UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Instituto Universitario de Microelectrónica Aplicada
Sistemas de información y Comunicaciones

Máster en
Tecnologías de Telecomunicación

Trabajo Fin de Máster

SÍNTESIS DE ALTO NIVEL DE UN DECODIFICADOR H.264/AVC SOBRE PLATAFORMA FPGA

Autor: Romén Neris Tomé
Tutores: Antonio Núñez Ordóñez
Pedro Pérez Carballo
Fecha: Septiembre 2012

IUMA
t +34 928 451 086 | iuma@iuma.ulpgc.es
f +34 928 451 083 | www.iuma.ulpgc.es
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria

Versión: 2.0 – 14 junio 2011

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Instituto Universitario de Microelectrónica Aplicada
Sistemas de información y Comunicaciones

Máster en
Tecnologías de Telecomunicación

Trabajo Fin de Máster

SÍNTESIS DE ALTO NIVEL DE UN DECODIFICADOR H.264/AVC SOBRE PLATAFORMA FPGA

HOJA DE FIRMAS

Alumno:	Romén Neris Tomé	Fdo.:
Tutor:	Antonio Núñez Ordóñez	Fdo.:
Tutor:	Pedro Pérez Carballo	Fdo.:
Fecha:	Septiembre 2012	

IUMA
t +34 928 451 086
f +34 928 451 083
iuma@iuma.ulpgc.es
www.iuma.ulpgc.es
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria

Versión: 2.0 – 14 junio 2011

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Instituto Universitario de Microelectrónica Aplicada
Sistemas de información y Comunicaciones

Máster en
Tecnologías de Telecomunicación

Trabajo Fin de Máster

SÍNTESIS DE ALTO NIVEL DE UN DECODIFICADOR H.264/AVC SOBRE PLATAFORMA FPGA

HOJA DE EVALUACIÓN

Calificación: .....

Presidente: Fdo.:

Secretario: Fdo.:

Vocal: Fdo.:

Fecha: Septiembre 2012

IUMA
t +34 928 451 086
f +34 928 451 083
iuma@iuma.ulpgc.es
www.iuma.ulpgc.es
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria

Versión: 2.0 – 14 junio 2011

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

En un mercado de alta competitividad como el actual, que intenta cubrir el amplio abanico de necesidades que se demandan, aparece la necesidad de proporcionar productos más funcionales, más económicos y con un consumo de potencia cada vez menor. Todo esto repercute en un aumento de la complejidad de los diseños, así como en la necesidad de reducir los plazos de producción. Es por esto que ha de mejorarse cada vez más la productividad del diseñador mediante la automatización de las metodologías de diseño electrónico.

La metodología tradicional de diseño de sistemas integrados, o SoCs, no es adecuada para sistemas complejos, ya que el aumento de la complejidad genera problemas con repercusiones muy importantes. La velocidad de los avances tecnológicos reduce cada vez más el tiempo de vida de los diseños. Además, el sistema debe tener una relación coste/prestaciones optimizada y un alto grado de fiabilidad.

La finalidad global de este trabajo será la aplicación de técnicas de síntesis a alto nivel en una metodología apropiada en el dominio de aplicaciones de vídeo, particularmente en el diseño de un decodificador H.264/AVC.

Para ello se hará uso de flujos de diseño complejos en el dominio hardware, que permitirán la síntesis y caracterización del sistema sobre una plataforma FPGA, con la finalidad de obtener un sistema completamente caracterizado que permita la realización de comparativas y extracción de conclusiones a raíz de los resultados obtenidos.

Con dichos resultados, se realizarán comparativas que reflejen las variaciones de parámetros importantes en todo diseño de estas características, tales como consumo de recursos y frecuencia de funcionamiento.

Este trabajo surge dentro del marco de investigación del IUMA, “ARTEMI+: Arquitecturas para terminales multimedia portátiles, multiestándar y multired”.

ABSTRACT

On a market of high competitiveness like the current one, which tries to cover the wide range of needs that are demanded, it appears the need to provide more functional, cheaper and with less energy consumption products. This implies an increase of the designs complexity, and also the need to reduce the production period. This is the reason why it is necessary to improve the productivity of the designer by means of the electronic design methodologies automation.

The traditional design methodology of integrated systems, or SoCs, is not adapted for complex systems, since the increase of the complexity generates problems with very important repercussions. The speed of the technological advances reduces increasingly the designs life. In addition, the system must have a relation cost / features optimized and a high degree of reliability.

The global purpose of this work will be the application of design techniques of high level synthesis in a methodology adapted in the video applications domain, particularly in the H.264/AVC decoder design.

It will use complex design flows in the hardware domain, which it will allow the synthesis and characterization of the system on a FPGA platform, with the purpose of obtaining a completely characterized system that allows the accomplishment of comparatives and extraction of conclusions after the obtained results.

With the obtained results, there will be made comparatives that they reflect the variations of important parameters in any design of these characteristics, such as consumption of resources and frequency.

This TFM is born from IUMA's research Project: "ARTEMI+: Arquitecturas para terminales multimedia portátiles, multiestándar y multired" ("ARTEMI+: Architectures for portable multimedia terminals, multistandard and multinetwork").

CAPÍTULO 1: Introducción

1.1. Antecedentes

A día de hoy, los sistemas audiovisuales tienen una gran importancia en la vida cotidiana de nuestra sociedad. Es habitual el uso de reproductores de audio y vídeo de alta calidad, tales como cámaras digitales, DVDs portátiles, telefonía móvil con vídeo-llamadas, televisión digital, etc. Es en este contexto donde es necesario disponer de estándares de codificación que buscan la optimización de los recursos disponibles para dicha tarea [1].

En estos sistemas es de vital importancia el procesamiento de las señales de vídeo, que han de tener en cuenta parámetros que limitan los canales de difusión, tales como el ancho de banda, el consumo de potencia, etc. Todo ello ha generado enormes esfuerzos de investigación en técnicas avanzadas de codificación de vídeo.

El presente Trabajo Fin de Máster (TFM) se engloba en este campo de investigación, desarrollándose en el ámbito de la decodificación de vídeo.

Existen multitud de aplicaciones que han de ser ejecutadas bajo restricciones temporales. Los sistemas controlados por estas aplicaciones se conocen como Sistemas en Tiempo Real (STR o RTS en inglés). El parámetro de diseño condicionante suele ser el tiempo máximo de ejecución de un algoritmo que permita que los tiempos de respuesta del sistema sean lo más bajos posibles y siempre por debajo de un límite impuesto por el entorno de la aplicación.

Para cumplir con las restricciones temporales impuestas por el entorno es posible la utilización de aceleradores *hardware* que, adaptados al tipo del problema a procesar, disminuye los tiempos de cómputo necesarios, aliviando las restricciones anteriormente indicadas. Este enfoque implica la realización de una partición del código de la aplicación *software* a ejecutar, separando en diferentes módulos su funcionalidad, y diseñar aceleradores *hardware* a medida para aquellos módulos críticos, con un mayor coste computacional [2].

Las FPGAs (*Field Programmable Gate Array*) son dispositivos adecuados para realizar los bloques aceleradores indicados ya que permiten aprovechar el paralelismo intrínseco del hardware. Por su facilidad a la hora de reprogramar y su bajo coste para un bajo número de unidades, serán el principal recurso hardware a utilizar [3].

1.1.1. Servicios multimedia

Dada la creciente demanda de nuevos servicios multimedia y los cada vez más exigentes requisitos de consumo de potencia y ejecución en tiempo real de los dispositivos empotrados, se está realizando un significativo esfuerzo en el desarrollo de sistemas multimedia de alto rendimiento y bajo coste [4].

Un sistema en tiempo real (STR) es aquel sistema digital que interactúa activamente con su entorno con una dinámica conocida entre sus entradas, salidas y restricciones temporales, para darle un correcto funcionamiento de acuerdo con los conceptos de predictibilidad, estabilidad, controlabilidad y alcanzabilidad [5].

El correcto funcionamiento de estos sistemas depende del tiempo utilizado en producir dicho resultado. De esta forma, atendiendo al entorno en el que se desarrolle la aplicación, existirá un tiempo límite que el sistema deberá cumplir para que sea exitoso, considerándose todos aquellos resultados obtenidos después del límite como fallos del sistema aún cuando estos sean los valores esperados. Se puede hablar de dos tipos de STR: *Hard* RT y *Soft* RT. Para el primero de los casos la restricción es completa, con tiempos de respuesta estrictos. Para el segundo, los tiempos de procesamiento pueden estar comprendidos en un determinado margen pero no son críticos cuando la salida sigue una determinada cadencia, aunque con cierta latencia inicial [6]. Un ejemplo de un sistema *Soft* RT es un sistema de transmisión de vídeo, incluyendo su codificación, transmisión y decodificación.

Los decodificadores H.264/AVC están disponibles en muchas soluciones y su esquema de compresión se está haciendo muy popular. El mayor mercado para ICs decodificadores H.264/AVC continúa siendo el que abarca los *Set Top Boxes.* Así mismo, más y más DTVs usan este tipo de decodificadores en lugar de sólo emplear el decodificador de MPEG-2. Como resultado, se prevé que los ICs decodificadores de H.264/AVC tengan un crecimiento anual que aumente hasta en un 20% a partir de 2013 [7].

1.1.2. Estándar H.264/AVC

Dado que lo que se pretende con la realización de este trabajo es la determinación del impacto de la utilización de las técnicas de diseño basadas en síntesis de alto nivel, en la utilización de recursos de la FPGA para un decodificador H.264/AVC diseñado directamente desde una descripción algorítmica, se considera necesario realizar una breve introducción a sus características generales. En el capítulo 2, se profundizará en la descripción de los principales bloques funcionales.

La investigación de los estándares de reproducción de vídeo nace a partir de la década de los 80, con la creación del estándar H.120 [8], precursor del H.261 [9], considerado el primer estándar de codificación de vídeo digital. Los posteriores estándares de codificación (MPEG-1 Parte 2 [10], H.262/MPEG-2 Parte 2 [11], H.263 [12], MPEG-4 Parte 2 [13], y H.264/AVC [14]) están basados en el diseño de H.261. Como ya se especificó, en la realización de este trabajo se hará uso del estándar H.264/AVC [14, 15].

El estándar H.264/AVC es un estándar que se encarga de definir un CODEC (COdificador – DECodificador) de vídeo de alta compresión, desarrollado conjuntamente por el ITU-T Video Coding Experts Group (VCEG) y el ISO/IEC Moving Picture Experts Group (MPEG) [16]. Con la creación de este proyecto se pretende diseñar un estándar capaz de proporcionar mejoras en la calidad de imagen, con una tasa binaria sensiblemente inferior a los estándares previos [1, 17].

En la figura 1 se puede ver como para una tasa de transferencia específica, el nivel de calidad y resolución obtenido con H.264/AVC va a ser mayor que para MPEG-2. Por ello, este estándar se ha propuesto para reemplazar a algunos de los formatos existentes, como MPEG-2, el estándar para vídeo en DVD, o DV, el estándar empleado por muchas cámaras de vídeo de gran consumo [18].

H.264/AVC cuenta con el hecho de que distintos tipos de dispositivos decodificadores pueden tener prestaciones o requisitos individuales. Por ejemplo, comparados con ordenadores de sobremesa, los dispositivos móviles pueden tener menos memoria interna, resolución de pantalla más bajas, menos capacidad de cálculo, etc. Por ello, se puede condicionar los requisitos de reproducción, seleccionando las mismas funciones de flujo de bits del estándar H.264/AVC utilizadas al crear el contenido [18].

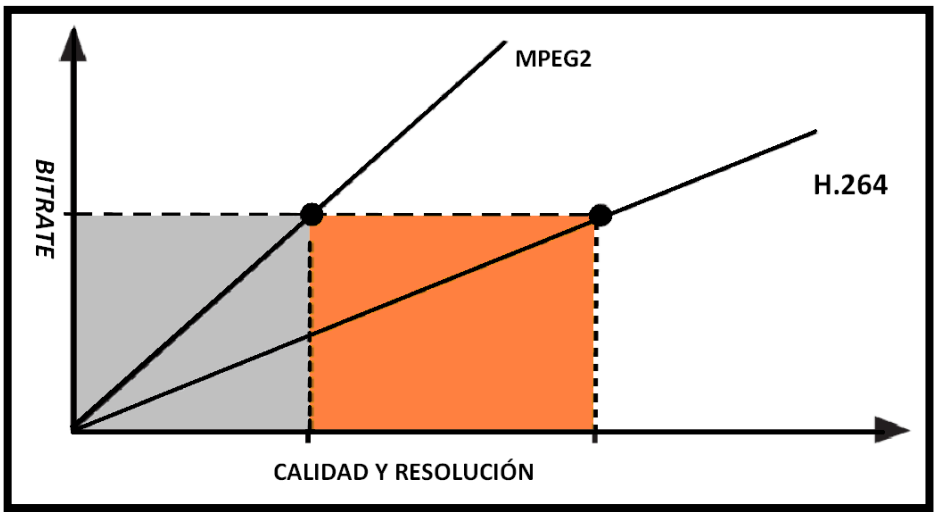


Figura 1. Comparativa de calidad y resolución entre H.264 y MPEG2. Adaptado de [15]

La codificación H.264/AVC disminuye la cantidad de información que se necesita para la transmisión de vídeo, ya que hace un uso eficiente de la redundancia de la información visual que se va codificando, tanto temporal como espacialmente. En el sentido temporal, el codificador procesa cada cuadro haciendo una subdivisión de la imagen contenida en una cuadrícula de bloques. Seguidamente, busca y compara las texturas de los cuadros anteriores y siguientes, una técnica que es comúnmente conocida como estimación de movimiento. Por cada coincidencia suficiente que detecta el codificador para procesar posteriormente la textura de cada bloque, un decodificador sólo necesitará un vector que sea el que apunte a la textura de referencia coincidente y una breve información para hacer la corrección de cualquier diferencia de textura. En el sentido espacial, cuando la estimación de movimiento no proporcione coincidencias suficientes, el codificador puede utilizar la textura de los bloques próximos dentro del mismo cuadro para hacer la predicción de la textura de cada bloque, almacenando sólo la diferencia entre la predicción y la textura real [18].

A pesar de que este método es más eficiente que almacenar la textura completa directamente, tiene un mayor coste que el método de estimación de movimiento. Los codificadores H.264/AVC actúan como compresores "*lossy*" (con pérdida de información). Su objetivo no es reproducir con exactitud cada cuadro original, sino elegir el método óptimo que reduzca la tasa de transferencia de datos, preservando al mismo tiempo la calidad visual en la medida de lo posible. Realizando los ajustes adecuados, las diferencias pueden llegar a ser imperceptibles, inclusive en niveles de compresión próximos a 100:1 con respecto a vídeo no procesado [18]. La figura 2 muestra una comparativa de la calidad de vídeo con respecto a la tasa de bits entre los estándares H.264/AVC, MPEG – 4 y JPEG.

Sin embargo, el diseño de la arquitectura de tanto el codificador como del decodificador H.264/AVC es complejo. Además de los altos requisitos de cómputo y de acceso a memoria, el camino de codificación es largo, incluyendo predicción, reconstrucción y decodificación de la entropía [19]. Comparándola con otros estándares, la complejidad *hardware* se incrementa hasta el doble en el caso del decodificador.

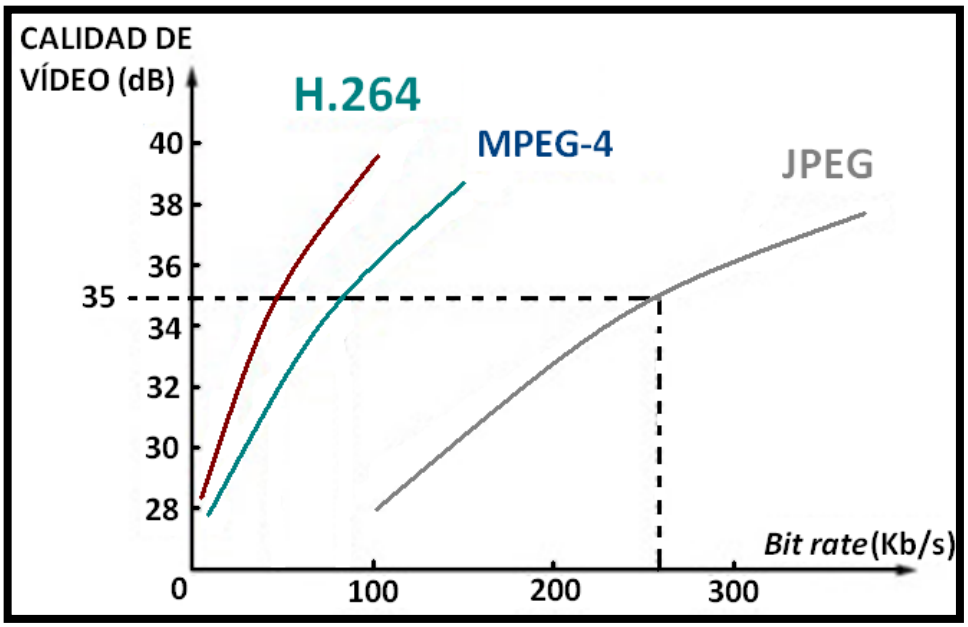


Figura 2. Comparativa de calidad de video entre H.264/AVC y MPEG4 y JPEG

1.1.3. System on Chip (SoC)

Los avances producidos recientemente en el entorno de la tecnología de procesos y la disponibilidad de nuevas herramientas de diseño están aumentando el campo de aplicación de los sistemas integrados, que se definen como un conjunto de chips en una placa o como un conjunto de IPs en un circuito integrado. Esta tendencia viene acompañada de la adopción de diseños basados en plataformas, lo que facilita el diseño y la verificación de sistemas complejos *SoC* con una amplia reutilización del *hardware* y *software* IP. Otro aspecto importante de la evolución de los sistemas integrados es la tendencia a interconectar nodos integrados aplicándose tecnologías de redes especializadas, conocidas frecuentemente como *Network On Chip – NoC* [4].

Los *SoC* se utilizan en el control de buena parte de aplicaciones, como en los dispositivos electrónicos de consumo (videoconsolas, reproductores de audio/vídeo…), en la automoción (control de airbag, climatizador…), en la industria (control de motores, robótica…), en las comunicaciones (teléfonos móviles, modem...), etc. En el diseño *SoC* se dan cita dos elementos conceptuales:

- Por una parte está la arquitectura *hardware* construida en torno a un sistema basado en microprocesador.
- La arquitectura *software* que incluye los sistemas operativos, los lenguajes de programación, compiladores, herramientas de modelado, simulación y evaluación.
- La integración de la arquitectura *hardware* y *software* proporciona la arquitectura global del *SoC* [20].

El concepto de *SoC* nace a mediados de los años 90, cuando la tecnología ASIC evolucionó con el propósito de integrar en un único circuito integrado sistemas completos. Un procesador *SoC* es un

circuito integrado complejo que combina los principales elementos o subsistemas funcionales de un producto completo.

En la figura 3 podemos ver el diagrama de bloques de un dispositivo *SoC*.

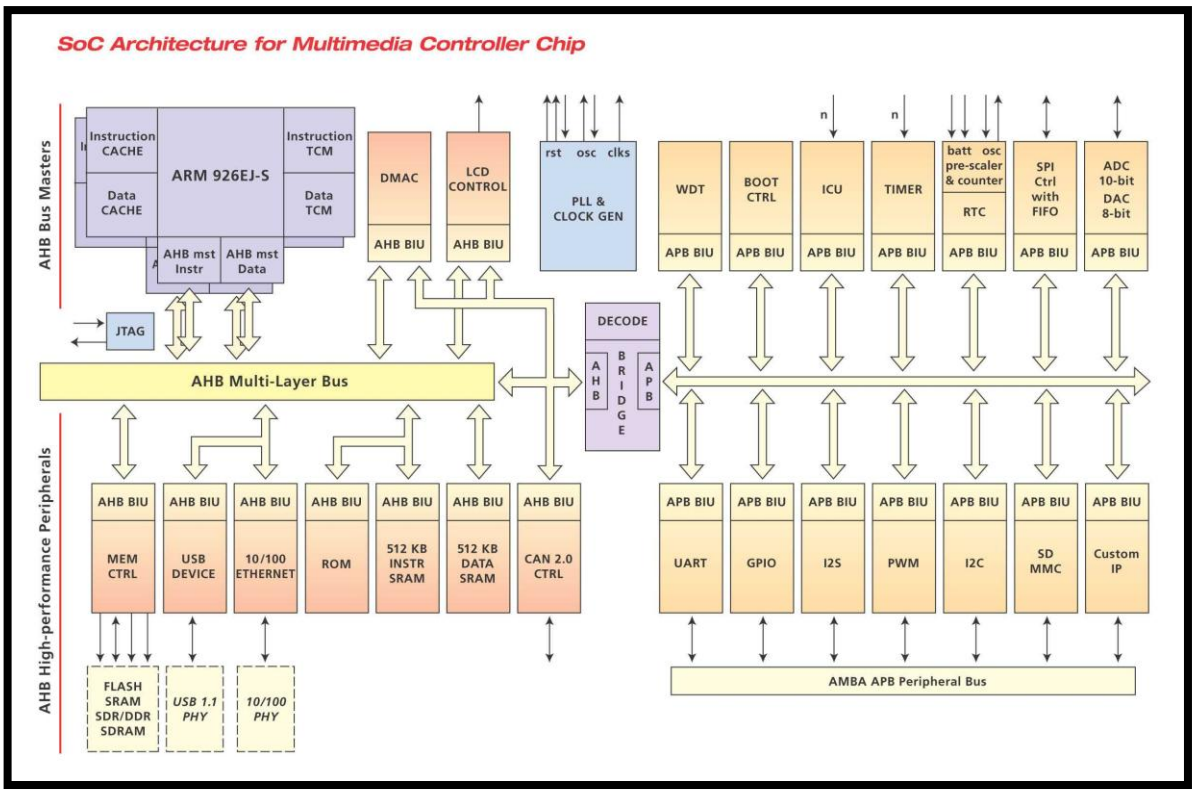


Figura 3. Arquitectura de un *System on Chip – SoC*. Adaptado de [21]

Los diseños *SoC* más exigentes incluyen al menos un procesador programable y a menudo una combinación de un procesador de control *RISC* y un *DSP* para el procesado de señales digitales. También incluyen estructuras de comunicaciones sobre chips: bus o buses de procesadores y de periféricos y, en ocasiones, un bus de sistema de alta velocidad. Para los procesadores *SoC* es muy importante que el chip tenga unidades de memoria jerarquizadas y enlaces con memorias externas.

La tecnología *SoC* puede incorporar sensores y actuadores basados en sistemas microelectro-mecánicos, o procesamiento de elementos y productos químicos (laboratorio en un chip).

El diseño y uso de *SoC* implica, además del *hardware*, diseño e ingeniería a nivel de sistemas, compromisos *hardware-software* y particiones, así como arquitectura, diseño e implementación de *software* [22].

1.2. Objetivos

La finalidad global de este Trabajo Fin de Máster será la síntesis de alto nivel de un decodificador de vídeo H.264/AVC sobre una plataforma FPGA.

Para ello, se definirá una metodología de trabajo partiendo de un diseño existente (obtenido en el proyecto ARTEMI+) del decodificador descrito en lenguaje *SystemC* y realizando los pasos necesarios para llegar a la correcta síntesis del decodificador sobre una plataforma FPGA. Se presentarán los resultados obtenidos más relevantes, formando todo ello parte del proyecto PCCMUTE llevado a cabo por la División de Sistemas Industriales y CAD – SICAD.

Más detalladamente, se explican a continuación los principales objetivos a conseguir:

- Como primer paso se realiza un estudio previo del estándar AVC así como de la composición y funcionamiento del decodificador.
- Entrando más en detalle, se analiza la arquitectura y el modelado de precisión de ciclos del diseño.
- Asimismo, se definen diferentes estrategias para el proceso de síntesis de alto nivel, teniendo en cuenta las restricciones impuestas por la plataforma FPGA, tanto relativas a los dispositivos FPGA que la forman como a sus posibilidades de configuración.
- Se estudia y define el flujo de diseño a seguir para su posterior aplicación y consecución de los objetivos del trabajo.
- Una vez definido formalmente el flujo de diseño, se procede a la síntesis de alto nivel del decodificador.
- Por último, se hace un estudio comparativo de los resultados obtenidos, se dan las conclusiones de proyecto y se documenta todo este proceso como parte de la memoria final del Trabajo Fin de Máster.

1.3. Peticionario

Actúa como peticionario de este TFM la División de Sistemas Industriales y CAD del Instituto Universitario de Microelectrónica Aplicada de la Universidad de Las Palmas de Gran Canaria, en el marco de las líneas de investigación promovidas por la citada división.

Por otro lado, la realización de un Trabajo Fin de Máster es requisito indispensable para la obtención del título de Máster en Tecnologías de Telecomunicación por el Instituto Universitario de Microelectrónica Aplicada perteneciente a la Universidad de Las Palmas de Gran Canaria.

1.4. Estructura del documento

El presente documento se divide en seis capítulos y dos anexos en los cuales se describe el trabajo realizado:

Capítulo 1: Introducción. Se detallan los antecedentes, objetivos, peticionario y estructura de la documentación.

Capítulo 2: Dominio de aplicación. Se expone en este capítulo el dominio de aplicación de este trabajo. Se da una descripción del estándar de decodificación H.264/AVC y más específicamente del decodificador H.264/AVC utilizado, analizando además su complejidad. Se describe así mismo la situación de partida de este trabajo.

Capítulo 3: Metodología de diseño. Breve introducción a los elementos que forman parte, directa o indirecta, del desarrollo del trabajo, como las herramientas de ayuda al diseño utilizadas, tecnologías y el flujo de diseño seguido.

Capítulo 4: Desarrollo. Se describe la etapa de síntesis así como los resultados obtenidos y su verificación. Se detalla la etapa de implementación así como los resultados y la verificación final.

Capítulo 5: Resultados. Presentación de resultados obtenidos durante las diferentes fases del flujo de diseño.

Capítulo 6: Comparativas. Presentación de comparativas. Se pasa a establecer comparativas entre los resultados obtenidos y con otros trabajos y publicaciones.

Capítulo 7: Conclusiones y trabajos futuros. Se exponen las conclusiones generales del presente TFM, así como las líneas de trabajo futuras.

CAPÍTULO 2: DOMINIO DE APLICACIÓN

2.1. Introducción

El dominio de aplicación en el cual se desarrolla este Trabajo Fin de Máster es el diseño de sistemas de decodificación de vídeo, y más concretamente el estándar H.264/AVC. Por tanto, se considera de interés estudiar en detalle en las características generales de dicho estándar.

2.2. Estándar H.264/AVC

El estándar H.264/AVC representa una evolución de los estándares de vídeo existentes, desarrollado en respuesta a la creciente necesidad de unas mayores tasas de compresión, requeridas por aplicaciones tales como la transmisión de vídeo en tiempo real, almacenamiento multimedia, difusión televisiva, uso de dispositivos móviles, etc. También ha sido diseñado para permitir el uso de un CODEC de vídeo que actuase de una manera flexible en la gran variedad de entornos de red existentes. El uso de este estándar permite manipular vídeo de manera que pueda ser almacenado, transmitido o recibido como datos, en cualquiera de los canales de difusión existentes o futuros [14].

MPEG-2 es el estándar predecesor a H.264/AVC, y ha sido el CODEC de vídeo más usado hasta ese momento. Dicha especificación introduce un nuevo nivel de creatividad y flexibilidad a las capacidades de representación de contenido digital, mediante funciones para la escalabilidad, precisión de muestras de N-bits y formato tipo 4:4:4 de color, y su tratamiento de las escenas visuales. Introduce un número de variaciones de diseño, denominadas perfiles, que actualmente llegan a 19, para una gran variedad de aplicaciones. Sin embargo, el proyecto H.264/AVC retorna a un enfoque más tradicional y limitado en cuanto a la eficiencia de compresión de imágenes, con mayor robustez frente a pérdidas de red [14].

El estándar H.264/AVC cuenta con los mismos bloques funcionales que sus antecesores, adoptando también un algoritmo híbrido de predicción y transformación para la reducción de la correlación espacial y de la señal residual, control de la tasa binaria, predicción por compensación de movimiento para reducir la redundancia temporal, así como codificación de entropía para reducir la correlación estadística. No obstante, lo que causa que este estándar proporcione una mayor eficiencia en la codificación es el modo en que opera cada bloque funcional.

H.264/AVC incluye:

- Predicción intra fotograma (INTRA), característica única de este estándar.
- Transformación por bloques de muestras 4x4, cuyos coeficientes transformados resultan enteros.
- Referencia múltiple para predicción temporal.
- Tamaño variable de los macrobloques a comprimir.
- Precisión de un cuarto de píxel para la compensación de movimiento.
- Filtro de desbloqueo.
- Codificador de entropía mejorado entre otros.

Todas estas mejoras vienen acompañadas de un aumento en la complejidad de la implementación [23, 24].

En H.264/AVC, al igual que en sus antecesores, se definen diferentes perfiles y niveles dentro de los cuales se especifican las restricciones en el flujo de bits. Cada uno de estos perfiles especifica un conjunto de características así como los límites del decodificador, aun cuando los codificadores no necesitan ningún conjunto particular de características de un perfil. Los niveles marcan los límites de los valores que pueden tomar los elementos de la sintaxis de la recomendación o estándar. Por norma general, la carga de procesamiento del decodificador y la capacidad de memoria para un perfil dado se desprende de los diferentes niveles.

En la primera versión de H.264/AVC se definieron tres perfiles: *baseline, main* y *extended*. El primero de ellos se aplica a los servicios de conversación en tiempo real, como videoconferencias o videollamadas. El perfil *main* se utiliza para aplicaciones de almacenamiento digital de vídeo y datos,

así como en la difusión de televisión. El último de los perfiles, el *extended*, es aplicable también a servicios multimedia en Internet. La figura 4 da una muestra de la relación existente entre estos perfiles [15].

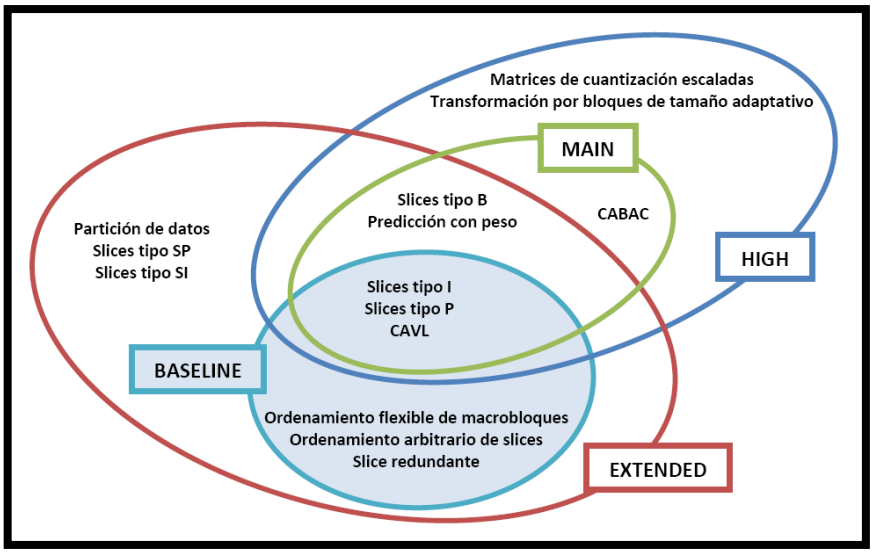


Figura 4. Perfiles comunes y específicos de H.264/AVC

A lo largo de sus años de vida, el estándar ha ido revisándose y modificándose para mejorar su rendimiento. La revisión aprobada en marzo de 2005 [25] contenía modificaciones que añadían cuatro nuevos perfiles, denominados *High, High 10, High 4:2:2,* y *High 4:4:4*, para aumentar la calidad de vídeo y para extender el rango de aplicaciones que abarcaba el estándar. Adicionalmente, se especificó una definición de nuevos tipos de datos suplementarios para ampliar aún más dicho rango, así como una corrección de los errores detectados.

La edición de H.264/AVC publicada en mayo de 2009 contenía extensiones mejoradas para soportar codificación multi-vídeo – MVC, la especificación de un perfil *Constrained Baseline*, y algunas correcciones y clarificaciones misceláneas [25].

Este Trabajo Fin de Máster está centrado en el perfil *baseline*. La tabla 1 muestra los diferentes tamaños de fotograma y número de macrobloques (MB) contenidos para distintos formatos de vídeo.

Formato	Tamaño del fotograma	Nº total de MBs (en un fotograma)
QCIF	176 x 144	99
CIF	352 x 288	396
VGA	640 x 480	1200
4CIF	704 x 576	1584
720p HD	1280 x 720	3600
4VGA	1280 x 960	4800

Tabla 1. Tamaño de fotograma y nº de MBs para distintos formatos de vídeo

2.3. Código de referencia del decodificador H.264/AVC

Tal como se indicó, la finalidad global de este Trabajo Fin de Máster es la síntesis de alto nivel de un decodificador de vídeo H.264/AVC sobre una plataforma FPGA a partir de su código de referencia. Se describe a continuación las características y especificaciones del diseño elegido para la realización de este trabajo.

El presente trabajo arranca de un modelo de decodificador H.264/AVC desarrollado en el proyecto ARTEMI+ [26], descrito en SystemC en alto nivel a partir del código original de referencia de H.264/AVC, correctamente compilado y libre de errores, que describe un sistema decodificador H.264/AVC.

El diagrama de bloques en que se divide el decodificador se muestra en la figura 5. En dicha figura se muestran los bloques y flujo de datos principales.

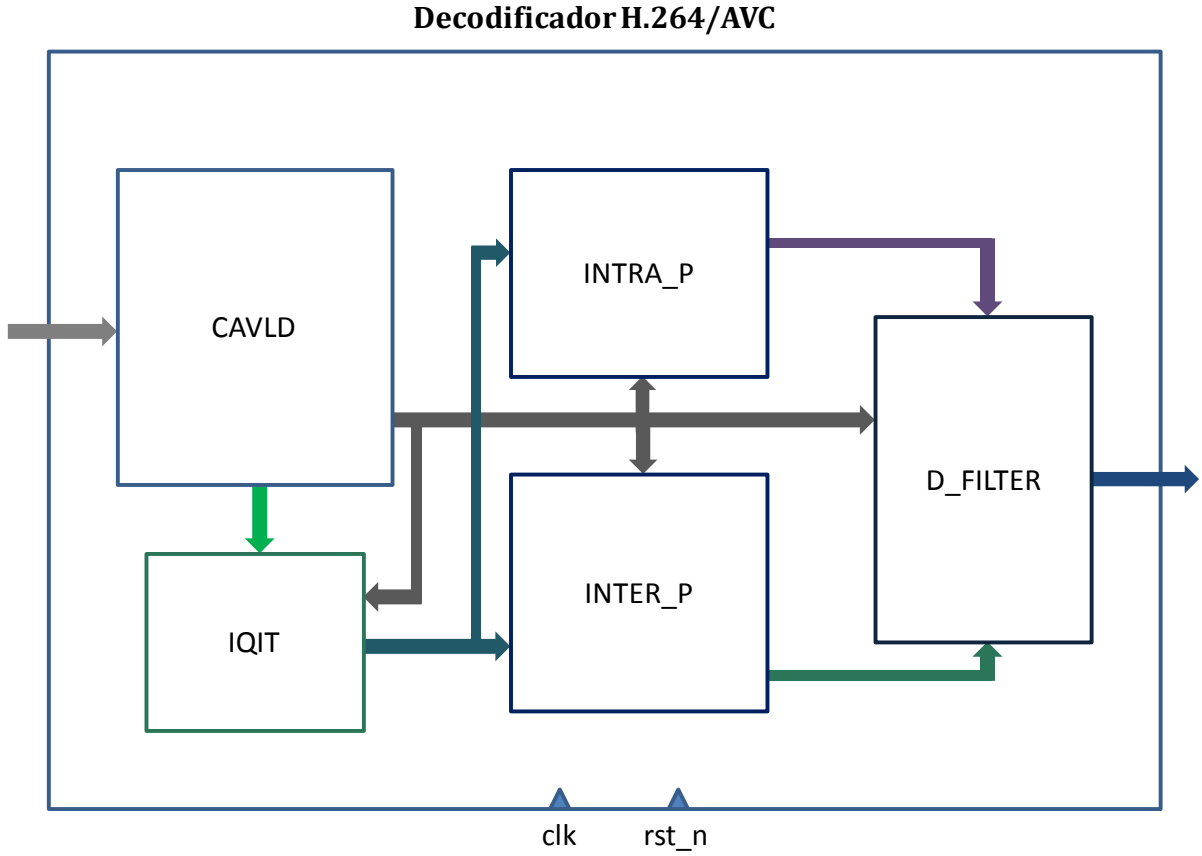


Figura 5. Diagrama de bloques del decodificador H.264/AVC

En los siguientes apartados se describe en detalle los diferentes módulos que componen cada uno de los bloques mostrados anteriormente.

2.3.1. CAVLD

El bloque CAVLD se compone tanto del decodificador CAVLC (*Context-Adaptive Variable Length Coding*), como de los diferentes módulos que participan en las tareas de control del flujo de datos de entrada. Su diagrama de bloques se presenta en la figura 6.

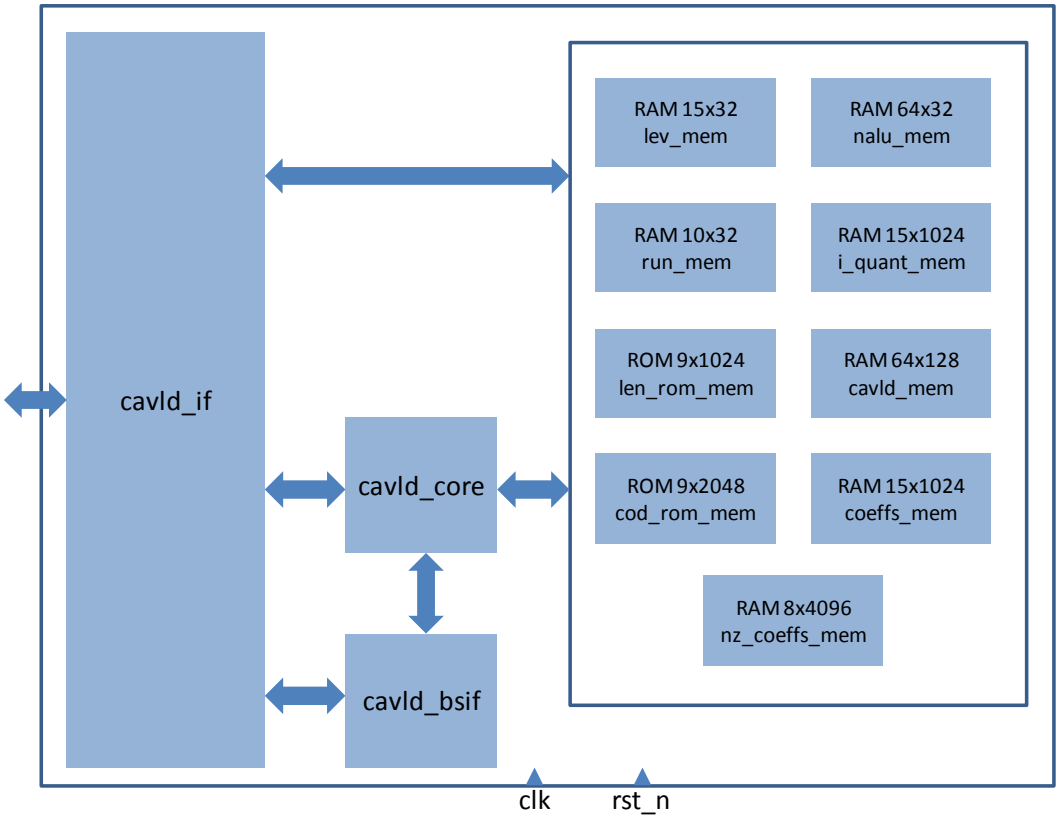


Figura 6. Diagrama de bloques de CAVLD

Se compone de cuatro bloques principales:

- Interfaz de E/S para la gestión del *stream* de entrada (cavld_if)
- Núcleo de procesamiento principal (cavld_core)
- Unidad de tratamiento del *bit-stream* de entrada
- Memorias locales de configuración y de almacenamiento del *bit-stream*

2.3.2. IQIT

Este bloque agrupa a las funciones de cuantización inversa (IQ – *Inverse Quantization)* y transformada inversa (IT – *Inverse Transform*). Se trata del bloque encargado de realizar la decodificación de la información residual proveniente del decodificador CAVLC. Su diagrama de bloques se muestra en la figura 7.

2.3.3. INTRA_P

El bloque INTRA_P incluye los módulos necesarios para que el decodificador realice el tratamiento de los macrobloques con **predicción intra**. Esto incluye la segmentación del flujo de predicción, la preparación de los registros para el almacenamiento de información y el propio procesado de los elementos.

En la figura 8 se muestra su diagrama de bloques. Como se muestra en la figura, hay bloques dedicados para el procesamiento concurrente de las componentes de luma y de croma.

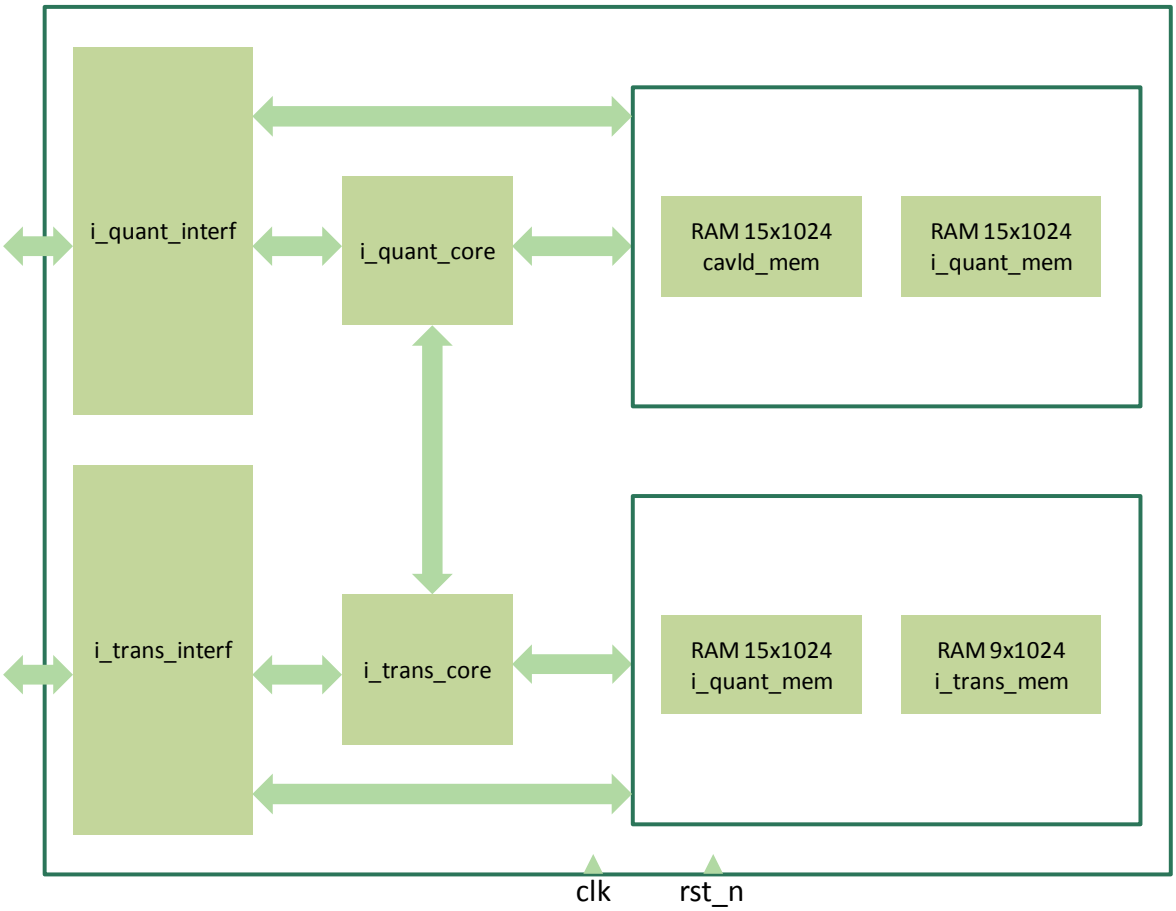


Figura 7. Diagrama de bloques de IQIT

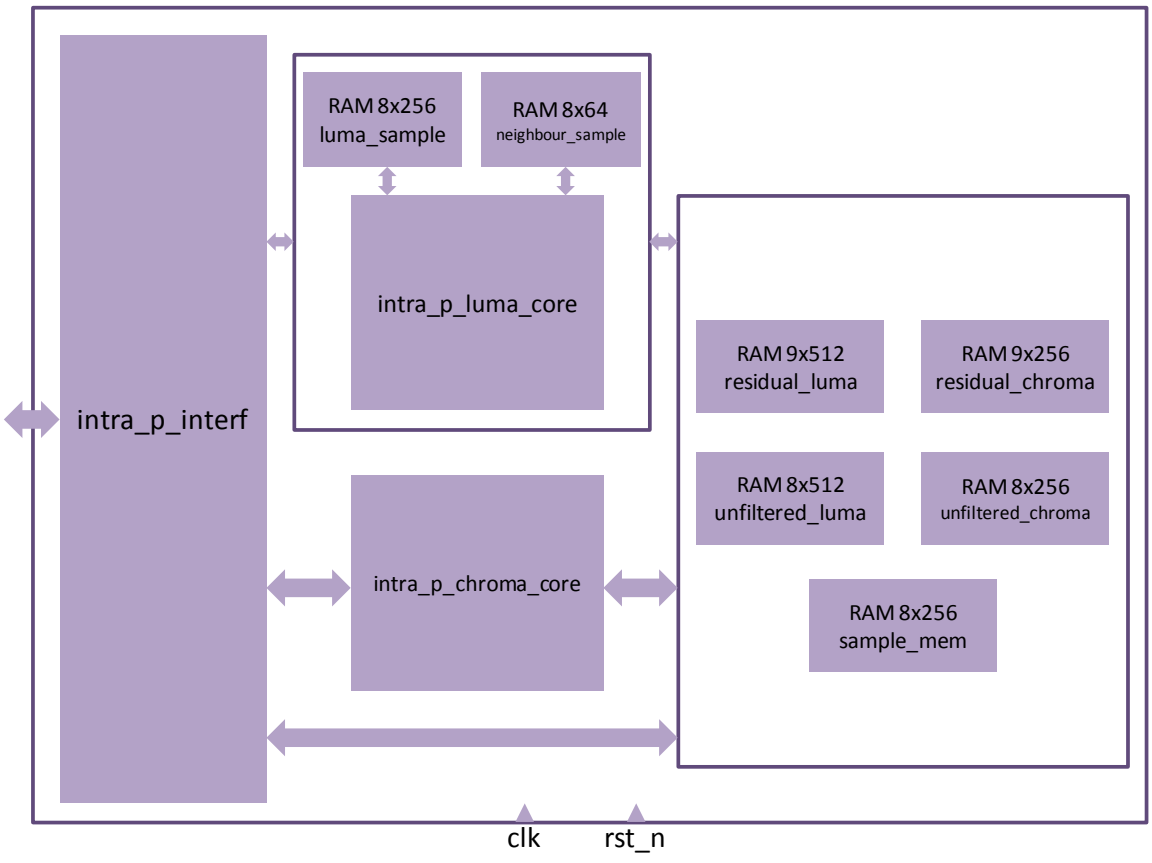


Figura 8.Diagrama de bloques de INTRA_P

2.3.4. INTER_P

Del mismo modo que el bloque INTRA_P, el bloque INTER_P contiene los módulos pertenecientes al flujo de procesado de la predicción tipo **inter**. Estas operaciones se efectúan a partir de los datos de entrada enviados por el bloque CAVLD, como se muestra en la arquitectura global del decodificador (figura 5). De igual forma se procesan de forma concurrente las componentes luma y croma y se dispone de memorias locales para el almacenamiento de los residuos, vectores de movimiento, bloques procesador y bloques listos para ser filtrados en pasos posteriores.

El diagrama de bloques correspondiente al INTER_P se expone en la figura 9 .

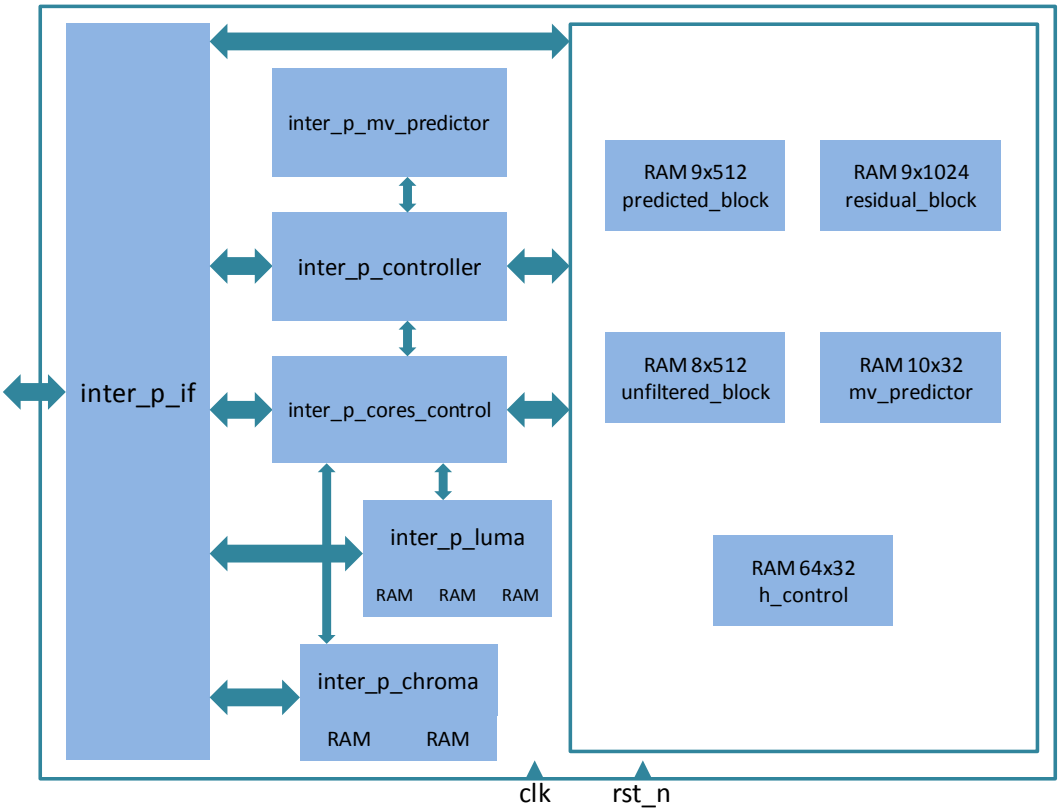


Figura 9. Diagrama de bloques de INTER_P

2.3.5. D_FILTER

El bloque D_FILTER incluye a todos los módulos pertenecientes a las funciones del filtro de suavizado de bordes.

Como en apartados anteriores, la figura 10 muestra su diagrama de bloques. Este bloque precisa de recursos de almacenamiento para el almacenamiento local de los macrobloques vecinos, de tal forma que se pueda realizar un filtrado y suavizado con los macrobloques predecesores del bloque actual, ya filtrados. El orden del filtrado es lexicográfico: izquierda a derecha y de arriba abajo. El resultado producido al final del procesamiento de este bloque es el fotograma reconstruido.

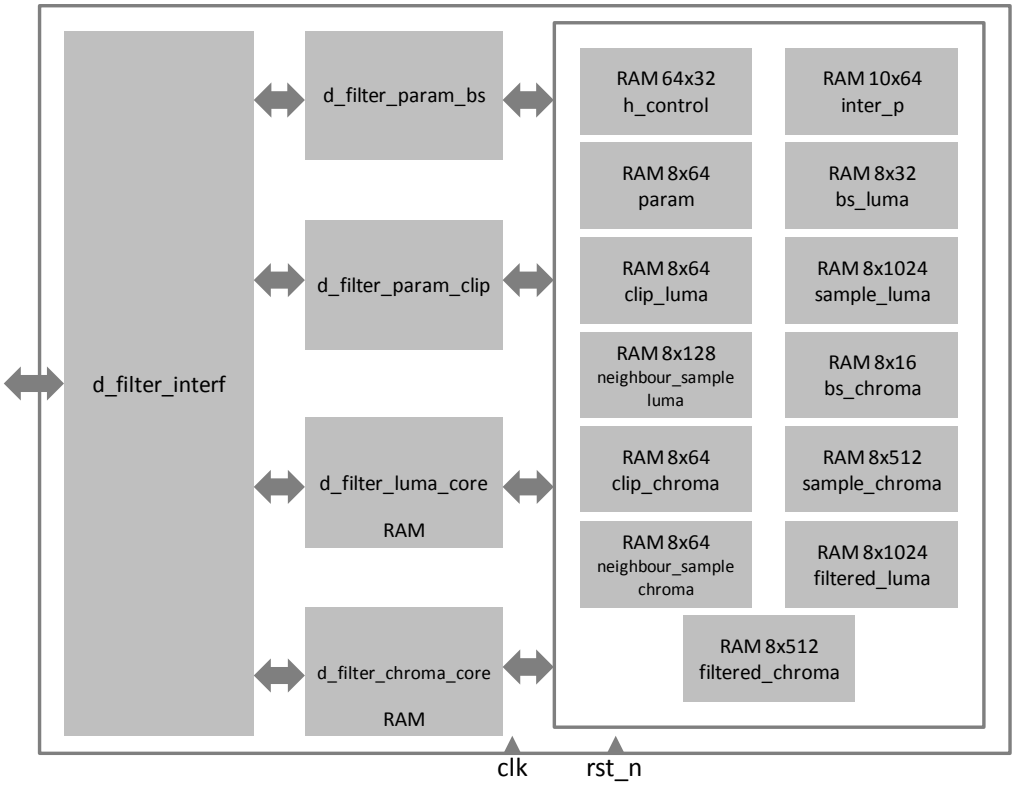


Figura 10. Diagrama de bloques de D_FILTER

2.4. Conclusiones

En este capítulo se ha mostrado el dominio de aplicación en el cual se desarrolla este Trabajo Fin de Máster.

Por una parte se ha explicado brevemente el estándar de codificación de vídeo H.264/AVC, haciendo una pequeña introducción histórica así como exponiendo sus características más destacables y los perfiles que lo conforman.

Para finalizar, se han presentado los bloques principales que constituyen el código de referencia del decodificador de vídeo utilizado, entrando en la jerarquía del sistema y dando información sobre la funcionalidad de los diferentes bloques que lo componen.

Con ello tenemos una visión clara de los bloques modelados a utilizar en el trabajo, sus entradas y salidas y la estructura interna del decodificador H.264/AVC a nivel de grandes bloques. Este nivel de granularidad es suficiente para la realización posterior de la síntesis del procesador.

CAPÍTULO 3: METODOLOGÍA DE DISEÑO

3.1. Introducción

Para cumplir los objetivos de este Trabajo Fin de Máster descritos con anterioridad, en este capítulo se describirá la metodología usada y como consecuencia de esta, los lenguajes, herramientas y tecnologías necesarias para su realización.

Se entiende como metodología al conjunto de métodos que se siguen en una investigación científica o en una exposición doctrinal. De la misma forma, se define método (del griego *meta* -más allá- y *hodos* –camino-) como el procedimiento que se sigue en ciencias para hallar la verdad y enseñarla [27]. Generalmente el método, en su más estricta versión original, utiliza tres fases: la observación y la experimentación, la recopilación de resultados y la comprobación de las hipótesis de partida

Para este trabajo, se ha seguido una metodología de diseño orientada a la síntesis de alto nivel, partiendo de una descripción en lenguaje SystemC. La transformación de los modelos funcionalmente correctos, permite verificar la hipótesis inicial de facilitar la ruta hacia la implementación hardware desde modelos funcionales, sin necesidad de realizar rediseños en niveles

intermedios (RT), con una penalización en consumo de recursos aceptables, manteniendo las prestaciones exigidas.

3.2. Metodología de diseño

En todo diseño, ya sea *hardware* y/o *software* se pueden enumerar tres fases claves en la metodología de diseño: especificación, diseño y verificación [6].

En la primera fase se analizarán las especificaciones del sistema a realizar, ya sean estas funcionales, temporales o de cualquier otra índole que se haya dado. Una vez definidas y acotadas, se procederá al diseño del sistema en el que se decidirá la tecnología, las etapas de desarrollo, la partición del diseño (si la hubiese), etc. Ello produce una primera versión funcional del sistema que se utilizará para su verificación.

Con una primera versión terminada, se verificará que cumple con las especificaciones definidas en primera instancia para adoptar las medidas correctoras necesarias en fases tempranas del ciclo de diseño, repitiendo el proceso hasta obtener una versión que cumpla con las especificaciones funcionales (figura 11).

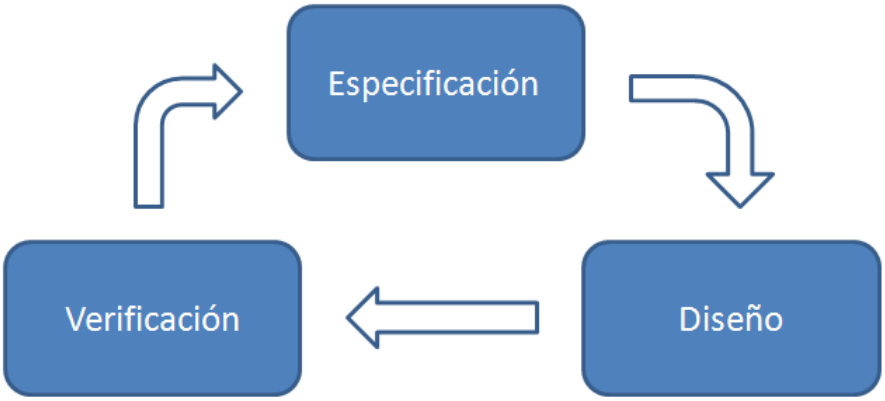


Figura 11. Simplificación del flujo de diseño. Adaptado de [28]

3.3. Lenguajes

Debido a la creciente complejidad de los diseños de sistemas electrónicos, es necesario disponer de nuevas herramientas para realizar la captura del diseño. En la actualidad, hay una tendencia cada vez mayor a la descripción del sistema electrónico a nivel de sistemas (ESL) [29], por lo que no han tardado en aparecer lenguajes de descripción hardware con un nivel de abstracción mayor tales como son SystemC y SystemVerilog.

3.3.1. SystemC

Es el lenguaje de descripción de alto nivel en el cual está diseñado el sistema a tratar en el presente TFM. Se detallan a continuación algunos conceptos de interés para la correcta comprensión de las innovaciones que aporta este lenguaje en las metodologías de diseño *hardware*.

Para profundizar en las características de SystemC se recomienda la lectura de [30], el manual de referencia de la última versión de la librería, así como con guías de diseño y ejemplos [31].

3.3.1.1. Definición

SystemC es una librería de clases de C++ desarrollada conjuntamente con una nueva metodología de diseño y verificación funcional para el modelado *hardware* a nivel de sistemas.

El uso de SystemC, en conjunto con las herramientas estándar de desarrollo para C++, tiene como objeto poder modelar a nivel de sistema el diseño electrónico, y poder validarlo con los menores costes temporales posibles, además de conseguir un modelo que cumpla las especificaciones con el fin de que la siguientes fases de diseño puedan usarlo como referencia de funcionalidad.

La razón de elegir C++ como lenguaje de alto nivel para el desarrollo de la librería SystemC es que se trata de un lenguaje de alto nivel muy extendido, además de que sus compiladores generan códigos muy eficientes.

SystemC proporciona aquellas características que no están soportadas por C++, necesarias para el desarrollo de sistemas electrónicos:

- **Información temporal**: C++ no mantiene información temporal de los instantes lógicos en los que se producen los eventos.
- **Concurrencia**: C++, y en general cualquier lenguaje orientado al diseño de *software* no está preparado para describir sistemas concurrentes, algo que es intrínseco en el diseño de *hardware*.
- **Tipos de datos**: Los tipos de datos que soportan los lenguajes de programación no contemplan factores necesarios en el *hardware* como son el valor de alta impedancia Z en una señal.

La librería de SystemC, a través de la definición de nuevas clases de objetos en C++, da solución a estas carencias sin la necesidad de redefinir sintácticamente el lenguaje. Esto sumado a que se permite insertar código en C y C++ en su diseño, permite realizar una coverificación HW/SW.

3.3.1.2. Elementos principales

A continuación se detallan algunos elementos de relevancia en el modelado de sistemas con SystemC.

1. Módulos

Es la clase usada por SystemC para modelar la estructura del diseño, dando soporte a conceptos tales como jerarquía, interconexión, etc. Un módulo encapsula un componente del diseño, que puede contener a su vez un conjunto de módulos interconectados a través de canales.

```
SC_MODULE (adder_reg) {
  …
};
```

2. Puertos

En cada módulo podrán definirse puertos de entrada, de salida y bidireccionales, con el fin de interconectar módulos entre sí, o si fuese el módulo jerárquicamente superior, para representar las entradas y salidas del diseño. Además del sentido del puerto, cada uno podrá tener un tipo de datos, que podrá ser cualquier tipo de datos soportado por C/C++, tipos definidos por el usuario, o tipos específicos de la librería SystemC.

```
sc_in<bool> clk;
sc_in<sc_int<8> > a;
sc_in<sc_int<8> > b;
sc_out<sc_int<9> > c;
```

3. Señales/canales

Al igual que los puertos, sirven para comunicar entre módulos la comunicación y procesos. Las señales son internas a cada módulo y no son visibles desde otros módulos externos al contenedor de la señal en cuestión. Como los puertos, los datos pueden ser de tipos de C/C++, de usuario o de SystemC.

```
    sc_signal<sc_int<9> > temp;
```

4. Procesos

En ellos se describe las funcionalidad de un módulo, que consistirá en un código escrito en C/C++ haciendo uso tanto de funciones del lenguaje raíz, como de métodos de las clases de la librería SystemC. Existen dos tipos principales de procesos: SC_METHOD, SC_THREAD. Para mantener la compatibilidad con versiones anteriores del lenguaje también están disponibles los SC_CTHREAD.

SC_THREAD

Son procesos que solo se ejecutan una vez al comienzo de la ejecución del sistema. Sin embargo, permiten suspender el proceso con el fin de ser reanudado en el mismo estado en el que fue suspendido cada vez que se produce un evento en su lista de sensibilidad. Cada vez que la ejecución del proceso encuentre una secuencia *wait()*; entrará en suspensión. En general, los

procesos de este tipo suelen definirse como un bucle infinito, de forma que cuando termine vuelva a ejecutarse, y se define su latencia como el número de eventos necesario para una ejecución del bucle. Puede usarse para definir la ejecución de un camino de datos segmentados, por ejemplo.

A continuación se muestra un ejemplo de declaración, definición, registro y lista de sensibilidad de un proceso de tipo SC_THREAD. Sin embargo, se recomiendo que la definición del mismo se encuentre separado en un fichero *.cpp, y el resto en un fichero de cabecera *.h [32].

```
SC_MODULE (adder_reg) {
…
void reg();
…
void adder_reg::reg() {
while (true) {
…
wait();
…
  }
}
…
SC_THREAD(reg);
sensitive << clk.pos();
```

SC_CTHREAD

Se trata de de una modificación sobre los tipo SC_THREAD. Su principal diferencia es que en el registro se le añade un evento como sensibilidad, de forma que en la definición pueden realizarse nuevas formas de suspensión. Un ejemplo es la realización de una suspensión que se reanude un número de eventos más tarde, en lugar de en el siguiente evento. Otra es la posibilidad de suspender el proceso hasta que se cumpla una condición (y se genere el evento).

Estas formas de suspensión pueden también conseguirse con el proceso SC_THREAD como se muestra a continuación, pero requiere que el *kernel* reanude el proceso para volver a ponerlo en reposo en cada evento, haciendo la ejecución más lenta.

A continuación se muestran los modos de suspensión comentados, a la izquierda en su implementación para un SC_THREAD y su equivalente en un SC_CTHREAD a la derecha.

`for (i=0; i!=N; i++) wait();`	`wait(N);`
`do wait() while(!expr);`	`wait_until(expr.delay);`

SC_METHOD

Los procesos de tipo SC_METHOD se ejecutan cada vez que se produce un evento en su lista de sensibilidad y no pueden ser suspendidos. En general se usan para simular un comportamiento combinacional ya que se realiza en tiempo cero y no se suspende para volver a reanudarse. Al igual que en el tipo SC_THREAD se debe indicar en la lista de sensibilidad los eventos que provocarán la ejecución del método.

```
SC_MODULE (adder_reg) {
…
void add();
…
void adder_reg::add() {
  …
}
…
SC_METHOD(add);
sensitive << a << b;
```

A continuación se detalla un ejemplo de diseño de un sumador con la salida registrada en el que se hace uso de dos procesos en paralelo. Uno de ellos se encargará del proceso combinacional de realizar la suma y otro de registrar la señal [33] (figura 12).

```
#include "systemc.h"

SC_MODULE(adder_reg) {
  sc_in<sc_int<8> > a;
  sc_in<sc_int<8> > b;
  sc_out<sc_int<9> > c;
  sc_in<bool> clk;

  sc_signal<sc_int<9> > temp;

  void add() { temp = a + b; }
  void reg() { while (1) {c = temp; wait();} }

  SC_CTOR (adder_reg) {
    SC_METHOD(add);
    sensitive << a << b;

    SC_THREAD(reg);
    sensitive << clk.pos();
  }
};
```

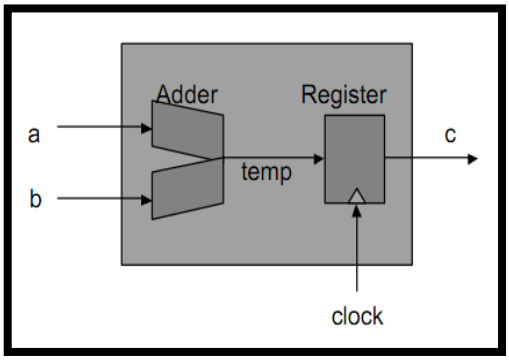


Figura 12. Diseño de ejemplo de uso de procesos en SystemC

Como puede observarse en el ejemplo anterior, por un lado existe un proceso de tipo SC_METHOD que se ejecuta en tiempo cero cada vez que se produzca un evento en alguno de los puertos de entrada (a y/o b), realizando la suma de ambas y almacenándola en *temp*. Por otro lado, en cada flanco de subida del reloj, el valor almacenado en *temp* se escribe en el puerto de salida *c*. Esta descripción cumple funcionalmente con el diseño propuesto en la Figura 12, incluyendo que en

un ciclo hayan varias modificaciones de las señales de entrada, almacenándose en la salida la última antes del flanco de subida del reloj (el proceso SC_METHOD se ejecutará, tantas veces como cambios haya en los puertos de entrada, pero solo se escribirá en el puerto de salida el valor que tuviese en el momento en el que se reanuda el proceso SC_THREAD).

5. Relojes

Los relojes son señales con una notación temporal, que permite dotar al sistema de relaciones temporales en su simulación. Se pueden definir varios relojes, con distintos tiempos de ciclo, ciclos de trabajo, así como fase relativa arbitraria.

```
sc_clock clk("clk", 10, SC_NS, 0.5, 0.0, SC_PS);
```

En el ejemplo anterior se ha declarado un reloj llamado *clk*, con un periodo de 10 ns, un ciclo de trabajo del 50% y un desfase de 0 ps.

Las variables temporales en SystemC se almacenan como un entero de 64 bits y las unidades en las que se pueden representar son: *SC_FS*, *SC_PS*, *SC_NS*, *SC_US*, *SC_MS*, *SC_SEC*.

3.3.1.3. Características principales

Algunas características fundamentales de SystemC que hacen de este lenguaje una buena alternativa en el diseño *hardware* de alto nivel son las siguientes:

a. **Simulación basada en eventos.** Al contrario que en una simulación por ciclos de reloj, la simulación basada en eventos presenta unos tiempos de simulación mucho más bajos, ya que no precisa de la ejecución de todos los procesos paralelos del sistema en cada ciclo de reloj, sino que pueden haber varios suspendidos en espera de que se cumpla uno de sus eventos.

b. **Múltiples niveles de abstracción.** SystemC permite realizar modelos *untimed* con varios niveles de abstracción, desde funcional hasta RTL. Esto permite refinar iterativamente el código hasta llegar a un modelo RTL que cumpla la funcionalidad y usarlo como fuente para pasar a un diseño en VHDL o Verilog y de ahí seguir el flujo estándar de diseño electrónico.

c. **Trazado de formas de onda.** Con SystemC es también posible el trazado de formas de ondas en los formatos VCD, WIF e ISDB.

3.3.1.4. Estructura de capas

La figura 13 muestra la arquitectura del lenguaje SystemC, siendo los bloques centrales el núcleo de este lenguaje. Como puede observarse, está construido sobre C++ [31].

Las capas superiores son librerías y estándares de diseño que el usuario puede decidir usar o no. Gracias a esta estructura se pueden seguir añadiendo librerías sobre el núcleo sin tener que modificar el mismo.

El núcleo del lenguaje está formado por el simulador orientado por eventos, junto a los elementos ya comentados (puertos, módulos, procesos, etc.). Los tipos de datos son necesarios para

el modelado *hardware* y ciertos tipos de desarrollo *software*. Los canales primarios son los canales incorporados que tienen un amplio uso tal como las señales y las FIFOs.

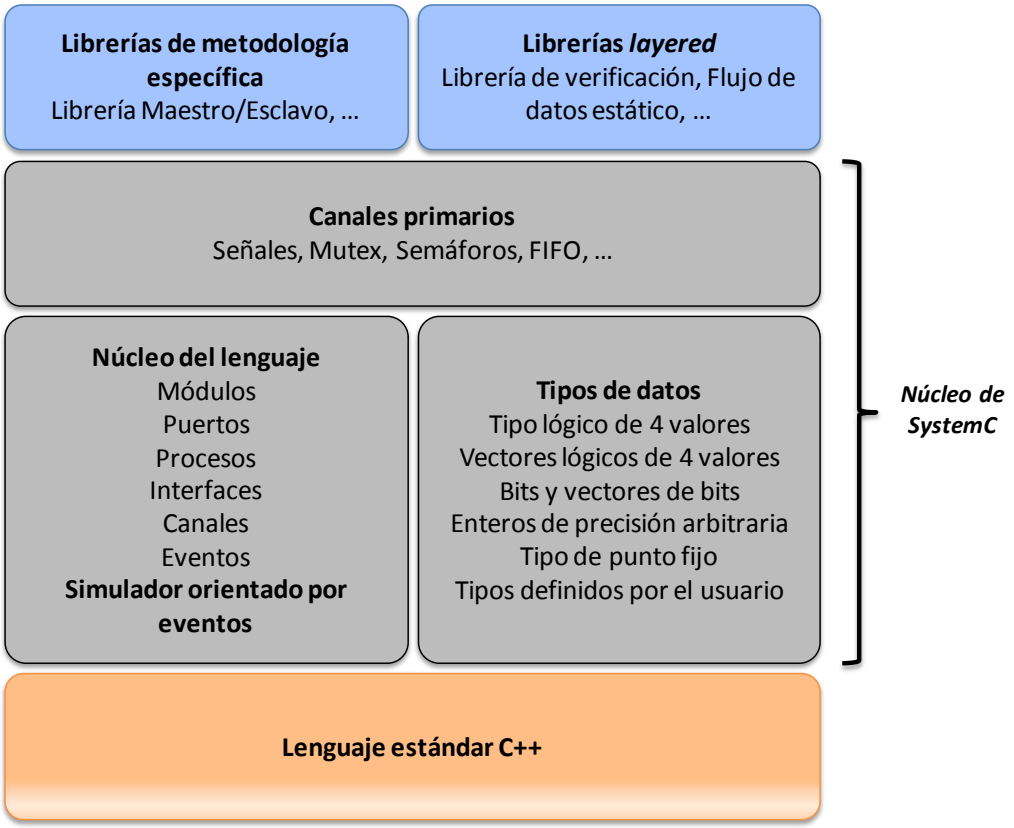


Figura 13. Arquitectura de capas de SystemC [6]

3.3.1.5. Metodología de diseño

Como ya se indicó anteriormente SystemC representa, además de una librería de clases, una metodología de diseño que tiene como objetivo acelerar las fases de diseño a nivel funcional así como su verificación.

En un flujo tradicional de diseño electrónico es una práctica habitual realizar un diseño funcional en C/C++ con el fin de crear un modelo de alto nivel que cumpla las especificaciones tras varias iteraciones de diseño y optimización, para luego realizar una traducción a un lenguaje de descripción *hardware* a nivel RTL (VHDL o Verilog). Este tipo de metodología presente inconvenientes conocidos como pueden ser:

- Traducir el código C/C++ a VHDL/Verilog puede producir errores difíciles de depurar. Aún con un modelo en C/C++ que cumpla con las especificaciones, es difícil realizar una traducción sistemática.
- Caducidad del modelo C/C++. Una vez el diseño se realiza a nivel RTL, todas las optimizaciones se realizarán sobre este, lo que llevará a que el modelo en alto nivel quede desfasado.
- Imposibilidad de reutilizar los test de diseño. La naturaleza de los tests de una aplicación en C/C++ son muy diferentes a un *testbench* de un sistema *hardware*, lo que obligará a rediseñar los sistemas de test para la versión RTL del diseño.

La metodología que presenta SystemC pretende dar solución a los problemas anteriormente citados y se representa en el diagrama de flujo de la figura 14.

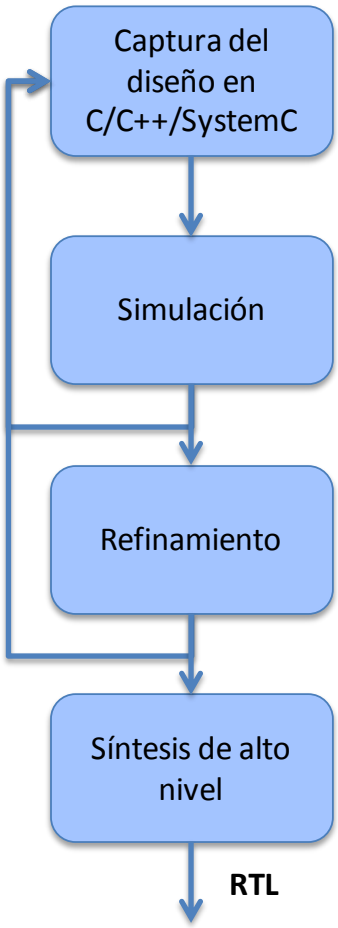


Figura 14.Metodología de diseño basada en SystemC

Las principales ventajas de este flujo modificado pueden resumirse en los siguientes puntos:

- La metodología de refinamiento permite escribir una primera versión del sistema en C/C++ e ir incluyendo características de SystemC al diseño en varias iteraciones.
- El realizar el sistema en SystemC para luego sintetizarlo hace que no sea necesario conocer otros lenguajes de descripción *hardware*.
- El tiempo de desarrollo se reduce, tanto en diseño como, lo que es más importante, en verificación.
- Los *testbenchs* pueden ser reutilizados desde la primera versión hasta la versión más refinada y cercana a RTL.

Una parte importante de la metodología implica la verificación de que la solución obtenida sea equivalente a la especificada. En este caso, se debería volver a realizar la verificación del sistema, adaptando el *testbench* inicial a los requerimientos de latencia presente en el modelo preciso a nivel de ciclos, obtenidos durante la síntesis de alto nivel. Otra forma de realizar la verificación es utilizar métodos de comparación formal entre las diferentes representaciones.

3.4. Herramientas

3.4.1. C-to-Silicon Compiler

Cadence CtoS (C-to-Silicon Compiler) es una herramienta de síntesis de alto nivel. Este tipo de herramientas tiene como objeto reducir el tiempo de diseño en sistemas complejos y, en la mayoría de los casos, mejorar la calidad de los resultados en comparación con las traducciones hechas a mano por los diseñadores.

CtoS genera una descripción RTL del diseño en el HDL Verilog, estándar IEEE 1364, de forma que el diseño puede seguir la ruta de síntesis mediante herramientas de síntesis lógica disponibles, ya sea Cadence Encounter RTL Compiler o cualquier otra de las que existen en el mercado (Xilinx Synthesis Technology, Synplify Pro/Premier, etc.). Además del código RTL, CtoS genera descripciones de comportamiento, tanto en SystemC como en Verilog, para su simulación funcional [34].

Existen dos modos de ejecución de la síntesis en CtoS, ya sea en modo interactivo mediante el uso de una interfaz gráfica y en modo *batch* mediante el uso de *scripts* en TCL. Normalmente, la primera vez que se sintetiza un diseño se hará en modo interactivo para facilitar la toma de decisiones de forma interactiva por parte del diseñador con la ayuda de las herramientas gráficas que posee el entorno. Cada decisión tomada en ella, generará una o varias órdenes de consola, que pueden consultarse para escribir el *script* de síntesis correspondiente.

3.4.1.1. Flujo de diseño

En la se representa el flujo de diseño en CtoS que incluye las etapas que se enumeran y describen a continuación.

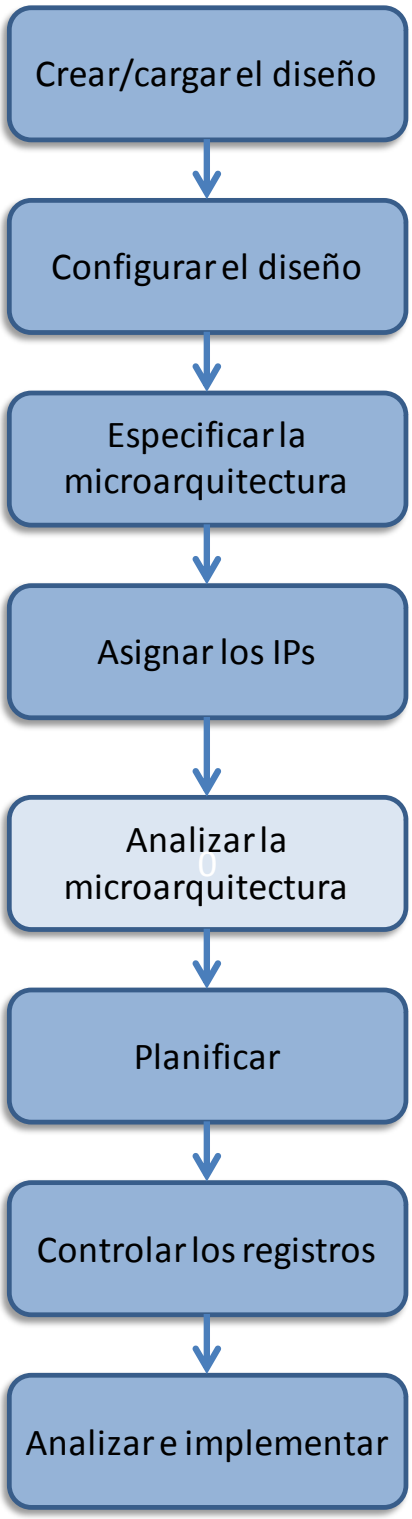
1. **Crear/cargar el diseño**

El primer paso a realizar consiste en crear un nuevo diseño o abrir uno existente. Existen diferentes alternativas, ya sea el diseño basado en C/C++, SystemC o TLM.

2. **Configuración del diseño**

Aquí habrá que indicar si el diseño será SystemC, TLM o C/C++. Se definirá aquí la fuente de reloj, con sus propiedades de nombre, frecuencia, ciclo de trabajo, etc. Se incluirán los ficheros fuente, así como aquel que corresponde al *top* del diseño.

3. Especificación de la micro-arquitectura



En este apartado se realizan las transformaciones arquitecturales para hacer el diseño sintetizable. Los elementos a transformar son funciones y bucles.

Algunas funciones no son sintetizables por tener dos caminos diferentes con distinta latencia, lo cual impide al que hace la llamada conocer la espera que debe realizar. Otra causa puede ser que acceda a vectores donde se pueda escribir. Esto se soluciona haciendo las funciones "en línea" (*inline*), es decir, sustituir la llamada por el código que la describe.

Por otro lado, los bucles combinacionales (bucles que se realizan idealmente en tiempo cero) no son sintetizables, por lo que puede decidirse entre varias alternativas de implementación:

- Desenrollar el bucle para ser realizado enteramente en tiempo cero. Esto puede ser inviable para bucles de muchas iteraciones, pues haría del bucle la ruta crítica con una latencia muy alta.
- Romper el bucle para que cada iteración se realice en un ciclo de reloj.
- Realizar una segmentación el bucle. Se divide la ejecución del bucle en un número determinado de etapas, de tal forma que cada iteración del bucle se encuentra en una etapa del mismo simultáneamente, paralelizando así varias iteraciones.

Decidir qué solución tomar es responsabilidad del diseñador, y deberá tener en cuenta restricciones tanto de área como de frecuencia. La Figura 15 muestra una imagen de la herramienta durante este paso

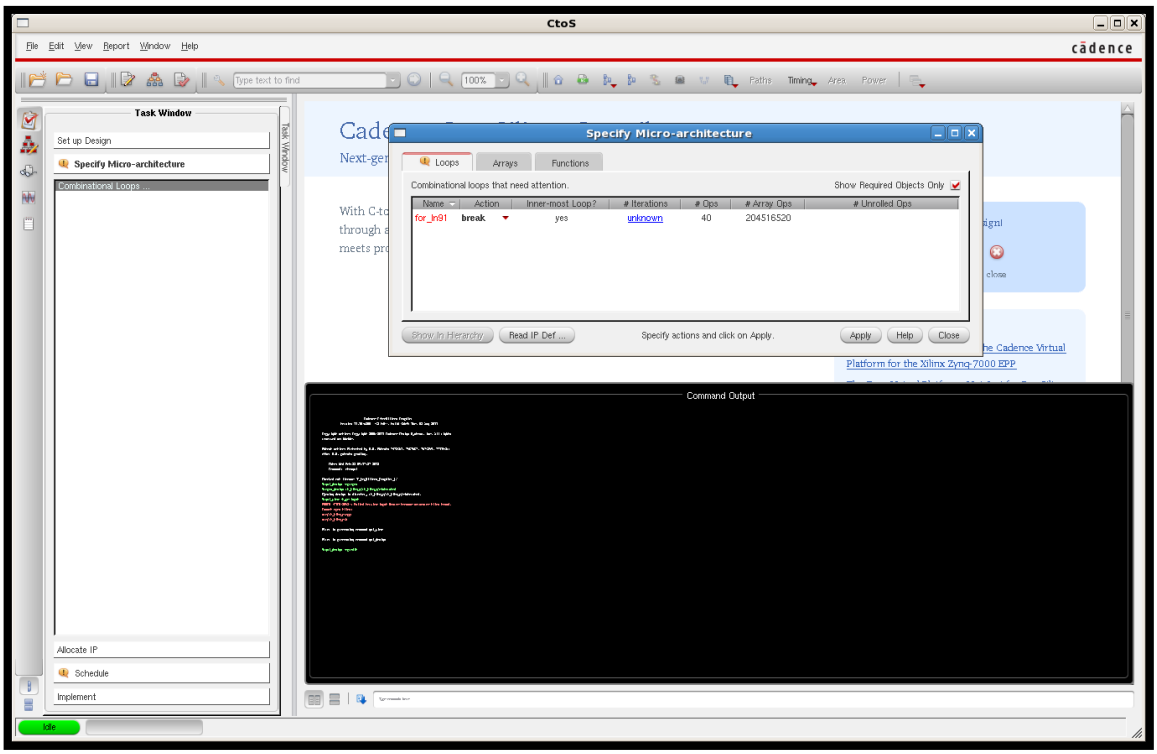


Figura 15. Etapa de especificación de micro-arquitectura en CtoS.

4. Asignación de memorias e IPs

En este paso se asignarán las variables de tipo vector/*array* del diseño a memorias internas de la FPGA, o a descripciones HDL de memorias. También es posible asignar IPs a determinadas funcionalidades.

5. Análisis de la micro-arquitectura (opcional)

Llegados a este punto, el diseño está completamente transformado, a falta de que CtoS realice las optimizaciones oportunas, por lo que puede realizarse un análisis temporal, de área y de potencia (menos exacto que el que se realiza tras el planificador) con el fin de modificar algunas de las decisiones tomadas anteriormente.

6. Planificación temporal/asignación de recursos (schedulling/allocation)

En este punto, el planificador de CtoS asignará recursos a las operaciones del diseño fuente. En caso de que el planificador no pueda resolver esta tarea, se presentan algunas acciones que faciliten la obtención de una solución:

a. Controlar la dependencia con vectores de datos. Esto hará que el planificador conozca las dependencias y pueda añadir estados para resolverlas.

b. Controlar los estados. En caso de que el planificador no pueda resolver conflictos secuenciales, el diseñador puede añadir manualmente estados para guiar a CtoS a encontrar una solución a dichos conflictos.

c. Controlar los recursos. El diseñador podrá a priori, crear unos recursos iniciales sobre los que mapear las operaciones del diseño.

7. Asignación y control de registros

En esta etapa puede decidirse si incluir una lógica de *reset* adicional para aquellos registros que por defecto no lo tengan. También puede decidirse si registrar o no todas las salidas de los recursos de lógica combinacional asociados a operaciones del diseño. Aquí se debe decidir entre minimizar registros (decisión normalmente asociada a diseño ASIC donde el coste en área de los registros es muy alto) o minimizar el número de multiplexores (asociado normalmente al diseño basado en FPGAs).

8. Análisis e implementación

Con objeto de analizar la calidad de los resultados obtenidos, la herramienta proporciona informes de distinto tipo, tales como consumo de recursos, o de latencia en ciclos de reloj de cada bloque. Todos los datos pueden ser mostrados sobre el grafo de flujos de datos y de control (CDFG – *Control Data Flow Graph*) de cada comportamiento implementado en el diseño.

También se podrán generar aquí los ficheros de salida, como puede ser la descripción RTL, un script que realice la mismas acciones que las realizadas a través de la interfaz gráfica, o un fichero contenedor en SystemC de la descripción RTL, con el fin de poder realizar una simulación de esta, usando el *testbench* en SystemC con el que se verificó el modelo en alto nivel. La **Figura 16** muestra el grafo de control y datos generado por CtoS.

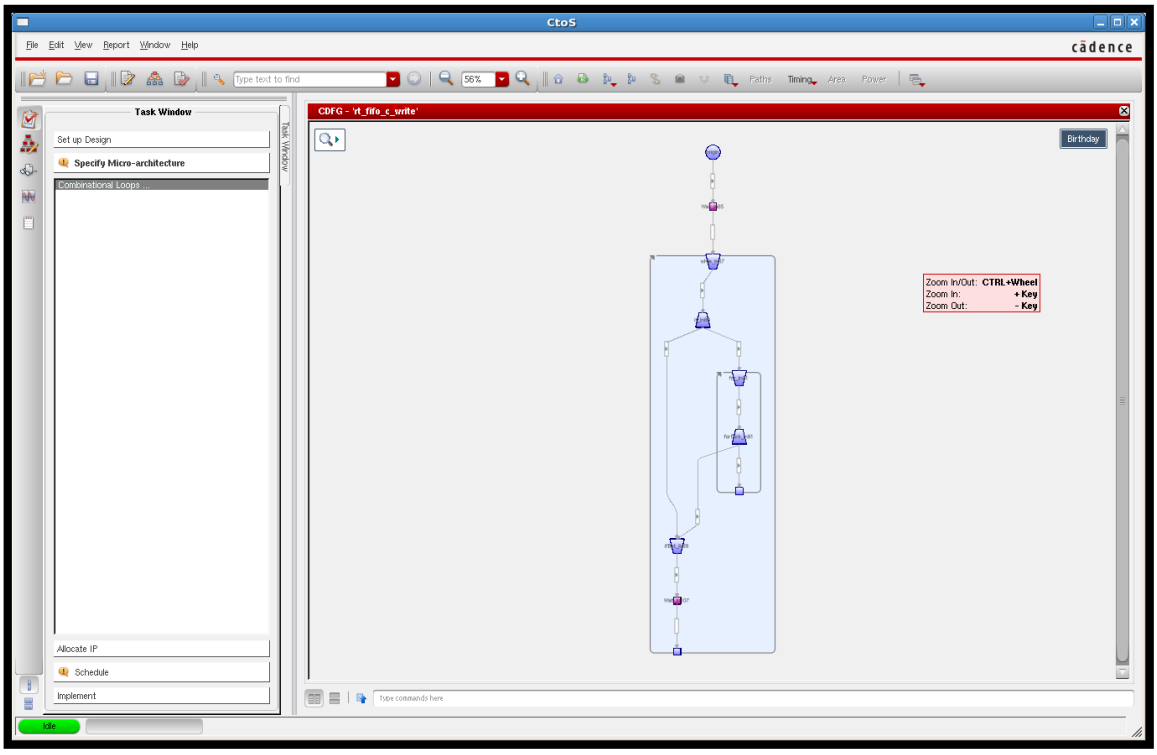


Figura 16. Grafo de control y datos en CtoS

3.4.1.2. Características no soportadas de SystemC

Como se comentó en el apartado de lenguaje SystemC, este está pensando no sólo para la realización de síntesis de alto nivel, sino también para descripciones puramente de comportamiento, y es por ello que muchas de sus funcionalidades no son sintetizables por las herramientas de síntesis de alto nivel.

Tipo de características	Características no soportadas	
Clases del lenguaje	get_child_objects sc_process_handle sc_event_and_list sc_event_or_list sc_event sc_time register_port	default_event sc_fifo sc_buffer sc_mutex sc_semaphore sc_event_queuesc_clock
Tipos de datos	get_child_objects sc_generic_base	sc_numrep float
Utilidad de clases	sc_trace sc_report sc_report_handler sc_exception	sc_copyright sc_version sc_release
Anclaje de puertos	Solo podrán hacerse estáticamente en el constructor del módulo.	
Punteros	Solo se podrán usar para apuntar a variables de forma que estáticamente cada puntero pueda saberse a que variable apunta.	

Tabla 2. Características no soportadas de SystemC en CtoS

3.4.2. Synplify Premier

Synopsys Synplify Premier es la herramienta seleccionada en este trabajo para la realización de la síntesis lógica. Está especialmente centrada en el diseño sobre FPGA [35].

Synplify Premier soporta las siguientes características necesarias en la fase de síntesis de diseño hardware:

- Diseño jerárquico. Debido a la creciente complejidad de los diseños y a su tamaño, cada vez más es típica la división en subdiseños con el fin de trabajar en paralelo. Synplify Premier permite la realización de proyectos divididos en subproyectos independientes. Permite tanto diseño *top-down* como *bottom-up*.
- Modo rápido. Es un modo de síntesis que se realiza con una aceleración de hasta 3x, permitiendo realizar un análisis inicial del sistema.
- Multiprocesado usando particiones. Pueden definirse puntos de compilación en terminología de Synplify (*Compile Points*) que son en realidad particiones del diseño con el fin de lanzar en paralelo la síntesis de cada una de las particiones, con el consecuente ahorro de tiempo de síntesis.
- Diseño incremental. Haciendo uso de los puntos de compilación, se pueden heredar soluciones de cada partición sintetizadas con anterioridad, para que únicamente sea

necesario resintetizar la partición que haya sido modificada. Así, durante la fase de optimización, se puede ahorrar una gran cantidad de tiempo de procesamiento si toda la optimización se realiza en un número reducido de particiones. Además, estas divisiones pueden portarse a las siguientes herramientas de posicionado y ruteado.

3.4.2.1. Vistas

En Synplify puede representarse el diseño en dos vistas, una antes de realizar la síntesis lógica, que se conoce como vista RTL, en la que se representa a modo de bloques la descripción de los ficheros fuentes HDL. La figura 17 muestra la vista RTL.

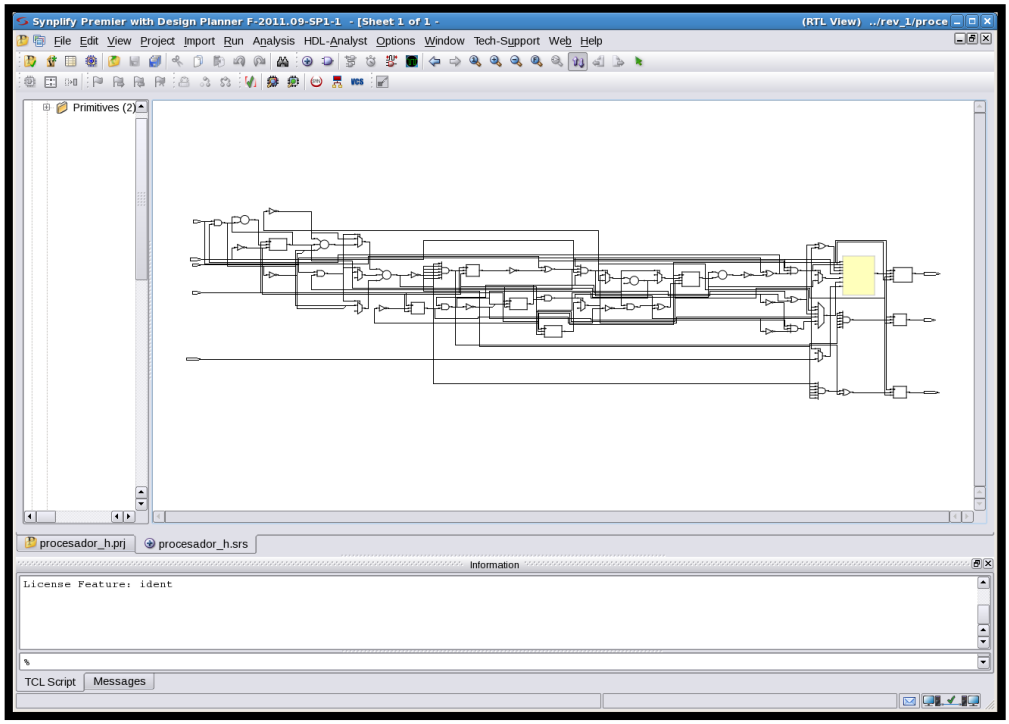


Figura 17. Vista RTL de Synplify Premier

La segunda vista es la tecnológica en la que cada bloque se representa como el conjunto de recursos sobre el que ha sido mapeado, ya sean estos registros, memorias de bloque RAM o LUTs. Esta vista se expone en la figura 18.

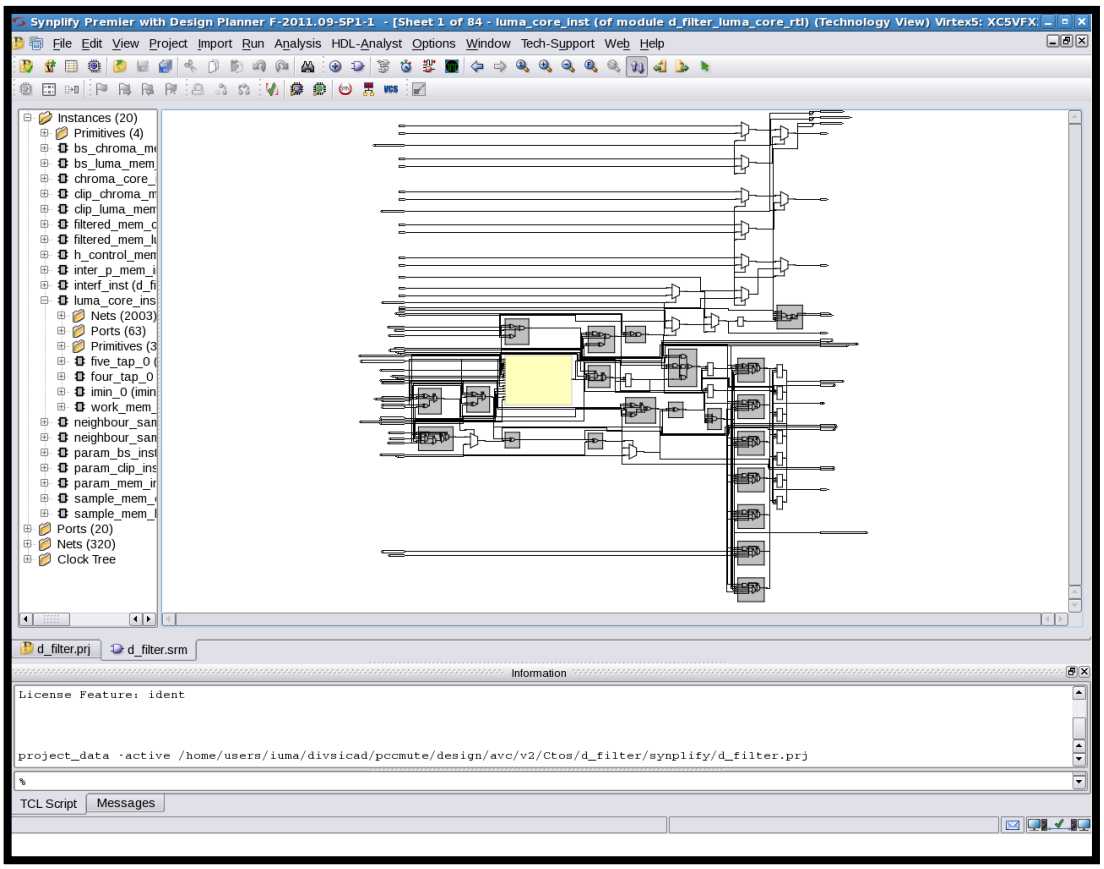


Figura 18. Vista tecnológica de Synplify Premier

3.4.3. PlanAhead

Xilinx PlanAhead es un entorno avanzado que facilita la realización de las tareas de síntesis, colocación y asignación de interconexiones de los recursos de la FPGA. Dispone de diferentes tipos de estrategias predefinidas (optimización global, temporal, optimización por bloques, etc.), permite la creación de estrategias definidas por el usuario a partir de la asignación de diferentes opciones de los programas de *back-end* integrados [36].

PlanAhead permite la importación de diseños en formato HDL, en cuyo caso requiere una fase previa de síntesis que realiza mediante la invocación de la herramienta XST, la herramienta de síntesis de Xilinx. Esta herramienta realiza la síntesis lógica que traduce la descripción HDL a una descripción estructural.

Además de la posibilidad de definir estrategias para los algoritmos de colocación o de ruteado, también pueden incluirse restricciones de área en la colocación. Se permite al diseñador definir áreas de la FPGA donde ubicar cada uno de los módulos del diseño, lo cual puede llevar a soluciones más eficientes.

Una vez concluida la fase de implementación, realiza un análisis temporal que comprobará si se cumplen todas las restricciones temporales, tanto impuestas por el usuario como las impuestas por la herramienta a través de la definición de reloj.

Permite además lanzar diferentes estrategias en paralelo en diferentes máquinas con el fin de paralelizar el proceso.

Una vez se obtiene un diseño completamente ruteado, permite llamar al generador del fichero de configuración de la FPGA (*bitstream*).

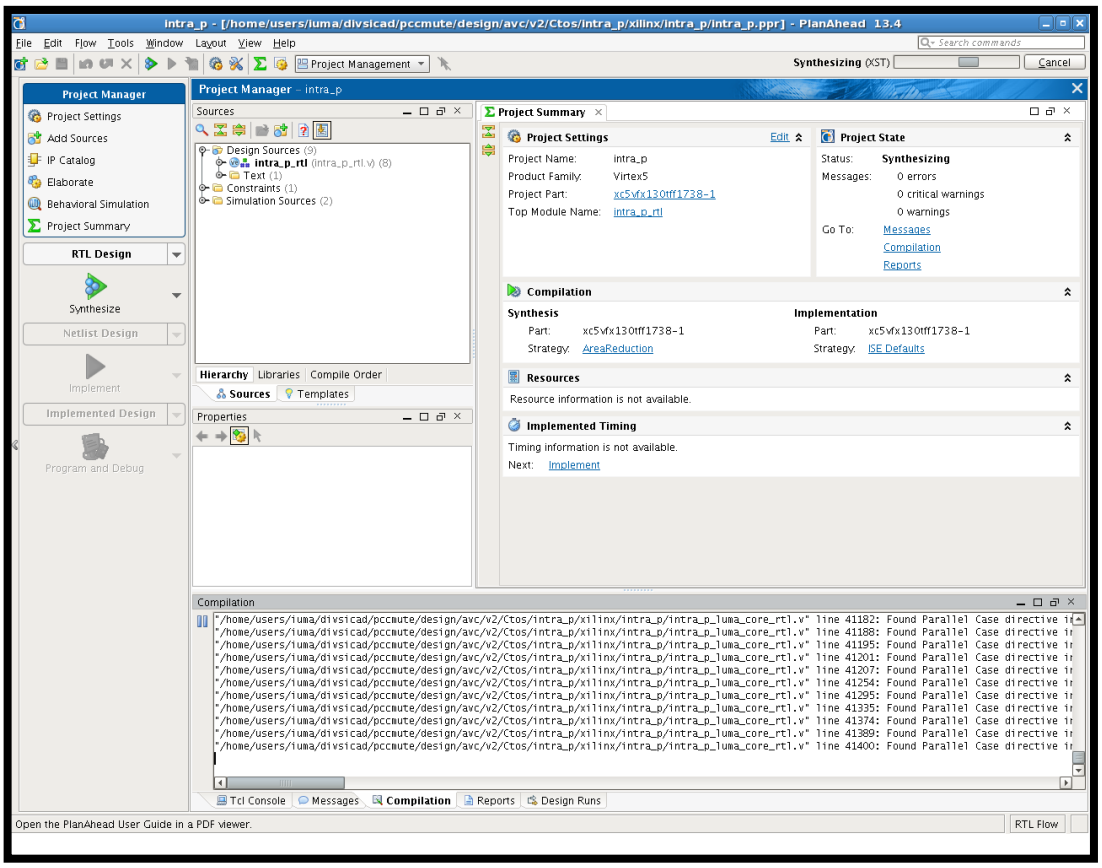


Figura 19. Interfaz gráfica de Xilinx PlanAhead

3.5. Tecnologías

En este apartado se presentan las tecnologías de dispositivos FPGA con las que se trabajará en este TFM.

3.5.1. Familia de FPGA Xilinx Virtex-5

La familia Virtex-5 de Xilinx usa la segunda generación de la arquitectura basada en columnas ASMBL™ (*Advanced Silicon Modular Block*), estando dividida en cinco sub-familias con el fin de cubrir todas las posibles necesidades de los diseños a implementar. Además de las células lógicas programables, las FPGAs de la familia Virtex-5 contienen una gran cantidad de bloques de tipo *Hard-IP* (bloques *hardware* no programables embebidos en la FPGA) como pueden ser, BlockRAMs y FIFOs de 36Kbit, DSPs 24 x 18, tecnología *SelectIO™* con impedancia controlada digitalmente (DCI), bloques de interfaz *ChipSync™ source-synchronous*, funcionalidad de monitorización del sistema, generadores de reloj interno mediante DCMs (*Digital Clock Manager*) y PLLs (*Phase-Locked-Loop*) y opciones avanzadas de configuración, incluyendo reconfiguración parcial. Otras funcionalidades de

la FPGA están soportadas por la presencia de bloques transceptores de alta velocidad y bajo consumo, bloques integrados de interfaz *PCI Express®*, MACs (*Media Access Controllers*) Ethernet tri-modo (10/100/1000 Mbps) y microprocesadores PowerPC® 440 de altas prestaciones. Está fabricada con tecnología de 65 nm [37].

3.5.1.1. Resumen de características de la familia Virtex-5

A continuación se listan los principales aspectos de la familia de FPGAs Virtex-5 de Xilinx. Se referencia al lector a la hoja de características para una descripción más completa [37].

- Cinco subfamilias:
  - Virtex-5 LX: Para aplicaciones de lógica general.
  - Virtex-5 LXT: Para aplicaciones de lógica general que requieran conectividad serie avanzada.
  - Virtex-5 SXT: Para aplicaciones de procesado de señales que requieran conectividad serie avanzada.
  - Virtex-5 TXT: Para sistemas que requieran conectividad serie avanzada de doble densidad.
  - Virtex-5 FXT: Para sistemas empotrados con conectividad serie avanzada.
- Compatibilidad entre plataformas:
  - Las familias LXT, SXT y FXT usan encapsulados compatibles, pudiéndose reutilizar diseños de PCBs modificando únicamente el voltaje.
- Estructura de lógica de altas prestaciones:
  - Tecnología de LUTs de 6 entradas.
  - Opción de LUTs de 5 entradas duales.
  - Rutas reducidas mejoradas.
  - Opción de RAM distribuida de 64 bits.
  - LUTs con registros de desplazamiento de 32 bits / opción dual de LUTs con registros de desplazamiento de 16 bits
- Gestión de reloj CMT:
  - Bloques DCM para retardo de *buffer* nulo, síntesis de frecuencia y desfase de reloj.
  - Bloques PLL para filtrado de *jitter*, retardo de *buffer* nulo, síntesis de frecuencia y división de reloj enganchado en fase.
- Bloques FIFO y RAM integrada de 36 Kbit:
  - Bloques de RAM de dos puertos.
  - Lógica de FIFO programable.
  - Programable:
    - Anchos de x36 para memorias de dos puertos.
    - Anchos de x72 para memorias de un puerto.

  - Circuitería interna opcional de corrección de errores.
  - Opcionalmente se puede programar cada bloque como dos bloques independientes de 18 Kbit.
- Tecnología *SelectIO™*:
  - Operaciones de entrada/salida con voltaje de 1,2 V a 3,3V.
  - Interfaz con tecnología *source-synchronous ChipSync™.*
  - Impedancia controlada digitalmente (DCI)
  - Bancos de E/S de granularidad fina.
  - Soporte para interfaz de memoria de alta velocidad.
- *Slices* DSP48E:
  - Multiplicadores 24 x 18 en complemento a dos.
  - Sumador, restador y acumulador opcional.
  - Segmentación opcional.
  - Funcionalidad de lógica a nivel de bits opcional.
  - Conectividad en cascada dedicada.
- Opciones de configuración flexibles:
  - Interfaces SPI y *Parallel* FLASH.
  - Soporte *multi-bitstream* con lógica de reconfiguración dedicada.
  - Detección automática de ancho de bus.
- Capacidad de monitorizar el sistema en todos los dispositivos:
  - Monitorización de temperatura on-chip/off-chip.
  - Monitorización de tensión de alimentación on-chip/off-chip.
  - Acceso por JTAG a todos los valores monitorizados.
- Bloques *Endpoint* integrados para diseños de PCI *Express®*.
  - Subfamilias LXT, SXT, TXT y FXT.
  - Acorde con la especificación base v1.1 de PCI *Express®*.
  - Soporte en cada bloque de pista x1, x4 o x8
  - Funcionalidad compartida conjuntamente con los transceptores *RocketIO™*.
- MACs Ethernet tri-modo 10/100/1000 Mbps.
  - Subfamilias LXT, SXT, TXT y FXT.
  - Los transceptores *RocketIO™* pueden ser usados como PHY o conectados a PHY externos usando las diferentes soluciones MII (*Media Independent Interface*).
- Transceptores *RocketIO™* GTP con tasas de 100 Mbps a 3,75 Gbps.
  - Subfamilias LXT y SXT.
- Transceptores *RocketIO™* GTX con tasas de 150 Mbps a 6,5 Gbps.

    - Subfamilias TXT y FXT.
- Microprocesadores PowerPC 440:
    - Solo la subfamilia FXT.
    - Arquitectura RISC.
    - Pipeline de 7 etapas.
    - Caches de instrucciones y de datos de 32 KB.
    - Estructura de interfaz de procesador optimizada.
- Tecnología de 65 nm de cobre.
- Voltaje de núcleo de 1,0 V.
- Encapsulados Flip-Chip estándar y opciones libres de plomo.

3.6. Flujo de diseño propuesto

A continuación, y a partir de los lenguajes, herramientas y tecnologías presentadas en los apartados anteriores, se concreta un flujo de diseño especificando las herramientas necesarias para cada etapa del mismo. En la descripción que sigue se ha hecho uso de la herramienta más adecuada para cada uno de los pasos del flujo de diseño. La elección de la herramienta forma parte de la experiencia del grupo de investigación donde se ha realizado el presente trabajo. Este flujo se presenta en la Figura 20.

En un primer paso se hará una verificación funcional del sistema. Esta verificación se ha realizado como parte del proyecto ARTEMI+ llevado a cabo por la división SICAD del IUMA y no forma parte de este TFM. Por tanto podemos decir que el trabajo parte de un modelo completo del decodificador H.264/AVC funcionalmente correcto, aunque adaptado a otros flujos de diseño.

Una vez comprobado que el sistema funciona según las especificaciones dadas, se continúa con la redefinición de la jerarquía, estableciendo un diseño compuesto por cinco módulos correspondientes a: IQIT, INTER_P, INTRA_P, CAVLD y D_FILTER.

El siguiente paso será la adaptación de cada uno de estos módulos al flujo de diseño de CtoS, teniendo en cuenta detalles tales como la inicialización de puertos de salida y señales, reasignación de señales de reloj y *reset*, etc. Esta etapa se explicará con más detalle en el Capítulo 4.

Cuando los módulos estén adaptados al flujo de diseño del Cadence CtoS, se pasa a la síntesis de alto nivel y, una vez generado el modelo RTL, a la síntesis lógica con Synplify.

Con los datos que se obtengan de la síntesis lógica, se procederá a la caracterización del diseño, lo que permitirá hacer una presentación de dichos resultados, y, en un capítulo posterior, realizar comparativas apropiadas con trabajos técnicos similares.

Figura 20. Flujo de diseño propuesto

3.7. Conclusiones

En el presente capítulo se ha presentado el lenguaje, la tecnología, herramientas y metodología y técnicas fundamentales para la correcta realización de este Trabajo Fin de Máster.

En un primer apartado, se ha presentado SystemC, el lenguaje de descripción *hardware* de alto nivel usado en este desarrollo.

Posteriormente, se han descrito las herramientas a las que se hará alusión durante la descripción del desarrollo del trabajo, explicando sus objetivos y a qué etapa del flujo pertenecen.

Por último, se ha visto la tecnología de implementación, que para el Trabajo Fin de Máster corresponde a la familia de FPGAs Virtex-5 de Xilinx. Se ha destacado de esta, aquellas propiedades que las hacen de interés para el desarrollo de este trabajo.

Para terminar, se ha presentando el flujo de diseño propuesto, describiendo brevemente cada una de las etapas que lo componen.

Capítulo 4: Desarrollo

4.1. Introducción

En este capítulo se describirán las etapas de síntesis de alto nivel y lógica a partir del modelo RTL generado con la herramienta Cadence CtoS. Se pretende definir el flujo de diseño partiendo de una especificación funcional en SystemC, obtener un modelo sintetizado y optimizado, con el que se pueda posteriormente realizar comparativas apropiadas con trabajos técnicos similares

Se parte, como punto de partida, de un diseño SystemC que describe un decodificador de vídeo para el estándar H.264/AVC correctamente compilado y libre de errores.

Se intenta en este capítulo definir directrices sobre los pasos a seguir para la síntesis tanto en alto nivel como lógica, de un sistema *hardware* complejo compuesto por un alto número de bloques funcionales.

4.2. Verificación funcional

Antes de iniciar los pasos de síntesis que llevarán a la obtención de un modelo a nivel RTL del sistema y su posterior síntesis lógica, se realizó una verificación funcional del mismo para comprobar su correcto funcionamiento. Una vez comprobado este paso, con resultados satisfactorios, se pudo pasar al siguiente paso del flujo de diseño propuesto.

4.3. Definición de la jerarquía por módulos

Como se explicó en el capítulo 2, para este TFM se decidió trabajar con una jerarquía por módulos, agrupando los principales bloques funcionales del decodificador H.264/AVC de la siguiente manera:

- Bloque CAVLD: el bloque denominado CAVLD engloba tanto el decodificador CAVLC, como todos los módulos que participan en la fase de control del flujo de datos entrante.
- Bloque IQIT: el bloque IQIT se encarga de la decodificación de la información residual proveniente del decodificador CAVLC, tal como se mostró en la arquitectura del sistema.
- Bloque INTRA_P: el bloque INTRA_P comprende los módulos necesarios para el modo de predicción *intra*. Esto incluye la segmentación del flujo de predicción, la preparación de los registros para el almacenamiento de información y el propio procesado de los elementos.
- Bloque INTER_P: al igual que el bloque INTRA_P, el bloque INTER_P contiene los módulos pertenecientes al flujo de procesado de la predicción tipo *inter*. Estas operaciones se efectúan a partir de los datos de entrada enviados por el bloque CAVLD, como se mostró en la arquitectura del sistema.
- Bloque D_FILTER: El bloque D_FILTER incluye a todos los módulos pertenecientes a las funciones del filtro de suavizado de bordes.

De este modo, se realizó una síntesis de alto nivel y lógica para cada uno de estos bloques por separado, lo que permite extraer datos para cada uno de ellos y poder realizar posteriormente las comparativas adecuadas entre ellos y entre otros trabajos similares.

4.4. Adaptación del sistema al flujo de diseño de CtoS

En este paso se adaptó el código de referencia del diseño al flujo de diseño de la herramienta Cadence CtoS. Para ello se llevaron a cabo principalmente 3 pasos fundamentales:

- Inicialización de los puertos de salida y señales para cada uno de los módulos: paso necesario para el posterior correcto funcionamiento del diseño, ya que los puertos de salida no contendrán información desconocida que pueda propagarse y acarrear errores de funcionalidad.
- Creación de las definiciones necesarias para la síntesis en Cadence CtoS.
- Inclusión de la señal de *reset* en todos los módulos que componen el diseño.

4.5. Síntesis de alto nivel con Cadence CtoS

Una vez adaptado el diseño al flujo de la herramienta de síntesis de alto nivel, se pasa a realizar la misma teniendo en cuenta las siguientes las consideraciones que se exponen en los siguientes apartados.

4.5.1. Criterios de optimización

En general, a la hora de realizar la transformación de una descripción funcional a una arquitectura, existen tres factores a optimizar: ciclos de latencia, periodo de reloj y uso de recursos. En general, estos factores dependen entre sí, de tal forma que disminuir el tiempo de ciclo de reloj puede ir asociado a un incremento en el uso de recursos o a un mayor número de ciclos de latencia. Esta dependencia entre los factores hace imposible optimizar todos conjuntamente, siendo necesario establecer un compromiso en función de los requisitos del sistema.

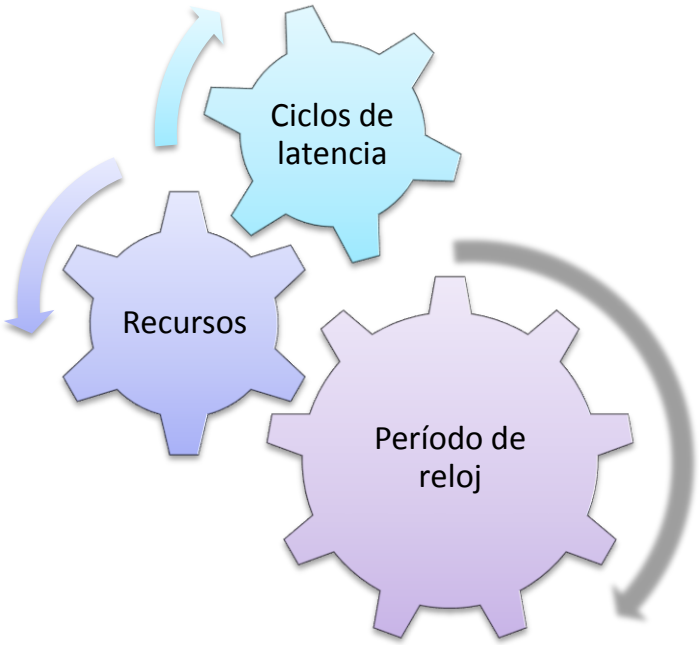


Figura 21. Factores de optimización

4.5.2. Estrategia de síntesis

A la hora de realizar la síntesis, existen principalmente dos estrategias a valorar: *top-down* y *bottom-up*. Ambas se basan en la partición del sistema en módulos más simples y ligeros, que al interconectarlos consiguen la funcionalidad del módulo principal. En general, al módulo que integra e interconecta al resto se le conoce como "*top* del diseño".

La estrategia *top-down* consiste en crear un proyecto único para la síntesis, que tenga como módulo principal el *top*, el cual instancia el resto de bloques, produciendo que la propia herramienta los sintetice.

Por el contrario, una estrategia *bottom-up* se basa en la síntesis por separado de cada bloque integrado en el sistema para finalmente integrarlos todos mediante un módulo *top* dedicado

únicamente a definir la conectividad entre ellos. Es necesario también tener en cuenta los efectos en los bordes de los módulos para facilitar la interconexión.

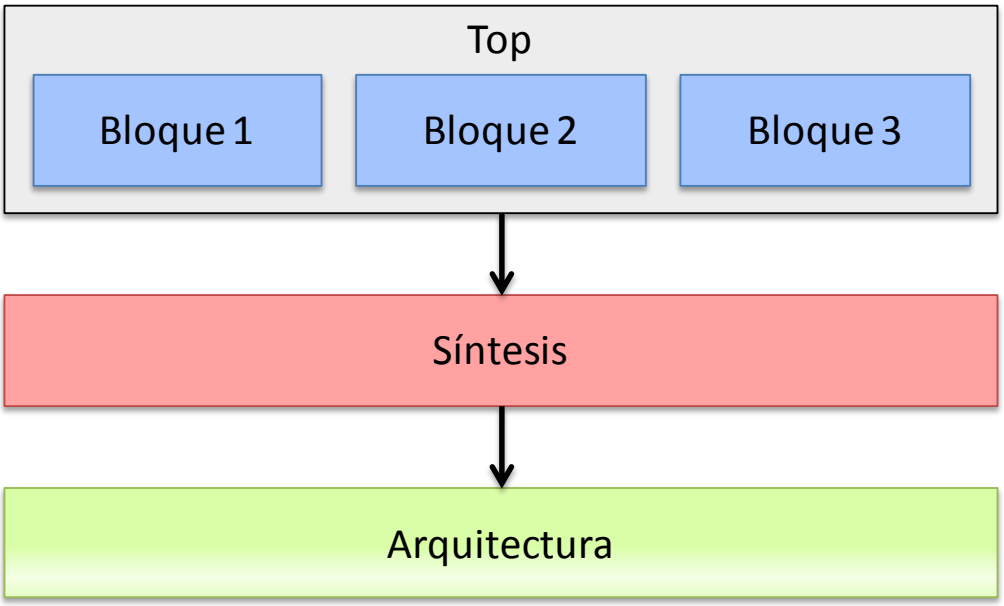


Figura 22. Estrategia *top-down*

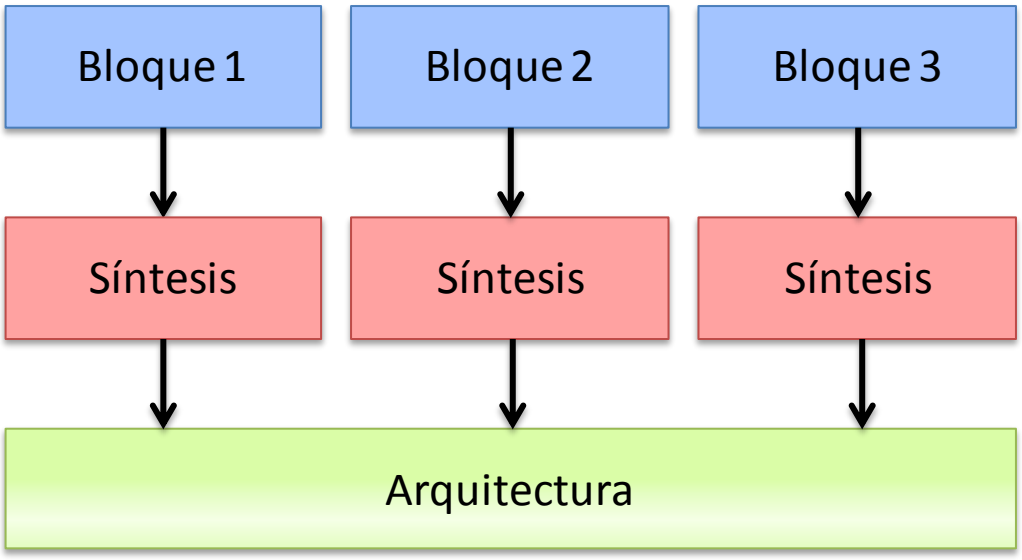


Figura 23. Estrategia *bottom-up*

En el presente trabajo se han tenido en cuenta dos consideraciones:

Por un lado, se ha seguido una estrategia *bottom-up* en cuanto a la división del decodificador completo en 5 módulos que agrupen las funciones principales. Esto se ha hecho en parte debido a las ventajas que ofrece y que se enumeran a continuación.

- **Paralelización de la síntesis**. Gracias a la creación de proyectos independientes para cada módulo, y aprovechando los servidores de cómputo, pueden lanzarse simultáneamente la síntesis de un alto número de módulos, minimizando el tiempo de síntesis.

- **Menor tiempo** de **depurado**. Gracias a la separación de la síntesis de cada módulo, durante la etapa de depurado en la verificación RTL, pueden realizarse modificaciones en un único módulo y conseguir una nueva versión del diseño sin tener que sintetizar el resto del diseño.

- **Obtención de resultados** para **cada módulo**. Dado el objetivo propuesto inicialmente de establecer comparativas relevantes entre los recursos, frecuencia, etc. de cada módulo, y entre otros trabajos, resulta más conveniente seguir esta estrategia ya que permitirá obtener resultados por separado.

Por otro lado, una vez hecha esta división por módulos principales, cada uno de ellos tiene a su vez un módulo *top* que engloba los diferentes submódulos que lo compone. En este punto, se ha seguido una estrategia de síntesis *top-down* para cada uno de ellos.

4.5.3. Síntesis de alto nivel

Para la síntesis de alto nivel, y como se explicó en el capítulo anterior, se ha optado por hacer uso de la herramienta C-to-Silicon (CtoS) de Cadence, la cual ofrece la posibilidad de realizar la síntesis partiendo de un diseño en SystemC, y obteniendo como resultado la descripción RTL del sistema en Verilog.

Como ya se adelantó en el capítulo de metodología, las herramientas de síntesis requieren de una guía que les proporcione cierta información sobre el objetivo del sistema, pues existen muchas posibles soluciones que cumplen con los requisitos de funcionalidad. Es por ello, que durante esta fase CtoS exige que se especifiquen ciertos parámetros, que se comentarán en el siguiente apartado.

4.5.3.1. Parámetros de la síntesis

A continuación se enumerarán aquellos parámetros que deben especificarse durante la etapa de síntesis de alto nivel para la herramienta CtoS. Además se indicará la sintaxis para realizar esta especificación en la consola del programa o, lo que es más importante, para un script TCL (*Tool Command Language*) para futuras ejecuciones.

1. Frecuencia de reloj

Al comienzo de la etapa de síntesis es necesario indicar la frecuencia de reloj, así como su ciclo de trabajo (los valores se dan en términos de periodo de reloj y *offset* de flanco de bajada). Esta restricción guiará a la herramienta de síntesis para añadir más o menos ciclos de latencia así como a replicar o no lógica con el fin de alcanzar el objetivo.

```
define_clock -name nombrereloj -period periodo -rise 0 -fall flancobajada
```

Los valores de periodo, flanco de subida y flanco de bajada se indicarán en picosegundos, como a continuación se muestra en el ejemplo para una señal de reloj llamada *clock* que tiene un periodo de 10ns (frecuencia de reloj de 100MHz) y un ciclo de trabajo del 50%.

```
define_clock -name clock -period 10000 -rise 0 -fall 5000
```

2. Bucles combinacionales

En el diseño en SystemC pueden existir, y de hecho son simulables, bucles enteramente combinacionales que, según su definición, deberían ejecutarse en su totalidad en un único ciclo de reloj. Evidentemente, esto no suele ser el objetivo del bucle, aunque puede traducirse. Es por ello que ha de indicarse que hacer en estos casos. Puede optarse por varias soluciones:

- **Desenrollar el bucle**. Realiza todas las iteraciones una detrás de la otra en un único ciclo, a costa de replicar la lógica interna del bucle una vez por cada iteración. Favorece una minimización del número de ciclos, aumentando los recursos *hardware* necesarios, así como el periodo de reloj.

- **Romper el bucle.** Es el equivalente a añadir una sentencia `wait()` al final de cada iteración. Así, cada iteración se realiza en un ciclo de reloj. Es normalmente la acción por defecto.

- **Realizar un *pipeline*.** Rompe el bucle de forma que se tardan varios ciclos en realizarse, pero además, varias iteraciones se realizan a la vez, pero cada uno ocupando una etapa del bucle. Esta solución mejora el rendimiento para bucles con gran carga en cada iteración, pero es ineficiente para bucles sencillos al añadir la lógica adicional para el control del *pipeline*.

Para el presente trabajo se realizará una acción de rotura del bucle en todos aquellos bucles combinacionales existentes, para lo cual a continuación se muestra la sintaxis TCL.

```
set combo_loops [find_combinational_loops]
foreach loop $combo_loops {
    break_combinational_loop  $loop
}
```

3. Funciones no planificables

En la descripción del diseño en alto nivel se permiten las llamadas a funciones globales. En general el planificador resuelve este problema mediante la síntesis de un bloque independiente y un mecanismo de comunicación mediante un protocolo de petición y respuesta. Sin embargo, pueden darse ocasiones en las que esto no ocurre por diversas causas, como puede ser una latencia no constante o la llamada a la función desde distintos procesos concurrentes.

En estos casos, la herramienta obliga a realizar una acción de *inline*, es decir, modificar la llamada a la subrutina por una copia íntegra del código asociado a dicha función. Esta acción genera un aumento en el consumo de recursos. Ello requiere una modificación del código fuente en algunos casos, pero en otros es inevitable.

También se permite realizar un *inline* de las funciones que no producen estos conflictos. Sin embargo, llevar esta acción sobre todas las funciones del diseño suele conducir a unos costes de recursos demasiado elevados para luego poder ser implementado en un dispositivo físico. A continuación se muestra la sintaxis para realizar *inline* de una función específica.

```
inline /designs/nombre_proyecto/modules/nombre_modulo/behaviors/funcion1
```

4. Asignación de memorias

Si existen vectores de datos en la descripción SystemC del diseño, se debe decidir los recursos a utilizar para almacenarlos. Existen diferentes opciones, que se enumeran a continuación:

- **RAM interna**. Se usa como recursos los bloques BRAM o las LUTs de los SLICEM para almacenar la información. El uso de uno u otro depende de los ciclos de espera introducidos entre la disposición de la dirección y la lectura del dato.
- **Registros**. Todos los datos se almacenan en registros. Esta solución solo debe realizarse en casos de vectores de pocos elementos y en los cuales los elementos no son excesivamente grandes en términos de longitud de bits.
- **RAM de terceros**. Pueden imponerse como recurso de memoria la descripción HDL de una RAM de terceros.

El uso de registros o BRAM/LUTs dependerá de la disponibilidad de recursos del dispositivo y del uso que haga el resto del diseño.

5. Uso de DSPs

Para cada módulo puede indicarse si debe o no usar DSPs. En general, y dependiendo de la naturaleza del diseño, puede ser interesante que un módulo haga uso de los DSPs para aquellas operaciones para los que han sido diseñados, y otros, menos importantes, usen lógica programable para realizarlas a costa de un mayor número de ciclos de latencia.

```
use_dsp /designs/$modulo
```

6. Relajación de la latencia

Esta opción permite a CtoS incrementar el número de ciclos de latencia de una función, con el fin de facilitar la planificación de la misma. Es importante no permitir esta acción sobre aquellas funciones dedicadas a implementar los protocolos de comunicación. Por ejemplo, no es conveniente permitir relajar la latencia de las funciones que hacen accesos a memoria, o que se dedican a leer y escribir sobre los puertos de control de comunicación, ya que añadir estados en medio de una de estas rutinas puede significar que el protocolo deje de ser correcto. A continuación se muestra como desactivar la relajación para todas las funciones del diseño.

```
set behaviours [find $top_path/modules/*/behaviors/*]
foreach beh $behaviours {
```

```
    set_attr relax_latency "false" $beh
}
```

4.5.3.2. Automatización de la síntesis

Como se adelantó con anterioridad, CtoS permite lanzarse en modo *batch*, haciendo uso de *scripts* TCL. De esta forma, realizar la síntesis durante la etapa de depuración es mucho más rápido, sin la necesidad de elegir todos los parámetros anteriormente descritos en cada síntesis.

Además, gracias a la estrategia elegida *bottom-up* pueden lanzarse paralelamente varios procesos de síntesis lanzando la aplicación en modo *batch* y con el *script* previamente escrito. Para lanzar la herramienta en modo *batch*, se especificará el alias de la aplicación en dicho modo, el *script* a ejecutar y, opcionalmente, un fichero de *log* donde almacenar la salida del proceso.

```
ctos scripts/ctos.tcl -log log/ctos.log
```

En el fichero de *script* TCL existirán dos etapas diferenciadas. La primera es la que configura el proyecto de CtoS, indicando los ficheros fuente, la frecuencia de reloj, así como el dispositivo de prototipado final.

```
new_design $modulo
set_attr auto_write_models "true" /designs/$modulo
define_sim_config -model_dir "./model" /designs/$modulo
set_attr source_files [list src/$modulo.cpp] /designs/$modulo
set_attr compile_flags " -w -I../../include -I./src" /designs/$modulo
set_attr auto_save_dir $modulo /designs/$modulo
define_clock -name clock -period 10000 -rise 0 -fall 5000
set_attr implementation_target FPGA [get_design]
set_attr fpga_target [list Xilinx Virtex5 xc5vfx130t-1-ff1738] [get_design]
set_attr fpga_install_path [exec which xst] [get_design]
set_attr fpga_work_dir "./fpga_work" [get_design]
```

La segunda parte del *script* es la que define los diferentes pasos de la síntesis previamente citados en este apartado. A continuación se muestra un ejemplo.

```
set combo_loops [find_combinational_loops]
foreach loop $combo_loops {
    break_combinational_loop  $loop
}
foreach beh [find $top_path/modules/*/behaviors/*] {
    set_attr relax_latency "true" $beh
}
schedule -effort high -passes 200 /designs/$modulo
allocate_registers
write_rtl -recursive -o [concat ./[string trim $model]] $top_path
/modules/$modulo
```

En la Figura 24 se muestra el diagrama de flujo del *script* creado.

Figura 24. Diagrama de flujo del *script* de automatización de síntesis

4.6. Síntesis lógica con Synplify

Una vez generado el modelo a nivel RTL del diseño, el siguiente paso para poder implementarlo es realizar la síntesis lógica, consistente en realizar una transformación desde un nivel RTL al nivel de puertas lógicas, en este caso primitivas de Xilinx interconectadas formando un *netlist* completo en formato EDIF.

Este paso de síntesis lógica dentro del flujo de diseño está soportado dentro del entorno de Xilinx por la herramienta XST (Xilinx Synthesis Tecnology) [38], que permite realizar la síntesis lógica a nivel RTL. Sin embargo, los resultados obtenidos por XST son, en general, peores a los de otros entornos especializados como es el caso de Synopsys Synplify Premier.

4.6.1. Estrategia de síntesis lógica

Al realizar la síntesis lógica, al igual que en el paso de síntesis de alto nivel, se puede optar por seguir dos estrategias: *bottom-up* o *top-down*. Para diseños con alta complejidad y elevados tiempos de síntesis puede optarse por seguir una estrategia *bottom-up*, estrategia soportada en Synopsys Synplify mediante un sistema de gestión de proyectos y subproyectos.

Para poner en práctica esta estrategia, es necesario crear un proyecto para cada módulo, un proyecto para el *top* y en este incluir como subproyectos, los proyectos anteriormente creados. De

esta forma, Synplify realizará la síntesis de cada módulo por separado, para luego realizar una optimización global de todo el diseño.

Alternativamente es posible seguir una estrategia *top-down*, en la que se crea un único proyecto que incluye todos los ficheros de descripción RTL de los módulos, y señalando el jerárquicamente superior como *top* del diseño. A partir de ahí, Synplify genera la estructura del diseño mediante las instancias realizadas en los distintos ficheros, sintetizando aquellos módulos que conformen el diseño completo.

4.6.1.1. Comparativa entre estrategias

Con el creciente incremento de capacidad de elementos lógicos programables en las FPGAs los tiempos de síntesis e implementación han ido creciendo en sintonía. Esto produce que, en muchos casos, sea inviable seguir estrategias clásicas de diseño *hardware* [6].

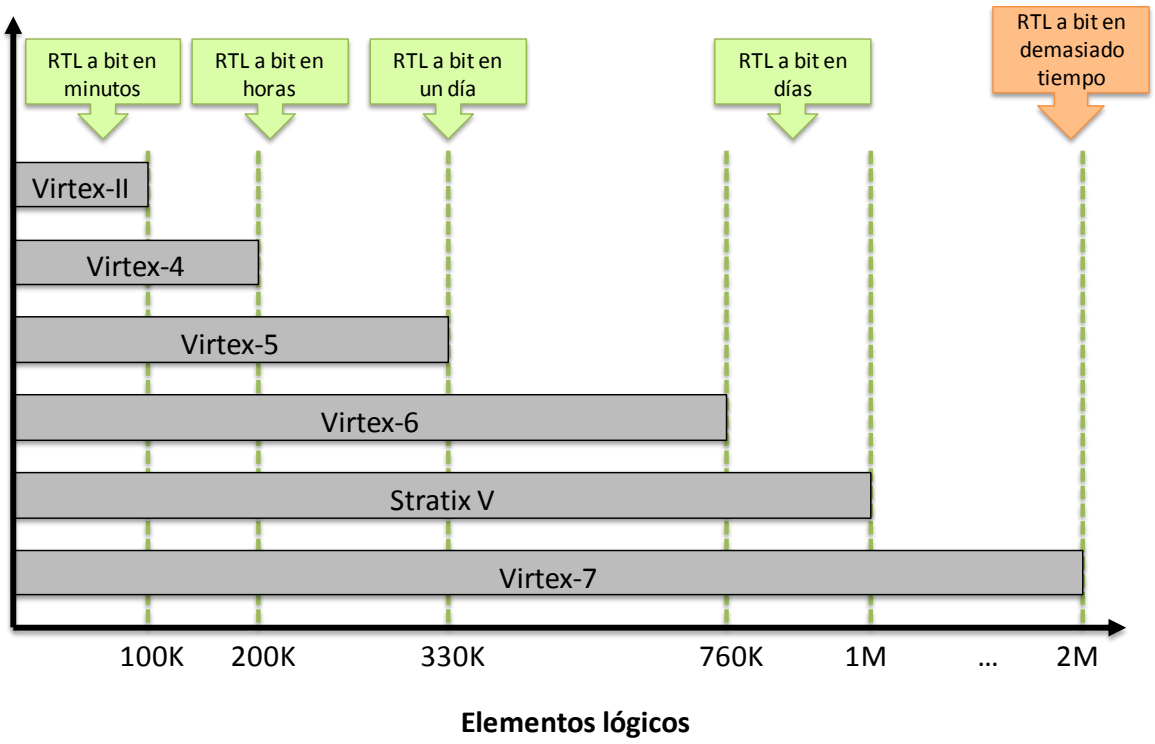


Figura 25. Tiempos de síntesis e implementación en familias de FPGA de Xilinx [6]

En casos donde el diseño es demasiado grande puede ocurrir que una estrategia *top-down* sea completamente inviable debido al tiempo necesario para realizar la síntesis o por los requerimientos de memoria que necesitaría la máquina donde se realizase. Por otro lado, en este tipo de diseños es normal que este sea partido en sub-módulos y desarrollados en paralelo, por lo que se precisan de modificaciones en el flujo de diseño para dar solución a estos requerimientos.

Una solución acertada para estos problemas ha sido la adopción de flujos *bottom-up* en el que cada sub-módulo es desarrollado mediante un flujo estándar de diseño, para luego ser integrado en un proyecto final con el fin de unirlos. Además, en la síntesis del sistema completo pueden lanzarse

las síntesis de varios de estos submódulos en paralelo, reduciendo así los tiempos de dicha etapa del flujo. Esta estrategia tiene por defecto un empeoramiento en la calidad de los resultados finales.

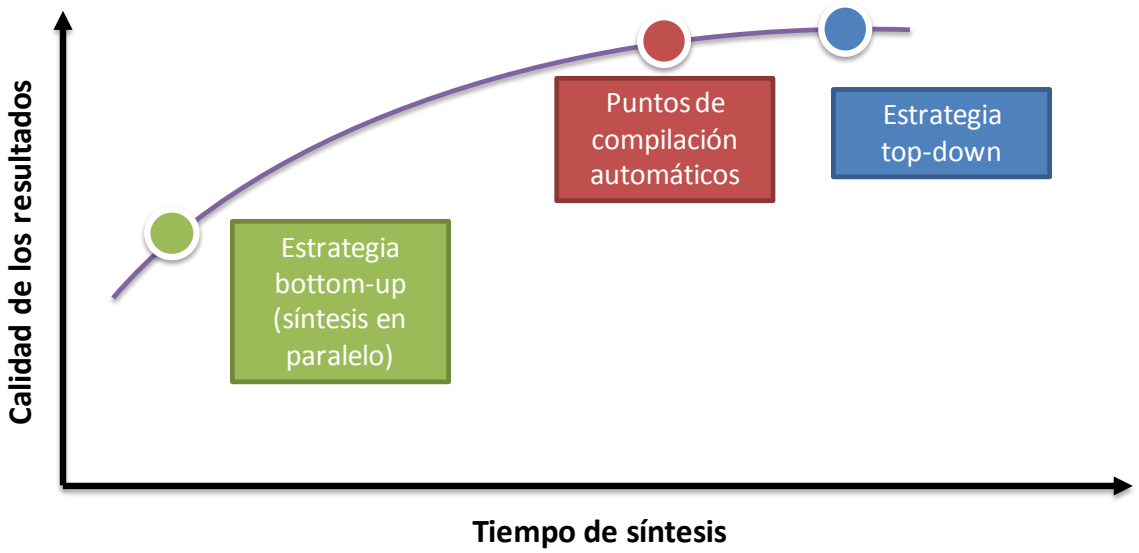


Figura 26. Calidad de los resultados frente a tiempos de síntesis.

En el caso de la herramienta de síntesis Synplify, existe además una estrategia de puntos de compilación automática con diseño incremental donde la propia herramienta crea particiones en el diseño de forma automática y, en donde es posible, mantiene los resultados de las síntesis anteriores de las particiones que no hayan sufrido cambios. Si además se activa el modo de multi-procesado cada partición o *Compile Point* es sintetizado en paralelo a los demás, reduciendo así los tiempos de síntesis.

En el caso de la estrategia *bottom-up* Synplify realiza una optimización global tras la síntesis de todos los sub-módulos del diseño, incrementando ligeramente el tiempo de síntesis pero consiguiendo una mejora notable en la calidad de los resultados, ahorrando recursos innecesarios en los bordes de los bloques.

En este trabajo se ha seguido, a partir de la partición del sistema completo en módulos principales anteriormente explicada, una estrategia *top-down* para cada uno de ellos.

4.6.1.2. Opciones de síntesis

A continuación se describen aquellas opciones a activar en las opciones de síntesis de la herramienta, con el fin de obtener los mejores resultados para el modelo de referencia con el que se trabaja.

Disable I/O Insertion

Consiste en impedir la inserción de bloques de entrada/salida (IOB) de la FPGA para las señales de entrada y salida del top del diseño. En el caso de este trabajo, se ha activado para no tener en

cuenta estos recursos a la hora de obtener resultados. Por tanto las E/S del IP estarán conectadas a otras señales internas de la plataforma.

Por defecto, las herramientas de síntesis dispondrán de IOBs para los puertos del top, entendiendo que dichas señales estarán asociadas a pines de la FPGA, con el fin de conectarlas a dispositivos externos.

FSM Compiler

Se trata de un optimizador de máquinas de estado. Synplify ofrece la posibilidad de optimizar la lógica de estado siguiente de las instancias de máquinas de estado del diseño que encuentre, siguiendo una estrategia diferente de codificación de estados en función del número de estos, según la siguiente tabla.

Número de estados	Tipo de codificación
< 5	Secuencial
5 – 24	One-Hot
> 24	Gray

Tabla 3. Codificación de estados de *FSM Compiler*

Resource Sharing

Permite compartir recursos con el fin de reducir el consumo del área del dispositivo de implementación, a costa de reducir la frecuencia.

Pipelining

Permite que varias operaciones se realicen a la vez sobre el mismo recurso, partiendo dicha ejecución en etapas, y permitiendo que cada dato se encuentre en una de ellas. En las tecnologías Virtex-5 de Xilinx, esta optimización va asociada a las memorias ROMs y a los multiplicadores del diseño.

Enable Advanced LUT Combining

Prepara el fichero *netlist* de salida para una posterior combinación de LUTs en los diseños sobre FPGAs de Xilinx.

Una vez terminado el proceso de configuración de la herramienta, se ejecuta la síntesis lógica que dará los resultados que se presentan en el siguiente capítulo.

4.7. Síntesis lógica con XST en PlanAhead

A pesar de que los resultados obtenidos utilizando la herramienta Synopsys Synplify suelen ser mejores que los obtenidos con la herramienta de síntesis XST integrada en Xilinx PlanAhead, se

considera de interés para completar este trabajo de investigación el hacer una síntesis lógica de los diferentes módulos utilizando dicha herramienta.

4.7.1. Estrategia de síntesis

PlanAhead permite establecer diversas estrategias de síntesis, dando la posibilidad de escoger entre las que ofrece la propia herramienta o crear estrategias propias. Para ello, hace uso de varios parámetros que permiten definir exactamente qué tipo de estrategia de síntesis seguirá la herramienta.

a. *Optimization mode:* especifica el objetivo global de optimización, pudiendo elegirse entre área (*area)* o velocidad (*speed*).

b. *Optimization effort level*: define el esfuerzo de optimización entre tres valores de 0 a 2. Cuanto más alto sea este valor mayor esfuerzo hará la herramienta para la optimización de utilización de recursos repercutiendo negativamente en la frecuencia máxima de funcionamiento.

c. *Register balancing:* permite la utilización de técnicas de *retiming* en los flip-flops.

d. *FSM encoding algorithm*: especifica un algoritmo de codificación para las máquinas de estados finitos que encuentre en el diseño.

e. *LUT combining*: permite la compartición de pares de LUTs con entradas comunes.

f. *Automatic BRAM packing*: posibilita el empaquetado de dos módulos BRAM en un único modo BRAM con doble puerto.

g. *Use DSP Block*: especifica la posibilidad o no de uso de las DSPs disponibles.

h. *Resource sharing*: permite la compartición de recursos destinados a operaciones aritméticas.

i. *Pack registers into IOBs*: posibilita el empaquetado de registros dentro de IOBs.

j. *Power reduction*: permite una optimización del diseño para la reducción del consumo de potencia.

k. *RAM style*: especifica el estilo de las RAM inferenciadas.

En este TFM la estrategia de síntesis utilizada definida por la propia herramienta ha sido la siguiente:

- Opción de síntesis*: AreaReduction* (XST 13).

Esta estrategia intenta optimizar el área frente a otros parámetros permitiendo la utilización de BRAM para integrar las diferentes memorias, el uso de DSPs si se considera necesario, así como la posibilidad de compartición de recursos para optimizar su utilización y reducir el área utilizada.

4.8. Conclusiones

En el capítulo 4 se explica el desarrollo seguido para llevar a cabo este Trabajo Fin Máster, detallando los pasos definidos en el flujo de diseño hasta la consecución de los resultados que se muestran en el capítulo siguiente.

Se ha partido de una verificación funcional del diseño, etapa llevada a cabo como parte del proyecto de investigación ARTEMI+ desarrollado por el grupo SICAD del IUMA.

Una vez realizada se ha definido la jerarquía a utilizar, se ha preparado el diseño para el flujo de diseño de la herramienta Cadence CtoS, para finalmente pasar a la síntesis de alto nivel y lógica.

Capítulo 5: Resultados

5.1. Introducción

En este apartado se presentan los resultados de síntesis, en términos de consumo de recursos y frecuencia máxima de utilización, para las estrategias de síntesis expuestas en los capítulos anteriores para la FPGA Virtex-5 FX130T de Xilinx integrada en la placa ML510. Una vez presentados estos resultados se pasará, en el capítulo 6, a la comparativa entre este y otros trabajos y publicaciones técnicas similares.

5.2. Resultados de la síntesis en alto nivel con Cadence CtoS

La herramienta Cadence CtoS realiza una primera estimación de los recursos utilizados durante la síntesis de alto nivel. Necesariamente, se trata de estimaciones realizadas con objeto de obtener información preliminar para realizar las operaciones de planificación y asignación de recursos, sin la información detallada de retardos debidos a las interconexiones, fan-out etc. Por ello el nivel de precisión de los resultados será menos si los comparamos con los obtenidos tras realizar la síntesis lógica. Sin embargo, se considera de interés el mostrarlos ya que, por un lado dan una idea de la

tendencia de consumo de recursos de los diferentes módulos integrados en el diseño, y por otra parte muestran la significativa diferencia de estimación de consumo de recursos que se va obteniendo a medida que se baja de nivel de diseño. En la tabla 4 se muestran los resultados de utilización de LUTs y DSPs obtenidos por la herramienta para una frecuencia de funcionamiento nominal de 100MHz.

	LUTs	DSP48s
D_FILTER	80.283,2	0
INTER_P	116.540,0	0
INTRA_P	79.055,6	0
IQIT	123.056,3	0
CAVLD	127.899,8	0

Tabla 4. Estimación de utilización de recursos obtenida con Cadence CtoS

Estos resultados se muestran en la figura 27. Se han obviado en esta gráfica los resultados de uso de DSPs dado que en todos casos la herramienta ha estimado que no será necesaria su utilización.

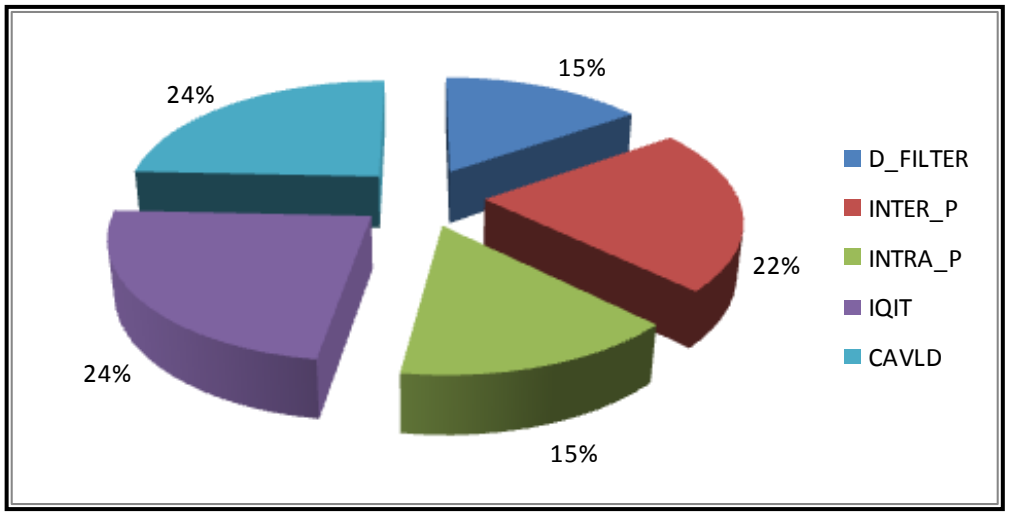


Figura 27. Porcentaje de utilización de LUTs obtenido en Cadence CtoS

Como puede observarse, tras la síntesis de alto nivel en CtoS el bloque CAVLD es el que mayor consumo de LUTs presentará, y el bloque D_FILTER el que menos.

5.3. Resultados de la síntesis lógica con XST

El siguiente paso a realizar en el flujo de diseño es la síntesis lógica a partir del modelo RTL generado, en este caso en lenguaje Verilog con Cadence CtoS. En este trabajo se ha realizado este paso con dos herramientas con el objeto de realizar una comparación de la calidad de los resultados obtenidos. Por una parte se ha utilizado Xilinx XST, herramienta de síntesis integrada en el entorno de PlanAhead and Vivado Synthesis. Como se expuso en el capítulo 4, se ha utilizado una estrategia de síntesis de optimización del área, buscando disminuir el consumo de recursos de la FPGA, dado

que, aun utilizando esta estrategia, la frecuencia de funcionamiento no se verá prácticamente afectada. Los resultados de la síntesis lógica con XST se muestran en la tabla 5.

	Frecuencia estimada (MHz)	Registros / %	BRAMs	LUTs / %	DSP48s
D_FILTER	109,082	33.993 / 41	0	22.534 / 28	0
INTER_P	108,357	66.313 / 81	0	40.343 / 49	0
INTRA_P	108,695	27.550 / 34	0	26.207 / 32	2
IQIT	97,809	62.466 / 76	0	38.681 / 47	0
CAVLD	128,688	44.542 / 54	1	23.804 / 29	0
	TOTAL:	**234.864 / 285**	**1**	**151.569 / 184**	**2**

Tabla 5. Utilización de recursos obtenido con XST

En la figura 28 se muestra la frecuencia máxima estimada por bloque. Se observa como el bloque de control CAVLD es el que presenta una mayor frecuencia de funcionamiento, siendo el bloque IQIT el bloque limitador de la frecuencia final.

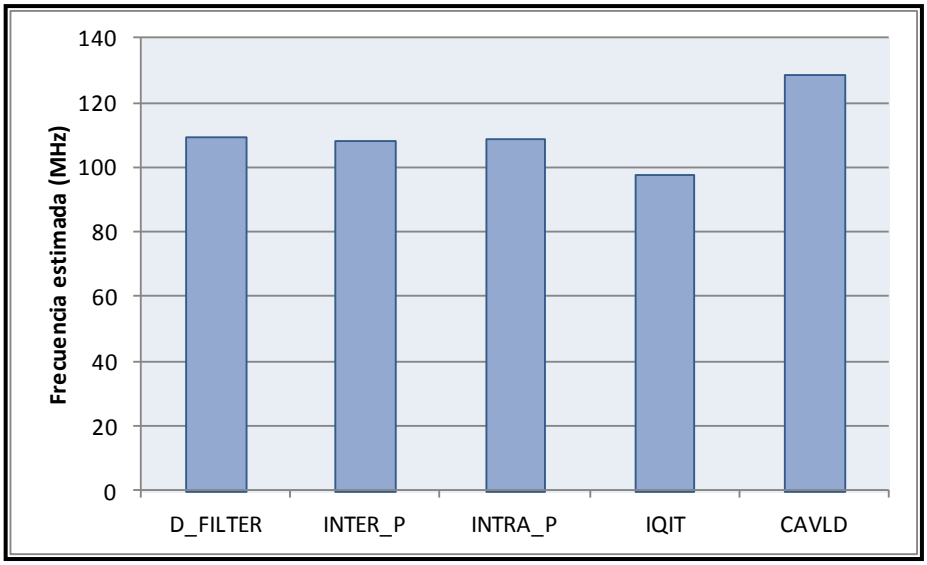


Figura 28. Frecuencia máxima por bloque estimada por XST

En la figura 29, se muestra la utilización de registros de la FPGA (implementados usando los F/F disponibles en los *slices*) para los diferentes bloques del diseño. En este caso, es el bloque INTER_P el que presenta una mayor utilización, siendo el que menor el bloque INTRA_P.

En la figura 30 se presenta la utilización de LUTs de cada bloque principal del diseño. Se puede observar que es el bloque INTER_P el que mayor uso hace de este recurso frente a los bloques CAVLD y D_FILTER que presentan una menor utilización de LUTs.

Para finalizar, la figura 31 presenta la utilización de BRAMs y DSPs, usándose un elemento de cada tipo en los bloques CAVLD e INTRA_P respectivamente.

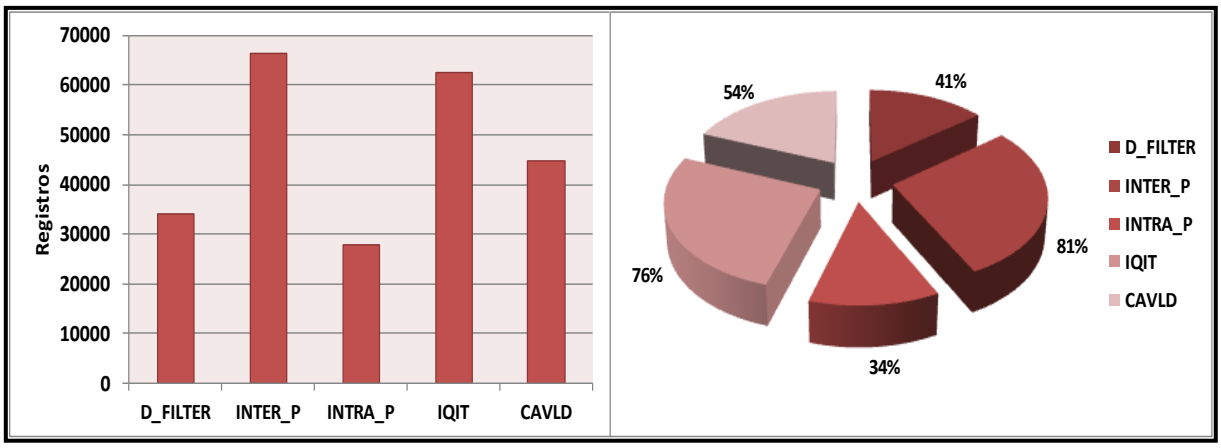


Figura 29. Utilización de registros obtenida en XST

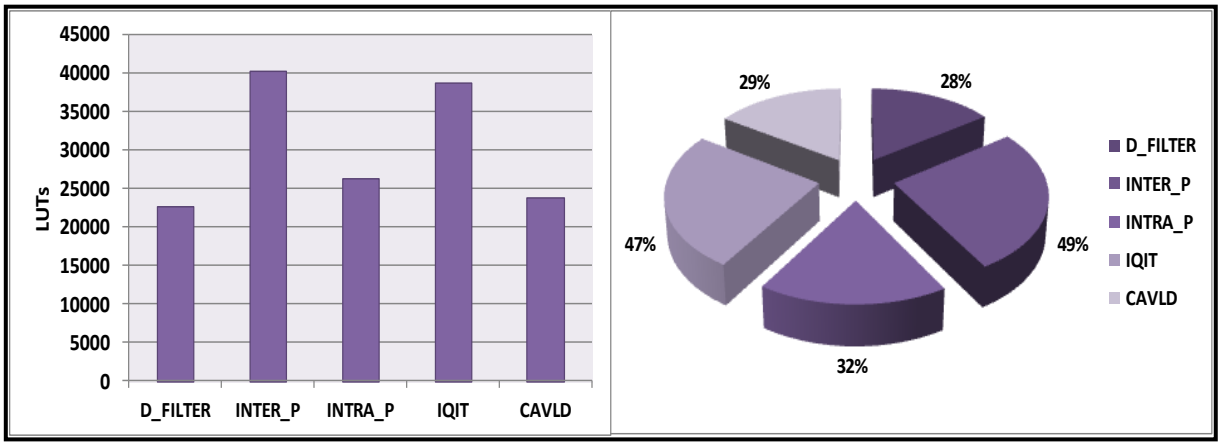


Figura 30. Utilización de LUTs obtenida en XST

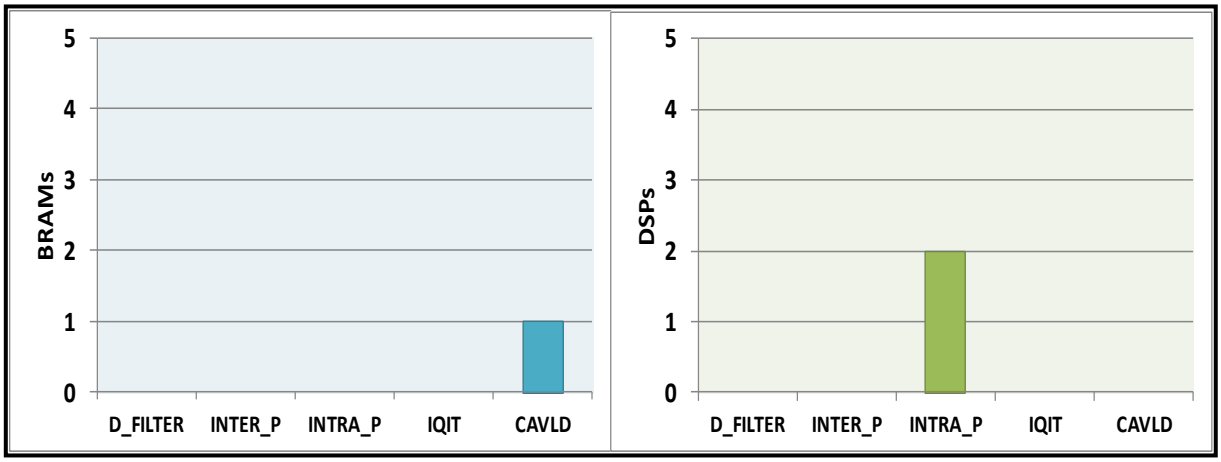


Figura 31. Utilización de BRAMs y DSPs obtenida en XST

5.4. Resultados de la síntesis lógica con Synplify

Como segunda alternativa, se ha utilizado Synopsys Synplify como herramienta de síntesis lógica. Como se puede observar en las diferentes tablas y figuras siguientes, los resultados obtenidos

mejoran tanto la frecuencia de funcionamiento y la ocupación de recursos en la FPGA. Dichos resultados pueden observarse en la tabla 6.

	Frecuencia estimada (MHz)	Registros / %	BRAMs	LUTs / %	DSP48s
D_FILTER	112,2	9.882 / 11	3	9.550 / 10	0
INTER_P	123,1	13.249 / 14	5	12.711 / 14	25
INTRA_P	122,3	9.940 / 11	1	20.492 / 22	3
IQIT	120,1	7.414 / 8	0	17.454 / 19	5
CAVLD	149,6	7.750 / 8	0	7.374 / 8	0
	TOTAL:	**48.235 / 52**	**9**	**67.581 / 73**	**33**

Tabla 6. Utilización de recursos obtenido con Synplify Premier

La figura 32 muestra la frecuencia máxima obtenida para cada uno de los bloques. El bloque de control CAVLD es el que presenta una mayor frecuencia de funcionamiento, siendo el bloque D_FILTER el que limitaría en este caso la frecuencia final del diseño. Otro aspecto destacable es el aumento de la frecuencia de funcionamiento de todos los bloques del diseño en un promedio de un 12%.

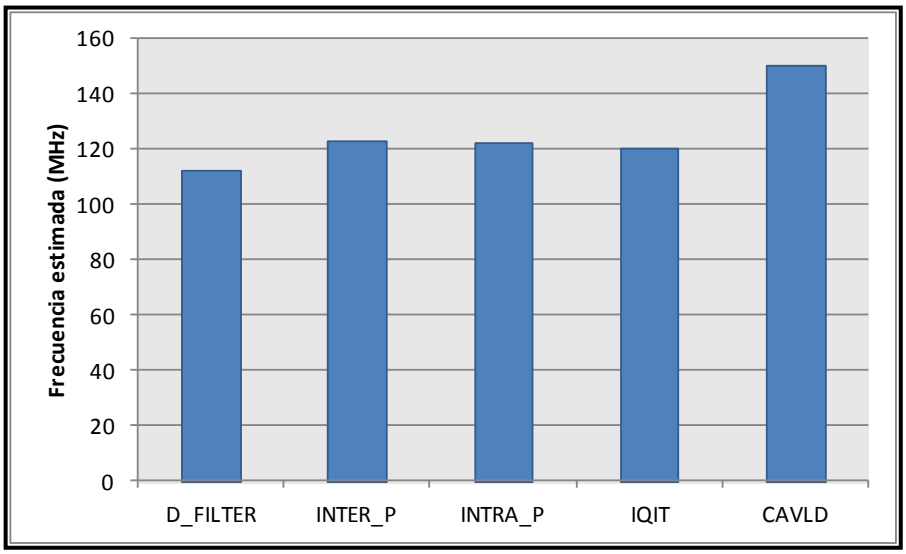


Figura 32. Frecuencia máxima por bloque estimada por Synplify

En la figura 33 puede observarse la utilización de registros para los diferentes bloques del diseño. En este caso es el bloque INTER_P el que tendría mayor utilización de este recurso, siendo el bloque IQIT el que menor uso haría de los mismos. En el caso de Synplify Premier, se produce una disminución drástica mediante simplificación del uso de registros con respecto a XST del orden del 400%.

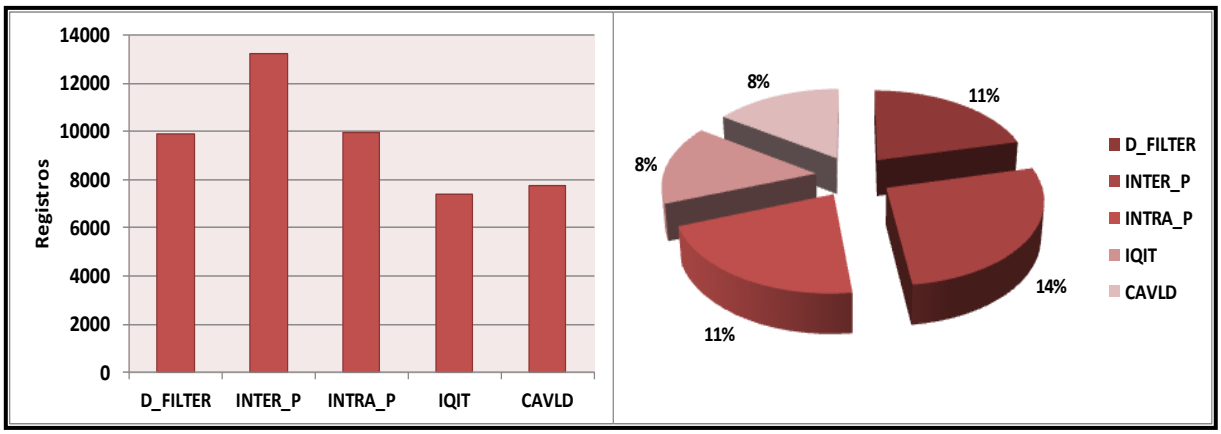


Figura 33. Utilización de registros obtenida en Synplify

La figura 34 permite ver la utilización de LUTs por cada uno de los bloques. Se observa que es el bloque INTRA_P el que mayor consumo presenta de LUTs, que se debe a que la implementación de las memorias que utilizan se ha realizado como RAM distribuida (*Select* RAM), como se verá a continuación. También se puede observar que es el bloque CAVLD el que menos utilización de LUTs posee. En el caso de la utilización de LUTs, Synplify Premier reduce el consumo de LUTS un 125% en promedio.

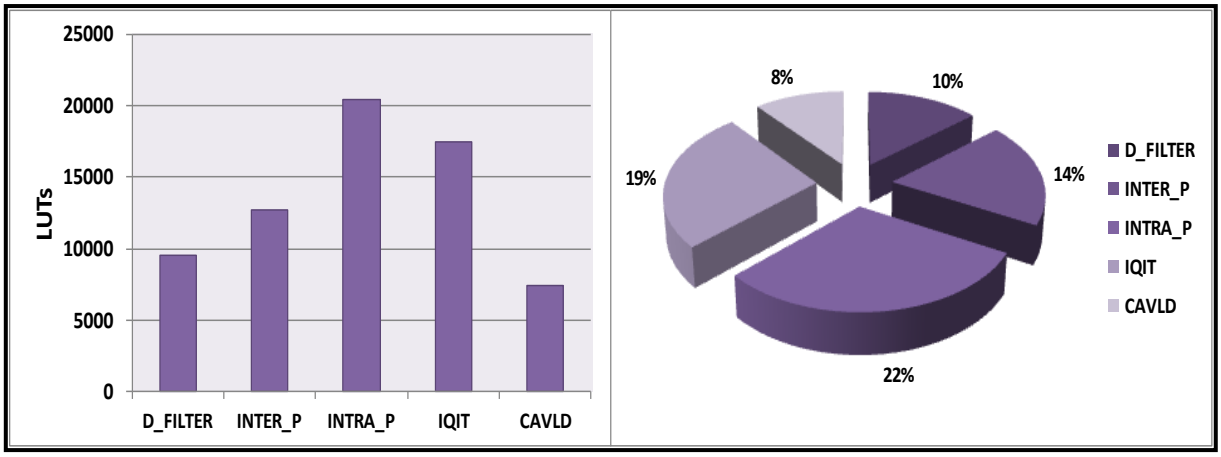


Figura 34.Utilización de LUTs obtenida en Synplify

La utilización de recursos singulares de la FPGA, como son las memorias de bloques (*Block* RAMs) y DSPs se muestra en la figura 35.

En el caso de las BRAMs se utilizan para almacenar variables de tipo array, de tamaño relativamente grande, que eleva el consumo de recursos de memoria distribuida. En el diseño del decodificador, es el bloque INTER_P el que mayor número utiliza, seguido del bloque D_FILTER. Si recordamos la arquitectura de los bloques indicados, los resultados obtenidos son congruentes con los previstos inicialmente. El bloque INTRA_P hace uso de una única BRAM, mientras que los bloques IQIT y CAVLD no hacen uso de este tipo de memoria, por lo que se deduce, como se expuso con anterioridad, que los módulos de memoria que utilizan están generados como RAM distribuida, incrementando por tanto su utilización de LUTs.

La utilización mayoritaria de los DSPs está agrupada nuevamente en el bloque INTER_P, coincidiendo con el alto número de multiplicaciones que debe hacer al realizar los cálculos para los vectores de movimiento, etc. También hacen uso de este recurso el bloque de IQIT, un bloque eminentemente dedicado al cálculo y el bloque INTRA_P.

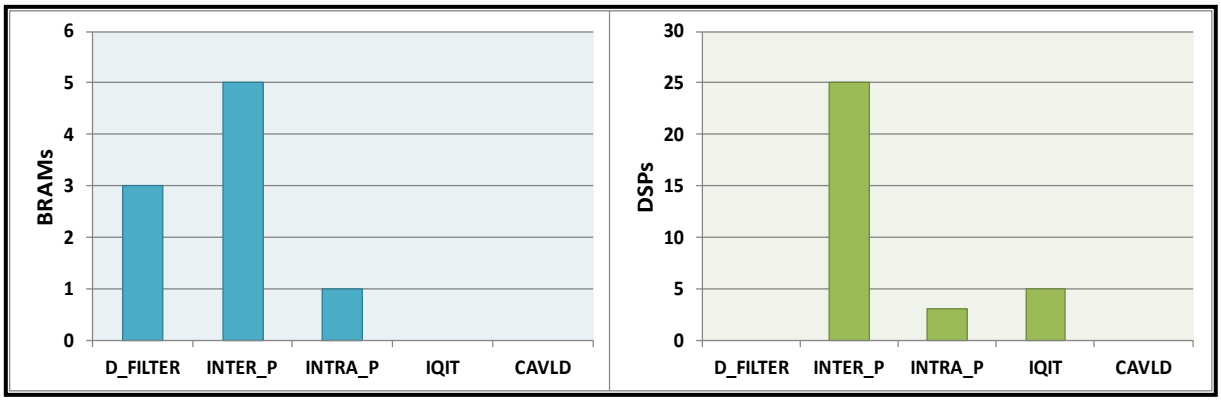


Figura 35. Utilización de BRAMs y DSPs obtenida en Synplify

5.5. Conclusiones

En este capítulo se han mostrado los resultados obtenidos tras realizar tanto la síntesis de alto nivel con Cadence CtoS, como la síntesis lógica utilizando Synopsys Synplify y Xilinx XST mediante PlanAhead.

Tras la realización de estas etapas del flujo de diseño, puede observarse que las estimaciones de utilización de recursos son más precisas a medida que se baja en el nivel de diseño, comparativa que se realizará con más detalle en el capítulo siguiente.

Referido a los resultados de utilización para cada uno de los bloques, se ve que en cuanto a consumo de recursos serán los bloques INTER_P e INTRA_P los que presentan mayor tamaño, mientras que en cuanto a frecuencia de funcionamiento será el bloque D_FILTER el limitador de la misma. Todos estos datos permiten vislumbrar cuáles van a ser los problemas que pueden surgir en el diseño una vez se vaya a realizar la implementación, permitiendo volver hacia atrás en el flujo de diseño para intentar realizar las optimizaciones que se crean convenientes.

Capítulo 6: Comparativas

6.1. Introducción

Una vez caracterizado el sistema, se pasa a comparar los resultados obtenidos en los diferentes pasos del flujo de diseño y con otras publicaciones técnicas similares. Para ello, se ha dividido este capítulo en tres apartados.

- El primer apartado establece comparativas entre los resultados obtenidos tras la síntesis tanto de alto nivel en CtoS como lógica con Synplify y XST.
- El segundo apartado establece una comparativa entre los resultados de síntesis con Synplify y otras publicaciones técnicas similares.
- Por último se pasará a la exposición de las conclusiones obtenidas a partir de estos datos.

6.2. Comparativa entre los resultados obtenidos

En la tabla 7 se muestran los diferentes resultados obtenidos tras la realización de los distintos pasos expuestos en el flujo de diseño relativo a la frecuencia de funcionamiento y el uso de LUTs y registros.

	Frecuencia estimada (MHz)		Registros		BRAMs		LUTs		
	XST	Synplify	XST	Synplify	XST	Synplify	CtoS	XST	Synplify
D_FILTER	109,082	112,200	33.993	9.882	0	3	80.283,2	22.534	9.550
INTER_P	108,357	123,100	66.313	13.249	0	5	116.540,0	40.343	12.711
INTRA_P	108,695	122,300	27.550	9.940	0	1	79.055,6	26.207	20.492
IQIT	97,809	120,100	62.466	7.414	0	0	123.056,3	38.681	17.454
CAVLD	128,688	149,600	44.524	7.750	1	0	127.899,8	23.804	7.374

Tabla 7. Resultados obtenidos tras la síntesis en alto nivel y lógica

Analizando los resultados mostrados en tabla 7, puede deducirse que la herramienta Synopsys Synplify presenta los mejores resultados tras la síntesis lógica. La estimación hecha por la herramienta Cadence CtoS, como ya se comentó, se aleja bastante de los resultados finales, pero puede estimar la tendencia para cada uno de los bloques, permitiendo tomar algunas decisiones previas a la síntesis lógica.

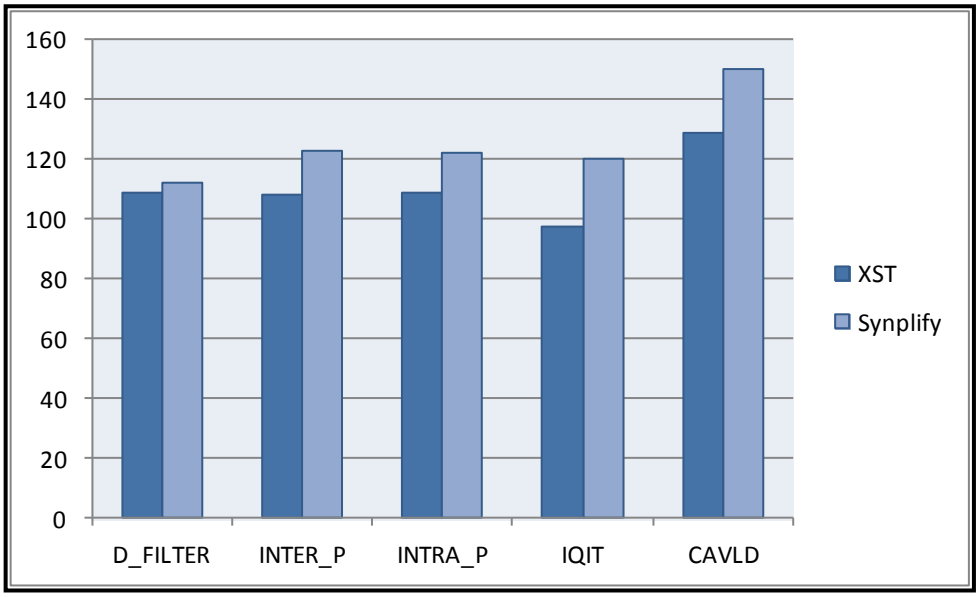


Figura 36. Frecuencia máxima estimada por XST y Synplify

La figura 36 presenta las frecuencias máximas estimadas por XST y Synplify. En ella se puede comprobar como la herramienta de síntesis de Synopsys da mejores resultados en cuanto a este parámetro para todos los bloques. Indicar que la mejora promedio es del 12 %.

En la figura 37 se presentan los resultados en cuanto a utilización de registros tras la síntesis lógica con XST y Synplify. De nuevo, la herramienta de Synopsys proporciona mejores resultados que la proporcionada por Xilinx, reduciendo considerablemente el consumo de este recurso.

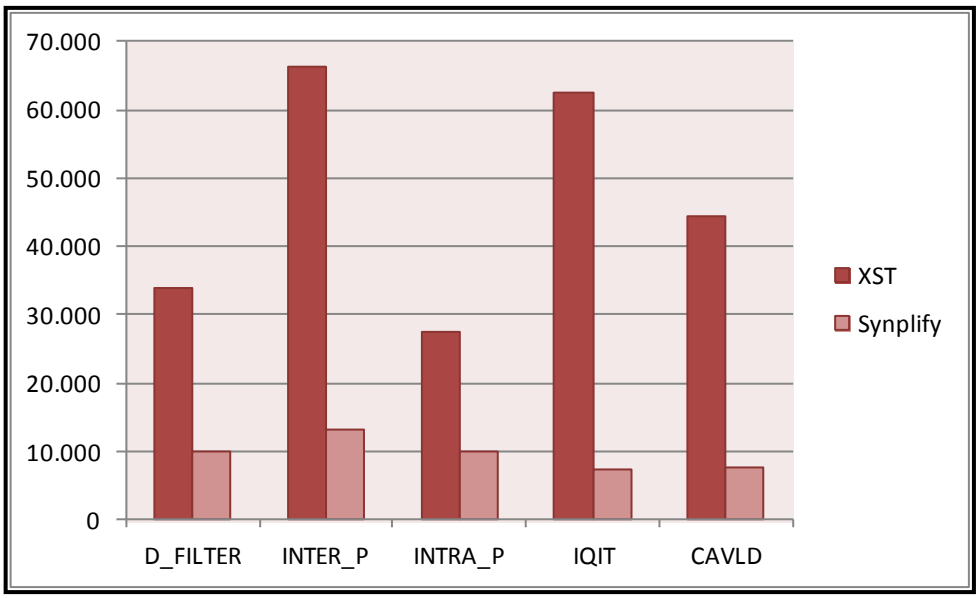


Figura 37. Utilización de registros tras la síntesis con XST y Synplify

La utilización de BRAMs se expone en la figura 38. En este caso se observa que la herramienta de Synopsys hace un mayor uso de este recurso, haciendo uso XST de una única memoria de este tipo. De esto se deriva el mayor uso de LUTs que presenta XST, dato que se presenta a continuación, dado que sintetiza los módulos de memoria como RAM distribuida.

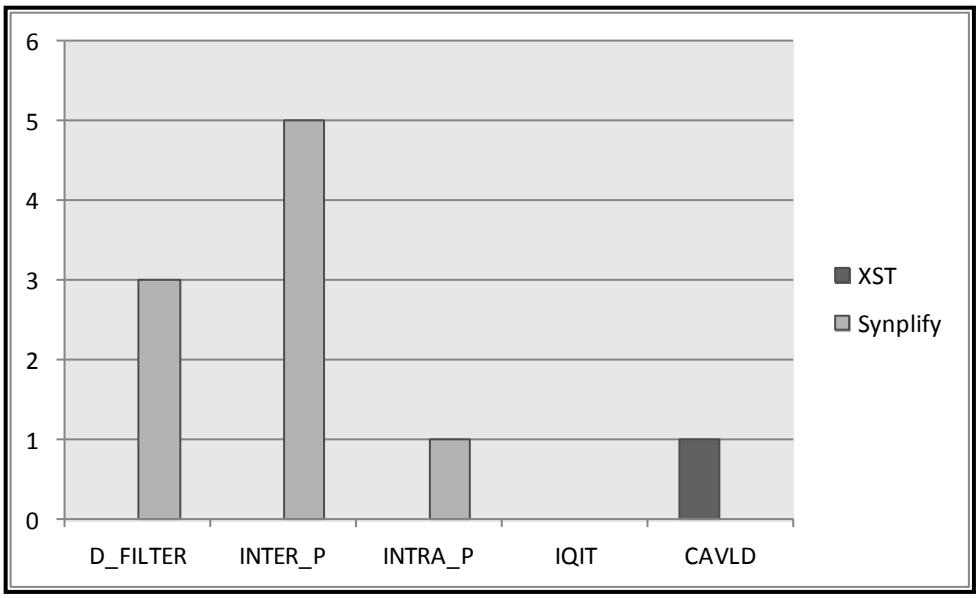


Figura 38. Utilización de BRAMs tras la síntesis con XST y Synplify

Por último, se presenta la utilización de LUTs, incluyendo la estimación realizada por la herramienta Cadence Ctos. Estos datos están representados en la figura 39 y permiten concluir que el la utilización de este recurso tras la síntesis en Synplify, es mucho menor que en los otros dos casos. Es de destacar que la estimación hecha por CtoS se aleja bastante de los resultados finales.

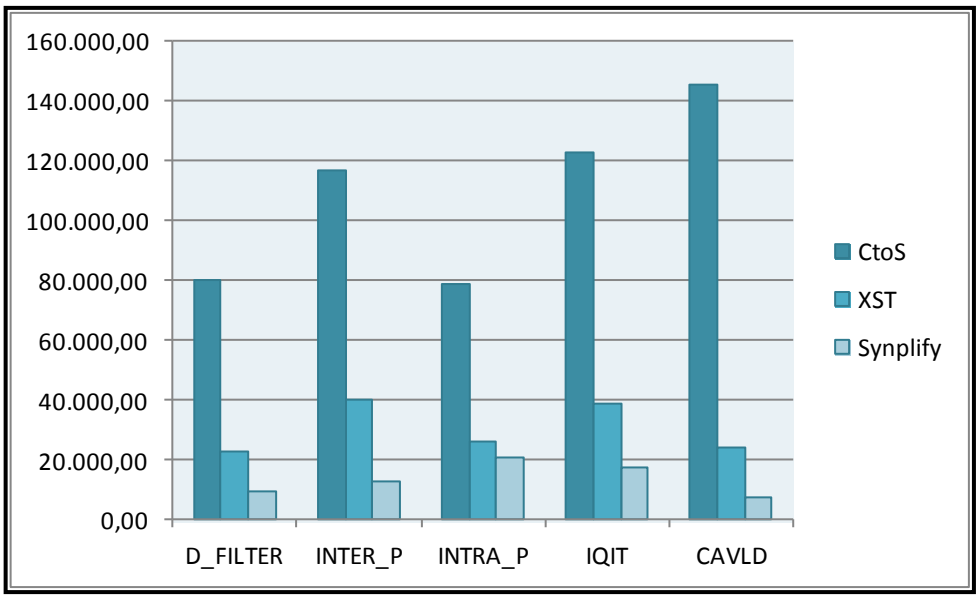


Figura 39. Utilización de LUTs tras la síntesis con XST y Synplify y la estimación hecha por CtoS

6.3. Comparativa con publicaciones técnicas similares

Para la realización de las comparativas adecuadas con otras publicaciones técnicas similares, se han examinado otros resultados obtenidos para la misma clasificación jerárquica por bloques realizada en este TFM. En la Tabla 8 se muestra la FPGA y el título de la referencia con el marcador bibliográfico utilizado para cada uno de los bloques.

BLOQUE	FPGA	Modelo del diseño	TÍTULO DE LA PUBLICACIÓN Y MARCADOR BIBLIOGRÁFICO
D_FILTER	Xilinx Virtex-II Pro	*RTL*	*FPGA design of a H.264/AVC main profile decoder for HDTV* [39]
D_FILTER	Xilinx Virtex-II Pro	*RTL*	*An HDTV H.264/AVC Deblocking Filter in FPGA with RGB Video Output* [40]
INTER_P	Xilinx Virtex-II Pro	*RTL*	*FPGA design of a H.264/AVC main profile decoder for HDTV* [39]
INTRA_P	Xilinx Virtex-II Pro	*RTL*	*FPGA design of a H.264/AVC main profile decoder for HDTV* [39]
INTRA_P	Xilinx Virtex 4	*RTL*	*A new architecture for high performance Intra Prediction in H.264/AVC decoder* [41]
INTRA_P	Xilinx Virtex-II Pro	*RTL*	*Architecture of an HDTV Intra frame Predictor for a H264 Decoder* [42]
IQIT	Xilinx Virtex-II Pro	*RTL*	*FPGA design of a H.264/AVC main profile decoder for HDTV* [39]
CAVLD	Stratix III	*RTL*	*A new fast architecture for HD H.264/AVC CAVLC multi syntax Decoder and its FPGA implementation* [43]
CAVLD	Stratix II	*RTL*	*Low cost and memoryless CAVLD architecture for H.264/AVC Decoder* [44]

Tabla 8. Publicaciones técnicas utilizadas para la realización de comparativas

Una vez examinadas estas publicaciones, se extraen los datos relevantes para el objetivo de este capítulo. La tabla 9 contiene un resumen de estos parámetros.

	REFERENCIA	Frecuencia (MHz)	LUTs	BRAMs
D_FILTER	TFM	112,20	9.550	3
	[39]	165,00	1.600	9
	[40]	135,00	4.331	65
INTER_P	TFM	123,10	12.711	5
	[39]	100,50	10.835	21
INTRA_P	TFM	122,30	20.492	1
	[39]	40,00	5.516	3
	[41]	84,80	3.975	N/D
	[42]	40,00	5.516	3
IQIT	TFM	120,10	17.454	0
	[39]	132,50	3.249	0
CAVLD	TFM	149,60	7.374	0
	[43]	74,25	3.266	N/D
	[44]	N/D	1.982	N/D

Tabla 9. Resumen de frecuencias de funcionamiento y consumo de recursos

La figura 40 contiene los datos relativos a la frecuencia de funcionamiento máxima para cada uno de los bloques. Puede observarse como en el caso de los bloques INTRA_P, INTER_P y CAVLD se alcanza una frecuencia superior que en el caso de los trabajos escogidos para la comparativa. En el caso de los bloques D_FILTER e IQIT la frecuencia máxima alcanzada es ligeramente inferior, estando de todos modos dentro de un rango similar.

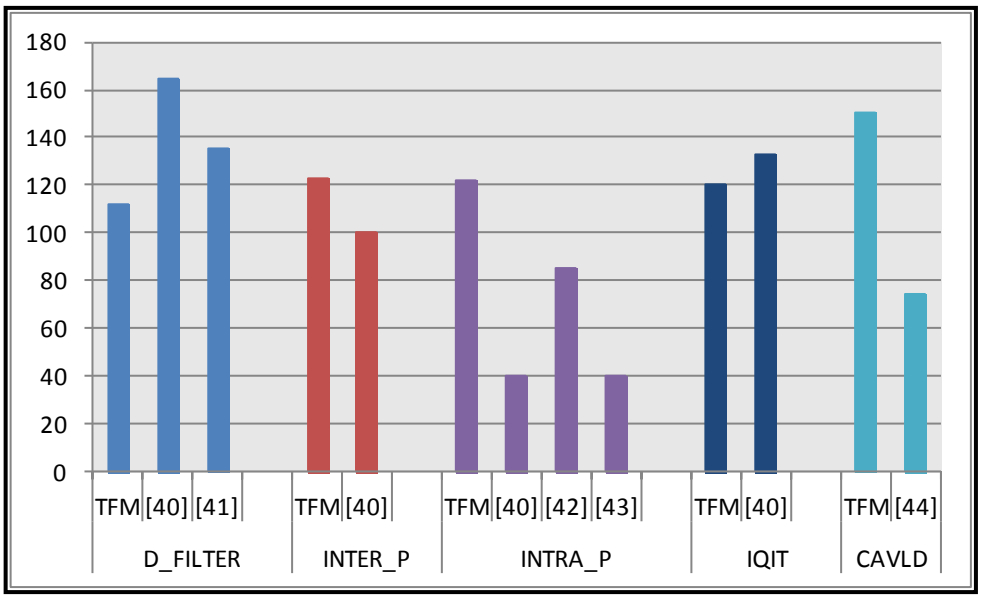


Figura 40. Frecuencias de funcionamiento por bloque

En la figura 41 se puede observar los datos concernientes al consumo de LUTs por bloque. En este caso puede verse como para el diseño escogido en este TFM sí que existen algunas diferencias significativas con respecto a otros trabajos, sobre todo en el caso de los bloques INTRA_P e IQIT. Esto podría explicarse en parte por una frecuencia de funcionamiento superior, lo cual conlleva a una duplicación de recursos, o a la implementación de las memorias como BRAMs o DRAMs.

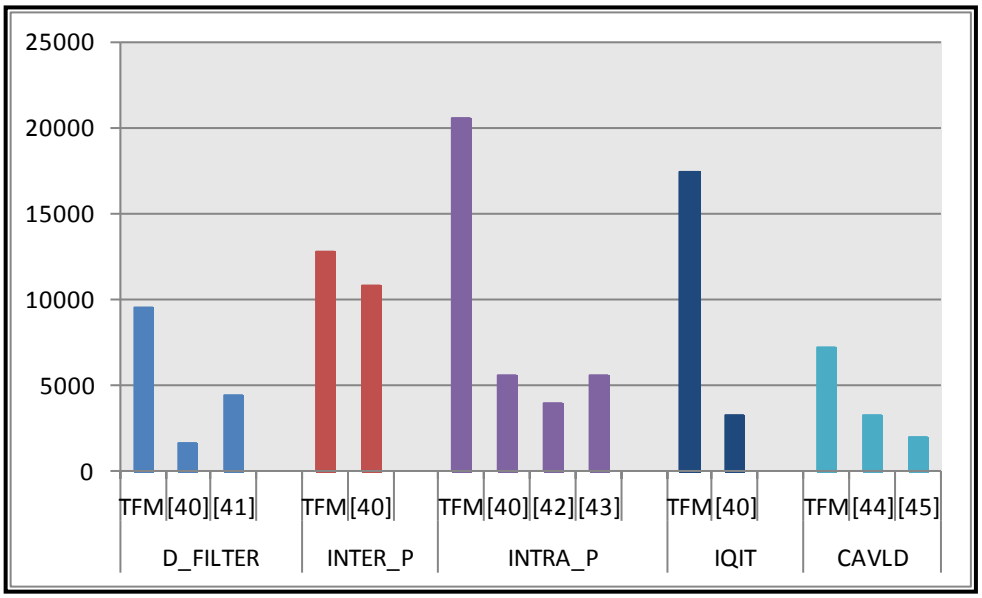


Figura 41. Consumo de LUTs por bloque

Por último, la figura 42 presenta la comparativa entre consumo de BRAMs por bloque. Puede comprobarse como el diseño de este TFM hace uso, en todos los casos, de un número menor de este recurso, lo cual puede explicar en parte el aumento en el consumo de LUTs visto en la figura 41. Es de destacar el considerable consumo de BRAMs que hace el bloque D_FILTER implementado en [40].

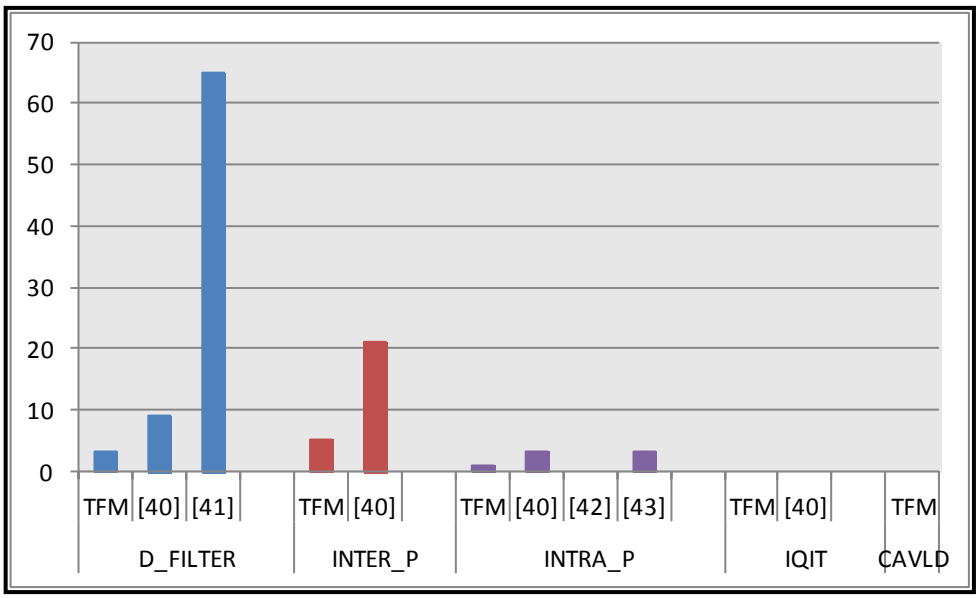


Figura 42. Consumo de BRAMs por bloque

6.4. Conclusiones

El presente capítulo hace una comparativa entre los resultados obtenidos por las distintas herramientas utilizadas en los diferentes procesos de síntesis, además de entre los resultados obtenidos tras la síntesis lógica haciendo uso de Synopsys Synplify y otras publicaciones similares.

Como conclusión de la primera parte de la comparativa se puede extraer que las estimaciones por bloques siguen aproximadamente la misma tendencia para cada uno de los bloques utilizando las diferentes herramientas, salvo algunas excepciones que en su mayor parte se producen por las diferencias entre el modo de implementación de las memorias o la frecuencia de funcionamiento.

Para la segunda parte de comparativas, se deduce la importancia de una buena planificación de los módulos de memoria, para que sean implementadas como BRAMs disminuyendo así el consumo de LUTs del diseño.

Con todos estos datos, se puede observar que se han obtenido unos resultados de síntesis lógica a partir un modelo descrito a alto nivel dentro de unos rangos razonables comparados con otros trabajos, obteniéndose una reducción de la complejidad en las líneas de código de aproximadamente x10.

Nº de líneas de SystemC: 36.148

Nº de líneas RTL Verilog: 354.567

Capítulo 7: Conclusiones y líneas futuras

7.1. Introducción

En este capítulo se presenta las conclusiones del presente TFM, así como posibles líneas futuras de trabajo a seguir en base al trabajo realizado.

7.2. Conclusiones

En este TFM se ha desarrollado una síntesis y caracterización, partiendo de una descripción de alto nivel de un IP decodificador de vídeo H.264/AVC sobre FPGA.

Se ha empezado con un estudio inicial del estándar de vídeo H.264/AVC, obteniendo una visión global de la estructura del decodificador, del formato de los datos y de la funcionalidad de sus bloques principales en cuanto a los métodos y tipos de predicción utilizados para la decodificación de imágenes.

Una vez realizado el estudio previo, se ha escogido un núcleo decodificador de vídeo H.264/AVC desarrollado por el grupo SICAD del IUMA y resultado de un proyecto de investigación, para su

síntesis y caracterización. Este núcleo decodificador descrito a alto nivel en el lenguaje SystemC, ha sido estudiado presentándose su arquitectura y funcionalidad a nivel de bloques.

Para alcanzar la síntesis y caracterización del diseño, se han seguido flujos de diseño complejos utilizando herramientas de diseño *hardware*. A raíz de esto, se han presentado las diferentes herramientas software de diseño microelectrónico que se han utilizado a lo largo del trabajo, indicando su utilización dentro del mismo. Estas herramientas han permitido la síntesis de un diseño complejo así como la obtención de datos que han permitido la posterior caracterización del mismo. El estudio de la tecnología de implementación utilizada ha sido necesario para conocer las características que la definen, y poder así tener un conocimiento del comportamiento general de la misma y de los resultados que se puede esperar en este proceso. Esto permitirá extraer conclusiones una vez realizada la caracterización del dispositivo, así como la creación de un flujo de diseño para la realización del trabajo basado en el conocimiento adquirido.

El desarrollo de este flujo de diseño para el decodificador ha sido un proceso complejo, que ha ocupado la mayor parte del tiempo de realización del TFM.

Para ello, se siguió un flujo de diseño que parte del código de referencia del núcleo decodificador, se continúa con la síntesis a alto nivel del dispositivo y se finaliza con síntesis lógica utilizando distintas herramientas para obtener una batería de resultados que permita extraer conclusiones.

Una vez realizadas, este proceso ha generado una serie de resultados en cuanto a consumo de recursos y frecuencia de funcionamiento. Con todo esto, se ha conseguido realizar una caracterización del dispositivo, extrayendo resultados que han permitido realizar comparativas entre las diferentes herramientas y con diferentes publicaciones similares.

Estos resultados ponen de manifiesto que a medida que se baja en el nivel de diseño, se reduce el consumo estimado de recursos. Sin embargo, también es reseñable el hecho de que la descripción de un diseño de este tipo a alto nivel es considerablemente más sencilla que a nivel RTL, obteniéndose una disminución en el esfuerzo realizado para obtener resultados y permitiendo llegar antes al mercado siguiendo este tipo de flujos de diseño. Este factor es muy importante en el mercado de aplicaciones multimedia actual, por lo que a pesar de ser necesario realizar algunas consideraciones previas al diseño, relativas a factores críticos como son la distribución de memorias, estrategias de síntesis, estructura jerárquica, etc., hacer uso de este tipo de herramientas puede reportar beneficios para el diseñador.

Por último, es de destacar que para añadir mayor rapidez a este flujo de diseño, se hace eminentemente importante el uso de *scripts* que aceleren el desarrollo y permitan paralelizar los diferentes procesos. El tiempo utilizado para la creación de estas herramientas se compensa totalmente con el ahorro de tiempo que se obtiene finalmente.

7.3. Líneas de trabajo futuras

Este TFM plantea una síntesis y caracterización de un IP decodificador de vídeo H.264/AVC sobre FPGA. Por tanto, como líneas de trabajo futuras se proponen las siguientes, teniendo en cuenta los trabajos que se desarrollan en el grupo.

Una mejora que es necesario introducir es la automatización del proceso de caracterización de tal forma que las gráficas obtenidas sean fácilmente generadas a partir de la colección de datos obtenidos de la ejecución de los experimentos desarrollados. Existe un conjunto de funciones semilla creadas, pero la integración en una aplicación completa se escapa de los objetivos de este trabajo.

De cara a su utilización como IP en un sistema en chip se está desarrollando una interfax AXI que facilite su utilización en ecosistemas basados en AMBA AXI, estándar industrial tanto en el mundo ASIC como FPGA para sistemas en CHIP. Este trabajo está avanzado y se obtendrán resultados en corto plazo.

Otro paso a considerar sería llevar este diseño a una implementación ASIC. Esto conllevaría una ampliación del flujo de diseño y a la utilización de nuevas herramientas, pero la obtención de resultados sería de gran utilidad para evaluar las mejoras en el funcionamiento. En esta línea se han desarrollado trabajos similares que podrían hacer más sencilla la implementación.

A medio plazo se están evaluando la posibilidad de disponer de modelos de procesadores, en concreto un IP de procesador ARM para su integración en un ASIC. La metodología desarrollada facilitará la integración y caracterización del sistema final.

BIBLIOGRAFÍA

[1] I. E. G. Richardson, *H.264 and MPEG-4 Video Compression: Video Coding for Next-Generation Multimedia.* John Wiley, 2003.

[2] N. Nedjah and L. de Macedo Mourelle, *Co-Design for System Acceleration: A Quantitative Approach.* Universidad de Michigan: Springer, 2007.

[3] M. Gokhale and P. S. Graham, *Reconfigurable Computing: Accelerating Computation with Field-Programmable Gate Arrays.* Birkhäuser, 2005.

[4] R. Neris Tomé, *Implementación y caracterización de un IP decodificador de vídeo H.264/AVC en tecnologías CMOS DSM.* EITE - ULPGC, 2012.

[5] S. Levi and A. K. Agrawala, *Real-Time System Design.* Universidad de Michigan: McGraw-Hill Pub. Co., 1990.

[6] O. Espino Santana, *Acelerador hardware para el procesado de eventos en tiempo real (AHPETIR)*, 21/05/12, 2012.

[7] Anonymous In-stat - press releases *(9/5/2012)*.

[8] International Telecommunication Union. ITU-T recommendation H.120: Codecs for videoconferencing using primary digital group transmission . 1993.

[9] International Telecommunication Union. ITU-T recommendation H.261: Códec vídeo para servicios audiovisuales A p x 64 Kbit/s. 1993.

[10] T. Sikora, *MPEG Digital Video Coding Standards.* In Digital Electronics Consumer Handbook, McGraw Hill Company, 1997.

[11] International Telecommunication Union. ITU-T recommendation H.262: Information technology – generic coding of moving pictures and associated audio information: Video. 2000.

[12] International Telecommunication Union. ITU-T recommendation H.263: Serie H sistemas audiovisuales y multimedios. Infraestructura de los servicios audiovisuales - codificación de imágenes en movimiento. 1998.

[13] International Standard ISO/IEC, "MPEG - 4 part 2: Information technology — coding of audio-visual objects," 12/2001.

[14] I. E. Richardson. The H.264 advanced video compression standard, 2010.

[15] A. Domínguez Hernández, *PFC: Metodología De Codiseño HW/SW Para Un Decodificador H.264/AVC.* Las Palmas de Gran Canaria: IUMA, Universidad de Las Palmas de Gran Canaria, 2008.

[16] Anonymous MPEG. ORG - MPEG home *(04/17/2012)*.

[17] Ke Xu and Chiu Sing Choy. Low-power H.264/AVC baseline decoder for portable applications. Presented at Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on. 2007.

[18] Anonymous DivX plus codec pack – DivX, AVI y MKV para cualquier reproductor multimedia | DivX.com *(9/5/2012)*.

[19] Anonymous Proceedings of the ASP-DAC 2006. Asia and south pacific design automation conference 2006 (IEEE cat. no.06EX1199C). Presented at Design Automation, 2006. Asia and South Pacific Conference on. 2006.

[20] H. Hassan, J. Soriano, J. Montagud and C. Domínguez, *Instrucción En El Diseño De Sistemas Empotrados: Aplicación Al Control De Un Brazo Robot.* Departamento de Informática de Sistemas y Computadores, Universidad Politécnica de Valencia: Universidad Politécnica de Valencia.

[21] Anonymous Renesas application specific products by renesas technology america, inc. - ARM connected community *(5/29/2012)*.

[22] G. Martin and R. Zurawski. Tendencias de los sistemas integrados. oportunidades y retos de la tecnología de sistemas en chips y de sistemas integrados interconectados en la automatización industrial. 2006.

[23] P. List, A. Joch, J. Lainema, G. Bjontegaard and M. Karczewicz. Adaptive deblocking filter. *Circuits and Systems for Video Technology, IEEE Transactions on 13(7),* pp. 614-619. 2003.

[24] M. Wien. Variable block-size transforms for H.264/AVC. *Circuits and Systems for Video Technology, IEEE Transactions on 13(7),* pp. 604-613. 2003.

[25] International Telecommunication Union. ITU-T recommendation H.264: Advanced video coding for generic audiovisual services. 2010.

[26] Anonymous ARTEMI: ARTEMI+ WebSite *(9/5/2012)*.

[27] Anonymous Real academia española *(9/5/2012)*.

[28] E. Valderrama. "Flujo de diseño," in Anonymous 2009.

[29] S. Rigo, R. Azevedo and L. Santos, *Electronic System Level Design: An Open-Source Approach.* Springer, 2011.

[30] Anonymous IEEE standard for standard SystemC language reference manual. *IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005)* pp. 1-638. 2012.

[31] T. Grötker, *System Design with SystemC.* Boston: Kluwer Academic Publishers, 2002.

[32] D. C. Black, J. Donovan and SpringerLink, *SystemC: From the Ground Up.* Springer, 2009.

[33] M. Ganguly, *SystemC Overview.* Synopsys, Inc., 2001.

[34] Cadence Design Systems Inc, "Cadence C-to-silicon compiler: User guide," Cadence Design Systems Inc, 2011.

[35] M. C. Conrath, B. Andrietti and J. Couleur. System designer's realization of a core-based ASIC in a comfortable environment [microcontrollers]. Presented at ASIC Seminar and Exhibit, 1990. Proceedings., Third Annual IEEE. 1990.

[36] Xilinx Inc, "PlanAhead: User guide," Xilinx, 2011.

[37] Xilinx Inc. Virtex-5 family overview. Xilinx Inc. 2009.

[38] Xilinx Inc. ISE design suite software manuals and help. 2010.

[39] L. V. Agostini, A. P. Azevedo Filho, V. S. Rosa, E. A. Berriel, T. G. S. Santos, S. Bampi and A. A. Susin. FPGA design of A H.264/AVC main profile decoder for HDTV. Presented at Field Programmable Logic and Applications, 2006. FPL '06. International Conference on. 2006.

[40] V. S. Rosa, A. A. Susin and S. Bampi. An HDTV H.264 deblocking filter in FPGA with RGB video output. Presented at Very Large Scale Integration, 2007. VLSI - SoC 2007. IFIP International Conference on. 2007.

[41] Xun He, Dajiang Zhou, Jinjia Zhou and S. Goto. A new architecture for high performance intra prediction in H.264 decoder. Presented at Intelligent Signal Processing and Communication Systems, 2009. ISPACS 2009. International Symposium on. 2009.

[42] W. T. Staehler, E. A. Berriel, A. A. Susin and S. Bampi. Architecture of an HDTV intraframe predictor for a H.264 decoder. Presented at Very Large Scale Integration, 2006 IFIP International Conference on. 2006.

[43] T. G. George and N. Malmurugan. A new fast architecture for HD H.264 CAVLC multi-syntax decoder and its FPGA implementation. Presented at Conference on Computational Intelligence and Multimedia Applications, 2007. International Conference on. 2007.

[44] T. L. da Silva, J. A. Vortmann, L. V. Agostini, A. A. Susin and S. Bampi. Low cost and memoryless CAVLD architecture for H.264/AVC decoder. Presented at VLSI, 2009. ISVLSI '09. IEEE Computer Society Annual Symposium on. 2009.