo**	3.0%	81.2%	1.9%	2.4%		6.2%	5.3%
	La Guerra de las Galaxias	3.1%	82.6%	2.1%	2.5%		6.5%	3.2%

Tabla 6-4. Porcentaje de uso de CPU de cada una de las tareas que componen el STB-IP empleando diferentes estándares de codificación.

6.3 Trabajos publicados en relación con la tesis

Los resultados obtenidos en esta tesis han dado lugar a cinco publicaciones en revistas (dos de ellas aceptadas para su publicación en el volumen de Mayo de 2011) y diez comunicaciones en congresos internacionales. Las publicaciones en revistas se resumen en los siguientes puntos:

- Fernando Pescador, César Sanz, Matías J. Garrido, Carlos Santos, Rafael Antoniello. "A DSP based IP Set-Top Box for Home Entertainment". *IEEE Transactions on Consumer Electronics*, vol. 52, no. 1, pp. 254-262. February 2006. [PSG+06].

  En este trabajo se presenta el proceso de optimización de un descodificador MPEG-2 y la integración del mismo en un sistema completo de recepción de televisión digital basado en el procesador digital de señal TMS320DM642. El sistema completo logra el funcionamiento en tiempo real para definición estándar (SD) dejando disponible más de un 50% de la CPU para otras tareas.

- Fernando Pescador, César Sanz, Matías J. Garrido, Eduardo Juárez, David Samper. "A DSP Based H.264 Decoder for a Multi-Format IP Set-Top Box". *IEEE Transactions on Consumer Electronics,* vol. 54, no. 1, pp. 145-153. February 2008. [PSG+08].

  En este artículo se presenta el proceso de optimización de un descodificador H.264 para el procesador TMS320DM642, los resultados obtenidos y la integración en el sistema completo de recepción de televisión digital. El sistema desarrollado logra el funcionamiento en tiempo real para definición estándar.

- Fernando Pescador, Gonzalo Maturana, Matías J. Garrido, Eduardo Juárez, César Sanz "An H.264 Video Decoder Based on a latest generation DSP". *IEEE Transactions on Consumer Electronics*, vol. 54, no. 1, pp. 205-212. February 2009. [PMG+09].

  Esta tercera referencia presenta la adaptación del sistema completo al procesador TMS320DM6437 y las mejoras en el rendimiento obtenidas gracias a la nueva arquitectura de este procesador.

- Fernando Pescador, Eduardo Juárez, Michael Raulet, César Sanz "A DSP Based H.264/SVC Decoder for a Multimedia Terminal" *IEEE Transactions on Consumer Electronics*. Aceptada para su publicación en el volumen de Mayo de 2011. [PJR+11].

  Este trabajo describe los resultados obtenidos tras aplicar la metodología propuesta en esta tesis a la optimización de un descodificador escalable H.264/SVC [OSVC09] para una arquitectura C64+. La optimización realizada permite obtener en funcionamiento en tiempo real para secuencias con seis capas y una resolución espacial máxima CIF.

- Pedro J. Lobo, Eduardo Juárez, Fernando Pescador, Gonzalo Maturana, M. César Rodríguez. "A DVB-H receiver and gateway implementation on a FPGA- and DSP-based platform". *IEEE Transactions on Consumer Electronics*. Aceptada para su publicación en el volumen de Mayo de 2011. [LJP+11].

  En este último artículo se describen los resultados logrados tras aplicar la metodología de optimización a una aplicación que no es, ni un codificador, ni un descodificador de vídeo. Se trata de un *gateway* DVB-H desarrollado dentro del grupo de investigación en el que se ha elaborado esta tesis.

Con relación a las comunicaciones en congresos, en el primero de los artículos se presenta la arquitectura *software* del sistema completo de descodificación de televisión digital y los mecanismos de sincronización entre las tareas que forman el sistema; en el segundo de ellos se muestran los resultados obtenidos tras el proceso de optimización del analizador de trama de transporte MPEG-2; en los dos siguientes se presentan respectivamente el proceso de optimización del descodificador MPEG-4 y su integración en el sistema completo; el quinto de los artículos explica el mecanismo de sincronización de audio y vídeo que se ha implementado en el sistema completo; los dos siguientes trabajos describen el proceso de optimización de un descodificador H.264 compatible con los perfiles *Baseline* y *Main* respectivamente. Finalmente, los tres últimos se corresponden con versiones reducidas de los tres primeros artículos publicados en revistas descritos anteriormente.

- Fernando Pescador, Matías J. Garrido, Rafael Antoniello, Carlos Santos, César Sanz. “A DSP based multiformat video decoder for an IP set-top box”. *EDERS2006*. Munich (Alemania). April 2006. [PGA06].
- Fernando Pescador, Matías J. Garrido, César Sanz, Eduardo Juárez, Ángel M. Groba, Francisco J. Sánchez. “An MPEG2 TS Parser for a DSP based Multi-format IP Set-top Box”. *Conference on Design of Circuits and Integrated Systems DCIS 2007*. 21-23 November 2007. [PGS$^+$07c].
- Fernando Pescador, Matías J. Garrido, César Sanz, Eduardo Juárez, David Samper, Rafael Antoniello. “Multiformat decoder for a DSP-based IP Set-Top Box”. *SPIE International Symposium on Microtechnologies for the New Millennium. VLSI Circuits and Systems Conference*. Las Palmas de Gran Canaria. May 2007. [PGS$^+$07d].
- Fernando Pescador, Matías J. Garrido, César Sanz, Eduardo Juárez, David Samper, Rafael Antoniello. “MPEG-4 SP/ASP decoder for a DSP-based Multi-Format IP Set-Top Box”. *Annual Conference of the IEEE Industrial Electronics Society IECON 2006*. November 2006. [PGS$^+$06].
- Esther Estevez, David Samper, Fernando Pescador, Matías J. Garrido, César Sanz. "Implementation of a media synchronization algorithm for multistandard IP set-top box systems". *SPIE International Symposium on Microtechnologies for the New Millennium. VLSI Circuits and Systems Conference*. Dresde, Alemania, 4-6 May 2009. [ESP$^+$09].
- Fernando Pescador, Matías J. Garrido, César Sanz, Eduardo Juárez, David Samper, Ángel Groba. “Real-time H.264 BP decoder based on a DM642 DSP”. *IEEE International Conference on Signal Processing and Communications ICSPC 2007*. Dubai, Emiratos Árabes Unidos, 24-27 November 2007. [PGS$^+$07].
- Fernando Pescador, Matías J. Garrido, César Sanz, Eduardo Juárez, M. César Rodríguez, David Samper. “A Real-Time H.264 MP Decoder Based on a DM642 DSP”. *14th IEEE International Conference on Electronics, Circuits and Systems ICECS 2007*. Marrakech, Marruecos, 11-14 December 2007. [PGS$^+$07b].
- Fernando Pescador, César Sanz, Matías J. Garrido, Carlos Santos, Rafael Antoniello, Javier Iglesias. “A DSP based IP Set-Top Box for Home Entertainment”. *International Conference on Consumer Electronics ICCE 2006*, pp. 271-272. 7-10 January 2006. [PSG$^+$06b].
- Fernando Pescador, Matías J. Garrido, César Sanz, Eduardo Juárez, David Samper. “A DSP Based H.264 Decoder for a Multi-Format IP Set-Top Box”. *International Conference on Consumer Electronics ICCE 2008,* no. 1, pp. 1-2. 9-13 January 2008. [PGS$^+$08].

- Fernando Pescador, Gonzalo Maturana, Matías J. Garrido, Eduardo Juárez, César Sanz *"*An H.264 video decoder based on a DM6437 DSP". *International Conference on Consumer Electronics ICCE 2009,* pp. 263-264. January 2009. [PMG+09b].

Adicionalmente, se han publicado otros siete artículos en congresos internacionales en los que se muestra la aplicación de la metodología propuesta a estándares de codificación más modernos y a otro tipo de aplicaciones. En el primero de ellos se comparan las prestaciones de dos descodificadores compatibles con H.264/SVC ejecutándose en una plataforma basada en DSP y se muestran los resultados obtenidos tras aplicar la metodología de optimización a uno de ellos. En el segundo, se describen las prestaciones del descodificador escalable integrado en un sistema completo. Los dos siguientes se centran en el consumo de energía del descodificador escalable optimizado en función de las capas descodificadas y se presenta un modelo para evaluar la calidad objetiva de las diferentes capas. El quinto trabajo describe el proceso de optimización de un *gateway* DVB-H implementado sobre un DSP. Finalmente, los dos últimos se corresponden con versiones reducidas de los últimos artículos publicados en revistas descritos anteriormente [PJR+11] [LJP+11].

- Fernando Pescador, Mederic Blestel, Eduardo Juárez, Mickael Raulet, Matías J. Garrido. "H.264/SVC decoder performance comparison for DSP-Based consumer electronic applications" *IEEE International Symposium on Consumer Electronics ISCE 2011*. Singapore, 14-17 June 2011. [PBJ+11]. Este trabajo ha sido merecedor del premio "*Best Paper Award*" al mejor trabajo presentado en el congreso, otorgado por un jurado de expertos propuesto por el comité organizador.
- Fernando Pescador, David Samper, Matías J. Garrido, Eduardo Juárez y Mederic Blestel. "A DSP based SVC IP STB using Open SVC Decoder". *IEEE International Symposium on Consumer Electronics ISCE 2010*. Braunschweig, Alemania, 7-10 June 2010. [PSG+10].
- Eduardo Juárez, Fernando Pescador, Pedro J. Lobo, Ángel Groba, César Sanz. "Distortion-Energy analysis of an OMAP-Based H.264/SVC Decoder". *6th International Mobile Multimedia Communications Conference MobiMedia 2010*. Lisboa, Portugal, 6-8 September 2010. [JPL+10].
- Fernando Pescador, Eduardo Juárez, David Samper, César Sanz, Mickael Raulet. "A Test Bench for Distortion-Energy Optimization of a DSP-Based H.264/SVC Decoder". *13th Euromicro Conference on Digital System Design DSD 2010*. Lille, Francia, 1-3 September 2010. [PJS+10].
- Pedro J. Lobo, Fernando Pescador, Ernesto Seisdedos, David Samper, César Sanz. "Implementation of a DSP based DVB-H gateway". *XXV Conference on Design of Circuits and Integrated Systems DCIS 2010*. Lanzarote, España, 17–19 November 2010. [LPS+10].
- Fernando Pescador, David Samper, Eduardo Juárez, Mickael Raulet, César Sanz "A DSP Based H.264/SVC Decoder for a Multimedia Terminal" *IEEE International Conference on Consumer Electronics ICCE 2011*. Las Vegas, EEUU, 9-12 January 2011. [PSJ+11].
- Fernando Pescador, Pedro J. Lobo, Ernesto Seisdedos, Eduardo Juárez. Matías J. Garrido. "Real Time DVB-H Gateway Based on DSP". *International Conference on Consumer Electronics ICCE 2011*. Las Vegas, EEUU, 9-12 January 2011. [PLS+11].

6.4 Otros resultados

Como resultado de los trabajos desarrollados en la última etapa de la tesis y que abren las líneas de trabajo futuras que se describen en el apartado siguiente, se ha establecido una estrecha colaboración internacional con el INSA (*Institut National des Sciences Appliquées*) de la Universidad de Rennes (Francia), lideres del proyecto "*Open SVC Decoder*" [OSVC09]. Esta colaboración se ha traducido en las siguientes acciones:

- Participación activa en el desarrollo del proyecto "*Open SVC Decoder*" con contribuciones en diferentes módulos y con la responsabilidad de incluir en el repositorio del proyecto las actualizaciones relacionadas con la optimización del descodificador para procesadores digitales de señal.
- Realización de publicaciones conjuntas: [PJR+11], [PBJ+11], [PSG+10], [PJS+10], [PSJ+11] y [PLS+11].
- Estancias de investigadores de INSA en UPM y viceversa para la impartición de seminarios de investigación. Más concretamente el Dr. Mickael Raulet impartió el seminario denominado "*Reconfigurable Video Coding*" en Abril de 2011 dentro del Master de Ingeniería de Sistemas y Servicios para la Sociedad de la Información de la UPM. Por otro lado, el autor de la tesis impartió el seminario "*Optimization of Video Decoding Algorithms for Digital Signal Processors*" en Febrero de 2011 en la Universidad de Rennes dentro del último curso del grado en la asignatura de "Procesadores Digitales de Señal".
- Firma de un acuerdo Erasmus entre ambas Universidades para el intercambio de estudiantes y profesores.

6.5 Líneas de trabajo futuro

En este apartado se enumeran una serie de tareas de indudable interés que, o ya se han puesto en marcha, como resultado de los trabajos de esta tesis, o podrían ser abordadas en el futuro. En primer lugar se relacionan aquellas que suponen la aplicación de la metodología propuesta en esta tesis a otros codificadores o descodificadores de vídeo o aplicaciones en las que puede ser de interés. Posteriormente se proponen otras líneas de trabajo que implicarían la ampliación de la metodología propuesta.

Los trabajos relacionados con la aplicación de la metodología son:

- Aplicación de la metodología propuesta a un descodificador de vídeo escalable. En 2007, la ITU publicó el anexo G del estándar H.264 en el que se define la trama de salida de un codificador escalable [ITU07]. Este estándar incluye algunas herramientas que suponen un reto desde el punto de vista de las metodologías de optimización como pueden ser la predicción *intra layer* o el filtrado de *upsampling.*

  En la actualidad existen dos implementaciones de código abierto de descodificadores compatibles con este estándar: el *software* de referencia desarrollado durante el proceso de estandarización y el desarrollado por la Universidad de Rennes (Francia) [OSVC09] denominado "*Open SVC decoder*". En 2009 se comenzó a aplicar la metodología propuesta para seleccionar uno de los descodificadores disponibles y realizar el proceso de optimización.

Como resultado se ha seleccionado el descodificador "*Open SVC decoder",* al que se le ha aplicado la metodología de optimización del tiempo de ejecución propuesta en esta tesis. Actualmente se ha logrado el funcionamiento en tiempo real sobre el procesador TMS320DM6437 [TI06b] para secuencias de resolución CIF con seis capas que incluyen escalabilidad espacial, temporal y de calidad o SNR. Los resultados de estos trabajos se han publicado en [PSG+10] y [PJR+11].

- Aplicación de la metodología a otros sistemas que requieran un procesamiento de datos con una elevada carga computacional. Dentro del grupo de trabajo en el que se ha realizado esta tesis se están desarrollando trabajos orientados a la implementación de un terminal basado en técnicas de *Radio Software*, utilizando un procesador digital de señal. La carga computacional demandada por algunos de los módulos de esta aplicación es muy elevada, por lo que se requiere un proceso de optimización que permita su funcionamiento en tiempo real. La metodología propuesta en esta tesis se está aplicando a la realización de esta aplicación habiendo obtenido ya los primeros resultados que han sido presentados en [LJP+11].
- Aplicación de la metodología a nuevos estándares de codificación de vídeo como pueden ser la codificación multivista (*Multi View Coding* o MVC) [ITU09] estandarizada en 2009 como anexo H del estándar H.264 o la codificación de alta eficiencia (*High Efficient Video Coding* o HEVC) en proceso de estandarización en 2011[155] [ITU11].

Algunos de los trabajos relacionados con la ampliación de la metodología son los siguientes:

- Incluir otros parámetros a tener en consideración en el proceso de optimización. La metodología propuesta está centrada en la reducción del tiempo de ejecución. Sin embargo, para determinadas aplicaciones, como pueden ser los terminales móviles, existen otros parámetros que deben ser tenidos en cuenta a la hora de plantear los objetivos de la optimización como pueden ser el consumo de energía asociado a la descodificación de vídeo. Para poder incluir este parámetro dentro de la metodología es necesario caracterizar el consumo del descodificador tras aplicar cada uno de las etapas de la metodología propuesta. En este sentido ya se han publicado algunos trabajos orientados a incluir el consumo como criterio de optimización [JPL+10] [PJS+10].
- Incluir en la metodología recomendaciones para implementaciones en sistemas multi-DSP. La carga computacional que demandan los nuevos estándares de codificación, así como las resoluciones de imagen empleadas, hacen que no sea posible emplear un solo procesador para implementar un descodificador. La tendencia actual es el empleo de dispositivos como el TMS320C6472 [TI11b] que integran seis núcleos de procesamiento C64x+. Si se desean emplear estos dispositivos, es necesario ampliar la metodología de optimización para tener en consideración el particionado de los algoritmos entre los núcleos y analizar la sobrecarga que suponen las comunicaciones entre ellos.

[155] En la actualidad (Abril 2011) sólo se encuentran disponibles las implementaciones de MVC y de HEVC realizadas durante el proceso de estandarización, desarrolladas en ambos casos en C++.

7 ANEXO A. PROTOTIPOS EMPLEADOS

Para validar las metodologías de optimización propuestas en un escenario real se han empleado tres tarjetas de desarrollo basadas en diferentes DSPs. La tarjeta DESCOS ha sido diseñada y fabricada durante el desarrollo de esta tesis mientras que las otras dos (EVMDM642 y DM6437 EVM) son tarjetas de prototipado comerciales. Debido a las limitaciones de tiempo, no ha sido posible implementar todos los descodificadores en todas las tarjetas. En la Tabla 7-1 se relacionan los descodificadores que se han implementado sobre cada una de ellas.

Tarjeta	MPEG-2 MP@ML	MPEG-4 ASP	H.264 BP	H.264 MP
DESCOS	X	X	X	
EVMDM642 a 720 MHz		X	X	X
DM6437 EVM a 600 MHz			X	X

Tabla 7-1. Relación de tarjetas y descodificadores evaluados.

A continuación, en los siguientes subapartados, se resumen las características más relevantes de cada una de las tarjetas.

7.1 Tarjeta DESCOS

La tarjeta DESCOS está basada en el procesador TMS320DM642 y ha sido diseñada y fabricada durante el desarrollo de la tesis[156]. La tarjeta constituye un sistema de desarrollo de bajo coste sobre el cual se ha prototipado un *Set-Top Box IP* (STB-IP).

[156] Se ha contado con la financiación de la empresa SIDSA, interesada en los resultados de esta investigación.

El diagrama de bloques de la tarjeta se muestra en la Figura 7-1. La tarjeta está constituida por un DSP TMS320DM642 al que se conectan los siguientes elementos:

- Dos memorias conectadas a través del interfaz EMIF (*External Memory InterFace*). La primera de ellas es una SDRAM de 16 MB mientras que la segunda es una FLASH de 1 MB.
- Un controlador Ethernet conectado a través del periférico EMAC (*Ethernet Medium Access Controller*) por el que se reciben los paquetes IP que contienen las tramas elementales de audio y vídeo.
- Un vídeo *encoder*[157] que genera la señal analógica para el monitor de televisión en formato ITU-R BT.601 a partir de los datos que recibe de un puerto de vídeo (*VideoPort*) del DSP. La salida analógica se genera tanto en vídeo compuesto (*Color, Video, Blank and Sync* o CVBS) como en formato S-video[158] (Y/C). La configuración de este *encoder* de vídeo se realiza empleando el bus I2C.
- Un Convertidor Digital Analógico (DAC) que genera la salida de audio a partir de los datos que proporciona el DSP a través del McASP (*Multichannel Asynchronous Serial Port*). Este DAC también se configura a través del bus I2C del DSP.
- Un receptor de infrarrojos conectado a un pin de entrada/salida de propósito general para que el usuario pueda interactuar con el STB-IP a través de un mando a distancia.

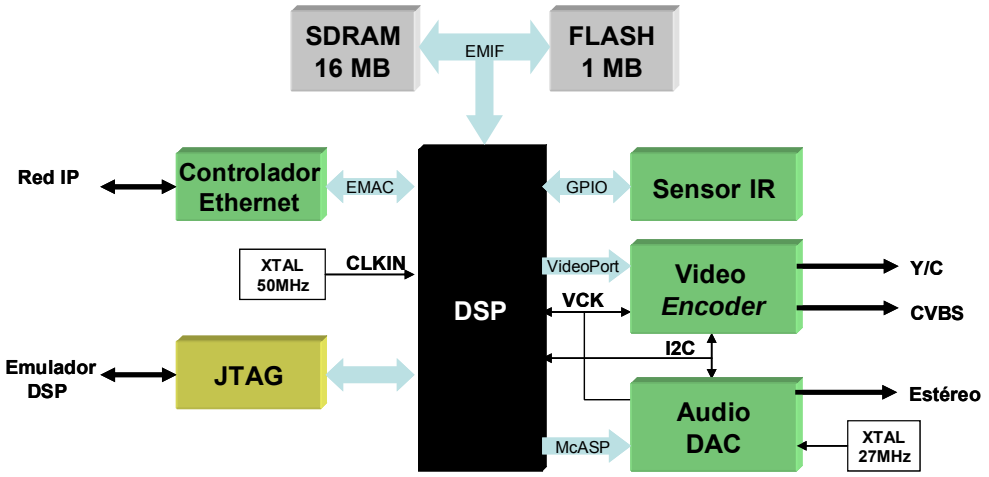


Figura 7-1 Diagrama de bloques de la tarjeta DESCOS.

La tarjeta posee dos osciladores, uno para el DSP de 50 MHz que es multiplicado internamente para obtener la frecuencia de funcionamiento de 600 MHz y otro de 27 MHz para el DAC de audio que permite a éste generar el reloj que emplea el *encoder* de vídeo. La frecuencia de este segundo reloj es modificable por parte del

[157] A lo largo del anexo se ha empleado el término inglés *encoder* en lugar de codificador para diferenciar el dispositivo que se encarga de generar la señal analógica de televisión a partir de la imagen digital del codificador de vídeo que realiza la compresión de una secuencia conforme un determinado estándar de codificación.

[158] *Separate Video* (S-video) es un formato de vídeo analógico en el que la información se transporta en dos señales separadas: luminancia y crominancia.

VCO (*Voltage Controlled Oscillator*) que posee el DSP para ajustar la frecuencia de presentación del transmisor y del receptor. Finalmente se ha incluido la circuitería necesaria para poder realizar la depuración del *software* que ejecuta el DSP mediante un interfaz JTAG (*Joint Test Action Group*). En la Figura 7-2 se muestra una fotografía de la tarjeta.

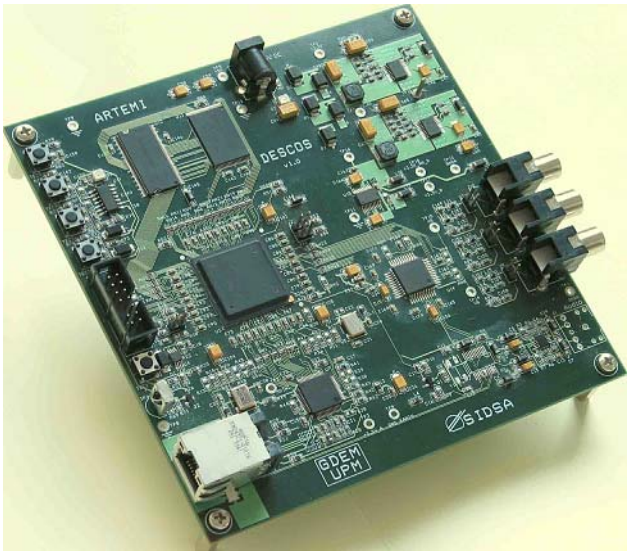


Figura 7-2 Tarjeta DESCOS.

7.2 Tarjeta EVMDM642

La tarjeta EVMDM642 [EVM642] es una tarjeta fabricada por Spectrum Digital que integra un DSP TMS320DM642[159] funcionando a 720 MHz y cuyo diagrama de bloques se presenta en la Figura 7-3[160]. Los elementos que se han empleado para los trabajos realizados en esta tesis son los siguientes:

- 32 MB de memoria DDR2 y 16 MB de memoria FLASH conectadas a través de un interfaz EMIF de 32 bits la primera y de 8 bits la segunda.
- Un *encoder* de vídeo [SAA] que facilita la generación de las señales digitales en formato RGB para ser presentada en un monitor o la señal analógica en los formatos de vídeo compuesto (CVBS) y S-video (C/Y) para un televisor. Este dispositivo recibe la señal digital conforme a la norma ITU-R BT.601 por parte del puerto de vídeo del DSP (*VideoPort* 2) a través de una FPGA[161]. La configuración de este *encoder* se realiza a través del bus I2C.
- Un Convertidor Digital Analógico [AIC23] que genera la salida de audio a partir de los datos que proporciona el DSP a través del McASP0. Este convertidor se configura también mediante el bus I2C del DSP.

[159] El procesador que emplea esta tarjeta es una versión idéntica a la empleada por la tarjeta DESCOS pero capaz de funcionar a mayor frecuencia.

[160] Al tratarse de una tarjeta de prototipado de propósito general incluye multitud de interfaces aunque no todos ellos se han empleado en el desarrollo de esta tesis.

[161] Dicha FPGA permite realizar un procesado de la señal de vídeo para, por ejemplo, incluir información para el usuario en pantalla (*On Screen Display*, OSD). Para el desarrollo del STB-IP, la FPGA facilita directamente al *encoder* los datos que recibe del puerto de vídeo del DSP.

- Un controlador Ethernet conectado al DSP a través del periférico EMAC por el que se reciben los paquetes Ethernet que contienen la trama de transporte MPEG-2.

Además de los interfaces que se han empleado para el desarrollo de la tesis la tarjeta de prototipado incluye dos vídeo *encoders* para poder digitalizar dos canales de entrada de vídeo, una serie de LEDs de propósito general, dos puertos serie e interfaces PCI (*Peripheral Component Interconnect*) y S/PDIF.

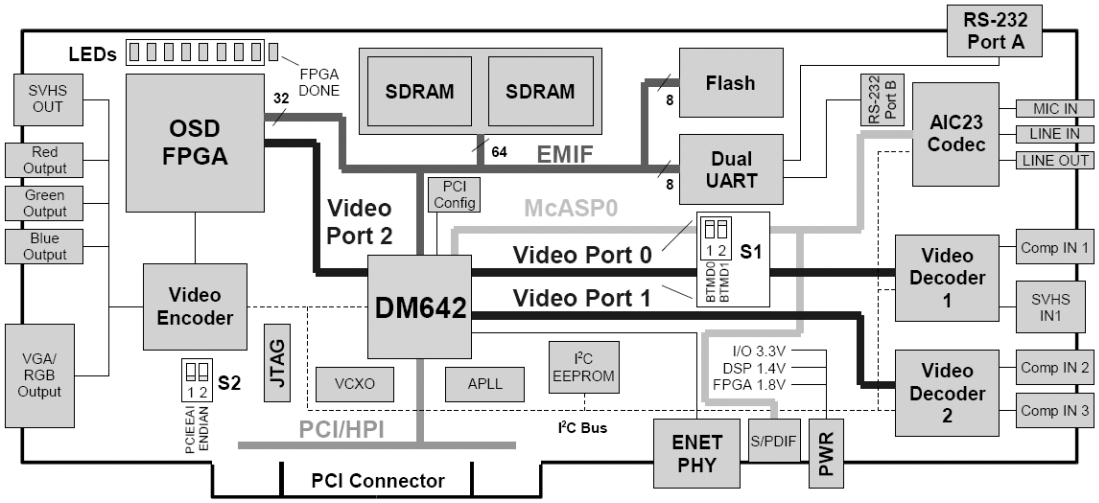


Figura 7-3 Diagrama de bloques de la tarjeta de prototipos EVMDM642.

En la Figura 7-4 puede verse una fotografía de la tarjeta.

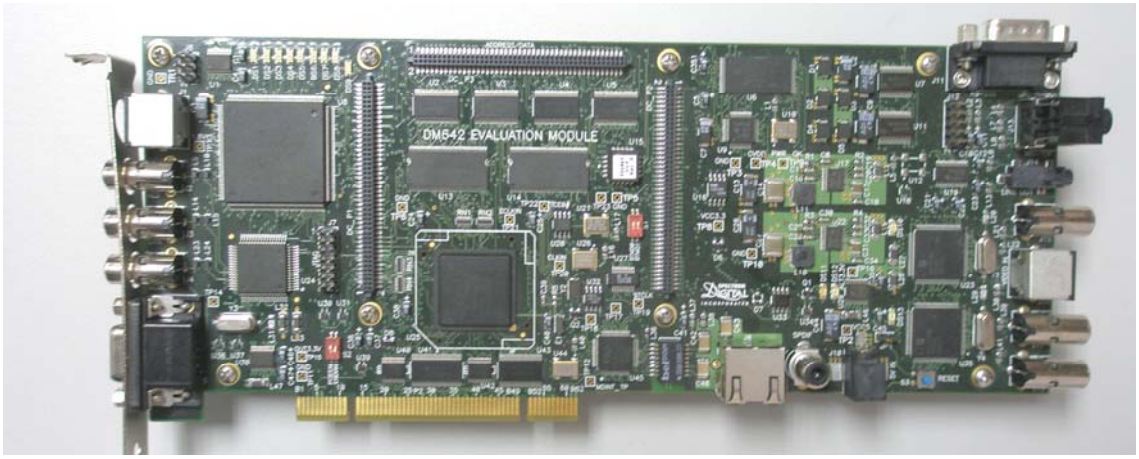


Figura 7-4 Tarjeta EVMDM642.

7.3 Tarjeta DM6437 EVM

La tarjeta DM6437 EVM (*EValuation Module*) [EVM6437] es una tarjeta de desarrollo, fabricada y comercializada por Spectrum Digital, que permite desarrollar y evaluar aplicaciones para el DSP TMS320DM6437 funcionando a 594 MHz. El diagrama de bloques de la tarjeta se presenta en la Figura 7-5. En dicho diagrama se aprecian los componentes que se encuentran conectados al DSP y las interfaces disponibles. Los más relevantes, desde el punto de vista del desarrollo que se ha realizado en esta tesis, son los siguientes:

- Una memoria DDR2 de 128 MB conectada a través de una interfaz específica de 32 bits, una memoria FLASH no volátil de 16 MB, una memoria FLASH NAND de 64 MB y una memoria SRAM de 2 MB, conectadas todas ellas al procesador mediante la interfaz EMIF.
- 4 convertidores digital-analógico que generan la señal analógica de salida para el televisor a partir de la señal digital generada por el *encoder* interno de vídeo del DSP.
- Un *codec* de audio estéreo [AIC33] conectado al DSP a través del McBSP (*Multichannel Buffered Serial Port*) y configurable a través del bus I2C que incluye el DSP.
- Una interfaz Ethernet 10/100 Mbps conectada al puerto EMAC del procesador que permite recibir los datos que se obtienen a través de la red Ethernet.
- 4 LEDs y 4 microinterruptores de propósito general. La interfaz con estos dispositivos se realiza a través del bus I2C.

Además de los periféricos e interfaces utilizados en el desarrollo de la tesis, la tarjeta de prototipado posee los siguientes elementos:

- Un TVP5146M2 [TVP51], que es un *encoder* de vídeo que recibe la entrada de vídeo analógica (soporta vídeo compuesto y S-vídeo) y genera la información de la imagen muestreada para el procesador.
- Una interfaz S/PDIF analógica y otra óptica.
- Una interfaz VLYNQ[162].
- Interfaces de entrada/salida RS-232 y CAN (*Controller Area Network*).
- Conector PCI que permite emplear la tarjeta como un periférico de este bus.
- Conectores de expansión para colocar una tarjeta auxiliar (*daughter card*).

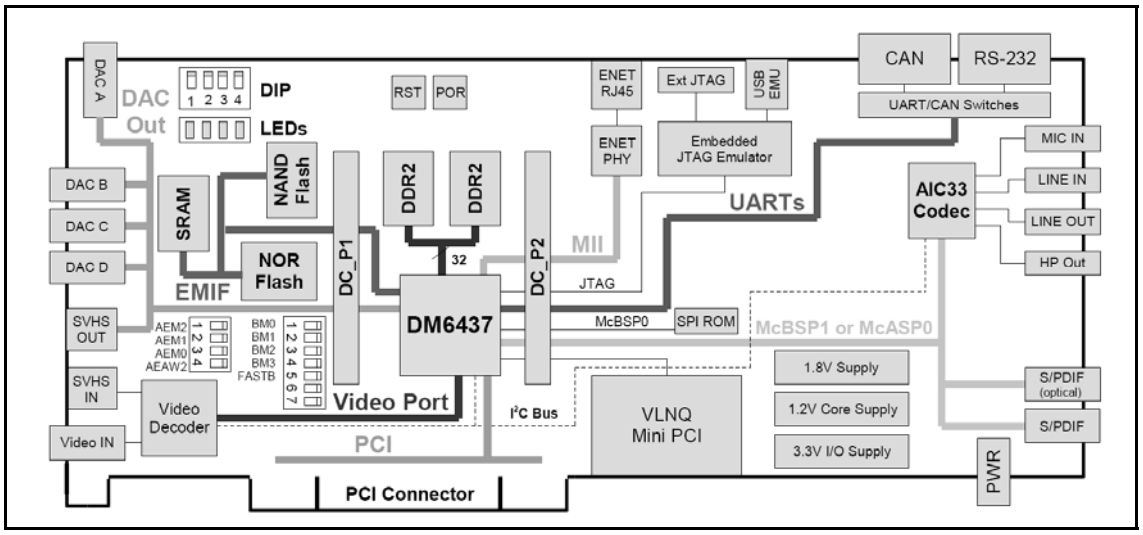


Figura 7-5. Diagrama de bloques de la tarjeta DM6437 EVM.

[162] VLYNQ es una interface propietaria de comunicaciones desarrollada por Texas Instruments para dispositivos de banda ancha. Permite que el procesador que lleva incorporada esta interface pueda conectarse con otros procesadores u otros dispositivos compatibles con este estándar.

En la Figura 7-6 se puede observar una fotografía de la tarjeta.

Figura 7-6. Tarjeta DM6437 EVM.

8 ANEXO B. ARQUITECTURA *SOFTWARE* DEL STB-IP

Sobre las tarjetas de prototipado utilizadas en esta tesis (ver Anexo A) se ha desarrollado el *software* necesario para implementar un STB-IP[163]. De esta forma, se han podido realizar pruebas con los diferentes descodificadores de vídeo desarrollados en la tesis en un entorno real. Estas pruebas han permitido que la metodología de diseño sintetizada tenga en cuenta el impacto que tiene el sistema completo en el rendimiento del descodificador de vídeo.

La Figura 8-1 muestra el diagrama de bloques de uno de los bancos de pruebas utilizado en la tesis para las medidas realizadas en un entorno real. En ella se aprecia que los datos provenientes de alguno de los distribuidores de televisión (terrenal, satélite o cable) son recibidos por un *gateway*[164] que encapsula los diversos programas disponibles en la trama de transporte recibida en paquetes UDP/IP y los distribuye a través de la red local a la que se encuentra conectado. Para esta distribución emplea direcciones IP *multicast* que facilitan que todos los usuarios que estén conectados a dicha red puedan visualizar simultáneamente cualquiera de los programas distribuidos. Junto con los datos asociados al audio y el vídeo de cada programa, el *gateway* genera información de servicio para que los usuarios sepan cuales son los programas disponibles en cada momento. Esta información suele ir integrada en paquetes SAP (*Session Announcement Protocol* [SAP]) y SDP (*Session Description Protocol* [SDP]) encapsulados en UDP/IP.

[163] Un STB-IP es un dispositivo que recibe a través una interface Ethernet, bien una trama de transporte MPEG-2, bien paquetes RTP (*Real Time Protocol*), extrae y descodifica las tramas elementales de audio y vídeo y genera las salidas analógicas que son presentadas en un televisor.

[164] Como *gateway* se ha empleado el dispositivo EtherTV fabricado por SIDSA [ETHTV]. Dicho *gateway* recibe la información de un transpondedor de televisión digital por satélite y encapsula los programas que se encuentran integrados en él en paquetes IP que son transmitidos a través de su interface de red empleando direcciones IP *Multicast*.

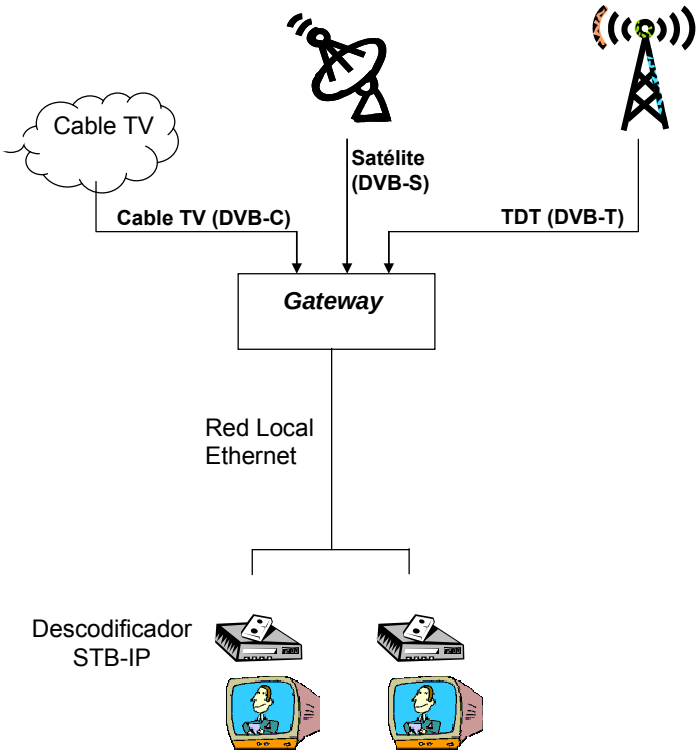


Figura 8-1 Diagrama de bloques de uno de los bancos de pruebas utilizado en la tesis.

8.1 Introducción

Para implementar un sistema completo (STB-IP) es necesario que el DSP ejecute diversas tareas (lectura de datos de la red Ethernet, análisis de la trama de transporte, descodificación de las tramas elementales, etc.) y acceda a varios periféricos, tanto internos como externos al DSP. Para facilitar esta tarea el fabricante del DSP proporciona un sistema operativo en tiempo real (*Real Time Operating System* o RTOS) denominado DSP/BIOS que permite, entre otras funcionalidades, la definición de tareas, la comunicación entre ellas y la inclusión de *drivers* para gestionar los periféricos.

En los siguientes subapartados se describen la configuración del sistema operativo para poder materializar el STB-IP. En primer lugar se realiza un resumen general del sistema indicando las tareas que se han definido, la comunicación entre ellas y los *drivers* empleados para, seguidamente, entrar en los detalles de la implementación haciendo hincapié en las optimizaciones del tiempo de ejecución que se han realizado en algunas de ellas.

8.1.1 Descripción de las tareas definidas

Para facilitar el diseño de aplicaciones complejas, el RTOS incluye un marco de referencia denominado *Reference Framework* (RF5) [TI03] compuesto por una serie de librerías que simplifican la inclusión de módulos *software* (como por ejemplo descodificadores) en el sistema, así como la comunicación entre las tareas. La arquitectura *software* que se ha realizado se presenta en la Figura 8-2 y está formada por 6 tareas, 5 colas de comunicación entre tareas y 3 controladores de periféricos. La funcionalidad de cada tarea se puede resumir del siguiente modo:

1. Tarea de transporte. Esta tarea es la encargada de recibir los datos a través de la pila TCP/IP que posee el RTOS. Estos datos pueden encontrarse

encapsulados en una trama de transporte MPEG-2 o en paquetes RTP[165]. En el primero de los casos, la tarea analiza la información específica de programa (*Program Specific Information* o PSI) para localizar los programas disponibles. Una vez que el usuario selecciona el programa que desea visualizar, se extraen las correspondientes tramas elementales (*Elementary Streams* o ES) de audio y vídeo con sus correspondientes PTSs (*Presentation Time Stamp*) y DTSs (*Decoding Time Stamp*). En el segundo caso, la tarea extrae directamente los ES de audio y vídeo de los paquetes RTP del programa que se desea visualizar[166].

Finalmente, y para ambos casos, se realiza un análisis de los ES para identificar el comienzo de cada una de las unidades de acceso (*Access Unit*[167] o AU) y facilitar la información a las tareas de descodificación. Dentro de esta tarea se definen dos *buffers* compartidos en los que se almacenarán de forma separada los ES de audio y vídeo.

2. Tarea de descodificación de vídeo. Esta tarea lee las imágenes codificadas que le facilita la tarea de transporte y procede a descodificarlas, almacenando el resultado en alguno de los tres *buffers* de salida[168] que comparte con la tarea de presentación de vídeo.

3. Tarea de descodificación de audio. Esta tarea realiza el mismo proceso que su homóloga de vídeo pero en este caso con las tramas de audio. A diferencia del caso anterior, en esta tarea sólo se define un *buffer* compartido con la tarea de presentación de audio.

4. Tarea de presentación de vídeo. Esta tarea transfiere las imágenes descodificadas al puerto de vídeo (*VideoPort*) del DSP para que éste las envíe de forma automática al *encoder* de vídeo. Esta tarea también recibe información de temporización por parte de las tareas de descodificación de audio y vídeo, necesaria para la presentación sincronizada de ambas tramas (ver Anexo B apartado 8.2.3.2). Finalmente, se ha incluido un sencillo módulo que permite presentar por pantalla al usuario los programas disponibles para que seleccione uno de ellos.

5. Tarea de presentación de audio. Esta tarea envía las tramas de audio descomprimidas al puerto serie McASP del DSP para que éstas sean transferidas al DAC que genera las salidas analógicas de audio. Al igual que ocurre con la tarea de presentación de vídeo, ésta recibe información de temporización por parte de la descodificación de audio para realizar tareas de sincronización.

---

[165] Tanto si los datos se encuentran encapsulados en una trama de transporte MPEG-2, como si lo están en paquetes RTP, antes de proceder a desencapsular los ES, es necesario realizar un análisis de los paquetes SAP que permite identificar las direcciones IP a través de las que se reciben los programas que se están transmitiendo.

[166] El análisis de la trama que hay que realizar cuando se emplea el protocolo RTP es más sencillo que el que se requiere para la trama MPEG-2. Por este motivo, a partir de este instante, las referencias que se realicen al algoritmo que implementa esta tarea estarán referidas al análisis de una trama de transporte MPEG-2.

[167] En el estándar se define una unidad de acceso como la mínima información de audio o vídeo que puede ser presentada. En el caso del vídeo se trata de un cuadro o campo y en el caso del audio se refiere a una trama de sonido.

[168] Se trata de tres *buffers* puesto que todos los estándares de codificación de vídeo que se han empleado (MPEG-2, MPEG-4 y H.264) definen tres tipos de imágenes por lo que es necesario disponer de un *buffer* para almacenar cada una de ellas.

6. Interfaz de usuario. Esta tarea permite al usuario controlar el funcionamiento del sistema. La única funcionalidad que soporta actualmente es la selección del canal que se desea visualizar de entre los disponibles.

El RTOS ejecuta las tareas de forma priorizada empleando un algoritmo de *Round Robin*. La tarea más prioritaria es la tarea de transporte para evitar que se pierdan datos recibidos por la red, en un nivel inferior se encuentran la tarea de descodificación de audio y las tareas de presentación de audio y vídeo; en el siguiente nivel de prioridad se ubica la descodificación de vídeo, ya que debido a su elevada carga computacional es necesario que las otras tareas puedan interrumpirla y, finalmente, la tarea menos prioritaria es la de aplicación puesto que el tiempo que tarda el sistema en atender las peticiones de usuario no es relevante en comparación con el del resto de las tareas.

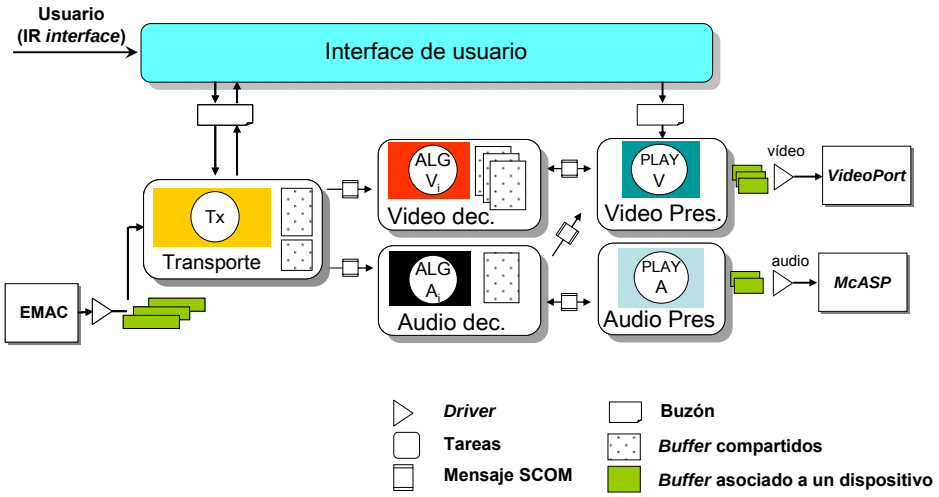


Figura 8-2 Arquitectura *software* del STB-IP.

8.1.2 Comunicación entre tareas

La comunicación y sincronización entre las tareas ha sido realizada mediante 5 colas de mensajes SCOM (*Synchronous COMmunications*). La utilidad de cada una de las colas y la información que intercambian entre las tareas que comunican se relacionan a continuación:

1. Transporte – Descodificación de vídeo. Se trata de una cola de mensajes unidireccional en la que la tarea de transporte deposita un mensaje cada vez que ha transferido una imagen codificada al *buffer* intermedio que comparten ambas tareas. El mensaje que intercambian incluye la siguiente información: dirección de memoria donde comienza la imagen y tamaño de la misma, estándar de vídeo con el que se encuentra codificada, el PTS y el DTS[169]. Esta cola está dimensionada para poder almacenar hasta un máximo de 100 mensajes lo que implica que en el *buffer* que comparten ambas tareas es posible almacenar 100 imágenes[170] (4 segundos de vídeo) como máximo.

[169] Hay que indicar que todas las imágenes y/o tramas de audio no tienen porqué incluir la información de PTS y/o DTS. Por tanto, las tareas de descodificación interpolan estos valores cuando no están disponibles para facilitárselos a las tareas de presentación.

[170] Esta afirmación es cierta siempre que el tamaño de las 100 imágenes no supere el del *buffer* en cuyo caso, se produce un desbordamiento que provoca la pérdida de imágenes con el correspondiente mal funcionamiento del sistema.

2. Transporte – Descodificación de audio. Se trata también de una cola de mensajes unidireccional en la que la tarea de transporte deposita un mensaje cada vez que ha transferido una trama codificada de audio al *buffer* intermedio que comparten ambas tareas. El mensaje que intercambian incluye la misma información que el mensaje indicado anteriormente para el vídeo con la excepción de que en este caso se indica el tipo de audio. Al igual que en el caso anterior, esta cola está dimensionada para almacenar hasta 100 mensajes con las mismas limitaciones que las mostradas anteriormente.

3. Descodificación de vídeo – Presentación de vídeo. Se trata de una cola de mensajes bidireccional[171] que comparten las tareas de descodificación y presentación de vídeo. La tarea de descodificación deposita un mensaje en esta cola cada vez que ha descodificado una imagen indicando la dirección de memoria en la que se encuentra, el tamaño de la misma, el PTS y el DTS. A continuación la tarea de descodificación queda bloqueada a la espera de una contestación por parte de la tarea de presentación. La tarea de presentación recoge el mensaje de la cola, envía la imagen al puerto de vídeo, avisa a la tarea de descodificación para que ésta pueda comenzar a descodificar la siguiente imagen[172] y queda bloqueada a la espera de recibir una nueva imagen.

4. Descodificación de audio – Presentación de audio. Su funcionamiento es completamente análogo al descrito para el vídeo con la diferencia que el mensaje que envía la descodificación a la presentación incluye la frecuencia de muestreo y el número de bits por muestra en lugar del tamaño de la imagen.

5. Descodificación de audio – Presentación de vídeo. Se trata de una cola unidireccional en la que el descodificador de audio deposita mensajes que incluyen la información del PTS y DTS de la trama de audio descodificada para que la tarea de presentación de vídeo tenga información de la temporización del audio y pueda realizar la sincronización entre ambas (para más información sobre el proceso de sincronización puede consultarse el apartado 8.2.3.2 de este Anexo).

La comunicación entre la tarea de aplicación y el resto de tareas se ha realizado empleando buzones (*mailboxes*). Se trata de un recurso que facilita el RF5 para la comunicación entre tareas cuando el tiempo de respuesta no es un parámetro relevante. La tarea que desea informar de algún evento a otra deposita un mensaje en estos buzones que será leído por el destinatario cuando no se esté realizando otras funciones. Para desarrollar esta aplicación se han definido dos buzones:

1. Tarea de transporte – Aplicación. A través de este buzón la tarea de transporte indica los programas disponibles para que sean presentados al usuario. Una vez que éste selecciona uno de ellos la tarea de aplicación informa a la tarea de transporte para que ésta filtre el programa adecuado.

2. Aplicación – Tarea de presentación de vídeo. Empleando este buzón la tarea de aplicación informa a la presentación de vídeo de los programas disponibles en cada momento para presentárselos al usuario y que éste seleccione el que desea visualizar.

---

[171] En realidad se trata de dos colas SCOM diferentes que emplean ambas tareas. Sin embargo, por simplicidad en la explicación, se considera como una única cola bidireccional.

[172] Para poder realizar la sincronización del audio y el vídeo de un programa (para más información consultar el apartado 8.2.3.2), la tarea de presentación informa a la de descodificación si debe descartar alguna imagen para recuperar la sincronización entre ambos ES en el caso de que el vídeo vaya retrasado respecto al audio.

8.1.3 Controladores de periféricos

Dentro de la arquitectura *software* se han integrado tres controladores de periféricos que están disponibles en el RTOS. Resumidamente la funcionalidad de cada uno de ellos es la siguiente:

1. *Ethernet Media Access Controller*. Se trata del controlador que permite el acceso al puerto de comunicaciones Ethernet que posee el DSP. Este controlador es capaz de almacenar las tramas recibidas y facilitárselas a las aplicaciones de alto nivel. Asociada a este controlador se encuentra una librería que materializa la pila TCP/IP basada en *sockets*.

2. *Videoport Display*. Se trata del controlador que transfiere las imágenes descodificadas que son facilitadas por la tarea de presentación de vídeo al periférico integrado en el DSP que se encarga de generar la trama de salida que es enviada al *encoder* de vídeo[173] en formato 4:2:2 progresivo

3. *dioCodec* y *devCodec*. Este controlador está compuesto por dos módulos que se encargan de recibir las tramas de audio descodificadas en formato estéreo PCM con 8 bits por muestra y enviarlas al convertidor digital-analógico a través del puerto McASP (TMS320DM642) o McBSP (TMS320DM6437).

8.2 Descripción de las tareas incluidas en el STB-IP

Cada una de las tareas que se han enumerado en el apartado anterior ha sido implementada teniendo en cuenta tanto su funcionalidad, como el impacto que la misma tiene en el rendimiento global del sistema. La metodología empleada en el diseño de cada tarea ha sido la siguiente:

- se realiza una primera versión del código y se comprueba su funcionalidad mediante simulación.
- se realiza un análisis de rendimiento.
- se realiza una segunda versión optimizada de aquellas tareas con un mayor coste computacional.

Si se excluye la tarea de descodificación de vídeo, la tarea de transporte es la que tiene un coste computacional más elevado, sobre todo con regímenes binarios de entrada elevados. Por ello sobre este código se ha realizado el mayor esfuerzo de optimización.

La tarea que se encarga de la presentación del vídeo es sencilla pero en su desarrollo ha sido necesario tener en cuenta dos cuestiones. Por una parte esta tarea incluye el algoritmo que se ha implementado para sincronizar las tramas de audio y vídeo ya que estas pueden llegar desincronizadas dentro de la trama de transporte. Por otro lado, esta tarea realiza un movimiento de datos elevado puesto que es la encargada de transferir las imágenes descodificadas al puerto de vídeo. A pesar de realizar este movimiento de datos empleando el controlador de DMA, para imágenes grandes[174], se requiere gran cantidad de ciclos de reloj por parte de la CPU debido al

[173] En el caso del sistema basado en el procesador TMS320DM6437 (consultar Anexo 7.3) el *encoder* de vídeo se encuentra integrado dentro del propio DSP. A pesar de ello, la tarea de presentación de vídeo envía a través del *driver* correspondiente los datos en formato 4:2:2 al *encoder* del mismo modo que se realiza para el TMS320DM642.

[174] Para el caso de una imagen con tamaño PAL en formato 4:2:2 es necesario transferir 720×576×2 = 810 kB por imagen.

elevado número de transferencias de DMA que es necesario configurar. Por este motivo se ha realizado un proceso de optimización para mejorar su rendimiento.

Finalmente, las tareas de descodificación de audio y presentación de audio tienen una complejidad relativamente baja y manejan un volumen de datos reducido en comparación con las demás. Por estas razones, no se ha considerado necesario realizar un esfuerzo de optimización en ellas.

En los siguientes apartados se realiza una descripción funcional de las tareas desarrolladas así como el análisis de rendimiento que se ha realizado de cada una de ellas. En aquellos casos en los que ha sido necesario realizar un proceso de optimización, éste se explica de forma resumida.

La optimización se ha realizado aplicando algunas de las metodologías presentadas en el capítulo 4 de esta memoria lo que ha permitido demostrar su utilidad como técnicas generales de optimización para aplicaciones desarrolladas sobre DSPs.

8.2.1 Tarea de transporte

8.2.1.1 Introducción a la trama de transporte MPEG-2

La norma ISO/IEC 13818 en su parte 1 (MPEG-2 Sistema) [ISO07] describe como se deben insertar tramas de audio y vídeo elementales en una única trama de salida en la que los datos se encuentren multiplexados. En este apartado se introducen los aspectos más destacados de la norma [175].

La Figura 8-3 muestra la estructura de un sistema multiplexor de trama de transporte (*Transport Stream* o TS). En el sistema se aprecian los codificadores de audio y vídeo que generan las tramas elementales que son posteriormente fragmentadas en paquetes PES (*Packetized Elementary Streams*) de longitud variable. El multiplexor recibe los paquetes PES provenientes de varios programas, los fragmenta en paquetes de transporte TSPs (*Transport Stream Packets*) de tamaño fijo e igual a 188[176] *bytes* y los mezcla para obtener una única trama de salida. Cada uno de estos TSPs posee una cabecera de tamaño fijo que incluye, entre otros datos, un identificador (*Program IDentifier* o PID) que distingue el contenido de los datos que transporta.

Habitualmente varios programas suelen estar incluidos en una trama de transporte de modo que cada uno de ellos se ha generado con una base de tiempos diferente. La base de tiempos se obtiene a partir de un contador de 48 bits funcionando con un reloj de 27 MHz. El valor de este contador, denominado PCR (*Program Clock Reference*), se incluye periódicamente en algunos TSPs para que el receptor pueda recuperar su valor y ajustar el reloj local de 27 MHz. Además, el generador de PES incluye en algunas cabeceras de PES información del PCR[177] en el instante en el que se codifica la imagen o la trama de audio. Esta información es

[175] La complejidad de esta norma impide que en este documento se realice un estudio exhaustivo, sin embargo, si se ha considerado interesante hacer una introducción a la misma que justifique el proceso de optimización que se ha realizado, así como el impacto que esta tarea tiene sobre el resto de elementos del sistema.

[176] El estándar permite que los paquetes TSPs tengan una longitud de 208 *bytes* en el caso de que incluyan un código *Reed-Solomon* para la corrección de posibles errores de transmisión.

[177] En realidad no se incluyen los 48 *bits* que posee el PCR si no que sólo se emplean los 33 *bits* más significativos puesto que con ellos se obtiene una resolución temporal de 3.14 ms que es suficiente para la sincronización de las tramas de audio y vídeo.

denominada PTS y sirven para que el receptor sepa en qué instante presentar sincronizadamente cada AU de audio o vídeo.

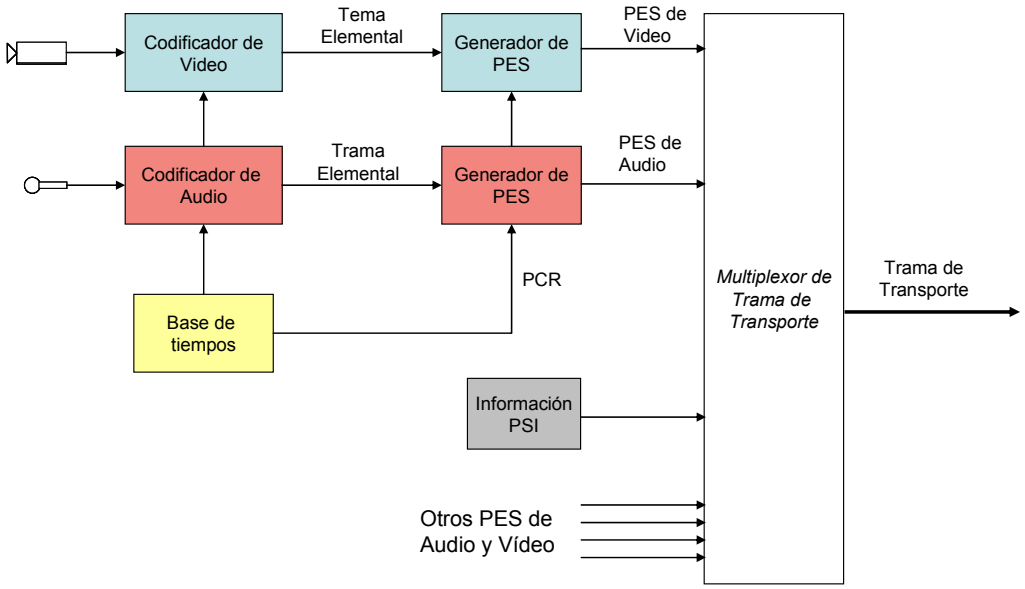


Figura 8-3 Estructura de un sistema generador de una trama de transporte MPEG-2.

Junto a la información del audio y el vídeo que compone la trama de transporte es necesario incluir información adicional (*Program Specific Information* o PSI) que sirve para que el receptor sepa qué programas están incluidos en la trama recibida y cómo debe extraerlos. Esta información está compuesta básicamente por una serie de tablas que son encapsuladas en secciones y éstas a su vez en TSPs. De todas las tablas que se definen en el estándar tienen una especial relevancia la PAT (*Program Allocation Table*) y la PMT (*Program Map Table*). La primera de ellas está formada por la lista de los PIDs de los TSPs que contienen las PMTs que están asociadas a cada uno de los programas incluidos en la trama. Por otro lado, cada PMT incluye la lista de identificadores de los TSPs que transportan los ES de audio y vídeo asociados a un programa. Por tanto habrá tantas PMTs como programas integre la trama de transporte.

El proceso simplificado para recuperar las tramas elementales de audio y vídeo multiplexadas en una trama de transporte es el siguiente. En primer lugar se van procesando los TSPs a la espera de encontrar uno que posea un PID igual a 0[178]. Cuando se detecta uno de estos TSPs es porque se trata de un paquete que contiene una PAT[179]; tras analizar el contenido de esta tabla es posible presentar al usuario del sistema los programas disponibles y pedirle que seleccione el que desea visualizar. El PID del programa seleccionado permite a su vez filtrar la PMT asociada a este programa. Tras ser extraída y analizada es posible conocer los PIDs asociados a los diferentes ES que componen el programa seleccionado. En el caso de disponer de

[178] Este es un PID reservado para la PAT de modo que sólo este tipo de tablas puede emplearlo.

[179] Un único TSP no suele ser suficiente para almacenar el contenido de una tabla PSI. En este caso la tabla se divide en varios trozos empleando un mecanismo de secciones similar al empleado para el encapsulado de los ES en los paquetes PES. Por tanto, para recuperar la tabla PSI es necesario obtener todas las secciones en las que se ha dividido antes de poder analizarla.

más de un PID para el audio o para el vídeo[180] el usuario debe seleccionar el audio y/o vídeo que desea escuchar y/o visualizar para, finalmente, proceder a filtrar todos los TSPs que poseen los PIDs de audio y vídeo seleccionados. Empleando estos TSPs el receptor extrae los paquetes PES asociados y a partir de estos los ES que debe descodificar.

Dado que los TSPs deben estar alineados al comienzo de las secciones de los PSIs o de los PES, y éstos no tienen porqué ser múltiplos de 184[181] *bytes,* el último paquete de cada sección de PSI o de cada PES se completa con *bytes* de relleno hasta alcanzar los 184 *bytes*. La Figura 8-4 muestra una trama en la que se aprecia el particionado que se realiza tanto de los ES como de la información de servicio.

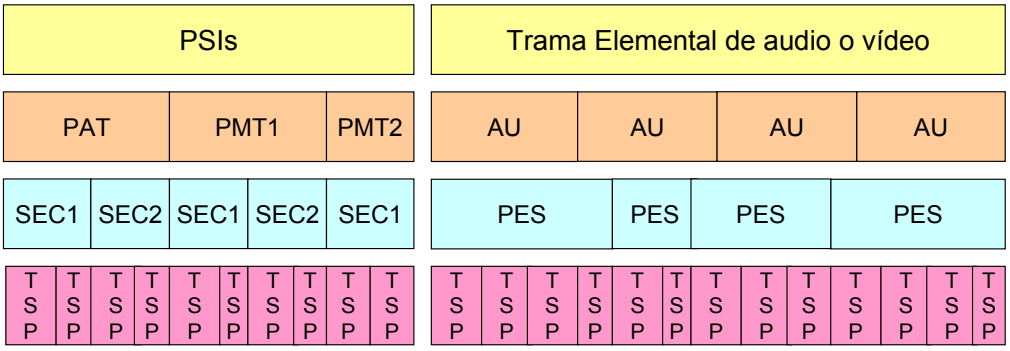


Figura 8-4 Estructura de una trama de transporte MPEG-2.

8.2.1.2 Descripción de la tarea de transporte

El módulo que realiza el análisis de la trama de transporte y extrae los PSIs y los ES de audio y vídeo se encuentra integrado en la tarea de transporte presentada en la Figura 8-2. El punto de partida para el desarrollo de este módulo fue la librería LibDVBPSI [LIBDVB] que implementa en lenguaje C el análisis de la PAT y las PMTs que se encuentran en una trama de transporte MPEG-2. Este código fue migrado al entorno del DSP y ampliado para poder extraer los ES de audio y vídeo y localizar el comienzo de cada uno de ellos. Inicialmente se desarrolló una versión íntegramente en lenguaje C para realizar pruebas funcionales que, posteriormente, fue optimizada para reducir su tiempo de ejecución.

La trama de transporte que se recibe se encuentra encapsulada en paquetes UPD/IP[182] como recomienda la norma [DVB07] de modo que en cada paquete Ethernet se incluyen 7 TSPs[183]. Estos paquetes UDP/IP son procesados por la pila

[180] Por ejemplo en programas que se emitan en varios idiomas o bien, aunque no es habitual, con varias resoluciones espaciales del vídeo, cada una en una trama elemental independiente.

[181] Los TSP poseen una cabecera de 4 *bytes* por lo que la capacidad útil es de 184 *bytes.*

[182] Dependiendo del tipo de aplicación también es posible que los paquetes TSPs se transmitan empleando el protocolo TCP. Sin embargo, para las aplicaciones de difusión de datos (*broadcasting*) es más habitual emplear el protocolo UDP puesto que simplifica las comunicaciones y evita tener que disponer de un canal de retorno entre el usuario y el productor de los contenidos.

[183] El motivo por el que se suelen incluir 7 TSPs en un paquete UDP o TCP se debe a que con este número de TSPs se obtiene un paquete de 1316 *bytes* que puede ser incluido directamente dentro de la capacidad útil de un paquete Ethernet, evitando de este modo el particionado de los paquetes UDP o TCP lo que simplifica la realización del receptor. En cualquier caso, y como es posible que algunas aplicaciones incluyan un número distinto de TSPs en cada paquete IP, el *software* desarrollado está preparado para poder procesar paquetes UDP o TCP con cualquier número de TSPs.

TCP/IP que entrega a la tarea de transporte los TSPs. Estos paquetes son procesados en tres etapas tal y como se muestra en la Figura 8-5:

- En una primera etapa (Gestión de Red) los datos extraídos de la pila TCP/IP son almacenados en un *buffer* intermedio quedando disponibles para posteriores etapas.
- En la segunda etapa (Analizador de TSPs) los TSPs son analizados con el objeto de identificar en primer lugar los que pertenecen a la PAT y a la PMT del programa seleccionado por el usuario. Una vez que se ha seleccionado un programa se filtran aquellos TSPs que son de interés, se reconstruyen los paquetes PES de audio y vídeo y se extraen los ES. Durante esta fase se separa la información de audio y vídeo que se almacena en *buffers* intermedios diferentes que se encuentran localizados en memoria interna.
- En la tercera y última etapa (Analizador de Tramas Elementales) se realiza una búsqueda de las AUs en los *buffers* intermedios de audio y vídeo de modo que las unidades de acceso completas que se localizan son enviadas a unos *buffers* compartidos de mayor tamaño ubicados en memoria externa a la vez que se señaliza a la correspondiente tarea de descodificación la existencia de una nueva trama de audio o una imagen que descodificar.

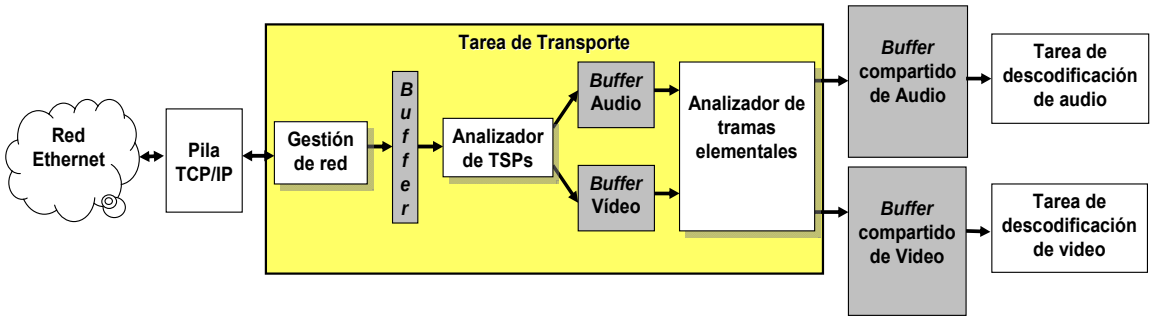


Figura 8-5 Estructura interna de la tarea de transporte.

El procesamiento que realiza esta tarea requiere un acceso intensivo a los datos recibidos a través de la red puesto que es necesario buscar continuamente cabeceras y realizar copias de datos entre diferentes *buffers*. Este tipo de procesamiento puede ser costoso en términos de tiempo de ejecución por parte de la CPU si no se tienen en cuenta algunos aspectos relacionados con la arquitectura del DSP y la jerarquía de memoria que posee.

Además, la carga computacional de la tarea se incrementa linealmente según aumenta el régimen binario de los datos de entrada. Por este motivo se ha realizado un proceso de optimización aplicando algunas de las técnicas que se han descrito en el capítulo 4.

8.2.1.3 Proceso de optimización

Una vez comprobado que el código desarrollado en lenguaje C era funcionalmente correcto, se analizó su rendimiento en términos de número de ciclos de reloj invertidos en la extracción de los ES de audio y vídeo. Como resultado de este análisis se observaron dos cuellos de botella que ralentizaban su ejecución. Por un lado, las copias de los ES a los *buffers* compartidos de audio y vídeo desde los *buffers* intermedios se realizaban mediante bucles poco eficientes que mantenían ocupada a

la CPU durante largos periodos de tiempo. Por otro lado, la búsqueda de cabeceras tanto dentro del TS, como en las AUs empleaba datos de 8 bits cuando estas cabeceras tienen habitualmente un tamaño mayor[184]; además la identificación del comienzo de cada ES se realizaba directamente en los *buffers* compartidos que, debido a su tamaño, debían alojarse en memoria externa.

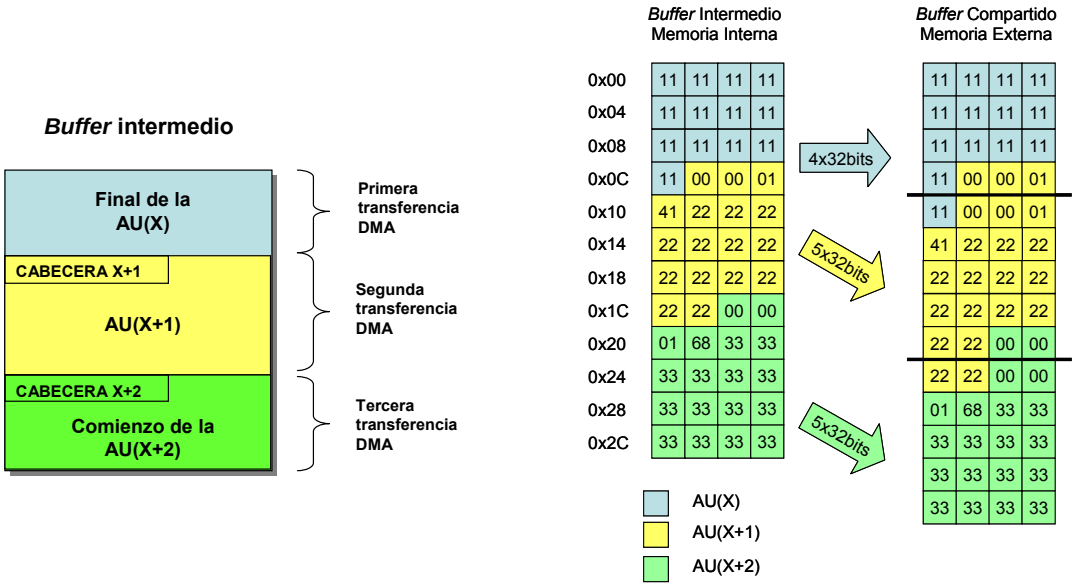
Para minimizar el efecto de estos cuellos de botella y mejorar el rendimiento de esta tarea se han utilizado técnicas similares a las que se emplearon para optimizar el tiempo de ejecución de los descodificadores de vídeo. En los siguientes apartados se resumen el proceso de optimización que se ha realizado.

8.2.1.3.1 Transferencias por DMA

El principal cuello de botella del algoritmo propuesto se encuentra en la copia de datos que se realiza entre los diferentes *buffers*. Por un lado es necesario copiar los datos extraídos de cada paquete Ethernet a un *buffer* alojado en memoria interna de tamaño igual a 1514 *bytes* para que queden disponibles para el analizador de TSPs. Seguidamente, el contenido de este *buffer* es analizado y los datos útiles de los ES de audio y vídeo copiados a dos *buffers* intermedios almacenados en memoria interna que contendrán normalmente fragmentos de una o varias AUs. Finalmente, los ES incluidos en los *buffers* intermedios se analizan para detectar las cabeceras de las AUs; de esta forma es posible discriminar y transferir cada AU al *buffer* compartido de vídeo (3 MB) o audio (128 kB) según corresponda. Debido al elevado tamaño de los *buffers* compartidos, éstos deben ser alojados en memoria externa por lo que es necesario optimizar el movimiento de datos hacia ellos haciendo uso de transferencias explícitas de DMA que permitan que la CPU continúe ejecutando código mientras se realiza el movimiento de datos.

En general, dentro de un *buffer* intermedio puede haber varios fragmentos de diferentes AUs. Así, la Figura 8-6-a presenta un ejemplo en el que el *buffer* intermedio contiene el final de una AU (X), otra AU (X+1) completa y el comienzo de una tercera AU (X+2). En este caso el módulo “analizador de tramas elementales” comienza el análisis del contenido hasta encontrar la cabecera de la AU (X+1). En ese momento se ordena una transferencia por DMA del fragmento de la AU (X); mientras se realiza la transferencia se continúa el análisis hasta localizar la cabecera de la AU (X+2) momento en el que se ordena una nueva transferencia de DMA de la AU (X+1). Cuando se alcanza el final del *buffer* intermedio se lanza una tercera transferencia de DMA correspondiente al comienzo de la AU (X+2). Finalmente, se espera a que finalice la transferencia y se vuelva a rellenar el *buffer* intermedio para comenzar de nuevo el análisis.

[184] En el caso de las cabeceras de los ES de vídeo éstas son de 32 bits mientras que las de los ES de audio son de 16 bits.



a) Ejemplo del contenido de un *buffer* intermedio que incluye información de tres AUs.

b) Transferencias de DMA que permiten que la dirección de comienzo esté alineada a 32 bits y que el tamaño de las transferencias sea múltiplo de 32 bits.

Figura 8-6 Transferencias de DMA de las AUs de audio y vídeo desde los *buffers* intermedios a los *buffers* compartidos.

La Figura 8-7 muestra la secuencia temporal de análisis y transferencia de datos para un ejemplo en el que primeramente se tiene un *buffer* intermedio que contiene una AU completa (AU (X+1)) y dos incompletas (AU (X) y AU (X+2))[185]; una vez finalizado el procesamiento de los datos del *buffer*, éste se vuelve a rellenar con datos que se corresponden con un nuevo fragmento de la AU (X+2). En el diagrama temporal se aprecia como se paraleliza el procesamiento de los datos en un *buffer* alojado en memoria interna con las transferencias a memoria externa.

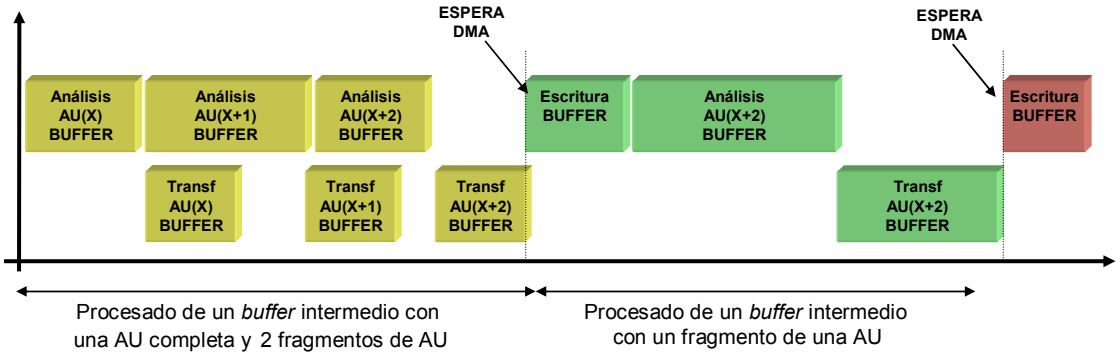


Figura 8-7 Paralelización de las transferencias de DMA con la escritura de datos.

[185] En el peor de los casos uno de estos *buffers* intermedios contendrá exclusivamente un fragmento de una AU por lo que tras finalizar la búsqueda de las cabeceras se realizará una única transferencia de todo el *buffer* empleando el controlador de DMA. Debido al tamaño de este *buffer*, el tiempo de espera para que finalice la transferencia por parte de la CPU es elevado.

Para mejorar aún más el rendimiento de este proceso es necesario que los datos a transferir estén alineados a 32 bits en memoria y tengan un tamaño múltiplo de 32 bits (ver apartado 3.2.1). Normalmente las AUs no cumplen estos requisitos por lo que se ha optado por transferir más *bytes* de los necesarios de modo que los últimos *bytes* de una AU y los de comienzo de la siguiente se transfieren en dos ocasiones tal y como muestra la Figura 8-6-b. En ella se aprecia que la AU (X+1) comienza en la dirección 0x0D y finaliza en la 0x1E. Por tanto, al realizar la transferencia del último fragmento de la AU (X), se incluye parte del comienzo de la AU (X+1); de igual modo, para la transferencia correspondiente al comienzo de la AU (X+1) se incluyen los últimos *bytes* de la AU (X). La misma circunstancia ocurre en el último fragmento de la AU (X+1) y el comienzo de la AU (X+2). De este modo para transferir el fragmento de la AU (X) que posee 13 *bytes* se realiza una transferencia de 4 palabras de 4 *bytes*, para la AU (X+1) y la AU (X+2) se transfieren 5 palabras de 4 *bytes* aunque realmente ambos fragmentos tienen un menor tamaño. La inclusión de estos datos extras en cada transferencia complica la gestión del *buffer* compartido y la señalización a las tareas de descodificación; a pesar de ello, la mejora en el rendimiento justifica su aplicación.

8.2.1.3.2 Empleo de doble *buffer o buffers* ping-pong

El uso de las transferencias explícitas presenta un problema que reduce su rendimiento puesto que, una vez que se ha finalizado el análisis del *buffer* intermedio y se ha ordenado la última transferencia de DMA, es necesario esperar a que ésta finalice para que el analizador de TSPs pueda escribir nuevos datos en los *buffers* intermedios. Si no se realizara esta espera los datos podrían ser sobrescritos si el controlador de DMA no hubiera finalizado la transferencia. Para evitar este problema el *buffer* intermedio se ha convertido en un doble *buffer* lo que permite que mientras se está realizando una transferencia por DMA empleando uno de ellos, la CPU pueda escribir nuevos datos en el otro. De este modo se paralelizan completamente las transferencias de DMA con el análisis de las cabeceras de las AUs.

La Figura 8-8 muestra la paralelización que se consigue gracias al empleo de la técnica del doble *buffer* que elimina las esperas y permite que la CPU ejecute instrucciones de forma continua.

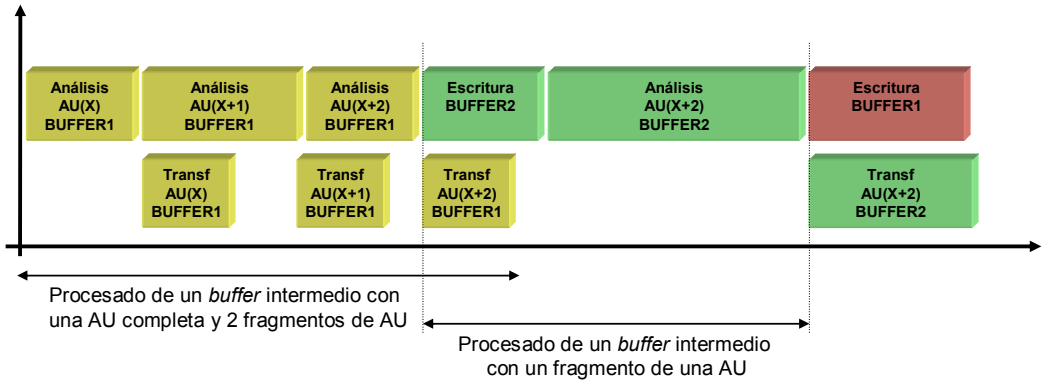


Figura 8-8 Paralelización de las transferencias de DMA con la escritura de datos empleando la técnica del doble *buffer.*

8.2.1.3.3 Procesado de las cabeceras de las AUs

Como se ha indicado anteriormente, las cabeceras de las AUs de audio y vídeo poseen más de 8 bits lo que implica varias lecturas en posiciones de memoria consecutivas si éstas se realizan a nivel de *byte*. Para el caso del vídeo, las cabeceras son de 32 bits lo que requiere la lectura de 4 *bytes*. Para mejorar la eficiencia en el

acceso a estos datos se ha implementado un algoritmo de "ventana deslizante" en el que se realizan dos lecturas de dos datos de 32 bits consecutivos y se almacenan en dos registros de la CPU. A continuación una ventana de 32 bits se va desplazando a lo largo del dato de 64 bits que se ha leído en búsqueda de una cabecera de vídeo[186]. Si la ventana se desplaza hasta consumir los 64 bits leídos sin encontrar una cabecera, se produce una nueva lectura de 32 bits y se continúa la búsqueda[187].

La Figura 8-9 muestra un ejemplo de búsqueda de las cabeceras de una NAL empleada en el estándar H.264 (ver apartado 2.2.4.2.1). En la primera fase le leen dos datos de 32 bits (Dato 1 y Dato 2) y se busca la cabecera en los 4 primeros *bytes* (Dato 1), si no se encuentra se pasa a la fase 2 en la que se reutilizan los 3 últimos *bytes* del Dato 1 y el primer *byte* del Dato 2. Este proceso se repite hasta la fase 6 en la que es necesario leer un nuevo dato que se almacena en Dato 1. Finalmente, en la fase 8, se localiza la cabecera que comienza en el séptimo *byte*.

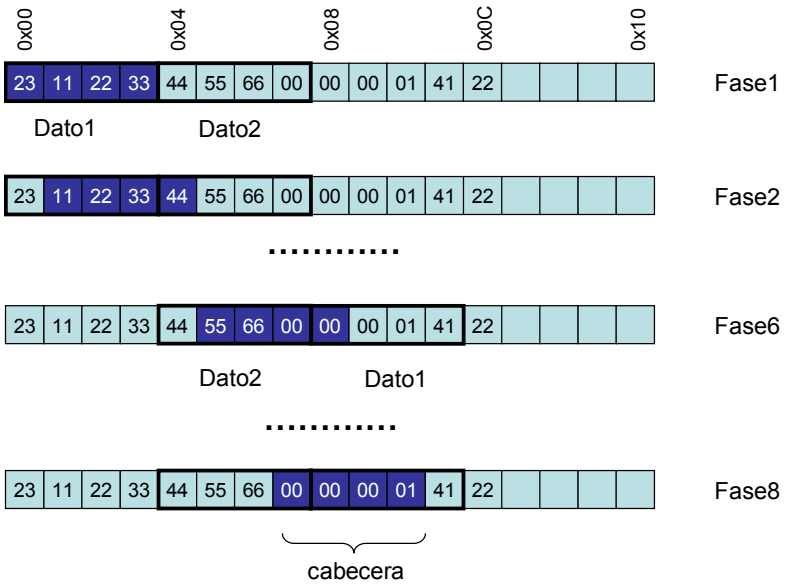


Figura 8-9 Búsqueda de cabeceras en la trama empleando el algoritmo de ventana deslizante.

8.2.1.4 Resultados

Para obtener el rendimiento del analizador de trama de transporte se han realizado dos medidas independientes. En primer lugar se ha preparado un banco de test en el que se analiza el rendimiento de esta tarea de manera independiente del resto del sistema. Empleando este banco de test y la herramienta de medida de rendimiento que facilita el fabricante del DSP [TI11], se han realizado medidas con varios ficheros que almacenan tramas de transporte con diferentes regímenes binarios.

[186] Esta búsqueda de cabeceras aprovecha algunas instrucciones específicas que posee el DSP y que facilitan este proceso.

[187] El mismo proceso que se realiza para las cabeceras de vídeo se repite para las de audio pero empleando 16 *bits*.

Los resultados de las pruebas se muestran en la Tabla 8-1[188]. La primera fila presenta el número de ciclos de reloj por segundo necesarios para extraer un programa compuesto por un ES de audio y otro de vídeo mientras que la segunda fila representa el porcentaje de CPU empleado suponiendo que se trata de un TMS320DM642 trabajando a 600 MHz[189].

Régimen binario	1 Mbps	2 Mbps	4 Mbps	8 Mbps
Ciclos de Reloj ($\times 10^6$)	7.8	13.8	24.0	48.6
% CPU a 600 MHz	1.3%	2.3%	4.0%	8.1%

Tabla 8-1. Rendimiento de la tarea de análisis de la trama de transporte de forma aislada.

El segundo banco de test mide el rendimiento de esta tarea una vez que se ha integrado en el sistema completo, para el caso en el que se recibe un TS que contiene un programa en el que tanto el audio como el vídeo se encuentran en formato MPEG-2. De este modo se puede analizar el impacto que tiene el uso del DMA que se realiza en esta tarea cuando se integra con un descodificador de vídeo que también emplea intensivamente este recurso. Para ello se han difundido a través de la red Ethernet las mismas tramas de transporte empleadas en el primer banco de test y se ha medido el rendimiento del analizador de trama en tiempo real[190]. Los resultados se muestran en la Tabla 8-2 en la que se aprecia que la carga computacional del analizador crece ligeramente respecto a los resultados presentados en la Tabla 8-1. Como ya se adelantó este hecho se debe fundamentalmente a que tanto el descodificador de vídeo como la tarea de análisis de la trama de transporte hacen un uso intensivo de los canales de DMA que posee el DSP, por lo que en algunas ocasiones las transferencias pueden ser demoradas[191].

Régimen Binario	1 Mbps	2 Mbps	4 Mbps	8 Mbps
Ciclos de Reloj ($\times 10^6$)	8.4	15.0	25.8	51.0
% CPU a 600 MHz	1.4%	2.5%	4.3%	8.5%

Tabla 8-2. Rendimiento de la tarea de análisis de la trama de transporte integrada con el resto de elementos del sistema.

[188] Para obtener estos resultados se ha empleado la misma configuración de memoria descrita en el apartado 4.4.2.1 que posee el STB-IP.

[189] Las medidas de rendimiento que se presentan en la Tabla 8-1 se han realizado también empleando la plataforma basada en TMS320DM6437 obteniendo unos resultados análogos.

[190] Para realizar medidas de rendimiento en tiempo real no es posible emplear el analizador de rendimiento que proporciona el fabricante del DSP por lo que se ha optado por emplear los temporizadores (*timers*) internos que incorpora el procesador. El método empleado para realizar la medida consiste en arrancar un temporizador al comienzo del código a medir y pararlo al finalizar. A partir de los datos leídos, y conociendo la frecuencia del reloj del contador, se obtiene el tiempo de ejecución. Para que esta medida sea fiable es necesario que el código a medir se encuentre en la tarea más prioritaria del sistema para de este modo evitar que otras tareas pasen a ejecutarse mientras se está realizando la medida.

[191] Las medidas de rendimiento de esta tarea se han repetido para tramas de transporte con ES de vídeo en formato MPEG-4 ASP y H.264 obteniendo unos resultados muy similares a los presentados en la Tabla 8-2.

8.2.2 Descodificadores de audio

Se han realizado implementaciones de los descodificadores de audio más implantados en la actualidad:

- MPEG-1 Capas II y III (*Layers* II y III). Esta familia de descodificadores de audio son los que se encuentran definidos en la parte 3 de la norma MPEG-2 [ISO98]. La capa II es la que habitualmente se emplea en las transmisiones de televisión digital tanto por satélite, como terrena o por cable. Por otro lado la capa III es lo que coloquialmente se denomina MP3 y no suele emplearse en sistemas de difusión. En ambos casos sólo se permiten dos canales de audio para generar una señal estéreo.
- AC-3 [ATSC05]. Se trata de un estándar de codificación de audio definido por el ATSC (*Advanced Television System Commitee*). Este estándar tiene una mayor capacidad de compresión que los formatos de compresión de audio definidos en MPEG-2 y soporta la inclusión de varios canales de audio para obtener sistemas de mayor calidad. Este estándar es el que suele emplearse para codificar los ES de audio en los DVDs.
- AAC (*Advanced Audio Coding*). Este algoritmo de codificación de audio se definió en la parte 7 de la norma MPEG-2 [ISO06] y posteriormente fue ampliado en la parte 3 de MPEG-4 [ISO09]. La complejidad del algoritmo es bastante superior a la de sus antecesores a cambio de una mayor capacidad de compresión. Al igual que AC-3 soporta varios canales de audio para sistemas de altas prestaciones. En la actualidad este estándar está empleándose en la difusión de programas de televisión de alta definición y en transmisiones DVB-H por lo que es probable que en poco tiempo sustituya al estándar MPEG-2 capa II en las difusiones de televisión digital.

Para implementar estos descodificadores se localizaron códigos en lenguaje C de libre distribución que los implementaran, se migraron al entorno DSP y se adaptaron los interfaces para poder integrarlos en la tarea correspondiente del STB-IP. Una vez realizadas pruebas funcionales se comprobó que la carga computacional que requerían, así como las necesidades de memoria, eran bajas por lo que no se consideró necesario realizar ningún proceso de optimización sobre ellos.

Los códigos de referencia que se han empleado para disponer de estos descodificadores de audio son los siguientes: para MPEG-1 capas II y III se tomó como referencia el código MAD (MPEG *Audio Decoder*) [MAD], para AC-3 se empleó el código desarrollado en el proyecto liba52 [LIBA52] y para AAC se optó por la librería FAAD (*Freeware Advanced Audio Decoder*) [FAAD].

Las medidas de rendimiento de estos descodificadores integrados en el sistema completo basado en TMS320DM642 trabajando a 600 MHz, tanto en ciclos de reloj por segundo como en porcentaje de CPU, se presentan en la Tabla 8-3[192]. Todas las tramas elementales de audio han sido codificadas en estéreo con un régimen binario de 128 kbps.

	MPEG-2 Capa II	MPEG-2 Capa III	AC-3	AAC
Ciclos de Reloj ($\times 10^6$)	11.10	11.88	15.24	23.5
% CPU a 600 MHz	1.85%	1.98%	2.54%	3.92%

Tabla 8-3. Ciclos de reloj invertidos por los diferentes descodificadores de audio.

[192] Los resultados son los mismos si se utiliza el prototipo basado en el procesador TMS320DM6437.

8.2.3 Presentación de audio y vídeo

8.2.3.1 Funcionalidad de las tareas de presentación de audio y vídeo

Las tareas de presentación de audio y vídeo son conceptualmente muy sencillas. Tras recibir un mensaje por parte de la correspondiente tarea de descodificación que las informan de que se dispone de una AU descodificada, solicitan a sus respectivos controladores de dispositivos (McASP/McBSP o *VideoPort*) un *buffer* para alojar los datos descodificados. Seguidamente, realizan una copia de los datos sobre dicho *buffer*, en el caso del audio, y dado que el número de datos a copiar es pequeño, se realiza directamente con un bucle; sin embargo, para el vídeo es necesario realizar una copia masiva de datos[193] por lo que es imprescindible emplear transferencias explícitas de DMA. Una vez finaliza la copia se informa a los controladores de los dispositivos y a las respectivas tareas de descodificación para que puedan continuar con la siguiente imagen o trama de audio. Finalmente, las tareas se quedan bloqueadas esperando un nuevo aviso por parte de las tareas de descodificación.

En el caso de la presentación de vídeo se han empleado las mismas metodologías de optimización de las transferencias de datos que han sido utilizadas en el descodificador de vídeo MPEG-2 (ver apartado 4.2.2.2). Además, en esta tarea descansa la algoritmia necesaria para poder realizar la sincronización entre las tramas de audio y vídeo tal y como se presenta en el siguiente apartado.

8.2.3.2 Sincronización de las tramas de audio y vídeo

La necesidad de la sincronización entre las tramas de audio y vídeo de un programa se debe fundamentalmente a dos problemas:

1. La frecuencia del reloj de 27 MHz con el que se obtienen el DTS y el PTS asociados a cada AU no es idéntica en el transmisor y en el receptor, lo que provoca que el número de imágenes de vídeo o tramas de audio que se reproducen por segundo no sean exactamente las mismas en se codifican. Este hecho origina que, si la frecuencia en el emisor es mayor que en receptor, se acumulen AUs que no pueden ser presentadas. Si la frecuencia en el emisor es menor que la frecuencia en el receptor, entonces éste puede quedarse sin AUs que presentar al usuario.

2. La generación de las tramas de audio y vídeo poseen latencias diferentes puesto que el proceso de codificación de cada una de ellas requiere distinto tiempo. Este hecho provoca que las tramas no se encuentren sincronizadas cuando se encapsulan dentro de la trama de transporte.

A estos dos efectos habituales en los sistemas de transmisión de televisión digital hay que añadir un tercero que aparece cuando la trama de transporte es enviada a través de una red de difusión de paquetes como es la red Ethernet. Este tipo de redes tiene una tasa de transferencia variable, de manera que la cadencia de llegada de las diferentes AUs puede no ser uniforme.

Para eliminar estos problemas de sincronización se ha optado por almacenar en los *buffers* compartidos de audio y vídeo (ver Figura 8-2) varias AUs antes de comenzar el proceso de descodificación y definir unos umbrales máximo y mínimo de llenado de dichos *buffers*. Inicialmente, cuando se selecciona un canal para ser visualizado, ambos *buffers* se llenan hasta la mitad y a partir de ese momento se

[193] El controlador del puerto de vídeo requiere que el formato de la imagen descodificada sea 4:2:2 por lo que para el caso de una imagen PAL es necesario transferir 720×576×2= 810 kB.

arranca el proceso de descodificación tanto del audio como del vídeo. Con ello se resuelven los problemas planteados del siguiente modo:

1. Cuando la frecuencia de reloj del transmisor y el receptor es ligeramente diferente[194], pueden darse dos situaciones: que la frecuencia del transmisor sea mayor que la del receptor, en cuyo caso el *buffer* compartido se irá llenando[195] al acumularse AUs en él, o que la frecuencia de transmisión sea menor que la de recepción y el *buffer* se vacíe progresivamente. En el primer caso, cuando el nivel de llenado supera un umbral máximo que se ha fijado en el 80%, la tarea de presentación correspondiente no envía alguna de las imágenes/tramas descodificadas al monitor/altavoces. Con esta estrategia se reduce el tiempo de procesado de la AU lo que provoca que se comience antes la descodificación de la siguiente y, por tanto, se vacíe ligeramente el *buffer* correspondiente. Este algoritmo se repite durante unos cuantos segundos[196] hasta que los *buffers* compartidos alcanzan un nivel de llenado próximo al 50%. Para el segundo caso, el algoritmo implementado se basa en repetir una vez la presentación de una imagen. Con ello se incrementa el tiempo de procesado de la AU lo que provoca que se llene ligeramente el *buffer* compartido. Al igual que en el caso anterior este algoritmo se aplica durante unos cuantos segundos hasta que el nivel de llenado de los *buffers* se aproxima al 50%.

2. La falta de sincronización de las AUs en la trama de transporte se resuelve empleando la información contenida en los PTSs y DTSs asociados a los ES de audio y vídeo. La tarea de presentación de vídeo recibe los PTSs (ver Anexo B apartado 8.1.2) asociados a las últimas AUs de audio y vídeo descodificadas. A partir de esta información se ha codificado un algoritmo [Est07] en dicha tarea[197] que, en función del desfase entre ellos, toma las siguientes decisiones:

- Si la diferencia entre ambos es menor que el tiempo de presentación de una imagen de vídeo, ésta se presenta normalmente y se puede decir que el sistema está sincronizado.
- Si la diferencia es mayor que el tiempo de presentación de una imagen y el PTS de vídeo es mayor que el de audio, implica que no hay sincronización puesto que el vídeo va “adelantado” respecto al audio. En este caso se repite la última imagen descodificada dos veces para retrasar el vídeo.
- Si la diferencia es mayor que el tiempo de presentación de una imagen y el PTS de vídeo es menor que el de audio, implica que no hay sincronización puesto que el vídeo va “retrasado” respecto al audio. Para solventar este problema es necesario “adelantar” el vídeo lo que implica no descodificar alguna de las imágenes almacenadas en el *buffer* compartido. Sin embargo,

[194] El desfase entre las frecuencias de reloj de transmisión y recepción se resuelve tradicionalmente modificando dinámicamente la frecuencia del reloj del receptor para irla aproximando a la del transmisor. Para ello se emplean los PCRs que se incluyen en algunos de los TSPs que componen la trama de transporte. El uso de esta técnica implica disponer de un VCO que pueda ser ajustado dinámicamente, sin embargo, las tarjetas de prototipos empleadas no disponen de este recurso por lo que esta técnica no ha podido ser empleada.

[195] Si la frecuencia de transmisión y recepción difieren en un 1% en el caso del vídeo se acumulará una imagen cada 4 segundos.

[196] Es necesario que este proceso se realice gradualmente durante un tiempo prolongado para que el usuario no perciba la eliminación de imágenes.

[197] El hecho de que el algoritmo propuesto se aplique sobre el vídeo en lugar de sobre el audio se debe a que la repetición/eliminación de una imagen tiene un menor efecto sobre la calidad subjetiva del programa que percibe el espectador.

no todas las imágenes que se encuentran en el *buffer* pueden ser eliminadas[198], por lo que es necesario modificar el descodificador de vídeo para que ante esta eventualidad descarte la descodificación de una imagen de tipo B, en el caso de que éstas existan en la trama, o tipo P en el caso de que no se disponga de imágenes de tipo B. Por tanto, la tarea de presentación de vídeo informa a la de descodificación para que no procese una AU y de este modo se gane el tiempo de presentación de una imagen.

3. Los desfases producidos por el *jitter* de la red Ethernet se resuelven de forma automática con el uso de los *buffers* compartidos que actúan como colas FIFO, posibilitando un régimen de salida de datos constante hacia los descodificadores aunque el régimen de entrada sea variable.

[198] Hay que tener en cuenta que si no se descodifica una imagen tipo I, todas las imágenes tipo P y B relacionadas con ella no podrán ser descodificadas. Igualmente si no se descodifica una imagen P, todas las imágenes P y B que dependen de ella tampoco podrán descodificarse correctamente.

9 ANEXO C: RELACIÓN DE RECOMENDACIONES PARA LA OPTIMIZACIÓN DE ALGORITMOS DE DESCODIFICACIÓN DE VÍDEO

Para facilitar su lectura, en este anexo se relacionan todas las fichas que resumen las técnicas de optimización empleadas para reducir el tiempo de ejecución de los algoritmos de descodificación de vídeo que fueron presentadas en el capítulo 4.

Referencia: Configuración de la memoria interna [ficha 001].
Descripción: Inicialmente configurar la totalidad de la memoria interna como caché. Realizar pruebas de rendimiento global y por funciones y tomarlas como una referencia del rendimiento. Realizar una ordenación de las funciones y los *buffers* en función del interés de su ubicación en memoria interna. Priorizar las funciones con un mayor coste computacional y los *buffers* que sean utilizados por esas funciones. Realizar pruebas con diferentes repartos y almacenar los resultados. Repetir la ordenación y las pruebas cada vez que se produzcan cambios importantes en la estructura del código a lo largo del proceso de optimización.
Aplicabilidad: Procesadores en los que es posible configurar un reparto entre la memoria caché y la memoria interna de propósito general.

Referencia: Ubicación de funciones y *buffers* en memoria interna [ficha 002].
Descripción: Durante el proceso de optimización se procurará limitar el tamaño de las funciones y *buffers* que realizan la descodificación de los macrobloques de manera que quepan en la memoria interna disponible. La pertenencia al conjunto de funciones y *buffers* que realizan la descodificación del macrobloque se utilizará como criterio adicional para priorizar la inclusión de estos elementos en la memoria interna.
Aplicabilidad: Funciones y *buffers* que realizan el proceso de descodificación macrobloque a macrobloque.

Referencia: Organización del código en memoria [ficha 003].
Descripción: Generar secciones de código de tamaño múltiplo del tamaño de la caché de programa de nivel 1. Agrupar en cada una de estas secciones aquellas funciones que se ejecuten de forma secuencial. Evaluar la influencia que tienen en el rendimiento del descodificador los diversos fragmentos del código que se generan para determinar si se alojan en memoria interna o externa. Ubicar cada sección a partir de direcciones de memoria que sean múltiplo del tamaño de dicha caché. Durante el proceso de optimización no debe aplicarse un gran esfuerzo a organizar el código en la memoria de programa puesto que, según se vayan realizando nuevas optimizaciones, el tamaño de las funciones va cambiando significativamente lo que obliga a tener que modificar continuamente el reparto.
Aplicabilidad: Aplicable a todos descodificadores que se ejecuten en procesadores en los que existan niveles de memoria caché de programa.

Referencia: Reducción del tamaño del código [ficha 004].
Descripción: Reducir el tamaño de las funciones que componen el bucle de descodificación de los macrobloques. Agrupar todas las funciones en una sección y comprobar si todas ellas pueden alojarse en la memoria caché de programa de nivel 1. Ubicar la sección en memoria interna de propósito general alineada al tamaño de la caché de nivel 1.
Aplicabilidad: Aplicable a todos los descodificadores que se ejecuten en procesadores en los que existan niveles de memoria caché de programa.

Referencia: Organización de los datos en memoria [ficha 005].
Descripción: Definir los *buffers* de datos con un tamaño múltiplo del tamaño de la línea de la caché de datos de nivel 1 y alinear su dirección de comienzo a una posición de memoria que también sea múltiplo de dicho tamaño. Agrupar los *buffers* que sean empleados por alguno de los módulos funcionales del descodificador en una sección de memoria y alinear la sección a una posición de memoria que sea múltiplo del tamaño total de una vía de la caché de datos de nivel 1. Los *buffers* que por su tamaño no puedan ubicarse en la memoria interna se alinearán al tamaño de la caché de nivel 2 en la memoria externa para facilitar el acceso a los mismos. El esfuerzo aplicado a distribuir los *buffers* de datos en la memoria debe realizarse de forma gradual a lo largo de todo el proceso de optimización puesto que los cambios en el código a menudo influyen, tanto en el tamaño de los datos que hay que manejar, como en su reparto en la memoria. Sólo cuando se finaliza el proceso de optimización se debe realizar un ajuste fino de la ubicación de los datos en las diferentes secciones de memoria.
Aplicabilidad: Aplicable a todos los descodificadores que se ejecuten en procesadores en los que existan niveles de memoria caché de datos.

Referencia: Paralelización de las transferencias de los macrobloques de referencia (fase C) con la IDCT (fase B) y del macrobloque reconstruido (fase E) con la extracción de los vectores de movimiento (fase A) [ficha 006].
Descripción: Definir en la memoria interna tantos *buffers* del tamaño de un macrobloque como sean necesarios para alojar todos los píxeles implicados en el proceso de reconstrucción de un macrobloque. Configurar el controlador de DMA inmediatamente después de obtener los vectores de movimiento, para transferir a los *buffers* intermedios los píxeles de las imágenes descodificadas anteriormente, mientras se ejecutan otras partes del algoritmo. Realizar todas las operaciones necesarias para descodificar cada macrobloque empleando exclusivamente estos *buffers* intermedios. Transferir el resultado a la imagen que se está descodificando en ese momento empleando de nuevo el controlador de DMA.
Aplicabilidad: Esta metodología puede aplicarse a cualquier otro descodificador de vídeo que procese las imágenes macrobloque a macrobloque en el que sea necesario el acceso frecuente a datos que se encuentren en memoria externa. En el caso del acceso a los píxeles de las imágenes de referencia es necesario que el controlador de DMA permita realizar transferencias bidimensionales puesto que los píxeles de la referencia no se encuentran en posiciones contiguas de memoria.

Referencia: Reducción del número de transferencias de DMA necesarias para realizar la compensación de movimiento [ficha 007].
Descripción: Declarar *buffers* en memoria interna para alojar en ellos todos los píxeles que permitan realizar la compensación de movimiento con resolución fraccionaria. Transferir todos los píxeles necesarios para calcular la compensación de movimiento con los datos de estos *buffers*. Obtener el macrobloque de referencia, almacenarlo en otro *buffer* intermedio y sumarle el error de predicción. Una vez obtenido el macrobloque reconstruido se transfiere a la imagen descodificada empleando de nuevo el controlador de DMA.
Aplicabilidad: A todos los estándares de codificación que empleen predicción con resolución de ½ píxel.

Referencia: Eliminación de esperas de la CPU para paralelizar la transferencia de un macrobloque reconstruido (fase E) con la extracción de los coeficientes de la DCT del siguiente macrobloque (fase B) [ficha 008].
Descripción: Comprobar si la CPU debe esperar a que finalicen las transferencias por DMA. En caso de que así sea, duplicar el tamaño de los *buffers* definidos en memoria interna de modo que el controlador de DMA realice transferencias a/desde uno de ellos mientras que la CPU procesa y almacena datos del siguiente macrobloque en el otro. Tras finalizar el procesamiento de un macrobloque se intercambian los *buffers* para el procesamiento del siguiente.
Aplicabilidad: A aquellos descodificadores en los que la CPU deba esperar a que finalice el movimiento de datos entre memoria interna y externa cuando se realicen transferencias explícitas por parte del controlador de DMA.

Referencia: Reorganización del bucle principal de descodificación [ficha 009].
Descripción: Rellenar los tiempos en los que el controlador está transfiriendo por DMA los píxeles necesarios para poder realizar la compensación de movimiento de un macrobloque con instrucciones que permitan que la CPU pueda continuar el procesamiento de otros datos. Dado que éstos no pueden pertenecer al mismo macrobloque, es necesario comenzar a descodificar el siguiente por lo que el bucle principal de descodificación se ve modificado para manejar simultáneamente datos de dos macrobloques diferentes. Aplicar esta técnica implica almacenar más *buffers* en memoria interna de propósito general lo puede conllevar tener que sacar de esta memoria otros *buffers* o funciones con la correspondiente pérdida de rendimiento. Por tanto, es necesario confirmar que la CPU realiza esperas de las transferencias de DMA antes de abordar la modificación del bucle de descodificación de macrobloques.
Aplicabilidad: Esta técnica, al igual que la [ficha 008], es aplicable para cualquier descodificador que se ejecute en un procesador en el que la CPU deba esperar la finalización de la transferencia por DMA de los píxeles necesarios para realizar la compensación de movimiento.

Referencia: Alineamiento de todas las transferencias de DMA empleadas en la compensación de movimiento [ficha 010].
Descripción: Forzar que todas las transferencias tengan un tamaño múltiplo de 32 bits y que sus direcciones de origen y destino también sean múltiplos de 32 bits. Si alguna transferencia no cumple estos requisitos, el bloque a transferir debe enmarcarse en otro de mayor tamaño que sí esté alineado y gestionar el desplazamiento del macrobloque de referencia respecto al bloque de píxeles transferidos. Al transferir más píxeles de los necesarios y tener que gestionar el desplazamiento de los datos útiles respecto a los transferidos es necesario evaluar si realmente se produce una mejora en el rendimiento global del descodificador
Aplicabilidad: A todos los procesadores en los que la eficiencia de las transferencias de DMA se incremente cuando el tamaño de los datos a transferir y las direcciones de origen y destino requieran algún requisito específico respecto a su tamaño y/o alineamiento en memoria.

Referencia: Gestión de las colas de petición de transferencias del controlador de DMA [ficha 011].
Descripción: Analizar el número de peticiones de transferencias que realiza el descodificador. Repartir homogéneamente entre las colas de petición del controlador de DMA las transferencias que se soliciten para evitar la sobrecarga de alguna.
Aplicabilidad: La técnica propuesta debe aplicarse a procesadores en los que existan colas de petición de transferencias que nunca deben desbordarse para evitar paradas (*stalls*) en el *pipeline* de la CPU.

Referencia: Transferencias de macrobloques reconstruidos por tiras [ficha 012].
Descripción: Almacenar los macrobloques descodificados de la imagen en un doble *buffer* intermedio de modo que, cuando se dispone de todos los que componen una tira de la imagen, se realiza una única transferencia de DMA. En estos casos hay que tener en cuenta que su aplicación requiere definir un doble *buffer* en memoria interna de propósito general de elevado tamaño, lo que no siempre es posible. Si no hay memoria interna de propósito general disponible, es necesario trasladar algunas funciones/datos a memoria externa para poder alojar estos *buffers*, con la correspondiente pérdida de rendimiento. Valorar si la mejora que proporciona transferir mediante el controlador de DMA tiras de macrobloques compensa respecto a la pérdida de rendimiento que puede suponer tener que trasladar a memoria externa algunas funciones y/o datos.
Aplicabilidad: Esta técnica puede ser aplicada a los procesadores en los que la eficiencia de las transferencias de DMA se incrementa si éstas son de tamaño elevado.

Referencia: Empleo de funciones de librería [ficha 013].
Descripción: Analizar las librerías optimizadas que facilitan los fabricantes de los procesadores para tratar de localizar aquellas funciones del descodificador que pueden ser sustituidas por alguna de estas funciones optimizadas. Adaptar el código del descodificador para cumplir los requisitos que requieran las funciones de librería. Evaluar la mejora que supone en el rendimiento del descodificador ya que en algunas ocasiones puede que la inclusión de estas funciones no implique una mejora global.
Aplicabilidad: Esta técnica es aplicable de forma general.

Referencia: Funciones en las que se realizan cargas, promedios y almacenamientos de varios píxeles [ficha 014].
Descripción: Recodificar las funciones en las que es posible emplear las instrucciones que permiten cargar/almacenar varios píxeles en los registros internos de la CPU y las instrucciones que permiten trabajar con ellos como si se tratara de varios datos de menor tamaño. Inicialmente se realiza una codificación de las funciones en lenguaje ensamblador lineal y en función del grado de paralelización que obtenga el compilador y la complejidad de la función, se valora la posibilidad de paralelizar las instrucciones manualmente.
Aplicabilidad: A todos los procesadores que posean una arquitectura SIMD. Para ello es necesario estudiar con detalle el repertorio de instrucciones del procesador. La función desarrollada es directamente aplicable a otros descodificadores que se desarrollen empleando procesadores que posean un repertorio de instrucciones compatible con el del TMS320DM642.

Referencia: Emplear instrucciones de empaquetado y desempaquetado de datos [ficha 015].
Descripción: Emplear las instrucciones que permitan compactar/expandir los datos que se almacenan en los registros internos de la CPU cuando se recodifican funciones que manejen datos de diferentes tamaños.
Aplicabilidad: A todos los procesadores que posean una arquitectura SIMD y que incluyan instrucciones de empaquetado y desempaquetado de datos. La funcionalidad desarrollada (suma del error de predicción), con algunas pequeñas modificaciones, puede ser directamente aplicada a otros descodificadores que se desarrollen para los procesadores de la misma familia.

Referencia: Asignación de prioridades a las tareas que forman el sistema [ficha 016].
Descripción: Evaluar el tiempo de ejecución de cada una de las tareas que forman el sistema y la posibilidad de pérdida de datos por parte de alguna de ellas si se demora su ejecución. Asignar mayor prioridad a aquellas tareas que requieran una atención más rápida y cuyo tiempo de ejecución sea bajo. Asignar menor prioridad a aquellas tareas que tengan un tiempo de ejecución elevado para que puedan ser interrumpidas por el resto.
Aplicabilidad: A cualquier sistema en el que exista un RTOS que permita asignar prioridades a las tareas que componen el sistema.

Referencia: Gestión de las peticiones de transferencias de DMA con el descodificador integrado en el sistema completo [ficha 017].
Descripción: Analizar las peticiones que realizan el resto de elementos que componen el sistema (resto de tareas y controladores de periféricos). Asignar sus peticiones a las diferentes colas teniendo en cuenta las que realiza el descodificador para lograr una distribución final uniforme entre todas las colas de petición. Comprobar experimentalmente que en ningún caso se encolan simultáneamente más solicitudes que el tamaño de cada una de las colas.
Aplicabilidad: A procesadores en los que el controlador de DMA disponga de colas de petición con prioridad.

Referencia: Ubicación en la jerarquía de memoria del código y los datos de las tareas que componen el sistema [ficha 018].
Descripción: Analizar el tamaño del código y los datos del sistema así como la pila asignada a cada una de las tareas. Ubicar en memoria interna la pila de las tareas así como el código y los datos del descodificador de vídeo siguiendo los criterios mostrados en el apartado 4.2.2.1.
Aplicabilidad: A los sistemas gestionados mediante tareas.

Referencia: Criterios adicionales de selección de bloques funcionales que deben alojarse en memoria interna [ficha 019].
Descripción: Identificar las funciones del descodificador que comportan una mayor carga computacional así como aquellas que provocan más fallos en el acceso a memoria caché de programa. Ubicar estas funciones en memoria interna y evaluar la mejora que supone en el rendimiento del descodificador. Realizar varias distribuciones hasta obtener el mayor rendimiento. Esta técnica debe aplicarse a lo largo del proceso de optimización puesto que la aplicación de las otras técnicas de optimización puede provocar cambios significativos en el tamaño y rendimiento de las funciones, que harán que sea necesario modificar la distribución.
Aplicabilidad: A cualquier descodificador en el que el código no pueda alojarse completamente en la memoria interna de propósito general.

Referencia: Tamaño de las transferencias de DMA para la compensación de movimiento con resolución de ¼ píxel [ficha 020].
Descripción: Analizar los píxeles implicados en la compensación de movimiento. Declarar *buffers* en memoria interna en los que se puedan almacenar todos los píxeles necesarios para realizar la compensación de movimiento con resolución de ¼ píxel. Transferir a estos *buffers* tanto el/los macrobloque/s de referencia como algunos píxeles de los macrobloques vecinos que permiten realizar la compensación de movimiento. Aplicar el filtrado sobre los datos transferidos almacenando el resultado en otro *buffer* intermedio alojado en memoria interna. El resto del proceso empleado para calcular el macrobloque reconstruido es idéntico al empleado en MPEG-2 (ver [ficha 007]).
Aplicabilidad: A todos los descodificadores que empleen vectores de movimiento con resolución de ¼ píxel.

Referencia: Tamaño de las transferencias de DMA para macrobloques en los que existen diferentes particiones [ficha 021].
Descripción: Definir un único *buffer* en memoria interna que permita alojar todas las posibles referencias que pueda tener un macrobloque en el caso en el que éste pueda estar dividido en otros de menor tamaño. Transferir a este *buffer* las referencias, incluyendo los píxeles adicionales necesarios para poder realizar la compensación de movimiento. Almacenar en una estructura interna el desplazamiento de cada uno de los bloques transferidos respecto al comienzo del *buffer*, para posteriormente poder realizar la compensación de movimiento. Guardar el resultado de la compensación de movimiento y la suma del error de predicción en uno de los doble *buffers* destinados a almacenar el macrobloque reconstruido. Finalmente, transferir el resultado a la imagen reconstruida.
Aplicabilidad: A descodificadores en los que un macrobloque pueda descomponerse en diversas combinaciones de bloques de menor tamaño.

Referencia: Reorganización del bucle principal de descodificación en H.264 [ficha 022].
Descripción: Rellenar los tiempos en los que el controlador está transfiriendo por DMA los píxeles necesarios para poder realizar la compensación de movimiento de un macrobloque con instrucciones que permitan a la CPU continuar el procesamiento de otros datos. Una vez reorganizado el bucle de descodificación, evaluar si en alguna situación la CPU debe esperar a la finalización de alguna transferencia. En este caso se paralelizará la descodificación con alguno de los módulos funcionales del proceso de descodificación del siguiente macrobloque.
Aplicabilidad: Esta técnica es aplicable para cualquier descodificador que se ejecute en un procesador en el que la CPU deba esperar la finalización de la transferencia por DMA de los píxeles necesarios para realizar la compensación de movimiento.

Referencia: Gestión de *buffers* intermedios para almacenar la información necesaria para la descodificación de macrobloques vecinos [ficha 023].
Descripción: Definir *buffers* intermedios en los que poder almacenar temporalmente los píxeles de macrobloques vecinos que son necesarios para la descodificación. La inclusión de estos *buffers* en memoria interna agiliza las copias de datos a/desde ellos. En el caso de tener que definir *buffers* de elevado tamaño puede ser necesario desalojar algunas funciones de la memoria interna. Antes de consolidar estos cambios es necesario evaluar de nuevo el rendimiento del descodificador para confirmar que se produce una mejora.
Aplicabilidad: Aplicable a cualquier descodificador que realice el procesamiento de la imagen a nivel de macrobloque y en el que la descodificación de uno de ellos emplee píxeles pertenecientes a macrobloques vecinos.

Referencia: Eliminación de condiciones en algoritmos de cálculo de píxeles [ficha 024].
Descripción: Calcular todas aquellas condiciones que aparezcan en el algoritmo. A partir de las condiciones generar para cada una de ellas máscaras de bits. Realizar todos los posibles cálculos independientemente de las condiciones que se cumplan. Asignar a las variables de salida los datos correctos aplicando las máscaras que se han calculado previamente.
Aplicabilidad: A algoritmos en los que se realicen cálculos intensivos con píxeles en los que las operaciones dependen de una serie de condiciones que se van evaluando a lo largo de la función.

Referencia: Codificación de funciones en ensamblador paralelizado manualmente [ficha 025].
Descripción: Identificar funciones de tamaño reducido y elevada carga computacional. Recodificar la función inicialmente en ensamblador lineal para comprobar su funcionalidad. Evaluar la mejora en el rendimiento y analizar el nivel de paralelización de las instrucciones que logra el compilador. En función de los resultados obtenidos, valorar la opción de paralelizar las instrucciones manualmente para mejorar el rendimiento. Durante todo el proceso se debe tener en cuenta que es posible que la sobrecarga asociada a la llamada a la función optimizada pueda reducir la mejora esperada.
Aplicabilidad: A funciones de tamaño reducido que sean ejecutadas en múltiples ocasiones y que, por tanto, supongan una elevada carga computacional.

Referencia: Empleo de instrucciones de multiplicación/acumulación y desplazamiento para operaciones de interpolación [ficha 026].
Descripción: Identificar algoritmos en los que se realice el filtrado de varios píxeles. Obtener los píxeles necesarios para poder realizar el filtrado y almacenar varios en una variable de 32 bits (ver [ficha 014]). Emplear las instrucciones de multiplicación con acumulación para calcular la salida del filtro. Analizar si el número de bits empleados por los datos obtenidos previamente permite empaquetar varios en variables de 32 bits. Emplear instrucciones de suma y/o desplazamiento para operar con los datos almacenados en estas variables (ver [ficha 015]). Empaquetar los datos obtenidos antes de almacenarlos.
Aplicabilidad: A algoritmos en los que se realiza la interpolación de los píxeles de un bloque.

Referencia: Resolución de problemas de funcionamiento en tiempo real cuando el tiempo de descodificación de algunas imágenes supera el intervalo de presentación [ficha 027].
Descripción: Cuando la carga computacional media del descodificador es próxima al 100% de la CPU, no es suficiente con asegurar que es menor al 100%, sino que es necesario analizar el tiempo de descodificación de cada una de las imágenes. Si el tiempo de descodificación de alguna de ellas supera el intervalo de presentación de una imagen, es necesario definir un *buffer* de imágenes descodificadas en el que almacenar las imágenes ya procesadas. La inclusión de este *buffer* provoca una latencia en la presentación proporcional al número de imágenes almacenadas. El tamaño del *buffer* debe ajustarse para garantizar que el tiempo que se invierte en descodificar el número de imágenes que se pueden alojar en él sea siempre menor que el tiempo de presentación de todas ellas.
Aplicabilidad: Sistemas en los que el tiempo de descodificación de algunas imágenes sea superior al intervalo de presentación.

Referencia: Criterios adicionales para la configuración de la memoria interna [ficha 028].
Descripción: Analizar los posibles repartos de las memorias internas entre memoria caché y memoria de propósito general tal y como se indicó en [ficha 001]. En el nivel 1 de memoria de programa y de datos priorizar inicialmente su configuración como memoria caché realizando pruebas de rendimiento global con diferentes repartos de la memoria de nivel 2. Una vez fijado el tamaño de la memoria de nivel 2 repetir las pruebas para diferentes configuraciones de las memorias de nivel 1. En cada uno de estos pasos se ordenarán las funciones y los *buffers* en función del interés de su ubicación en memoria interna. El procedimiento debe repetirse cada vez que se produzcan cambios importantes en la estructura del código a lo largo del proceso de optimización.
Aplicabilidad: Procesadores en los que es posible configurar un reparto entre la memoria caché y la memoria interna de propósito general en los diferentes niveles de la arquitectura de memoria.

Referencia: Empleo de los controladores de DMA para optimizar las transferencias de datos [ficha 029].
Descripción: Almacenar en memoria interna de datos L1D los parámetros necesarios para realizar todas las transferencias mediante en controlador de EDMA. Transferir los parámetros almacenados a los registros de configuración del EDMA empleando el controlador de IDMA. Configurar el controlador EDMA para que enlace las transferencias configuradas. Comprobar que ha finalizado la última antes de comenzar a usar los datos transferidos.
Aplicabilidad: Procesadores en los que existen varios controladores de DMA que permiten optimizar las transferencias entre memoria interna y memoria externa.

10 BIBLIOGRAFÍA

[4i2i05] 4i2i Communications technical staff, H.264/MPEG-4 Part-10 Baseline Video Decoder IP Core, 4i2i Communications Ltd., revision 1.0, 2005

[AIC23] TLV320AIC23 “Low-Power Stereo CODEC with HP Amplifier”. Disponible en http://focus.ti.com/docs/prod/folders/print/tlv320aic32.html

[AIC33] TLV320AIC33 “Low-Power Stereo CODEC with 10 Inputs, 7 Outputs, HP/Speaker Amplifier and Enhanced Digital Effects”. Disponible en http://focus.ti.com/docs/prod/folders/print/tlv320aic33.html

[ARG03] Satish Arora, Ratna Reddy, Jeremiah Golston. “Multichannel MPEG-2 Decoder Implementation for the DM642 Multimedia Processor”. *Global Signal Processing Expo*. GSPx 2003. Los libros de actas no están disponibles.

[ATEME] ATEME. “AMK 430 AVC encoder”. Disponible en http://www.ateme.com.

[ATSC05] ATSC. Digital Audio Compression Standard (AC-3, E-AC-3) revision B. Document A/52B, 14 June 2005. Disponible en http://www.atsc.org/standards/ a_52b.pdf

[BCK07] Eric Bodden, Malte Clasen, Joachim Kneis. “Arithmetic Coding revealed”. McGill University. Sable Research Group www.sable.mcgill.ca.

[BF549] ADSP-BF549: High Performance Convergent Multimedia Blackfin Processor. Disponible en http://www.analog.com/en/embedded-processing-dsp/ blackfin/adsp-bf549/processors/product.html#Diagrams.

[BUL04] Using the Philips PCF50606 with the Intel PXA27x Processor Family. Disponible en http://int.xscale-freak.com/XSDoc/PXA27X/28000801.pdf.

[CCC+09] Chih-Da Chien, Cheng-An Chien, Jui-Chin Chu, Jiun-In Guo, Ching-Hwa Cheng. “A 252Kgates/4.9Kbytes SRAM/71mW Multistandard Video Decoder for High Definition Video Applications”. *ACM Transactions on Design Automation of Electronic Systems*, vol. 14 Issue 1, January 2009. Paper 17. Pages: 17.

[CCFS02] Saverio Cacopardi, Michele Caponi, Fabrizio Frescura, Simone Sabina. "A DSP based MPEG-2 video decoder for HDTV or multichannel SDTV”, *IEEE Workshop on Multimedia Signal Processing, 2002*, pp. 134-137, 9-11 Dec. 2002.

[CCKM03] Byeong-Doo Choi, Kang-Sun Choi; Sung-Jea Ko; Aldo W. Morales. "Efficient real-time implementation of MPEG-4 audiovisual decoder using DSP and RISC chips" *IEEE International Conference on Consumer Electronics, 2003*. ICCE. 2003, pp. 246-247, 17-19 June 2003.

[Cha89] W. K. Cham. “Development of Integer Cosine Transforms by the Principle of Dyadic Symmetry", IEE Proceedings, vol. 136, pt. I, no. 4, pp. 276-282, August 1989.

[CHC+05] To-Wei Chen, Yu-Wen Huang, Tung-Chien Chen, Yu-Han Chen, Chuan-Yung Tsai, Liang-Gee Chen. "Architecture design of H.264/AVC decoder with hybrid task pipelining for high definition videos”, *IEEE International Symposium on Circuits and Systems*, ISCAS 2005, pp. 2931-2934, vol. 3, 23-26 May 2005

[DIVX] Implementación parcial de la norma MPEG-4. Disponible en http://www.divx.com/.

[DM64x] Relación de elementos de la familia. Disponible en http://focus.ti.com/paramsearch/docs/parametricsearch.tsp ?family=dsp&sectionId=2&tabId=1852&familyId=1300.

[DSPBIOS] DSP/BIOS 6.x Real-Time Operating System. Disponible en http://focus.ti.com/docs/toolsw/folders/print/dspbios6.html.

[DVB07] Transport of MPEG 2 Transport Stream (TS) Based DVB Services over IP Based Networks. Digital Video Broadcasting. DVB Document A086 Rev. 5 July 2007. Disponible en http://www.dvb.org/technology/ standards/a086r5.tm3727r2.dTS102034.V1.3.1.DVB-IP1.3.pdf.

[ELECARD] YUV Player. Visor de ficheros en formato YUV. Disponible en http://www.elecard.com/en/products/professional/analysis/yuv-viewer.html.

[ESP+09] Esther Estevez, David Samper, Fernando Pescador, Matías J. Garrido, César Sanz. "Implementation of a media synchronization algorithm for multistandard IP set-top box systems". *SPIE International Symposium on Microtechnologies for the New Millennium. VLSI Circuits and Systems Conference*. Dresde, Alemania, 4-6 May 2009.

[Est07] Esther Estévez. Desarrollo de un Algoritmo de Sincronización de Audio y Vídeo para TV Digital. Proyecto Fin de Carrera de la EUITT-UPM. Noviembre 2007.

[ETHTV] SIDSA. EtherTV. Disponible en http://www.sidsa.com/DATASHEETS/ SIDSA_EtherTV_brochure_big_box.pdf.

[EVM642] DM642 Evaluation Module with TVP Video Decoders, 720 MHz. Disponible en http://www.spectrumdigital.com/product_info.php ?&products_id=121.

[EVM6437] Spectrum Digital. TMS320DM6437 Evaluation Module Technical Reference. Diciembre 2006. Disponible en http://c6000.spectrumdigital.com/evmdm6437/reve/files/EVMDM6437_TechRef.pdf.

[FAAD] Freeware Advanced Audio Coder (MPEG2-AAC, MPEG4-AAC) library. Código en punto fijo de libre distribución que implementa el estándar ATSC A/52. Disponible en http://sourceforge.net/projects/faac/.

[Fer98] José Manuel Fernández. “Arquitecturas VLSI para la codificación de imágenes en tiempo real”. Tesis doctoral leída en el Departamento de Ingeniería Electrónica de la E.T.S.I. de Telecomunicación de la U.P.M. Mayo 1998.

[FFMPEG] *Software* de libre distribución disponible en http://www.ffmpeg.org.

[Gar04] Matías J. Garrido. “Arquitectura versátil para la codificación de vídeo multi-estándar: aportaciones metodológicas para el diseño de sistemas reutilizables y sistemas en un chip”. Tesis doctoral leída en el Departamento de Ingeniería Electrónica de la E.T.S.I. de Telecomunicación de la U.P.M. Septiembre 2004.

[Gol06] Salomon W. Golomb. “Run-Length Encodings”, *IEEE Transactions on Information Theory*. vol. 12, pp. 399-401, July 1966.

[GSS+06] Hongxing Guo, Tao Sheng, Weiping Sun, Jingli Zhou, Shengsheng Yu. "Cache optimization for an embedded MPEG-4 video decoder", *8th International Conference on Signal Processing*, vol. 1, 16-20 November 2006.

[HD10] Honghua Hu, Derong Chen. "Optimization techniques for a DSP based H.264 decoder". *2nd International Conference on Signal Processing Systems (ICSPS)*, vol. 1, pp.V1-649-V1-652. 5-7 July, 2010.

[HJC+08] Ting-Yu Huang, Guo-An Jian, Jui-Chin Chu, Ching-Lung Su, Jiun-In Guo. "Joint algorithm/code-level optimization of H.264 video decoder for mobile multimedia applications". *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2189-2192, March 31-April 4, 2008.

[HJKH03] Michael Horowitz, Anthony Joch, Faouzi Kossentini, Antti Hallapuro. “H.264/AVC baseline profile decoder complexity analysis” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no.7, pp. 704-716, July 2003.

[HL97] Ching-Yu Hung, Paul Landman. “Compact Inverse Discrete Cosine Transform Circuit for MPEG Video Decoding”, *Proceedings of the IEEE Workshop on Signal Processing Systems (SIPS),* pp. 364-373, November 1997.

[HMK02] Atsushi Hatabu, Takashi Miyazaki, Ichim Kumda. “QVGA/CIF resolution MPEG-4 video codec based on a low-power and general-purpose DSP”. *IEEE Workshop on Signal Processing Systems*, pp. 15-20. 16-18 October, 2002.

[HSMC04] Y. Hu, A. Simpson, K. Mcadoo, J. Cush. “A high definition H.264/AVC hardware video decoder core for multimedia SoC’s”. *Proceedings of the IEEE International Symposium on Consumer Electronics ISCE 2004,* pp. 385–389. 1-3 September, 2004.

[IEEE98] IEEE G.216. “Presentation of IEEE G.216 Video Compression Measurement Subcommitte on IEEE 1180/1190 Standard. Discrete Transform Accurate Test”. Enero 1998.

[ISO00] ISO/IEC. “Generic coding of moving pictures and associated audio information: Video”. ISO/IEC 13818-2. 2000.

[ISO01] ISO/IEC. “Coding of audio-visual objects - Part 5: Reference software”. ISO/IEC 14496-5. 2001.

[ISO04] ISO/IEC. “Generic coding of moving pictures and associated audio information: Conformance testing” ISO/IEC 13818-4. 2004

[ISO04b] ISO/IEC. “Coding of audio-visual objects - Part 2: Visual”. ISO/IEC 14496-2. 2004.

[ISO04c] ISO/IEC. “Coding of audio-visual objects -- Part 4: Conformance testing”. ISO/IEC 14496-4. 2004.

[ISO05] ISO/IEC. “Generic coding of moving pictures and associated audio information: Software simulation”. ISO/IEC 13818-5. 2005.

[ISO06] ISO/IEC. “Generic coding of moving pictures and associated audio information: Advanced Audio Coding”. ISO/IEC 13818-7. 2006.

[ISO07] ISO/IEC. “Generic coding of moving pictures and associated audio information: Systems”. ISO/IEC 13818-1. 2007.

[ISO09] ISO/IEC. “Coding of audio-visual objects - Part 3: Audio”. ISO/IEC 14496-3. 2009.

[ISO98] ISO/IEC. “Generic coding of moving pictures and associated audio information: Audio”. ISO/IEC 13818-3. 1998

[ITU02] ITU-R Rec. BT.709-5 “Parameter values for the HDTV standards for production and international programme exchange”. April 2002.

[ITU03] ITU-T Rec. H.264 “Advanced video coding for generic audiovisual services”. May 2003.

[ITU07] ITU-T Rec. H.264 “Advanced video coding for generic audiovisual services”. November 2007. Scalable Video Coding (SVC).

[ITU09] ITU-T Rec. H.264 “Advanced video coding for generic audiovisual services”. March 2009. Multi View Coding (MVC).

[ITU11] “Draft and Test Model editing” Editado por K. McCann, T. Wiegand, B. Bross, W. J. Han, J. R. Ohm, J. Ridge, S. Sekiguchi, G. J. Sullivan. January 2011.

[ITU95] ITU-R Rec. BT.601-5 “Studio encoding parameters of digital television for standard 4:3 and wide screen 16:9 aspect ratios”. October 1995.

[ITU98] ITU-T Rec. H.263 “Video Coding for Low Bit-Rate Communication”. February 2003.

[JPL+10] Eduardo Juárez, Fernando Pescador, Pedro J. Lobo, Angel Groba, César Sanz. "Distortion-Energy analysis of an OMAP-Based H.264/SVC Decoder". *6th International Mobile Multimedia Communications Conference MobiMedia 2010*. Lisboa, Portugal, 6-8 September 2010.

[JRGS02] Anurag Jain, Ratna Reddy, Jeremiah Golston, Jagadeesh Sankaran. "Programable Real time MPEG-2 Encoding". *Global Signal Processing Expo.* GSPx 2002. Libro de actas no disponible.

[JSVM] Joint Scalable Video Model JSVM. Disponible en el repositorio de Rheinisch-Westfälische Technische Hochschule (RWTH) Aachen.

[KA03] Nejat Kamaci, Yucel Altunbasak. "Performance Comparison Of The Emerging H.264 Video Coding Standard With The Existing Standards" *International Conference on Multimedia and Expo* (ICME), vol. 1, pp. I-345-348. 6-9 July 2003.

[KHCL07] Chih-Hung Kuo, Guan-Chih Huang, Li-Chuan Chang, Bin-Da Liu. "Source code flow optimization for H.264/AVC video decoder implementing on a low-cost embedded system platform", *IEEE Region 10 Conference* (TENCON), pp. 1-4. October 30 - November 2, 2007.

[KHH+03] Jin-Hau Kuo, Chia-Chiang Ho, Kan-Li Huang, Jim Shiu, Ja-Ling Wu "A Low-Cost Media-Processor Based Real-Time MPEG-4 Video Decoder". *IEEE Transactions on Consumer Electronics*, vol. 49, no. 4, pp. 1488-1497. November 2003.

[KJB+04] Hae-Yong Kang, Kyung-Ah Jeong, Jung-Yang Bae, Young-Su Lee, Seung-Ho Lee. "MPEG-4 AVC/H.264 decoder with scalable bus architecture and dual memory controller". *Proceedings of the IEEE International Symposium on Circuits and Systems ISCAS 2004*, vol. II, pp. 145–148. 26 May 2004.

[KOS00] T. Kamemaru, H. Ohira, H. Suzuki, K. Asano, M. Yoshimoto, T. Murakami. "Media Processor Core Architecture for Realtime, Bi-Directional MPEG4/H.26X Codec with 30 fr/s for CIF-Video". *Proccedings of the IEEE Custom Integrated Circuits Conference* (CICC), pp. 473-476. May 2000.

[KS05] Youngsoo Kim, Suleyman Sair. "Designing real-time H.264 decoders with dataflow architectures", *Third IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis* (CODES+ISSS), pp. 291-296. September 2005.

[KTM+06] Nobuhiro Kato, Kazuaki Takeuchi, Seiji Maeda, Mitsuru Shimbayashi, Ryuji Sakai, Hiroshi Nozue ,Jiro Amemiya. "Digital media applications on a CELL software platform", *IEEE International Conference on Consumer Electronics ICCE 2006*. Digest of Technical Papers, pp. 347-348. 7-11 Jan. 2006.

[LIBA52] A/52 stream decoder. Código en punto fijo de libre distribución que implementa el estándar ATSC A/52. Disponible en http://liba52.sourceforge.net/.

[LIBDVB] Librería de libre distribución que analiza y genera las tablas PAT y PMT que se encuentran en una trama de transporte MPEG-2. Disponible en http://www.videolan.org/developers/libdvbpsi.html.

[LJL+03] Peter List, Anthony Joch, Jani Lainema, Gisle Bjøntegaard, Marta Karczewicz. “Adaptive Deblocking Filter”. *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 637-644. July 2003

[LJP+11] Pedro J. Lobo, Eduardo Juárez, Fernando Pescador, Gonzalo Maturana, Manuel César Rodriguez. “A DVB-H receiver and gateway implementation on a FPGA- and DSP-based platform”. *IEEE Transactions on Consumer Electronics 2011*. Aceptada para su publicación en el volumen de Mayo de 2011.

[LLWL06] Tsu-Ming Liu, Ting-An Lin, Sheng-Zen Wang, Chen-Yi Lee. "A low-power dual-mode video decoder for mobile applications”. *IEEE Communications Magazine*, vol. 44, no. 8, pp. 119-126, August 2006.

[LMS04] Juyup Lee, Sungkun Moon, Wonyong Sung. "H.264 decoder optimization exploiting SIMD instructions”. *Proceedings of the 2004 IEEE Asia-Pacific Conference on Circuits and Systems*, vol. 2, pp. 1149-1152, 6-9 December 2004.

[LPK+06] Suh Ho Lee, Ji Hwan Park, Seon Wook Kim, Sung Jea Ko, Suki Kim, "Implementation of H.264/AVC decoder for mobile video applications”, *Asia and South Pacific Conference on Design Automation*, pp. 120-121, 24-27 January 2006.

[LPS+10] Pedro J. Lobo, Fernando Pescador, Ernesto Seisdedos, David Samper, César Sanz. "Implementation of a DSP based DVB-H gateway". *XXV Conference on Design of Circuits and Integrated Systems DCIS 2010*. Lanzarote, España, 17–19 November 2010.

[LXZ08] Zhengming Li, Qiuyan Xing, Xiaoyong Zhu. “H.264 video encoder implementation and optimization based on DM642 DSP” *IEEE International Conference on Networking, Sensing and Control*, pp. 891-894. 6-8 April 2008.

[MAD] MPEG-2 Audio Decoder. Código en punto fijo de libre distribución que implementa las capas I, II y III de la norma ISO/IEC 13818-3. Disponible en http://www.underbit.com/products/mad/.

[MBS+03] S. Moch, M. Berekovic, H.J. Stolberg, L. Friebe, M.B. Kulaczewskim A. Dehnhardt, P. Pirsch. “HiBRIC-Soc: a Multi-core arquitecture for Image and Video Applications”. *International Conference on Image Processing*. Proceedings, vol. 3. 14-17 September 2003.

[MMT+01] N. Minegishi, N. Motoyama, M. Takagi, F. Ogawa, K. Shibata, N. Goda, K. Akiyoshi, T. Kamemaru, K. Asano “A simple chip H.32X multimedia comunication processor with CIF 30f/s MPEG-4/H.26X Bi-directional codec”. *Proceedings of the 27th European Solid State Circuits Conference*. September 2001.

[MSW03] Detlev Marpe, Heiko Schwarz, Thomas Wiegand. “Context-based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard”. *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 620-636. July 2003.

[MYN+06] Adam Major, Ying Yi, Ioannis Nousias, Mark Milward, Sami Khawam, Tughrul Arslan. "H.264 Decoder Implementation on a Dynamically Reconfigurable Instruction Cell Based Architecture". *IEEE International SOC Conference*, pp. 49-52. 24-27 September 2006.

[OBL+04] Jörn Ostermann, Jan Bormans, Peter List, Detlev Marpe, Matthias Narroschke, Fernando Pereira, Thomas Stockhammer, and Thomas Wedi. "Video coding with H.264/AVC: tools, performance, and complexity". *IEEE Circuits and Systems Magazine*, vol. 4, no. 1, pp. 7-28, 2004.

[OSVC09] Código fuente en C que implementa un descodificador compatible con el anexo G de la norma H.264. Disponible en http://sourceforge.net/projects/opensvcdecoder/

[PBJ+11] Fernando Pescador, Mederic Blestel, Eduardo Juárez, Mickael Raulet, Matías J. Garrido. "H.264/SVC decoder performance comparison for DSP-Based consumer electronic applications" *14th IEEE International Symposium on Consumer Electronics ISCE 2011*. Singapore, 14-17 June 2011.

[PE02] Fernando Pereira, Touradj Ebrahimi. "The MPEG-4 Book". IMSC Press Multimedia Series. Ed. Prentice Hall PTR. 2002.

[PGA06] Fernando Pescador, Matías J. Garrido, Rafael Antoniello, Carlos Santos, César Sanz. "A DSP based multiformat video decoder for an IP set-top box". *EDERS2006*. Munich (Alemania). April 2006.

[PGS+06] Fernando Pescador, Matías J. Garrido, César Sanz, Eduardo Juárez, David Samper, Rafael Antoniello. "MPEG-4 SP/ASP decoder for a DSP-based Multi-Format IP Set-Top Box". *Annual Conference of the IEEE Industrial Electronics Society IECON 2006*, pp. 3397-3402. November 2006.

[PGS+07] Fernando Pescador, Matías J. Garrido, César Sanz, Eduardo Juárez, David Samper, Ángel Groba. "Real-time H.264 BP decoder based on a DM642 DSP". *IEEE International Conference on Signal Processing and Communications ICSPC 2007*. Dubai, Emiratos Árabes Unidos, pp. 1491-1494. 24-27 November 2007.

[PGS+07b] Fernando Pescador, Matías J. Garrido, César Sanz, Eduardo Juárez, M. César Rodríguez, David Samper. "A Real-Time H.264 MP Decoder Based on a DM642 DSP". *14th IEEE International Conference on Electronics, Circuits and Systems ICECS 2007*. Marrakech, Marruecos, pp. 1248-1251. 11-14 December 2007.

[PGS+07c] Fernando Pescador, Matías Garrido, César Sanz, Eduardo Juárez, Angel M. Groba, Francisco J. Sánchez. "An MPEG2 TS Parser for a DSP based Multi-format IP Set-top Box". *Conference on Design of Circuits and Integrated Systems DCIS 2007*. Sevilla 2007.

[PGS+07d] Fernando Pescador, Matías J. Garrido, César Sanz, Eduardo Juárez, David Samper, Rafael Antoniello. "Multiformat decoder for a DSP-based IP Set-Top Box". *SPIE International Symposium on Microtechnologies for the New Millennium. VLSI Circuits and Systems Conference*. Las Palmas de Gran Canaria. May 2007.

[PGS+08] Fernando Pescador, Matías J. Garrido, César Sanz, Eduardo Juárez, David Samper. "A DSP Based H.264 Decoder for a Multi-Format IP Set-Top Box". *International Conference on Consumer Electronics ICCE 2008,* no. 1, pp. 1-2. 9-13 January 2008.

[Pha05] D. Pham *et al.* "The Design and Implementation of a First-Generation CELL Processor", *IEEE International Solid-State Circuits Conference, ISSCC 2005*. Digest of Technical Papers, pp. 184-185, February, 2005.

[PJR+11] Fernando Pescador, Eduardo Juárez, Michael Raulet, César Sanz "A DSP Based H.264/SVC Decoder for a Multimedia Terminal" *IEEE Transactions on Consumer Electronics 2011*. Aceptada para su publicación en el volumen de Mayo de 2011.

[PJS+10] Fernando Pescador, Eduardo Juárez, David Samper, César Sanz, Mickael Raulet. "A Test Bench for Distortion-Energy Optimization of a DSP-Based H.264/SVC Decoder" *13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools DSD 2010*. Lille, Francia, pp. 123-129, 1-3 September 2010.

[PLS+11] Fernando Pescador, Pedro J. Lobo, Ernesto Seisdedos, Eduardo Juárez. Matías J. Garrido. "Real Time DVB-H Gateway Based on DSP". *International Conference on Consumer Electronics ICCE 2011*. Las Vegas, EEUU, pp. 749-750. 9-12 January 2011.

[PMG+09] Fernando Pescador, Gonzalo Maturana, Matías J. Garrido, Eduardo Juárez, César Sanz "An H.264 Video Decoder Based on a latest generation DSP". *IEEE Transactions on Consumer Electronics*, vol. 54, no. 1, pp. 205-212. February 2009.

[PMG+09b] Fernando Pescador, Gonzalo Maturana, Matías J. Garrido, Eduardo Juárez, César Sanz "An H.264 video decoder based on a DM6437 DSP". *International Conference on Consumer Electronics ICCE 2009,* pp. 263-264. January 2009.

[PMK+07] Anjaneya Prasad, Aditya Mittal, Mangesh Kumar, Prasad Sankaran, Ruturaj Chandpur "Optimization and Comparison of Computational Complexities of Standard Compliant Video Decoders on SIMD Processor", *IEEE International Conference on Acoustics, Speech and Signal Processing ICASSP 2007*, vol. 2, pp.II-93-II-96. 15-20 April 2007.

[PNX17] PNX17xx Series Data Book. Connected Media Processor. Disponible en http://www.nxp.com/#/pip/pip=[pip=PNX17XX_SER_N_1] |pp=[t=pip, i= PNX17XX_SER_N_1].

[PSG+06] Fernando Pescador, César Sanz, Matías J. Garrido, Carlos Santos, Rafael Antoniello. "A DSP based IP Set-Top Box for Home Entertainment". *IEEE Transactions on Consumer Electronics*, vol. 52, no. 1, pp. 254-262. February 2006.

[PSG+06b] Fernando Pescador, César Sanz, Matías J. Garrido, Carlos Santos, Rafael Antoniello, Javier Iglesias. "A DSP based IP Set-Top Box for Home Entertainment". *International Conference on Consumer Electronics ICCE 2006*, pp. 271-272. 7-10 January 2006.

[PSG+08] Fernando Pescador, César Sanz, Matías J. Garrido, Eduardo Juárez, David Samper. “A DSP Based H.264 Decoder for a Multi-Format IP Set-Top Box”. *IEEE Transactions on Consumer Electronics,* vol. 54, no. 1, pp. 145-153. February 2008.

[PSG+10] Fernando Pescador, David Samper, Matías J. Garrido, Eduardo Juárez, Mederic Blestel. "A DSP based SVC IP STB using Open SVC Decoder". *14th IEEE International Symposium on Consumer Electronics ISCE 2010*. Braunschweig, Alemania, 7-10 June 2010.

[PSJ+11] Fernando Pescador, David Samper, Eduardo Juárez, Mickael Raulet, César Sanz “A DSP Based H.264/SVC Decoder for a Multimedia Terminal” *IEEE International Conference on Consumer Electronics ICCE 2011*. Las Vegas, EEUU, pp. 123-129. 9-12 January 2011.

[QJSJ05] Xue Quan, Liu Jilin, Wang Shijie, Zhao Jiandong. "H.264/AVC baseline profile decoder optimization on independent platform”, *International Conference on Wireless Communications, Networking and Mobile Computing*. Proceedings, vol. 2, pp. 1253-1256. 23-26 September 2005.

[RH96] P.K. Rao, J.J. Hwang “Techniques and Standards for Image Video and Audio Coding”. Ed. Prentice Hall. 1996.

[RR04] Iain E. Richardson, Iain E. G. Richardson. “H.264 and MPEG-4 video compression”. Ed. Wiley. Ed. 2004.

[SAA] Philips Semiconductors. “SAA7120 Data Sheet”. Disponible en http://www.ortodoxism.ro/datasheets/philips/ SAA7120.pdf.

[San98] César Sanz. “Arquitectura VLSI para la estimación de movimiento en codificación de imágenes”. Tesis doctoral leída en el Departamento de Ingeniería Electrónica de la E.T.S.I. de Telecomunicación de la U.P.M. Marzo 1998.

[SAP] RFC 2974. Session Announcement Protocol. SAP. Disponible en http://www.ietf.org/ rfc/rfc2974.txt.

[SBPR01] H.-J. Stolberg, M. Berekovic, P. Pirsch, H. Runge. "Implementing the MPEG-4 advanced simple profile for streaming video applications*", IEEE International Conference on Multimedia and Expo ICME 2001*, pp. 230-233. 22-25 August 2001.

[SDP] RFC2327. Session Description Protocol. SDP. Disponible en http://www.ietf.org/rfc/ rfc2327.txt.

[SH98] Sundararajan Sriram, Ching-Yu Hung. “MPEG-2 Video Decoding on the TMS320C6X DSP Architecture”. *Thirty-Second Asilomar Conference on Signals, Systems & Computers*. Conference Record, vol. 2, pp. 1735-1739. 1-4 November 1998.

[Sie05] Peter Siebert. "Implementation of H.264/MPEG-4 AVC in low cost set top boxes”. *Ninth International Symposium on Consumer Electronics ISCE 2005*. Proceedings, pp. 310-314. 14-16 June 2005.

[SMP06] SMPTE 421M. "VC-1 Compressed Video Bitstream Format and Decoding Process". Abril 2006.

[SMP08] SMPTE 2019. “VC-3 Picture Compression and Data Stream Format”. Enero 2008

[SMSO03] S. Swanson, K. Michelson, A. Schwerin, M. Oskin. “WaveScalar”. *36th Annual International Symposium on Microarchitecture* (MICRO). December 2003.

[STB04] STB04xxx Digital Set-Top Box Integrated Controller. Disponible en https://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/AFCA142AA0DAEB9A852578250050E3B6/$file/STB04xxxFINAL.pdf

[STL04] Sullivan G.J.; Topiwala P.; Luthra A.; “The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions” *SPIE Conference on Applications of Digital Image Processing*, vol. 5558, pp. 454-462. August 2004.

[TI01] Texas Instruments. “TMS320C6000 DSP Peripherals Overview Reference Guide”. Enero 2001. Disponible en http://focus.ti.com/docs/prod/folders/print/tms320dm642.html.

[TI02] Texas Instruments. “TMS320C64x Image/Video Processing Library”. Abril 2002. Disponible en http://www.ti.com.

[TI03] Texas Instruments. “Reference Frameworks for eXpressDSP Software: RF5 an Extensive, High-Density System” (SPRA795A-April 2003). Disponible en http://focus.ti.com/lit/an/spra795a/spra795a.pdf.

[TI06] Texas Instruments. “Code Composer Studio Development Tools v3.3 Getting Started Guide”. SPRU509 Rev. H. October 2006. Disponible en http://focus.ti.com/lit/ug/spru509h/spru509h.pdf.

[TI06b] Texas Instruments. “TMS320DM6437 Digital Media Processor” (Rev. D). SPRS345D November 2006. Disponible en http://focus.ti.com/lit/ds/symlink/tms320dm6437.pdf.

[TI08] Texas Instruments. “TMS320DM6437 Digital Media Processor”. Junio 2008. Disponible en http://focus.ti.com/docs/prod/folders/print/tms320dm6437.html

[TI10] Texas Instruments. “TMS320C64x/C64x+ DSP CPU and Instruction Set. SPRU732J”. July 2010. Disponible en http://focus.ti.com/lit/ug/spru732j/spru732j.pdf.

[TI10b] Texas Instruments. “TMS320DM642 Video/Imaging Fixed-Point Digital Signal Processor” (Rev. N). SPRS200N. October 2010. Disponible en http://focus.ti.com/lit/ ds/symlink/tms320dm642.pdf.

[TI11] Texas Instruments. “TMS320C6000 Optimizing Compiler v 7.2. User’s Guide”. SPRU187S. March 2011. Disponible en http://focus.ti.com/lit/ug/spru187s/spru187s.pdf

[TI11b] Texas Instruments. “TMS320C6472 Fixed-Point Digital Signal Processor”. SPRS612F. February 2011. Disponible en http://focus.ti.com/lit/ds/symlink/tms320c6472.pdf

[TIVR] TIVR technical staff, H.264 Baseline Profile (BP) Video Decoder, TIVR Communications Private Ltd.

[TRIM] TM1300 IREF Reference Manual”. Disponible en http://oops.tepkom.ru/padla/trimedia/Tm1300_IREF_Man_V2-0-3.PDF.

[TVP51] TVP5146M2 10-bit High Quality Single-Chip Digital Video Decoder That Digitizes And Decodes NTSC/PAL/SECAM. Disponible en http://focus.ti.com/docs/prod/folders/print/tvp5146m2.html.

[TWT+05] Yi-Shin Tung, Sung-Wen Wang, Chien-Wu Tsai, Ya-Ting Yang, Ja-Ling Wu, "DSP-based multi-format video decoding engine for media adapter applications", *IEEE Transactions on Consumer Electronics*, vol. 51, no. 1, pp. 273- 280. February 2005.

[VLC] VideoLan player. Disponible en http://www.videolan.org.

[VN07] Nicholas Vun, T. N. A. Nguyen. "Development of H.264 Encoder for a DSP Based Embedded System". *IEEE International Symposium on Consumer Electronics ISCE 2007*, pp. 1-4. 20-23 June 2007.

[WC06] Zhe Wei, Canhui Cai. "Realization and optimization of DSP based H.264 encoder". *IEEE International Symposium on Circuits and Systems ISCAS 2006*, pp. 1921-1924. 2006.

[WHL06] Hong-Jun Wang, Yong-Jian Huang, Hua Li. "H.264/AVC Video Encoder Implementation Based on TI TMS320DM642". *International Conference on Intelligent Information Hiding and Multimedia Signal Processing IIHMSP 2006*, pp. 503-506. December 2006.

[WHL07] Hong-Jun Wang, Yan-yan Hou, Hua Li. "H.264/AVC Video Encoder Algorithm Optimization Based on TI TMS320DM642". *Third International Conference on Intelligent Information Hiding and Multimedia Signal Processing IIHMSP 2006*, vol. 1, pp. 529-532. 26-28 November 2007.

[WK07] Woon-Seng Gan, Sen M. Kuo. "Embedded Signal Processing with the Micro Signal Architecture". Ed. John Wiley & Sons. 2007. Disponible en http://ebooks.ebookmall.com/ebook/231945-ebook.htm.

[WKBM06] Imen Werda, Faouzi Kossentini, Mohamed-Ali Ben Ayed, Nouri Massmoudi. "Analysis and Optimization of UB Video's H.264 Baseline Encoder Implementation on Texas Instruments TMS320DM642 DSP". *IEEE International Conference on Image Processing*, pp. 3277-3280. 8-11 October 2006.

[WSBL03] Thomas Wiegand, Gary J. Sullivan, Gisle Bjøntegaard, Ajay Luthra. "Overview of the H.264/AVC video coding standard" *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560-576. July 2003.

[WSJ+03] Thomas Wiegand, Heiko Schwarz, Anthony Joch, Faouzi Kossentini, Gary J. Sullivan. "Rate-constrained coder control and comparison of video coding standards". *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 688-703. July 2003.

[X264] Código fuente en C que implementa la norma H.264. Disponible en http://www.x264.nl/.

[XMUX] Elecard XMuxer Pro. Generador de Trama de Transporte. Disponible en http://www.elecard.com/en/products/professional/editing/xmuxer-pro.html.

[XVID] Código fuente en C que implementa una versión parcial de la norma MPEG-4. Disponible en http://www.xvid.org/.

[YGLZ06] Zhigang Yang, Wen Gao, Yan Liu, Debin Zhao. "Deeply pipelined DSP solution to deblocking filter for H.264/AVC". *IEEE Transactions on Consumer Electronics*, vol. 52, no. 4, pp. 1267-1274. November 2006.

[ZWFS07] Li Zhuo, Qiang Wang, David Dagan Feng, Lansun Shen. “Optimization and Implementation of H.264 Encoder on DSP Platform” *IEEE International Conference on Multimedia and Expo*, pp. 232-235. 2-5 July 2007.

11 ACRÓNIMOS

Acrónimo	Inglés	Castellano
AAC	*Advanced Audio Coding*	Codificación de Audio Avanzada
AC3	*Audio Coding 3* o Dolby Surround.	Codificación de Audio 3
ADSL	*Asymmetric Digital Subscriber Line*	Línea de Suscripción Digital Asimétrica
AMBA	*Advanced Microcontroller Bus Architecture*	Arquitectura Avanzada para el Bus de Microcontroladores
API	*Application Program Interface*	Interfaz para la aplicación
ASIC	*Application Specific Integrated Circuit*	Circuito Integrado de Aplicación Específica
ASO	*Arbitrary Slice Order*	Forma de organización arbitraria de los macrobloques de una imagen
ASP	*Advanced Simple Profile*	Perfil Simple Avanzado
ATSC	*Advanced Television Systems Committee*	Comité de Sistemas Avanzados de Televisión
AU	*Access Unit*	Unidad de Acceso
AVC	*Advanced Video Coding*	Codificación de Vídeo Avanzado
BIFS	*BInary Format for Scenes*	Formato Binario de Descripción de Escenas
BIOS	*Basic Input Output System*	Sistema Básico de Entrada/Salida
BP	*Baseline Profile*	Perfil Base
BS	Boundary Strength	Fuerza de filtrado del borde

Acrónimo	Inglés	Castellano
CABAC	*Context-Adaptive Binary Arithmetic Coding*	Codificación Binaria Aritmética Adaptada al Contexto
CAN	*Controller Area Network*	Controlador de Red Local
CAVLC	*Context-Adaptive Variable-Length Coding*	Codificación de Longitud Variable Adaptada al Contexto
CBP	*Coded Block Pattern*	Modo de codificación del bloque
CCS	*Code Composer Studio*	
CCIR	*Comité Consultatif International des Radiocommunications*	Comité Consultivo Internacional de Radiocomunicación
CVBS	*Color, Video, Blank and Sync*	Color, Video y Sincronización
DAC	*Digital Analog Converter*	Convertidor Digital Analógico
DCT	*Discrete Cosine Transform*	Transformada Discreta del Coseno
DMA	*Direct Memory Access*	Acceso Directo a Memoria
DSM-CC	*Digital Storage Media Command and Control*	Extensión para la Gestión de Contenidos
DSP	*Digital Signal Processor*	Procesador Digital de Señal
DTS	*Decoding Time Stamp*	Instante de descodificación
DVD	*Digital Video Disk*	Disco de Vídeo Digital
EDMA	*Enhanced* DMA	Controlador de DMA mejorado
EMAC	*Ethernet Medium Access Controller*	Controlador de Acceso a Ethernet
EMIF	*External Memory InterFace*	Interfaz con Memoria Externa
ES	*Elementary Stream*	Trama Elemental
FAAD	*Freeware Advanced Audio Decoder*	Descodificador de Audio Avanzado de libre acceso
FIFO	*First In, First Out*	Primero en entrar, primero en salir
FIR	*Finite Impulse Response*	Respuesta Finita al Impulso
FPGA	*Field-programmable Gate Array*	
Gcc	*GNU Collection Compiler*	
GMC	*Global Motion Compensation*	Compensación Global de Movimiento
GOP	*Group of Pictures*	Grupo de Imágenes
GOV	*Group Of VOP*	Grupo de VOP
GPP	*General Purpose Processor*	Procesador de Propósito General
HEVC	*High Efficient Video Coding*	Codificación de Alta Eficiencia
I2C	*Inter-Integrated Circuit*	Bus de Comunicaciones entre Circuitos
ICT	*Integer Cosine Transform*	Transformada Entera del Coseno

Acrónimo	Inglés	Castellano
IDCT	*Inverse Discrete Cosine Transform*	Transformada Discreta del Coseno Inversa
IDMA	*Internal DMA*	DMA Interno
IICT	*Inverse Integer Cosine Transform*	Transformada Entera del Coseno Inversa
IPMP	*Intellectual Property Management and Protection*	Gestión y protección de la propiedad intelectual
IPTV	*IP Television*	Televisión por Internet
ISO	*International Organization for Standardization*	Organización Internacional de Estandarización
JTAG	*Joint Test Action Group*	
LPS	*Less Probable Symbol*	Símbolo Menos Probable
MAD	MPEG *Audio Decoder*	Descodificador de Audio MPEG
MB	*MacroBlock*	Macrobloque
MBAFF	*Macroblock-Adaptive Frame/Field.*	Codificación Adaptativa Cuadro/Campo para parejas de macrobloques en formato H.264
MBAFF	*MacroBlock Adaptative Frame/Field*	Decisión Campo/Cuadro Adaptativa a nivel de MB
MBE	*Macroblock Engine*	Motor de Gestión de macrobloques
McASP	*Multichannel Asynchronous Serial Port*	Puerto Serie Asíncrono Multicanal
McBSP	*Multichannel Buffered Serial Port*	Puerto Serie Multicanal con capacidad de almacenamiento
MFC	*Memory Controller*	Controlador de Memoria
MIPS	*Million of Instructions per Second*	Millones de Instrucciones por Segundo
MMX	MultiMedia eXtension	Extensiones Multimedia
MP	*Main Profile*	Perfil Principal
MP@ML	*Main Profile at Main Level*	Perfil Principal y Nivel Principal
MPEG	*Motion Picture Expert Group*	Grupo de Expertos en Imágenes en Movimiento
MPS	*Most Probable Symbol*	Símbolo Más Probable
MSA	*Micro Signal Architecture*	Arquitectura “Micro Signal”
MSG	*Multiple Slice Groups*	Grupos de Rebanadas
MV	*Motion Vector*	Vector de Movimiento
MVC	*Multiview Video Coding*	Codificación Multivista
NAL	*Network Abstraction Layer*	Capa de Abstracción de Red
NTSC	*National Television System Committee*	

Acrónimo	Inglés	Castellano
OSD	*On Screen Display*	Información en pantalla
PAFF	*Picture-Adaptive Frame/Field*	Codificación Adaptativa Cuadro/Campo para imágenes en formato H.264
PAL	*Phase Alternating Line*	
PAT	*Program Allocation Table*	Tabla de Información de Programas
PCI	*Peripheral Component Interconnect*	Interconexión de Periféricos
PCM	*Pulse-Code Modulation*	Modulación de Pulsos Codificados
PCR	*Program Clock Reference*	Referencia de Reloj para un Programa
PES	*Packetized Elementary Streams*	Paquetes de las Tramas Elementales
PID	*Program IDentifier*	Identificador de Programa
PIXC	*Pixel Compositor*	Sistema de Procesado de Vídeo para realizar OSD
PMT	*Program Map Table*	Tabla de Información de un Programa
PPE	*Power Processor Element*	Elemento de Procesado
PPS	*Picture Parameter Set*	Parámetros Asociados a la Imagen
PSI	*Program Specific Information*	Información Específica de Programa
PSNR	*Peak Signal to Noise Ratio*	Relación Señal/Ruido de Pico
PTS	*Presentation Time Stamp*	Instante de Presentación
QDMA	*Quick* DMA	Acceso Rápido al Controlador de DMA
QP	*Quantizer Parameter*	Factor de Cuantificación
RBSP	*Raw Byte Sequence Payload*	Zona de Datos Válidos de una Unidad NAL
RF5	*Reference Framework 5*	Marco de Referencia para el desarrollo de aplicaciones número 5
RGB	*Red, Green and Blue*	Rojo, Verde y Azul
RISC	*Reduced Instruction Set Computer*	Procesador con Repertorio de Instrucciones Reducido
RLC	*Run Length Coding*	Codificación "*Run Length*"
RTI	*Real Time Interface*	Interfaz en Tiempo Real
RTOS	*Real Time Operating System*	Sistema Operativo en Tiempo Real

Acrónimo	Inglés	Castellano
RTP	*Real Time Protocol*	Protocolo de Transmisión en Tiempo Real
RVC	*Reconfigurable Video Coding*	Codificación de Vídeo Reconfigurable
RVLC	*Reversible VLC*	Códigos VLC Reversibles.
SAD	*Sum of Absolute Differences*	Suma de Diferencias Absolutas
SAP	*Session Announcement Protocol*	Protocolo de anuncio de sesión
SCOM	*Synchronous COMmunications*	Comunicaciones Síncronas
SDP	*Session Description Protocol*	Protocolo de Descripción de Sesión
SDRAM	*Synchronous Dynamic Random Access Memory*	Memoria RAM Dinámica y Síncrona
SIMD	*Single Instruction Multiple Data*	Única Instrucción con Múltiples Datos
SMPTE	*Society of Motion Picture and Television Engineers*	Sociedad para Imágenes Móviles e Ingeniería de Televisión
SoC	*System on Chip*	Sistema en un Chip
SP	*Simple Profile*	Perfil Simple
SPDIF	*Sony/Philips Digital Interconnect Format*	Formato de Interconexión Digital
SPE	*Synergistic Processor Element*	Procesador de Control
SPORT	*Serial PORT*	Puerto Serie Síncrono
SPS	*Sequence Parameter Set*	Parámetros Asociados a la Secuencia
STB-IP	*Set Top Box IP*	Descodificador de TV Digital con Entrada IP
SVC	*Scalable Video Coding*	Codificación de Vídeo Escalable
S-Video	*Separate Video*	
TS	*Transport Stream MPEG-2*	Trama de Transporte MPEG-2
TSP	*Transport Stream Packets MPEG-2*	Paquetes de las Tramas de Transporte MPEG-2
UIT/ITU	*International Telecomunications Union*	Unión Internacional de Telecomunicaciones
VCEG	*Video Coding Expert Group*	Grupo de Expertos en Codificación de Vídeo
VCL	*Video Coding Layer*	Capa de Codificación de Vídeo
VCO	*Voltage Controlled Oscillator*	Oscilador Controlado por Tensión
VLC	*Variable Length Coding*	Codificación de Longitud Variable
VLD	*Variable Length Decoding*	Descodificación de Longitud Variable

Acrónimo	Inglés	Castellano
VLIW	*Very Long Instruction Word*	Procesador con Instrucciones Complejas
VO	*Visual Object*	Objeto Visual
VoD	*Video on Demand*	Vídeo bajo Demanda
VOL	*Visual Object Layer*	Capa de Objeto Visual
VOP	*Video Object Plane*	Objeto de Vídeo
VPBE	*Video Processing Back End*	Puerto de Captura de Vídeo
VPFE	*Video Processing Front End*	Puerto de Presentación de Vídeo
VS	*Visual Sequence*	Secuencia de Objeto Visual