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

Diseño e Implementación mediante Síntesis de Alto Nivel de un IP para el filtrado y clasificación de paquetes TCP/IP

Autor: Benjamín Vega del Pino
Tutor(es): Pedro Pérez Carballo
Antonio Núñez Ordóñez
Fecha: Julio de 2016

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

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

Diseño e Implementación mediante Síntesis de Alto Nivel de un IP para el filtrado y clasificación de paquetes TCP/IP

HOJA DE FIRMAS

Alumno/a:	Benjamín Vega del Pino	Fdo.:
Tutor/a:	Pedro Pérez Carballo	Fdo.:
Tutor/a:	Antonio Núñez Ordóñez	Fdo.:

Fecha: Julio de 2016

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

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

Diseño e Implementación mediante Síntesis de Alto Nivel de un IP para el filtrado y clasificación de paquetes TCP/IP

HOJA DE EVALUACIÓN

Calificación: .....

Presidente Fdo.:

Secretario Fdo.:

Vocal Fdo.:

Fecha: Julio de 2016

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

Índice

Índice de figuras

Índice de Tablas

ACRÓNIMOS

ACP	Accelerator Coherence Port
AMBA	Advanced Microcontroller Bus Architecture
API	Application Programming Interface
APU	Application Processing Unit
ARM	Advanced RISC Machine
ARP	Address Resolution Protocol
ASIC	Application-Specific Integrated Circuit
AXI	Advanced Extensible Interface
BRAM	Block RAM
CAD	Computer-Aided Design
CAN	Controller Area Network
CLB	Configurable Logic Block
CLI	Command Line Interface
CMOS	Complementary Metal Oxide Semiconductor
CMT	Clock Management Tiles
CPU	Central Processing Unit
DDR	Double Data Rate
DMA	Direct Memory Access
DPI	Deep Packet Inspection
DSP	Digital Signal Processor
EDIF	Electronic Design Interchange Format
EMIO	Extended Multiplexed Input Output

ESL	Electronic System Level
FIFO	First In, First Out
FPGA	Field Programmable Gate Array
FMC	FPGA Mezzanine Card
GPIO	General Purpose Input/Output
GPP	General Purpose Processor
GPU	Graphics Processing Unit
HAL	Hardware Abstraction Layer
HDL	Hardware Description Language
HDMI	High-Definition Multimedia Interface
HLS	High-Level Synthesis
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
HW	Hardware
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronics Engineers
IIC	Inter-Integrated Circuit
ILA	Integrated Logic Analyzer
IP	Intellectual Property
IP	Internet Protocol
IOP	Input/Oputput Pheripheric
IUMA	Instituto Universitario de Microelectrónica Aplicada
JTAG	Join Test Action Group

LED Light-Emitting Diode

LPDDR Low Power DDR

LUT LookUp Table

lwIP lightweight IP

LVDS Low-Voltage Differential Signaling

LVCMOS Low Voltage CMOS

MAC Media Access Control

MMCM Mixed-Mode Clock Manager

MIO Multiplexed Input Output

MTU Maximum Transmission Unit

OCM On-Chip Memory

PC Personal Computer

PHY Physical Layer

PL Progammable Logic

PLL Phase-Locked Loop

PS Processing System

RAM Random Access Memory

RGMII Reduced Gigabit Media Independent Interface

ROM Read-Only Memory

RTL Register Transfer Level

SD/SDIO Secure Digital/Secure Digital Input Output

SDK Software Development Kit

SICAD Sistemas Industriales y CAD

SIMD Single Instruction, Multiple Data

SO Sistema Operativo

SoC System-on-Chip

SSTL Stub Series Terminated Logic

SW Software

TCL Tool Command Language

TCP Transmission Control Protocol

TEMAC Tri-mode Ethernet MAC

TIC Tecnologías de la Información y de las Comunicaciones

TLM Transaction-Level Modeling

UART Universal Asynchronous Receiver/Transmitter

UDP User Datagram Protocol

USB Universal Serial Bus

VCO Voltage-Controlled Oscillator)

VHDL Very High Speed Integrated Circuit Hardware Description Language

Capítulo 1. Introducción

1.1 Antecedentes

1.1.1 Seguridad en la red y equipos de DPI (Deep Packet Inspection)

La seguridad en la red es un aspecto crítico en los sistemas de Tecnologías de la Información y Comunicaciones (TIC). Tal como indican Agrawal y Vieira, la cantidad de dispositivos que acceden a Internet ha aumentado un 300% entre 2010 y 2015. Estos dispositivos realizan tareas de almacenamiento de datos y hacen uso de servicios de autenticación y verificación que usan datos personales [1]. Como consecuencia, la densidad de tráfico de Internet ha experimentado un crecimiento acorde a dicho aumento de dispositivos con acceso a la red. La principal vulnerabilidad de estos sistemas es el hecho de que el aumento de los servicios de esta índole no va a la par con la evolución de los sistemas de seguridad, según se detalla en el trabajo de Ahmed y otros [2].

En la actualidad, incluso las grandes empresas que invierten en sistemas de seguridad corren riesgos de sufrir ciberataques debido a los procesos M&A *(Merger and Aquisitions*) con otras empresas que no tienen una política tan estricta. Es común que tras los procesos de M&A las empresas sufran ataques que provienen de hackers que se habían infiltrado y permanecían dormidos en las empresas absorbidas [3].

La diversidad de víctimas de ciberataques denota que no se trata de un problema sólo de empresas y personales de relevancia social y política general, sino un problema extendido para todos los usuarios de la red. Esta realidad impulsa el desarrollo de sistemas de seguridad basados en DPI (*Deep Packet Inspection*), los cuales realizan análisis

exhaustivos en los paquetes de red. Este tipo de análisis se centra mayormente en la inspección y búsqueda de ciertos patrones en el *payload*, de manera que pueden detectarlos desde las capas más bajas hasta la capa de aplicación. Este tipo de análisis en los paquetes de red ayuda a la detección de ataques e intrusiones. Sin embargo, el aumento del ancho de banda en las líneas de comunicación y la densidad de tráfico requieren que los equipos que realizan este tipo de análisis sean capaces de operar con tráfico continuo, a alta velocidad y bajas latencias para no afectar al rendimiento de las líneas de comunicación. Estas características requieren de procesadores de red especializados, con alto consumo de potencia, para su implementación *software*. Los sistemas *hardware* basados en FPGA suponen una alternativa eficiente debido a su adaptabilidad al problema, el aumento de su paralelismo y la reducción en el consumo de potencia, con altas tasas de eficiencia y latencia reducida. Ello supone la atención del sector industrial para la aplicación de los sistemas *hardware* basados en FPGA en estos campos [4].

1.1.2 Aceleradores hardware

La capacidad actual de integración en el diseño de CIs permite diseñar SoCs compuestos por núcleos procesadores y dispositivos programables tales como FPGAs. Este tipo de SoC sirve de soporte para sistemas que requieren flexibilidad y altas capacidades de cómputo, usándose en los sistemas como aceleradores *hardware*. Este tipo de dispositivos permite realizar ajustes de configuración tanto en *software* como en *hardware*, aumentando la flexibilidad del sistema y adaptándose al problema específico [5][6].

La implementación de sistemas que requieren cómputo masivo en soportes FPGA permite reducir los costes en consumo de potencia. El compromiso que se alcanza entre la capacidad de cómputo y el consumo del sistema supone un problema latente en los proyectos de sistemas electrónicos en la actualidad. En la Figura 1 se puede apreciar el gap existente entre las prestaciones requeridas y las que realmente ofrecen los dispositivos.

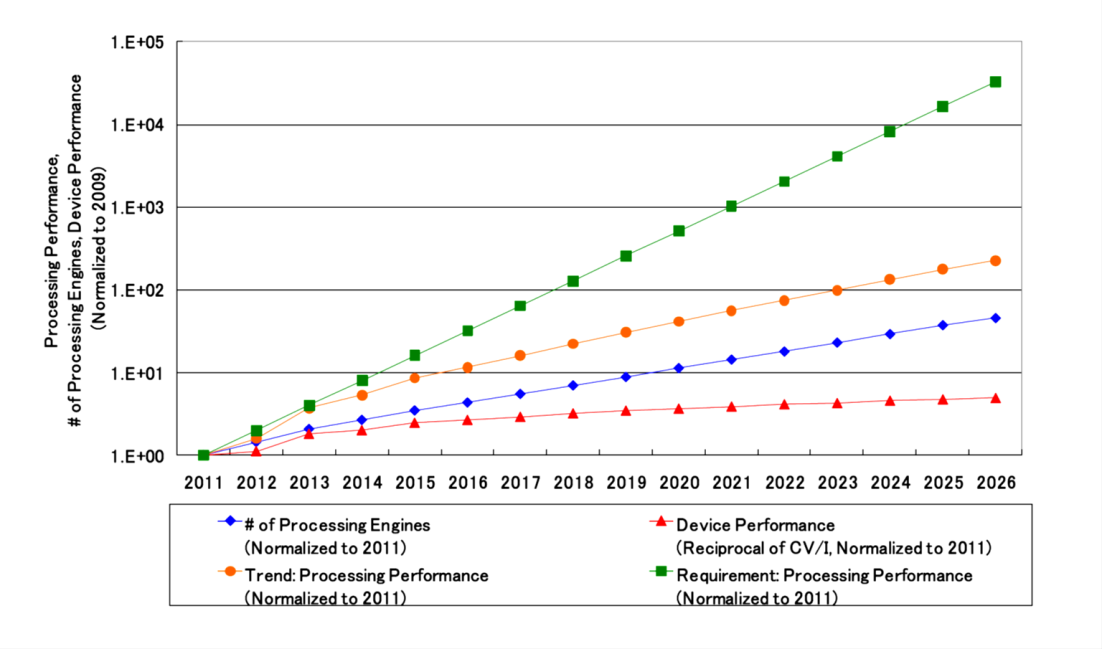


Figura 1. Tendencias en el incremento de necesidades de prestaciones

Las características de los SoCs heterogéneos los hacen óptimos para el desarrollo de una gran cantidad de aplicaciones, como procesamiento de audio, vídeo y sistemas de comunicación digitales de alto ancho de banda, entre otros [7].

La tendencia en el uso de SoCs en el ámbito de las comunicaciones incrementa de forma constante con el paso del tiempo. El uso de sistemas heterogéneos intenta alcanzar nuevos valores de compromiso en las distintas tecnologías disponibles en la ingeniería electrónica y requisitos que demanda el usuario. En la Figura 2, Qualcomm presenta los objetivos de sus investigaciones en innovación y desarrollo de soluciones en los cuales se incluyen los sistemas heterogéneos [8].

Figura 2: Compromisos en los sistemas 5G (Qualcomm)

Intel, tras los resultados del proyecto **Catapult** [9], ha adquirido Altera y en 2016 ha lanzado los primeros procesadores Intel con FPGA en el mismo *socket*, tal como presenta la Figura 3 . Este tipo de procesador está orientado a servidores que dan servicio a sistemas y plataformas para *cloud processing*. Las razones para incluir dispositivos FPGA en procesadores dedicados a servicios de Internet varían según la necesidad de aumento de las prestaciones para mejorar el rendimiento y reducir la carga de los Centros de Datos, incluyendo la reducción del consumo de potencia de los servidores que implementan estos sistemas [10].

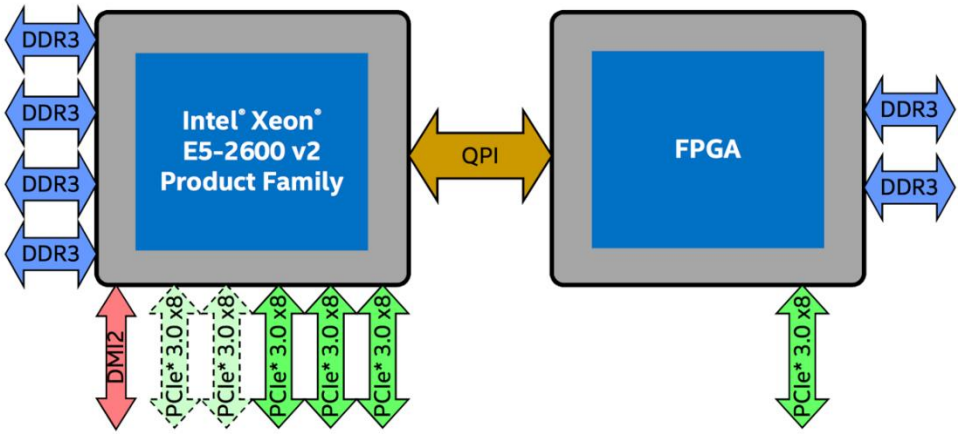


Figura 3: Diagrama Intel Xeon + FPGA

Desde 2014 Xilinx da soporte a soluciones para comunicaciones basadas en FPGAs. Para ello ha creado una herramienta propia que permite desarrollar equipos para la formación de redes **SDN** (*Software Defined Network*) conocida como **SDNet** [11]. Esta herramienta es capaz de generar una plataforma hardware, haciendo uso de núcleos de procesamiento, para sistemas integrables en redes SDN a partir de las especificaciones del diseñador. Esta iniciativa de Xilinx es otro ejemplo de la tendencia al uso de sistemas heterogéneos para llevar a cabo tareas de comunicación digital.

1.1.3 SoC Programable Xilinx Zynq 7000

Xilinx ofrece, dentro de su gama de productos, un dispositivo que implementa dos núcleos de procesamiento ARM Cortex A9 y una FPGA. A esta gama de productos se le conoce como SoC Zynq y los modelos dentro de la serie se diferencian por la tecnología usada en la FPGA y los *hard IP* que integra. Para el bloque lógico programable se utiliza ya sea la familia Artix o a la Kintex, dentro de la serie 7 de Xilinx, dependiendo del dispositivo elegido. Para este proyecto se ha optado por el uso de estos dispositivos de Xilinx debido a que el grupo de investigación tiene un amplio conocimiento y experiencia en las herramientas y dispositivos de Xilinx.

Este tipo de SoC utiliza la infraestructura de comunicación AMBA AXI para comunicar el núcleo de procesamiento con la FPGA. Este protocolo de buses en chip también se utiliza en la comunicación entre bloques IP en la FPGA. El uso de AMBA AXI facilita el diseño de plataforma con IP diseñados por terceros fomentando la creación de sistemas heterogéneos (MPSoC) (Figura 4) [12].

Para el desarrollo de este proyecto se ha utilizado una placa de desarrollo Xilinx ZC706. El dispositivo utilizado (XC7Z045 FFG900 -2) incluye un sistema de procesamiento con dos núcleos ARM Cortex A9 con una arquitectura de 32-bit y una FPGA programable Kintex de la serie 7. Este sistema de desarrollo dispone de recursos suficientes para la implementación de un sistema DPI completo.

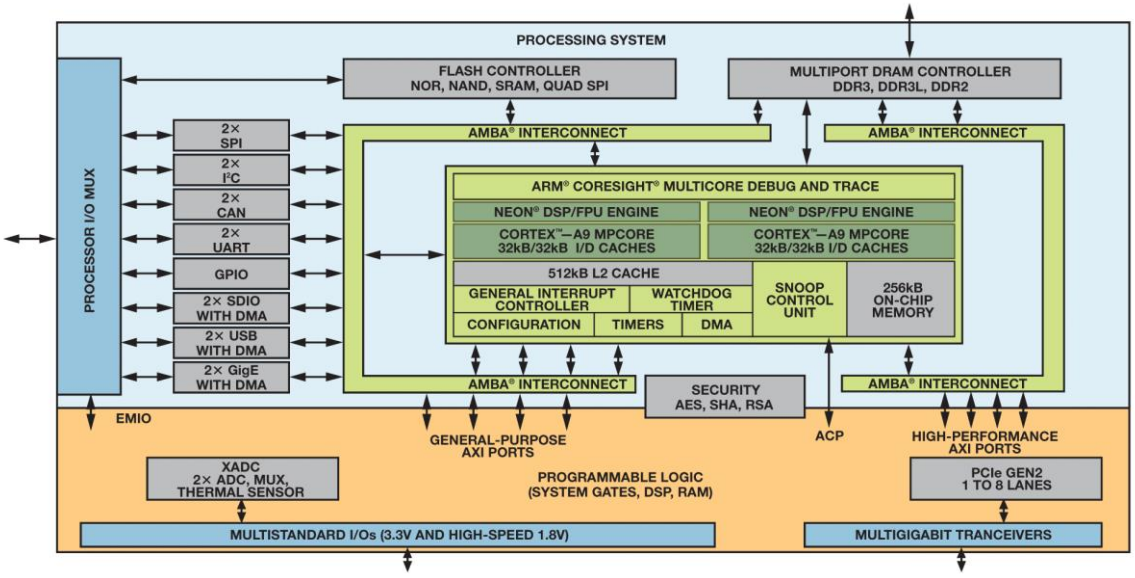


Figura 4. Plataforma Zynq de Xilinx

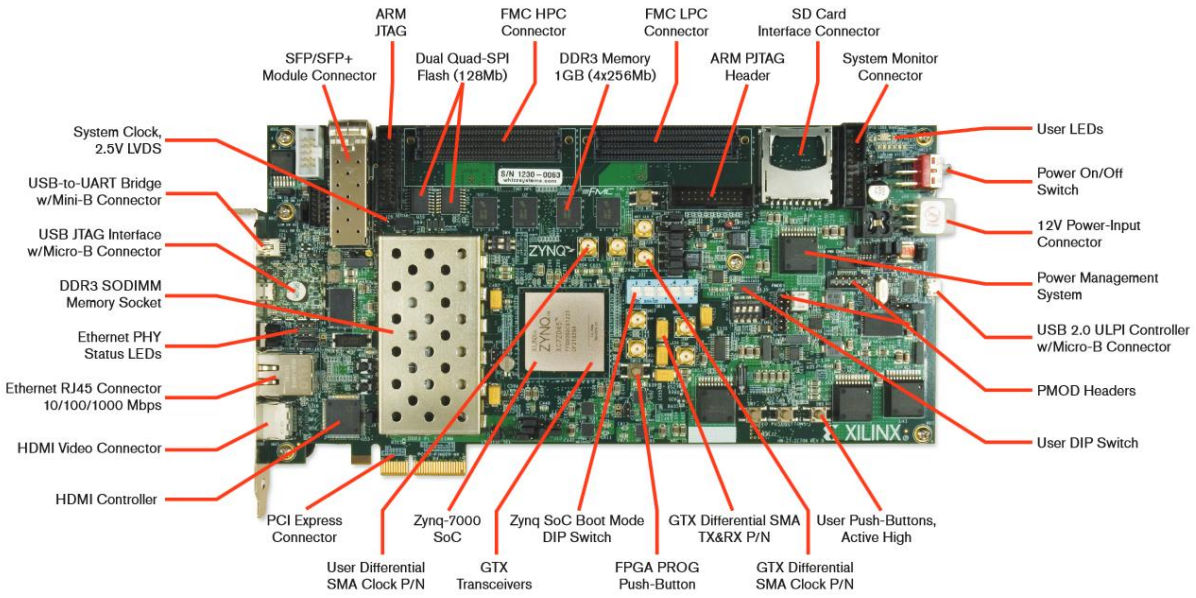


Figura 5. Plataforma Xilinx Zynq-7000 ZC706

1.1.4 Flujo de diseño

La incorporación de uno o varios núcleos procesadores a la FPGA, formando un sistema en chip, complementa la FPGA con los aspectos de flexibilidad aportados por el núcleo procesador. Ello genera una plataforma embebida donde el flujo de diseño se amplía de forma significativa. Se pasa de un flujo de diseño *hardware* basado en HDL

(VHDL/Verilog), que incluye la simulación y síntesis como núcleo central de las actividades de diseño, a un flujo de diseño que incluye varios puntos principales:

1. Análisis de la aplicación y partición *hardware/software* del problema
2. Diseño *hardware* del coprocesador
3. Diseño e integración de la plataforma basada en IPs, donde los buses del sistema forman el núcleo central de la misma
4. Diseño de las interfaces *hardware/software*
5. Diseño e integración del *software* del sistema, ya sea del sistema operativo empotrado como de la aplicación
6. Integración y verificación

Como se puede apreciar, el problema de diseño ahora se mapea tanto en el dominio *hardware* como en el dominio *software*.

Dentro del proceso de diseño indicado anteriormente, uno de los aspectos clave es facilitar el diseño de los elementos coprocesadores. En este sentido, la síntesis de alto nivel supone un avance significativo en el proceso de diseño. Para ello es necesario disponer de flujos de diseño que a partir de C/C++/SystemC sean capaces de generar coprocesadores a medida de la aplicación. Esta flexibilidad añadida permite la reconfiguración de la FPGA para distintas aplicaciones de forma sencilla y transparente para el diseñador de aplicaciones.

En el caso de Xilinx, conjuntamente con la plataforma extensible y configurable Zynq se ofrece un ecosistema de herramientas de diseño que permiten abordar este proceso de forma integrada. Vivado HLS permite, a partir de una descripción C, C++ o SystemC del problema obtener la arquitectura a nivel de transferencias de registros (RTL) del coprocesador y generar interfaces estandarizadas para la integración con el sistema (mapeada en memoria, AMBA AXI, etc.). El flujo de diseño continúa con herramientas que permiten integrar ese coprocesador o IP en la plataforma, implementar todo el sistema en la FPGA, desarrollar el *software* empotrado y depurarlo sobre la plataforma. Este conjunto de herramientas se conoce como Vivado Design Suite (Figura 6) [13].

La metodología usada en este TFM modela el funcionamiento del bloque en SystemC. Haciendo uso de herramientas de síntesis de alto nivel, se obtiene la microarquitectura a nivel de transferencias de registros – RTL – haciendo uso de la herramienta C-to-Silicon [14]. La síntesis lógica del bloque se lleva a cabo en Synplify realizando los procesos de mapeado y optimización de la ruta crítica y por tanto de la frecuencia de funcionamiento [15]. Finalmente, la integración de los bloques IP para generar la plataforma se realiza en Vivado Design Suite. Esta metodología está respaldada por los resultados de varios proyectos realizados [16].

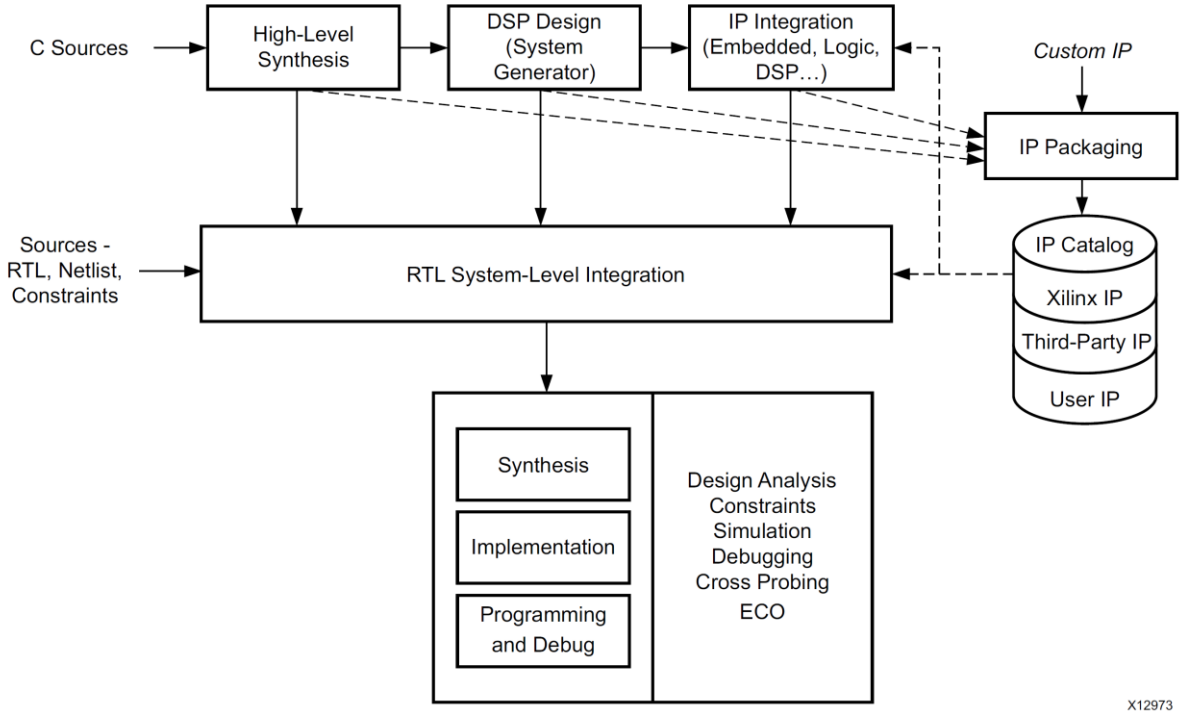


Figura 6. Flujo de diseño con Vivado Design Suite [17]

1.2 Objetivos

En un sistema de inspección profunda de paquetes de red (*Deep Packet Inspection*, **DPI**), la latencia del sistema es determinante. Este tipo de sistemas generalmente usan una arquitectura en la que los paquetes entrantes se almacenan en la RAM del sistema y son procesados mediante una aplicación *software* ejecutada en el procesador o direccionados a un acelerador *hardware*. La arquitectura requiere de un DMA que permita la escritura en RAM de los paquetes entrantes desde la interfaz de red

y el acceso en memoria a los paquetes que se precisan retransmitir, así como de un *software* de gestión de la interfaz de red.

En este Trabajo Fin de Máster se ha desarrollado un bloque IP que permita la conexión directa con un bloque controlador MAC (**TEMAC**), de manera que el bloque realizado tenga acceso directo al flujo de datos entrante y saliente del controlador Ethernet (Figura 7). Este bloque tendrá como finalidad realizar la detección y filtrado de paquetes por contenidos en las cabeceras MAC y Ethernet correspondiente a la capa de Enlace y la capa de Internet en el modelo TCP/IP.

La motivación que impulsa el desarrollo de este bloque en *hardware* es prescindir de la implementación *software* del *stack* TCP/IP necesario para la captura de paquetes. El *stack* TCP/IP gestiona las diferentes capas, realiza la configuración de las interfaces de red y la programación de un socket del tipo RAW para la captura de los paquetes que fluyen a través de la interfaz de red del sistema. La implementación de dicho *stack* es una opción válida en los sistemas en los que la latencia sufrida desde que se recibe un paquete en la interfaz de red hasta que se accede a la información que transporta no es un aspecto crítico. Sin embargo, para el caso de los sistemas de DPI, se requiere tener acceso a los paquetes entrantes en tiempos muy reducidos que no provoquen retardos significativos en la transferencia de los datos monitorizados. Por esta razón será preciso explorar soluciones hardware como la presentada en este trabajo.

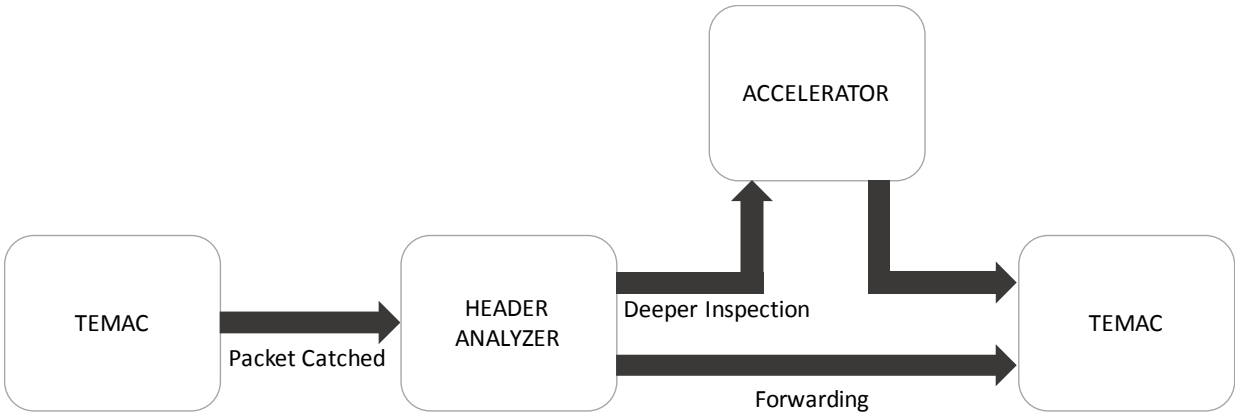


Figura 7. Esquema de conexión del bloque en el sistema

Independientemente de la gestión *software* del paquete capturado, el sistema tiene una latencia importante asociada a la velocidad de acceso a datos fuera del SoC

debido al almacenamiento y lectura del paquete en la RAM a través de los DMAs. Llevar a cabo la tarea de filtrado por protocolo y dirección mediante una implementación hardware (en vez de en *software* a través del *stack* TCP/IP) permite eliminar la latencia de comunicación con la memoria RAM y reduce la latencia asociada a este proceso por su procesamiento mediante *software* en el microprocesador.

El filtrado realizado en hardware se lleva a cabo mediante análisis a nivel de *flit*s en vez de a nivel de paquetes. Esta característica afecta a la arquitectura del sistema, permitiendo así adoptar soluciones del tipo *dataflow*, evitando tener que esperar por la recepción del paquete completo para poder filtrarlo.

El bloque IP que se desarrolla en este trabajo asume las tareas de filtrado de paquetes entrantes según los campos de la cabecera de enlace y de red. Configurando correctamente los parámetros de análisis del bloque, se puede establecer condiciones de filtrado específicas como, por ejemplo, el filtrado por tipo de protocolo, la dirección de origen o conjuntos de parámetros que puedan filtrar una conexión concreta. El análisis llevado a cabo en el bloque sirve como toma de decisión en el que se determina si el paquete continúa en circulación por la red o debe ser redirigido hacia un sistema hardware que extraiga información del *payload (*Figura 8).

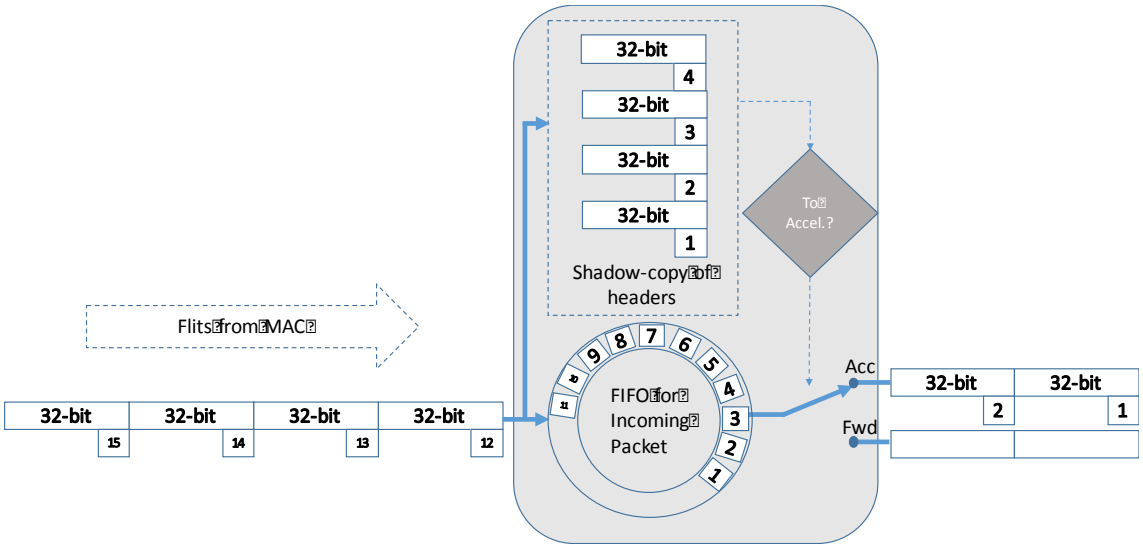


Figura 8. Estructura general de funcionamiento del bloque a desarrollar

El bloque que se plantea y desarrolla en este TFM forma parte de un sistema de DPI para redes TCP/IP cableadas a través de Ethernet. A lo largo del documento se

explican las decisiones de diseño que se han tomado para el desarrollo de dicho bloque IP con el fin de que cumpla los objetivos de reducir el consumo de potencia y latencias con respecto a una arquitectura DPI de referencia implementada usando un *stack* TCP/IP. El resultado del desarrollo de este bloque servirá como base para la implementación global de un sistema que explote las características del proceso de análisis del flujo de datos entrante en hardware.

1.3 Peticionario

El solicitante del desarrollo de este trabajo fin de máster es la División SICAD del Instituto Universitario de Microelectrónica Aplicada, en la línea de trabajo de métodos de diseño de sistemas integrados.

1.4 Estructura del documento

El documento se estructura en capítulos. A continuación se detallará el contenido del documento por capítulos.

El Capítulo 1 recoge una breve pincelada del estado del arte en cuanto a sistemas de seguridad de redes, soluciones usadas para su implementación y la técnica del desarrollo de sistemas *hardware* con el objetivo de acelerar procesos.

El Capítulo 2 recoge los resultados del análisis realizado de la arquitectura de los sistemas de seguridad en la red más usados hoy en día. En base a los resultados y conclusiones alcanzados se propone una arquitectura que ofrezca ventajas sobre la arquitectura estudiada.

El Capítulo 3 describe la metodología seguida para el desarrollo del bloque IP necesario para llevar a cabo la implementación de la arquitectura propuesta en el Capítulo 2. La descripción de la metodología de diseño incluye el análisis, discusión y elección de las herramientas y procedimientos disponibles para llevar a cabo el diseño del bloque IP.

El Capítulo 4 abarca el estudio y modelado de las interfaces de comunicación necesarias para la integración del bloque IP en el sistema sobre el que se prototipa una solución que sigue la arquitectura propuesta. En este capítulo se explica el

funcionamiento general de cada uno de los protocolos implementados y se explica el uso de cada uno de los buses de comunicación en la arquitectura.

El Capítulo 5 recoge el proceso de diseño e implementación del bloque. En él se describe la estructura interna del bloque, las funciones desempeñadas y los resultados de síntesis e implementación del bloque IP.

El Capítulo 6 describe el proceso de diseño de la plataforma de validación y la plataforma que finalmente se utilizará para estudiar casos reales. En este capítulo se describe el diseño de la plataforma, la funcionalidad *software* necesaria para configurar el sistema y se presentan elementos adicionales necesarios para el montaje del prototipo tales como interfaces de red adicionales.

El Capítulo 7 recoge la estructura del test de validación y el montaje del sistema para su uso en un caso real. Los resultados obtenidos son particulares del soporte usado para el prototipo del sistema.

El Capítulo 8 esta conformado por el análisis de los resultados obtenidos y las comparaciones con sistemas ya desarrollados. En este capítulo se estudia la mejora en cuanto a prestaciones y consumo que ofrece la arquitectura propuesta frente a la arquitectura que generalmente usan los sistemas de análisis de redes.

El Capítulo 9 presenta las conclusiones del trabajo realizado y se propone una serie de trabajos futuros para mejorar la funcionalidad y prestaciones del bloque IP desarrollado.

Capítulo 2. Análisis del problema

2.1 Introducción

La arquitectura de los sistemas DPI que usan aceleradores *hardware* para aumentar sus prestaciones generalmente comunica el controlador de red, el sistema de procesamiento y el acelerador *hardware* a través de diferentes bloques DMAs. En este tipo de arquitectura se presentan varios cuellos de botella y limitaciones de velocidad en los mecanismos de transmisión del paquete entre los distintos bloques del sistema DPI y almacenamiento en la RAM [18] [19].

En este capítulo se analiza el problema planteado y se propone una alternativa de diseño a la arquitectura usada generalmente. Esta arquitectura alternativa implica el desarrollo de un bloque IP con una serie de características funcionales que eliminen los problemas indicados y que permita aumentar las prestaciones finales. Las características y estrategias tomadas en el diseño del bloque se explican a lo largo del capítulo.

2.2 Análisis general de la arquitectura

En la Figura 9 se muestra la arquitectura de un sistema DPI que utiliza tanto una CPU como un acelerador *hardware* para el análisis del *payload*. Esta arquitectura comunica los controladores de red y la memoria RAM a través de un DMA. De esta manera, la CPU es notificada cuando el paquete ha sido almacenado en la RAM. En ese momento un programa se encarga realizar una consulta de los valores de cabecera del paquete. Es entonces cuando la CPU decide si enviar el paquete al acelerador *hardware* a través de otro DMA o devolverlo a la red a través de la interfaz de red de salida. A este tipo de arquitectura se le denomina de ahora en adelante como **CDBA** (CPU-DMA Based Architecture).

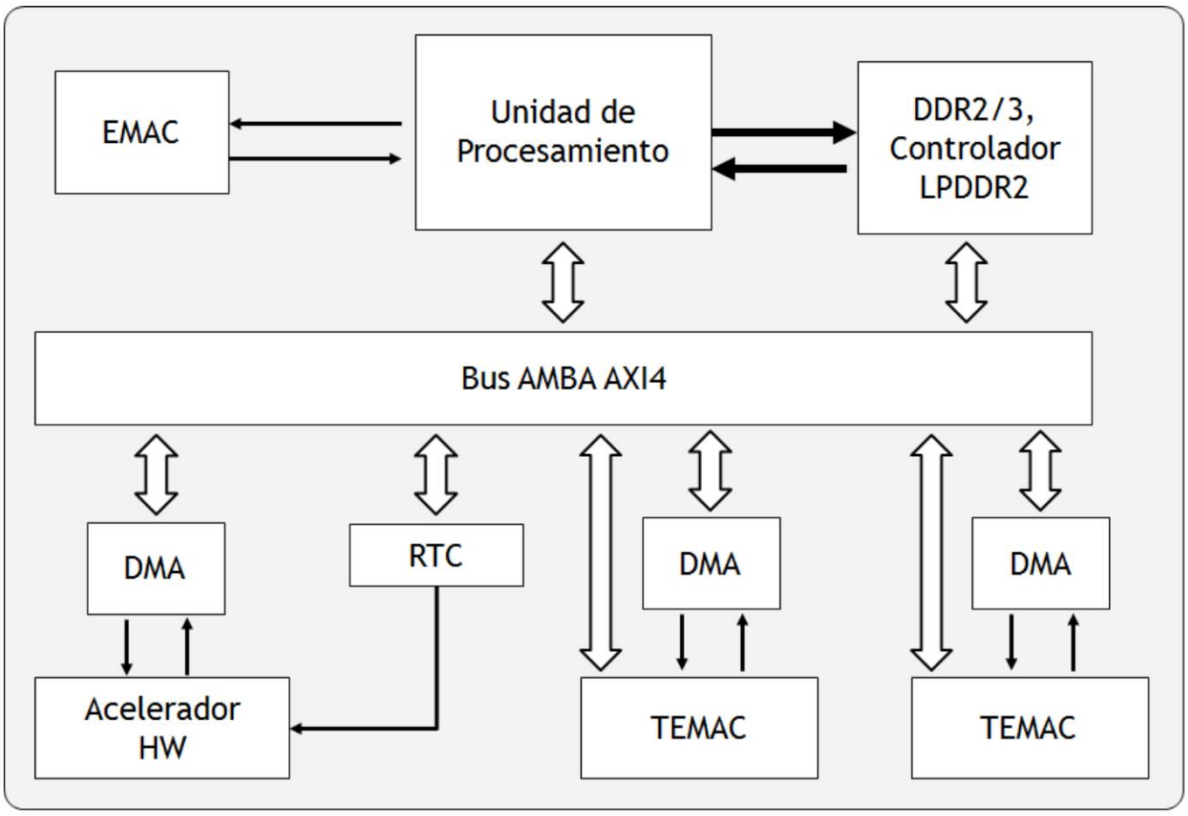


Figura 9: Arquitectura de sistemas DPI

La estructura de este proceso presenta una serie de aspectos que impactan directamente en la latencia del sistema. El hecho de almacenar los paquetes capturados en la RAM implica que, para que la CPU pueda tomar una decisión sobre el paquete, éste debe de haber sido recibido íntegramente por la interfaz de red. Otro aspecto importante es la latencia de transmisión del paquete a la RAM para la toma de la decisión en la CPU. El sistema almacena el paquete completo cuando los campos de cabecera necesarios para la decisión residen en los primeros *filts* del paquete.

La latencia mínima del sistema vendrá determinada por los tiempos de transmisión del paquete a la RAM, el tiempo de decisión de la CPU y el tiempo de transmisión del paquete de nuevo desde la RAM al controlador de red. El resultado dependerá del controlador RAM utilizado y de las características de la memoria. Esta observación refleja las limitaciones que presenta la arquitectura del sistema en el entorno de uso del sistema en una red u otra en base al régimen binario del tráfico que circula por ella.

Este TFM aborda el problema de forma diferente. Para ello se realiza el diseño de un bloque IP que es conectado con la unidad TEMAC y es capaz de extraer información de la cabecera del paquete entrante para su análisis. Por tanto, la inclusión de este IP no precisa de uso de DMAs. Con la implementación del bloque en el sistema se busca reducir el cómputo necesario en la CPU para la gestión del *stack* TCP/IP correspondiente al desencapsulado de las tramas y a la consulta de los campos de las cabeceras. La reducción de carga en la CPU asignada a estas tareas en *software* tiene impacto a nivel de consumo y en la latencia global del sistema. De esta manera, se consigue no solo mejorar las prestaciones, sino también el consumo de un sistema que implementa dicha arquitectura. A esta arquitectura se le denominará de ahora en adelante como **HIBA** (Hardware-IP Based Architecture).

En este TFM se recogen los datos del consumo en la arquitectura CDBA y la arquitectura HIBA que se origina en base a la integración del bloque IP que se desarrolla en este TFM. En base a los datos de latencia se calcula el factor de mejora con respecto a la arquitectura CDBA. A su vez, los resultados obtenidos serán comparados con bloques IP similares analizando las posibles ventajas y desventajas de cada solución.

2.3 Análisis del consumo y latencia de la CPU

Los sistemas de DPI que se estructuran acorde a la arquitectura CDBA capturan todo tipo de tráfico de red y envían al acelerador los paquetes que cumplen las características de inspección configuradas por el usuario. Esto significa que la CPU realiza rutinas de filtrado según los campos de cabecera acorde con el criterio de análisis sobre todos los paquetes que se capturan en la interfaz de red. Además, todos los paquetes que cumplen los criterios de análisis son procesados añadiéndoles una cabecera especial en función de su tamaño y otros factores relevantes para una correcta recepción y tratamiento en el acelerador *hardware* que realiza el DPI.

La evaluación del consumo de la CPU para el desarrollo de esta tarea se realiza teniendo en cuenta el número de ciclos de reloj que toma en ejecutar el código de la tarea en cuestión sobre la CPU. Para la toma de esta medida se utiliza un contador

hardware que comparte la señal de reloj de la CPU y teniendo en cuenta que la CPU no realiza otras tareas durante la medida.

2.4 Análisis del consumo y latencia en transferencias de datos

Una parte importante del consumo energético que se pretende reducir es propiciada por las transferencias en los DMAs. El sistema original recibe el flujo de paquetes de datos en la FPGA; sin embargo, se requiere enviar estos datos a la CPU para el filtrado de paquetes y la preparación del mismo antes de enviarlo al sistema de DPI *hardware*. El impacto energético de esta solución no es sólo debido al uso del bus en sí, sino además a los accesos off-chip a través del controlador de memoria y el consumo de la memoria.

Para evaluar tanto el consumo como la latencia del proceso de comunicación con el DMA, se realiza una medida en ciclos de reloj del tiempo que toma al sistema realizar dicha tarea. La estimación de la potencia será realizada teniendo en cuenta los datos del fabricante de la RAM y teniendo en cuenta el consumo de los bloques IP del DMA. Tomar la medida en ciclos de reloj permite evaluar directamente la latencia de las transferencias de datos entre la interfaz de red, la RAM y el acelerador *hardware* que realiza el proceso DPI.

2.5 Conclusión

La arquitectura que se plantea en este TFM pretende reducir el consumo y la latencia con la implementación del bloque IP desarrollado. La arquitectura planteada difiere de la arquitectura CDBA en el hecho de que la arquitectura HIBA mantiene el paquete en *hardware* desde que se captura hasta que se termina de analizar en el acelerador *hardware* de DPI.

Este cambio con respecto a la arquitectura CDBA provoca una reducción de la comunicación con la CPU, reduce la cantidad de tareas que se ejecutan en la CPU y define un sistema que implementa una estructura *dataflow* desde la llegada hasta la salida del paquete. Esta estructura permite tener el control del flujo entrante y saliente a nivel de *flit,* lo cual permite adoptar estrategias de *pipelining*, *dataflow* y paralelismo[20]. Estas

estrategias permiten realizar el análisis sobre la parte relevante del paquete sin necesidad de esperar a la llegada completa del mismo.

El proceso de consulta de los valores de cabecera de los paquetes se realiza en una copia (*shadow copy*) de los *flits* que almacenan esta información. Esta técnica permite almacenar el paquete completo en una FIFO mientras se realiza el proceso de análisis. Al finalizar el análisis, el bloque permite la salida de los *flits* almacenados en la FIFO y los que aún siguen llegando y que pertenecen al paquete ya analizado.

Capítulo 3. Metodología de diseño del bloque

3.1 Introducción

En este capítulo se explica la metodología de diseño seguida para el modelado, síntesis e implementación del bloque. A lo largo de este capítulo se justificará la elección de las decisiones tomadas y las herramientas utilizadas para el desarrollo de este proyecto. Se analizarán las alternativas de codificación del modelo en alto nivel y se hará un análisis comparativo de las herramientas disponibles para la tarea de síntesis y verificación. Finalmente, se determina qué herramienta de implementación ofrece mayores ventajas para este proyecto.

3.2 Elección de lenguaje de modelado. Justificación del uso de SystemC

Actualmente existe un amplio rango de lenguajes disponibles para la descripción de sistemas *hardware* en alto nivel. En este proyecto se ha considerado el uso de dos lenguajes: C/C++ y SystemC. A lo largo de este apartado se analizarán las características de lenguaje en su uso para la descripción de sistemas *hardware* y se justificará el uso de uno de ellos para el desarrollo de este proyecto.

La elección de un lenguaje u otro radica en el nivel de abstracción deseado y el tipo de bloque que se desarrolla. La elección de uno u otro conlleva una serie de ventajas y desventajas que vale la pena analizar antes de abordar el modelado del bloque IP.

La utilización del lenguaje C presenta la ventaja de que ofrece un nivel de abstracción alto. El diseñador puede capturar la funcionalidad del sistema para crear modelos no temporales que facilitan la realización de la partición *hardware/software*. La arquitectura resultante se obtiene mediante la aplicación de directivas de síntesis al

modelo funcional. El secuenciamiento de operaciones, y por tanto la posible extracción del paralelismo, se realiza en las fases de planificación, una vez extraído el grafo de control y de flujos de datos.

Estas características, aparentemente positivas, pueden presentar desventajas. En este modelo de uso, el control detallado de las interfaces o la planificación de las señales de control de flujo de datos se realiza de forma automática, a diferencia de las metodologías RTL, donde el control es total por parte del diseñador. Las discrepancias entre el modelo original y el modelo RTL obtenido observadas durante el proceso de verificación deben resolverse modificando el modelo funcional original. Esta situación puede implicar cambios en el modelo, cambios en las interfaces o en las estrategias de síntesis, que no son directamente implementables en el modelo C/C++ debido a la ausencia de mecanismos para modelar el comportamiento temporal del *hardware*.

Estas dificultades en la síntesis son complejas de resolver debido a que el diseñador solo tiene acceso a la modificación de un algoritmo que funciona durante la simulación en C/C++. La decisión de utilizar C/C++ para el modelado de sistemas *hardware* tiene como característica que durante las fases iniciales de modelado el diseñador no debe preocuparse sobre latencia, planificación de las tareas ni la utilización de los recursos disponibles. Estas limitaciones se imponen directamente por las herramientas y metodologías de síntesis de alto nivel. Estas características son adecuadas para la implementación de bloques que llevan a cabo funciones puramente algorítmicas y de complejidad reducida.

SystemC es una librería de C++ que permite la descripción de sistemas *hardware* teniendo en cuenta aspectos de funcionalidad, ya incluidos en el lenguaje original, aspectos temporales, y de estructura. El soporte de los aspectos temporales supone uno de los puntos diferenciales de SystemC con respecto a C/C++, ya que el diseñador tiene control sobre la fase de planificación de la síntesis RTL. De esta manera, el diseñador determina el tipo de operaciones que se deben ejecutar cada ciclo de reloj y permite la sincronización de procesos a nivel de ciclos. Para obtener buenos resultados con el uso de estos aspectos de modelado temporal en SystemC se requiere conocer la tecnología en la que se implementará el sistema, ya que debe tener en cuenta el tipo de operaciones que

es capaz de realizar cada ciclo de reloj. Un exceso de operaciones entre ciclos repercutirá de forma directa en la frecuencia de funcionamiento del bloque. Todas estas características hacen que SystemC sea un lenguaje óptimo para el modelado de sistemas cuando la funcionalidad del bloque IP desempeña tareas de control, señalización y desarrollo de protocolos.

El modelado del bloque IP propuesto en el TFM se ha llevado a cabo en SystemC debido a las ventajas indicadas sobre C/C++ en cuanto a la planificación de cada uno de los eventos dentro del bloque. El desarrollo en SystemC del bloque permite llevar a cabo un modelo funcional que sigue una estructura **dataflow,** modelando un sistema que incluye paralelismo segmentando el cauce de procesamiento *(pipeline*). Otros lenguajes de programación para HLS como C/C++ no permiten realizar un modelo con estructuras *dataflow*, dejando en manos del sintetizador la tarea de organizarlos y sincronizarlos para alcanzar dicha estructura. Ello puede producir diferencias entre en modelo funcional y el modelo RTL obtenido.

El uso de SystemC está fundamentado no solo en el control temporal que ofrece a la hora de planificar las tareas, sino también al respaldo que se le da como lenguaje de diseño de alto nivel para sistemas *hardware* de cualquier fabricante. La estandarización de un conjunto de clases y variables sintetizables y las garantías de funcionamiento de la síntesis fortalecen la decisión de optar por SystemC como lenguaje de modelado de sistemas *hardware*. Igualmente existe un proceso de estandarización para un subconjunto de construcciones sintetizables recogidas en el estándar [21] [22].

3.3 Herramientas de Síntesis de Alto Nivel

La herramienta de síntesis de alto nivel juega un papel crucial en el éxito de un diseño en el que se aplica una metodología de diseño que parte de una especificación algorítmica. La elección de la herramienta debe hacerse en función de las ventajas y desventajas que se presentan en función del tipo de diseño a realizar.

A la hora de abordar este TFM se estudió la posibilidad de utilizar dos herramientas y sus correspondientes metodologías de trabajo: Xilinx Vivado HLS y Cadence C-to-Silicon (**CtoS**).

Vivado HLS ofrece un nivel de abstracción adicional en la descripción de funcionalidad *hardware* desde código de alto nivel como C/C++, ya que permite al usuario abstraerse del funcionamiento de los buses de comunicación, las estrategias de planificación seguidas en la síntesis y el tipo de memoria utilizada. No obstante, existen directivas que permiten al usuario incluir restricciones en este nivel para guiar a la herramienta de síntesis. En esta fase de síntesis, la herramienta podrá descartar aquella directiva impuesta por el usuario cuando sean contradictorias o cuando no estén acordes al espacio de soluciones buscado por la herramienta.

En el caso de Cadence CtoS, se solicita de forma explícita al usuario que tome algunas decisiones claves dentro del proceso de diseño, tales como: la forma de implementar los bucles combinacionales (ejecución serial, desenrollado completo o parcial, etc.), las decisiones en cuanto a la implementación de variables y *arrays* como memorias, el *inline* de las funciones o su segmentación para el caso de que pertenezcan a un bucle, así como el proceso de planificación global. Si por alguna razón las directivas que se han usado no son las acertadas para la realización de una síntesis correcta del bloque, el CtoS guiará al usuario en los errores en la síntesis, siempre tratando de generar un bloque acorde a dichas especificaciones introducidas. El entorno de esta herramienta permite una interacción mayor del diseñador, presentando un mayor grado de control en esta fase del proyecto.

También se ha observado como Xilinx Vivado HLS consigue alcanzar frecuencias de reloj más altas que en Cadence CtoS. La estrategia seguida en Xilinx Xilinx Vivado HLS es la de añadir registros en rutas críticas para reducir el número de operaciones planificadas por ciclo. Para aquellos modelos donde las interfaces están definidas a nivel de ciclo y de bits (*Cycle Accurate Bit Accurate Modelling* – CABA) la introducción de registros adicionales (i.e. la introducción de ciclos) puede desincronizar procesos del módulo, perdiendo así la funcionalidad del bloque.

CtoS no toma el tipo de decisiones comentadas, salvo que se indique en la fase de planificación. La optimización de las rutas críticas del diseño se basa en un análisis temporal exhaustivo, modificando en el modelo aquellos aspectos del comportamiento

temporal que no afecten al protocolo. Normalmente este proceso se realiza usando conjuntamente herramientas de síntesis lógica y de síntesis de alto nivel. La Figura 10 muestra una vista del diagrama de funcionalidad de un proceso de un bloque IP durante la fase de síntesis.

Vivado HLS infiere interfaces de comunicación a través de directivas especiales. Esta característica permite modelar sistemas sin tener en cuenta cuestiones de comunicación. Sin embargo, no da la posibilidad al usuario de utilizar la señalización de las comunicaciones para cuestiones de control algorítmico en el modelado del bloque. De momento, CtoS no permite la inferencia automática de interfaces de comunicación. Esto significa que el usuario debe modelar el funcionamiento de la interfaz de comunicación y sintetizarlo como parte del código funcional del bloque. De esta manera, CtoS limita la reutilización del código del bloque para otro tipo de interfaces de comunicación, mientras que en Vivado HLS un reajuste en el valor de la directiva permite adaptar el bloque para trabajar con otro tipo de interfaz de comunicación.

El entorno de HLS utilizado en el desarrollo de este TFM para obtener el sistema RTL del bloque descrito en SystemC ha sido C-to-Silicon (**CtoS**) de Cadence. La elección de este entorno frente a otros como puede ser Vivado HLS es motivada a raíz del éxito alcanzado gracias al grado de control que ofrece CtoS frente a los problemas de planificación, síntesis y dificultades para la simulación con el RTL sufridas con **Vivado HLS**.

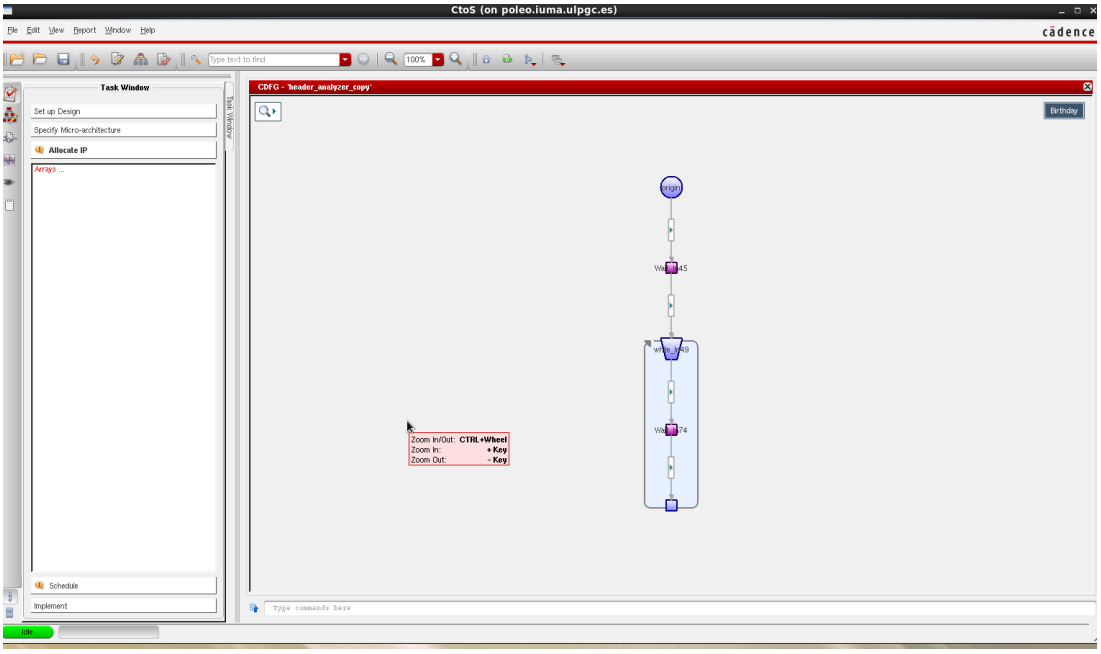


Figura 10: Grafo de control de flujo en CtoS

3.4 Herramientas de síntesis lógica

La realización de la síntesis lógica es el siguiente paso en el flujo de diseño, tomando como entrada la descripción RTL generada en la síntesis de alto nivel. Las herramientas disponibles en el flujo de diseño son Xilinx **Vivado** y Synopsys **Synplify Premier Design Planner**. En este apartado se presentarán las características que se han tenido en cuenta para la elección de una herramienta u otra.

La síntesis lógica en Xilinx Vivado se realiza tras el diseño de toda la plataforma. Esto significa que,a la hora de realizar la síntesis, Vivado tiene en cuenta los requerimientos de recursos de los bloques del resto de la plataforma. Vivado realiza tareas de optimización de recursos de la FPGA y temporal en las rutas entre bloques durante la fase de **P&R** (*Placement and Routing*).

Synplify puede ser utilizado para realizar la síntesis de toda la plataforma utilizando un conjunto de directivas complejas. Igualmente, es preciso ejecutar Vivado para la generación del *bitstream*. Por tanto, en el flujo de diseño se utiliza Synplify para la síntesis lógica del bloque, de manera independiente. Synplify utiliza información acerca de los recursos disponibles en el dispositivo en el que se implementa el bloque y, además, permite planificar la ubicación física de los bloques, previamente al **P&R**.

Ambas herramientas ofrecen directivas para optimizar tanto la ocupación como el comportamiento temporal y las prestaciones. Sin embargo, las estrategias fijadas por Vivado afectan a la síntesis de la plataforma y no se pueden especificar estrategias de manera individual en cada bloque en función de las prioridades definidas por el diseñador en cada uno de ellos.

En el flujo de diseño en alto nivel se hace un uso extensivo del análisis temporal de Synplify, ya que permite analizar las rutas críticas internas del bloque antes de ser implementadas en la FPGA, con una precisión superior a la herramienta de síntesis de alto nivel. Esta información permite volver al modelo SystemC, modificar el código fuente para optimizar la ruta y ejecutar de nuevo el flujo de síntesis en CtoS y en Synplify. Este proceso concluye con la generación de un archivo **EDIF** que contienen el *netlist* del bloque listo para ser integrado en una plataforma e implementado en la FPGA como *Black Box*.

La optimización y la síntesis lógica del bloque se han llevado a cabo en **Synplify**, ya que ofrece la posibilidad de aplicar estrategias de optimización al bloque IP de manera independiente. Una vez sintetizado el bloque, éste se integra con el resto de la plataforma como un ***Black Box*** en Vivado**.** De esta manera, se asegura que las estrategias de optimización seguidas por Vivado no afectarán al bloque IP ya sintetizado y optimizado. La Figura 11 muestra una vista del Synplify en la que representa el sistema descrito en RTL.

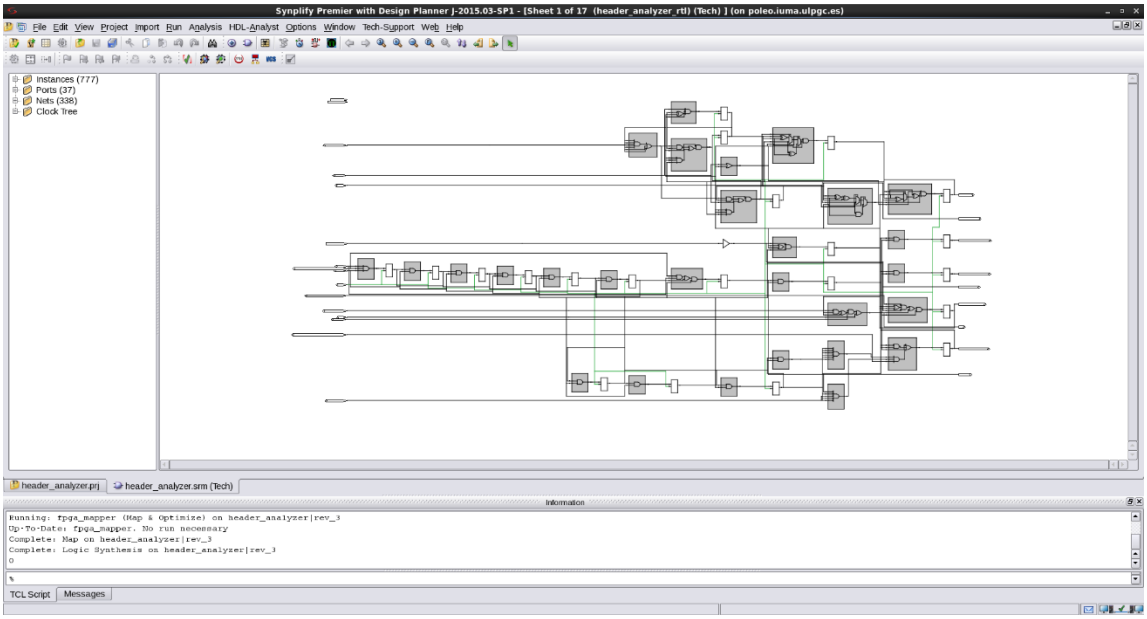


Figura 11: Detalle Synplify Premier Design Planner

3.5 Verificación del sistema

Durante la fase de modelado y síntesis de alto nivel se requiere de un conjunto de herramientas y metodologías de verificación que permitan la comprobación de un correcto funcionamiento del modelo en SystemC y del diseño RTL generado por las herramientas de síntesis de alto nivel.

Para la verificación funcional del bloque se ha utilizado **Incisive** de Cadence. La elección de esta herramienta de depuración frente a utilizar QuestaSim de Mentor radica en la integración del flujo de diseño que presenta con la herramienta de síntesis CtoS.

Incisive ofrece un entorno de verificación que permite la simulación tanto de modelos SystemC, modelos RTL (VHDL y Verilog) y cosimulación de ambos modelos de forma concurrente. Su entorno ofrece varias ventanas que correlacionan la información entre el modelo SystemC, el modelo RTL, la visualización temporal (formas de onda), como del depurador de cada una de las señales del sistema bajo verificación. Además, ofrece un sistema de depuración “paso a paso” que actualiza el valor de las señales del sistema cada ciclo de reloj.

Gracias a las *wrappers* generados por CtoS que integran el modelo RTL, es posible reutilizar el *testbench* del modelo SystemC para verificar el modelo RTL obtenido durante

la síntesis de alto nivel y parte del bloque ya implementado. Este sistema de reutilización del mismo test sobre los distintos modelos obtenidos a lo largo del flujo de diseño permite obtener resultados de verificación coherentes y reduce la posibilidad de cometer errores a la hora de realizar nuevos test en diferentes lenguajes.

El proceso de verificación de la síntesis de alto nivel se ha llevado a cabo instanciando no solo el modelo RTL sino también el modelo SystemC del bloque en el mismo entorno. De esta manera se puede hacer una comparación directa entre el modelo y el sistema obtenido en RTL. En el caso del bloque desarrollado, se puede comprobar como para cada ciclo del reloj, el comportamiento del sistema RTL concuerda con el sistema modelado. Esta característica ofrece garantía de funcionamiento en la plataforma, ya que se ha tenido en cuenta aspectos de la tecnología a la hora de obtener el modelo en RTL.

La metodología de diseño del bloque que se ha elegido, se compone de un conjunto heterogéneo de herramientas. La calidad del resultado del bloque radica en el control de las fases de modelado, síntesis e implementación. Se han analizado distintas alternativas disponibles para cada una de las fases del flujo de diseño del bloque y se han adoptados las más convenientes en cada caso. En la Figura 12 se presenta un diagrama del flujo de diseño y las herramientas usadas.

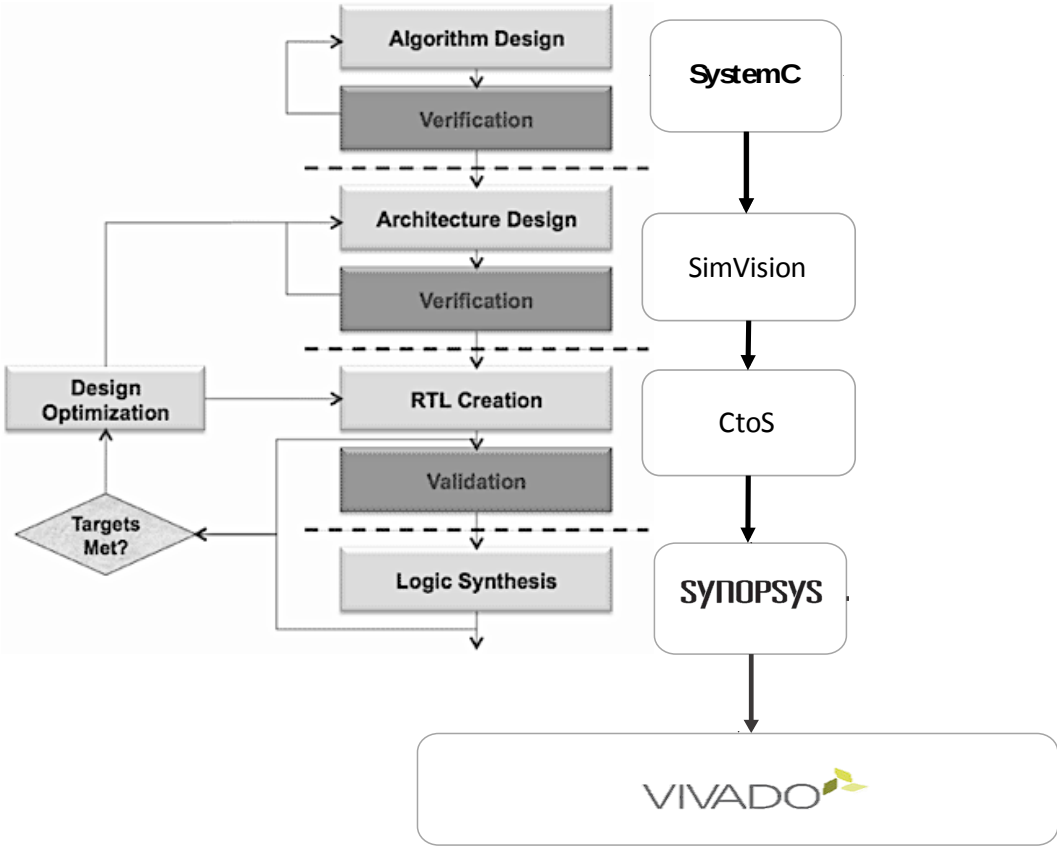


Figura 12: Flujo de diseño

3.6 Conclusión

Para el desarrollo del bloque propuesto en este TFM se ha optado por modelar el funcionamiento del bloque con SystemC. La elección de SystemC frente a otros lenguajes radica en que SystemC permite tener un control más detallado sobre la planificación de las tareas e implementación de las interfaces.

Una vez modelado se realiza la síntesis de alto nivel, obteniendo el código RTL mediante la herramienta CtoS. Se ha decidido utilizar CtoS como herramienta de síntesis por sus resultados de planificación y por las opciones disponibles al diseñar durante el proceso de síntesis.

La implementación del bloque se lleva a cabo en Synplify por sus resultados en optimización de las rutas críticas y aumento de la frecuencia de funcionamiento. Finalmente, la integración de los bloques IP para generar la plataforma se realiza en Vivado Design Suite.

Capítulo 4. Modelado de las interfaces de comunicación

4.1 Introducción

En este capítulo se presentan los protocolos de comunicación que utiliza el bloque para su comunicación con el resto de elementos del sistema. El uso de los protocolos de comunicación viene impuesto por el dispositivo elegido para la implementación del sistema.

La elección de la herramienta de síntesis de alto nivel influye en la metodología de desarrollo de tareas en el bloque IP que se encargan de la comunicación. Como ya se explicó en el capítulo 3. , a diferencia de Vivado HLS, CtoS no infiere las interfaces de comunicación AMBA®AXI de manera automática en el proceso de síntesis. Para implementar el protocolo AMBA AXI se ha creado un conjunto de procesos y funciones para modelar el protocolo en SystemC, lo que ha resultado clave para el desarrollo del modelo del IP. Estos procesos gestionan las señales propias del protocolo a nivel de ciclo y de bit (Cycle Accurate Bit Accurate – CABA) [23].

El modelado y la síntesis de procesos que permitan la comunicación del bloque IP son tareas críticas en el proceso de desarrollo, ya que de ellas depende la capacidad de comunicación del bloque con el resto de elementos de la plataforma. Es por ello por lo que se ha dedicado un capítulo a la explicación del proceso de modelado de los distintos protocolos de comunicación.

4.2 Interfaces de entrada y salida de datos al bloque. Modelo SystemC de la interfaz AXI4-Stream

La elección del protocolo de comunicación utilizado para la entrada y salida de datos del bloque viene impuesta desde la plataforma, ya que los bloques TEMAC tienen interfaces **AXI4-Stream** de entrada y salida de datos para la red Ethernet. Se trata de un protocolo de comunicación orientado al flujo de datos cuya señalización de control utiliza un *handshake* basado en ready/valid.

La implementación de este tipo de interfaces como entrada y salida del bloque bajo diseño requiere el modelado del comportamiento de interfaces maestras y esclavas. El modelado se basa en la interpretación correcta de las señales de control del protocolo en los tiempos de ciclo especificados en el manual de referencia. En la Figura 13 se muestra un ejemplo de una transacción en AXI4-Stream. Cada una de las señales de la interfaz se modela como puertos independientes entre sí. Es en la fase de integración en la plataforma en la que se realiza un proceso de encapsulado que permite empaquetar todas las señales en un bus de datos interpretable por el **IP Integrator** de Vivado. El empaquetado de los puertos como una única interfaz facilita el proceso de interconexión del bloque con el resto de la plataforma.

El modelo desarrollado permite, mediante la creación de procesos SystemC, la lectura y escritura concurrente de los datos en las interfaces. De esta manera, el diseño es capaz de procesar datos de entrada y salida en cada ciclo de reloj sin pérdida de datos. La independencia de las operaciones de lectura/escritura de los procesos de análisis es clave en la fase de síntesis. La ventaja que se observa con dicha independencia es que el número de operaciones planificadas que se realizan en cada ciclo es menor en cada proceso frente a la alternativa de un único proceso que controle la actividad de lectura y análisis. La reducción del número de tareas por ciclo planificado permite, durante la síntesis, obtener frecuencias de trabajo mayores.

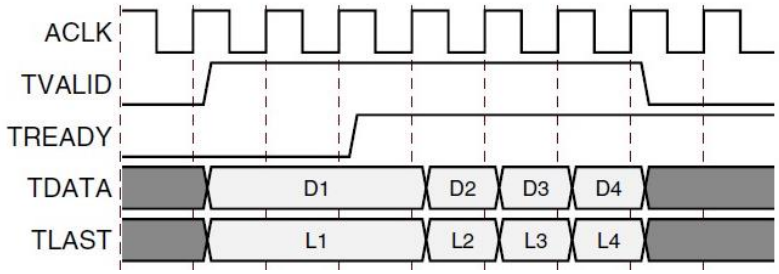


Figura 13: Protocolo AXI4-Stream

Una vez modelado el sistema de comunicación en SystemC acorde a las especificaciones del protocolo AMBA AXI4-Stream, se ha realizado la síntesis de alto nivel. Antes de continuar con el diseño de la funcionalidad del bloque, es preciso realizar un test para comprobar que el modelo RTL generado a partir del modelo SystemC cumple las especificaciones del bus. En la Figura 14 se puede observar como el comportamiento de las señales tanto del modelo de referencia como del diseño sintetizado coinciden a nivel de ciclo y de valores.

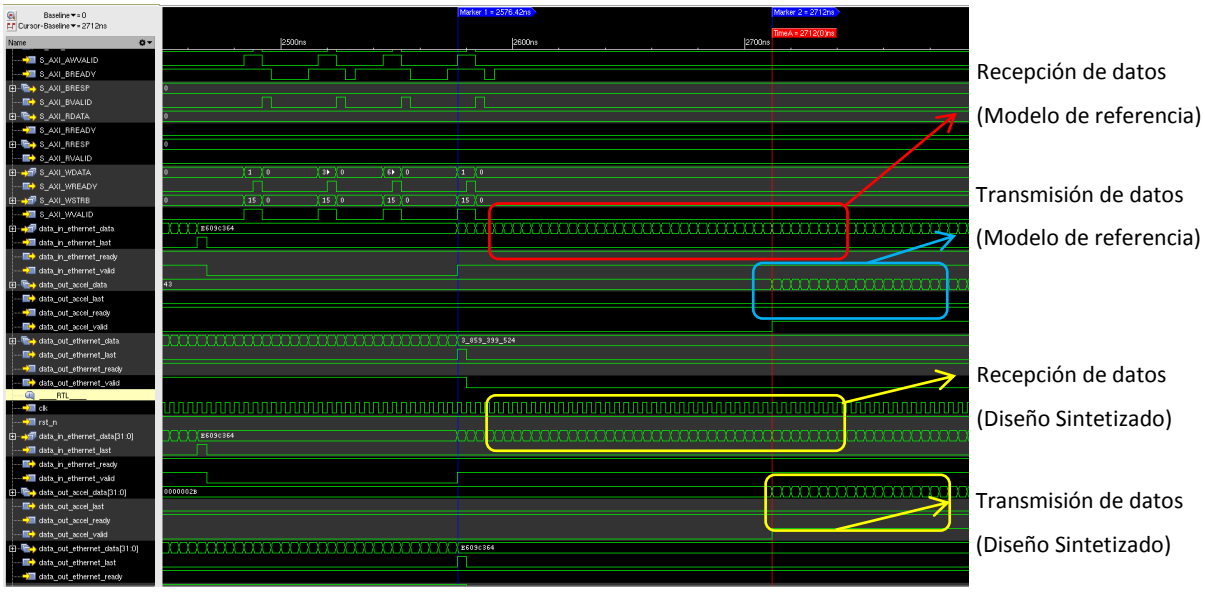


Figura 14: Verificación del modelo AXI4-Stream del diseño sintetizado

4.3 Interfaz de configuración del bloque. Modelo de interfaz AXI4-Lite

El bloque desarrollado realiza consultas en los distintos campos de la cabecera de los paquetes TCP/IP. El mecanismo de análisis se basa en comparación directa de valores de la cabecera de los paquetes entrantes y los valores de los registros de configuración en

el bloque IP. En función del resultado de la comparación, el bloque toma la decisión de enviar el paquete al acelerador DPI o continuar por la interfaz de red de salida.

La manera más cómoda de consultar y actualizar los registros de configuración del bloque, desde el punto de vista del usuario, es mapeando estos registros en la memoria del sistema de procesamiento. De esta manera, el usuario puede gestionar el funcionamiento del bloque desde el *software* empotrado actualizando el valor de los registros.

Para llevar a cabo esta solución se ha optado por implementar una interfaz AXI4-Lite en el bloque IP. La implementación de una interfaz AXI4-Lite permite mapear registros de un bloque IP en la FPGA dentro del mapa de memoria del ARM como un registro más del sistema. Sin embargo, CtoS no permite la inferencia automática de interfaces AXI4-Lite. Para poder añadir esta interfaz al bloque IP se ha desarrollado una interfaz que modela el protocolo de comunicación AXI4-Lite. Esta interfaz ha sido modelada acorde a las especificaciones del protocolo recogidas en el documento de referencia[23].

El desarrollo de esta interfaz se ha realizado desde SystemC, ya que permite el modelado de sistemas con precisión de ciclos y pines necesaria para la implementación de este protocolo. La Figura 15 muestra un ejemplo de una transacción de lectura. Desarrollar esta interfaz de manera independiente al resto del bloque tiene la ventaja de ser reutilizable para otros proyectos, actualizando la estructura y el número de registros en el mapa de memoria sin afectar al funcionamiento principal del bloque que lo instancie.

Otra de las ventajas del desarrollo de la interfaz como un diseño totalmente independiente del resto del sistema es que permite ser instanciado a posteriori como un elemento dentro de otro módulo SystemC. Si se requiere cambiar el protocolo de comunicación, solo será necesario instanciar una nueva interfaz que implemente el protocolo requerido.

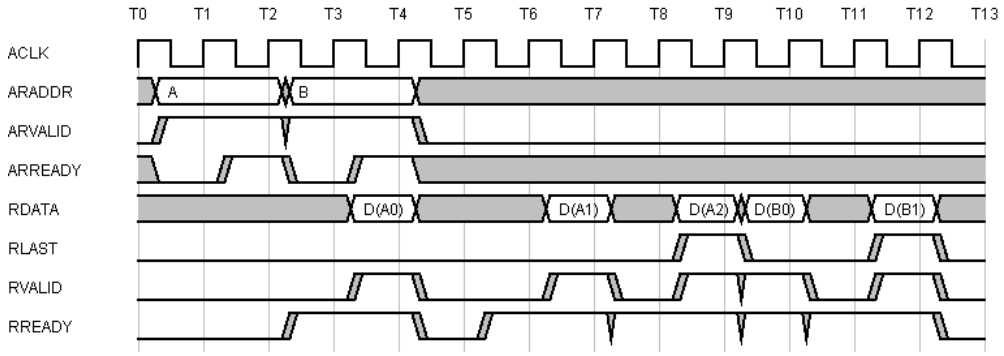


Figura 15: Detalle Protocolo AXI4-Lite para lectura

Para poder llevar a cabo la verificación del bloque AXI4-Lite se ha realizado un test que genera transacciones de escritura y lectura en los registros del bloque AXI4-Lite desarrollado. La actividad de las señales del bus se ha contrastado con la especificación del bus AMBA AXI para comprobar que el modelado se ha llevado a cabo de manera correcta. La Figura 16 muestra en detalle el proceso de verificación en SimVision, en la cual se observa una transacción de escritura a través del bus de comunicación. En dicha figura se puede apreciar como el modelo de referencia coincide con el modelo RTL generado a partir de la síntesis de alto nivel.

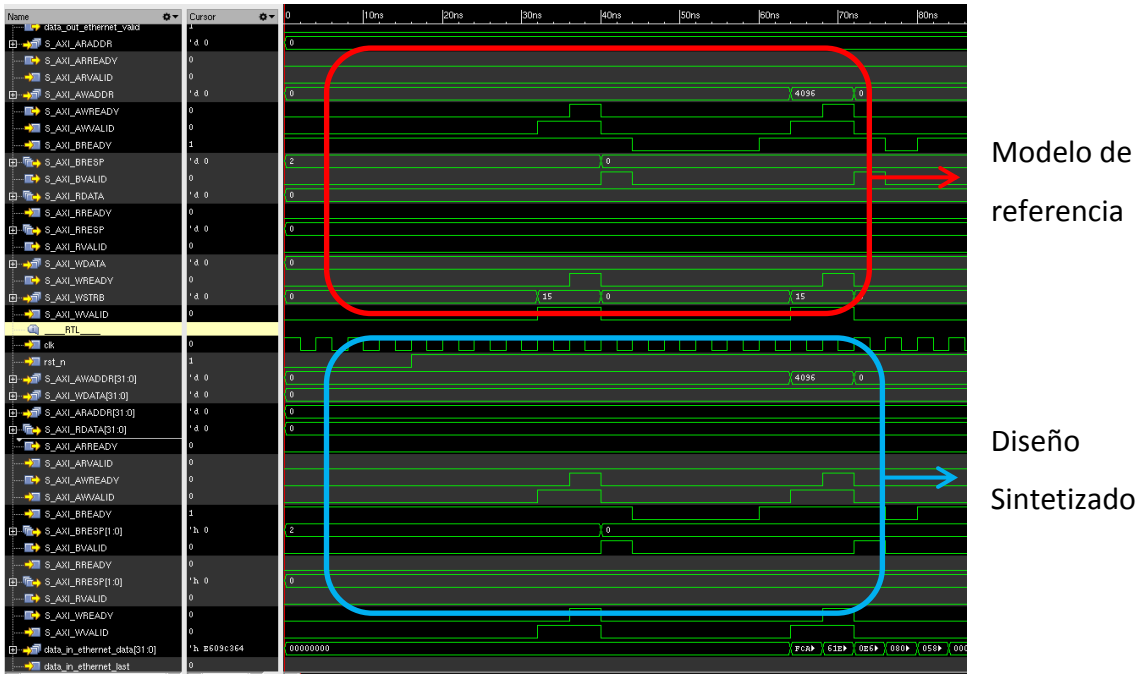


Figura 16: Verificación de una transacción de escritura AXI4-Lite

4.4 Conclusión

La elección de CtoS como herramienta de síntesis de alto nivel requiere el modelado SystemC de cada una de las interfaces de comunicación necesarias en el bloque IP bajo desarrollo. Se ha mostrado el modelado de las interfaces de comunicación AMBA AXI4 necesarias para la correcta integración del bloque en la plataforma. Para ello se ha modelado el protocolo AXI4-Stream para la recepción de datos desde el controlador de red mientras que para acceder a los registros de configuración y funcionamiento del bloque se ha usado AXI4-Lite.

El modelado de estas interfaces requiere un estudio previo del protocolo y un esfuerzo extra en el diseño del bloque que no sería necesario si se utiliza otro tipo de herramienta de síntesis de alto nivel como por ejemplo Vivado HLS. Sin embargo, el esfuerzo invertido en el modelado de las interfaces se gana en la rapidez del proceso de síntesis con CtoS. Más aún, el modelado de las interfaces de comunicación permite hacer uso de las señales del bus para la simplificación de tareas de control, arranque y parada de procesos del bloque.

Capítulo 5. Diseño del bloque IP

5.1 Introducción

En este capítulo se explica el diseño realizado para el bloque IP, así como la arquitectura que se ha elegido para su diseño. Se explica la estructura y funcionamiento de las tareas que conforman el bloque.

Para facilitar la tarea de desarrollo del bloque se ha abordado el modelado subdividiendo el bloque en procesos sencillos. De esta manera el bloque se compone de cuatro procesos que en combinación llevan a cabo las tareas del bloque. Esta metodología de desarrollo permite verificar el correcto modelado de cada funcionalidad implementada sin necesidad de tener un modelo completo. A continuación, se explicará la funcionalidad de cada uno de los procesos.

5.2 Descripción del bloque

Con el objetivo de mejorar las especificaciones temporales del sistema, durante el proceso de diseño del bloque IP se deben de aplicar estrategias que le permitan trabajar con un flujo constante de datos en la entrada mientras se realizan los procesos de análisis de cabeceras. Para ello se ha modelado la tarea de recepción de datos de manera independiente al resto de tareas.

Para la descripción ordenada de cada funcionalidad de los distintos procesos que componen el bloque IP se ha numerado cada proceso. En este apartado se hace referencia a cada uno de los procesos por el número con el que se le ha etiquetado en la Figura 17.

Acorde a la Figura 17, el primer proceso (proceso de copia del dato entrante) controla las señales del bus para permitir la comunicación entre el TEMAC y el bloque en desarrollo. El proceso se encarga de la copia de los *flits* entrantes en una FIFO externa. El proceso utiliza la señal de ***LAST*** del protocolo de comunicación para la identificación del final de un paquete y el comienzo del siguiente.

También se encarga de realizar una copia local de los 10 primeros *flits* de cada paquete (***shadow copy***). Una vez completada la copia de los 10 *flits*, el proceso activa durante un ciclo de reloj una señal. Esta señal es leída por el segundo proceso, que se encarga de realizar el análisis sobre ese conjunto de datos. De esta manera, puede permitir al bloque de análisis comenzar mientras el bloque de copia continúa copiando las palabras siguientes en la FIFO.

La razón por la que se copian los 10 primeros *flits* del paquete se debe a la estructura del paquete TCP/IP, que se representa en la Figura 18. El bloque requiere la recepción de estos *flits* antes de tomar una decisión sobre la redirección del paquete. El requisito de espera por estos *flits* no provoca problemas en paquetes cortos debido a que este tamaño es el mínimo de paquete a recibir, ya que cualquier paquete que pueda llegar a la interfaz de red estará compuesto como mínimo por la capa de enlace y la capa de Internet (o red).

Estos 10 *flits* contienen la información necesaria para realizar el análisis, ya que en ellos se encuentra la información de cabeceras del nivel de enlace y de red. En base a la información recogida en estos campos se podrá tomar la decisión de redirección del paquete hacia la interfaz de red o hacia el bloque de búsqueda en el *payload*. El segundo proceso (proceso de toma de decisión) se encarga del análisis. El proceso de copia y el proceso de toma de decisión están comunicados con una señal interna. El proceso de toma de decisión comprueba cada ciclo de reloj la señal interna controlada por el proceso de copia del dato entrante. Esta señal permanece con el valor lógico ‘0’ hasta que hayan copiado los 10 primeros *flits* del paquete entrante. Una vez ha copiado los *flits*, la señal tiene valor lógico ‘1’ durante un ciclo de reloj. Cuando se activa la señal, el proceso de toma de decisión extrae de cada uno de los *flits* almacenados los distintos campos de la cabecera del paquete. Tras el acceso a los campos de cabecera, el proceso compara el

valor de cada uno de ellos con los valores almacenados en los registros AXI4-Lite. Estos registros son configurados por el usuario y contienen los valores de cabeceras que cumplen el criterio de realizar un análisis más exhaustivo. El proceso de análisis controla el valor de la señal interna que comunica el proceso de toma de decisión y el tercer proceso (proceso de redirección), de manera que es capaz de indicar al tercer proceso si el paquete debe ser enviado hacia bloque de análisis del *payload* o puede continuar por la red. Esta decisión se toma en función del resultado de la comparación de los valores de la cabecera del paquete entrante y de la configuración establecida. Si algún valor coincide con la configuración, la decisión que se toma en el proceso de redirección es la de enviar el paquete al bloque de análisis para una inspección más detallada.

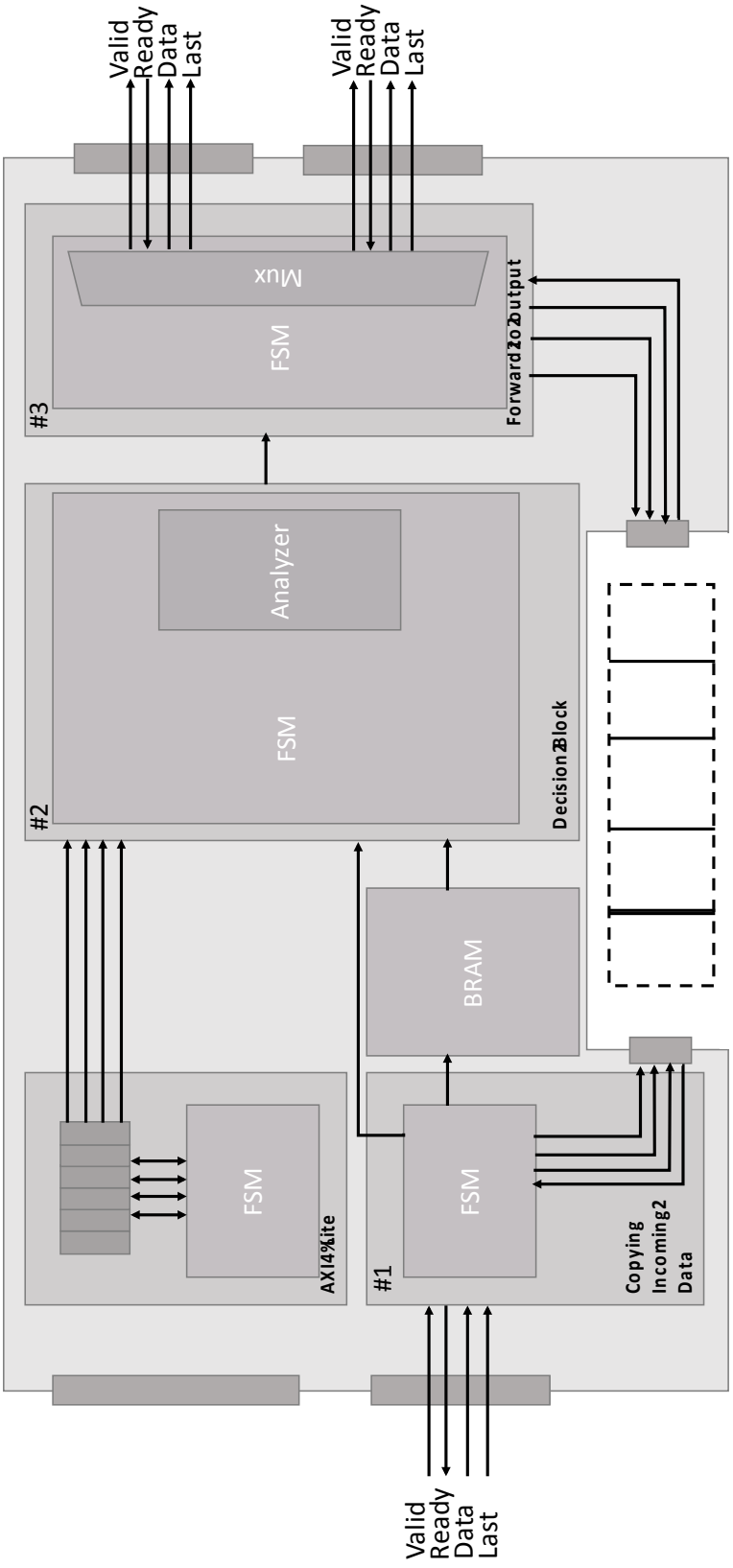


Figura 17: Estructura detallada del bloque en desarrollo

El tercer proceso lleva a cabo la función de redirección. Este proceso se encarga de leer la señal generada por el proceso de análisis y enviar el paquete almacenado en la FIFO a la interfaz de salida adecuada. Este proceso comienza a leer la FIFO y genera la señalización adecuada en el bus para la correcta transmisión de cada uno de los datos leídos. Durante el proceso de lectura de la FIFO, probablemente el primer proceso ya haya comenzado a almacenar los *flits* de un nuevo paquete. El proceso de redirección es capaz de leer exactamente un paquete sin leer los *flits* del paquete siguiente. Esta autonomía del proceso de redirección se ha conseguido gracias al mecanismo de almacenamiento de paquetes adoptado en el diseño de este bloque. Este mecanismo se explica en el apartado Estructura de Memoria.

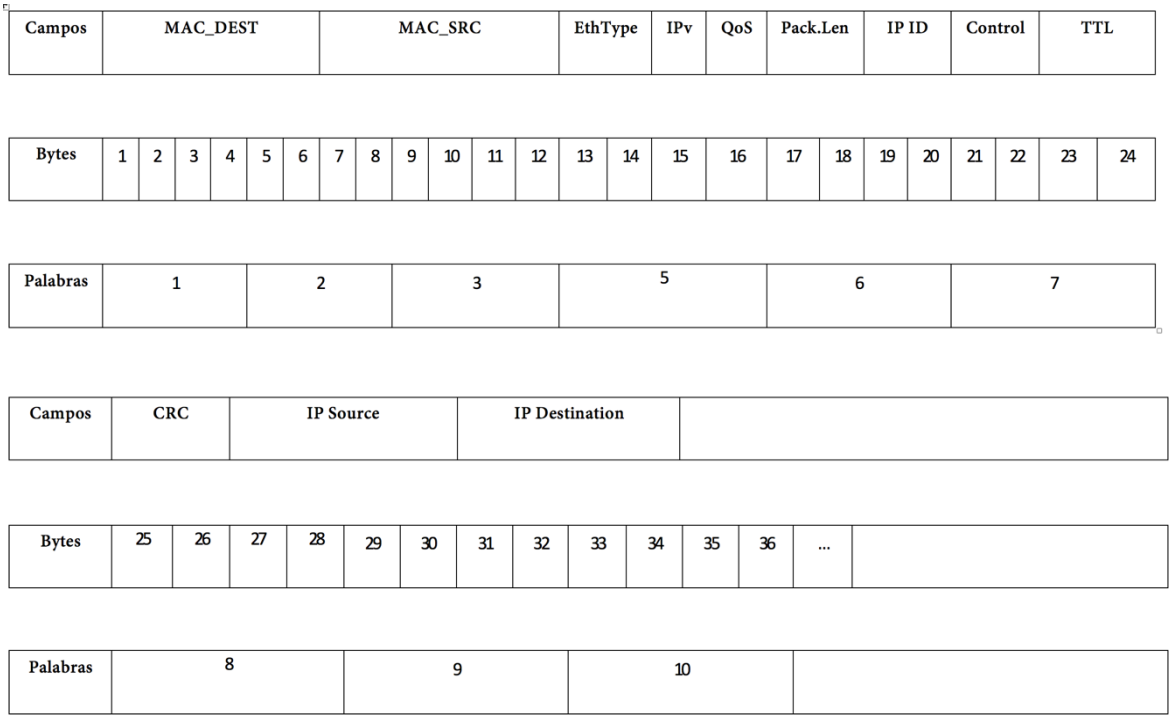


Figura 18: Estructura del paquete TCP/IP

El bloque que controla la interfaz AXI4-Lite es un módulo SystemC que se ha instanciado dentro del bloque IP bajo desarrollo. El modelado del bloque AXI4-Lite en un bloque permite que el bloque IP sea diseñado independientemente del protocolo de comunicación usado. En un futuro, si se necesita cambiar el protocolo de comunicación,

sólo sería necesario instanciar el módulo que implementa el protocolo de comunicación deseado.

5.3 Estructura de Memoria

El bloque IP desarrollado hace uso de dos memorias principales para su funcionamiento. Se trata de un bloque de memoria RAM y una FIFO. Mientras el bloque de memoria **RAM** pertenece a la estructura interna del bloque, la FIFO es un bloque IP independiente que se conecta al bloque en desarrollo en el momento de implementación de la plataforma.

En este TFM se ha optado por guardar la señal del bus **LAST** asociada a cada dato. De esta manera, el proceso que lee de la FIFO y escribe los datos en la interfaz de salida es capaz de identificar el final del paquete sin necesidad de tener una señal que indique explícitamente el tamaño de cada paquete. Esta decisión permite la reducción de lógica de control que se encargue de gestionar el número de *flits* recibidos por paquete y de contabilizar el número de paquetes almacenados en la FIFO. De esta forma, el proceso de escritura en la FIFO y el de lectura son independientes y no hay riesgo de pérdida de datos.

El estudio sobre el dimensionamiento de la memoria del sistema se ha realizado teniendo en cuenta la tasa binaria de entrada de datos y la cantidad de ciclos necesarios para llevar a cabo el análisis. La FIFO de entrada debe poder almacenar hasta dos paquetes completos de 1500 bytes. Esta medida permite que el bloque pueda admitir paquetes sin pérdidas si se interrumpe el flujo de salida del bloque. El bloque de memoria **RAM** utilizado para el almacenamiento de la copia de la cabecera del paquete entrante es de un tamaño menor. Esta memoria tiene accesos de escritura por parte del proceso que se encarga de copiar las cabeceras y accesos de lectura por parte del proceso que realiza el análisis. Es por ello por lo que se ha utilizado una memoria del tipo **Dual-Port RAM** que está disponible en la tecnología de la FPGA usada. La dimensión de esta memoria es de 10 posiciones, ya que es suficiente para el almacenamiento de los datos de interés. La sobreescritura de los datos de esa memoria a medida que llegan nuevos paquetes no afecta al proceso de análisis, ya que se registran los valores de interés durante el análisis.

El bloque **TEMAC** trabaja con datos de 32-bit. Sin embargo, al optar por almacenar la señalización TLAST del bus, el ancho de la palabra almacenada en la FIFO de entrada es de 33-bit. Sin embargo, la memoria compartida entre el proceso de copia y el proceso de análisis es de 32-bit, ya que no se requiere almacenar información adicional asociada a cada dato.

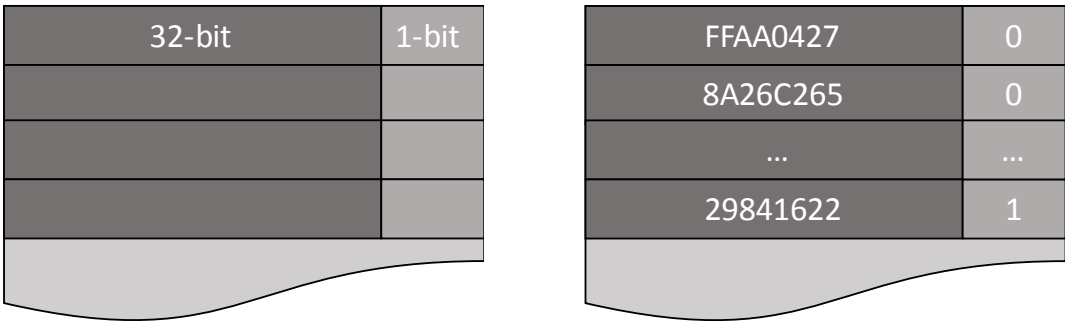


Figura 19: Disposición de datos en memoria

5.4 Registros de configuración

El proceso que controla el protocolo AXI4-Lite tiene mapeados en su estructura de memoria el conjunto de registros de configuración para el proceso de análisis. Estos registros pueden ser consultados y modificados desde la CPU y sólo consultados por el proceso de análisis. La relación de registros de configuración se lista en la Tabla 1 , en la que además se describe brevemente su funcionalidad.

La estructura de memorias que se utiliza en el bloque AXI4-lite se compone únicamente de una memoria BRAM y un sistema de decodificación de direcciones. Los valores de los registros son consultados por el bloque IP cada vez que se inicia un proceso de análisis. El proceso de análisis registra los valores de estos registros para evitar que la configuración cambie durante el proceso de análisis.

Tabla 1: Registros de configuración del bloque en desarrollo

Nombre	Descripción	Tamaño
MAC_dest	Registro en el que se almacena el dirección MAC de destino.	6 Bytes
MAC_src	Registro en el que se almacena el dirección MAC del emisor.	6 Bytes
Ethernet Type	Indica el protocolo que encapsula el paquete. Ejemplo: IP, ARP, CorbaNet …	2 Bytes
Check_Network	Determina si se realiza análisis em la cabecera de red.	Booleano
ToS	Indica un valor específico del campo *Type of Service*. Ejemplo: *Best effort, Crítical, Network Contro*l …	1 Byte
Protocol	Indica el tipo de protocolo que encapsula la cabecera de red. Ejemplo: TCP, UDP, ICMP …	1 Byte
IP_src	Determina un valor para la dirección IP de la fuente	4 Bytes
IP_dest	Determina un valor para la dirección IP del destino	4Bytes

5.5 Co-Simulación SystemC y RTL

El proceso de verificación funcional del bloque se ha llevado a cabo tanto en la fase de modelado como tras la síntesis de alto nivel y una vez implementado el bloque.

La verificación consiste en el envío de varios paquetes de red conformados desde el nivel de enlace. La obtención de los paquetes para el test se ha realizado con la herramienta *Wireshark* [24]. Esta herramienta permite capturar los paquetes entrantes y salientes de una interfaz de red. Para la realización del test correspondiente, se han realizado capturas de tráfico TCP, tráfico UDP y tráfico ARP entre otros. La variedad en el tipo de tráfico permite verificar un correcto funcionamiento del bloque cuando se reciben paquetes de longitudes grandes y pequeñas.

La elaboración del *testbench* en SystemC permite realizar la verificación en la fase de modelado de manera directa, instanciando el bloque bajo verificación, el bloque que lleva a cabo la introducción de estímulos y realizando un correcto conexionado entre ambos. Durante la fase de verificación es necesario también instanciar un bloque SystemC que modele el funcionamiento de una FIFO externa para el funcionamiento del bloque.

Los paquetes se envían al bloque bajo verificación a la vez que se modifican los registros de configuración. Este mecanismo permite verificar una correcta recepción y

transmisión del paquete a la vez que se verifica el correcto funcionamiento del bloque AXI4-Lite y la ejecución correcta del proceso de análisis.

Durante la fase de síntesis de alto nivel, la herramienta CtoS genera un *wrapper* que permite reutilizar el *testbench* de SystemC para verificar el bloque descrito en RTL. Esta característica de CtoS ahorra tiempo en el proceso de verificación y reduce la posibilidad de error producido por las diferencias entre el diseño del *testbench* en SystemC y RTL.

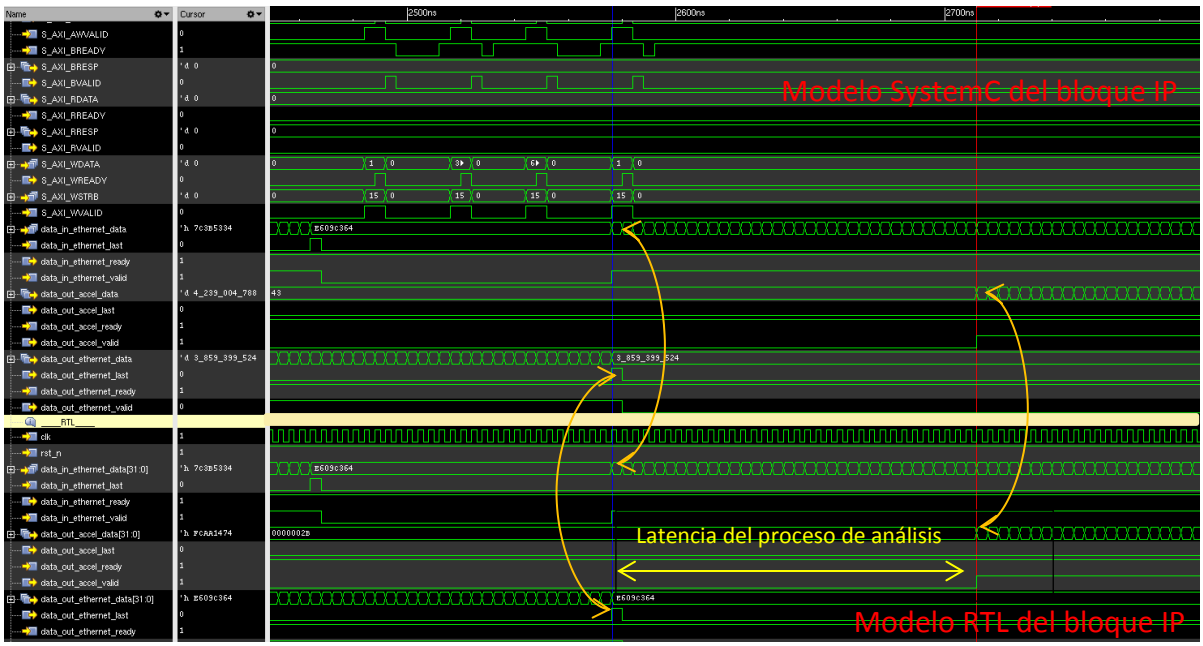


Figura 20: Detalle de cosimulación en Incisive

En la Figura 20 se puede observar con las flechas como el modelo RTL se comporta de manera idéntica al modelo en SystemC. Este diagrama permite comprobar que la síntesis de alto nivel se ha llevado a cabo de manera correcta y además permite ver el funcionamiento y la latencia del bloque para los casos de test diseñados en la verificación del modelo.

5.6 Resultados de la síntesis del bloque

Tras el proceso de síntesis de alto nivel y síntesis lógica, se ha obtenido un bloque funcional capaz de realizar el filtrado de paquetes TCP/IP teniendo en cuenta el valor de los campos de las cabeceras de enlace y de red. Durante el proceso de diseño se han

tomado varias decisiones en el algoritmo de funcionamiento y se han aplicado distintas estrategias de optimización durante la síntesis para conseguir los objetivos planteados que cumplan con las especificaciones del bloque. En este apartado se analizarán los resultados del bloque en cuanto a ocupación y prestaciones.

Se pretende que la utilización de recursos del bloque sea reducida. Esta característica es importante ya que supone que pueda ser utilizado varias veces en la plataforma sin provocar un alto impacto en el consumo de recursos totales de la FPGA. También hay que tener en cuenta que en la plataforma existen bloques que realizan tareas de búsquedas exhaustivas en el *payload* que requieren igualmente de un alto uso de recursos. Este factor motiva la optimización en consumo de recursos del bloque desarrollado. La Figura 21 recoge el resultado de la síntesis lógica del bloque utilizando la herramienta de síntesis lógica y optimización Synopsys Synplify.

El bloque diseñado es capaz de trabajar a una frecuencia de **460 MHz**. Ello permite obtener valores de la tasa de transferencia de datos (*throughput)* máxima, latencias mínimas y porcentaje de aceleración con respecto a la arquitectura. La Figura 22 muestra el aspecto del bloque una vez empaquetado para su uso en el **Vivado IP Integrator**.

Project Settings			
Project Name	header_analyzer	Implementation Name	rev_3
Top Module	header_analyzer_rtl	Pipelining	1
Retiming	0	Resource Sharing	1
Fanout Guide	10000	Disable I/O Insertion	0
Disable Sequential Optimizations	0	Clock Conversion	1

Run Status								
Job Name	Status				CPU Time	Real Time	Memory	Date/Time
Compile Input (compiler) Detailed report	Complete	79	26	0	-	00m:06s	-	3 May 2016 17:23:12
Premap (premap) Detailed report	Complete	7	2	0	0m:03s	0m:04s	161MB	3 May 2016 17:23:23
Map & Optimize (fpga_mapper) Detailed report	Complete	33	2	0	0m:34s	0m:44s	249MB	3 May 2016 17:24:08

Area Summary			
I/O ports (io_port)	323	Non I/O Register bits (non_io_reg)	950 (0%)
I/O Register bits (total_io_reg)	33	Block Rams (v_ram)	0 (545)
DSP48s (dsp_used)	0 (900)	LUTs (total_luts)	615 (0%)
Detailed report		Hierarchical Area report	

Timing Summary			
Clock Name (clock_name)	Req Freq (req_freq)	Est Freq (est_freq)	Slack (slack)
header_analyzer_rtl|clk	542.1 MHz	460.8 MHz	-0.326
Detailed report		Timing Report View	

Optimizations Summary	
Combined Clock Conversion	1 / 0 more

Figura 21: Resumen de síntesis lógica en Synplify

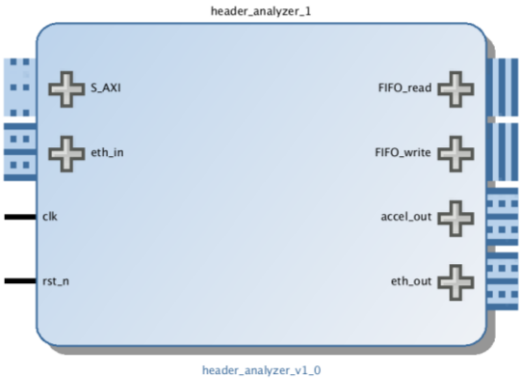


Figura 22: Detalle del Bloque IP

Name	Slice LUTs	Slice Registers	Slice	LUT as Logic	LUT as Memory	LUT Flip Flop Pairs	Block RAM Tile
header_analyzer	576	947	298	532	44	876	0

Tras la síntesis, los resultados obtenidos por el Synplify determinan que el bloque diseñado alcanza una frecuencia máxima de 460Mhz. Tal como está diseñado, el bloque acepta un flujo de entrada con un ancho de 32-bit. De esta manera, se consigue un bloque capaz de trabajar con una **tasa de transferencia de datos de 14 Gbps.**

5.7 Conclusión

En este capítulo se ha explicado la metodología seguida para el desarrollo del bloque IP. Se ha tomado la decisión de dividir el funcionamiento del bloque en distintas tareas para facilitar el modelado del bloque siguiendo una metodología de diseño incremental.

Subdividir el bloque en varios procesos ha permitido abordar el desarrollo de manera que se ha podido optimizar el funcionamiento de cada una de las tareas de forma individual. Las tareas realizadas incluyen el diseño de un proceso encargado de la adquisición y almacenamiento de los datos de entrada desde la red de manera independiente. Posteriormente se diseñaron los procesos de análisis y de escritura de datos en la salida del bloque. El sistema AXI4-Lite se ha implementado como un bloque totalmente independiente con sus procesos de lectura, escritura y control. Finalmente, el bloque AXI4-lite se ha instanciado dentro del bloque IP bajo desarrollo.

El bloque está preparado para trabajar con distintos anchos de palabra. El uso de 32-bit como ancho del bus está limitado por el controlador MAC utilizado. De esta manera, el bloque sería capaz de trabajar con una tasa de transferencia de datos mayor en caso de que se incremente el ancho del bus de comunicación.

Capítulo 6. Diseño de la plataforma e Integración Hardware-Software

6.1 Introducción

En este capítulo se explica la plataforma para la verificación del funcionamiento del bloque y la plataforma de validación del sistema. A continuación, se aborda el desarrollo del código empotrado en el ARM.

La plataforma de validación difiere de la plataforma de verificación en que los paquetes que se transmiten al bloque no proceden de la memoria del sistema, sino que vienen directos de las interfaces de red. De esta manera, el *software* no participa de la captura, análisis y retransmisión del paquete.

6.2 Validación del sistema

Se ha realizado la validación del sistema sobre el prototipo realizado sobre la placa ZC706. Los datos de entrada son proporcionados desde la CPU empotrada. Para ello se ha creado una plataforma que incluye un bloque PS, la infraestructura de comunicación y se ha integrado el bloque IP diseñado. Además, se ha creado el *software* empotrado necesario que se ejecuta sobre el procesador ARM Cortex-A9 funcionando en una configuración Bare-Metal. Los datos sintéticos de entrada se almacenan en memoria del procesador y son enviados por éste hacia el bloque IP a través de la infraestructura de comunicación.

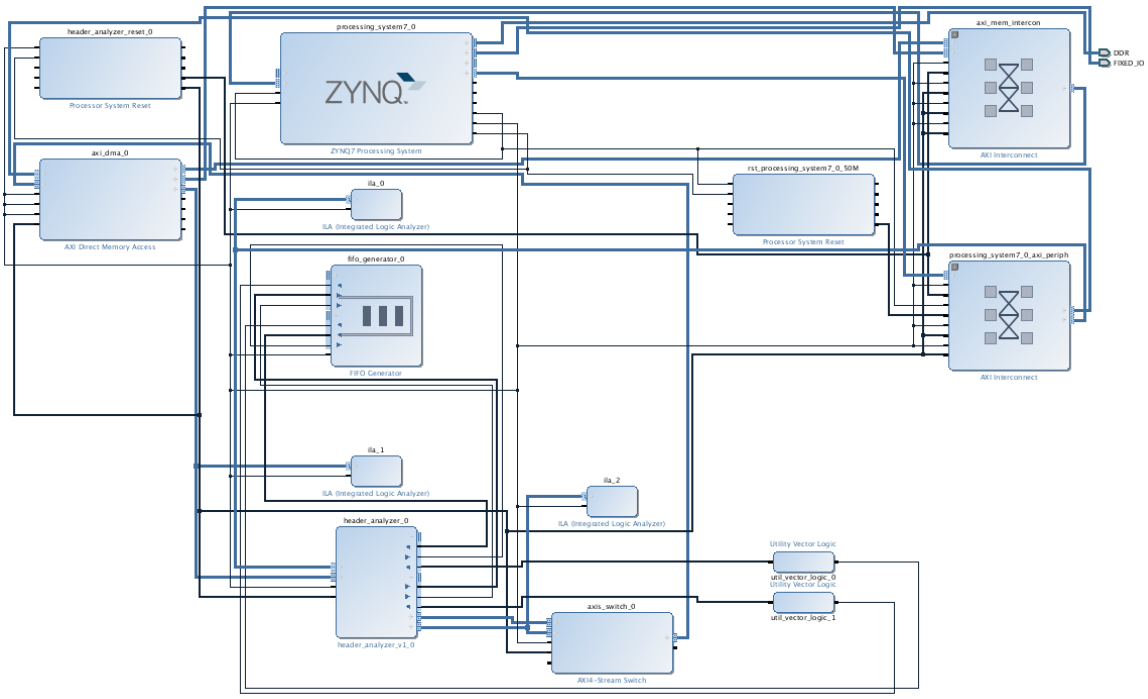


Figura 23: Plataforma de verificación

La elaboración de la validación para el bloque una vez integrado en la plataforma requiere de elementos adicionales y el desarrollo del código empotrado que difiere del usado en la verificación de los modelos SystemC y RTL, usando técnicas de simulación. En la plataforma de verificación se requiere añadir un bloque DMA que permita la transferencia de los paquetes del test al bloque. Además, el conjunto de paquetes del test debe estar cargado en el sistema para poder hacer uso de ellos desde el *software*. Estas características requieren un desarrollo *software* para realizar rutinas de inicialización de los bloques dedicados a la comunicación con el bloque bajo verificación y procesos de carga de los paquetes en la memoria del sistema.

6.3 Estructura de la plataforma funcional

Tal como se presenta en la Figura 7, el bloque trabaja con flujos unidireccionales que permiten la comunicación entre dos interfaces de red. El montaje de la plataforma se hace de tal manera que se instancia un bloque IP desarrollado por cada canal de comunicación del controlador TEMAC. De esta forma, en el modo de funcionamiento en el que el bloque no realiza análisis del *payload*, y por tanto permite la comunicación transparente entre interfaces, cada paquete que llega por una interfaz es retransmitido hacia la otra interfaz. Esta configuración permite la comunicación entre las dos interfaces

de red haciendo que el sistema no afecte en la configuración de la red. En la Figura 24 se puede apreciar la integración de los distintos bloques para la realización de la plataforma.

Para poder llevar a cabo esta plataforma es necesario incluir en el diseño bloques de controladores TEMAC en la FPGA. Estos controladores TEMAC requieren de una conexión con un controlador de capa física **PHY** que no está incluida en la placa de desarrollo. Estas conexiones se han mapeado en el puerto FMC de la placa de manera que se pueda conectar una placa externa que de soporte a la capa física de la conexión. Las características de esta placa externa se detallarán a continuación.

6.4 Conexión de las interfaces físicas Ethernet FMC

Para implementar la interfaz física de red, en este proyecto se ha utilizado la placa EthernetFMC de Ospero [25] con objeto de dar soporte físico a las interfaces de red implementadas en la FPGA. Esta placa se ha instalado en el conector de expansión Mezzanine Card de la Zynq ZC706 de baja densidad de pines (LPC). Esta placa contiene 4 controladores físicos Ethernet Marvell 88E1510 que permiten tráfico Gigabit. De los cuatro controladores disponibles sólo son necesarios dos para la elaboración de este sistema. La Figura 25 muestra la placa EthernetFMC.

Como se ha indicado anteriormente, la placa implementa únicamente la capa PHY del sistema de comunicación. Es por ello por lo que es necesario integrar en la plataforma un controlador TEMAC que se encarge de controlar las señales del protocolo Ethernet.

La conexión entre el TEMAC y los controladores PHY de cada una de las interfaces de red requiere del estudio del *pinout* de la placa EthernetFMC y de la FPGA. La manera de realizar la relación entre un pin del conector FMC y la etiqueta de la señal del TEMAC es a través de un fichero de restricciones (*constraints)* en la fase de implementación de la plataforma. Este fichero de restricciones define los niveles de tensión necesarios para la comunicación, la asignación entre las etiquetas usadas en el diseño de la plataforma y los pines físicos del conector.

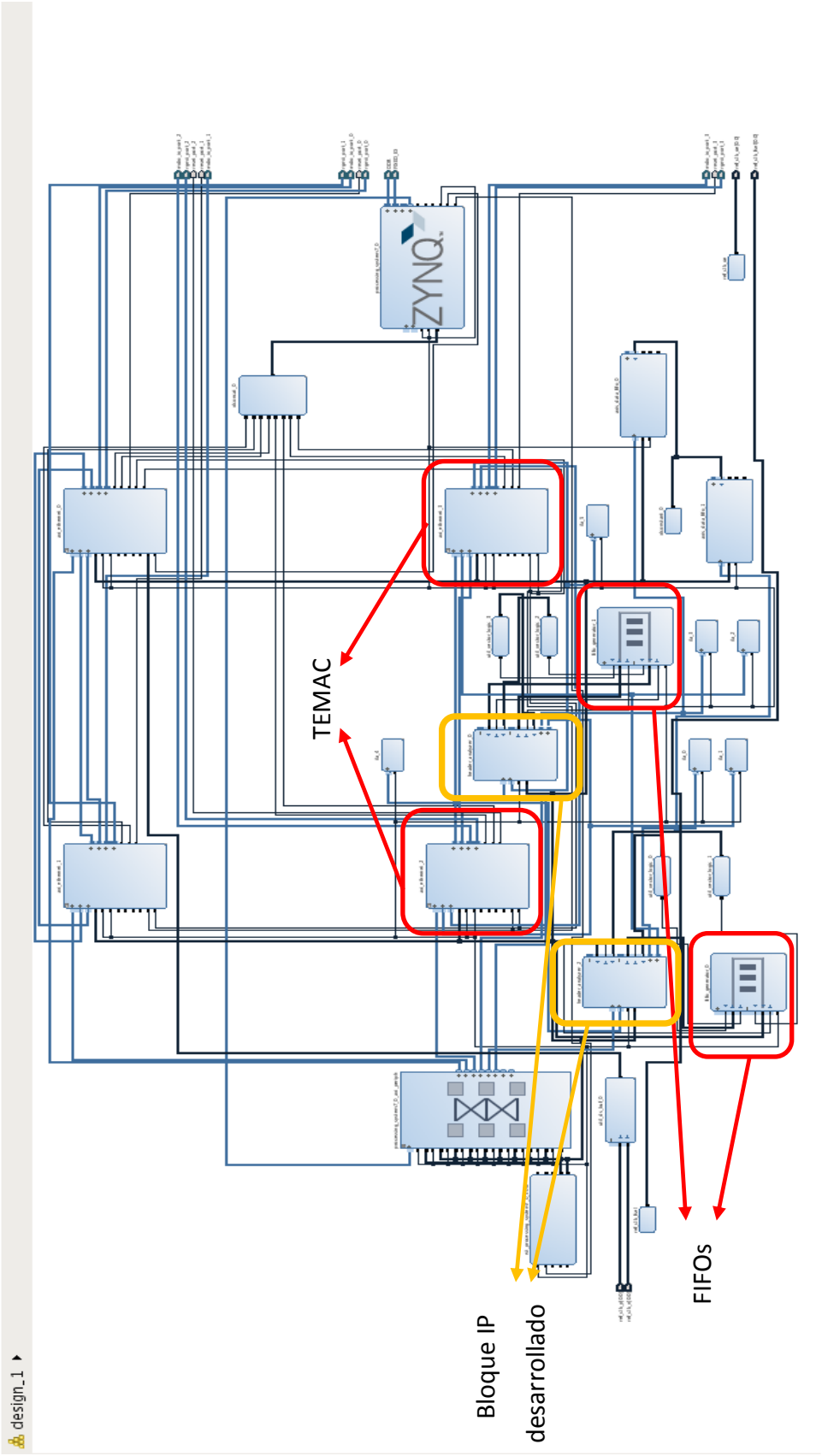


Figura 24: Detalle de la plataforma funcional en Vivado IP Integrator

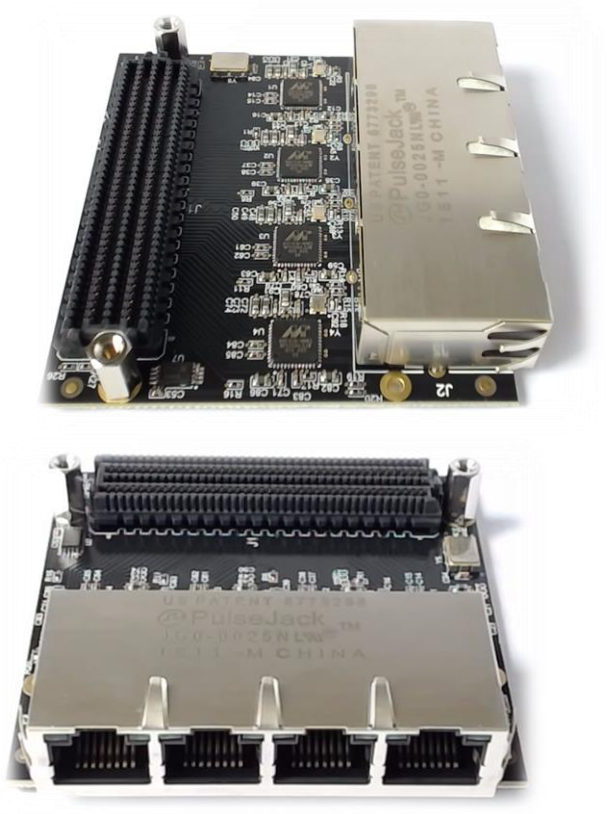


Figura 25: Detalle de la placa EthernetFMC

6.5 Desarrollo software en el SDK

La plataforma requiere del desarrollo de la aplicación empotrada del sistema en el entorno SDK de Xilinx. En este entorno se codifican las rutinas de inicialización de los bloques y se desarrollan las librerías necesarias para la configuración de los registros del bloque a través de AXI4-Lite.

Al igual que para el sistema de validación, el sistema final no requiere de un sistema operativo que gestione los núcleos de procesamiento y por tanto se adopta un modelo de programación con configuración Bare-Metal. La razón por la cual no es necesario el uso de un sistema operativo es que no se requiere de un software complejo que controle semáforos, varias tareas en ejecución y la implementación de un *stack* TCP/IP completo. El *software* que ejecuta el sistema es prácticamente de inicialización y cambio de la configuración del bloque IP desarrollado. La estructura del *software* se divide en dos partes: la inicialización del sistema y la fase de cambio de configuración del bloque IP. La fase de inicialización se encarga de realizar las tareas de configuración de los

bloques TEMAC para su uso. La configuración consiste en la asignación de direcciones MAC a dichos controladores, de manera que puedan ser reconocidas e integradas en las redes de comunicación de manera automática. La estrategia adoptada para la realización de una conexión transparente es la asignación de la misma dirección MAC a los controladores de entrada y salida de datos.

Las instancias del bloque desarrollado también requieren inicialización. Esta inicialización consiste en la asociación de un *struct* y un bloque IP físico. Esta asociación permite al programa dirigirse a cada uno de los bloques a través del nombre del *struct* de una manera directa. Finalmente, en la fase de configuración del bloque IP es necesario realizar las escrituras pertinentes en los registros de configuración de los bloques de análisis. El ajuste de la configuración se hará a través de una librería que se ha realizado en el desarrollo de este TFM. Esta librería se compone de una función ***set*** y ***get*** para cada uno de los registros de configuración descritos en la Tabla 1. Estas funciones hacen uso de librerías de acceso a direcciones de memoria que provee Xilinx para sus dispositivos Zynq (Xil_io). Además, es preciso el mapa de memoria que se ha usado en el proceso AXI4-Lite para ubicar los registros de configuración en la memoria. Este mapa de memoria se ha diseñado en el momento de desarrollo de dicho proceso y está recogido en un fichero llamado `reg_addr.h` en el directorio del proyecto.

Como se puede apreciar en la Figura 26, los espacios entre registros no coinciden necesariamente con el tamaño del valor almacenado en el registro. Esto se debe a dos razones principales. La primera razón es que se han utilizado direcciones de memoria alineadas al ancho del bus de comunicación (32-bit), de manera que el dato llegue alineado y se pueda almacenar directamente sin tener en cuenta la señal de *strobe*. Esta característica permite simplificar la implementación del protocolo en la fase de modelado en SystemC. La segunda razón es que estas direcciones no se utilizan directamente como las direcciones de memoria, sino que actúan a modo de decodificador de direcciones para acceder a las direcciones de memoria en las que realmente están almacenados los datos.

```
#ifndef AXI4_REG_ADDR_H
#define AXI4_REG_ADDR_H

/*
 *  0x00000000
 *
 *  0x00000001 check_ether      Byte
 *
 *  0x00000010 dest_mac         6 Bytes
 *
 *  0x00000040 org_mac          6 Bytes
 *
 *  0x00000070 ether_type       2 Bytes
 *
 *  0x00010001 check_network    Byte
 *
 *  0x0001010 QoS               Byte
 *
 *  0x0001018 protocol          Byte
 *
 *  0x0001040 ip_src            4Byte
 *
 *  0x0001060 ip_dest           4Byte
 */


#define CHECK_ETHER_REG_ADDR        0x00000000
#define DEST_MAC_REG_ADDR_LSB       0x00000010
#define DEST_MAC_REG_ADDR_MSB       0x00000020
#define ORG_MAC_REG_ADDR_LSB        0x00000040
#define ORG_MAC_REG_ADDR_MSB        0x00000050
#define ETHER_TYPE_REG_ADDR         0x00000070
#define CHECK_NETWORK_REG_ADDR      0x00001000
#define QOS_REG_ADDR            0x00001010
#define PROTOCOL_REG_ADDR       0x00001018
#define IP_SRC_REG_ADDR         0x00001040
#define IP_DEST_REG_ADDR        0x00001060


#endif /* AXI4_REG_ADDR_H */
```

Figura 26: Mapa de memoria del AXI4-Lite

6.6 Conclusiones

Este capítulo ha descrito el montaje de una plataforma de validación y la plataforma funcional. La plataforma de validación sirve como teste de la última fase del flujo de diseño que es la implementación del bloque en la FPGA. La plataforma funcional permite probar el prototipo en un caso de conexión real con tráfico entre dos equipos.

La plataforma de validación tiene almacenados los paquetes del test en la memoria del sistema. A través de un DMA se envían los paquetes al bloque IP. A través de este mecanismo se puede comprobar el correcto funcionamiento del bloque calculando previamente el tipo de resultado esperado en cada configuración del bloque IP.

La validación del bloque una vez se ha implementado en la FPGA requiere que la introducción de los estímulos se lleva a cabo desde dentro del mismo SoC. Es por ello por lo que se ha necesitado codificar un pequeño programa en el ARM para verificar el funcionamiento del bloque. Este programa transmite el conjunto de paquetes utilizados en el test del modelo SystemC y configura el bloque con diferentes valores en la configuración del análisis con la misma estructura que el *testbench* para el modelo en SystemC.

La plataforma funcional permite la captura de paquetes de la red a través de interfaces de red externas. En el caso de este TFM se ha utilizado el conector de expansión FMC para la integración de estas interfaces adicionales. El uso de esta placa adicional ha requerido el conocimiento y estudio de las directivas de implementación para la tarea de asociar señales al conjunto de pines del conector FMC.

El *software* de la plataforma se encarga de inicializar los bloques TEMAC para la recepción de paquetes de red y la configuración del bloque IP en función del análisis que se quiera llevar a cabo. El *software* se encarga de esperar cambios en las configuraciones del bloque IP por parte del usuario. De esta manera, se ha desarrollado un código en el que el *software* trabaja de manera independiente durante el proceso de análisis de cabeceras.

Capítulo 7. Proceso de validación

7.1 Introducción

La plataforma funcional permite el uso del bloque en un entorno real en el que el bloque desempeña su tarea de manera transparente a las redes que comunica el sistema. Para validar el sistema se ha creado un prototipo de manera que el sistema está ubicado en medio de una comunicación directa entre dos equipos. El sistema recibe todo el flujo de la comunicación entre los equipos en ambos sentidos y realiza las tareas de análisis para las que se ha configurado.

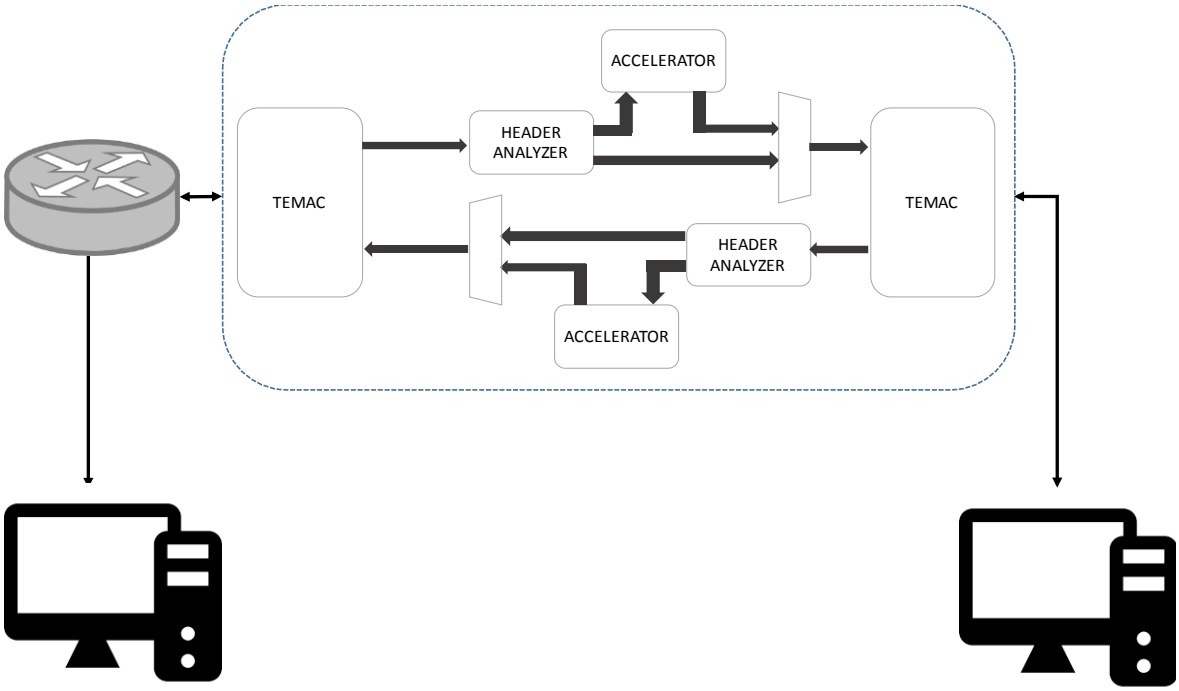


Figura 27: Diagrama del montaje de validación

El prototipo de la Figura 27 permite establecer cualquier tipo de comunicación entre dos equipos ubicados en distintas redes sin que el sistema afecte a dicha comunicación.

Para llevar a cabo esta validación se ha ejecutado un test en el que un equipo genera tráfico y el otro equipo lo recibe. El generador de tráfico es alimentado con un fichero en formato PCAP [26] que contiene paquetes de distintos tamaños y distintos protocolos. De esta manera se puede comprobar que el sistema tolera un flujo continuo y heterogéneo de paquetes de datos entrantes en el sistema.

7.2 Obtención de resultados

Una vez se ha validado el sistema, se puede realizar la medida de la latencia del mismo en un caso de uso real. En este capítulo se describe el proceso de medición de los parámetros de latencia y de consumo.

Durante la medición de la latencia ha sido fundamental el uso de un bloque IP que actúa como analizador lógico (*Integrated Logic Analyzer*, ILA). Este bloque se ha incluido en la plataforma para monitorizar el valor de las señales del bloque desarrollado. La visualización de las señales capturadas y la configuración del *trigger* del mismo se llevan a cabo en el Vivado Design Suite en el entorno de **Hardware Manager**.

En Figura 28 se presenta en detalle el entorno de Vivado que permite controlar los ILA integrados en el diseño. La flecha roja indica el eje de tiempo en ciclos de reloj. La llave azul indica la relación de señales controladas por el ILA. La actividad de las señales se representa en la zona bordeada con la línea naranja.

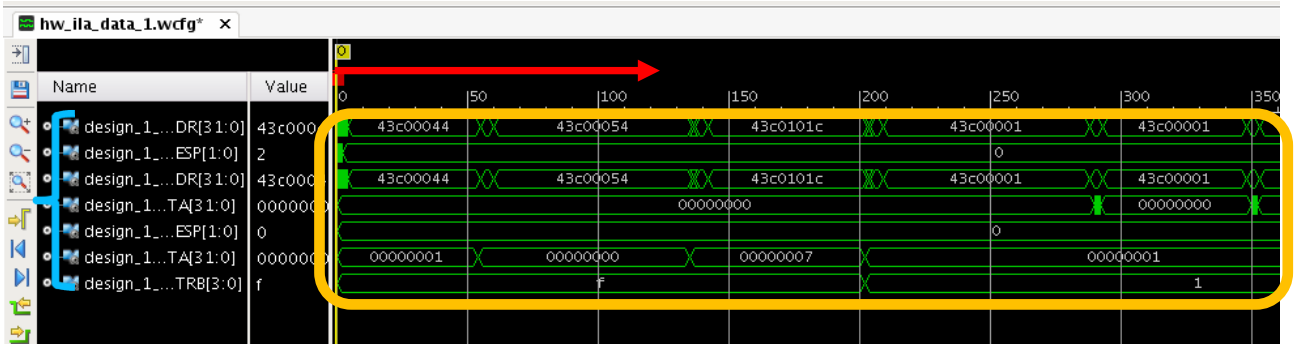


Figura 28: Detalle del ILA en el Hardware Manager

Con el ILA se ha podido ver las señales del sistema en funcionamiento en el prototipo con tráfico real. El ILA permite detectar alguna anomalía y controlar los paquetes que se están enviando al acelerador *hardware*.

El resultado de la validación confirma el funcionamiento del sistema. A raíz del resultado se puede determinar que el sistema es válido para implementación en varias topologías de red. A continuación, se representa en la Figura 29un ejemplo de topología válida y que puede resultar de interés.

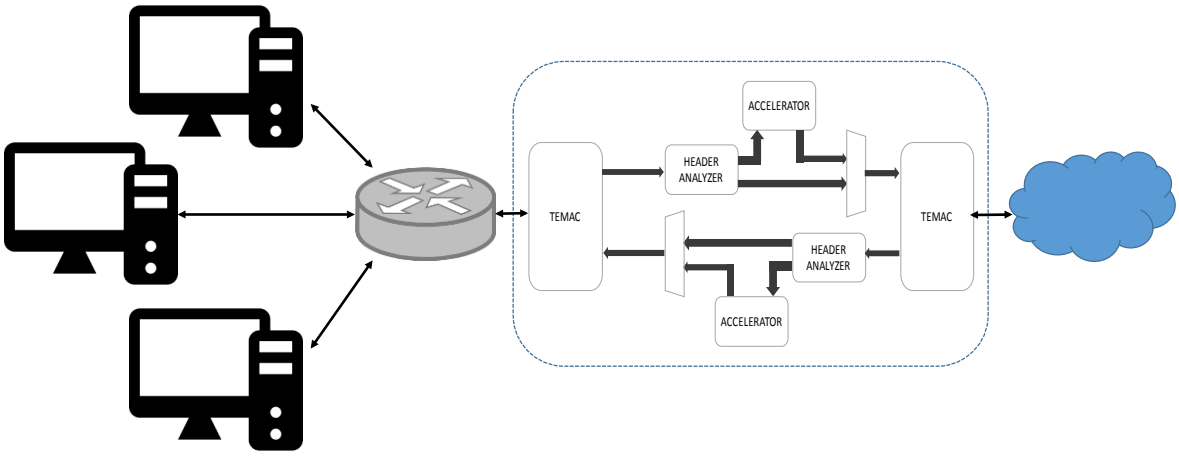


Figura 29: Topologías de red propuestas

La latencia del bloque depende del modo de análisis especificado en la configuración. Para configuraciones en las que el bloque no realiza análisis con todos los paquetes entrantes, la latencia es de 10 ciclos de reloj desde la entrada del primer dato de un paquete hasta la salida del mismo. Cuando la configuración implica análisis de todos y cada uno de los registros disponibles en la configuración, la latencia del bloque es de poco menos de 40 ciclos.

Como conclusión, se puede determinar que en cualquiera de los casos el bloque ofrece una latencia aceptable para trabajar con flujos de datos del orden de Gigabits. Sin embargo, la limitación de la tasa de transferencia de datos de las interfaces de red no ha permitido hacer un análisis exhaustivo en una red Gigabit con tráfico significativo.

El bloque diseñado es apto para uso en sistemas DPI más complejos que tengan como objetivo el control y análisis del tráfico entrante y saliente de una subred. Esta

topología es usada en industrias, oficinas y hogares. De esta manera, el sistema es bastante flexible y admite varias configuraciones en la topología de red.

7.3 Conclusiones

En este capítulo se ha explicado la metodología de validación del sistema en un entorno real con tráfico de datos entre dos equipos. La conexión entre los equipos se ha realizado a través de un *router*, de manera que cada equipo pertenezca una subred distinta. Estas características de la topología de red diseñada para la validación se asemejan a un caso de uso doméstico o empresarial de un sistema de seguridad de red.

Para comprobar el funcionamiento del bloque IP y el re-direccionamiento de los paquetes que cumplen el criterio de análisis, se ha utilizado un conjunto de núcleos de depuración implementados en la FPGA llamados ILAs (Integrated Logic Analyzer). Estos núcleos de depuración permiten capturar y visualizar los valores de las distintas señales durante el uso del sistema.

Este sistema de validación ha servido de soporte para la medición de la latencia del bloque tanto durante el proceso de análisis como durante el proceso de re-direccionamiento sin aplicar análisis.

Finalmente, se ha propuesto otra topología de red en la que el sistema diseñado puede ser igualmente válido.

Capítulo 8. Análisis de resultados y comparación con otras soluciones

8.1 Introducción

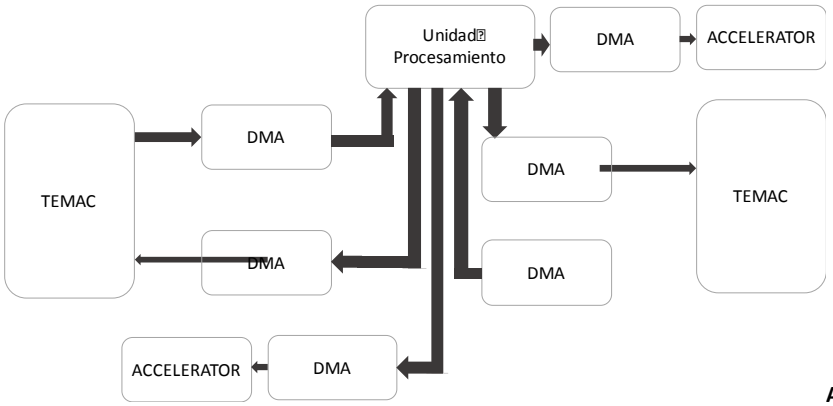
Como se ha indicado, el objetivo de este trabajo es realizar el desarrollo de un bloque IP que reduzca la latencia desde que un paquete llega a un sistema DPI hasta que se toma la decisión de analizarlo o permitir que continúe su trayecto a través de la red. En este capítulo se analizan los resultados obtenidos tanto de latencia como de consumo.

8.2 Comparativa con Arquitectura CDBA

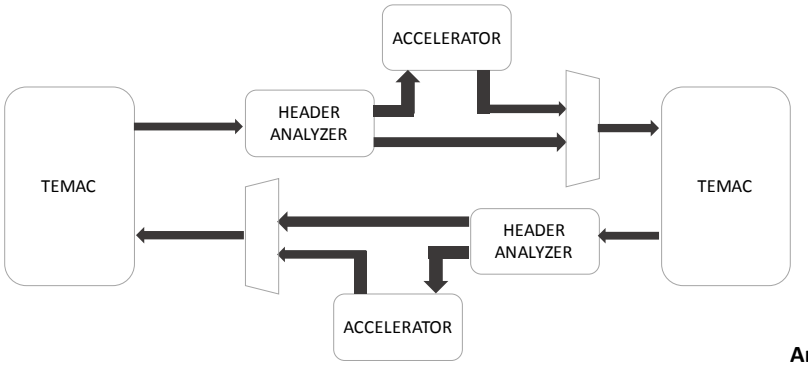
Para la comparativa se han obtenido datos de una plataforma que sigue la arquitectura CDBA descrita en el capítulo correspondiente al análisis del problema. Se ha trabajado con una frecuencia de reloj en la CPU de 800Mhz y frecuencia de reloj en la FPGA de 200MHz. Se ha realizado una medición en ciclos de reloj desde que un paquete es transmitido por el bloque TEMAC hasta que éste llega al acelerador *hardware*. La medida de estos ciclos de reloj se ha realizado mediante un bloque ILA implementado en *hardware*.

8.2.1 Comparativa de latencia

Para contabilizar el tiempo tomado por la CPU para la toma de decisión se ha utilizado un contador *hardware* integrado en el bloque de procesamiento de la Zynq. Debido a que las frecuencias de los bloques *hardware* y los procesos de la CPU no son la misma, la medida se representará en microsegundos.



Arquitectura CDBA



Arquitectura HIBA

Figura 30: Diagrama de las soluciones comparadas

Tabla 2: Comparación con Arquitectura Clásica

Parámetro	Definición	Arquitectura CDBA	Arquitectura HIBA	Mejora CDBA/HIBA
t_{ad}	Tiempo desde que llega el paquete hasta que se comienza el proceso de toma de decisión.	13,6 µs	0,04 µs	340
t_{td}	Tiempo requerido para la toma de decisión.	4µs	0,2µs	20
t_{dt}	Tiempo tomado desde la toma de decisión hasta la transmisión del paquete en el acelerador *hardware.*	9µs	0,01µs	900
Ttot	Tiempo total de todo el proceso.	26,6µs	0,25µs	**106**

En base a los resultados de la Tabla 2, si se analiza el tiempo que toma cada una de las soluciones se puede observar que el bloque diseñado presenta una aceleración de **x106**. Sin embargo, la duración de la transmisión del paquete al acelerador es bastante similar ya que en ambas soluciones se transmite una palabra de 32-bit cada ciclo de reloj.

La solución que implementa la arquitectura CDBA tiene el límite del aumento de frecuencia del reloj en los datos de entrada en la frecuencia de funcionamiento del DMA, ya que éste solo puede trabajar a un máximo de 150 MHz. En la solución que sigue la arquitectura HIBA el límite está en el bloque diseñado, cuya frecuencia máxima de funcionamiento es de 460 MHz. Las limitaciones de frecuencia de funcionamiento corresponden a la implementación del sistema en el dispositivo Zynq 7045.

8.2.2 Comparativa en uso de recursos

La arquitectura para sistemas DPI que hace uso del bloque desarrollado permite llevar a cabo las tareas que en la arquitectura CDBA se realizan con varios bloques y procesos. Es por ello por lo que cabe hacer un análisis no solo en cuanto a la respuesta temporal sino a los recursos ocupados en la FPGA.

Los recursos comparados serán los utilizados para la realización de las tareas de adquisición del paquete de red, transmisión del paquete al proceso de toma de decisión y la transmisión del paquete al siguiente bloque correspondiente. En el caso de la solución que adopta la arquitectura CDBA, se tendrán en cuenta los bloques DMA de transmisión del paquete hacia la CPU y desde la CPU hacia la FPGA. En el caso de la plataforma que adopta la arquitectura con el bloque IP desarrollado, se analizará la ocupación del bloque IP desarrollado y las FIFOs necesarias para su funcionamiento.

Como se puede observar en la Tabla 3, la solución que implementa el bloque IP desarrollado ofrece una reducción en un orden de magnitud de los recursos utilizados. La reducción en uso de recursos conlleva por ende una reducción en el consumo del sistema.

Tabla 3: Comparación uso de recursos

	Slice LUTS	Slice Registers	LUTs as Memory	LUTs as Logic	LUTS as FF	BRAMS
Arquitectura CDBA						
- DMA TX	3033	5254	371	2662	4777	4
- DMARX	3033	5254	371	2662	4777	4
Total	6066	10508	742	5324	9554	8
Arquitectura HIBA						
- Bloque IP	578	980	44	534	878	0
- FIFO	43	49	0	43	57	0,5
Total	621	1029	44	577	935	0,5

8.2.3 Comparativa de consumo de potencia

Uno de los objetivos principales del bloque es reducir el consumo energético en el sistema. El consumo del sistema está relacionado con la cantidad de recursos que requiere cada una de las soluciones comparadas en este TFM. En este apartado se hará una estimación del consumo estático de la parte del sistema en la que entra en juego el bloque desarrollado.

En este análisis se ha llevado a cabo teniendo en cuenta el consumo de los bloques implementados en la FPGA. En el cálculo del consumo total la potencia consumida por el procesador se ha omitido debido a la complejidad de su cálculo en base a la actividad del procesador. No se puede, por el contrario, asumir que el consumo de la CPU en la solución basada en HIBA es inexistente ya que es necesario el uso de la CPU para la fase de configuración del bloque. No obstante, es razonable intuir que el consumo de la CPU será menos en la solución HIBA ya que ésta realiza una actividad puntual frente a la actividad intensiva que se hace en la solución CDBA.

Como dato de interés, según Wai Wang y Tnima Dey, el consumo de un ARM Cortex A9 es del orden de 0.5W en modo ahorro y de 1.9W en modo altas prestaciones

[27]. A pesar de que estos datos son para una tecnología de 40nm, resultan de ayuda para tener una visión del orden de magnitud del consumo en la CPU.

Para hacer la estimación del consumo en la FPGA de cada una de las soluciones se utilizará el estimador de potencia generado por Xilinx para sus productos. En este estimador se introducen los datos de recursos utilizados en cada solución. Como se puede ver en la Tabla 4, la solución HIBA propuesta en este TFM realiza las tareas de la solución CDBA con un consumo menor.

Tabla 4: Comparación de potencia consumida

	CDBA	HIBA	CDBA/HIBA
Potencia [W]	**0.056**	**0.008**	**7**

8.3 Comparativa con soluciones existentes

8.3.1 Sistema de comparación con el núcleo NIFIC

El trabajo realizado por un equipo de investigación de distintas universidades de República Checa describe un sistema *hardware* basado en FPGA que clasifica paquetes de red para un posterior procesado en *software*. Se trata de una arquitectura que soporta un tráfico de entrada y salida de 10Gbps. La clasificación de los paquetes se lleva a cabo a partir de un análisis a nivel de cabeceras. El resultado del análisis determina si el paquete será procesado por el *software* o no. El filtrado y la clasificación realizados en *hardware* es capaz de trabajar a velocidades de 10Gbps. Sin embargo, la tasa de transferencia de datos de la notificación al *software* de cada uno de los eventos detectados está limitada a 5Gbps [28].

Para poder llevar a cabo una comparativa entre dicho bloque IP y el bloque desarrollado en este TFM, se tendrá en cuenta únicamente la parte *hardware* del sistema. Además, se sintetizará el bloque IP desarrollado el en TFM para una FPGA de la familia Virtex-5 de Xilinx, ya que es la utilizada en el sistema de referencia. Una vez obtenidos los datos de síntesis, se comparan los resultados de ambos sistemas.

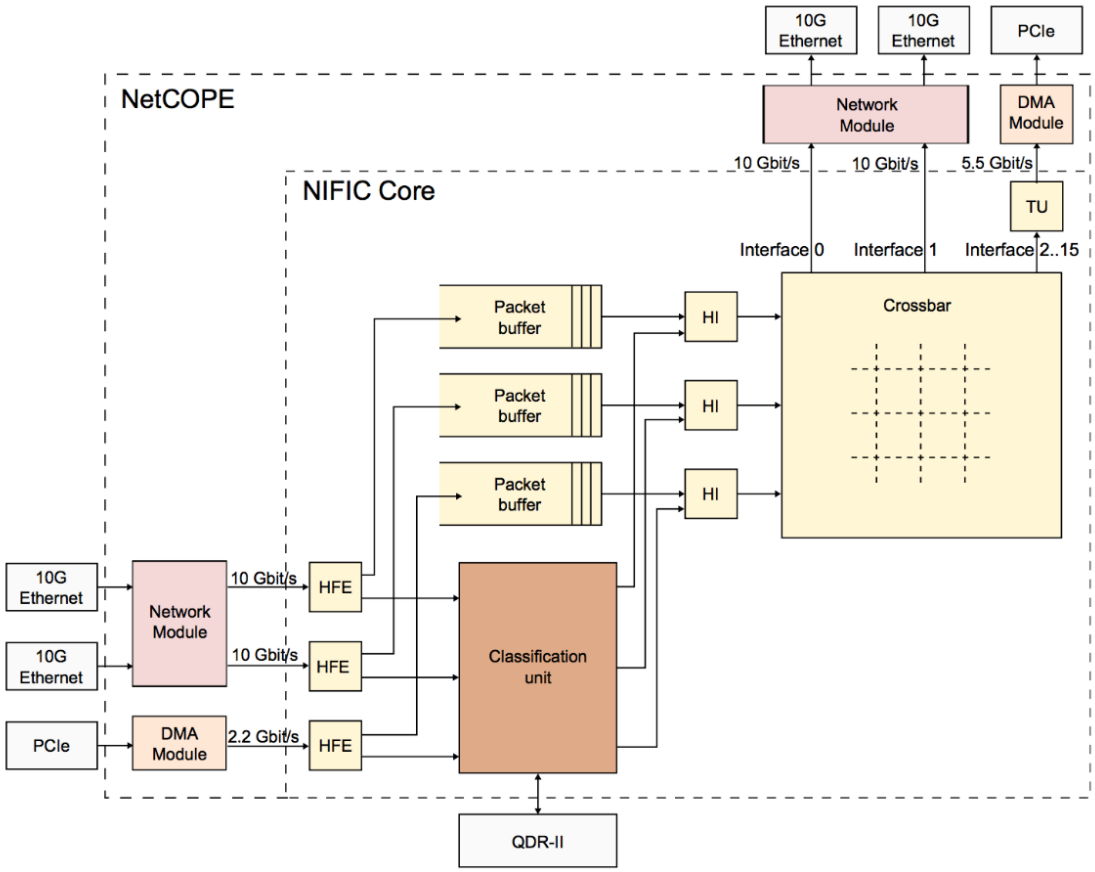


Figura 31: Arquitectura del sistema de comparación

El bloque desarrollado en el TFM alcanza una frecuencia máxima de 383,4 Mhz, lo que supone una tasa de transferencia de datos o *throughput* de 12,2 Gbps. Acorde a la arquitectura usada en el bloque de referencia (Figura 31), un sistema análogo implementado con el bloque desarrollado en este trabajo será capaz de trabajar a 24,4 Gbps.

De los resultados anteriores, se observa que el bloque IP desarrollado es competente con otras soluciones realizadas con anterioridad. El bloque IP desarrollado cumple con las características de sistemas de seguridad de red citados en el estado del arte.

Como conclusión, cabe decir que el límite del sistema está en la tecnología de la Virtex-5 y no en el diseño de los bloques de inspección de cabeceras. Para poder lograr mejorar los resultados, se debería migrar el sistema a otro dispositivo con *transceivers* de

mayor ancho de banda y una tecnología que mejore la frecuencia máxima de funcionamiento.

8.3.2 Sistema de comparación con el núcleo DPFEE

En 2015 se realizó una implementación de un sistema con varios bloques internos que realizan funciones de DPFEE (*Deep Packet Field Extraction Engine*) llamada HSE (*Header Search Engine*). Este bloque HSE presenta una funcionalidad similar a la que realiza el bloque desarrollado en este TFM. Este bloque está implementado en una FPGA de la familia Virtex-7 de Xilinx que provee de un flujo de entrada de varias interfaces de red simultáneamente. El artículo presenta resultados en el sistema desarrollado alcanza unas prestaciones de 25.71 Gbps [4].

Sin embargo, el sistema citado realiza una actividad más compleja, ya que analiza los campos de cabeceras de capas superiores. Si se hiciera una configuración en paralelo de varios bloques como el desarrollado en este TFM, se podría llevar a cabo una tarea similar en la que cada instancia del bloque realiza un análisis en distintas posiciones del paquete. Un ejemplo de esta organización de bloques puede ser la que se representa en la Figura 32.

Para poder hacer una comparativa entre este sistema y el desarrollado en este TFM, se ha analizado el *throughput* del sistema descrito en el artículo por cada interfaz de red y para la misma ocupación. Para ello, se ha sintetizado el bloque desarrollado en el TFM en una Virtex-7 y comparando los resultados de la síntesis.

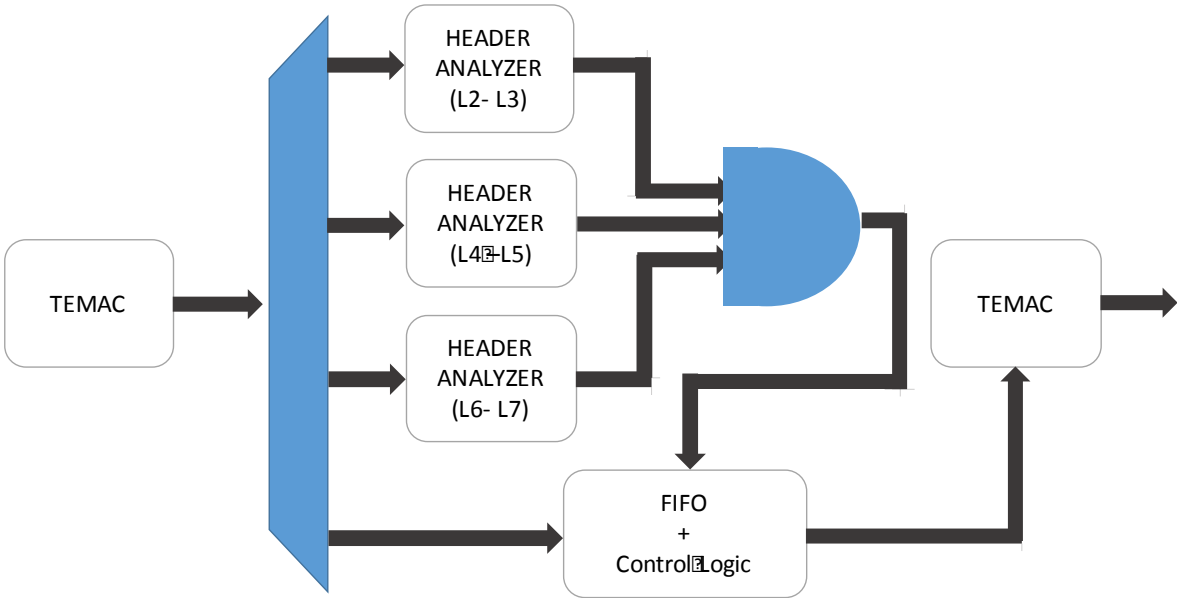


Figura 32: Organización de bloques IP para la comparación con DPFEE

Tabla 5 Tabla análisis soluciones sobre Virtex7

	LUTs	**Throughput**	**FoM**
DPFEE	6%	25.71 Gbps	4.16
Organización de bloques IP	3.66 %	16 Gbps	4.37

La conclusión que se puede obtener de este análisis comparativo es que el bloque desarrollado en el TFM permite trabajar con sistemas con anchos de banda de 16Gbps con un porcentaje de ocupación muy reducida. El bloque citado puede trabajar con sistemas de un ancho de banda 1,5 veces mayor que el ancho de banda conseguido con el bloque IP desarrollado. Sin embargo, el bloque desarrollado presenta características de velocidad bastante buenas para los recursos que demanda del sistema. De esta manera, definiendo una figura de mérito que relacione el ancho de banda alcanzado y los recursos ocupados ($\frac{Ancho\ de\ banda\ [Gbps]}{Área\ ocupada\ [\%]}$), el sistema desarrollado en el TFM presenta mejores resultados tal como se observa en la Tabla 5. Como valor de área ocupada se ha tenido en cuenta el recurso de la FPGA que se presenta como factor limitante en el diseño. En el caso de este diseño el recurso que presenta tales características es el porcentaje de LUTs utilizado.

Capítulo 9. Conclusiones y trabajos futuros

9.1 Conclusiones

El aumento de prestaciones y servicios de tipo Cloud o remoto ha sido superior al desarrollo de sistemas de seguridad en la red. La infraestructura de red trabaja con anchos de banda cada vez más grandes y con mayor densidad de tráfico. Para poder ofrecer servicios de seguridad tanto en entornos domésticos como empresariales y en la infraestructura pública de comunicación es necesario equipos que sean capaces de realizar análisis de seguridad en líneas de alta velocidad sin provocar, por ello, congestiones y retrasos considerables.

Las FPGAs y diseños ASIC juegan un papel importante en el campo de la seguridad de redes de comunicación. Sus características de altas prestaciones, bajo consumo y tamaño reducido hacen de los sistemas basados en FPGA una solución muy competitiva en esta índole. Por estas razones se ha propuesto el desarrollo de un bloque IP encargado de una de las tareas críticas del proceso de análisis en un sistema de estas características: el filtrado de paquetes que cumplen el requisito de análisis.

La idoneidad de los sistemas de seguridad para su uso en una red u otra es determinado por su máxima tasa de transferencia de datos. Esto implica tener en cuenta no solo la latencia del sistema durante el proceso de análisis, sino también la latencia de sistema en permitir el flujo de paquetes que no cumplen el criterio de análisis. Es por ello por lo que se ha planteado el desarrollo de un bloque IP que sea capaz de discernir si redirigir un paquete a la interfaz de red o enviarlo a un sistema de análisis más exhaustivo de una manera más veloz y eficiente.

En este TFM se ha modelado en alto nivel, sintetizado e implementado un bloque IP capaz de realizar filtrado de paquetes TCP/IP en base a los campos de cabecera de los mismos. La motivación a la realización del proyecto es encontrar una arquitectura de sistemas DPI en la que los paquetes de red sean procesados íntegramente en *hardware,* de manera que se evite las latencias en trasferencias a memoria RAM y procesos en *software*.

Para lograr los objetivos que se plantean, se ha optado por una arquitectura tipo *dataflow,* de manera que el bloque diseñado sea capaz de soportar un flujo constante de datos de entrada. Para ello se han explotado las ventajas de la paralelización en *hardware* y el sistema de *pipeline*. De esta manera, el bloque diseñado ha cumplido los objetivos planteados en el Capítulo 1. Se ha logrado diseñar un bloque que es capaz de redirigir tráfico de red a un sistema DPI o de vuelta a la red sin necesidad de procesos *software* para la toma de dicha decisión. Esta característica tiene implicaciones en consumo de potencia y en las latencias del proceso de decisión de la redirección.

El modelado del bloque se ha realizado usando el lenguaje SystemC. El uso de SystemC ha sido clave para desarrollar modelos que requieren de lógica de control y gestión de señales a nivel de ciclos. El proceso de modelado de SystemC es bastante cercano al comportamiento de un sistema modelado en RTL al permitir planificar tareas teniendo en cuenta los ciclos de reloj. Esta característica facilita al desarrollador plasmar la funcionalidad *hardware* deseada de una manera más cercana al resultado final, que es el modelo RTL del bloque descrito en SystemC.

La síntesis de alto nivel se ha llevado a cabo usando la herramienta de CtoS. Esta herramienta permite realizar la síntesis a través de un proceso guiado en el que el desarrollador toma decisiones en cuestiones de implementación de memoria, optimización de bucles y planificación de tareas. Este proceso de toma de decisión sobre la síntesis dota al diseñador de un grado de control sobre el resultado mayor que con otras herramientas automáticas.

La comunicación del bloque IP con el resto de elementos de la plataforma ha sido posible al implementar procesos que controlen las señales del bus para modelar los protocolos de comunicación usados en la plataforma. Para llevar a cabo el modelado de

dichos procesos de comunicación se ha hecho un estudio en detalle de los buses de comunicación AMBA AXI4-Lite y AXI4-Stream. El resultado de este estudio ha permitido afrontar el modelado y síntesis de controladores de bus para cada uno de los protocolos.

El bloque IP desarrollado ha alcanzado los objetivos planteados en el proyecto, ya que la reducción de la latencia debida al *software* y a los accesos a memoria RAM del sistema ha provocado una aceleración de la toma de decisión en un factor de **x100**. En cuanto al ahorro en consumo de potencia, se ha conseguido reducir la potencia en un factor de 7 en el conjunto de bloques que llevaban a cabo las tareas equivalentes en ambas soluciones.

Con respecto a otras soluciones existentes, se han conseguido los objetivos de incrementar el *throughput* por canal de comunicación y reducir la ocupación de área para funcionalidades análogas en las diferentes comparaciones que se han hecho.

9.2 Trabajos futuros.

El desarrollo de este bloque sirve como filtro de paquetes para un sistema de análisis mucho más complejo. Una de las tareas que queda pendiente es la integración del bloque en un sistema completo que explote las características de máximas prestaciones del bloque, usando una interfaz Gigabit Ethernet y un sistema de análisis del *payload.*

Se propone como trabajo futuro el desarrollo de una interfaz de configuración en *runtime* a través de un formulario web alojado en la parte de procesamiento del sistema. De esta manera, el usuario puede consultar y reformular las condiciones de análisis de una manera más cómoda e incluso de manera remota.

Bibliografía y referencias

[1] S. Agrawal y D. Vieira, «A survey on Internet of Things», *Abakós, Belo Horizonte*, vol. 1, n.º 2, pp. 78-95, 2015.

[2] M. Ahmed, A. Naser Mahmood, y J. Hu, «A survey of network anomaly detection techniques», *Journal of Network and Computer Applications*, vol. 60, pp. 19-31, 2016.

[3] PWC, «Turnaround and transformation in cybersecurity - Key findings from The Global State of Information Security Survey 2016», 2015. [En línea]. Disponible en: https://www.pwc.com/gx/en/consulting-services/information-security-survey/assets/pwc-gsiss-2016-financial-services.pdf.

[4] V. Jyothi, S. K. Addepalli, y R. Karri, «Deep Packet Field Extraction Engine (DPFEE): A Pre-processor for Network Intrusion Detection and Denial-of-Service Detection Systems», *Computer Design (ICCD), 2015 33rd IEEE International Conference on, New York, NY*, pp. 266-272, 2015.

[5] F. Eberli, «Next Generation FPGAs and SOCs - How Embedded Systems Can Profit», *Computer Vision and Pattern Recognition Workshops (CVPRW), 2013 IEEE Conference on*. pp. 610-613, 2013.

[6] R. Dobai y L. Sekanina, «Towards evolvable systems based on the Xilinx Zynq platform», en *2013 IEEE International Conference on Evolvable Systems (ICES)*, 2013, pp. 89-95.

[7] P. P. Carballo, O. Espino, R. Neris, P. Hernandez-Fernandez, T. M. Szydzik, y A. Nunez, «Scalable Video Coding Deblocking Filter FPGA and ASIC Implementation

Using High-Level Synthesis Methodology», *Digital System Design (DSD), 2013 Euromicro Conference on*. pp. 415-422, 2013.

[8] E. Sperling, «Plotting The Next Semiconductor Road Map», 2016. [En línea]. Disponible en: http://semiengineering.com/plotting-the-next-semiconductor-road-map/. [Accedido: 20-jun-2016].

[9] Microsoft Inc, «Project Catapult», 2014. [En línea]. Disponible en: http://research.microsoft.com/en-us/projects/catapult/. [Accedido: 11-may-2016].

[10] P. K. Gupta, «Xeon+ FPGA Platform for the Data Center», *Fourth Workshop on the Intersections of Computer Architecture and Reconfigurable Logic*, vol. 119, 2015.

[11] Xilinx Inc., «Software Defined Specification Environment for Networking (SDNet)», 2014. [En línea]. Disponible en: http://www.xilinx.com/publications/prod_mktg/sdnet/backgrounder.pdf. [Accedido: 05-jun-2016].

[12] Xilinx Inc., «Zynq-7000 All Programmable SoC», 2014. [En línea]. Disponible en: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_6/ug873-zynq-ctt.pdf. [Accedido: 15-may-2016].

[13] Xilinx Inc., «Vivado Design Suite», 2015. [En línea]. Disponible en: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_2/ug893-vivado-ide.pdf. [Accedido: 06-may-2016].

[14] Cadence, «C-to-Silicon Compiler High-Level Synthesis User Manual», pp. 1-4, 2010.

[15] Synplify, «Synplify Pro and Premier». [En línea]. Disponible en: https://www.synopsys.com/Tools/Implementation/FPGAImplementation/CapsuleModule/synplify-pro-premier.pdf. [Accedido: 28-abr-2016].

[16] P. P. Carballo, «Aportaciones a la metodología de diseño basada en síntesis de alto nivel. Aplicaciones al diseño de IPs para procesado de eventos complejos y codificación de vídeo», IUMA, Las Palmas de Gran Canaria, 2016.

[17] Xilinx Inc., «Vivado Design Suite User Guide: Design Flows Overview», 2015. [En

línea]. Disponible en: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_1/ug892-vivado-design-flows-overview.pdf. [Accedido: 15-may-2016].

[18] H. Nakahara, T. Sasao, y M. Matsuura, «A regular expression matching using non-deterministic finite automaton», *8th ACM/IEEE International Conference on Formal Methods and Models for Codesign, MEMOCODE 2010*, pp. 73-76, 2010.

[19] B. Vega, P. P. Carballo, P. Hernández-Fernández, A. Domínguez, y A. Núñez, «TCP/IP Packet Analyzer on a Zynq Platform», en *DSD 2015 -- Euromicro Digital Systems Design 2015 (WIP)*, 2015, p. 2.

[20] A. Rao y P. Udupa, «A hardware accelerated system for deep packet inspection», *8th ACM/IEEE International Conference on Formal Methods and Models for Codesign, MEMOCODE 2010*, pp. 89-92, 2010.

[21] A. Takach y M. Meredith, «SystemC Synthesis Working Group». [En línea]. Disponible en: http://accellera.org/activities/working-groups/systemc-synthesis. [Accedido: 15-abr-2016].

[22] J. Verhaegh y A. Su, «SystemC Synthesizable Subset», *Synthesis*, 2009.

[23] Xilinx Inc., «UG761 AXI Reference Guide», 2011. [En línea]. Disponible en: http://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf. [Accedido: 06-may-2016].

[24] G. Combs, «Wireshark». [En línea]. Disponible en: https://www.wireshark.org. [Accedido: 10-abr-2016].

[25] Ospero, «EthernetFMC». [En línea]. Disponible en: http://ethernetfmc.com. [Accedido: 30-abr-2016].

[26] V. Jacobson, C. Leres, y S. McCanne, «TCPDUMP & Libpcap». [En línea]. Disponible en: http://www.tcpdump.org. [Accedido: 28-abr-2016].

[27] W. Wang y T. Dey, «A Survey on ARM Cortex A Processors», 2013. [En línea]. Disponible en: http://www.cs.virginia.edu/~skadron/cs8535_s11/ARM_Cortex.pdf.

[Accedido: 06-jun-2016].

[28] R. Novotny, J., Celeda, P., Dedek, T., and Krejci, «Hardware Acceleration for Cyber Security», *CESNET PRAGUE (CZECH REPUBLIC)*, 2010.