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

METODOLOGÍA DE DISEÑO PARA PLATAFORMAS EXTENSIBLES BASADA EN FPGA ZYNQ 7000 DE XILINX

Autor: Alejandro García Nieto
Tutor(es): Antonio Núñez Ordóñez
Pedro Hernández Fernández
Fecha: Septiembre 2014

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

METODOLOGÍA DE DISEÑO PARA PLATAFORMAS EXTENSIBLES BASADA EN FPGA ZYNQ 7000 DE XILINX

HOJA DE FIRMAS

Alumno/a:	Alejandro García Nieto	Fdo.:
Tutor/a:	Antonio Núñez Ordóñez	Fdo.:
Tutor/a:	Pedro Hernández Fernández	Fdo.:

Fecha: Septiembre 2014

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

METODOLOGÍA DE DISEÑO PARA PLATAFORMAS EXTENSIBLES BASADA EN FPGA ZYNQ 7000 DE XILINX

HOJA DE EVALUACIÓN

Calificación: .....

Presidente: Dr. D. Roberto Sarmiento Rodríguez Fdo.:

Secretario: Dr. D. Sebastián López Suárez Fdo.:

Vocal: Dr. D. Roberto Esper-Chaín Falcón Fdo.:

Fecha: Septiembre 2014

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

Índice

Índice de Figuras

Índice de Tablas

Agradecimientos

En primer lugar, quisiera agradecerle a toda mi familia, y en especial a mis padres, Manuel José y María Teresa el apoyo en la realización de este Máster y en especial en el Trabajo Fin de Máster.

También quisiera agradecerle a mis tutores, Antonio Núñez Ordóñez y Pedro Hernández Fernández, así como a Pedro P. Carballo su dedicación en el proyecto y haberme proporcionado la oportunidad de realizarlo.

Por último, por supuesto, me acuerdo de mis compañeros y amigos, Mohamed y Sandro. También de mis compañeros del Máster y de otras personas y compañeros, como Eva, Yanis, Dani, Andy y más que no he nombrado.

Resumen

La ocupación de tráfico de vídeo en el tráfico global de Internet en redes de telefonía móvil sigue una tendencia creciente en los últimos años. En este sentido, un factor crítico relacionado con los codificadores/decodificadores de vídeo es el incremento en el consumo de potencia.

Con el objetivo de aumentar la rapidez del proceso de codificación/decodificación pueden utilizarse aceleradores *hardware*, al mismo tiempo que se controla el consumo de potencia. Se necesita por tanto de plataformas que soporten la implementación *hardware* del acelerador y una infraestructura de procesamiento para soportar la ejecución del *software* empotrado asociado. Una opción para la implementación del acelerador *hardware* es el uso de FPGA.

En el caso de las FPGA, en las familias más avanzadas se dispone de procesadores empotrados en formato de tipo *hard* IP o *soft* IP, como es el caso de las plataformas de la serie ZYNQ-7000 de Xilinx. Esta serie de plataformas proporcionan múltiples puertos de comunicación, como UART, USB o Gigabit, que facilitan la integración en un sistema de uso industrial. Además soportan los principales controladores para diferentes tipos de memorias.

Por otra parte, existen diferentes herramientas *software* que permiten realizar el proceso de generación de los aceleradores *hardware* sobre FPGA siguiendo una metodología de diseño de manera que se acoten los tiempos de diseño.

En este Trabajo Fin de Máster se parte del diseño en el nivel RTL del *Deblocking Filter* del proyecto de investigación PCCMUTE para comprobar los resultados de la herramienta Vivado Design Suite en las etapas de síntesis lógica e implementación siguiendo las estrategias de síntesis *top-down* y *bottom-up*. Asimismo, se realizan ambas etapas con la herramienta Synplify Premier DP y se comparan los resultados entre las dos herramientas y entre estrategias de síntesis. Los resultados muestran los mejores resultados que ofrece Synplify Premier para la implementación del diseño.

Por otra parte, la herramienta Vivado permite modelar un sistema mediante bloques IP, de los que proporciona muchos tipos, centrándose este diseño principalmente en los relacionados con la interfaz AXI4. Por esto, se emula la plataforma utilizada, considerándose la parte de procesamiento y la lógica programable usada.

Abstract

The global traffic of video on internet and for mobile phone networks follows a growing trend in recent years. For this reason it is more important to create efficient video codecs. In this respect, a critical factor related to the video codecs is the increase power consumption.

With the goal of increasing the speed of the codec process and the control of the power consumption, hardware accelerators can be used. Therefore platforms that support the implementation of hardware accelerators and also processing infrastructure to support associated embedded software execution are required. An option for hardware accelerator implementation is the use of reconfigurable and extensible platforms based on FPGA.

In the case of the FPGAs, the most advanced families provide embedded processors in hard IP or soft IP versions, such as ZYNQ-7000 serie platforms of Xilinx. This platform series provide a multiprocessor core based on ARM Cortex A9, multiple communication ports, as UART, USB or Gigabit, that facilitate the integration on an industrial use system. Also, those support the main interfaces to different types of memory systems.

Moreover, there are different electronic designtools which let perform the generation process of hardware accelerator on FPGA following a design methodology so that the design times are reduced.

This Final Master Project starts from RTL level of a Deblocking Filter obtained in the PCCMUTE research project to validate the results of the Vivado Design Suite tool in the logic synthesis and implementation stages. During the project, top-down and bottom-up synthesis strategies have been evaluated. Also, both tasks are performed with Synplify Premier DP tool and the results between both tools and both synthesis strategies are compared.

Also, the Vivado tool let create a IP blocks interface, which provide many types of blocks interfaces for AXI4.

Acrónimos

ACP	*Accelerator Coherency Port*
APB	*Advanced Peripheral Bus*
APU	*Application Processor Unit*
AXI	*Advanced eXtensible Interface*
BRAM	*Block RAM*
CAN	*Controller Area Network*
CLB	*Configurable Logic Blocks*
CPU	*Central Processing Unit*
DAP	*Debug Access Port*
DDR	*Double Data Rate*
DMA	*Direct Memory Access*
DRC	*Design Rule Check*
ECT	*Embedded Cross Trigger*
EMIO	*Extended Multiplexed I/O*
ETB	*Embedded Trace Buffer*
FMC	*FPGA Mezzanine Card*
FPGA	*F*ield Programmable Gate Array
FTM	*Fabric Trace Monitor*
GMII	*Gigabit Media Independent Interface*
GUI	*Graphical User Interface*
HLS	*High-Level Synthesis*
HP	*High Performance*
I/O	*Input/Output+*
IOP	*Input/output Peripherals*
ITM	*Instrument Trace Macrocell*
IUMA	Instituto Universitario de Microelectrónica Aplicada
L2	*Level 2*
LL	*LocalLink*
LPDDR	*Low Power Double Data Rate*
LUT	*Look Up Table*
MII	*Media Independent Interface*
MIO	*Multiplexed I/O*
OTG	*On-the-Go*
PCAP	*Processor Configuration Access Port*
PCCMUTE	*Power Consumption Control in Multimedia Terminals*
PL	*Programmable Logic*
PS	*Processing System*
PTM	*Program Trace Macrocell*
RAM	*Random-access memory*

RGMII	*Reduced Gigabit Media Independent Interface*
SCU	*Snoop Control Unit*
SD	*Secure Digital*
SDHC	*Secure Digital High Capacity*
SDHS	*Secure Digital High-Speed*
SDIO	*Secure Digital Input Output*
SGMII	*Serial Gigabit Media Independent Interface*
SIMD	*Single Instruction, Multiple Data*
SMC	*Static Memory Controller*
SOC	*System-On-Chip*
SWDT	*System WatchDog Timer*
TBI	*10-bit Parallel Interfaces*
TNS	*Total Negative Slack*
TTC	*Triple Timer Counters*
U-Boot	*Universal Bootloader*
ULPI	*UTMI+ Low Pincount Interface*
UTMI+	*UTMI extension supporting USB host and on-the-go*
WNS	*Worst Negative Slack*
XAC	*Analog-to-digital converter*

Capítulo 1: Introducción

1.1. Antecedentes

La implementación de sistemas en chip complejos requiere de plataformas reconfigurables que permitan realizar una fase de prototipado. Es más, esta plataforma reconfigurable puede disponer de los recursos necesarios para ser utilizada como implementación definitiva del sistema en chip cuando los requisitos de potencia e integración sean suficientes.

La utilización de FPGA (*Field Programmable Gate Array*) avanzadas ha facilitado el proceso de migración desde una implementación *software*, con un uso intensivo de sistemas multiprocesador, a soluciones *hardware* más eficientes en cuanto a respuesta temporal y consumo de potencia para aplicaciones que requieren prestaciones de tiempo real.

La codificación de vídeo

Un caso típico es el de codificación y decodificación de vídeo. Actualmente el porcentaje de utilización de aplicaciones que incluyen el procesado de vídeo es muy importante tanto en dispositivos móviles como en otro tipo de dispositivos. En la Figura 1 se muestra la tendencia en cuanto a la ocupación del tráfico de vídeo en tráfico global de Internet. A modo de ejemplo, en la Figura 2 se muestra el elevado porcentaje de tráfico relacionado con contenido de vídeo en redes de telefonía móvil entre los meses de marzo y junio del año 2013 tanto en Estados Unidos como en el resto del mundo [1].

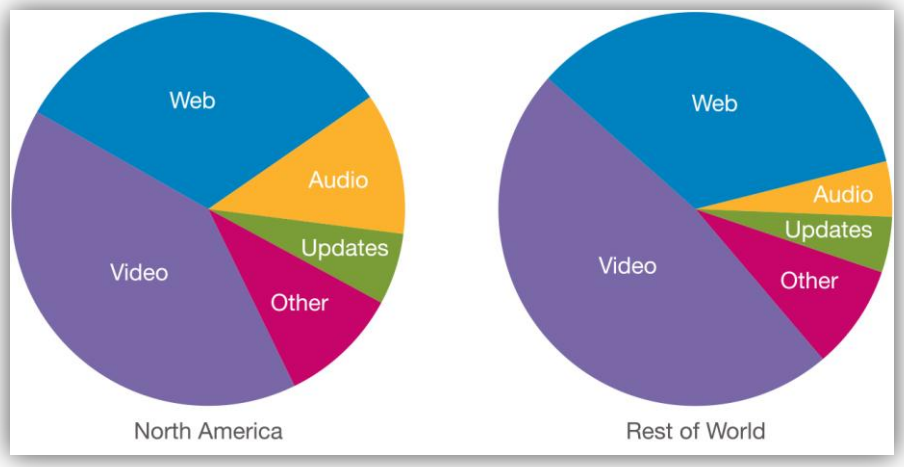

Figura 1. Tráfico de vídeo en las redes de telefonía móvil

En el diseño de aceleradores *hardware* una posibilidad interesante es la utilización de plataformas FPGA, que permiten ser programadas múltiples veces y suponen un coste limitado [2].

Por otra parte, uno de los aspectos críticos relacionados con los codificadores/decodificadores de vídeo, y por tanto, con sistemas sobre los que se ejecuten aplicaciones multimedia, es el incremento en el consumo de potencia.

La plataforma extensible Zynq

Para facilitar la implementación de sistemas complejos, como por ejemplo los codificadores de vídeo, es necesario disponer de plataformas que soporten la implementación *hardware* del acelerador y una infraestructura de procesamiento para soportar la ejecución del *software* empotrado.

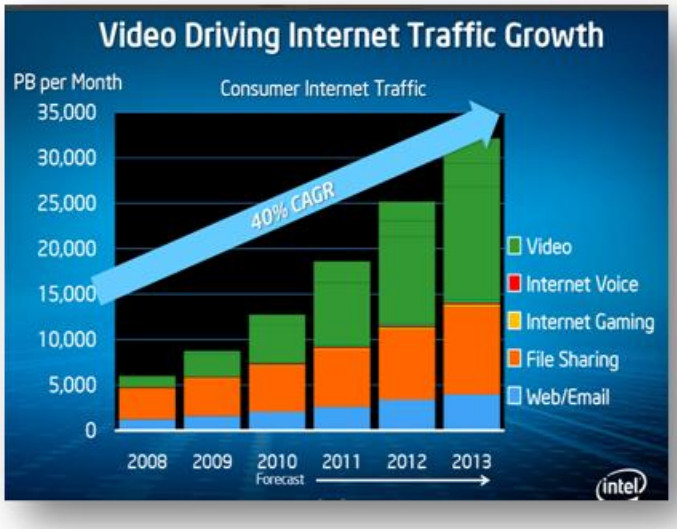

Figura 2. Contenido de vídeo en el tráfico de internet

En el caso de las FPGA, en las familias más avanzadas se dispone de procesadores empotrados en formato de tipo *hard* IP o *soft* IP. ZYNQ da un paso más al crear una plataforma extensible compuesta por dos núcleos ARM Cortex A9 con la extensión NEON, una infraestructura de comunicación en chip basada en AMBA AXI y una FPGA. Además dispone de un ecosistema que incluye un conjunto de herramientas e IP que facilitan su utilización eficiente para la creación de sistemas multiprocesadores heterogéneos (MPSoC).

Como se muestra en la Figura 3, la plataforma ZYNQ soporta un conjunto de dispositivos de comunicación integrados, comunes en un sistema empotrado (UART, USB, Gigabit, etc.) que facilitan la integración en un sistema de uso industrial. De igual forma soporta los principales controladores para diferentes tipos de memorias.

Metodologías de diseño

La implementación de sistemas en chip en plataformas extensibles basadas en FPGA requiere un flujo de diseño integrado que contemple los diferentes problemas a abordar durante el proceso de diseño. El objetivo final es acelerar la puesta en el mercado del producto, respetando los plazos y costes.

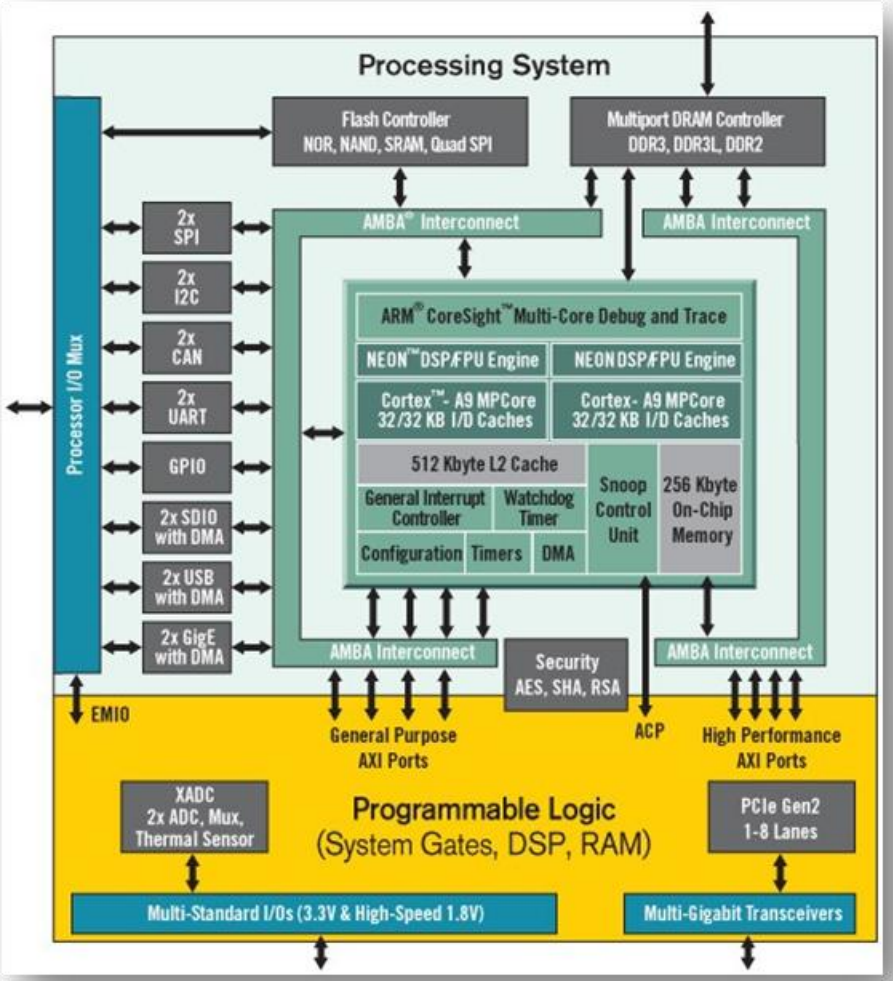

Figura 3. Plataforma ZYNQ de Xilinx

Por ello, Xilinx ha abordado la creación del flujo de diseño integrado Vivado [3] que facilita las tareas precisas para transformar una descripción de alto nivel en los ficheros de configuración necesarios para crear el sistema final. El flujo de diseño se muestra en la Figura 4. Como se puede apreciar, el flujo permite la captura desde varias fuentes (alto nivel, Matlab, HDL, etc.), integrar todo el diseño a nivel RTL y proceder a su implementación, propagando las restricciones impuestas hacia la implementación y asegurando su concordancia en distintos niveles de abstracción. El entorno también facilita la integración *hardware/software* y facilita su depurado.

El sistema ayuda a la toma de decisiones presentando resultados de los parámetros más importantes de diseño en las diferentes etapas: frecuencia de funcionamiento, recursos utilizados y potencia disipada. El control de estos parámetros es necesario realizarlo en etapas tempranas del diseño ya que aportan un impacto mayor sobre las prestaciones del mismo (Figura 5).

Para la generación del sistema y su síntesis de alto nivel Xilinx proporciona la herramienta Vivado HLS [4]. El resto del flujo de diseño se realiza mediante la herramienta Vivado Design Suite [5].

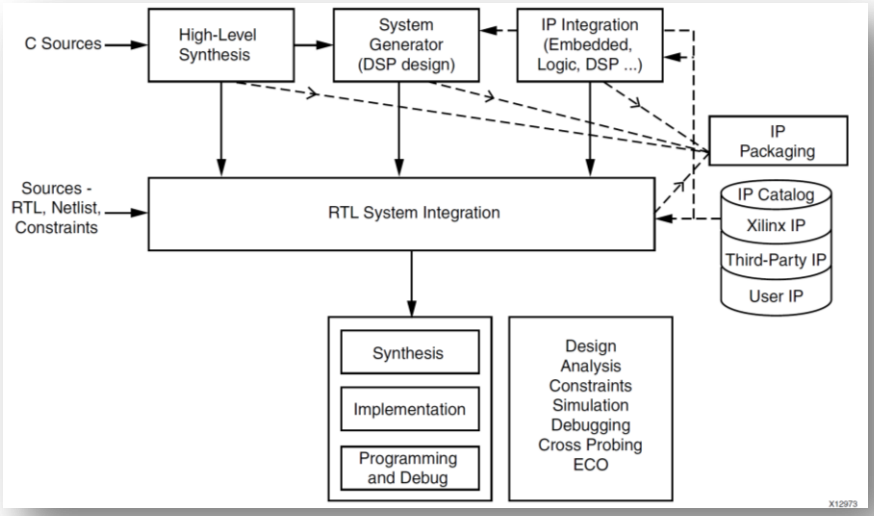

Figura 4. Flujo de diseño Vivado

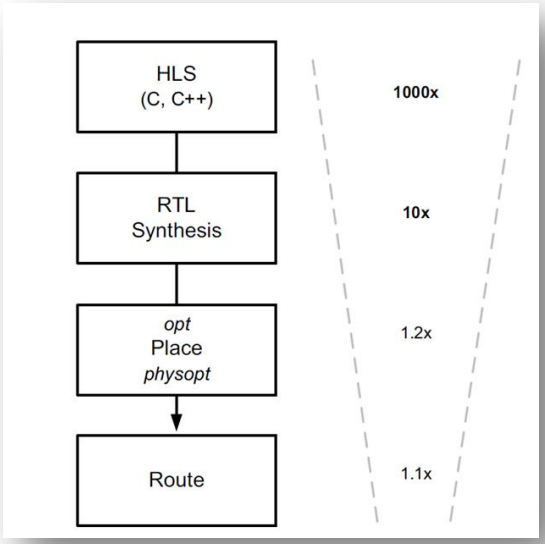

Figura 5. Impacto de las decisiones de diseño sobre las prestaciones finales

Aplicación

Para demostrar el flujo de implementación hacia la plataforma extensible ZYNQ de Xilinx se va a utilizar el diseño del módulo *Deblocking Filter* (DF) que forma parte de un decodificador H.264/SVC, desarrollado en el proyecto de investigación PCCMUTE [6], realizado conjuntamente por el Instituto Universitario de Microelectrónica Aplicada (IUMA) y la Universidad Politécnica de Madrid. Este módulo está descrito a nivel algorítmico en SystemC y en el proyecto de investigación PCCMUTE se implementó en la plataforma FPGA Virtex-5.

En este Trabajo Fin de Máster interesa partir de la descripción del sistema en RTL y, por medio de la herramienta Vivado, obtener los resultados de implementación proporcionados por la herramienta en la plataforma FPGA. El nivel RTL desde el que se parte en este trabajo se obtiene por tanto del diseño realizado dentro del proyecto de investigación PCCMUTE.

1.2. Objetivos

Este trabajo se enfoca al desarrollo de la metodología de diseño en la herramienta Vivado, principalmente con el objetivo de realizar una comparativa entre dicha herramienta y otras análogas, como es el caso de Synopsys Synplify Premier DP, partiendo del nivel RTL hasta obtener resultados de implementación sobre una plataforma FPGA extensible. Los objetivos primordiales pueden resumirse en los siguientes:

OG.1 Estudiar las características de las plataformas Zynq-7000, incluyendo sus prestaciones.
OG.2 Estudiar la metodología de diseño con la herramienta Vivado.

Los objetivos operativos definidos para este Trabajo Fin de Máster se pueden establecer en los siguientes:

OO.1 Estudiar la arquitectura de las plataformas Zynq-7000, centrándose en la plataforma sobre la que se estudie la metodología de diseño.
OO.2 Realizar la síntesis lógica del Deblocking Filter del proyecto PCCMUTE con la herramienta Vivado Design Suite.
OO.3 Realizar la implementación física del Deblocking Filter del proyecto PCCMUTE con la herramienta Vivado Design Suite.
OO.4 Establecer las comparaciones oportunas entre los resultados obtenidos mediante la herramienta Vivado Design Suite con los resultados del proyecto PCCMUTE.
OO.5 Realizar la síntesis lógica y la implementación del diseño mediante la herramienta Synplify Premier DP.
OO.6 Establecer las comparaciones pertinentes entre los resultados obtenidos con la herramienta Vivado Design Suite respecto a los resultados obtenidos con la herramienta Synplify Premier DP.

1.3. Peticionario

Actúa como peticionario de este Trabajo Fin de Máster la División de Sistemas Industriales y CAD (SICAD) del Instituto Universitario de Microelectrónica Aplicada (IUMA) de la Universidad de Las Palmas de Gran Canaria.

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

1.4. Estructura del documento

Este documento se divide en siete capítulos en los que se describe el trabajo realizado. La estructura se indica a continuación:

- **Capítulo 1: Introducción.** Se pone en contexto la importancia de los contenidos de vídeo en dispositivos móviles y se mencionan las herramientas Vivado dentro del flujo de diseño y las ventajas de las plataformas ZYNQ de Xilinx así como los objetivos del Trabajo Fin de Máster.
- **Capítulo 2: Estudio de la arquitectura Zynq-7000.** Se estudia la arquitectura de los SoC de la serie Zynq-7000 7z020 y 7z045.
- **Capítulo 3: Metodología de diseño.** Se trata la metodología de diseño, las herramientas empleadas, el lenguaje y las plataformas usadas.
- **Capítulo 4: Síntesis lógica.** Se realiza la síntesis lógica del diseño con la herramienta Vivado Design Suite, más tarde con la herramienta Synplify Premier DP con las estrategias *top-down* y *bottom-up*. Asimismo, se comparan los resultados obtenidos.
- **Capítulo 5: Implementación.** Se realiza la implementación del diseño mediante la herramienta Vivado Design Suite, posteriormente con la herramienta Synplify Premier DP siguiendo las estrategias *top-down* y *bottom-up* y realizando comparaciones entre los resultados.
- **Capítulo 6: Bloques IP.** Se utiliza la herramienta Vivado IP Integrator, que está incluida dentro de Vivado Design Suite, para definir el sistema completo mediante bloques IP, incluyendo la lógica programable utilizada y el sistema de procesamiento.
- **Capítulo 7: Conclusiones y líneas futuras.** Se definen las conclusiones generales del Trabajo Fin de Máster y se plantean posibles líneas de trabajo.

Capítulo 2: Arquitectura de la plataforma Zynq-7000

2.1. Introducción

En este Trabajo Fin de Máster se han usado dos plataformas de la serie Zynq-7000: Zynq-7000 ZC702 [7] y Zynq-7000 ZC706 [8]. La arquitectura de los dispositivos Zynq se dividen en la unidad de procesamiento (PS) y en la lógica programable (PL). Todos los SOC de la serie Zynq-7000 disponen del mismo sistema de procesamiento (PS), mientras que varían en la parte de Lógica Programable (PL) y recursos de entrada/salida.

Dentro de la serie Zynq-7000 existen varios dispositivos: 7z010, 7z015, 7z020, 7z030, 7z045 y 7z100. Los dispositivos cuya lógica programable se basa en la lógica Artix-7, como es el caso de los dispositivos 7z010, 7z015 y 7z020 tienen menor consumo de potencia, al contrario de los dispositivos con la lógica programable basada en la tecnología-7 Kintex-7, que proporciona mayor cantidad de recursos. Tal es el caso de los dispositivos 7z030, 7z045 y 7z100. Xilinx denomina a estos dispositivos como Plataforma de procesamiento extensible (EPP).

Asimismo, Xilinx comercializa plataformas en las integra algunos de estos dispositivos, como la plataforma Zynq-7000 ZC702, que integra el SoC 7z020, y la plataforma Zynq-7000 ZC706, que integra el SoC 7z045. La tecnología de fabricación de los dispositivos de la serie Zynq-7000 es CMOS 28nm.

2.2. Plataforma Zynq-7000 EPP ZC702

Los conectores que proporciona la plataforma de desarrollo Zynq-7000 EPP ZC702 [9, 10], del fabricante Xilinx [11], se indican en la Figura 6 y la Tabla 1. La FPGA de la plataforma se basa en la tecnología Artix-7.

Tabla 1. Conectores de la plataforma ZC702

N.	Descripción	N.	Descripción
1	Conector HDMI	15	Interruptor DIP para usuario
2	Transceptor del *bus* CAN	16	Zynq XC7Z020-CLG484-1 EPP
3	Interruptor del *bus* I^2C	17	Módulo de potencia
4	Interruptor *On/Off*	18	Botón de programación de la FPGA
5	Sistema de reloj	19	Reloj de Tiempo Real (RTC) conectado mediante el *bus* I^2C
6	Interruptor para el modo de configuración	20	Memoria DDR3 de 1 GB (16 bits de ancho)
7	Conector de tarjeta SD	21	Oscilador RGMII de *Ethernet* (25 MHz)
8	Memoria *Flash* Quad-SPI de 1 Gb	22	Conector de expansión del *bus* I^2C de 12 *pines*
9	Conector FMC1 LPC	23	Conector de expansión PMOD de 6 *pines* y conector de expansión PMOD de 12 *pines*
10	*Bus* I^2C programable por el usuario	24	Conector miniUSB tipo B para comunicación serie
11	Conector de expansión de XADC	25	Conector miniUSB tipo B
12	Botones de usuario	26	*LED* de estado de *Ethernet*
13	Conector FMC2 LPC	27	Conector RJ45 para *Ethernet*
14	*LEDs* de usuario	28	Conector miniUSB tipo B para JTAG

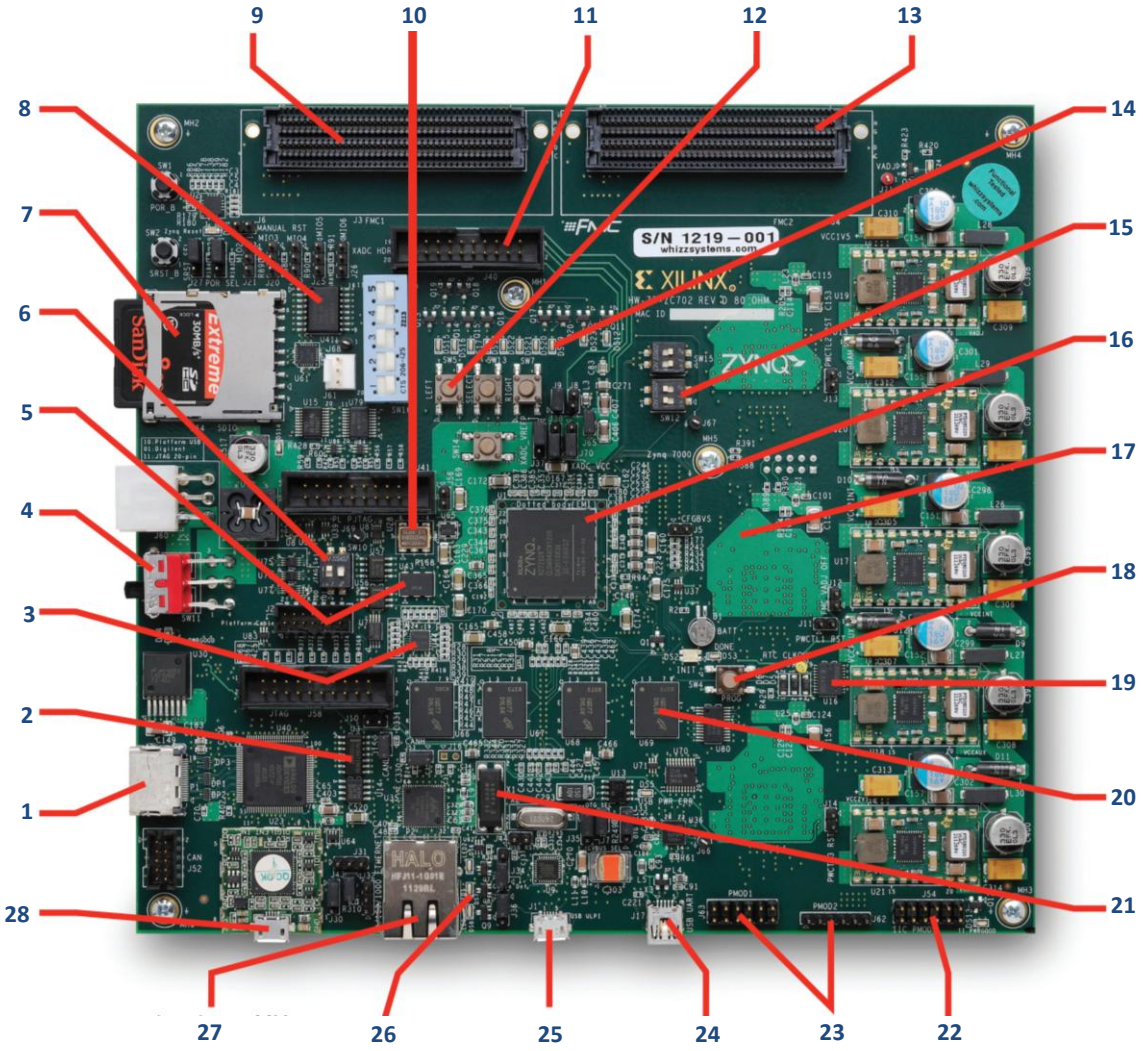

Figura 6. Plataforma Zynq-7000 ZC702

Por su parte, en la Figura 7 se muestra el diagrama de bloques de la plataforma Zynq-7000 ZC706. La parte inferior de la misma (Sistema de Procesamiento y Lógica Programable) se corresponde con la arquitectura *hardware* de la plataforma, mientras que la parte superior (*Kernel* de Linux y Espacio de Usuario de Linux) se corresponde con su arquitectura *software*.

2.3. Arquitectura *hardware* de la plataforma

En la Figura 8 se muestra la arquitectura *hardware* de los SoC de la serie Zynq-7000. La plataforma se divide en dos partes principales:

- Sistema de procesamiento (PS).
- Lógica programable (PL).

En el sistema de procesamiento se cuenta con *pines* de entrada/salida multiplexados (MIO) para la conexión de periféricos. Además, la arquitectura dispone de *pines* de entrada/salida

multiplexados extendidos (EMIO) para la conexión de la lógica programable con el sistema de procesamiento con el fin de la conectar PL con dichos periféricos.

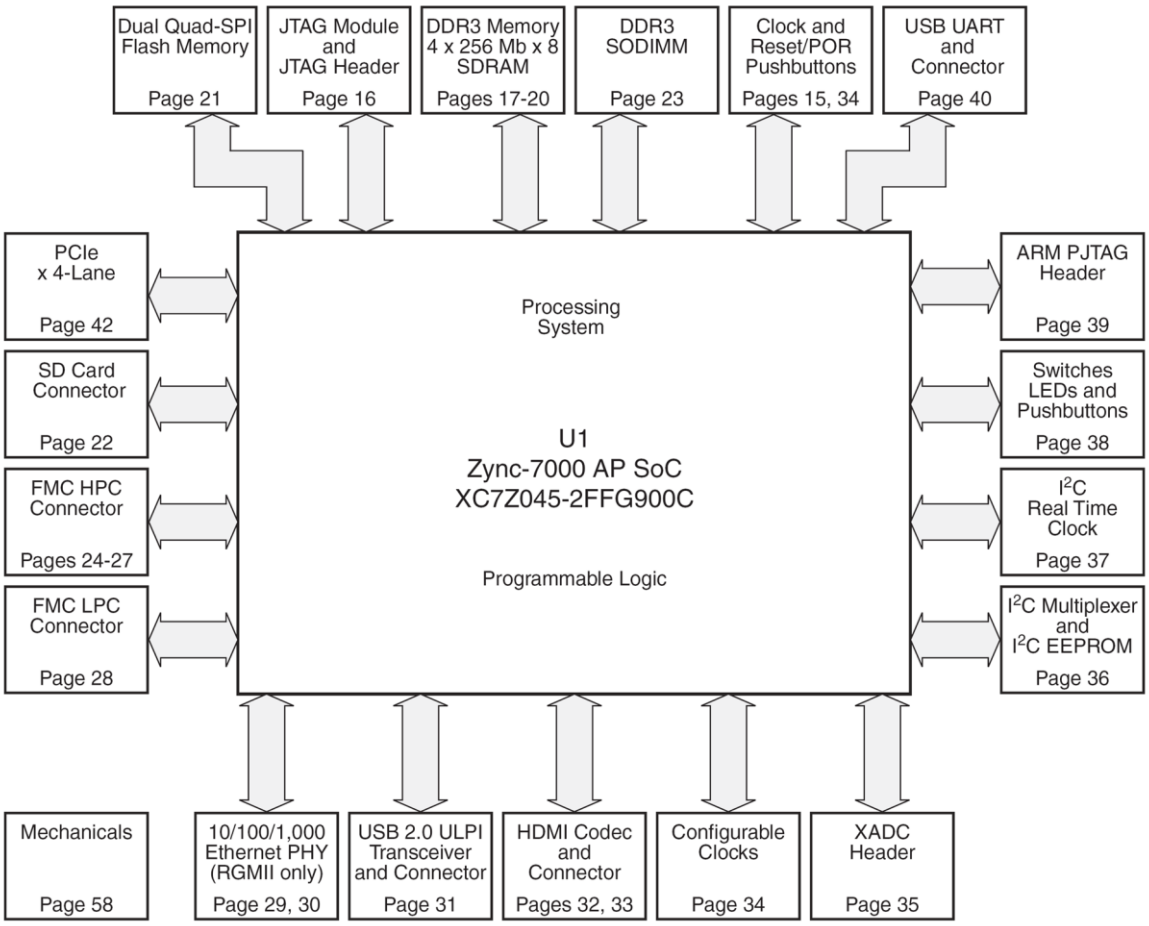

Figura 7. Diagrama de bloques de la plataforma Zynq-7000 EPP ZC706

2.3.1. Sistema de procesamiento

El sistema de procesamiento (PS) se divide en cuatro partes principales:

- Unidad de procesamiento de aplicación (APU).
- Interconexiones.
- Periféricos de entrada/salida (IOP).
- Interfaces de memoria.

Por otra parte, el sistema de procesamiento dispone de dos temporizadores/contadores triples (TTC). Por tanto, dispone de hasta seis temporizadores/contadores de 16 bits. Asimismo se dispone de un temporizador *watchdog* de 24 bits. Además, cada núcleo posee un temporizador de 32 bits y un temporizador *watchdog* de 32 bits y comparten un temporizador de 64 bits.

Unidad de procesamiento de aplicación (APU)

Se trata de la unidad que controla la arquitectura. En la Figura 9 se muestra su diagrama de bloques. Esta unidad incluye el procesador ARM Cortex- A9 [12], diseñado por ARM Holdings plc. Se trata de la CPU de la plataforma. Su arquitectura superescalar está basada en la arquitectura

RISC (*Reduced Instruction Set Computer*) de la serie ARMv7 Cortex A. Sus principales características se indican en la Tabla 2.

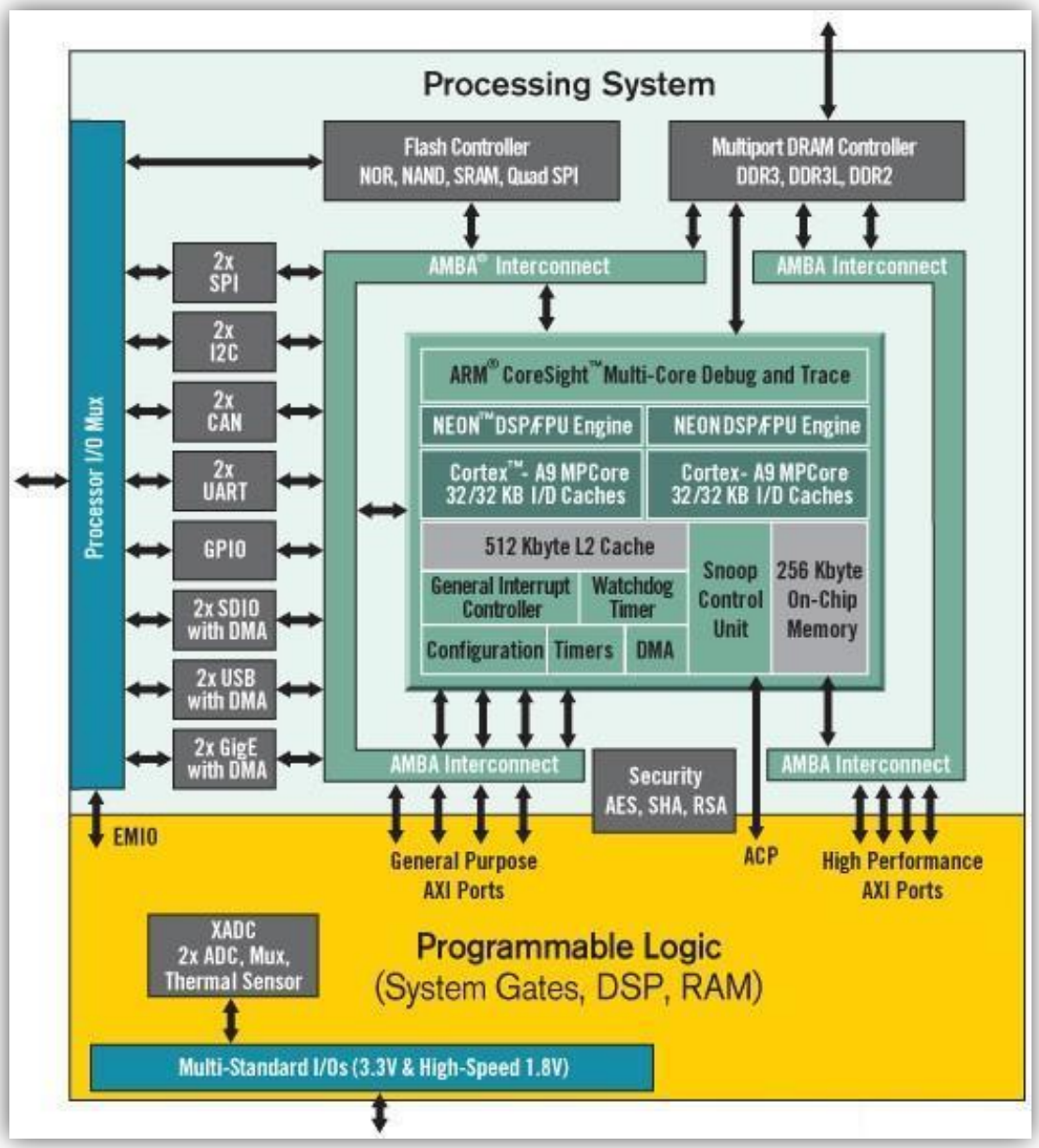

Figura 8. Diagrama de bloques de la arquitectura *hardware* Zynq-7000

Además, APU incluye una unidad SCU [13] que asegura la coherencia de las memorias *cache* de primer nivel de ambos núcleos, un controlador memoria de caché de segundo nivel (L2) basado en el modelo PL310 [14] de ARM y la correspondiente memoria, de 512 KB, un primer nivel de memoria caché (L1) con memoria de datos e instrucciones para cada núcleo separadas y 32 KB cada una, un controlador DMA con ocho canales DMA asociados, un sistema de temporización *watchdog* (SWDT) y tres controladores de temporizadores triples (TTC). Dispone asimismo de dos memorias *On-Chip*: una memoria RAM de 256 KB y una memoria ROM *On-Chip* de arranque de 128 KB. Ambas se conectan a puertos esclavos AXI de 64 bits.

El controlador de interrupciones de la plataforma, que soporta hasta 64 interrupciones, admite interrupciones de los periféricos de entrada/salida y de la lógica programable. El modelo del controlador es PL390 [15]. Dispone de una interfaz ACP esclava, que facilita los accesos coherentes desde PL hacia la CPU a través de la memoria caché de nivel 2 de las mismas. Se trata de una interfaz AXI esclava de 64 bits.

Depuración

La arquitectura soporta depuración tanto el sistema de procesamientos como de la lógica programable simultáneamente. La plataforma proporciona dos puertos JTAG que pueden ser utilizados en conjunto o de manera independiente. Si se utilizan de manera independiente uno de ellos se usa para depurar el sistema de procesamiento (PS) mientras que el otro es usado para depuración de la FPGA. La arquitectura dispone de dos puertos DAP para dar soporte a la depuración a través de JTAG. Asimismo cada puerto DAP da soporte para depuración a nivel interno desde la interconexión APB esclavo y a la interconexión AHB maestro.

La depuración del sistema de procesamiento se basa en la arquitectura CoreSight v1.0 [16] de ARM. Esta arquitectura incluye los módulos ETB [17], PTM [18] e ITM [19]. En la Figura 10 se muestra el diagrama de bloques de la arquitectura CoreSight. El módulo ITM genera las trazas, PTM construye la traza del flujo de ejecución del procesador. El módulo ECT se encarga del control en la recepción y envío de *triggers*. ETB es un *buffer* que proporciona almacenamiento de alta velocidad en tiempo real para datos de depuración (tiene un tamaño de hasta 4 KB), mientras que el módulo TPIU envía esos datos hacia la lógica programable o hacia un *chip* externo a través de *pines* EMIO o MIO respectivamente.

Tabla 2. Características del procesador ARM Cortex A9

Característica	Descripción
Arquitectura	ARMv7-A Cortex
Frecuencia	667 MHz
Número de núcleos	Dos núcleos
Bus de direcciones	32 bits
Bus de datos	64 bits
Repertorio de instrucciones	ARM, *Thumb/Thumb-2,* Extensión DSP, NEON (SIMD), VFPv3
Gestión de memoria	Direccionamiento virtual (MMU)
Tecnología Jazelle DBX y RCT	Acelera la ejecución de código Java
Depuración y traza	CoreSight DK-A9/CoreSight™ SOC-400
Memorias *cache* asociadas	2 niveles (L1 y L2)

Para la depuración de la lógica programable (PL) la plataforma dispone del módulo FTM (*Fabric Trace Monitor*), también basado en la arquitectura CoreSight de ARM. Además, este módulo da soporte a la comunicación en ambos sentidos entre PS y PL en la depuración.

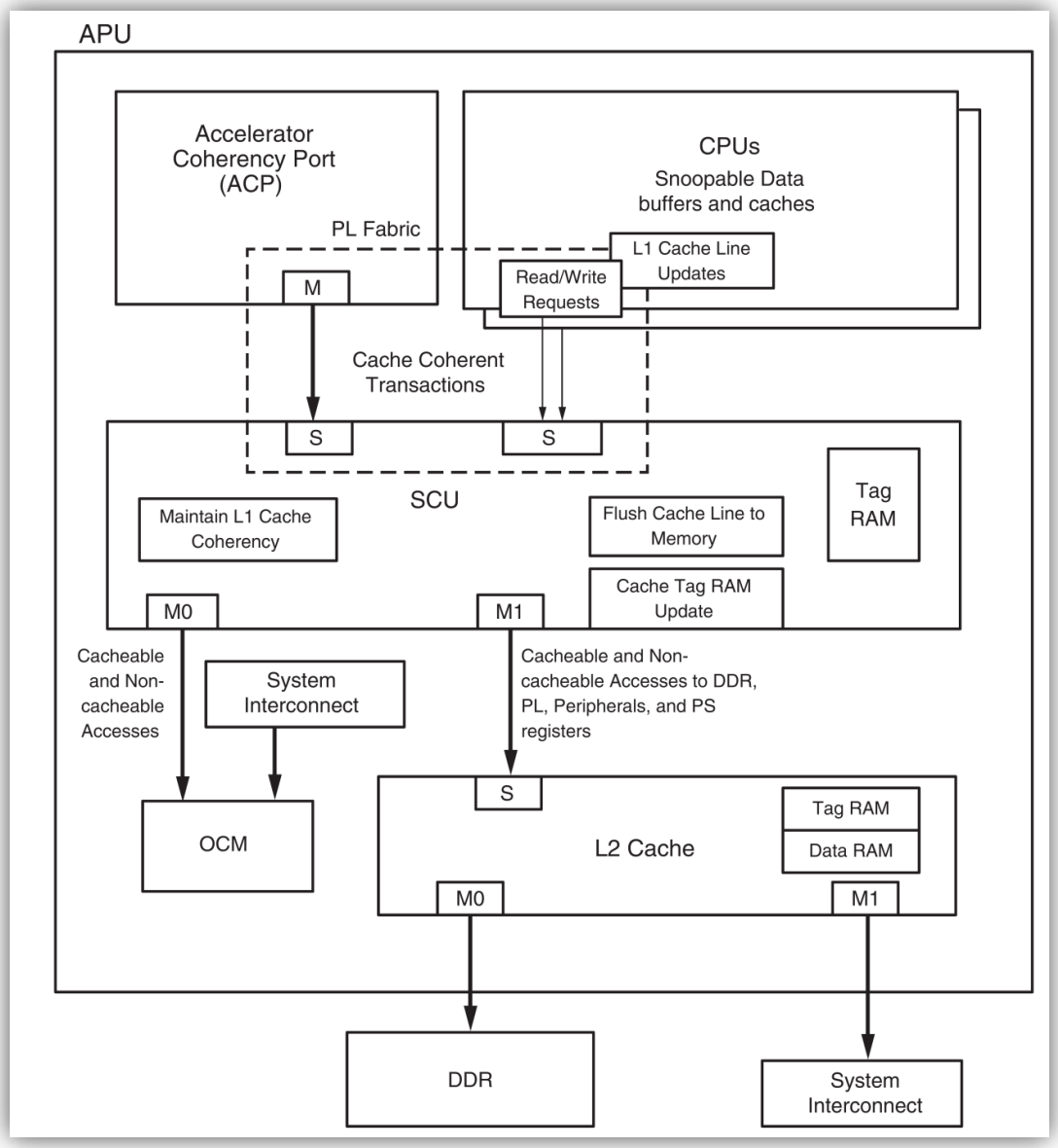

Figura 9. Diagrama de bloques de APU

DMA

El controlador DMA presente en la arquitectura se conecta a una interfaz AXI de 64 bits de ancho. Por tanto, las transferencias pueden tener un ancho de hasta 64 bits, siendo el tamaño máximo de ráfaga de 16. Por otra parte, este controlador puede manejar hasta 8 canales. En las transferencias DMA la memoria de origen y de destino puede encontrarse tanto en el sistema de procesamiento (PS) como en la lógica programable (PL).

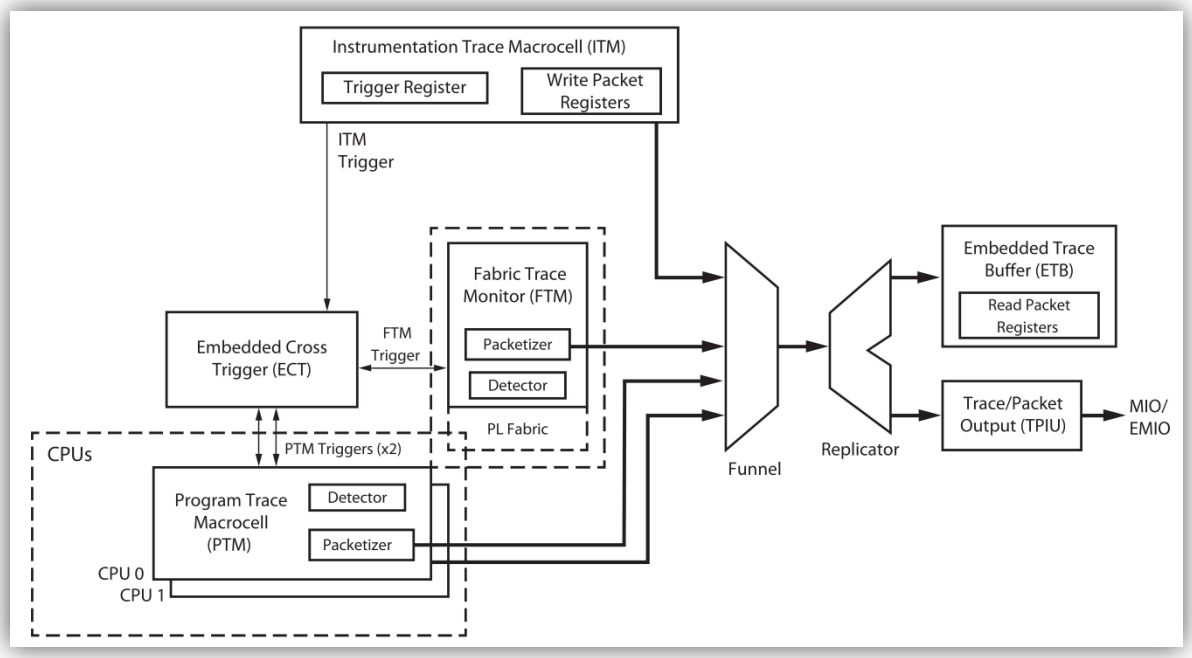

Figura 10. Diagrama de bloques de la arquitectura CoreSight

2.3.2. Interconexiones

En la conexión entre PS y PL hay dos tipos de interfaces: funcionales y de configuración. En la Figura 11 se muestra el diagrama de interconexiones de la arquitectura. Las interfaces funcionales permiten la conexión desde PS con los bloques IP diseños en PL. Se listan a continuación:

- AXI. En concreto las interfaces AXI son las indicadas en la Tabla 3.
- EMIO.
- De interrupción.
- De control del flujo DMA.
- De reloj.
- De depuración.

Tabla 3. Interfaces AXI

Nombre	Descripción
AXI_ACP	Su ancho es de 64 bits y sirve para la conexión entre PL y la unidad SCU de la CPU a través del puerto ACP
AXI_HP	Se dispone de cuatro interfaces (de 32 ó 64 bits) maestras AXI de alto rendimiento en PL que conectan PL con las memorias DDR y RAM *On-Chip*.
AXI_GP	Son interfaces de propósito general. Se proporcionan cuatro interfaces de este tipo, de 32 bits cada una. Desde el punto de vista de PS, tienen asociados dos puertos maestros y dos esclavos

La interconexión OCM proporciona acceso a la memoria *On-Chip* desde PL y desde la interconexión central. Por otro lado, la interconexión central conecta los periféricos de

entrada/salida y el controlador DMA con el controlador de memoria DDR, la memoria RAM *On-Chip* y las interfaces AXI_GP.

El módulo interconexión de memoria sirve para regular el tráfico entre las interfaces AXI_HP y el controlador de memoria DDR por una parte y por otra parte la interconexión OCM, que proporciona conexión con la memoria RAM *On-Chip*. El módulo de interconexión esclavo y el módulo de interconexión maestro aumentan la velocidad de transferencia.

Además, se dispone de un puerto maestro del *bus* APB para poder acceder a los registros de configuración del sistema de procesamiento.

Las interfaces de configuración, usadas para control, son las siguientes:

- Puerto de acceso a la configuración del procesador (PCAP).
- De configuración de estado.
- SEU.
- De estado.

La comunicación entre XADC y PS se produce a través de la interfaz PS-XADC, que consiste en una interfaz de 32 bits APB esclava desde el punto de vista de PS, y la interfaz AXI maestra que conecta PS con PL.

2.3.3. Periféricos de entrada/salida (IOP)

Se dispone de periféricos de entrada/salida para controladores *Ethernet*, USB, I^2C y SD, localizados dentro del sistema de procesamiento. También se dispone de 192*pines* de entrada/salida de Propósito General (GPIO).

GPIO

La arquitectura cuenta con 192 *pines* de entrada/salida, que se agrupan en torno a cuatro bancos de registros. Permiten la comunicación entre el sistema de procesamiento (PS) y la lógica programable (PL). La mitad de los *pines* de entrada/salida se conectan a la arquitectura a través de los *pines* EMIO y la otra mitad, a excepción de 10 que están reservados, a través de *pines* MIO. Los *pines* de entrada/salida EMIO son utilizados para conectar PS y PL entre sí. De estos, puede configurarse para que haya 64 *pines* de entrada desde la PL a PS y 64 *pines* de salida desde PS a PL, o que haya 128 *pines* de salida desde PS a PL. Los restantes 54 *pines* de entrada/salida MIO son usados para la conexión de periféricos a la arquitectura.

***Bus* I^2C**

Se dispone de un controlador del *bus* I^2C [20]. El fabricante de este *bus* es Philips Semiconductors. Sus principales características son:

- Está definido según la especificación del *bus* I^2C versión 2.
- Soporta lectura y escritura.
- Puede actuar en modo maestro y esclavo.
- La velocidad máxima de transferencia de 400 Kb/seg.

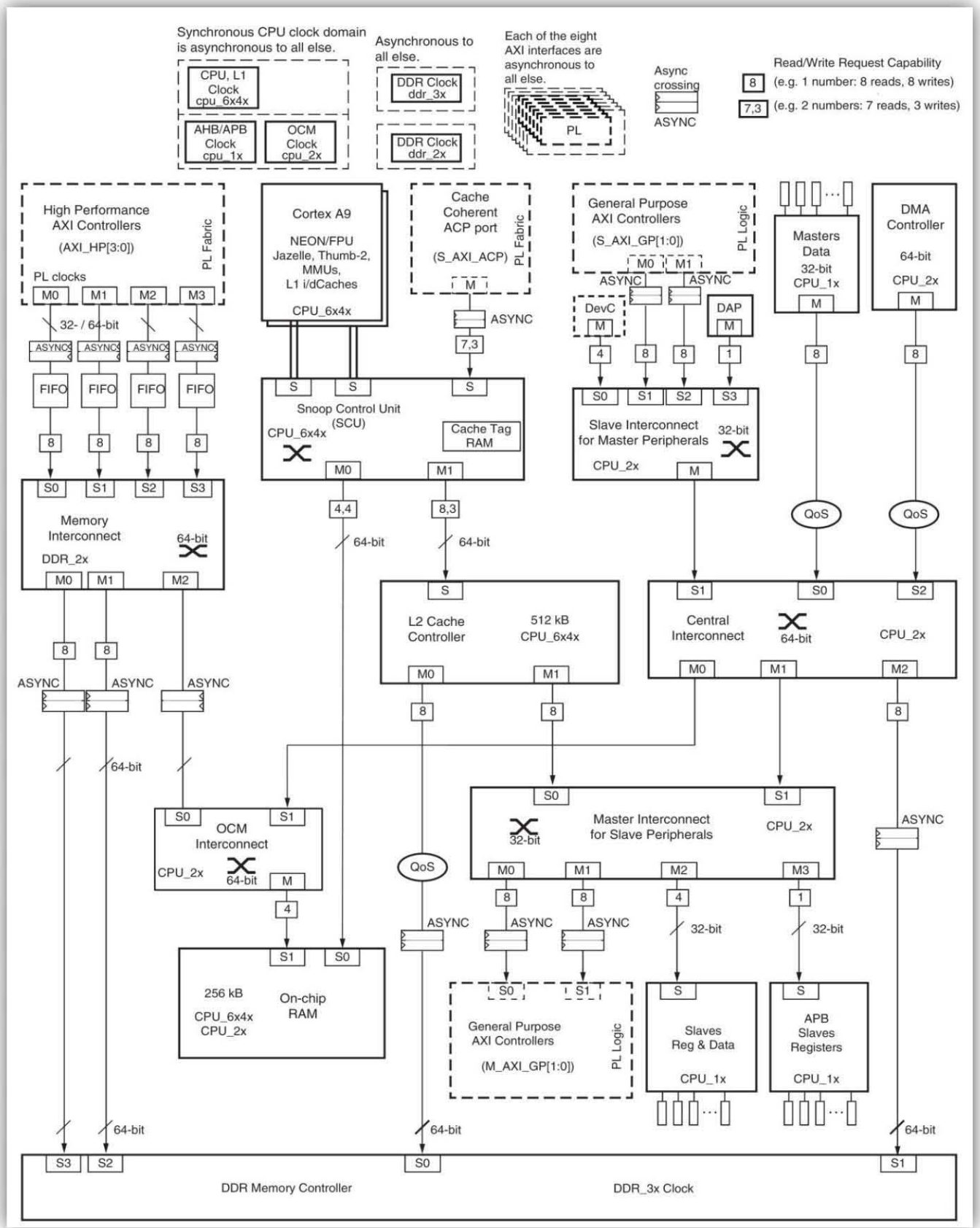

Figura 11. Diagrama de interconexiones de la arquitectura

UART

La arquitectura proporciona dos controladores para comunicación serie UART. Por su parte, la plataforma proporciona un puerto miniUSB tipo B que da soporte a la comunicación UART. Ambos controladores pueden trabajar conjuntamente conectándolos entre sí.

Controladores CAN

La arquitectura proporciona dos controladores CAN [21]. Son compatibles con los estándares *ISO 11898 -1, CAN 2.0A*, and *CAN 2.0B*. Sus velocidades de transferencia máxima son de 1 Mb/seg. Soportan los siguientes modos de funcionamiento: *configuration*, *normal*, *sleep*, *loop Back* y *Snoop*. Ambos controladores CAN pueden trabajar conjuntamente en el sistema de procesamiento (PS). Para ello se conecta el *pin* de recepción de uno con el *pin* de transmisión del otro.

***Bus* SPI**

La arquitectura proporciona dos controladores del *bus* SPI [22], que permite la conexión de la plataforma con diferentes tipos de periféricos, como memorias, sensores de temperatura, conversores analógico-digital, RTC, displays o tarjetas SD. Ambos controladores soportan los siguientes modos: maestro, esclavo y multimaestro. Cada *bus* SPI funciona a 50 MHz si se conecta a PL, a través de *pines* EMIO, y a 25 MHz si se conecta a PS, a través de *pines* MIO. Ambos *buses* SPI pueden conectarse entre sí.

Controladores SD

La arquitectura proporciona dos controladores SD que permiten comunicarse con dispositivos con interfaz SDIO, tarjetas de memoria SD y tarjetas MMC. Las interfaces de la arquitectura permiten a la plataforma la conexión con dispositivos de aplicaciones como GPS, WiMAX o UWB.

Estos controladores son compatibles con la versión 2.0 de la especificación Part A2 con SDMA, ADMA1 y ADMA2. Se da soporte asimismo al acceso de alta velocidad (SDHS) y a tarjetas de elevada capacidad (SDHC) cumpliendo las especificaciones SD2.0/SDIO 2.0. Para los controladores de SDIO se da soporte al estándar MMC3.31. Si se selecciona la velocidad de transferencia baja, esta puede variar entre 1 KHz y 400 KHz, mientras que si se selecciona velocidad de transferencia alta esta puede variar entre 1 MHz y 50 MHz (25 MB/seg).

Controlador USB

Se dispone de dos controladores USB integrados dentro de PS. Ambos controladores soportan los modos de operación *host*, *device* y OTG. El modo de operación *host* soporta la velocidad de 480 Mb/s (USB 2.0), 12 y 1,5 Mbps (*high*/*full*/*low*). El modo *device* soporta las velocidades de 12 y 1,5 Mbps (*full*/*low*).

Cada controlador se comunica con una interfaz ULPI [23], que a su vez se conectará genéricamente a un puerto USB. No obstante, la interfaz del controlador es UTMI+ [23], por lo que se implementa un *wrapper* para adaptar las señales de UTMI+ a la interfaz ULPI y viceversa.

Controlador Gigabit Ethernet

La arquitectura cuenta con dos controladores *Ethernet* que soportan las velocidades de 10/100/1000 Mb/s y que son capaces de operar en modo *half* y *full duplex*. Ambos se encuentran dentro del sistema de procesamiento (PS). Disponen de una interfaz RGMII a través de *pines* MIO y de una interfaz GMII [24] hacia PL a través de *pines* EMIO. En PL se puede configurar la interfaz GMII/MII de tal manera que sea SGMII, 1000 Base-X, RGMII o TBI.

PCI Express

La arquitectura proporciona un bloque para PCI Express compatible con la revisión 2.1 de PCI Express. Soporta velocidades de 2.5 y 5 Gb/s. Además, soporta comunicación con la interfaz AXI4 Stream.

2.3.4. Interfaces de memoria

Se dispone de interfaces de memoria FLASH para los siguientes tipos: NOR, NAND y Quad-SPI. Asimismo, se dispone de interfaces y controladores para memorias DDR2, DDR3, DDR3L y LPDDR2. Para la conexión con el controlador de memoria DDR se tienen cuatro puertos AXI, uno de los cuales se conecta con las CPUs, mientras que otros dos se conectan con la lógica programable, siendo el restante compartido por los otros tres.

Mapa de memoria

El espacio de memoria que se puede direccionar está compuesto por 4 GB (0x00000000 - 0xFFFFFFFF). En la Tabla 4 se indica el mapa de memoria y en el documento de referencia de la plataforma Zynq [10] se muestra detalladamente el mapa de memoria. Por otra parte, indicar que posee soporte para direccionamiento virtual.

Tabla 4. Mapa de memoria

Dirección base	Dispositivos
0x00000000	DDR DRAM y memoria *On-Chip*
0x00100000	Puerto de PL AXI 0 esclavo
0x80000000	Puerto de PL AXI 1 esclavo
0xE0000000	Periféricos de entrada/salida (IOP) (controladores de UART, USB, I^2C, SPI, CAN, GPIO, *Ethernet*, Quad-SPI y SDIO) y memorias NAND, SRAM/NOR
0xF8000000	Registros de configuración del PS accesibles mediante el *bus* AMBA APB
0xFC000000	Quad-SPI
0xFFFC0000	Memoria *On-Chip* si esta se mapea en la parte alta de memoria

Controlador Quad-SPI Flash

Se dispone de un controlador Quad-SPI Flash, que forma parte de los periféricos de entrada/salida (IOP) dentro del sistema de procesamiento (PS). El controlador soporta la conexión en paralelo de dos dispositivos.

La capacidad de direccionamiento para cada dispositivo es de 16 MB. El controlador puede operar en dos modos: *Linear Addressing* (únicamente lectura) e I/O (lectura y escritura). El ancho puede ser de 4 o 8 bits. Asimismo, soporta lectura simple, doble y cuádruple y escritura simple y

cuádruple. El controlador se conecta con la plataforma a través de una interfaz AXI de 32 bits y de interfaz APB de 32 bits.

Controlador de memoria estática

El controlador de memoria estática (SMC) es el controlador de los siguientes tipos de memoria:

- NAND Flash.
- SRAM asíncronas.
- NOR Flash.

El controlador accede a las memorias a través de *pines* MIO. La comunicación entre el controlador y el resto de la plataforma se realiza mediante interfaz AXI. El acceso a los registros de configuración del controlador se produce a través de interfaz APB.

El controlador soporta una memoria NAND Flash de capacidad máxima 1 GB, con anchos de acceso de 8 y 16 bits. Por otra parte, soporta la conexión de memoria SRAM y NOR Flash en paralelo, siendo el ancho del *bus* de datos de 8 bits. En caso de que solo esté conectada una de las memorias la capacidad máxima que soporta el controlador es de 64 MB, y si están conectadas en paralelo la capacidad máxima de cada es de 32 MB.

Controlador de memoria DDR

El controlador de memoria DDR soporta memorias DDR2, DDR3, DDR3L y LPDDR2. El ancho de acceso a las memorias se puede configurar para que sea de 16 o 32 bits. El diagrama de bloques del controlador se muestra en la Figura 12. Dispone de un módulo asociado a las interfaces (DDR Interfaces), un módulo de control de todo el bloque controlador (DDR Core) y el módulo encargado de controlar las transacciones generando las señales correspondientes (DDR PHY).

En el módulo DDR Interfaces dispone de cuatro interfaces AXI síncronas de 64 bits para conectarse con la plataforma. Disponen de puertos de lectura y escritura separados y soportan ráfagas de hasta dieciséis tramas de 1, 2, 4 o 8 bytes cada trama. De las cuatros interfaces, una de ellas se conecta con las CPUs y con la interfaz ACP conectada a PL y que facilita los accesos coherentes desde PL hacia las CPUs a través de la memoria caché de nivel 2 de las mismas, mientras que otros dos se conectan con la lógica programable, siendo la restante compartida por las otros tres.

2.3.5. Dominios de reloj y reset

Respecto a los dominios de reloj de PS, existe un dominio de reloj asociado a la CPU, dos dominios de reloj asociados al controlador de memorias DDR, y un dominio de reloj para los siguientes periféricos:

- USB.
- *Ethernet*.
- SDIO.
- SMC.
- *Bus* SPI.

- Quad-SPI.
- UART.
- *Bus* CAN
- *Bus* I^2C.
- GPIO.

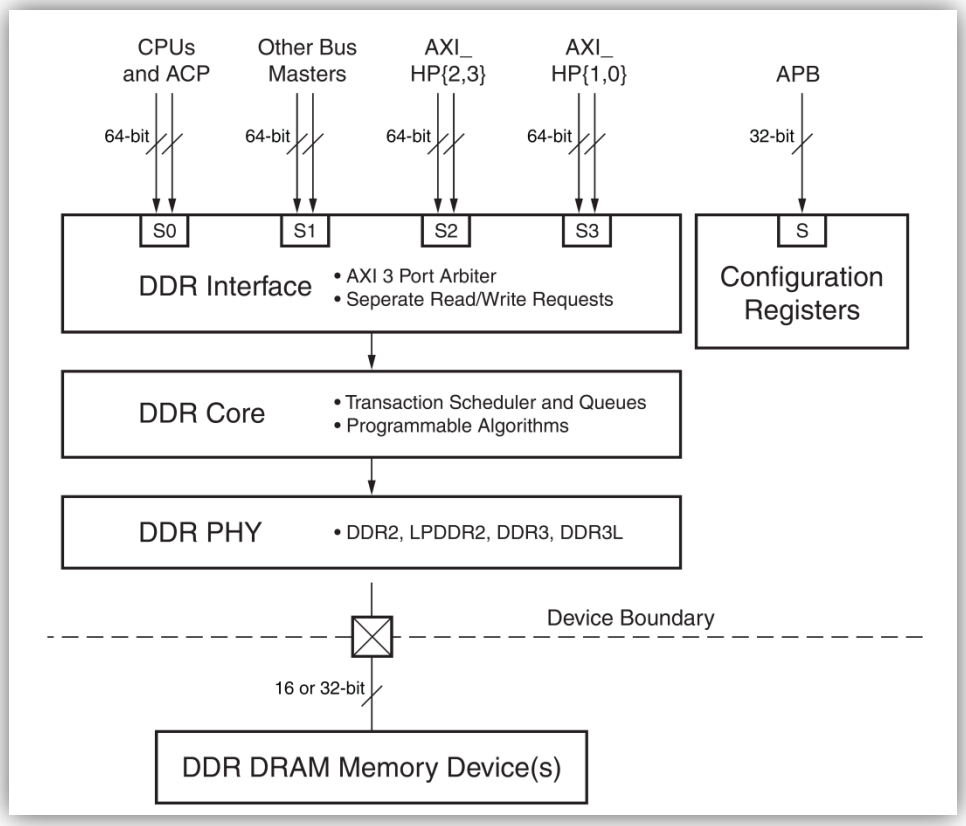

Figura 12. Diagrama de bloques del controlador de memoria DDR

Por otro lado, la mayoría de los módulos de PS soportan *clock gating*, que facilita la reducción del consumo de potencia. También se puede reducir el consumo de potencia disminuyendo la frecuencia de reloj de la unidad APU y de acceso a memoria DDR. Por su parte, los modos de *reset* soportados por PS se indican en la Tabla 5.

Tabla 5. Modos de *reset*

Tipo de *reset*	Descripción
Power-on Reset	Se corresponde con un *reset* global del sistema
External System Reset	Sirve para *resetear* la CPU y la lógica de depuración
Software Reset	Se trata de un *reset* que no afecta a todos los subsistemas de la arquitectura. Inicializa la CPU y la lógica de depuración. Además, permite programar el *reset* de los periféricos
System Debug Reset	Este *reset* es similar al anterior. Se activa desde la interfaz JTAG
Debug Reset	Inicializa la lógica de depuración. Se activa desde la interfaz JTAG

2.3.6. Lógica programable de los SoC 7Z020 y 7Z045

La lógica programable proporcionada por el SoC Zynq-7000 EPP 7z020, integrado en la plataforma Zynq-7000 ZC702, se indica en la Tabla 6. La lógica programable de la plataforma está basada en la lógica de FPGA Artix-7. Cabe destacar que la FPGA proporcionada por esta plataforma, y por la plataforma Zynq-7000 ZC706 soportan reconfiguración dinámica parcial. Por otra parte, en el dominio de potencia de la parte de la lógica programable se encuentran dos módulos conversores analógico-digital.

Tabla 6. Lógica programable en el SoC 7z020

Componente	Cantidad
Slices de lógica total	13,300
Slices de lógica tipo L	8,950
Slices de lógica tipo M	4,350
LUTs total	53,200
Flip-flops	106,400
Células lógicas	85,120
LUTRAM	1,088
SRL	544
Block RAM	140

La lógica programable proporcionada por el SoC Zynq-7000 EPP 7z045, integrado en la plataforma Zynq-7000 ZC706, se indica en la Tabla 7. La lógica programable de la plataforma está basada en la lógica de FPGA Kintex-7.

Tabla 7. Lógica programable en el SoC 7z045

Componente	Cantidad
Slices de lógica total	54,650
Slices de lógica tipo L	37,050
Slices de lógica tipo M	17,600
LUTs total	218,600
Flip-flops	437,200
Células lógicas	349,760
LUTRAM	4,400
SRL	2,200
Block RAM	545

A continuación se exponen las características de los componentes de la lógica programable de manera más detallada. Sus principales características son las siguientes:

- Bloques Lógicos Configurables. (CLB). Cada uno está compuesto por:
  - Dos *slices*, cada uno con dos LUTs.
  - Capacidad de memoria dentro de las LUTs.
  - Funcionalidades de registro y de registro de desplazamiento.
  - Sumadores en cascada.

- Cada LUT, de seis entradas, está formada básicamente por dos *flip-flops*, un multiplexor y lógica de acarreo.
- Bloques RAM de 36 KB. Se caracterizan por:
  - Puertos duales.
  - Palabras de hasta 72 bits de ancho.
  - El bloque puede ser configurado como dual de 18 KB.
  - Los puertos de los bloques RAM pueden programados con diferentes anchos: 32K × 1, 16K × 2, 8K × 4, 4K × 9, 2K × 18, 1K ×36 o 512 × 72.
  - Pueden ser usados para dar soporte a FIFO programables.
- *Slice* de DSP. El modelo de *slice* DSP es DSP48E1 [25]. Se dispone de hasta 220 *slice* DSP. Proporcionan una mejora en la velocidad y en la eficiencia en aplicaciones que procesen señales. Se basa en multiplicadores de 25x18 en complemento a 2 y acumuladores de 48 bits. Además hay presentes sumadores de 25 bits antes de los multiplicadores para optimizar las operaciones de aplicaciones con filtros simétricos. Asimismo, los acumuladores pueden ser utilizados como contadores en procesos de sincronización. La frecuencia máxima a la que funciona cada DSP es 741 MHz. En la Figura 13 se muestra la funcionalidad del *slice* DSP48E1. Además, opcionalmente se puede usar como ALU y pueden usarse características de *pipeling*.

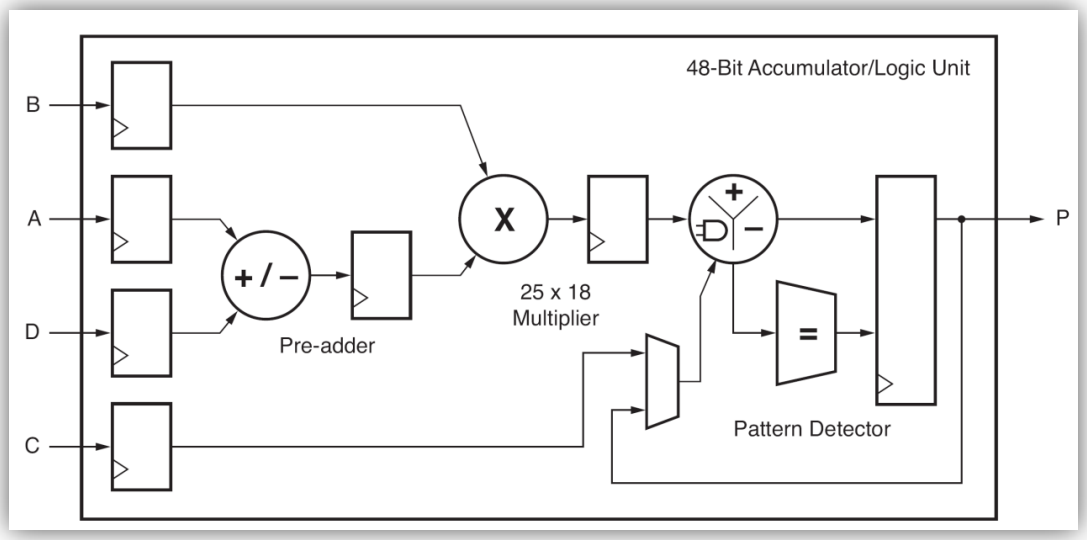

Figura 13. Funcionalidad del slice DSP48E1

- Gestión del reloj. El manejo de reloj en la lógica programable incluye *buffers* de alta velocidad y distribución de las líneas de reloj con el objetivo de tener un bajo *skew* y generación de reloj con bajo *jitter* además de filtrado de *jitter*. La distribución de reloj se realiza a través de seis tipos líneas de reloj diferentes enfocadas al cumplimiento con los requerimientos en cuanto a elevado *fanout,* baja propagación del retardo y un bajo *skew*. Cuenta con treinta y dos líneas globales de reloj que tienen un alto *fanout*. Respecto a las líneas de reloj locales, estas afectan a regiones que abarcan cincuenta *pines* de entrada/salida de señales de reloj y cincuenta CLBs. Los relojes de entrada/salida sirven a la lógica de entrada/salida y a los circuitos serializadores/deserializadores.

Por otro lado, todos los pines de entrada/salida se agrupan en bancos de cincuenta *pines* cada uno. La lógica de entrada/salida permite retrasar la entrada/salida de hasta 2,496 ns. Además, permite serialización y deserialización.

La lógica programable gestiona su propio reloj a partir de las señales proporcionadas por PS. Por otro lado, el sistema de *reset* es generado a partir de cuatro señales de *reset* proporcionadas por el sistema de procesamiento.

Cabe destacar que con el objetivo de reducir el consumo de potencia la lógica programable (PL) puede ser desconectada si no se necesita.

Conversor analógico-digital (XADC)

El módulo XADC es un módulo en el dominio de potencia de la parte de lógica programable de la arquitectura que contiene dos conversores analógico-digitales. Se trata de conversores de 12 bits y que pueden muestrear a 1 Msps. Su utilidad consiste en trabajar con sensores externos conectados a la plataforma.

Soportan depuración a través de interfaz JTAG y se pueden comunicar con PS mediante la interfaz PS-XADC y la interfaz AXI que conecta PS con PL.

2.4. Arquitectura *software*

El proceso de arranque de la plataforma consta de una serie de etapas que se describen a continuación. Por una parte se diferencia el arranque de PS del arranque de PL.

- *Bootloader*. El código de arranque tiene como misión arrancar la plataforma. En la etapa de preinicialización, se habilitan los *pines* implicados en los dominios de potencia, reloj y *reset* así como los relacionados con la configuración del espacio de memoria de arranque para la CPU, con el objetivo de posibilitar la alimentación del dispositivo y su arranque. Seguidamente se ejecuta un código de arranque almacenado en la memoria ROM *On-Chip*, que inicializa el *hardware* necesario y carga el gestor de arranque de la siguiente etapa en la memoria RAM *On-Chip*. La función principal de este gestor de arranque (u-boot) consiste en cargar en memoria DDR la imagen del *kernel* de Linux e iniciar su ejecución, pasando el control al mismo. Por último, una vez ejecutada la imagen del *kernel* se monta el sistema de ficheros raíz.

  PL es alimentada si es necesario, esto es, si va a ser configurada. Tanto el gestor de arranque de segundo nivel como el *kernel* y el sistema de ficheros pueden residir en uno de los siguientes dispositivos: Quad-SPI, NAND, NOR y Tarjeta SD.

El arranque puede producirse en modo seguro o no seguro y en modo JTAG, que es un modo no seguro que permite el acceso a los puertos de depuración. En el caso de un arranque en modo no seguro, si no se requiere la lógica programable esta no se alimenta. En cambio, si se arranca en modo seguro o JTAG sí se alimenta PL. En el modo seguro normalmente el gestor de arranque de primer nivel y el *bitstream* que configura PL están encriptados. La encriptación se realiza mediante la herramienta de Xilinx Bootgen [26].

- *Kernel* Xilinx de Linux. Se trata de un *kernel* de Linux configurado por Xilinx para dar soporte a los distintos módulos IP presentes en la plataforma. Mediante los *drivers* que proporciona da soporte a la utilización de los módulos de la plataforma.
- Sistema de ficheros raíz. Por último se monta el sistema de ficheros de la distribución.

Por otro lado, el arranque de PL consiste en cuatro etapas:

1. *Start-up*. Se corresponde con los primeros instantes en que la lógica programable es alimentada.
2. Inicialización. En esta etapa se borran las celdas SRAM.
3. Configuración. Se corresponde con el estado en que se integra el diseño *hardware* en la lógica programable.
4. Activación. La lógica programable se encuentra configurada y lista para ser usada.

La arquitectura (*hardware* y *software*) de la plataforma se presenta en la Figura 14.

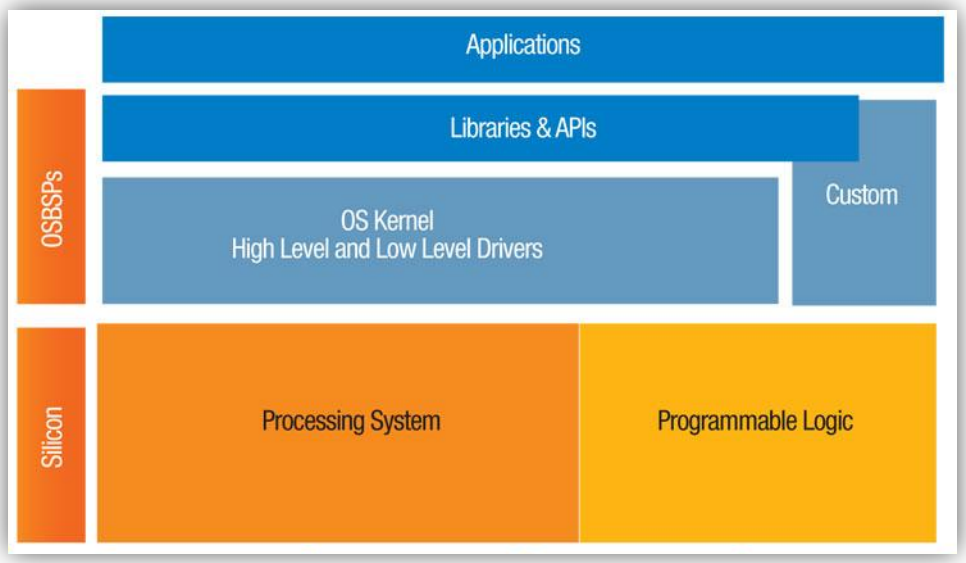

Figura 14. Arquitectura de la plataforma de desarrollo

2.5. Conclusiones

Los SoC de la serie Zynq-7000 de Xilinx proporcionan un sistema de procesamiento (PS) y lógica programable (PL). El sistema de procesamiento no se diferencia entre los distintos SoC de la serie, mientras que sí existen variaciones en cuanto a los recursos de la lógica programable y recursos de entrada/salida.

De los seis SoC de esta serie, cuya tecnología de fabricación es 28nm, la lógica programable de los dispositivos 7z010, 7z015 y 7z020 se basa en la tecnología Artix-7, mientras que en los otros tres, 7z030, 7z045 y 7z100, esta se basa en la tecnología Kintex-7, estos últimos tienen un mayor consumo de potencia.

Estos SoC pueden se integran en plataformas, que proporcionan recursos de entrada/salida para el dispositivo, como pueden ser puertos o conectores de expansión, además de recursos adicionales, como memorias.

Xilinx proporciona plataformas para algunos de estos SoC. Específicamente, se trata en este capítulo las plataformas Zynq-7000 ZC702, que integra el SoC 7z020 y Zynq-7000 ZC706, que integra el SoC 7z045. La lógica programable de la plataforma Zynq-7000 ZC706 proporciona mayor cantidad de recursos.

Estas plataformas proporcionan, entre otros, conector HDMI, conector de tarjeta SD, memoria Flash Quad-SPI, memoria DDR3, conector miniUSB, módulo de potencia, reloj de tiempo real y conector RJ45 de *Ethernet*.

Capítulo 3: Metodología de diseño

3.1. Introducción

En el desarrollo de cualquier investigación científica se requiere de una metodología [27], entonces, seguir un conjunto de métodos ya establecido por la comunidad científica a través de los cuales poder desarrollar la misma. Por tanto, en este capítulo se describen la metodología de diseño, esto es, los lenguajes, herramientas y tecnologías empleadas.

3.2. Metodología de diseño

Tanto en los diseños *hardware* como *software* la metodología de diseño se divide en tres fases básicas: especificación, diseño y verificación [28], como se muestra en la Figura 15. En la fase de especificación se corresponde con el análisis de la aplicación a desarrollar, para así definir las especificaciones del diseño en su conjunto, que pueden ser funcionales, temporales o de otro. A continuación se define la tecnología a utilizar, las fases de desarrollo o la partición del diseño en una parte *hardware*, para aquellas funciones que llevan asociado un cómputo elevado, y una parte *software*, para aquellas funciones que no lleven asociado un cómputo elevado o que requieran flexibilidad.

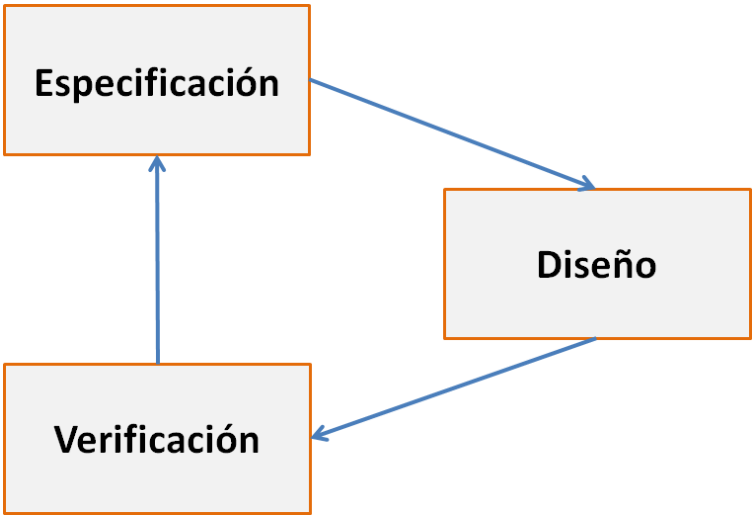

Figura 15. Flujo de diseño simple

Con el fin de cumplir con las especificaciones definidas, una vez se completa el diseño, se debe verificar que cumple con las mismas, realizando las modificaciones necesarias, principalmente en las primeras fases del ciclo de diseño en caso de que las especificaciones no se cumplan. En lo concerniente a este Trabajo Fin de Máster se dan por definidas las especificaciones, teniéndose que cumplir con una frecuencia de funcionamiento de 100 MHz.

La fase de diseño se divide en síntesis lógicas y síntesis física (implementación), estando fuera del alcance de este Trabajo Fin de Máster la etapa de programación de la lógica programable, al igual que la síntesis de alto nivel, realizada en el proyecto PCCMUTE.

3.3. Herramientas

La principal herramienta usada en el desarrollo de este Trabajo Fin de Máster es Vivado Design Suite [3], propiedad de Xilinx. También se utiliza la herramienta Synplify Premier DP [29] de Synopsys. La utilización de esta herramienta tiene por objeto realizar comparaciones respecto a los resultados obtenidos con Vivado Design Suite.

Vivado Design Suite integra el flujo de diseño desde el nivel RTL en la misma herramienta, sin que sea necesaria la utilización de diferentes herramientas según la etapa del flujo de diseño en la que se esté presente. Para realizar la síntesis de alto nivel Xilinx proporciona la herramienta Vivado_HLS [4], que soporta síntesis de alto nivel para los lenguajes C, C++ y SystemC.

En el nivel RTL Vivado Design Suite soporta los lenguajes Verilog y VHDL. El diseño proporcionado se basa en ficheros Verilog [30]. Cabe destacar que, aunque en este diseño no será necesario realizar simulaciones de comportamiento temporal debido a que estas simulaciones se corresponden con la parte de desarrollo del sistema dentro del proyecto PCCMUTE, este tipo de simulación es soportada para diseños Verilog.

Una característica interesante de Vivado Design Suite es que ofrece la posibilidad de realizar el flujo de diseño incluyendo la característica de reconfiguración parcial de la FPGA sobre la que se implemente.

Por otra parte, Vivado Design Suite proporciona la posibilidad de ser ejecutado en modo gráfico o directamente desde el terminal, denominado modo *batch*, mediante comandos Tcl [31].

Asimismo, una vez se está en modo gráfico, se pueden utilizar los comandos Tcl desde la consola Tcl que proporciona Vivado en la GUI, como se refleja en la Figura 16.

Figura 16. Consola para comandos Tcl

Por otra parte, los hilos de síntesis e implementación que hayan sido creados se muestran en la ventana *Design Runs*. Además, en esta ventana se indica si alguno esté ejecutándose. Por otro lado, se dispone de la pestaña Log, que permite hacer un seguimiento de la evolución del proceso de síntesis o implementación. Asimismo, se dispone de una barra de estado de procesos en ejecución en la esquina superior derecha de la ventana de la herramienta. En la Figura 17 se muestra mientras que ejecuta un proceso de síntesis lógica.

Figura 17. Barra de estado de procesos

El número de núcleos sobre los que se ejecuta la síntesis lógica o la implementación puede seleccionarse mediante la opción *Number of jobs* en el momento de ejecutar el proceso. Esta opción se encuentra señalada en color rojo en la Figura 18, en la que se muestra la selección de cuatro *cores* para la ejecución de un proceso de síntesis lógica.

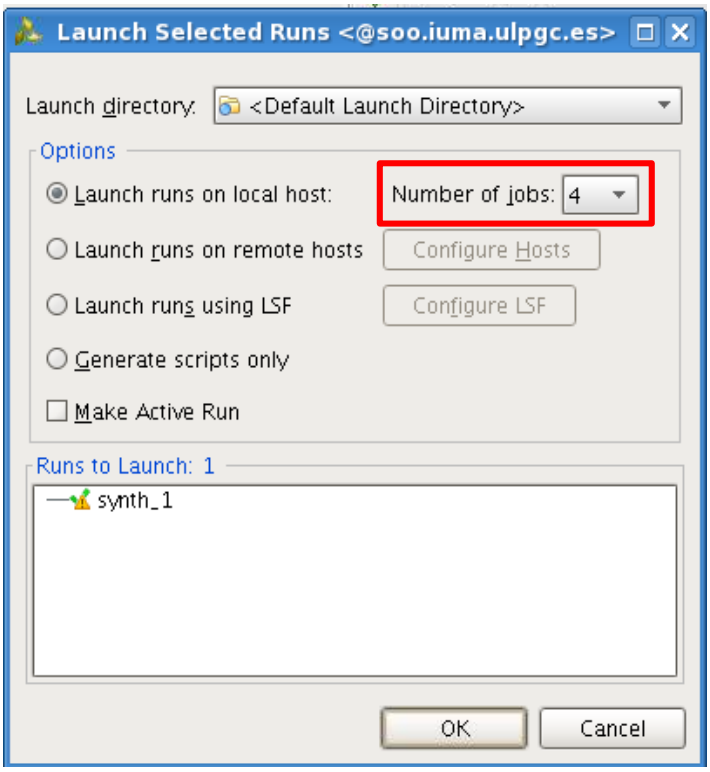

Figura 18. Selección del número de *cores*

Respecto a los informes post-síntesis lógica y post-implementación, Vivado Design Suite permite visualizar distintos tipos de informes. Entre estos informes se encuentra el informe temporal, de utilización de recursos, de potencia, de ruido o acerca del árbol de reloj. Por otra parte, la herramienta también permite realizar una comprobación de las reglas de diseño (DRC), tanto a nivel RTL, como después de la síntesis lógica e implementación.

3.4. Plataformas

En el presente Trabajo Fin de Máster se han usado dos plataformas basadas en dispositivos Zynq. Por una parte la plataforma Zynq-7000 EPP ZC702 soporta un dispositivo Zynq basado en Artx-7. De igual modo se ha usado la plataforma Zynq-7000 EPP ZC706 que utiliza el dispositivo XC7Z045 basado en Kintex-7. Se realiza la síntesis lógica del diseño para ambas plataformas.

Por otra parte, la plataforma sobre la que se desarrolla el proyecto PCCMUTE es el modelo Virtex5 FX70T [32, 33]. La tecnología de fabricación de las plataformas Zynq-7000 es CMOS 28 nm, mientras que en el caso de la Virtex5 es de CMOS 65 nm.

3.5. Diseño del proyecto de investigación PCCMUTE

El diseño del que se parte en este Trabajo Fin de Máster se corresponde con el desarrollado dentro del proyecto de investigación PCCMUTE [34]. El objetivo final de PCCMUTE es el desarrollo de una implementación del *Deblocking Filter* del decodificador de vídeo de H.264/SVC [35] que consiga reducir el consumo de potencia en terminales móviles que usen decodificación H.264. En la Figura 19 se muestra el diagrama de bloques de dicho diseño, que fue desarrollado sobre lógica programable. El modelo funcional se basa en el modelo del *Deblocking Filter* para AVC utilizando el código fuente de OpenSVC [36] como referencia.

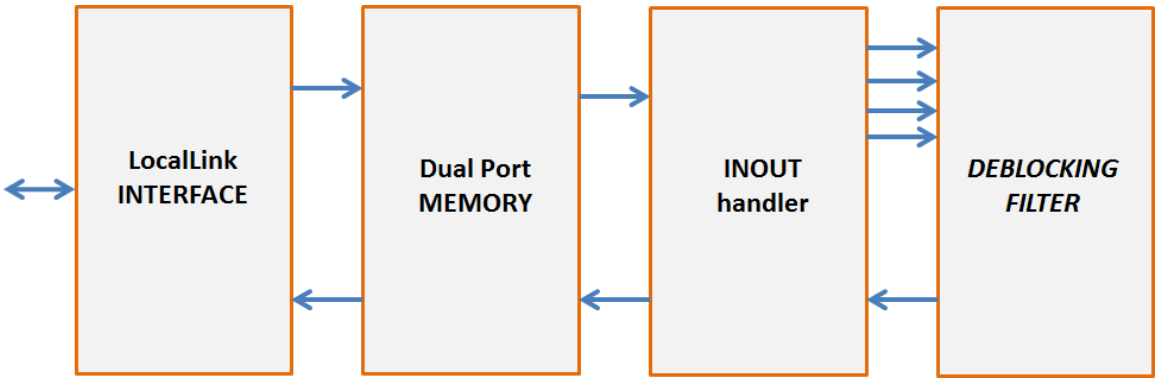

Figura 19. Diagrama de bloques del diseño PCCMUTE

Este diseño consta de los siguientes módulos:

- Interfaz LocalLink. El diseño se comunica con el exterior a través de una interfaz LocalLink. En realidad este módulo se corresponde con dos interfaces LocalLink, ya que se dispone de dos interfaces LocalLink, una para cada sentido de comunicación. Esto se debe a que el diseño recibe información que es procesada en última instancia por el *Deblocking Filter* y la información procesada se transmite hacia el exterior.

- Memoria de doble puerto. Está formado por bloques de memoria RAM (BRAM). Se encargan de almacenar muestras de píxeles de imagen además de información necesaria para el filtrado por parte del *Deblocking Filter*. Este módulo está formado por cuatro bloques de memoria.

- *Gestor* de entrada/salida. Su utilidad consiste en manejar peticiones relacionadas con el *Deblocking Filter* y operaciones de lectura y escritura en la memoria principal.

- *Deblocking Filter* del decodificador de vídeo H.264/SVC. Este módulo dispone de cinco interfaces: para las muestras no filtradas (*samples*), para las muestras de macrobloques vecinos (*neighbours*), de vectores de movimiento (*motion_vectors*), de control (*hw_control*) y la última de salida para las muestras filtradas (*filtered*). El diagrama de bloques del *Deblocking Filter* se muestra en la Figura 20. Además de las interfaces, dispone de cuatro módulos principales [37]: PARAM_BS, que calcula el parámetro *Boundary Strength*, PARAM_CLIP, que calcula los parámetros límite α y β para cada uno de los ejes de filtrado tanto de luminancia como de crominancia, LUMA_CORE, encargado de aplicar las ecuaciones de filtrado para la componente de luminancia, y CHROMA_CORE, que aplica las ecuaciones de filtrado para las dos componentes de crominancia.

3.6. Adaptación de Local-Link a AXI4-Stream

La interfaz del diseño obedece al protocolo LocalLink [38]. No obstante, la plataforma utilizada en este TFM no soporta esta interfaz, por lo que para la conexión del diseño con el bloque de procesamiento se necesita otro tipo de interfaz. La comunicación entre la lógica programable de la plataforma y el sistema de procesamiento se debe realizar por medio de interfaz AXI4.

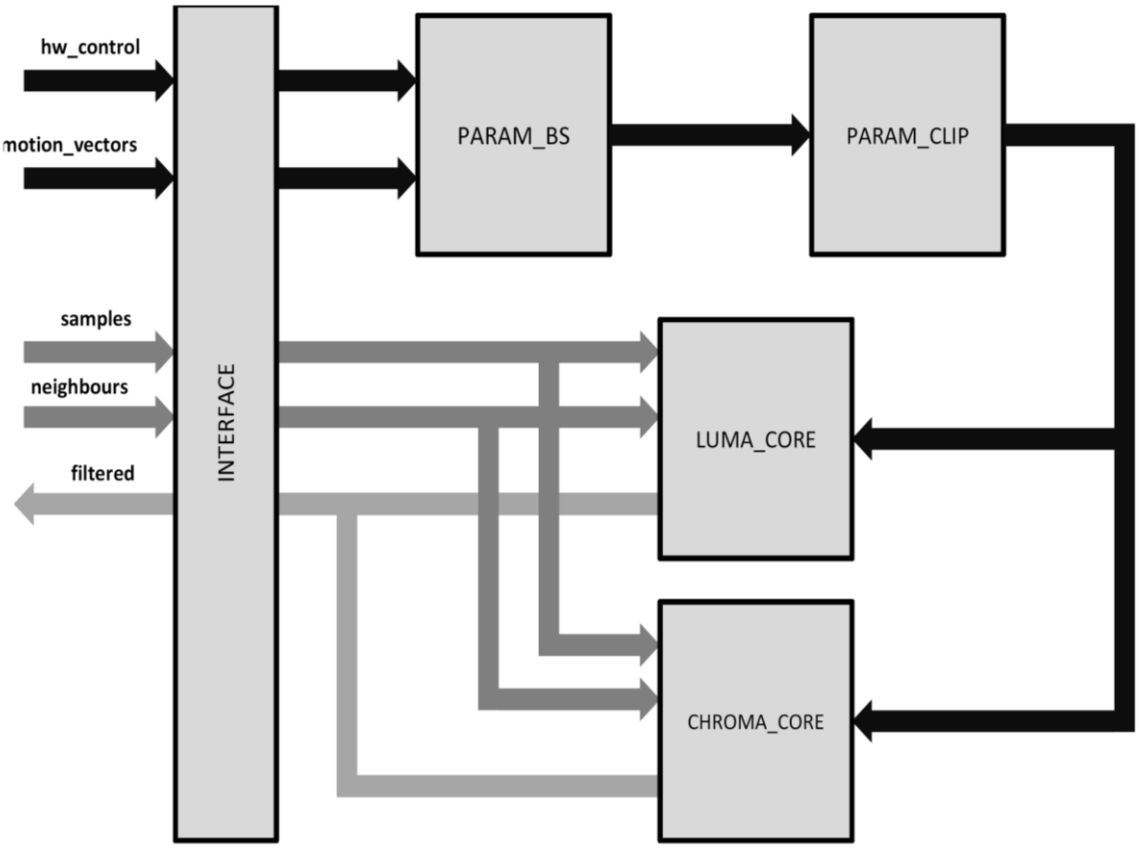

Figura 20. Diagrama de bloques del *Deblocking Filter*

Por tanto, se diseña un *wrapper* que adapte la interfaz LocalLink a AXI4. En particular, el *wrapper* diseñado adapta el protocolo LocalLink a AXI4-Stream [39]. En la documentación proporcionada por Xilinx se dispone de la correspondencia entre ambos protocolos [40]. Dicha correspondencia se refleja en la Tabla 8.

Tabla 8. Correspondencia LocalLink-AXI4-Stream

Señal de LocalLink	Descripción	Señal de AXI4-Stream
CLK	Señal de reloj	ACLK
RST_n	Señal de *reset*	ARESETN
DATA	*Bus* de datos	TDATA
SRC_RDY_n	Indica que la fuente está preparada para transmitir datos	TVALID
DST_RDY_n	Indica que el destino está preparada para recibir datos	TREADY

Debido a que en el diseño se dispone de dos interfaces LocalLink, una para transmisión y otra para recepción, se diseñan dos *wrapper*, uno para cada interfaz LocalLink. La Figura 21 presenta la arquitectura del diseño completo incluyendo los dos *wrappers*. La adaptación de los *wrappers* se centra únicamente en las señales de control, puesto que ni la señal de reloj, ni los *buses* de datos ni la señal de *reset* (activa a nivel bajo en ambos) varían.

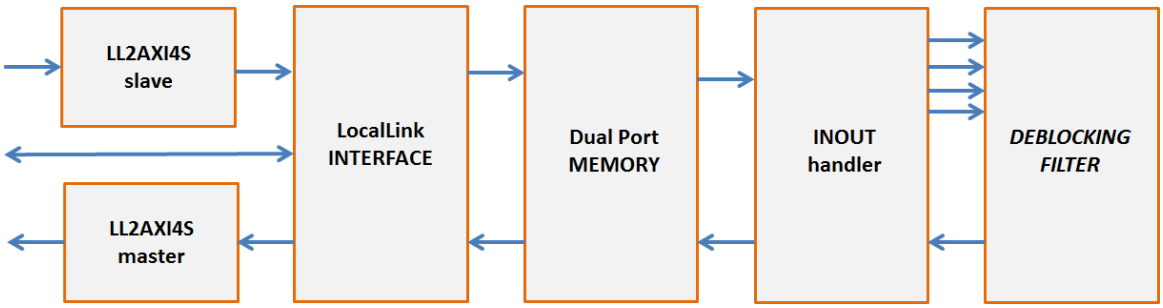

Figura 21. Diagrama de bloques del diseño completo

En cambio, las señales de control en LocalLink SRC_RDY_n y DST_RDY_n son activas a nivel bajo, siendo sus equivalentes de AXI4-Stream activas a nivel alto. Esto motiva que cada *wrapper* conste de dos inversores. Por tanto, los *wrapper*, que son módulos combinacionales, se encargan de adaptar señales de control entre ambos protocolos. En la Tabla 9 se muestra la nomenclatura de las señales usadas por el *wrapper* para transmisión (LL2AXI4S slave) mientras que en la Tabla 10 se muestra la nomenclatura empleada para las señales del *wrapper* de recepción (LL2AXI4S master).

Tabla 9. Señales del *wrapper* esclavo

Señal LocalLink	Señal AXI4-Stream
ll_src_rdy_n_tx	tvalid_tx
ll_dst_rdy_n_tx	tready_tx

Tabla 10. Señales del *wrapper* maestro

Señal LocalLink	Señal AXI4-Stream
ll_src_rdy_n_rx	tvalid_rx
ll_dst_rdy_n_rx	tready_rx

La Figura 22 se corresponde con el esquemático del *wrapper* para recepción obtenido desde Vivado Design Suite, que se corresponde con la salida de datos del diseño. El esquemático del *wrapper* para transmisión es equivalente. Puede observarse que se compone de dos inversores, uno para cada señal de control, asignándose una LUT para cada inversor. En este *wrapper* las señales de entrada son *tready_tx*, que se corresponde con uno de los puertos de entrada del sistema, y la señal interna *ll_src_rdy_n_rx*. Por su parte, las señales de salida son *tvalid_rx*, que se corresponde con uno de los puertos de salida del sistema, y la señal interna *ll_dst_rdy_n_rx*.

3.7.Flujo de diseño propuesto

Una vez definido el lenguaje, las herramientas y las tecnologías a utilizar, se concreta el flujo de diseño. En primer lugar se adapta el diseño de PCCMUTE a nivel RTL a las necesidades de la plataforma, esto es, añadir *wrappers* al diseño para la adaptación de las interfaces LocalLink a AXI4-Stream.

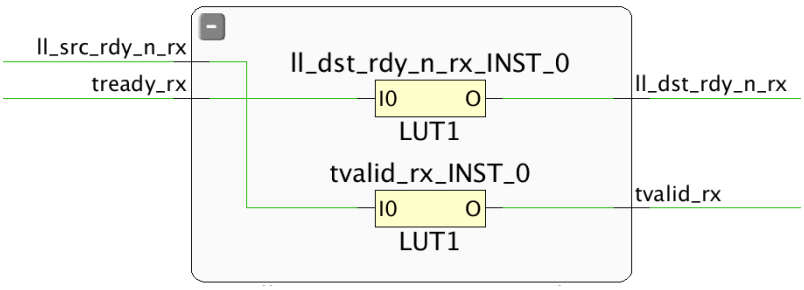

Figura 22. Esquemático del *wrapper* para recepción

A continuación se realiza la síntesis lógica mediante las estrategias *bottom-up* y *top-down* con las herramientas Vivado Design Suite y Synplify Premier DP para comparar resultados.

El siguiente paso en el flujo de diseño consiste en la implementación a partir del *netlist* obtenido de la síntesis lógica. La implementación también se realiza con las estrategias *bottom-up* y *top-down* mediante las herramientas Vivado Design Suite y Synplify Premier DP, igualmente comparando los resultados.

Se comparan los resultados de implementación para los módulos principales que componen el *Deblocking Filter* entre las dos herramientas y los obtenidos en el proyecto de investigación PCCMUTE, siguiendo la misma estrategia de síntesis que en este, *top-down*.

Por último, se genera el sistema completo mediante bloques IP por medio de la herramienta Vivado IP Integrator, integrada en Vivado Design Suite.

3.8.Conclusiones

En este capítulo se presentan las herramientas utilizadas y la metodología asociada a este Trabajo Fin de Máster. Se parte del diseño del proyecto de investigación PCCMUTE en el nivel RTL. El lenguaje en el que está definido dicho proyecto es Verilog.

Desde el nivel RTL se realiza la síntesis lógica y se obtiene el netlist del diseño. El siguiente paso consiste en realizar la implementación partiendo de dicho *netlist*.

La síntesis lógica y la implementación se llevan a cabo con la herramienta Vivado Design Suite, herramienta en la que se centra este Trabajo Fin de Máster. Asimismo, ambas etapas también se realizan con la herramienta Synplify Premier DP, con el fin de realizar comparaciones entre los resultados de ambas herramientas. Finalmente, se presenta el flujo de diseño propuesto para este Trabajo Fin de Máster.

Capítulo 4: Síntesis lógica

4.1. Introducción

La síntesis lógica se realiza teniendo como base el nivel RTL del diseño. En este Trabajo Fin de Máster se dispone de los ficheros del diseño en lenguaje Verilog. Previamente al proceso de síntesis lógica se debe comprobar que el diseño RTL es adecuado, asegurando por un lado que cumple con el comportamiento funcional del diseño mediante las simulaciones correspondientes y por otro lado realizando una comprobación DRC. Para este Trabajo Fin de Máster no es necesario realizar esa comprobación ya que se parte de un diseño ya validado mediante prototipado en el proyecto PCCMUTE.

Una vez el funcionamiento del diseño a nivel RTL es correcto se pasa a realizar la síntesis lógica. En este paso se fundamenta en transformar un diseño desde el nivel RTL a nivel de puertas lógicas, proporcionando un *netlist* con las conexiones entre las mismas.

En este capítulo se comparan los resultados obtenidos mediante las estrategias *top-down* y *bottom-up* así como las diferencias al realizar la síntesis lógica en Vivado Design Suite y Synplify Premier DP.

4.2. Nivel RTL

La vista estructural del nivel *top* diseño completo en el nivel RTL proporcionada por Vivado Design Suite se presenta en la Figura 23. Los dos *wrappers* se encuentran conectados al módulo *top* del diseño de partida, cuya jerarquía interna se muestra en la Figura 24. Para mostrar esta vista se selecciona la opción *Elaborated Design* dentro del menú *RTL Analysis* de la herramienta. En la misma puede verse que están presentes los cuatro módulos de mayor jerarquía dentro de dicho diseño: la interfaz LocalLink, el bloque de memoria, el gestor de entrada/salida y el *Deblocking Filter*. Pulsando en la opción se abre el esquemático del diseño.

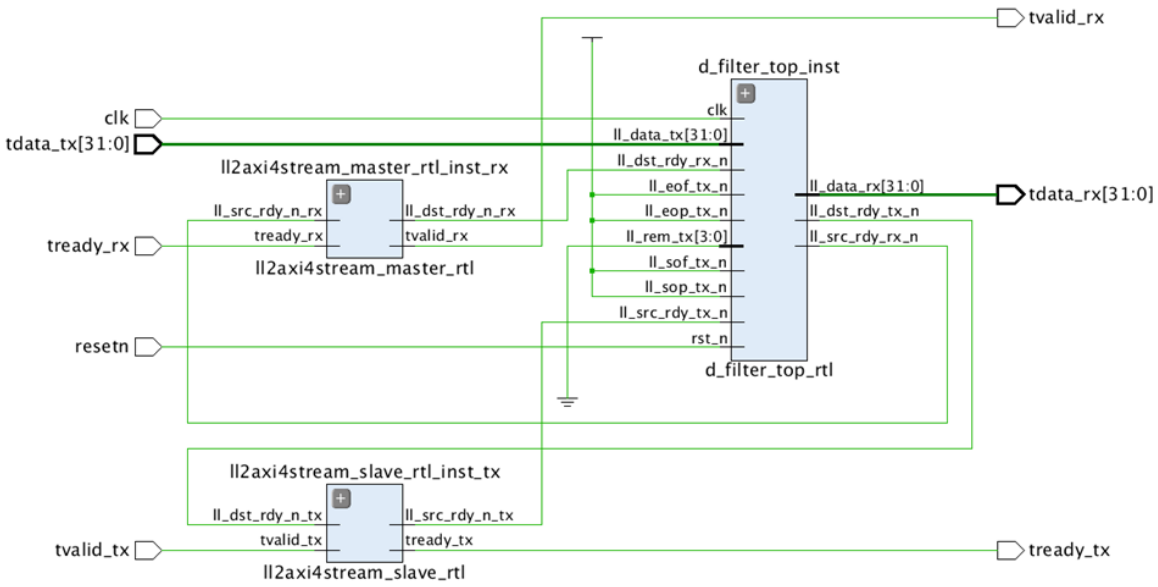

Figura 23. Esquemático RTL

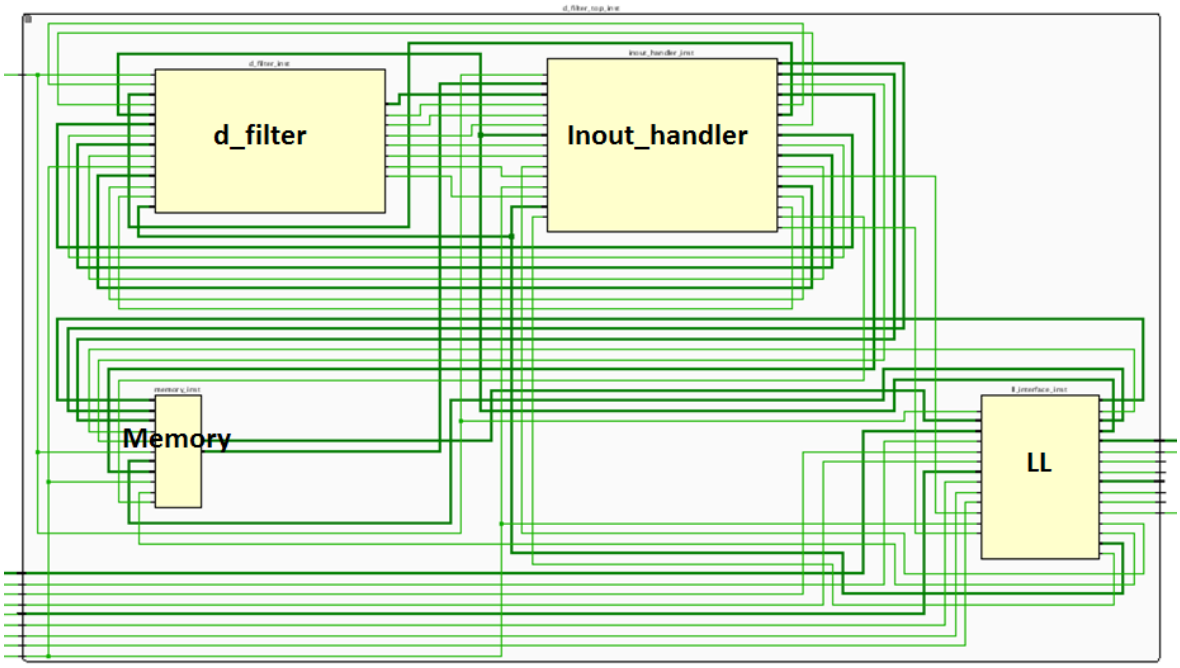

Figura 24. Esquemático del diseño PCCMUTE

4.3. Estrategias de síntesis

Las estrategias de síntesis (en diferentes etapas del diseño, por ejemplo, síntesis de alto nivel y síntesis lógica o síntesis física) que pueden seguirse son fundamentalmente *top-down* y *bottom-up*.

La estrategia *bottom-up*, cuya secuencia genérica se indica en la Figura 25 se basa en la síntesis en primer lugar de los módulos de menor nivel en la jerarquía, sintetizándose a continuación los módulos que estén por encima, integrando los módulos nivel inferior como cajas negras, y así sucesivamente hasta llegar al nivel top del diseño.

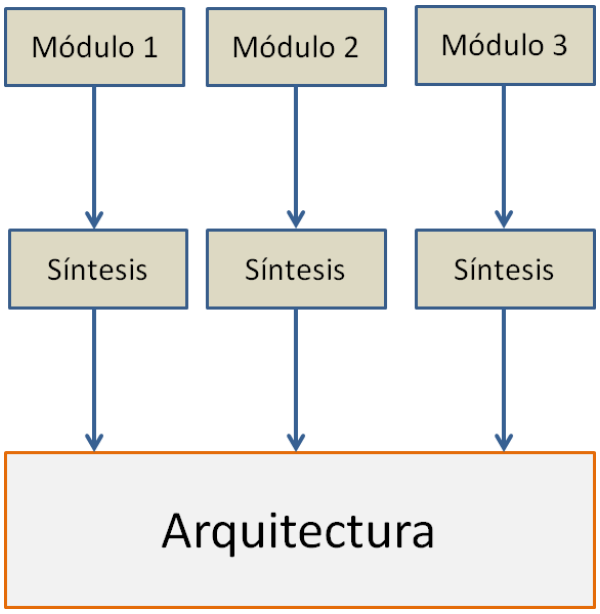

Figura 25. Estrategia *bottom-up*

En la Figura 26 se muestra el concepto de la estrategia *top-down*. En esta estrategia, se sintetizan todos los módulos del diseño en el mismo proceso, lo que proporciona mayor flexibilidad en la optimización del diseño completo.

La estrategia *bottom-up* permite sintetizar en paralelo diferentes subsistemas reduciéndose de esta manera el tiempo de cómputo respecto a la estrategia top-down. Además, se cuenta con la ventaja de que en el caso de realizar modificaciones en uno de los módulos, se puede sintetizar este sin que sea necesario volver a sintetizar todo el diseño.

Por el contrario, una estrategia *top-down* permite obtener mejores prestaciones debido precisamente a la mayor flexibilidad en la optimización del diseño en su conjunto, sin considerar módulos como cajas negras con unas prestaciones definidas previamente. Ambas estrategias pueden ser combinadas con el fin de obtener un equilibrio entre calidad de resultados y tiempo de cómputo.

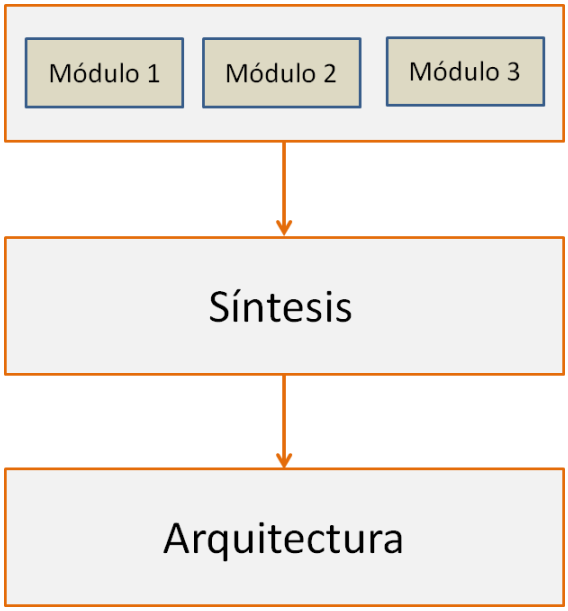

Figura 26. Estrategia *top-down*

Vivado Design Suite contempla las estrategias *top-down* y *bottom-up* [41]. En ambos casos se deben declarar los módulos de niveles inferiores como fuera de contexto mediante la opción *out of context*. Los módulos declarados como tal son considerados como cajas negras por el nivel superior. De este modo la estrategia *top-down* tiene la ventaja de poder agilizar los tiempos de diseño, ya que los módulos fuera de contexto pueden ser sintetizados de manera independiente, esto es, pueden ser sintetizados en paralelo. Además, un cambio en uno de los módulos no provoca que se tenga que sintetizar el diseño completo.

Por un lado se puede ejecutar una estrategia *bottom-up*, en la que la síntesis de los módulos de niveles inferiores desconoce las restricciones temporales y físicas del sistema general. Por tanto, se debe definir las restricciones para cada módulo.

Por otro lado, si se sigue una estrategia *top-down* los módulos de niveles inferiores también deben declararse como fuera de contexto. En cambio, el nivel *top* del diseño se encarga de definir

las restricciones temporales y físicas, que son utilizadas para guiar la implementación de los módulos declarados como fuera de contexto.

Los submódulos pertenecientes a módulos declarados como fuera de contexto son sintetizados siguiendo una estrategia *top-down* pura, sino que se basa en asignar módulos de niveles inferiores como fuera de contexto mediante la opción *out of context*, de tal manera que estos se sintetizan de manera independiente y son considerados por el nivel *top* como cajas negras. En este Trabajo Fin de Máster se han declarado como módulos fuera de contexto, la interfaz LocalLink, la Memoria de puerto dual, el Gestor de entrada/salida, el *Deblocking Filter* y ambos *wrappers*. Dentro de estos módulos asignados como *out of context* sí existen niveles inferiores, por lo que para dichos módulos sí que se aplica una estrategia *top-down*.

Por otra parte, Vivado Design Suite permite utilizar una estrategia *bottom-up* dentro de un único proyecto, en lugar de tener que crear un proyecto para cada submódulo y uno para el módulo *top* del diseño que integre los proyectos de los submódulos. Por su parte, siguiendo una estrategia *top-down* es también es necesario únicamente crear un proyecto. Por tanto, para ambas estrategias Vivado permite crear un solo proyecto desde el que trabajar.

Para realizar síntesis *bottom-up* en Vivado, debe seleccionarse la opción *Set Out of Context* para cada subsistema que se pretenda sintetizar independientemente. No obstante, cada bloque está formado por distintos submódulos, siguiéndose entonces dentro de cada bloque una estrategia *top-down*. El módulo *top* del diseño debe ser sintetizado una vez hayan sido sintetizados todos los módulos que se encuentren en niveles jerárquicos inferiores. Al sintetizar el módulo *top* del sistema también se sintetiza el *top* del diseño de PCCMUTE, en el que se interconectan los submódulos que lo componen. En esta última síntesis, dichos submódulos, al igual que los *wrappers*, son considerados cajas negras. En la Figura 27 se muestra cómo se asigna el módulo INOUT_HANDLER como *out of context*.

El módulo de memoria ya está asignado como fuera de contexto, lo que se indica con un rectángulo naranja. Cabe destacar que al declarar que un módulo sea considerado como fuera de contexto no se insertan buffers I/O dentro del módulo. Asimismo, se crean dos hilos, uno para la síntesis lógica y otro para la implementación del mismo.

En el caso de asignar un módulo fuera de contexto al utilizar la estrategia *bottom-up* no se heredan las restricciones del nivel *top*. Por tanto, es necesario definir las restricciones de cada módulo que se sintetice e implemente de manera independiente. La asociación de las restricciones a un módulo determinado se realiza mediante comandos Tcl.

Por otra parte, en la herramienta Synplify Premier DP para definir la estrategia *bottom-up* se crean subproyectos de los módulos que se sinteticen de manera independiente del nivel *top* del diseño.

4.4. Opciones de síntesis lógica

Vivado Design Suite proporciona diferentes opciones de síntesis lógica con el fin de configurarla para el diseño que se pretende sintetizar. Algunas de estas se encuentran en la Tabla 11 [42].

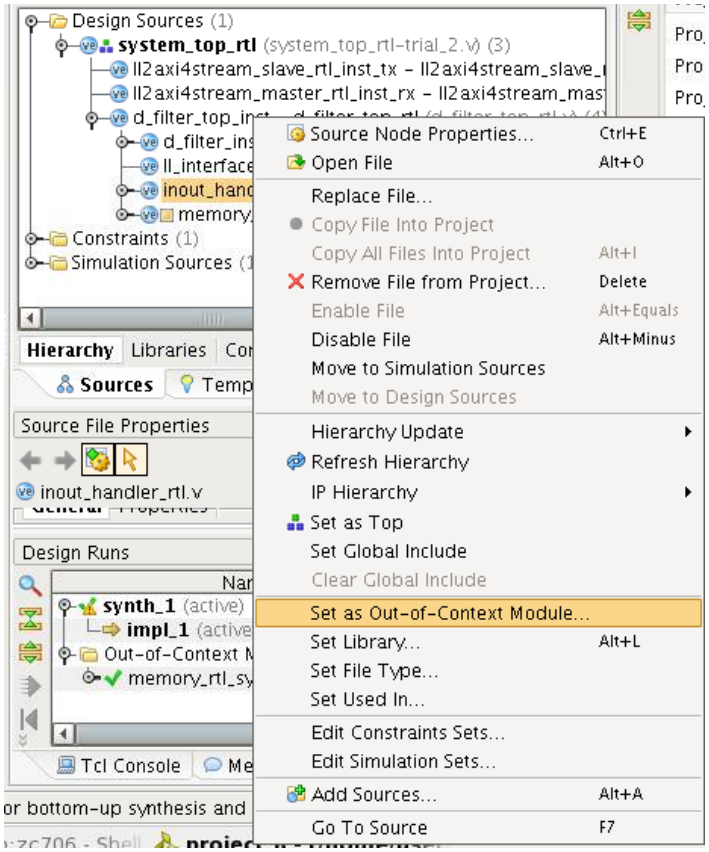

Figura 27. Declaración de módulo como *Out-of-Context*

Tabla 11. Opciones de síntesis lógica

Opción	Descripción
flatten_hierarchy	Permite romper la jerarquía del diseño, de tal manera que no se mantenga necesariamente la jerarquía del nivel RTL. Se aplica durante el mapeo de las LUT
Full	Se indica a Vivado que únicamente el nivel top del diseño esté en un nivel superior
Rebuilt	Permite romper la jerarquía aunque se toma como referencia la jerarquía del nivel RTL, por lo que la jerarquía final será similar a la del nivel RTL
None	Se deshabilita esta opción
directive	Permite seleccionar la opción de optimizar en tiempo de ejecución
resource_sharing	Permite que diferentes señales compartan operadores aritméticos, reduciéndose el área de utilización de un diseño
On	Activación de esta opción
Auto	Se permite la compartición de recursos, pero esta dependerá de las condiciones temporales del diseño
Off	Desactivación de esta opción
fanout_limit	Indica el límite máximo de *fanout* soportado
keep_equivalent_registers	Previene la unión de registros que disponen de la misma lógica a su entrada

Opción	Descripción
max_bram	Permite indicar un número máximo de bloques BRAM a usar en el diseño a sintetizar. Con el valor -1 se indica que se pueden utilizar todos los bloques BRAM disponibles
max_dsp	Permite indicar un número máximo de DSP a usar en el diseño a sintetizar. Mediante el valor -1 se indica que pueden usarse todos los DSP disponibles
fsm_extraction	Sirve para controlar la codificación de las máquinas de estado de un diseño. El valor "auto" indica a la herramienta que seleccione la codificación más adecuada para las FSM
mode out_of_context	Permite declarar un módulo como fuera de contexto. Asimismo, asegura que no se inserten *buffers* de entrada/salida (IOB) de la FPGA para los uertos del módulo. Se introduce desde el menú *More Options*

Si el límite de *fanout*, a través de la opción *fanout_limit* se pone en un valor muy bajo los resultados serán peores debido a que el *fanout* soportado por los módulos es menor, con lo que se necesita utilizar más lógica.

Si en la opción *directive* se selecciona optimizar en tiempo de ejecución los resultados obtenidos son mejores.

Por otra parte, si se desactiva la opción *keep_equivalent_registers* se utiliza más lógica debido a que ya no se unen registros con la misma lógica de entrada.

Se activa la opción *resource_sharing* debido a que la utilización de recursos se reduce a cambio de disminuirse la frecuencia. No obstante, se cumple con los 100 MHz de especificación.

4.5. Restricciones

Se necesitan incluir restricciones tanto temporales como de área para guiar el proceso de síntesis lógica e implementación. Se han definido por tanto dos tipos de restricción: restricciones temporales y restricciones físicas.

Las restricciones en Vivado Design Suite se imponen en los ficheros de restricción, que tienen asociada la extensión XDC. Dentro de estos ficheros, las restricciones se declaran mediante comandos Tcl.

La herramienta permite indicar si un fichero de restricciones se utiliza para la etapa de síntesis lógica, implementación o para ambas. En modo gráfico se señala en las propiedades del fichero. Un ejemplo de los comandos Tcl a utilizar se muestra seguidamente, donde constraints.xdc es el fichero de restricciones:

```
set_property used_in_synthesis true [get_files constraints.xdc]

set_property used_in_implementation true [get_files constraints.xdc]
```

Las restricciones se toman en cuenta para las etapas en las que se asigne el valor *true*, y no válidas si se asigna *false*.

Por otra parte, debido a que en la estrategia *bottom-up* las restricciones no se heredan, se debe crear un fichero de restricciones que contenga las restricciones del módulo a sintetizar. Una vez hecho esto y asignado el módulo correspondiente como fuera de contexto, se debe asociar el fichero de restricciones creado con el módulo.

Primero se crea un conjunto de ficheros al que se le asocia únicamente el fichero de restricciones para el módulo mediante el comando Tcl `add_files`. A continuación se asignan las propiedades de este fichero señalando que se asocian a un módulo fuera de contexto a través del comando Tcl `set_property USED_IN`. A continuación se muestra un ejemplo de este proceso para el módulo de memoria, siendo el fichero de restricciones correspondiente occ_memory_rtl_constraints.xdc. El conjunto de ficheros se denomina ooc_memory_rtl.

```
add_files -fileset ooc_memory_rtl [get_files
ooc_memory_rtl_constraints.xdc]

set_property USED_IN {synthesis implementation out_of_context} [get_files
ooc_memory_rtl_constraints.xdc]
```

4.5.1. Restricciones temporales

Existen multitud de aplicaciones que han de ser ejecutadas bajo restricciones temporales. El factor que condiciona suele ser el tiempo máximo de ejecución de un algoritmo con el fin de lograr que permita tiempos de respuesta del sistema lo más bajos posibles y siempre por debajo de un umbral determinado por el entorno de la aplicación. Este es el caso del diseño PCCMUTE, pensado para ser ejecutado en tiempo real.

En este sentido, las especificaciones de diseño imponen una frecuencia de reloj de 100 MHz, equivalente a un periodo de 10 ns.

En primer lugar se crea el reloj del sistema, cuyo valor se fija en nanosegundos. La señal de reloj se denomina clk, siendo su periodo de 10 ns.

```
create_clock -name clk -period 10 [get_ports clk]
```

Mediante la opción `-waveform` se puede indicar el momento del flanco ascendente y del flanco descendente. Por defecto se toma el flanco descendente en la mitad del periodo.

La herramienta por defecto incluye el impacto de la latencia del reloj derivada de la propagación del árbol reloj a través del diseño. Además, incluye el retardo entre la generación del reloj y su inserción en el diseño (latencia de fuente). Por tanto, no es necesario indicar dichos valores de latencia mediante los comandos `set_propagated_clock` y `set_clock_latency`.

Por otra parte, para considerar los efectos de la sincronización entre el diseño implementado sobre la lógica programable y el reloj que procede de un sistema externo a dicha lógica se deben utilizar restricciones que añadan un retardo a la entrada y a la salida de los puertos del sistema. Para esto se cuenta con el comando `set_input_delay` para los puertos de entrada y con el comando `set_output_delay` para los puertos de salida.

```
set_input_delay -clock clk 1 [get_ports resetn]
set_input_delay -clock clk 1 [get_ports tready_rx]
set_input_delay -clock clk 1 [get_ports tvalid_tx]
set_input_delay -clock clk 1 [get_ports tdata_tx]
```

```
set_output_delay -clock clk 1 [get_ports tvalid_rx]
set_output_delay -clock clk 1 [get_ports tready_tx]
set_output_delay -clock clk 1 [get_ports tdata_rx]
```

4.6. Esquemático de la síntesis lógica

El esquemático correspondiente a la síntesis lógica se obtiene seleccionando la opción *Schematic* dentro del menú *Synthesis* de la herramienta. En la Figura 28 se muestra el esquemático resultante de la síntesis lógica.

Se usa la el valor *rebuilt* en la opción *flatten_hierarchy*. Puede observarse que la jerarquía del nivel RTL se mantiene en líneas generales.

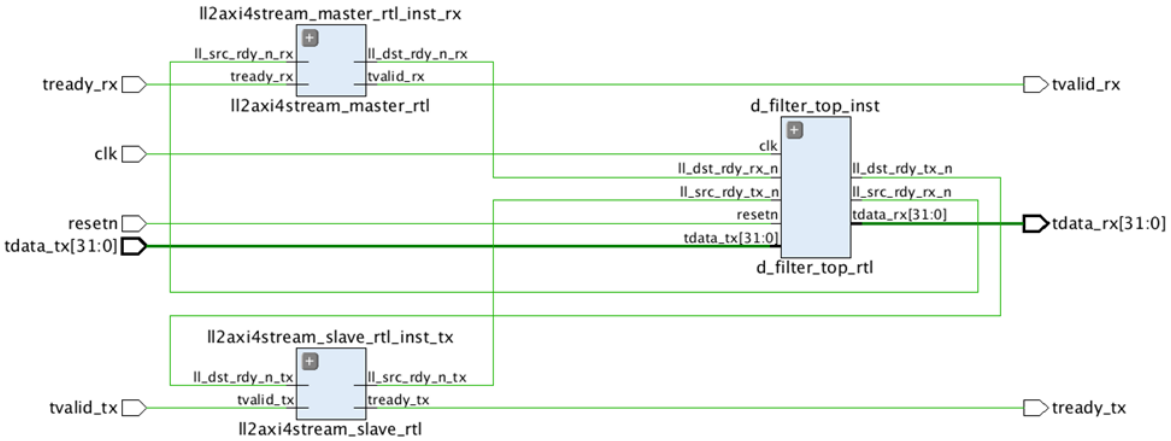

Figura 28. Esquemático síntesis lógica

En cambio, al asignar en la opción de síntesis *flatten_hierachy* el valor *full*, la jerarquía no se mantiene, como se refleja en el esquemático de la Figura 29. El esquemático muestra que los cuatro módulos principales del diseño de PCCMUTE y los dos *wrappers* están en el mismo nivel de jerarquía tras la síntesis lógica, cuando realmente los dos *wrappers* están definidos en un nivel de jerarquía superior.

La herramienta también permite visualizar el esquemático de un bloque en concreto en una ventana diferente. Para ello debe seleccionarse la opción *Collapse Outside* de *Expand/Collapse*. Se regresa a la vista anterior en el mismo menú a través de la opción *Expand Outside*. El esquemático únicamente del diseño de PCCMUTE se muestra en la Figura 30.

4.7. Resultados con Vivado Design Suite

A continuación se indican los resultados de utilización de recursos, análisis temporal, potencia obtenidos de la síntesis lógica siguiendo estrategia *top-down* y *bottom-up* para las herramientas Vivado Design Suite, en la que se centra este Trabajo Fin de Máster y para Synplify Premier DP, empleada con el objetivo de comparar los resultados entre ambas herramientas.

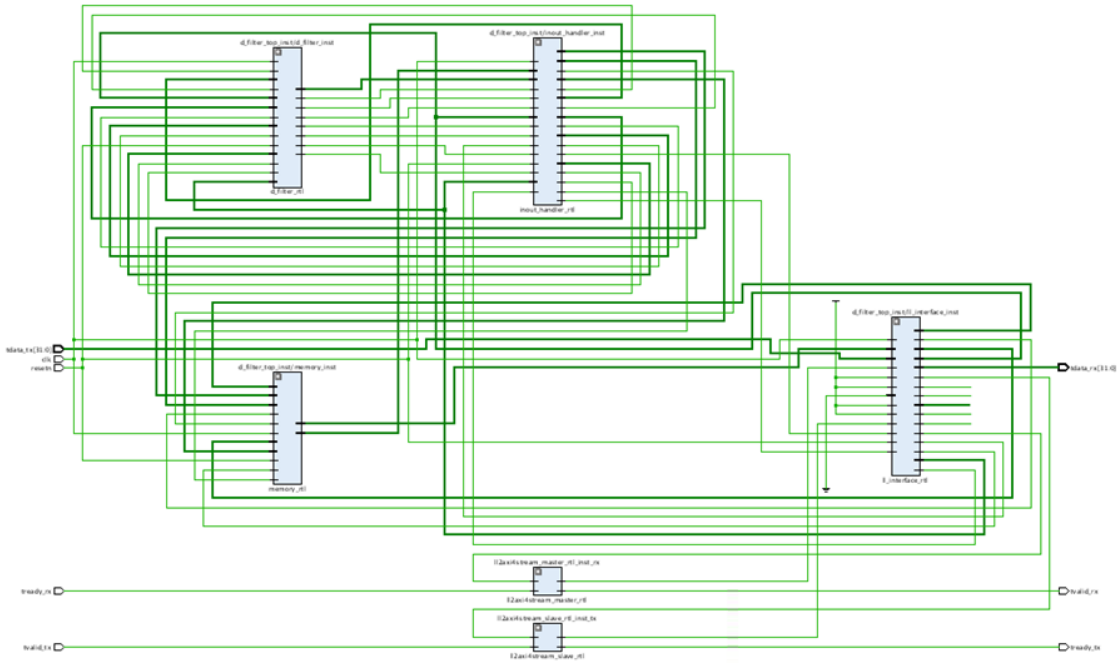

Figura 29. Esquemático de la síntesis lógica eliminando jerarquía

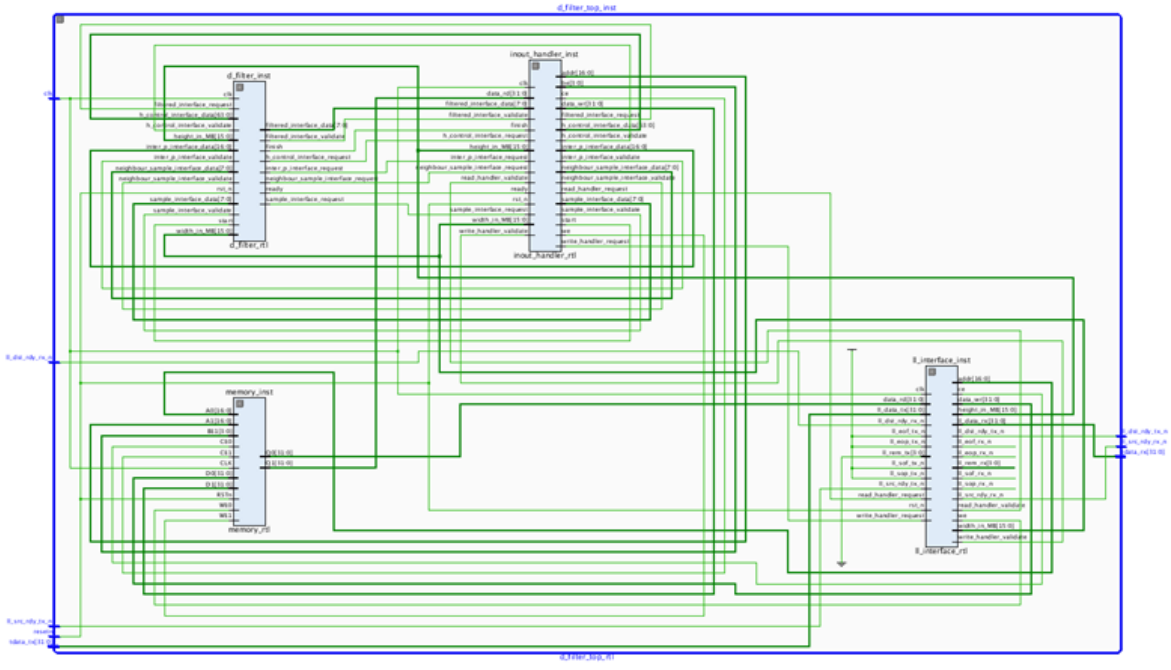

Figura 30. Esquemático síntesis lógica del diseño PCCMUTE

4.7.1. Estrategia *Bottom-up*

En primer lugar se presentan los resultados de utilización, análisis temporal y de potencia asociados a la síntesis lógica del diseño sobre la plataforma Zynq-7000 ZC702. En este caso se siguió una estrategia *bottom-up*. Después se presentan los resultados obtenidos de la síntesis lógica sobre la plataforma Zynq-7000 ZC706.

Plataforma Zynq-7000 ZC702

- Utilización de recursos

Los recursos utilizados por el diseño completo se reflejan en la Tabla 12. Según los resultados de utilización obtenidos se utilizan una gran cantidad de los recursos disponibles a excepción de los DSP.

Tabla 12. Utilización de recursos en la plataforma Zynq-7000 ZC702

Recurso	Cantidad	Porcentaje (%)
LUT	47.431	89,16
FLIP-FLOP	64.389	60,52
BRAM	128	91,43
DSP	2	0,91

La mayoría de los recursos se dedican al *Deblocking Filter*, cuyos recursos se indica en la Tabla 13. En cambio, los bloques de memoria RAM son utilizados únicamente por el módulo de memoria. En la Figura 31 se muestra un gráfico con la utilización de LUTs por parte de cada uno de los módulos principales del diseño.

Tabla 13. Recursos utilizados por el *Deblocking Filter*

Recurso	Cantidad	Porcentaje (%)
LUT	41.876	78,71
FLIP-FLOP	60.578	56,93
BRAM	0	0,00
DSP	1	0,45

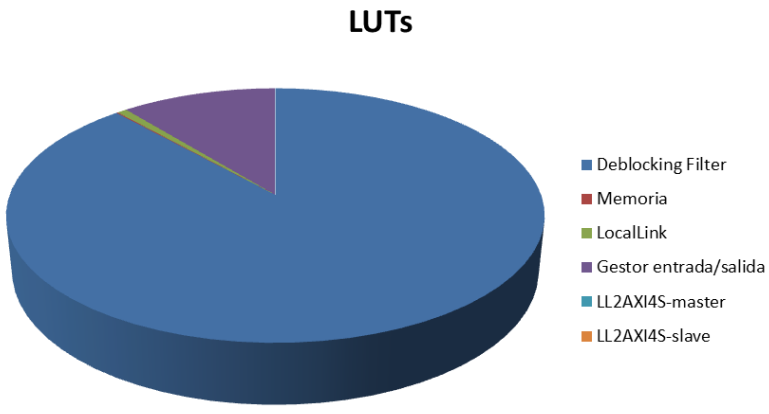

Figura 31. LUTs por cada bloque del diseño (ZC702)

- Análisis temporal

El parámetro WNS indica el valor de *slack* más negativo. Si su valor es negativo, como es este caso, se está ante un *slack* negativo. Por tanto, no se cumple la frecuencia objetivo de 100 MHz.

Además, esta situación no ocurre únicamente en un camino, debido a que el parámetro TNS, que mide la suma de *slakcs* negativos, es mayor que WNS. Ambos valores se muestran en la Tabla 14. Ante esta circunstancia, se decide cambiar a la siguiente plataforma en prestaciones, el modelo Zynq-7000 ZC706 [8].

Tabla 14. Parámetros temporales en la plataforma Zynq-7000 ZC702

Parámetro	Tiempo (ns)
WNS	-3,624
TNS	-320,968

Plataforma Zynq-7000 ZC706

- **Utilización de recursos**

La utilización de los recursos de la lógica programable por parte del diseño en su conjunto aparece en la Tabla 15, mientras que en el gráfico de la Figura 32 se refleja el porcentaje de utilización de recursos sobre esta plataforma.

Tabla 15. Utilización de recursos en la estrategia *bottom-up*

Recurso	Cantidad	Porcentaje (%)
LUT	44.785	20,49
FLIP-FLOP	64.390	14,73
BRAM	128	23,49
DSP	2	0,22

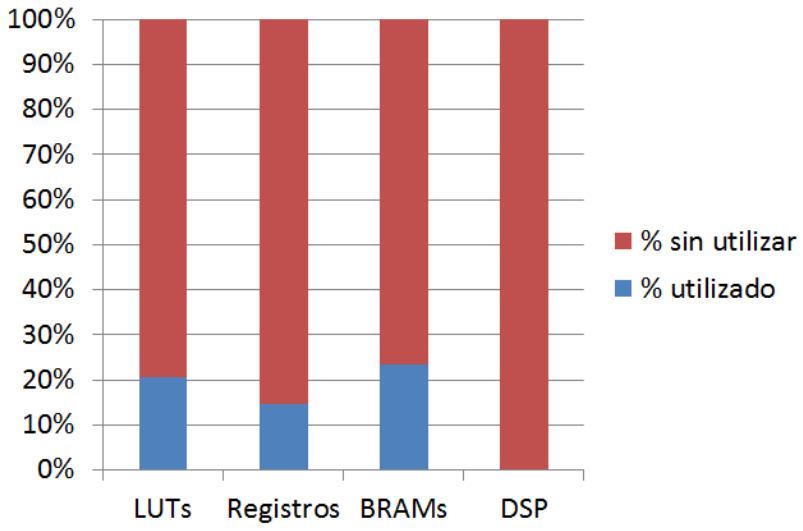

Figura 32. Porcentaje de utilización de recursos

En la Tabla 16, Tabla 17, Tabla 18, Tabla 19 y Tabla 20 se indican los recursos que consumen cada uno de los módulos declarados como fuera de contexto: el *Deblocking Filter*, el gestor de entrada/salida, el bloque de memoria, la interfaz LocalLink y los dos *wrappers* respectivamente.

Los dos *wrappers* usan la misma cantidad de recursos. El *Deblocking Filter* es el bloque que más recursos utiliza a excepción de los BRAM, que son utilizados por el bloque de memoria. Aunque el gestor de entrada/salida (INOUT_handler) utiliza una cantidad considerablemente menor, supone un valor porcentual mucho mayor que la interfaz LocalLink.

Tabla 16. Utilización de recursos del *Deblocking Filter* con la estrategia *bottom-up*

Recurso	Cantidad	Porcentaje (%)
LUT	39.218	17,94
FLIP-FLOP	60.579	13,85
BRAM	0	0,00
DSP	1	0,11

Tabla 17. Utilización de recursos del INOUT_handler con la estrategia *bottom-up*

Recurso	Cantidad	Porcentaje (%)
LUT	5.186	2.37
FLIP-FLOP	3.392	0.77
BRAM	0	0.00
DSP	1	0.11

Tabla 18. Utilización de recursos del bloque de memoria con la estrategia *bottom-up*

Recurso	Cantidad	Porcentaje (%)
LUT	50	0,02
FLIP-FLOP	5	0,01
BRAM	128	23,49
DSP	0	0,00

Tabla 19. Utilización de recursos de la interfaz LocalLink con la estrategia *bottom-up*

Recurso	Cantidad	Porcentaje (%)
LUT	327	0,15
FLIP-FLOP	414	0,09
BRAM	0	0,00
DSP	0	0,00

Tabla 20. Utilización de recursos de cada *wrapper* con la estrategia *bottom-up*

Recurso	Cantidad	Porcentaje (%)
LUT	2	0,001
FLIP-FLOP	0	0,000
BRAM	0	0,000
DSP	0	0,000

- **Análisis temporal**

Respecto del análisis temporal del diseño cabe destacar que se cumple con las restricciones de frecuencia, resultado el *slack* positivo, siendo por tanto el valor de suma total de valores de *slack* negativos (TNS) cero, según se refleja en la Tabla 21. El peor resultado de *slack* (WNS) es de 2.628 ns, correspondiéndose con la conexión entre *state_read_work_luma_word_32_reg[5]* y *imin_0_ln1675_imin_out_0_0_reg[1*], dentro del bloque de luma del *Deblocking Filter*.

Tabla 21. Parámetros temporales con la estrategia *bottom-up*

Parámetro	Tiempo (ns)
WNS	2.628
TNS	0.00

- **Análisis de potencia**

En relación con el análisis de potencia, los resultados arrojan que la potencia dinámica supone la mayor parte de la potencia que se estima consumiría el diseño. En la Figura 33 se refleja dicha relación. Sin embargo, respecto al informe de potencia obtenido para la plataforma Zynq-7000 ZC702 el porcentaje de potencia estática ha aumentado. Esto también demuestra que la herramienta realiza un menor esfuerzo de optimización a causa de disponer de muchos más recursos en la plataforma de los necesarios para el diseño.

La cifra total de potencia consumida estimada tras la síntesis lógica es de 1.287 W, correspondiéndose 0.235 W con potencia estática y 1.052 W con potencia dinámica.

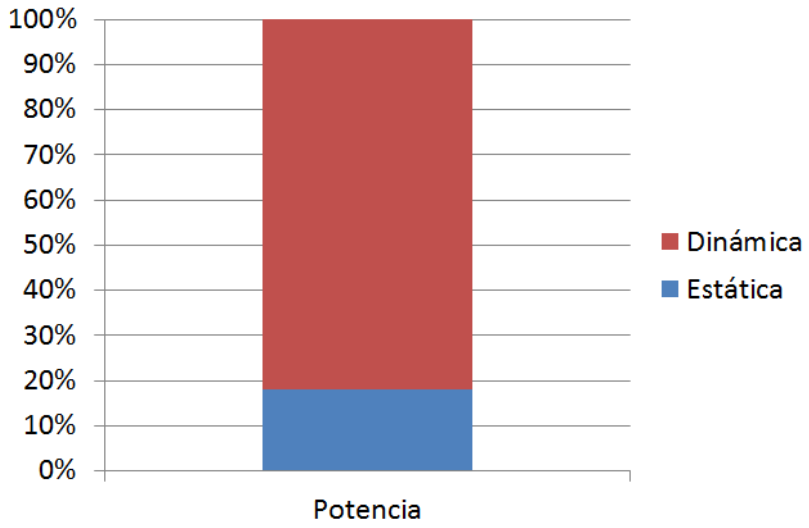

Figura 33. Consumo de potencia con *bottom-up* en síntesis lógica

4.7.2. Estrategia *Top-down*

A continuación se indican los resultados de la síntesis lógica siguiendo una estrategia *top-down* respecto a la utilización de recursos como al análisis temporal y de potencia para la plataforma Zynq-7000 ZC706.

- **Utilización de recursos**

En la Tabla 22 se muestran los recursos usados por el diseño a partir de la estrategia *top-down* en Vivado Design Suite. De igual manera que con la estrategia *bottom-up*, los registros son el recurso más usado en nivel absoluto pero porcentualmente sobran menos LUTs por utilizar en la lógica programable. Por otro lado, en la Tabla 23, Tabla 24, Tabla 25, Tabla 26 y Tabla 27 se indican los recursos que consumen cada uno de los módulos declarados como fuera de contexto: el *Deblocking Filter*, el gestor de entrada/salida, el bloque de memoria, la interfaz LocalLink y los dos *wrappers* respectivamente. De los recursos usados la mayor parte corresponden con el *Deblocking Filter*, a excepción de los BRAM, que son utilizados por el bloque de memoria. Por su parte, cada *wrapper* usa dos LUTs, una para cada inversor.

Tabla 22. Utilización de recursos con la estrategia *top-down*

Recurso	Cantidad	Porcentaje (%)
LUT	45.049	20,61
FLIP-FLOP	68.275	15,62
BRAM	128	23,49
DSP	2	0,22

Tabla 23. Utilización de recursos del *Deblocking Filter* con la estrategia *top-down*

Recurso	Cantidad	Porcentaje (%)
LUT	39.192	17,92
FLIP-FLOP	60.582	13,85
BRAM	0	0,00
DSP	1	0,11

Tabla 24. Utilización de recursos del bloque INOUT_handler con la estrategia *top-down*

Recurso	Cantidad	Porcentaje (%)
LUT	5.169	2,36
FLIP-FLOP	3.392	0,77
BRAM	0	0,00
DSP	1	0,11

Tabla 25. Utilización de recursos del bloque de memoria con la estrategia *top-down*

Recurso	Cantidad	Porcentaje (%)
LUT	54	0,020
FLIP-FLOP	8	0,001
BRAM	128	23,490
DSP	0	0,000

Tabla 26. Utilización de recursos de la interfaz LocalLink con la estrategia *top-down*

Recurso	Cantidad	Porcentaje (%)
LUT	327	0,14
FLIP-FLOP	414	0,09

BRAM	0	0,00
DSP	0	0,00

Tabla 27. Utilización de recuros de cada *wrapper* con la estrategia *top-down*

Recurso	Cantidad	Porcentaje (%)
LUT	2	0,001
FLIP-FLOP	0	0,000
BRAM	0	0,000
DSP	0	0,000

- **Análisis temporal**

El diseño siguiendo la estrategia *top-down* cumple con la restricción de 100 MHz de frecuencia puesto que se tiene un *slack* positivo. Por tanto, la suma de todos los *slacks* negativos (TNS) es cero. El peor resultado de *slack* (WNS) se corresponde con el camino de la señal *read_inout_handler_width_in_MB_ln91_Z_0_0_reg[1]* hacia el puerto *sub_ln1022_Z_3_1_reg* en el interior del módulo *INOUT handler*. En la Tabla 28 se muestran los valores de WNS y TNS.

Tabla 28. Parámetros temporales con la estrategia *top-down*

Parámetro	Tiempo (ns)
WNS	1,012
TNS	0,000

- **Análisis de potencia**

En la Figura 34 se muestra potencia consumida por el diseño diferenciando las potencias dinámica y estática. La potencia dinámica supone un 73% de la cifra de potencia total frente al 27% de la potencia estática.

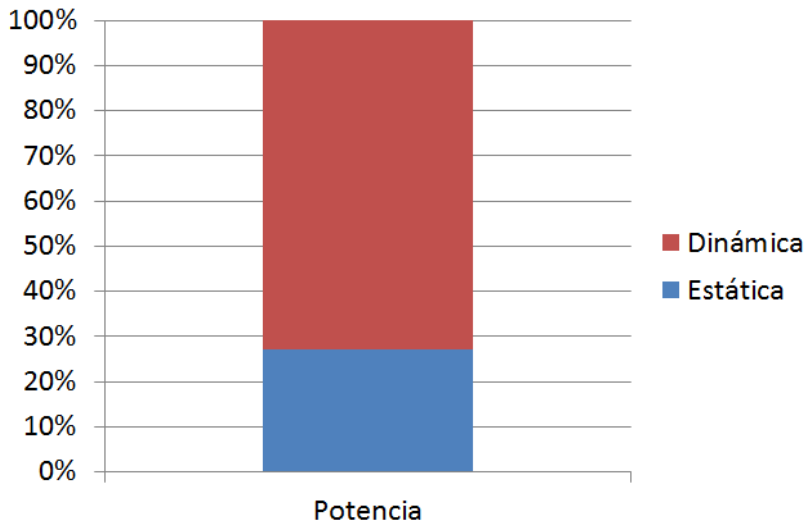

Figura 34. Consumo de potencia con *top-down* en síntesis lógica

4.7.3. Comparativa

Se presenta la comparativa respecto a la utilización de recursos de la lógica programable entre las estrategias *bottom-up* y *top-down* en la Tabla 29.

Las principales diferencias entre ambas estrategias se producen en cuanto a los resultados de LUTs y registros utilizados. No obstante, estas diferencias no son especialmente significativas. Cabe destacar que la estrategia *top-down* seguida en Vivado Design Suite no es una estrategia *top-down* pura, puesto que se declaran módulos como fuera de contexto, y estos junto con los submódulos que contengan se sintetizan de manera independiente. En la Figura 35 se muestra de manera gráfica la diferencia en número de LUTs y de registros usados por las dos estrategias.

Tabla 29. *Bottom-up* vs. *top-down* en síntesis lógica con Vivado

	Bottom-up		*Top-down*	
Recurso	**Cantidad**	**Porcentaje (%)**	**Cantidad**	**Porcentaje (%)**
LUT	44.785	20,49	45.049	20,61
FLIP-FLOP	64.390	14,73	68.275	15,62
BRAM	128	23,49	128	23,49
DSP	2	0,22	2	0,22

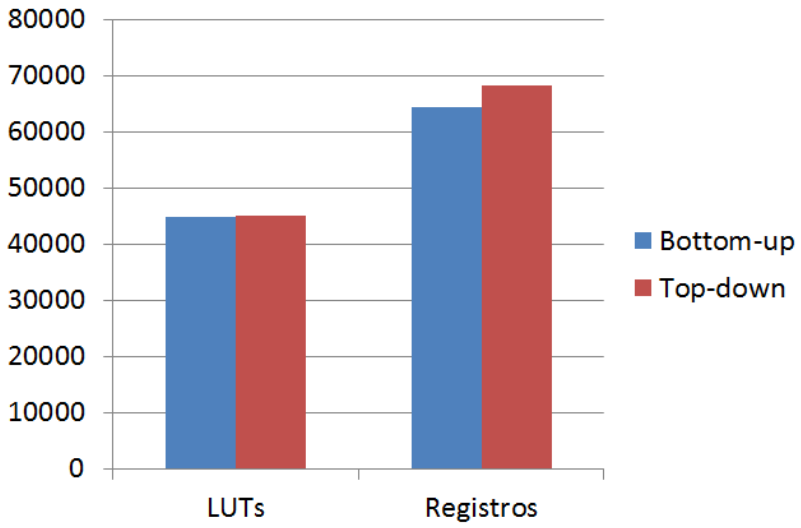

Figura 35. Comparativa de LUTs y registros en síntesis lógica entre estrategias

4.8. Resultados con Synplify Premier DP

4.8.1. Estrategia *Bottom-up*

- **Utilización de recursos**

Se presentan a continuación, en la Tabla 30, los resultados totales de utilización de recursos de la lógica programable para la síntesis lógica con Synplify Premier DP.

Tabla 30. Utilización de recursos en Synplify con estrategia *bottom-up*

Recurso	Cantidad	Porcentaje (%)
LUT	34.075	15,58
FLIP-FLOP	21.944	5,01
BRAM	106	19,44
DSP	3	0,30

4.8.2. Estrategia *Top-down*

- **Utilización de recursos**

La utilización de recursos para el diseño completo usando la estrategia *top-down* se indican en la Tabla 31. Tanto en porcentaje como en valores absolutos el recurso más utilizado es la LUT.

Tabla 31. Utilización de recursos en Synplify con estrategia *top-down*

Recurso	Cantidad	Porcentaje (%)
LUT	27.043	12,37
FLIP-FLOP	17.730	4,05
BRAM	104	19,08
DSP	2	0,22

- **Análisis temporal**

Se realiza asimismo el análisis temporal, siendo el peor valor de *slack* para el diseño es de 3.579 ns, por lo que el diseño cumple con la restricción de 100 MHz de frecuencia o 10 ns de periodo. La frecuencia estimada para el peor *slack* es de 155.7 MHz, que equivale a un periodo de 6.421 ns.

4.8.3. Comparativa

Según se muestra en la Tabla 32, con la herramienta Synplify Premier DP se establece una diferencia importante en número de LUTs, registros, BRAMs e incluso DSPs. Mediante una estrategia *top-down* se consiguen mejores prestaciones a cambio de sacrificar tiempo de cómputo y tiempo de dedicación a un proyecto, ya que cualquier modificación en un bloque implica sintetizar nuevamente todo el diseño.

Tabla 32. Comparativa entre estrategias de síntesis con Synplify

	Bottom-up		*Top-down*	
Recurso	Cantidad	Porcentaje (%)	Cantidad	Porcentaje (%)
LUT	34.075	15,58	27.043	12,37
FLIP-FLOP	21.944	5,01	17.730	4,05
BRAM	106	19,44	104	19,08
DSP	3	0,30	2	0,22

4.9. Comparativa entre Vivado Design Suite y Synplify Premier DP

Respecto a los recursos utilizados, los resultados reflejan diferencias notables entre ambas herramientas tanto en número de LUTs, de registros como de BRAMs instanciados con las estrategias *top-down* y *bottom-up*. Cada herramienta utiliza diferentes algoritmos en la etapa de síntesis lógica y de optimización, lo que conlleva diferencias en los resultados obtenidos.

Además, en el caso de la estrategia *top-down* en Vivado, la circunstancia de tener que declarar módulos como fuera de contexto puede no ayudar a reducir los recursos que se consumen respecto a Synplify.

En la Figura 36 se muestran los recursos en LUTs y registros que usan cada una de las herramientas en la estrategia *top-down*, mientras que en la Figura 37 se muestra la diferencia en número de BRAMs, siendo el número de DSPs el mismo, una cantidad baja, más complicada de optimizar.

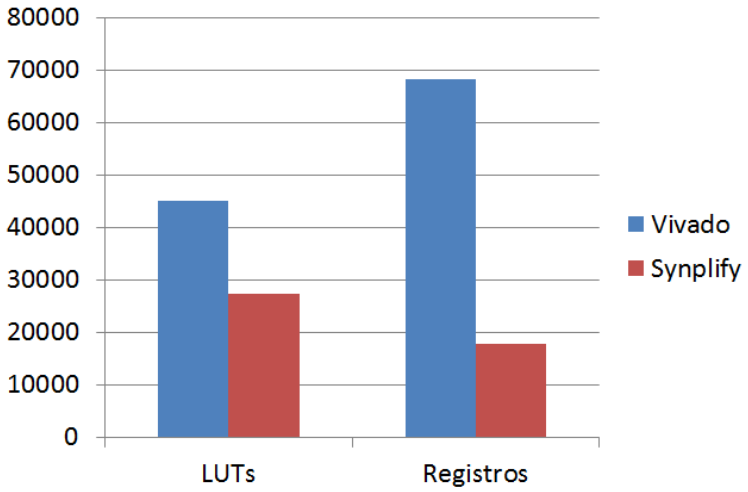

Figura 36. Comparativa LUTs y registros entre herramientas con *top-down*

La utilización de LUTs y registros es considerablemente mayor en Vivado Design Suite que en Synplify Premier DP. Esto guarda relación con los algoritmos de síntesis inherentes y de optimización a cada herramienta. Por ejemplo, en Vivado el *Deblocking Filter* no instancia ningún bloque RAM (BRAM), por lo que necesita de más LUTs y registros para implementarlas. A pesar de esto, Vivado Design Suite emplea más BRAM que Synplify Premier DP. Los resultados obtenidos con Synplify son mejores que con Vivado.

Por su parte, en las gráficas de las Figura 38 y Figura 39 se reflejan las diferencias en cantidad LUTs y de registros, y de BRAMs y DSP respectivamente siguiendo una estrategia *bottom-up*. Con esta estrategia se observa el mismo efecto, a pesar de que en el caso de los DSPs Synplify instancie una unidad más.

Asimismo, los recursos usados también tienen un efecto importante en los resultados del análisis temporal del diseño, debido a que las distancias recorridas por las señales tienden a ser menores. A pesar de que con ambas herramientas se cumple con la restricción de 100 MHz de frecuencia o 10 ns de periodo, con Synplify también se obtienen mejores resultados temporales se consigue una mayor frecuencia de funcionamiento en base al valor de *slack*.

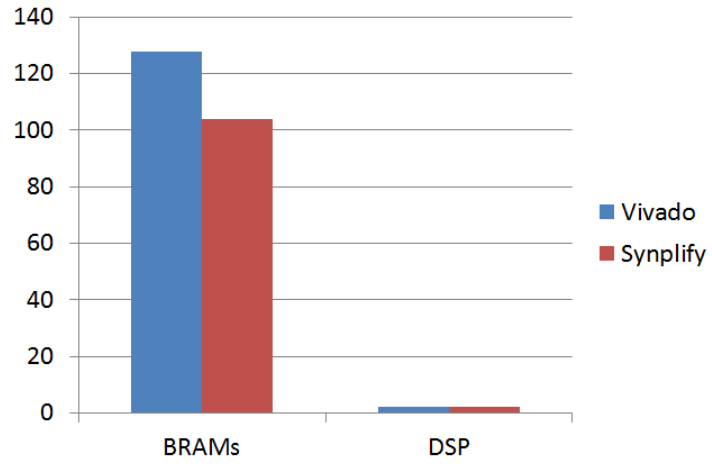

Figura 37. Comparativa BRAM y DSP entre herramientas con *top-down*

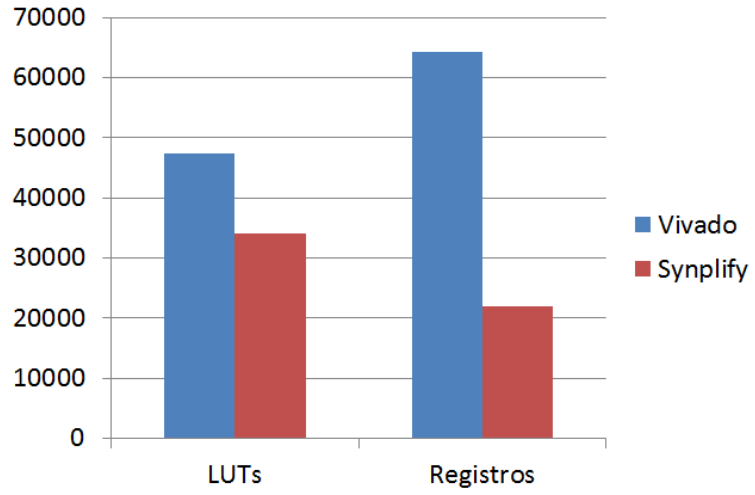

Figura 38. Comparativa LUTs y registros entre herramientas con *bottom-up*

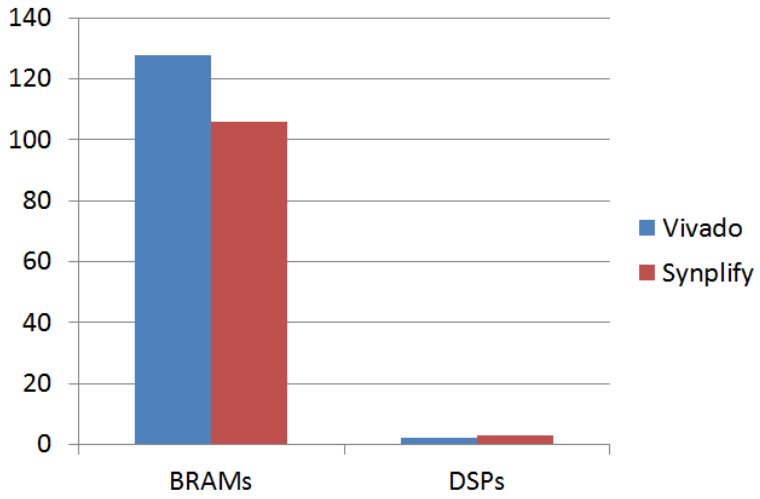

Figura 39. Comparativa BRAM y DSP entre herramientas con *bottom-up*

4.10. Conclusiones

La ejecución de la síntesis lógica se realiza desde el nivel RTL del diseño. Se sintetiza para las estrategias *bottom-up* y *top-down*. Se obtienen resultados de utilización de recursos, temporales y de potencia. La síntesis lógica se realiza primero con la herramienta Vivado Design Suite y más tarde con Synplify Premier DP.

La comparación entre las estrategias de síntesis con Vivado Design Suite refleja que no hay grandes diferencias en cuanto a resultados de utilización de recursos. Esto estaría relacionado con la similitud entre ambas estrategias en esta herramienta a causa de la declaración como fuera de contexto de módulos del diseño.

Respecto a los resultados obtenidos con Synplify Premier DP y su comparación con los obtenidos con Vivado Design Suite, teniendo en cuenta que cada herramienta utiliza algoritmos de síntesis diferentes, se logran mejores resultados de utilización de recursos y temporales con Synplify Premier DP.

Capítulo 5: Implementación

5.1. Introducción

Una vez se obtiene el *netlist* adecuado de la síntesis lógica, el siguiente paso consiste en realizar la implementación o síntesis física. Vivado Design Suite divide el proceso de implementación en varias etapas agrupadas en tres pasos:

- Optimización lógica.
- Colocación.
- *Ruteado*.

Cada una de las etapas se corresponde a su vez con un grupo de opciones de implementación. Dichas etapas son las siguientes:

- *Opt Design*. Se optimiza la lógica del diseño.
- *Power Opt Design*. Se realiza optimización con el objetivo de reducir el consumo de potencia del diseño.
- *Place Design*. Se coloca el diseño sobre la lógica programable.
- *Post-Place Power Opt Design*. Se optimiza de nuevo el diseño con objetivo de reducir el consumo de potencia una vez se ha realiza el *placement*.
- *Post-Place Phys Opt Design*. Se realiza optimiza en aquellas rutas con *slack* negativo.
- *Route Design*. Se *rutea* el diseño sobre la lógica programable.
- *Post-Route Phys Opt Design*. Se optimiza la lógica del diseño, el *placement* y el *routing* considerando los retardos reales.

En este capítulo se comparan los resultados obtenidos mediante las estrategias *top-down* y *bottom-up* así como las diferencias al realizar la implementación en Vivado Design Suite y Synplify Premier DP.

5.2. Opciones de implementación

Las opciones de implementación sirven para configurar el proceso de implementación. En Vivado, estas comprenden los siguientes grupos, acordes con las fases involucradas en esta etapa de diseño:

- Optimización del diseño (*opt_design*).
- Optimización de potencia (*power_opt_design*).
- Colocación del diseño (*place_design*)
- Optimización física del diseño (*phys_opt_design*).
- Ruteado del diseño (*route_design*).

Después de realizar diversos hilos de implementación, las opciones de implementación utilizadas son las siguientes:

- Optimización del diseño, se utiliza la directiva *ExploreArea*.
- Colocación del diseño, se selecciona la directiva *SpreadLogic_high*.

- Optimización de potencia, se realiza después de la colocación del diseño.
- Optimización física del diseño, se utiliza la directiva de Vivado por defecto.
- *Ruteado* del diseño, se escoge la directiva *MoreGlobalIterations*.
- Optimización física del diseño nuevamente una vez se ha *ruteado* el diseño.

5.3. Restricciones físicas

Cabe señalar que las restricciones físicas son obviadas en la etapa de síntesis lógica. Existen diferentes tipos de restricciones físicas. Se puede crear un bloque asociado al módulo fuera de contexto mediante el comando `create_pblock`. A este bloque hay que añadirle las celdas asociadas al módulo fuera de contexto a través del comando `add_cells_to_pblock`. Además, activando la opción CONTAIN_ROUTING se indica que en el bloque no se usen recursos externos para *ruteado*. En la siguiente línea se muestra un ejemplo para el módulo la interfaz Local Link.

```
create_pblock pblock_d_filter_inst

add_cells_to_pblock [get_pblocks pblock_d_filter_inst] [get_cells
d_filter_top_inst/d_filter_inst]

set_property CONTAIN_ROUTING true [get_pblocks pblock_d_filter_inst]
```

Para fijar una localización de los puertos de un módulo se usa la restricción HD.PARTPIN_RANGE. Esta especifica un rango de SLICE que pueden ser usados para un puerto indicado. En la siguiente línea se muestra un ejemplo de uso de esta restricción para los puertos de entrada del *wrapper* esclavo AXI4-Stream a LocalLink.

```
set_property HD.PARTPIN_RANGE SLICE_X160Y174:SLICE_X161Y174 [get_ports
tvalid_tx]

set_property HD.PARTPIN_RANGE SLICE_X160Y174:SLICE_X161Y174 [get_ports
ll_dst_rdy_n_tx]
```

5.4. Esquemático de la implementación

En la Figura 40 se muestra la implementación del diseño sobre la lógica programable del SoC Zynq-7000 7z045. Como puede observarse, se tiene una baja ocupación de los recursos disponibles.

5.5. Resultados con Vivado Design Suite

5.5.1. Estrategia *Bottom-up*

Seguidamente se presentan los resultados obtenidos de la implementación siguiendo una estrategia *bottom-up* en cuanto a utilización de recursos, análisis temporal y de potencia.

- **Utilización de recursos**

Los recursos usados por el diseño completo se reflejan en la Tabla 33. Porcentualmente las LUTs son el recurso más usado, aunque en números absolutos se utilizan más registros.

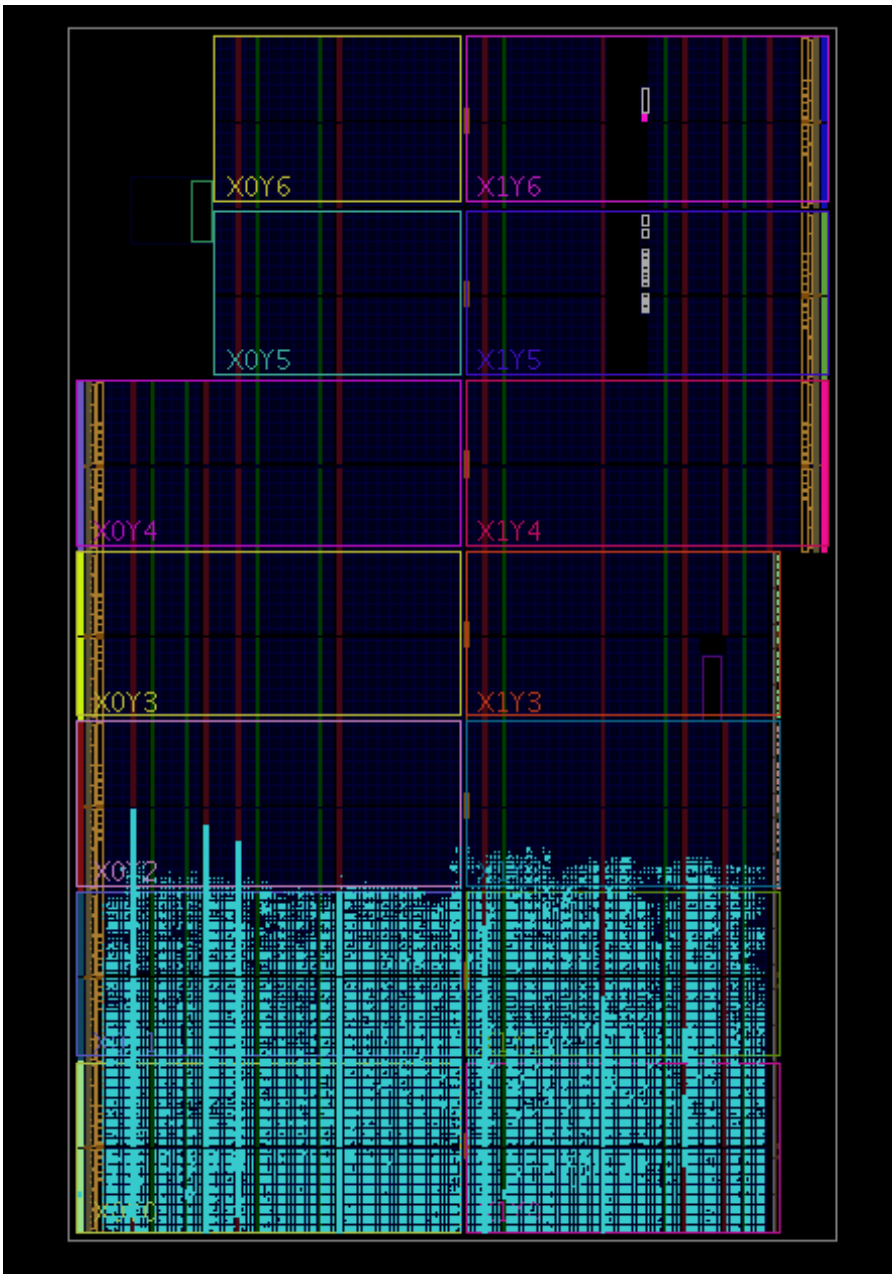

Figura 40. Implementación sobre la lógica programable

Tabla 33. Utilización de recursos en implementación con la estrategia *bottom-up*

Recurso	Cantidad	Porcentaje (%)
LUT	44.411	20,31
FLIP-FLOP	64.278	14,70
BRAM	128	23,49
DSP	2	0,22

En la Tabla 34, Tabla 35, Tabla 36, Tabla 37 y Tabla 38 se indican los recursos empleados los distintos módulos declarados como fuera de contexto: el *Deblocking Filter*, el gestor de entrada/salida, el bloque de memoria, la interfaz LocalLink y los dos *wrappers* respectivamente. Al

igual que en la síntesis lógica, los dos *wrappers* usan la misma cantidad de recursos. También el *Deblocking Filter* es el bloque que más recursos utiliza a excepción de los BRAM, que únicamente son usadas por el bloque de memoria. Asimismo, el gestor de entrada/salida (INOUT_handler) utiliza una cantidad considerablemente menor, pero supone un valor porcentual mucho mayor que la interfaz LocalLink.

Tabla 34. Utilización de recursos del *Deblocking Filter* con estrategia *bottom-up*

Recurso	Cantidad	Porcentaje (%)
LUT	39.027	17,85
FLIP-FLOP	60.579	13,85
BRAM	0	0,00
DSP	1	0,11

Tabla 35. Utilización de recursos del bloque INOUT_handler con estrategia *bottom-up*

Recurso	Cantidad	Porcentaje (%)
LUT	4.347	1,98
FLIP-FLOP	3.392	0,77
BRAM	0	0,00
DSP	1	0,11

Tabla 36. Utilización de recursos del bloque de memoria con estrategia *bottom-up*

Recurso	Cantidad	Porcentaje (%)
LUT	178	0,080
FLIP-FLOP	15	0,003
BRAM	128	23,490
DSP	0	0,000

Tabla 37. Utilización de recursos de la interfaz LocalLink con estrategia *bottom-up*

Recurso	Cantidad	Porcentaje (%)
LUT	339	0,15
FLIP-FLOP	414	0,09
BRAM	0	0,00
DSP	0	0,00

Tabla 38. Utilización de recursos de cada *wrapper* con estrategia *bottom-up*

Recurso	Cantidad	Porcentaje (%)
LUT	2	0,001
FLIP-FLOP	0	0,000
BRAM	0	0,000
DSP	0	0,000

- **Análisis temporal**

Después de la etapa de implementación se continúa cumpliendo con la restricción temporal ya que se obtiene un *slack* positivo. El peor resultado de *slack* (WNS) se corresponde con la señal *state_d_filter_param_bs_svc_main_reg[88]* dentro del módulo PARAM_BS del *Deblocking Filter*. En la Tabla 39 se muestran los valores de WNS y TNS.

Tabla 39. Análisis temporal con estrategia *bottom-up*

Parámetro	Tiempo (ns)
WNS	1,150
TNS	0,000

- **Análisis de potencia**

La potencia dinámica consumida es mucho mayor que la potencia estática según la estimación de consumo de potencia del diseño. En la Figura 41 se refleja el porcentaje que corresponde a cada una.

El porcentaje de potencia estática ha aumentado respecto al obtenido en la síntesis lógica para esta misma estrategia. Esto tiene relación con el aumento en conexiones a causa del *ruteado* realizado.

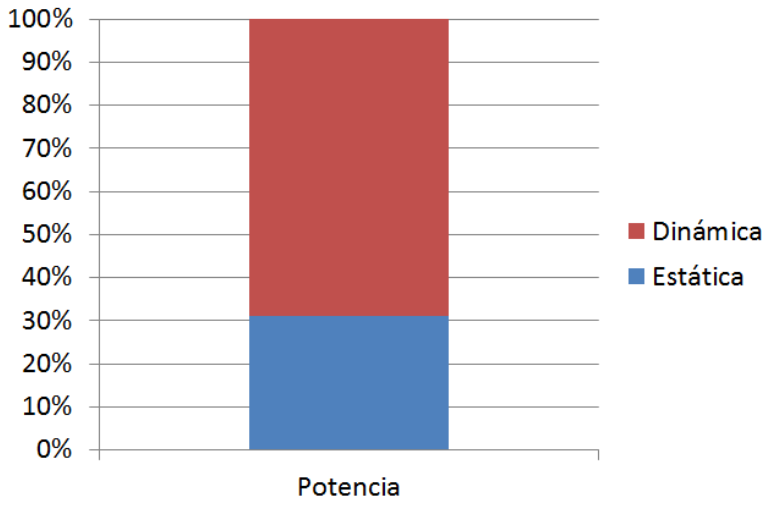

Figura 41. Consumo de potencia con *bottom-up* en implementación

5.5.2. Estrategia *Top-down*

Los resultados obtenidos de la implementación siguiendo una estrategia *top-down* en relación con la utilización de recursos, análisis temporal y de potencia se indican a continuación.

- **Utilización de recursos**

Los recursos del diseño completo que se han usado se muestran en la Tabla 40. Como ocurre en la síntesis lógica, el recurso más utilizado porcentualmente son las LUTs.

Tabla 40. Utilización de recursos en implementación con estrategia *top-down*

Recurso	Cantidad	Porcentaje (%)
LUT	43.997	20,13
FLIP-FLOP	64.434	14,74
BRAM	128	23,49
DSP	2	0,22

Respecto al uso de los recursos que conlleva cada módulo declarado como fuera de contexto y su arquitectura: el *Deblocking Filter*, el gestor de entrada/salida, el bloque de memoria, la interfaz LocalLink y los dos *wrappers*, se señalan en las Tabla 41, Tabla 42, Tabla 43, Tabla 44 y Tabla 45 respectivamente. El *Deblocking Filter* supone la mayor parte de los recursos a excepción de los BRAM, instanciados en la lógica programable únicamente por el bloque de memoria.

Tabla 41. Utilización de recursos del *Deblocking Filter* con estrategia *top-down*

Recurso	Cantidad	Porcentaje (%)
LUT	39.116	17,89
FLIP-FLOP	60.582	13,85
BRAM	0	0,00
DSP	2	0,11

Tabla 42. Utilización de recursos del bloque INOUT_handler con estrategia *top-down*

Recurso	Cantidad	Porcentaje (%)
LUT	4.384	2,00
FLIP-FLOP	3.392	0,77
BRAM	0	0,00
DSP	0	0,00

Tabla 43. Utilización de recursos del bloque de memoria con estrategia *top-down*

Recurso	Cantidad	Porcentaje (%)
LUT	182	0,008
FLIP-FLOP	18	0,004
BRAM	128	23,490
DSP	0	0,000

Tabla 44. Utilización de recursos de la interfaz LocalLink con estrategia *top-down*

Recurso	Cantidad	Porcentaje (%)
LUT	339	0,15
FLIP-FLOP	414	0,09
BRAM	0	0,00
DSP	0	0,00

Tabla 45. Utilización de recursos de cada *wrapper* con estrategia *top-down*

Recurso	Cantidad	Porcentaje (%)
LUT	2	0,001
FLIP-FLOP	0	0,000
BRAM	0	0,000
DSP	0	0,000

- **Análisis temporal**

Se sigue cumpliendo con la restricción temporal de 10 ns de frecuencia de reloj (100 MHz) debido a que el *slack* es positivo. El peor resultado de *slack* (WNS) se da en la ruta *add_ln158_Z_0_0_reg[0]_rep–memwrite_d_filter_param_bs_svc_ref_curr_ln599_Z_0_0_reg[192]* dentro del módulo PARAM_BS del *Deblocking Filter*. En la Tabla 46 se muestran los valores de WNS y TNS.

Tabla 46. Análisis temporal con estrategia *top-down*

Parámetro	Tiempo (ns)
WNS	1,572
TNS	0,000

- Análisis de potencia

La potencia dinámica es más del doble que la potencia estática según la estimación de consumo de potencia del diseño. En la Figura 42 se muestra la distribución de las mismas. El porcentaje de potencia estática se mantiene en un porcentaje similar respecto al obtenido en la síntesis lógica para esta misma estrategia.

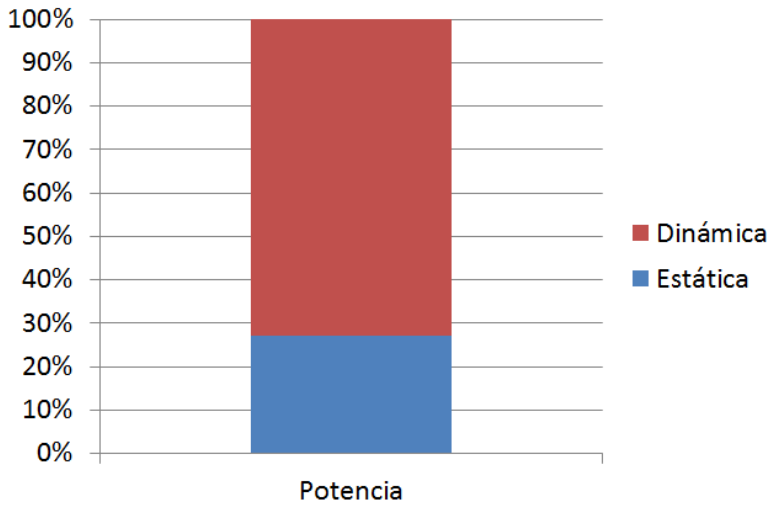

Figura 42. Consumo de potencia con *top-down* en implementación

5.5.3. Comparativa

Los resultados de utilización para el diseño obtenidos con las estrategias *bottom-up* y *top-down* se presentan en la Tabla 47.

En el caso de la implementación ocurre como con la síntesis lógica, los resultados siguiendo ambas estrategias no son muy diferentes, consumiéndose menos LUTs con la estrategia *top-down*. En cambio, siguiendo esta estrategia se consumen más registros, un 0,04 % más.

Respecto a las diferencias entre los resultados de la síntesis lógica y de implementación, son mejores los obtenidos en la implementación. En esta etapa se incluyen restricciones físicas, que facilitan el acotamiento de las posibles soluciones de colocación y ruteado sobre la lógica programable.

Por otra parte, el porcentaje de BRAM usadas no es elevado y se utiliza un bajo número de DSPs, por lo que su optimización es más complicada. En la Figura 43 se muestra gráficamente la cantidad de LUTs y de registros usados por las dos estrategias en la implementación.

Tabla 47. *Bottom-up* vs. *top-down* en implementación con Vivado

	Bottom-up		*Top-down*	
Recurso	Cantidad	Porcentaje (%)	Cantidad	Porcentaje (%)
LUT	44.411	20,31	43.997	20,13
FLIP-FLOP	64.278	14,70	64.434	14,74
BRAM	128	23,49	128	23,49
DSP	2	0.22	2	0,22

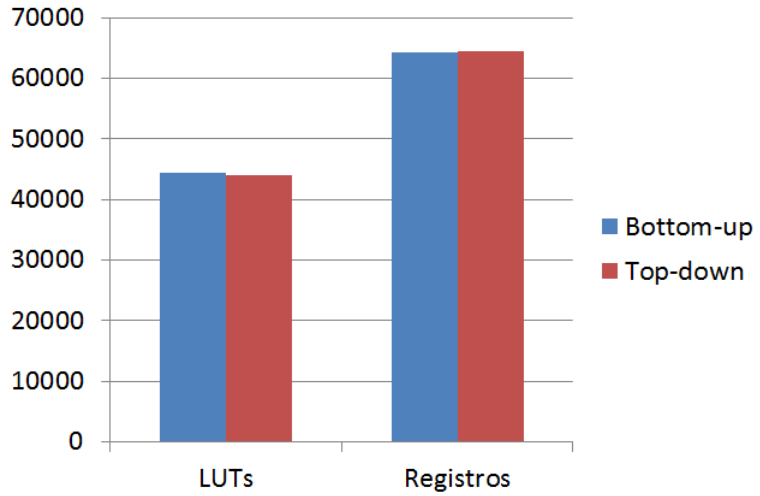

Figura 43. Comparativa del número de LUTs y registros en implementación

5.6. Resultados con Synplify Premier DP

5.6.1. Estrategia *Bottom-up*

- **Utilización de recursos**

Respecto a la síntesis lógica con esta misma estrategia, se produce un aumento de los recursos en LUTs, pero no así en los demás, siendo la LUT el componente más instanciado. En la Tabla 48 se indican los valores de utilización.

Tabla 48. Utilización de recursos en implementación con estrategia *bottom-up*

Recurso	Cantidad	Porcentaje (%)
LUT	36.182	16,55
FLIP-FLOP	21.892	5,00
BRAM	106	19,44
DSP	3	0,33

5.6.2. Estrategia *Top-down*

- **Utilización de recursos**

La utilización de recursos para el diseño completo usando la estrategia *top-down* se indican en la Tabla 49. Al igual que ocurre en la síntesis lógica, tanto en porcentaje como en valores absolutos el recurso más utilizado es la LUT.

Tabla 49. Utilización de recursos en implementación con estrategia *top-down*

Recurso	Cantidad	Porcentaje (%)
LUT	30.142	13,78
FLIP-FLOP	17.730	4,05
BRAM	104	19,08
DSP	2	0,22

- **Análisis temporal**

El peor valor de *slack* obtenido es positivo, 2.586 ns, y se da en la conexión entre *d_filter_top_inst.d_filter_inst.chroma_core_inst.state_filter_2x2_block_3or6_1[5]* y *d_filter_top_inst.d_filter_inst.chroma_core_inst.iclip1_1_iclip1_out[0]*. La frecuencia estimada a partir de ese *slack* es de 134.9 MHz. En la Figura 44 se muestra la ruta crítica.

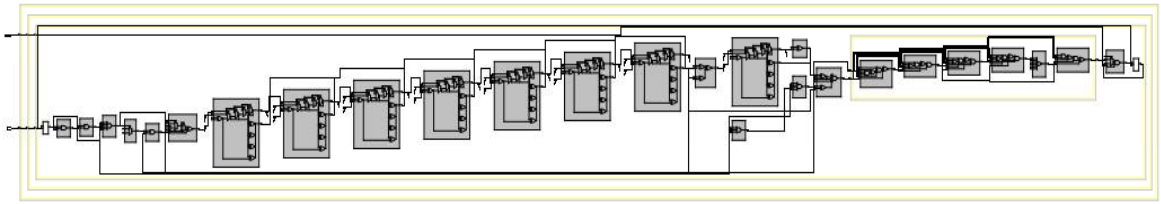

Figura 44. Ruta crítica

5.6.3. Comparativa

La estrategia *top-down* consigue mejores prestaciones en comparación con la estrategia *bottom-up*, como se demuestra en la Tabla 50. *Top-down* consigue una mayor optimización de los

recursos debido a que dispone de toda la jerarquía, sin cajas negras con unos parámetros ya definidos, para realizar la implementación.

Tabla 50. Comparativa *bottom-up* vs. *top-down* en implementación con Synplify

	Bottom-up		*Top-down*	
Recurso	Cantidad	Porcentaje (%)	Cantidad	Porcentaje (%)
LUT	36.182	16,55	30.142	13,78
FLIP-FLOP	21.892	5,00	17.730	4,05
BRAM	106	19,44	104	19,08
DSP	3	0.33	2	0,22

5.7. Comparativa entre Vivado Design Suite y Synplify Premier DP

En la implementación también se encuentran diferencias entre ambas herramientas en relación con el número de recursos utilizados. Cada herramienta utiliza diferentes algoritmos en la etapa de implementación y de optimización, influyendo en la calidad de los resultados que se obtienen.

Los recursos usados para la implementación por parte de Vivado Design Suite son más elevados que en Synplify Premier DP, como puede apreciarse en las Figura 45 y Figura 46 para la estrategia *top-down* y para la estrategia *bottom-up* en las Figura 47 y Figura 48. A excepción de los DSP, de los que se instancian un numero de bloques bajo, en los demás recursos las diferencias son notables, siendo por ejemplo más del doble los registros instanciados por Vivado Design Suite. De igual modo que en la síntesis lógica, los resultados obtenidos con Synplify son mejores que con Vivado.

Por otra parte, en los resultados de utilización de la herramienta Synplify sí se aprecian diferencias entre las estrategias *bottom-up* y *top-down*, consumiéndose menos recursos con esta estrategia. No obstante, *top-down* implica mayor tiempo de cómputo, no siendo los tiempos de cómputo un factor determinante.

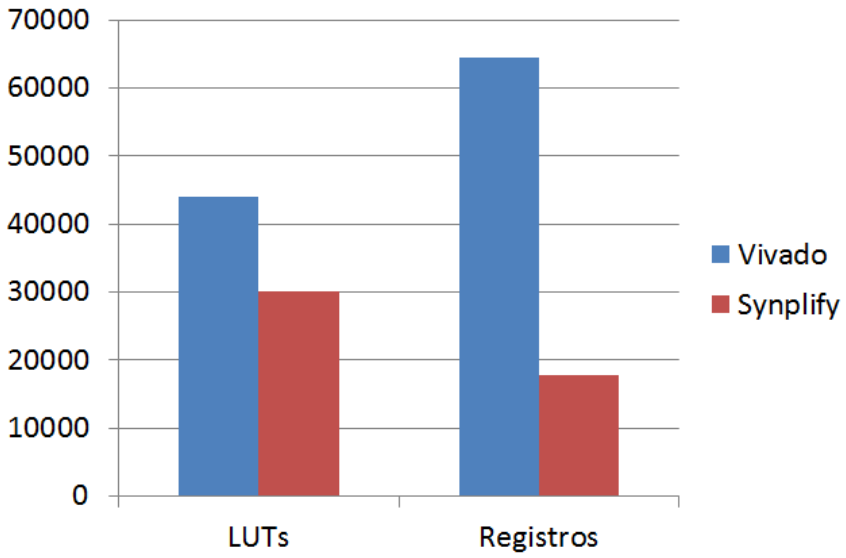

Figura 45. Comparativa LUTs y registros entre herramientas

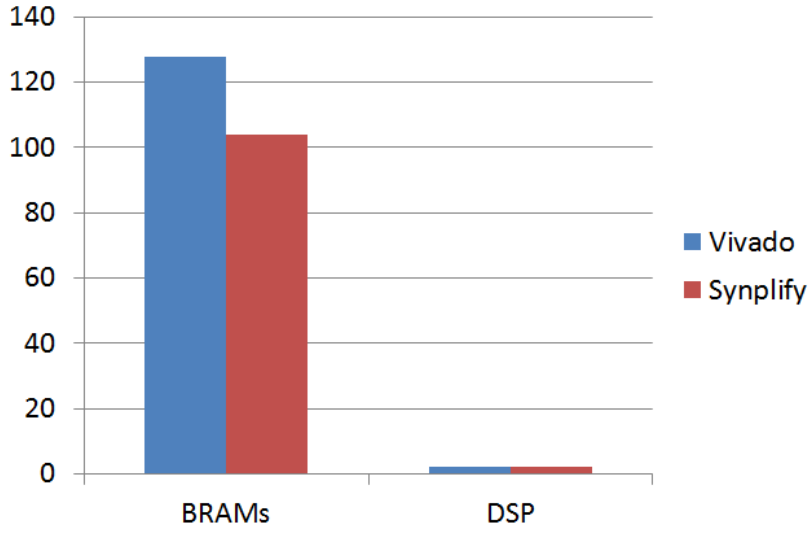

Figura 46. Comparativa BRAMs y DSP entre herramientas

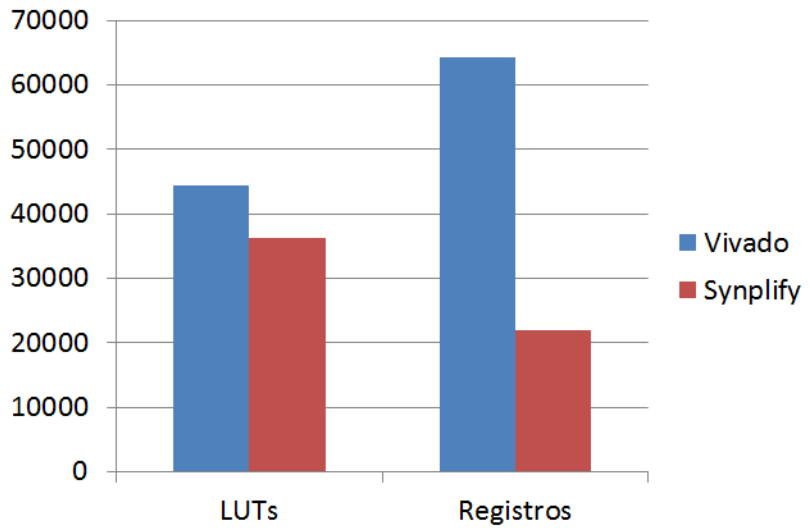

Figura 47. Comparativa LUTs y registros entre herramientas con *bottom-up*

5.8. Comparativa entre Vivado Design Suite, Synplify Premier DP y PCCMUTE

Los resultados de utilización de la implementación obtenidos en el proyecto PCCMUTE [34] se presentan en la Tabla 51 desglosados para el *Deblocking Filter*, el elemento del diseño que más recursos utiliza.

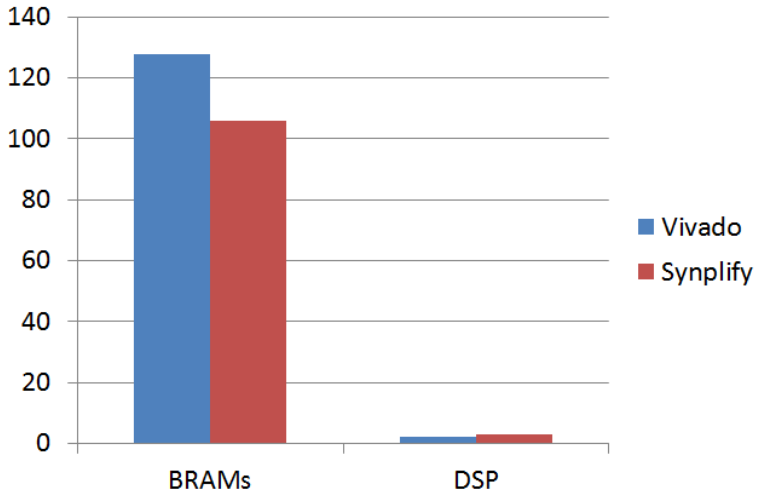

Figura 48. Comparativa BRAMs y DSP entre herramientas con *bottom-up*

Tabla 51. Recursos del *Deblocking Filter* en PCCMUTE

	LUTs	Registros	DSPs	BRAM
d_filter	**27.192**	**18.359**	**1**	**10**
Interface	637	446	0	0
param_BS	9.552	9.523	0	0
param_clip	1.911	877	1	0
luma_core	9.832	4.170	0	1
chroma_core	4.470	3.099	0	1
ll_interface	**336**	**414**	**0**	**0**
Inout_handler	**6.599**	**3.413**	**2**	**0**
Memory	**2**	**0**	**0**	**96**

En las Tabla 52 y Tabla 53 se presentan los resultados de implementación obtenidos para los mismos bloques del diseño con la herramienta Vivado Design Suite y Synplify Premier DP respectivamente, para la misma estrategia seguida en el proyecto PCCMUTE, *top-down*.

En primer lugar cabe destacar que el diseño PCCMUTE se implementó para una plataforma diferente de la empleada en este Trabajo Fin de Máster. La implementación realizada en Synplify Premier DP para lógica programable de la plataforma ZC706 tanto en utilización general de LUTs y registros como de BRAMs instanciados refleja que se utilizan menos recursos. Por tanto, Synplify Premier DP proporciona mejores resultados en cuanto a recursos para el *Deblocking Filter*.

Sin embargo, la diferencia de número de LUTs y registros entre Vivado Design Suite y Synplify Premier DP en el módulo d_filter está relacionada con que Vivado no instancia BRAMs para este módulo, mientras que Synplify instancia en total 8.

Tabla 52. Recursos del *Deblocking Filter* con Vivado

	LUTs	Registros	DSPs	BRAM
d_filter	**39.116**	**60.582**	**1**	**0**
Interface	460	527	0	0
param_BS	8.347	9.485	0	0
param_clip	1.793	2.209	1	0
luma_core	11.353	12.530	0	0
chroma_core	3.472	3.308	0	0
ll_interface	**339**	**414**	**0**	**0**
Inout_handler	**4.384**	**3.392**	**0**	**0**
Memory	**182**	**18**	**0**	**128**

La diferencia entre el número de LUTs y de registros totales del módulo d_filter y la suma de los cinco módulos internos expuestos se debe a las memorias que también se instancian dentro del *Deblocking Filter*.

Tabla 53. Recursos del *Deblocking Filter* con Synplify

	LUTs	Registros	DSPs	BRAM
d_filter	**25.588**	**15.909**	**0**	**8**
Interface	538	317	0	0
param_BS	8.031	7.457	0	0
param_clip	1.901	813	0	0
luma_core	9.503	4.068	0	1
chroma_core	4.893	3.050	0	1
ll_interface	**235**	**269**	**0**	**0**
Inout_handler	**4.244**	**1.546**	**2**	**0**
Memory	**75**	**6**	**0**	**96**

5.9. Script Tcl

Los procesos ejecutados desde el entorno gráfico de Vivado Design Suite pueden realizarse también desde comandos Tcl, que proporcionan un mayor control sobre cada paso. Desde el mismo entorno gráfico se pueden ejecutar estos comandos desde la consola habilitada a tal efecto. Asimismo, la herramienta puede ejecutarse en modo *batch*, procesándose los comandos Tcl a partir de un *scripts* o bien directamente ejecutándolos en un terminal.

En este Trabajo Fin de Máster se ha escrito un *script* Tcl para ejecutar tanto la síntesis lógica como la implementación mediante comandos Tcl. Para ello se arranca la herramienta desde el terminal de usuario en modo *batch* y se llama a dicho *script*, denominado *commands_tcl.tcl* como se indica en la siguiente línea:

```
$ vivado –mode batch –source commands_tcl.tcl
```

A continuación se describen los diferentes pasos del *script*. En primer lugar, se leen los ficheros del proyecto, en este caso Verilog, mediante el comando `read_verilog`.

```
read_verilog [ glob project_10/*v ]
```

A continuación se leen los ficheros de restricciones con el comando `read_xdc`. Si se sigue una estrategia *bottom-up* debe de asociarse el fichero de restricciones al módulo fuera de contexto que corresponda. Para poder asociarlo, previamente hay que crear un conjunto de ficheros (*fileset*), al que se asocia el fichero de restricciones, pasándose el *fileset* como argumento en la síntesis del módulo. A modo de ejemplo se muestra el caso para el fichero de restricciones del módulo de memoria de puerto dual del diseño. Para asociar el fichero de restricciones al *fileset* se usa el comando `add_files`. Además, se pueden definir las propiedades del fichero de restricciones a través del comando `set_property USED_IN`. En el ejemplo se define para que sea tenido en cuenta en la síntesis lógica y en la implementación del módulo de memoria.

```
read_xdc ooc_memory_rtl_constraints.xdc

#Se crea el conjunto para las restricciones de OOC memory_rtl

create_fileset -constrset ooc_memory_rtl_constraints

#Se añade al conjunto el fichero de restricciones correspondiente
```

```
add_files -fileset ooc_memory_rtl_constraints [get_files
ooc_memory_rtl_constraints.xdc]

#El fichero de restricciones se habilita para síntesis e implementación

set_property USED_IN {synthesis implementation out_of_context} [get_files
ooc_memory_rtl_constraints.xdc]
```

Llegado este punto se puede proceder a realizar la síntesis lógica del módulo mediante el comando `synth_design`. Entre sus argumentos se deben indicar el dispositivo para el que se diseña, si se trata de un bloque fuera de contexto, así como las opciones de síntesis. Seguidamente se muestra el comando de síntesis lógica utilizado para sintetizar el módulo de memoria de puerto dual, incluyendo las opciones de síntesis que se seleccionaron. Con la opción *-mode out_of_context* se indica que se trata de un módulo fuera de contexto.

```
synth_design -part xc7z045ffg900-2 -mode out_of_context -constrset
ooc_memory_rtl_constraints -top memory_rtl -flatten_hierarchy rebuilt -
gated_clock_conversion on -directive runtimeoptimized -bufg 12 -no_lc -
fanout_limit 10000 -shreg_min_size 3 -fsm_extraction auto -
resource_sharing on -max_bram -1 -max_dsp -1
```

Después de realizar la síntesis lógica pueden leerse los informes correspondientes. Se han obtenido los informes de utilización y de análisis temporal y de potencia mediante los comandos `report_utilization`, `report_timing_summary` y `report_power`. *datos_salida* es el directorio de salida de los resultados y `utilization_synt_memory.rpt`, `timing_synt_memory.rpt` y `power_synt_memory.rpt` los correspondientes informes.

```
#Informe de utilización

report_utilization -file $datos_salida/utilization_synt_memory.rpt

#Informe temporal

report_timing_summary -file $datos_salida/timing_synt_memory.rpt

#Informe de potencia

report_power -file $datos_salida/power_synt_memory.rpt
```

Para leer los informes se ha utilizado el editor de texto *gedit*. También puede usarse el entorno gráfico para visualizar los informes añadiendo la opción –name <nombre> en los comandos.

Por otra parte, se permite la creación de *checkpoints* en cualquier punto del flujo gracias al comando `write_checkpoint` tanto en la síntesis lógica como en la implementación. En la siguiente línea se muestra la ejecución de este comando tras la síntesis lógica del módulo de memoria.

```
write_checkpoint -force $datos_salida/post_synth_d_filter.dcp
```

El proceso de implementación se realiza en tres pasos: optimización de lógica, colocación y *ruteado*. Cada uno lleva un comando Tcl asociado, lo que resulta transparente al ejecutar la implementación en el entorno gráfico.

Primero se ejecuta el comando de optimización lógica, `opt_design`, a continuación el comando de colocación, `place_design`, y por último, el comando asociado al *ruteado*, `route_design`. Además, después de la etapa de colocación se han realizado dos optimizaciones, una orientada a potencia con el comando `power_opt_design` y una optimización física mediante el comando `phys_opt_design`.

Asimismo, se pueden obtenerse distintos informes. Se ha obtenido el informe de utilización del sistema de reloj mediante el comando `report_clock_utilization`. También se ha obtenido el informe del estado del ruteado una vez se ejecuta la etapa de *ruteado*. Para ello se ejecuta el comando `report_route_status`. Los informes de utilización, de análisis temporal y de potencia se leen de la misma manera que se muestra en el caso de la síntesis lógica.

```
#Optimización lógica
opt_design –directive ExploreArea
#Colocación
place_design –directive SpreadLogic_high
#Informe de utilización del sistema de reloj
report_clock_utilization -file $datos_salida/clock_utilization.rpt
#Optimización de potencia
power_opt_design
#Optimización física
phys_opt_design
#Ruteado
route_design –directive MoreGlobalIterations
#Informe del estado del ruteado
report_route_status -file $datos_salida/post_route_status.rpt
```

El *bitstream* de salida se genera mediante el comando *write_bitstream*.

```
write_bitstream -force $ datos_salida/salida.bit
```

5.10. Conclusiones

Partiendo del *netlist* proporcionado por la síntesis lógica se pasa a la implementación del diseño, considerando las restricciones físicas además de las restricciones temporales. Se implementa siguiendo las estrategias *bottom-up* y *top-down*. Se obtienen resultados en cuanto a utilización de recursos, de análisis temporal y de potencia con Vivado Design Suite. Una vez hecho esto se realizan comparaciones entre ambas estrategias de síntesis.

Posteriormente, se implementa con la herramienta Synplify, se comparan los resultados entre las dos estrategias de síntesis. Asimismo, se comparan los resultados de ambas herramientas. Los resultados obtenidos demuestran que del mismo modo que en la síntesis lógica, en la implementación Synplify también proporciona mejores resultados.

Capítulo 6: Bloques IP

6.1. Introducción

Vivado Design Suite incluye la herramienta Vivado IP Integrator [43], una herramienta que proporciona un entorno para el manejo de bloques IP. Dispone de multitud de bloques IP predefinidos. Algunos de estos se corresponden con la interfaz AXI4 y para la conexión de bloques a través de interfaz AXI4 con memoria mediante DMA. Otros son módulos de interfaz para el *bus* CAN o para el *bus* PCI, bloques de memoria RAM, DSP, registros de desplazamiento, *Ethernet*, LTE, filtros, módulos para transformadas DFT y FFT, sumadores, restadores, multiplicadores, divisores, módulos para operaciones en punto flotante, así como para el procesamiento de vídeo e imagen, por ejemplo, se cuenta con un módulo IP corrector del factor gamma. También se dispone de la unidad de procesamiento de la arquitectura Zynq-7000.

Asimismo, la herramienta permite que el usuario genere su propio bloque IP y lo añada al catálogo disponible en la herramienta. El nuevo módulo puede haberse creado partiendo de módulos IP ya definidos anteriormente o habiéndolo diseñado desde el inicio.

En este Trabajo Fin de Máster se ha realizado una emulación del sistema completo en la plataforma Zynq-7000 Z706. Para ello se ha utilizado Vivado IP Integrator, que como se menciona en este capítulo está integrada dentro Vivado Design Suite.

Una vez se ha generado el diseño cumpliendo con las restricciones de las especificaciones se procede a empaquetarlo en un módulo IP.

6.2. Creación de un módulo IP

El procedimiento para crear el módulo IP se indica a continuación. En primer lugar, en el menú *Tools* se selecciona la opción *Create and Package IP*, como se muestra en la Figura 49. En la siguiente opción se elige empaquetar el propio proyecto e incluir los ficheros .xci en el proyecto IP. A continuación se muestra un menú en el que se debe asignar un nombre de proveedor del módulo IP en la opción *IP identification*. Seguidamente se puede generar el módulo IP a través de la opción *Package IP*, que se encuentra señalada en un recuadro de color rojo en la Figura 50.

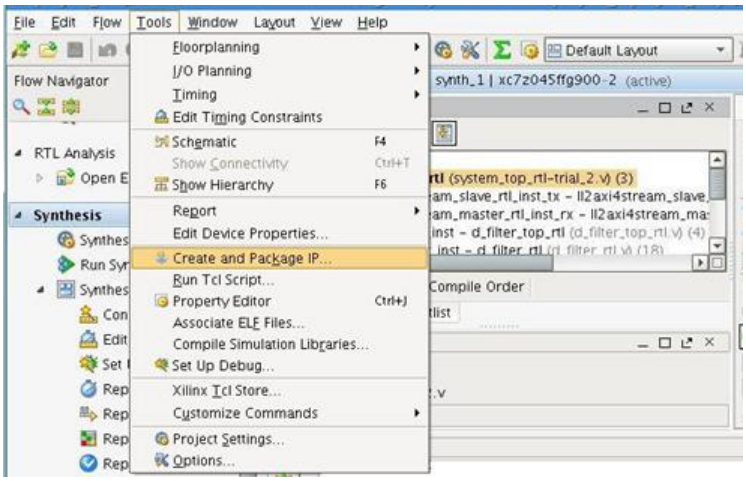

Figura 49. Opción *Create and Package IP*

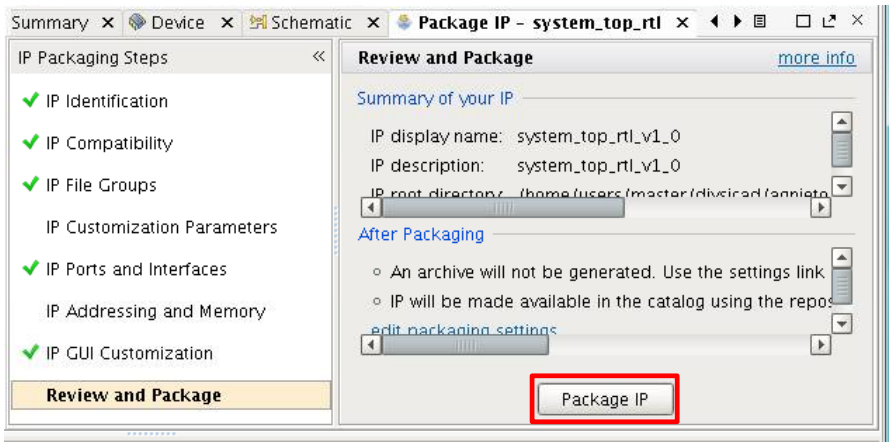

Figura 50. Generar módulo IP

Mediante la opción IP dentro del menú de configuración del proyecto se añade el repositorio correspondiente al módulo IP generado, según aparece en la Figura 51.

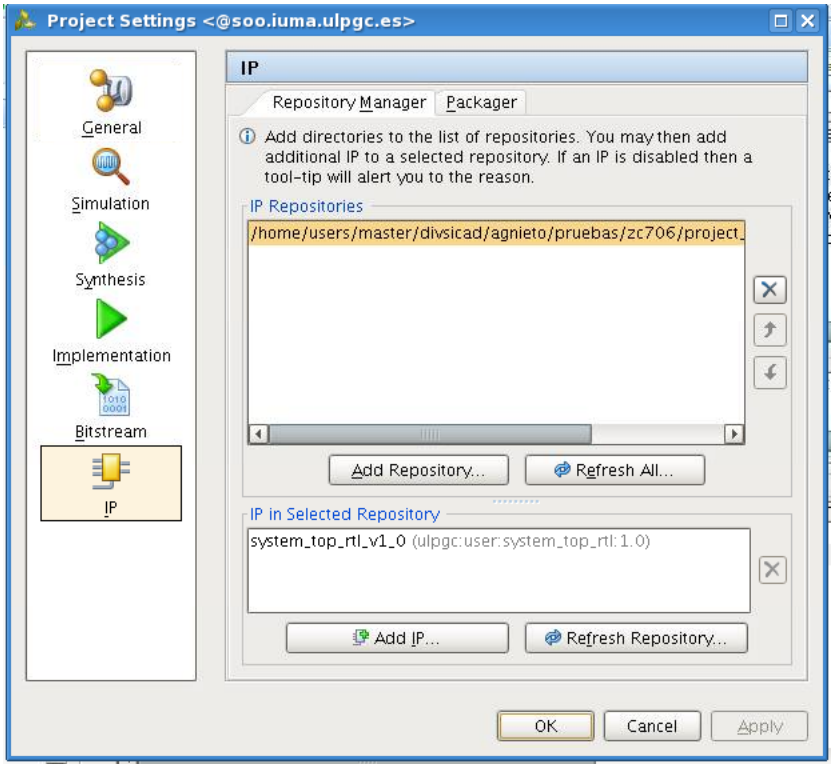

Figura 51. Añadir módulo IP generado al repositorio

6.3. Emulación del sistema

El objetivo de emular el sistema es la representación más fidedigna posible del sistema físico que se ejecuta. Para ello en un nuevo proyecto o en uno ya existente se abre el catálogo de bloques IP [44]. Se debe tener en cuenta que para añadir el módulo generado debe haberse añadido el repositorio en el que esté guardado. En la Figura 52 se muestra cómo puede instanciarse el módulo IP generado a partir del diseño.

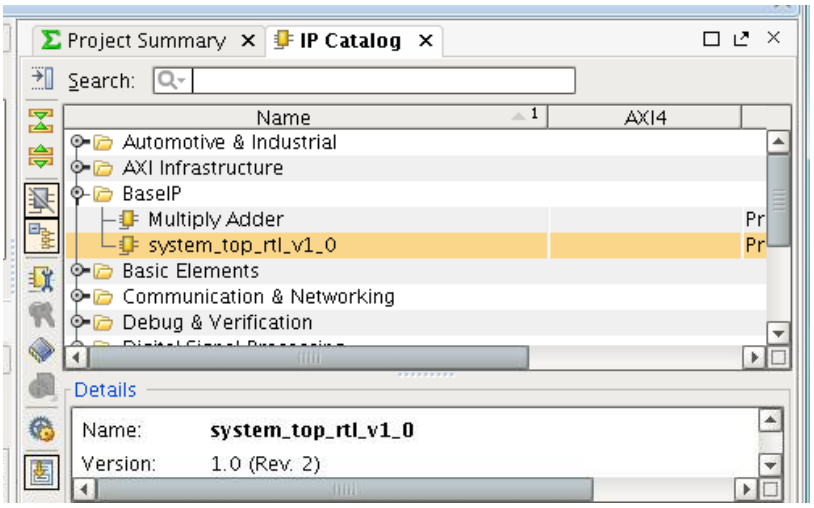

Figura 52. Selección del módulo IP del diseño

Dicho módulo IP se presenta en la Figura 53, una vez ha sido instanciado. Dispone del puerto de reloj *clk*, del puerto de *reset*, y de los puertos de entrada *t_data_tx[0:31]*, *tvalid_tx* y *tready_rx*, y de salida *tdata_rx[0:31]*, *tvalid_rx* y *tready_tx*.

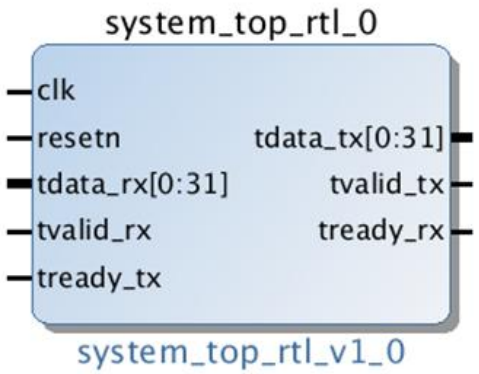

Figura 53. Módulo IP del diseño

Este sistema debe ser conectado al sistema de procesamiento de la plataforma, que ejecutaría un *software* empotrado y sería el encargado de ordenar las transferencias hacia el diseño en lógica programable y de mostrar los *frames* tratados por el *Deblocking Filter*.

Se utilizan interfaces AXI4 para comunicar el diseño en la lógica programable con el sistema de procesamiento. La herramienta proporciona diversos módulos IP para interfaces AXI4. Estos módulos pueden disponer de una de los siguientes tipos de interfaz AXI:

- AXI4-MM: Para interfaces mapeadas en memoria. Soportan ráfagas de hasta 256 ciclos de transferencia de datos.
- AXI4-Lite: interfaces mapeadas en memoria usadas para transferencias simples.
- AXI4-Stream: Soportan ráfagas sin límite de tamaño.

No obstante, las plataformas Zynq-7000 disponen únicamente de interfaz AXI4-MM para comunicación entre la lógica programable (PL) y el sistema de procesamiento (PS).

Se utiliza el bloque IP AXI_DMA para comunicar entre la unidad de procesamiento (PS) y el bloque IP del diseño. El bloque AXI_DMA [45], mostrado en la Figura 54, permite transferir paquetes entre interfaz AXI4 y AXI4-MM-Stream.

De este bloque se utilizan la interfaz maestra AXIS_MM2S y la interfaz esclava AXIS_S2MM. La interfaz AXIS_MM2S es una interfaz AXI4-Stream cuya entrada se adapta a interfaz AXI4-MM, mientras que la interfaz AXIS_S2MM es también una interfaz AXI4-Stream cuya salida se ha adaptado de AXI4-MM a AXI4-Stream. Entonces resulta necesario instanciar un único módulo IP AXI4_DMA, abarcando el mismo los dos canales de comunicación del diseño.

Asimismo, se utiliza la interfaz AXI4-Lite. Desde esta conexión el sistema de procesamiento se comunica con dicho bloque para configuración del mismo o monitorización de transferencia de datos. El módulo IP también dispone de interfaces de lectura y de escritura AXI *Scatter Gather* [46].

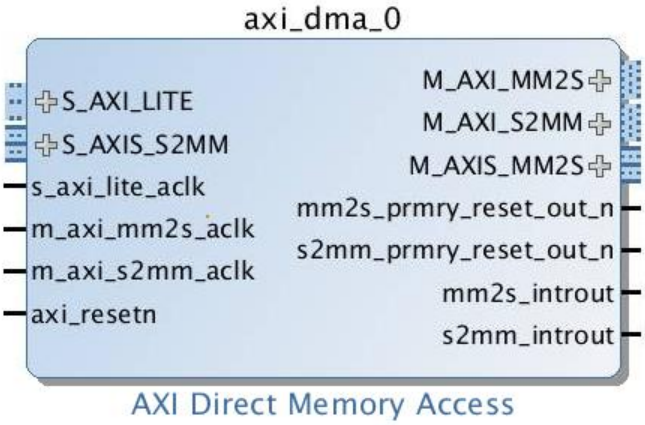

Figura 54. Bloque IP AXI_DMA

A continuación se instancia el bloque IP que representa la unidad de procesamiento (PS) de la plataforma. Este bloque no instancia por defecto las interfaces AXI, por lo que para conectarlo al bloque AXI_DMA se instancia la interfaz AXI HP esclava. La Figura 55 se corresponde con la arquitectura de la plataforma, siendo la parte del sistema de procesamiento (PS) el bloque IP instanciado.

Vivado proporciona la opción *Run Connection Automation*¸ que detecta posibles conexiones entre bloques instanciados con el fin de realizar conexiones de manera automática. Esta opción se muestra en la Figura 56. Se selecciona la conexión entre la interfaz AXI4-MM esclava (S_AXI_HP_0) de PS y la interfaz maestra AXI4-MM2S del bloque AXI_DMA.

Para poder realizarse esta conexión se necesita utilizar el bloque AXI_Interconnect [47] que se encarga de interconectar ambas y es generado automáticamente por Vivado, al igual que el módulo de *reset* de PS, a cuya señal de *reset* se conectan las señales de *reset* de todos los módulos. Este módulo genera la señal de *reset* de todos los módulos instanciados. La interfaz esclava HP AXI4-MM de PS se conecta a la interfaz maestra AXI4-MM de AXI_Interconnect. Desde este módulo se conecta con las interfaces del bloque AXI_DMA.

La conexión de PS con la interfaz AXI4-S2MM se realiza también mediante la opción *Run Connection Automation*. A continuación se conecta el bloque IP del propio diseño con el bloque AXI_DMA a través de las interfaces de este, AXIS_MM2S, para transferencias desde PS hacia el diseño, y AXIS_S2MM para transferir los *frames* procesados hacia PS. En la Figura 57 se muestran estas conexiones. La señal de reloj de todos los módulos se conectan con la señal de reloj de salida del sistema de procesamiento (PS).

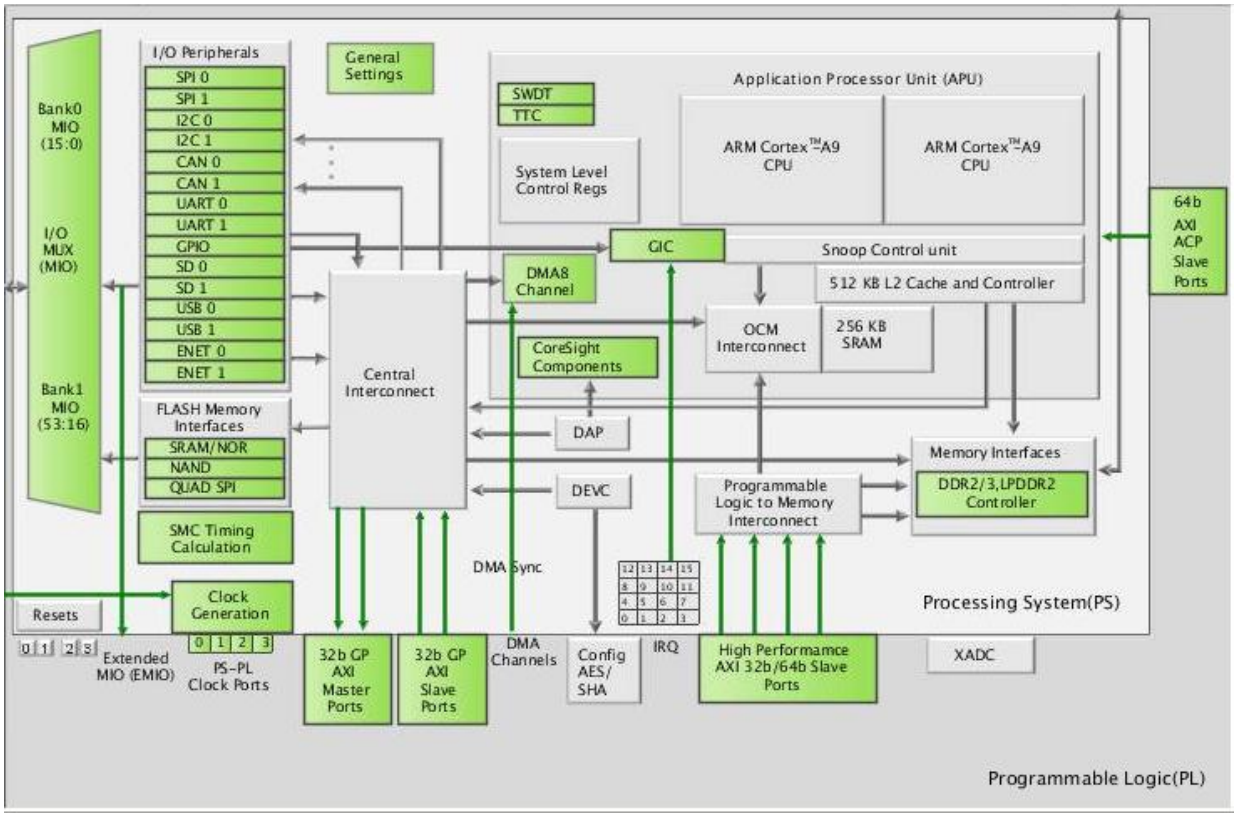

Figura 55. Bloque IP Zynnq-7000

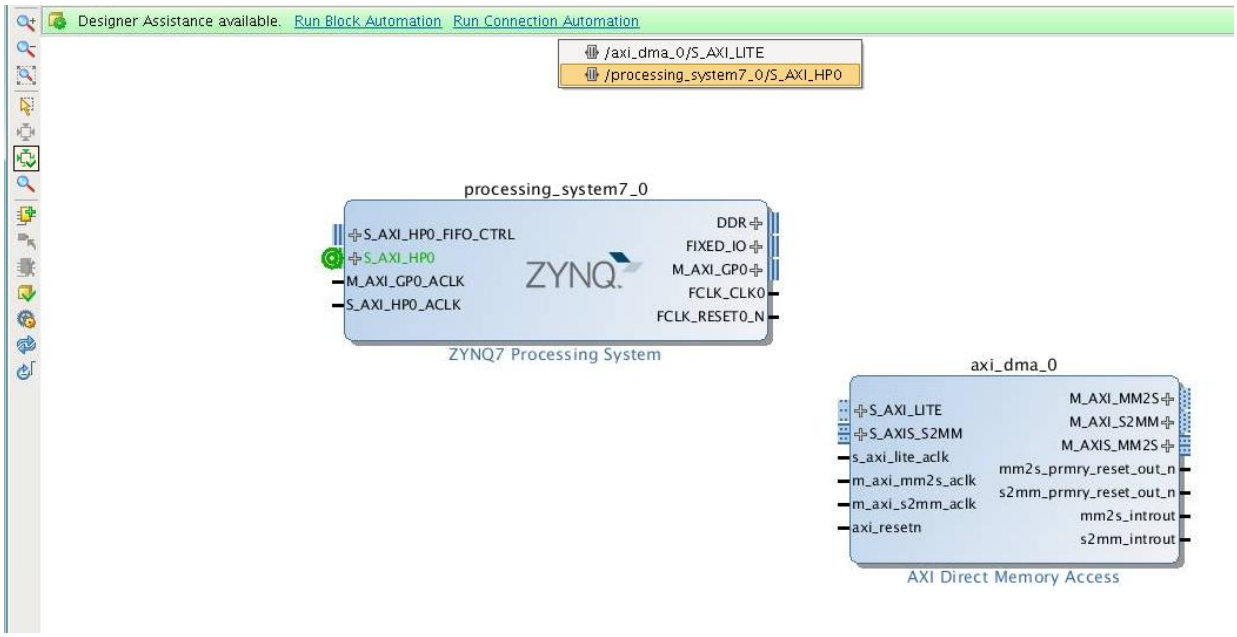

Figura 56. Opción *Run Connection Automation*

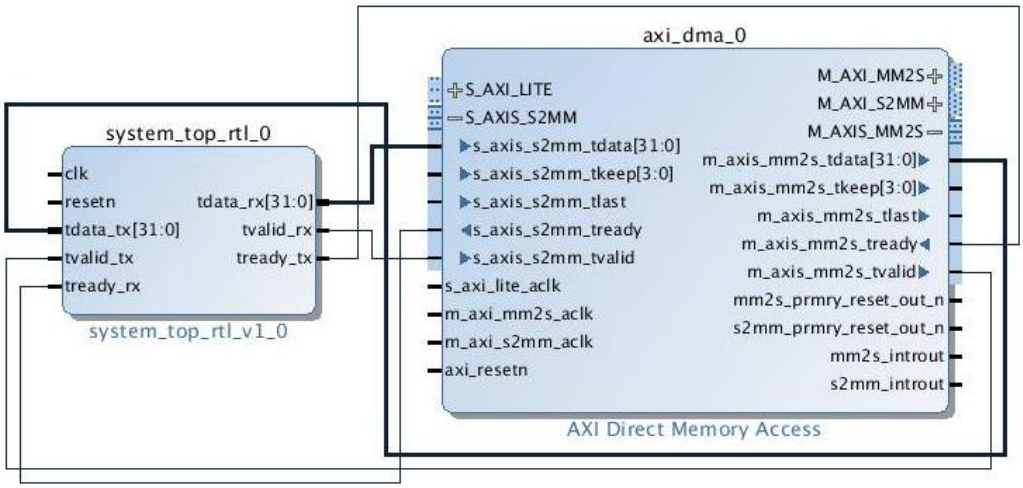

Figura 57. Conexión entre el bloque IP del diseño y AXI_DMA

Asimismo, también por medio de la opción *Run Connection Automation* se conecta PS a la interfaz AXI4-Lite del bloque IP AXI_DMA. Por último, se activa la interrupción la interrupción por parte de DMA, tanto para las dos interfaces que transfieren datos. A su vez, se instancia el bloque IP Concat, cuya utilidad consiste en conectar el puerto para la señal de interrupción de PS con los dos puertos de interrupción del bloque AXI_DMA. En la Figura 58 se muestran el diagrama de bloques utilizados para la plataforma.

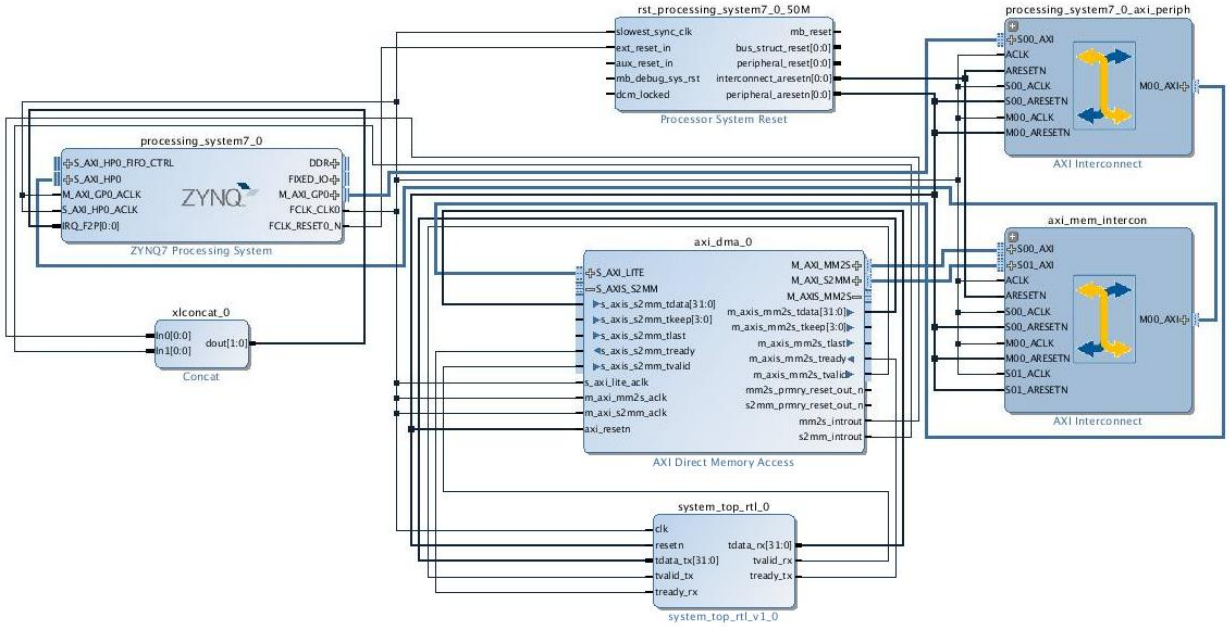

Figura 58. Conexionado de todos los bloques IP del sistema

6.4. Conclusiones

La herramienta Vivado IP Integrator, incluida dentro de Vivado Design Suite, permite trabajar con bloques IP. Se proporcionan de diferentes tipos, aunque este capítulo se centra en los necesarios para emular el sistema completo del SoC, incluyendo el sistema de procesamiento y los bloques de la lógica programable necesarios.

Además, se permite generar bloques IP diseñados por el usuario y añadirlos al repositorio de la herramienta. Así, se ha podido instanciar el bloque IP del diseño. Asimismo, se han instanciado un bloque AXI_DMA que se conecta con las dos interfaces AXI4-Stream del diseño por un lado y por el otro lado se conecta con el bloque AXI_MEM_INTERCON, que hace de puente entre el bloque AXI_DMA y el sistema de procesamiento. Además, se instancian el bloque AXI_PERIPH, que hace de puente entre la interfaz AXI-Lite del bloque AXI_DMA y el sistema de procesamiento, y el bloque de *reset* para el sistema. Por último, se instancia el bloque xlconcat, que conduce las dos señales de interrupción correspondientes a cada interfaz del diseño desde el bloque AXI_DMA hacia el sistema de procesamiento.

Capítulo 7: Conclusiones y líneas futuras

7.1. Introducción

En este capítulo exponen las conclusiones generales del Trabajo Fin de Máster documentado en esta memoria y así como posibles líneas de trabajo con la intención de que se aproveche el trabajo realizado.

7.2. Conclusiones

En este Trabajo Fin de Máster se ha partido de una plataforma ZC702, de la serie Zynq-7000, que incluye un sistema de procesamiento y lógica programable, y del diseño en el nivel RTL del proyecto de investigación PCCMUTE. Se ha estudiado la arquitectura de las plataformas de la serie Zynq-7000. Entre las plataformas de esta serie existen diferencias importantes en cuanto a los recursos de lógica programable. Finalmente la plataforma con la que se ha trabajado es el modelo de la misma serie ZC706.

Estas plataformas no soportan la interfaz LocalLink, por lo que ha sido necesario crear *wrappers* para adaptar dicha interfaz a AXI4-Stream.

Se ha utilizado la herramienta Vivado Design Suite para realizar la síntesis lógica e implementación del diseño PCCMUTE con el fin de comprobar los resultados de esta herramienta. Por ello, posteriormente se ha realizado la síntesis lógica e implementación del mismo diseño con la herramienta Synplify Premier DP.

Cabe destacar que los resultados obtenidos con Synplify Premier DP han resultado ser más optimizados que en el caso de Vivado Design Suite.

En lo concerniente a las estrategias de síntesis, Vivado introduce diferencias respecto a la estrategia de síntesis *top-down* tradicional, permitiendo sintetizar bloques en paralelo, que son guiados en el diseño completo por las restricciones del nivel *top*. En este diseño influye de tal manera que los resultados de utilización son similares a los de la estrategia *bottom-up*. No ocurre así con Synplify.

Asimismo, se ha generado el sistema completo mediante bloques IP con la herramienta Vivado Design Suite. Incluye el sistema de procesamiento y los bloques necesarios sobre la lógica programable. La herramienta dispone de diversos tipos de bloques IP. En el contexto de este diseño, interesan los bloques IP relacionados con la interfaz AXI4.

Algunos otros bloques IP están relacionados con el *bus* CAN, el *bus* PCI, bloques de memoria RAM, DSP, registros de desplazamiento, *Ethernet*, LTE, filtros, módulos para transformadas DFT y FFT, sumadores, restadores, multiplicadores, divisores, módulos para operaciones en punto flotante, o para el procesamiento de vídeo e imagen.

7.3. Líneas futuras

Considerando las conclusiones generales del Trabajo Fin de Máster desarrollado se proponen posibles nuevas líneas de actuación a partir de lo ya desarrollado:

- Estudiar el *software* empotrado del diseño de PCCMUTE para el procesador PowerPC y realizar las modificaciones oportunas para adaptarlo la plataforma Zynq-7000 ZC706.

- Generar un *bitstream* del diseño, volcarlo sobre la lógica programable de la plataforma y verificar su validez.
- Hacer funcionar el diseño sobre la plataforma, incluyendo tanto el *software* empotrado como la parte correspondiente a la lógica programable.

Bibliografía

[1] Citrix. (Julio 2013). Q2 2013 Analytics Report – Mobile Music A Growing Content Category. Available: http://blogs.citrix.com/2013/07/15/q2-2013-analytics-report-mobile-music-a-growing-content-category/.

[2] P. Garcia, F. Salgado, P. Cardoso, J. Cabral, M. Ekpanyapong and A. Tavares. A FPGA based C runtime hardware accelerator. Industrial Informatics (INDIN), 2011 9th IEEE International Conference On [En línea]. pp. 805-809. 2011. . DOI: 10.1109/INDIN.2011.6034996.

[3] Xilinx, Inc. Vivado Design Suite. Available: http://www.xilinx.com/products/design-tools/vivado/.

[4] Xilinx, Inc. Vivado High-Level Synthesis. Available: http://www.xilinx.com/content/xilinx/en/products/design-tools/vivado/integration/esl-design/.

[5] Xilinx, Inc. (2014). Vivado Design Suite. Available: http://www.xilinx.com/products/design-tools/vivado/.

[6] IUMA and UPM. (2010). PccMuTe: Power Consumption Control in MUltimedia TErminals. Available: https://www.citsem.upm.es/index.php/proyectos-es?view=project&task=show&id=39.

[7] Xilinx, Inc. Xilinx Zynq-7000 All Programmable SoC ZC702 Evaluation Kit. Available: http://www.xilinx.com/products/boards-and-kits/EK-Z7-ZC702-G.htm.

[8] Xilinx, Inc. ZYnQ-7000 SoC ZC706 Evaluation Kit. Available: http://www.xilinx.com/publications/prod_mktg/Zynq_ZC706_Prod_Brief.pdf.

[9] Xilinx, Inc. Xilinx Zynq-7000 All Programmable SoC ZC702 Evaluation Kit. Available: http://www.xilinx.com/products/boards-and-kits/EK-Z7-ZC702-G.htm.

[10] Xilinx, Inc. Zynq-7000 all programmable SoC. technical reference manual. [En línea]. pp. 1836. 2012. Available: http://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf.

[11] Xilinx, Inc. XILINX. Available: www.xilinx.com.

[12] ARM Holdings plc. (). Cortex-A9 Processor. Available: http://www.arm.com/products/processors/cortex-a/cortex-a9.php;.

[13] ARM Holdings plc. "Snoop control unit," in Cortex-A9 MPCore Technical Reference ManualAnonymous 2008, [En línea]. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0407i/CDDEHDDG.html;.

[14] ARM Holdings plc. PL310 cache controller. 2007. Available: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0246a/DDI0246A_l2cc_pl310_r0p0_trm.pdf.

[15] ARM Holdings plc. PrimeCell generic interrupt controller (PL390). [En línea]. 2008. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0416b/Cacddhga.html.

[16] ARM Holdings plc. (). CoreSight. Available: http://www.arm.com/products/system-ip/coresight/index.php.

[17] ARM Holdings plc. Embedded trace buffer technical reference manual. [En línea]. 2001. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0242b/index.html.

[18] ARM Holdings plc. CoreSight program flow trace architecture specification. [En línea]. 2011. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0035b/I83164.html.

[19] ARM Holdings plc. "Chapter 12. instrumentation trace macrocell," in CoreSight Components Technical Reference Manual (ARM Holdings plc ed.)Anonymous 2009, [En línea]. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0314h/Chdbicbg.html.

[20] Philips Semiconductors, "The I2C-Bus Specification Version 2.1," Enero 2000, .

[21] CAN in Automation (CiA). Controller Area Network (CAN). Available: http://www.can-cia.org/index.php?id=systemdesign-can.

[22] Oregon State University. Serial Peripheral Interface (SPI). Available: http://web.engr.oregonstate.edu/~traylor/ece473/lectures/spi.pdf.

[23] ULPI Working Group, "Introduction to the UTMI+ Low Pin Interface (ULPI)," Octubre 2009, .

[24] Washington University. Gigabit Ethernet. Available: http://www.cs.wustl.edu/~jain/cis788-97/ftp/gigabit_ethernet.

[25] Xilinx, Inc. 7 series DSP48E1 slice. user guide. [En línea]. pp. 58. 2013. Available: http://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf.

[26] Xilinx, Inc. Zynq-7000 all programmable SoC. software developers guide. [En línea]. pp. 93. 2014. Available: http://www.xilinx.com/support/documentation/user_guides/ug821-zynq-7000-swdev.pdf.

[27] Real Academia Española. (). Metodología. Available: http://lema.rae.es/drae/?val=metodolog%C3%ADa.

[28] O. Espino. Acelerador hardware para el procesado de eventos en tiempo real (AHPETIR). [Online]. 2012.

[29] Synopsys. (2014). Synplify Premier. Available: http://www.synopsys.com/Tools/Implementation/FPGAImplementation/FPGASynthesis/Pages/SynplifyPremier.aspx.

[30] M. McNamara. (). Verilog Resources. Available: http://www.verilog.com.

[31] Xilinx, Inc. Vivado design suite. user guide. using tcl scripting. [Online]. 2012. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2012_3/ug894-vivado-tcl-scripting.pdf.

[32] Xilinx, Inc. ML505/ML506/ML507 evaluation platform. Available: http://www.xilinx.com/support/documentation/boards_and_kits/ug347.pdf.

[33] Xilinx, Inc. Virtex-5 family overview. [Online]. 2009. Available: http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf.

[34] P. P. Carballo, O. Espino, R. Neris, P. Hernandez-Fernandez, T. M. Szydzik and A. Nunez. Scalable video coding deblocking filter FPGA and ASIC implementation using high-level synthesis methodology. [Online]. 2013. Available: http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6628307&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6628307.

[35] ITU-T. H.264 : Advanced video coding for generic audiovisual services. Available: https://www.itu.int/rec/T-REC-H.264.

[36] C. Varoqui and C. Galibern. (). What is OpenSVC. Available: http://docs.opensvc.com/introduction.html#what-is-opensvc.

[37] O. Espino. Prototipado sobre plataforma FPGA de un deblocking filter para H.264/SVC. [Online]. 2013.

[38] Xilinx, Inc. LocalLink interface specification. [Online]. 2005. Available: http://www.xilinx.com/aurora/aurora_member/sp006.pdf.

[39] Xilinx, Inc. LogiCORE IP AXI4-stream. 2012.

[40] Xilinx, Inc. AXI reference guide. [Online]. pp. 5. 2011. Available: http://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf.

[41] Xilinx, Inc. Vivado design suite. user guide. hierarchical design. 2014.

[42] Xilinx, Inc. Vivado design suite. user guide. synthesis. Junio 2014.

[43] Xilinx, Inc. (2014). Block-based IP Integration with Vivado IP Integrator. Available: http://www.xilinx.com/products/design-tools/vivado/integration.

[44] Xilinx, Inc. Creating IP subsystems with vivado IP integrator. [Online]. Available: http://www.xilinx.com/training/vivado/creating-ip-subsystems-with-vivado-ip-integrator.htm.

[45] Xilinx, Inc. LogiCORE IP AXI DMA v7.1. product guide. Abril 2014.

[46] National Instruments. (2009). What is Scatter-Gather DMA (Direct Memory Access)?. Available: http://digital.ni.com/public.nsf/allkb/65B0708FE161D8C0852563DA00620887.

[47] Xilinx, Inc. LogiCORE IP AXI interconnect v2.1. product guide. [Online]. Abril 2014. Available: http://www.xilinx.com/support/documentation/ip_documentation/axi_interconnect/v2_1/pg059-axi-interconnect.pdf.