UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

**Universitat Politècnica de València**

**Escuela Técnica Superior de Ingeniería Informática**

**Ingeniería Informática**

**Proyecto Final de Carrera**

---

# Diseño de un bloque hardware de interconexión para un supercomputador

---


***Autor: Ferran Pérez López***

***Directores: Federico Silla Jiménez***
***Yana Esteves Krasteva***


***Febrero de 2012***

# Resumen

Este proyecto final de carrera ha consistido en la adaptación a PCI Express de un diseño de memoria distribuida compartida, MEMSCALE, basado en HyperTransport. También se ha implementado un modelo del supercomputador para poder probar el nuevo diseño. Los resultados de las pruebas realizadas han mostrado que es posible implementar MEMSCALE sobre PCI Express.

**Palabras clave** Memoria distribuida compartida, PCI-Express, FPGA, Modelo supercomputador

# Abstract

In this project a distributed shared memory design based on HyperTransport, MEMSCALE, has been ported to PCI Express. A supercomputer model has also been implemented to be able to test the new design. The results of the tests have shown that it is possible to implement MEMSCALE on PCI Express.

**Keywords** Distributed shared memory, PCI-Express, FPGA, Supercomputer model

# Índice general

# Índice de figuras

# Capítulo 1
# Introducción

## 1.1. MEMSCALE

MEMSCALE [14] es una nueva arquitectura de memoria compartida distribuida para clusters. La característica más importante de MEMSCALE es que la compartición de memoria en el cluster se realiza de forma no coherente permitiendo de esta manera evitar la sobrecarga inherente al protocolo de coherencia entre nodos, que limita las prestaciones y la escalabilidad del sistema. Además MEMSCALE permite accesos a memoria remota de muy baja latencia debido a que los accesos se realizan exclusivamente por hardware, reduciendo así el coste total de distribuir la memoria del sistema. Como se ha mencionado, la clave para alcanzar una baja latencia en los accesos a memoria remota es el uso de hardware específico. En nuestro caso aprovechan la FPGA Virtex4 disponible en cada nodo donde se han implementado diseños propietarios. La tarjeta HyperTransport eXpansion (HTX) FPGA utilizada para ello se puede ver en la figura 1.1. El funcionamiento de las tarjetas es el siguiente, se configuran para que tengan asignado en cada nodo un espacio de direcciones igual al total de memoria que está compartido por todos los nodos (por ejemplo: para 256 GB de memoria compartida, el rango 0x80_0000_0000-0xBF_FFFF_FFFF). De este modo, cada dirección remota se corresponde con una dirección física en cada nodo. Esto permite a las aplicaciones en espacio de usuario hacer una llamada al sistema para mapear en su espacio de direcciones parte (o todo) del espacio de la tarjeta, permitiendo acceder a la memoria remota de la misma forma que si fuera memora local reservada con malloc(). Nótese que una vez reservada la memoria remota, para el resto de los accesos no será necesario hacer más llamadas al sistema operativo, lo que permite las bajas latencias de acceso obtenidas.

MEMSCALE tiene varios escenarios de uso:

- El uso por parte de un nodo de memoria de otros nodos de forma exclusiva, los nodos donde la memoria está físicamente conectada no acceden a la misma. Útil para aplicaciones que no escalan a más cores de los presentes en una placa base pero que pueden beneficiarse de tener más memoria.
- Todos los nodos leen la memoria compartida de todos los nodos. Puede usarse cuando muchos procesos necesitan acceder rápidamente a una gran cantidad de información que no cambia. Los procesos y los datos se distribuirán entre todos los nodos.
- Todos los nodos acceden a la memoria de todos los nodos. Puede usarse como un supercomputador de memoria compartida. Como no hay un protocolo de coherencia entre

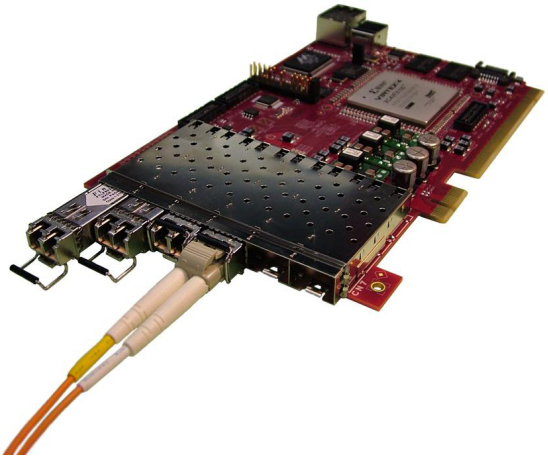

**Figura 1.1:** Tarjeta HTX del prototipo

nodos a nivel hardware, deberán extenderse las primitivas de sincronización entre hilos para que tengan en cuenta esta característica de la arquitectura. La idea, a grandes rasgos, es proporcionar coherencia solo cuando sea necesario.

En el Grupo de Arquitecturas Paralelas (GAP) de la Universitat Politècnica de València (UPV) se ha construido un prototipo de 64 nodos con 1024 núcleos y 1TB de memoria que se puede ver en las figuras 1.2 y 1.3. Cada nodo lleva una tarjeta HyperTransport (HT) con una Field-Programmable Gate Array (FPGA).

Una de las lineas de investigación más relevantes que se llevan a cabo con el prototipo es implementar sobre MEMSCALE una base de datos distribuida, con la que se ha conseguido un gran incremento de prestaciones comparándolo con un disco Solid State Disk (SSD) .

**Figura 1.2:** Prototipo de 64 nodos

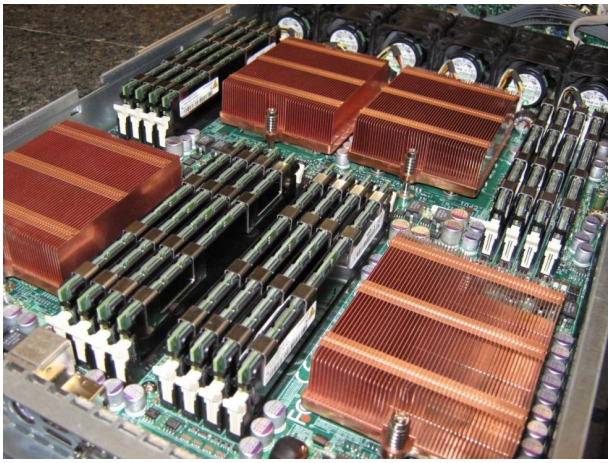

**Figura 1.3:** Un nodo del prototipo

## 1.2. Motivación y Objetivos del Proyecto

El principal objetivo de este Proyecto Final de Carrera (PFC) es portar MEMSCALE a tarjetas PCI Express (PCIe). El conector HTX utilizado actualmente no está disponible para sistemas Intel y los sistemas para AMD que lo implementan son escasos. En cambio, el conector PCIe está presente en prácticamente todos los servidores y sistemas de escritorio modernos. Por lo tanto, una versión de MEMSCALE con tarjetas PCIe ampliaría el rango de sistemas en los que puede usarse y disminuiría el coste construir un cluster con tarjetas MEMSCALE.

En este proyecto final de carrera se desarrollará parte de la tecnología necesaria para portar la arquitectura MEMSCALE a PCI Express. En concreto en este PFC se va a abordar:

1. Adaptar los drivers existentes para Linux.

2. Modificar el diseño hardware: substituir el IP Core para HyperTransport por uno para PCI Express y adaptar el banco de registros.

3. Crear un modelo del supercomputador. El modelo del supercomputador servirá para evaluar las prestaciones del desarrollo y se basará en el comportamiento del prototipo HTX actual. Esto permitirá probar el correcto funcionamiento del diseño PCI Express en una configuración lo más parecida posible a un entrono de uso real dadas las limitaciones del diseño PCI Express actual (aun no se ha implementado la red entre tarjetas), permitiendo hacer una comparación preliminar de prestaciones entre las tarjetas HTX y PCIe.

La pruebas y comparativas se realizaran con test sintéticos y con una base de datos que está usándose actualmente en el prototipo de HyperTransport.

Los resultados de este PFC se integrarán con otros trabajos que se están desarrollando en el GAP para conseguir una solución completa con la que se pueda probar la viabilidad de una versión de MEMSCALE con tarjetas PCIe. Más concretamente, en este trabajo han participado un total de tres personas, incluyendo al proyectando

Además el presente trabajo es uno de los primeros diseños hardware con FPGAs que se abordarán dentro del grupo de investigación. Por ello, parte del proyecto está dedicado a pruebas iniciales de hardware y software.

## 1.3. PCI Express

PCI Express (PCIe) [3] es un estándar para tarjetas de expansión de ordenadores. Es una evolución de Peripheral Component Interconnect (PCI) que aumenta hasta 32 veces la velocidad de PCI 2.1. Este incremento de velocidad se consigue sustituyendo la estructura del bus, evolucionándola de un bus paralelo compartido a enlaces serie punto a punto. Este cambio conlleva una serie de ventajas:

- Al no compartirse el bus, cualquier dispositivo puede comunicarse full-duplex con cualquier otro sin restricciones por el protocolo.

- Las tarjetas lentas no interfieren con las rápidas, en PCI el bus debe funcionar a la frecuencia de la tarjeta más lenta conectada.

- El protocolo serie permite evitar el sesgo de reloj que se produce en señales paralelas a altas frecuencias, cuando por variaciones en el tamaño de los cables unas señales llegan antes que otras.
- Todas la comunicaciones se realizan a través de paquetes por el enlace, permitiendo reducir el numero de pines del conector ya que no se usan pines para interrupciones, errores de paridad, gestión del bus o gestión de energía.
- No cambia el modo en el que los drivers gestionan los dispositivos, permitiendo que programas antiguos que no soportan explícitamente PCIe utilicen los nuevos dispositivos y reduciendo el tiempo de adaptación de los programas y de los desarrolladores.

También permite agrupar varios canales serie (lanes) en un conector (x2, x4, x8, x16, x32). Las placas base incorporan puertos de expansión de diferente tamaño y las tarjetas se diseñan con un conector acorde al ancho de banda que va a necesitar. Esto permite usar el bus PCIe para dispositivos de bajo coste que no necesiten un gran ancho de banda y para tarjetas de altas prestaciones.

Se han publicado tres revisiones principales de la especificación de PCI Express:

**1.x** Tiene un ancho de banda de 250MB/s por cada lane y funciona a 2.5GHz

**2.x** Dobla la frecuencia base a 5GHz consiguiendo un ancho de banda de 500MB/s por lane

**3.0** Tiene un ancho de banda de 1GB/s por lane.

## 1.4. Organización del documento

La estructura de este proyecto final de carrera va a ser la siguiente:

En el segundo capítulo, ***Descripción de los equipos***, se detallarán las especificaciones de las tarjetas y los equipos utilizados.

En el tercer capítulo, ***Pruebas iniciales***, se describirán las pruebas iniciales realizadas con los equipos para comprobar la compatibilidad entre los mismos y familiarizarse con los programas que se utilizarán en el resto del proyecto.

En el cuarto capítulo, ***Diseño Hardware***, se dará una visión general del diseño implementado en la tarjeta y una descripción más detallada de los módulos desarrollados en este proyecto final de carrera.

En el quinto capítulo, ***Adaptación del Software***, se explicarán las modificaciones que se han realizado en el software para adaptarlo a las nuevas tarjetas.

En el sexto capítulo, ***Modelado del supercomputador***, se describirán y compararán los dos módulos desarrollados para modelar los retardos de la red existente en el prototipo actual.

En el séptimo capítulo, ***Pruebas realizadas con el diseño***, se explicará el proceso de desarrollo y depuración del diseño, se analizarán las prestaciones del nuevo diseño y se compararán con las del prototipo actual con tarjetas HTX.

En el último capítulo se valorarán los datos obtenidos en los capítulos anteriores y se propondrán posibles líneas futuras de investigación.

# Capítulo 2
# Descripción de los equipos

En este capítulo se describirán las tarjetas y ordenadores utilizados en este proyecto, detallando sus características y las diferencias entre los mismos.

## 2.1. Xilinx Virtex®-6 FPGA ML605 Evaluation Kit

La tarjeta ML605 [17], mostrada en la figura 2.1, es fabricada por Xilinx y lleva abundante documentación, diseños de referencia y ejemplos. Por ello es una de las más extendidas.

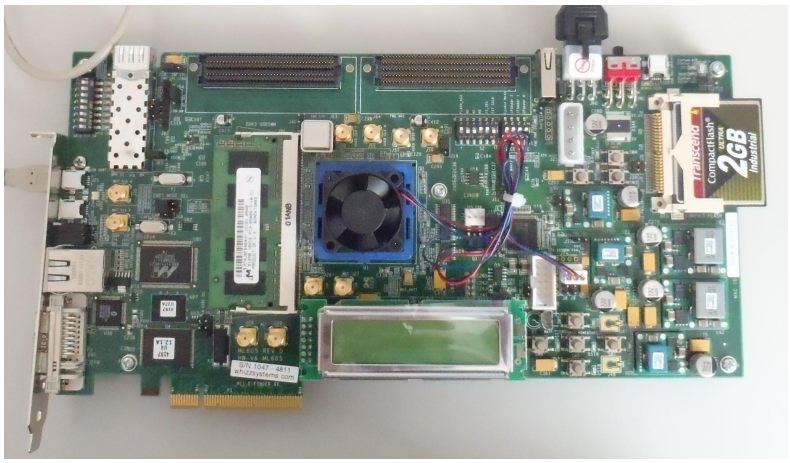

**Figura 2.1:** Xilinx ML605 Evaluation Board

La FPGA montada en la tarjeta es una XC6VLX240T-1FFG1156 que permite configurar la tarjeta en modo PCIe x8 generación 1 o PCIe x4 generación 2, pero no PCIe gen2 x8, lo que limita el ancho de banda máximo a 2GB/s. La tarjeta incluye una memoria de 16MB Platform Flash XL que puede configurar la FPGA en menos de 0.1 segundos, el tiempo máximo que permite la especificación PCI Express [3] para el encendido de los dispositivos. Además, dispone de diversos módulos de comunicaciones:

- Conector PCI Express x8
- 10/100/1000 Tri-Speed Ethernet
- Conector Small form-factor pluggable (SFP) transceiver

- Puente Universal Serial Bus (USB) a Universal Asynchronous Receiver/Transmitter (UART)
- Puerto USB Host y puerto USB dispositivo

También incluye un slot Small Outline Dual In-line Memory Module (SO-DIMM) Double Data Rate 3 Synchronous Dynamic Random Access Memory (DDR3 SDRAM) que viene equipado con un módulo de memoria de 512 MB PC3-8500.

Además de los puertos USB ya mencionados, la tarjeta integra un programador USB, por lo que solo hace falta un cable USB para reprogramar la tarjeta, no hace falta un programador externo.

La tarjeta viene con una memoria CompactFlash precargada con demos y test para probar las funcionalidades de la tarjeta.

## 2.2. HiTech Global VirtexTM-6 LX240T

La tarjeta HiTech Global (HTG) [11], mostrada en la figura 2.2, lleva una FPGA XC6VLX240T-2FFG1156, la única diferencia ente esta FPGA y la de la ML605 es que esta tiene un Speed Grade de 2. Al ser más rápida, permite configurar la tarjeta en PCIe x8 generación 2, consiguiendo el doble de ancho de banda que con la ML605: 4 GB/s.

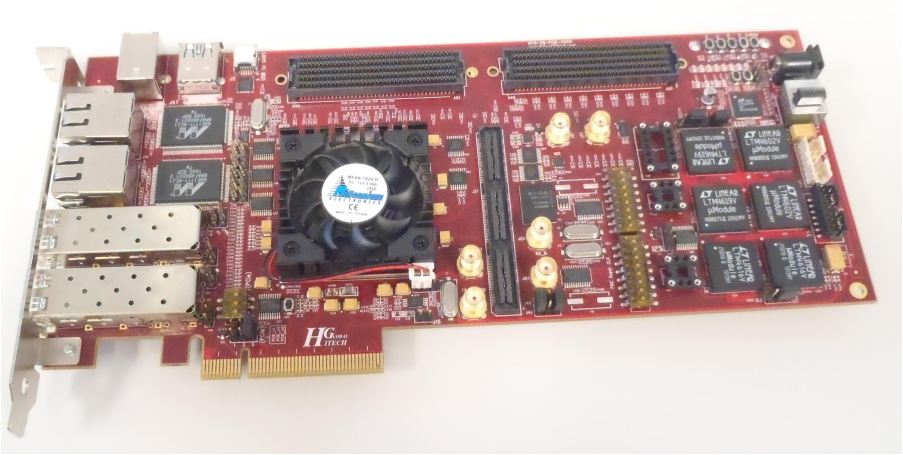

**Figura 2.2:** HiTech Global Virtex-6 LX240T

La documentación en esta tarjeta es más reducida y solo incluye un ejemplo con código fuente.

La tarjeta dispone de los siguientes conectores para comunicaciones:

- USB 2.0 y 3.0 Host
- USB 2.0 y 3.0 Dispositivo
- Dos puertos Small form-factor pluggable transceiver
- Dos Puertos Ethernet 10/100/1000 También incluye un slot SO-DIMM DDR3 que viene equipado con una memoria de 1GB PC3- 8500.

Para programar esta tarjeta, es necesario un programador USB independiente como el que se muestra en la figura 2.3, debido a que esta tarjeta no incluye un programador como la ML605.


**Figura 2.3:** Programador para FPGAs Xilinx USB 2.0

## 2.3. Tarjeta HyperTransport

Se han utilizado 2 tarjetas HyperTransport como las del prototipo para familiarizarse con los drivers y realizar comparativas con el nuevo diseño. Las tarjetas tienen las siguientes características:

- Conector HTX con una interfaz bidireccional de 16-bit LVDS
- 3.2 GByte/s de ancho de banda a través de la interfaz HyperTransport HTX 400MHz
- FPGA Xilinx Virtex-4 FX100
- 256 MB de DDR2 DRAM, interfaz 32-bit a la FPGA
- 512 Mb de memoria flash, interfaz 16-bit a la FPGA
- Interfaz JTAG y USB1.1 para programar la PROM FX100
- Interfaz USB para comunicaciones FPGA-a-PC
- 6 enlaces de alta velocidad con transceptores Small Form Factor Pluggable (SFP)
- Transceptor Gigabit Ethernet con conecto RJ45
- Conector SATA
- Alimentación a través del conector HTX, no necesita alimentación externa.

En la figura 2.4 se muestra una de las tarjetas HTX con un transceptor SFP de 4,25 Gbps para fibra óptica instalado.

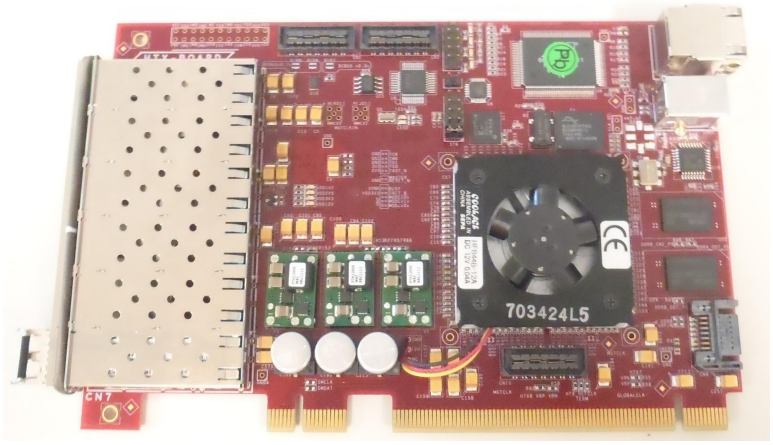


**Figura 2.4:** Tarjeta HyperTransport

## 2.4. Equipos de pruebas

En los siguientes puntos se detallarán las características de los dos equipos que han utilizado para probar el funcionamiento de las tarjetas y los diseños implementados en el proyecto

### 2.4.1. Servidor: Supermicro H8QM8E

Se disponen de dos servidores Supermicro H8QM8E-2+ [7] con las siguientes características:

- 4 procesadores quad core AMD Opteron 8350
- 16GB de RAM DDR2 667 ECC distribuidos de forma uniforme entre los 4 procesadores
- North Bridge: nVidia MCP55 Pro
- South Bridge: AMD-8132
- Puertos de expansión
  - 1 ranura PCI Express x16
  - 1 ranura HT (Hyper Transport)

Estos servidores son idénticos a los que se usan con las tarjetas HTX en el prototipo. En la figura 2.5 se puede ver a uno de los dos servidores Supermicro con una tarjeta HiTech Global instalada y el programador USB.

### 2.4.2. Ordenador personal: Asus M4A78T-E

Para el desarrollo del proyecto final de carrera se utilizará un ordenador con una placa base Asus M4A78T-E [8] con las siguientes características:

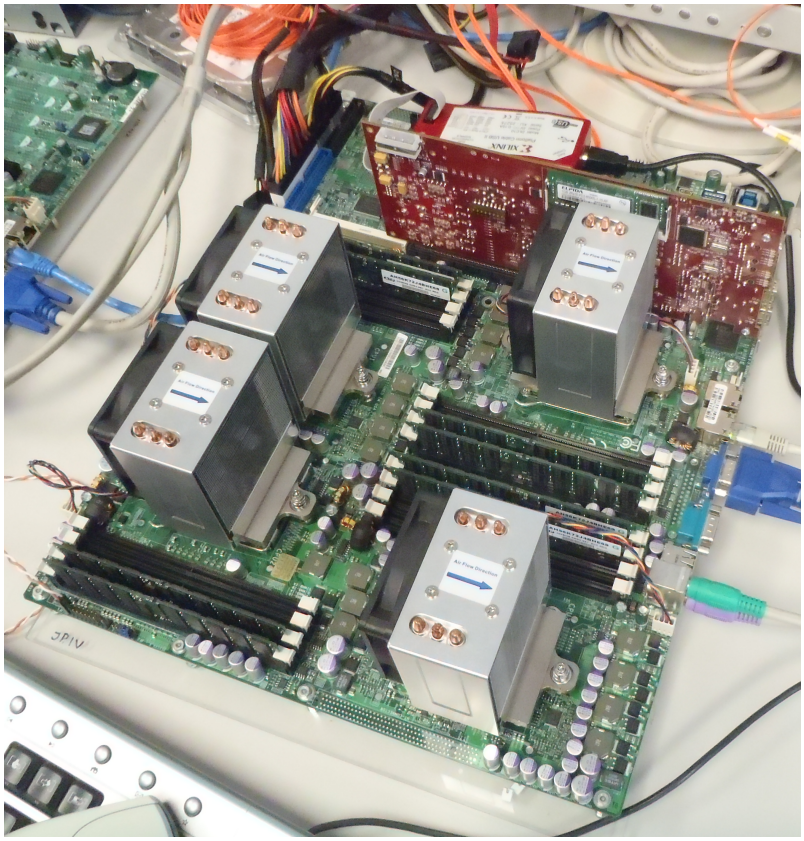

**Figura 2.5:** Servidor Supermicro H8QM8E con tarjeta HiTech Global

- 1 procesador quad core AMD Phenom II X4 945
- 8GB de memoria RAM DDR3 distribuidos en 4 modulos de 2GB
- North Bridge: AMD 790GX
- South Bridge: AMD SB750
- Puertos de expansión:
  - 2 ranura PCIe 2.0 x16, uno x16 o dual a x8.
  - 2 ranura PCIe x1
  - 2 ranura PCI 2.2

Una de las ranuras PCIe 2.0 x16 ya estaba ocupada por la tarjeta gráfica (nVidia GeForce 8400 GS), por lo que se han instalado la tarjeta FPGA en la segunda ranura x16. Al haber dos tarjetas instaladas en las ranuras x16, estas funcionarán a x8, esto no supone ningún problema ya que las tarjetas FPGA tienen un conector de x8.

En la figura 2.6 puede verse el PC Asus M4A78T-E con una tarjeta HTX y el programador USB conectado al mismo ordenador.

**Figura 2.6:** Asus M4A78T-E con una tarjeta HiTech Global VirtexTM-6 LX240T

Por último, en la figura 2.7 se muestra el entorno de trabajo completo que, además, incluye un portátil Lenovo T510i.

**Figura 2.7:** Organización para depurar el diseño en el PC

Este portátil se ha utilizado para poder depurar los diseños ejecutando el programa ChipScope en los numerosos casos en los que el PC quedaba bloqueado imposibilitando la depuración.

# Capítulo 3
# Pruebas iniciales

En este capítulo se introducirá el software utilizado durante el proyecto y las pruebas iniciales realizadas para comprobar el correcto funcionamiento de las tarjetas y las capacidades de los equipos.

La tarjeta Xilinx Virtex®-6 FPGA ML605 y el servidor Supermicro han sido los primeros en estar disponibles. Se han utilizado para familiarizarse con los programas de diseño y depuración que se han utilizado en el proyecto.

El primer contacto con la tarjeta ha sido la ejecución de los tests incluidos en su memoria CompactFlash para probar el correcto funcionamiento de la misma. El test más relevante para este proyecto es el de "PCI Express PIO Demonstration"(XTP044) [19]. En este test, la tarjeta se configura como un dispositivo PCIe que acepta escrituras y lecturas de 4 bytes a una memoria de 2KB.

Para comprobar que el PC detecta la tarjeta y puede acceder a esta, se utilizó PCITree.

## 3.1. PCITree

PCITree [1] es una utilidad gratuita para Windows que permite leer el espacio de configuración de los dispositivos PCI (y derivados como: PCI-X, PCIe o HyperTransport) y acceder y modificar la memoria disponible en los Base Address Registers (BARs).

Los BARs representan diferentes funciones de un dispositivo. Puede haber un máximo de seis por tarjeta. Tienen dos atributos principales: el tamaño, que nunca cambia e indica el espacio de direcciones que necesita cada función y la dirección base, que se asigna cada vez que arranca el ordenador y permite a los programas acceder a los BARs. Cada BAR tiene una dirección de 32 bits, pero pueden combinarse dos BARs si se necesita una dirección base y rango de direcciones de 64 bits.

En la figura 3.1 se muestra la pantalla principal de PCITree con la tarjeta ML605 seleccionada. En la lista de la izquierda se encuentran todos los dispositivos PCI detectados y a la derecha se muestra el espacio de configuración PCI. Puede reconocerse la tarjeta ML605 por sus identificadores de fabricante (0x10EE) y dispositivo (0x6018). Solo está habilitado el BAR0, que tiene asignada la dirección base 0xDFF00000.

Una funcionalidad destacada del programa es que permite comprobar la configuración de los registros específicos, como el de configuración y el de capacidades (features). Estos registros indican la velocidad máxima a la que puede funcionar la tarjeta y la velocidad con la que está configurada actualmente.

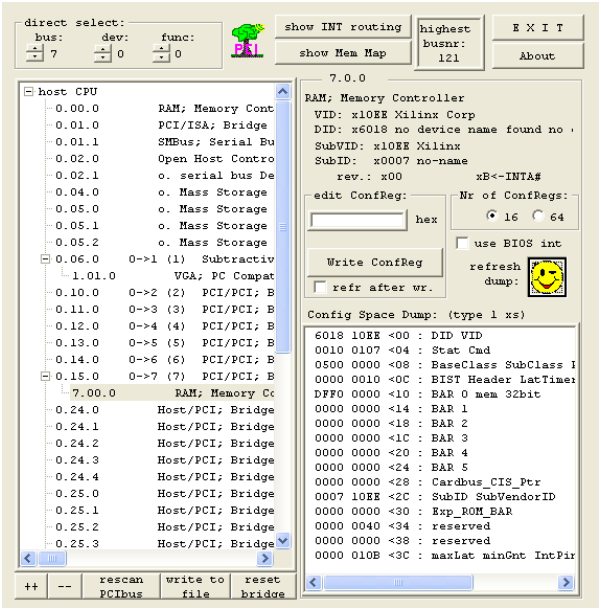


**Figura 3.1:** Pantalla principal de PCITree

En la figura 3.2 se ha seleccionado el registro 0x6C que muestra las capacidades la tarjeta: PCIe gen. 1 x8 (0x81). En el registro 0x70 se muestra el estado actual del enlace entre el servidor y la tarjeta que es el máximo que permite la tarjeta (gen.1 x8).

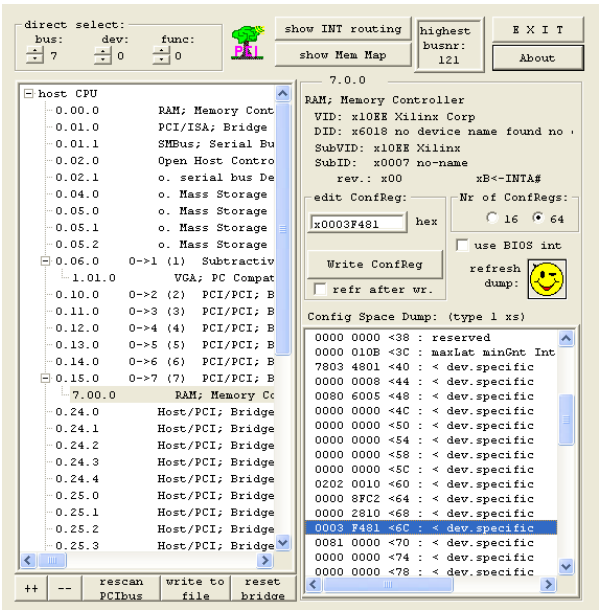


**Figura 3.2:** PCITree mostrando el registro de capacidades

PCITree también permite leer y escribir en la memoria apuntada por los BARs de la tarjeta

palabra a palabra (4 bytes), en bloques de hasta 1024 bytes o ficheros desde 1KB hasta el tamaño especificado por el BAR.

En la figura 3.3 se muestra la pantalla de acceso a los BARs, que permite leer y escribir en la memoria apuntada por el BAR seleccionado. A la izquierda se muestra el rango de 1KB al que se está accediendo (puede cambiarse desde las barras de "select view range"). En la primera columna se muestra el contenido de la memoria en hexadecimal, en la segunda las direcciones de memoria y en la tercera el contenido de la memoria en caracteres ASCII. A la derecha se muestran las distintas opciones para leer y escribir la memoria, el estado que se muestra es después de escribir todo el rango de 1024B con la opción count.

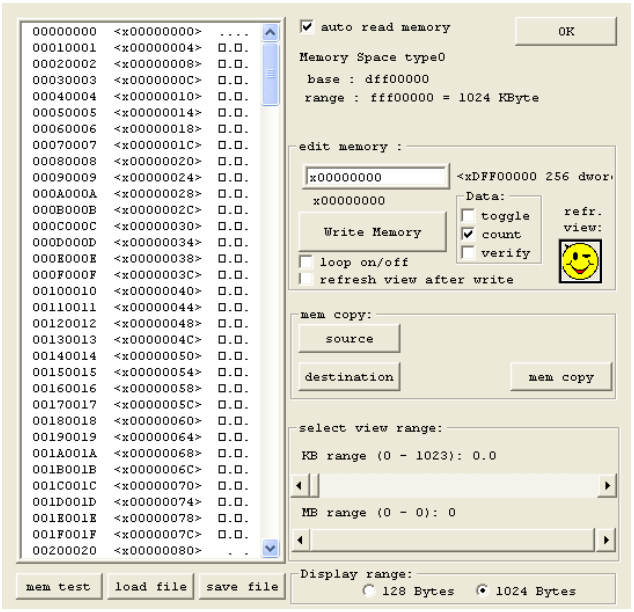


**Figura 3.3:** Pantalla de acceso a la la memoria de los BARs de PCITree

Se han realizado numerosas pruebas con el PCITree, sobre todo para comprobar y controlar las distintas configuraciones de las tarjetas ML605 y HTG.

Uno de las principales limitaciones de PCITree es que solo funciona en Windows de 32 bits, no funciona en las versiones de 64 bits. Esto imposibilita probar los diseños realizados para las nuevas tarjetas ya que estos necesitan un espacio de direcciones de más de 4GB.

## 3.2. Xilinx ISE Design Suite

ISE Design Suite [2] es un conjunto de programas que permite desarrollar diseños para las FPGAs fabricadas por Xilinx. Incluye programas para sintetizar e implementar los diseños, realizar análisis de tiempos, simular los diseños o configurar las FPGAs con un programador.

Las siguientes pruebas realizadas han consistido en configurar la FPGA con las herramientas de desarrollo que proporciona Xilinx. Primero se generó un core con el Xilinx CORE Generator con las mismas características que el que se incluye de fábrica en las memoria CompactFlash de la tarjeta ML605, se sintetizó con los scripts que se crean junto al core y se programó la memoria

Platform Flash XL de la tarjeta con el ISE iMPACT.

En la figura 3.4 se muestra la primera pantalla del asistente para generar el IP Core para PCI Express. Este asistente permite configurar todos los parámetros del core: generación y número de lanes, frecuencia de funcionamiento, configuración de los BARs, generación de interrupciones, identificadores de la tarjeta, configuración de las colas de paquetes o gestión de energía entre otros.

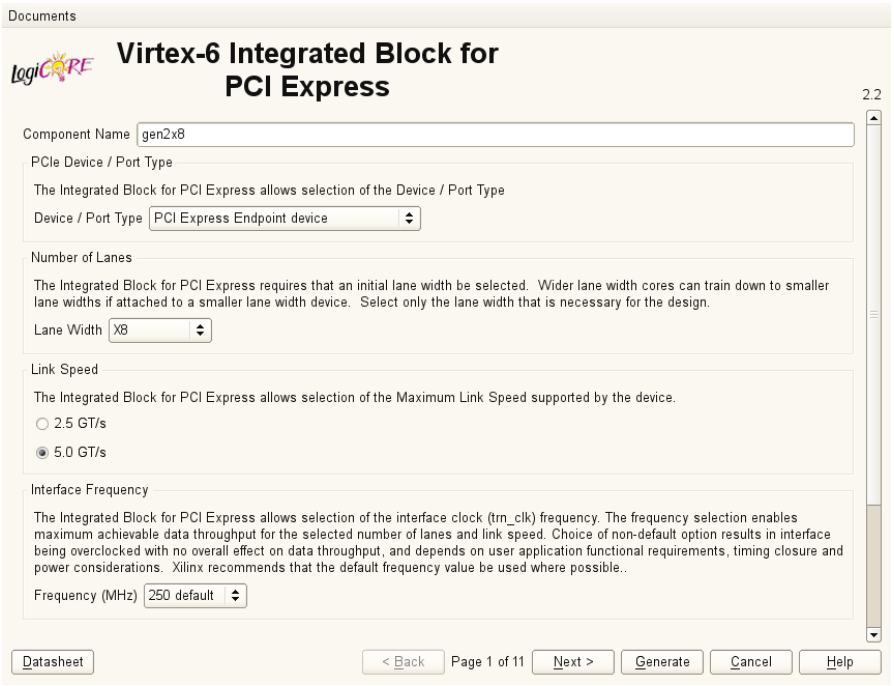


**Figura 3.4:** Xilinx CORE Generator

En la figura 3.5 se muestra iMPACT programando la Platform Flash XL de la tarjeta ML605. IMPACT permite programar los bitfiles directamente en la FPGA y crear ficheros .mcs para programar las memorias flash conectadas a la FPGA.

Después de comprobar que la tarjeta funciona con el nuevo fichero de configuración, se generaron varios ficheros de configuración cambiando la generación (1 y 2) y número de lanes (2, 4 y 8). Estos ficheros de probaron en el servidor Supermicro H8QM8E con y sin el raiser-card (figura 3.6) que se utiliza para las tarjetas HyperTransport. Se han probado los distintos cores con el PCITree. De las pruebas se comprobó que el raiser-card no funciona con la tarjeta PCIe y la placa base no soporta PCI Express generación 2.

En la figura 3.7 se muestra la pantalla principal de ISE Project Navigator, este programa permite gestionar todo el proceso de desarrollo, desde la escritura del código fuente a la programación de las FPGAs. A la izquierda de la imagen se encuentran los ficheros de código fuente organizados según la jerarquía del diseño y los procesos necesarios para programar la tarjeta, que puede ejecutarse desde la misma interfaz. En la parte inferior se encuentra la consola que muestra los mensajes generados por los procesos. En la parte central y derecha se muestran el resumen del diseño con todos los informes que han generado los procesos. En esta área también pueden editarse los ficheros de código fuente.

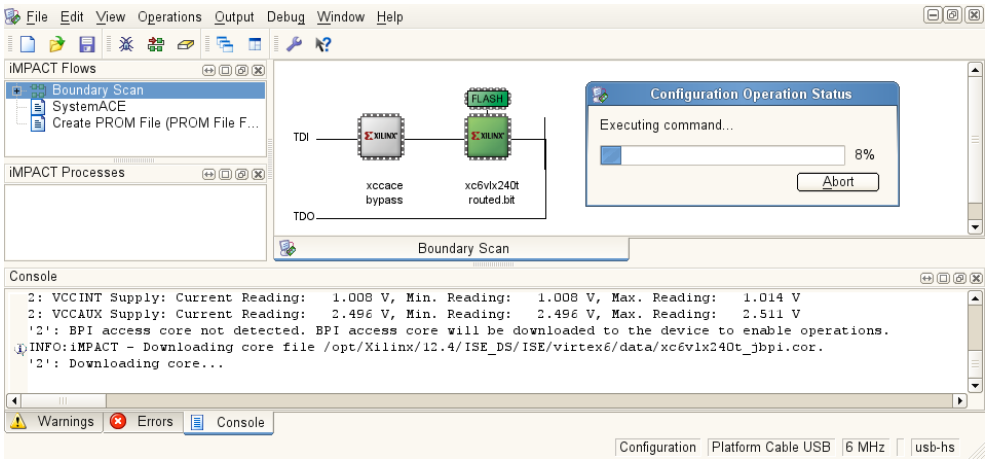


**Figura 3.5:** iMPACT programando la Platform Flash XL

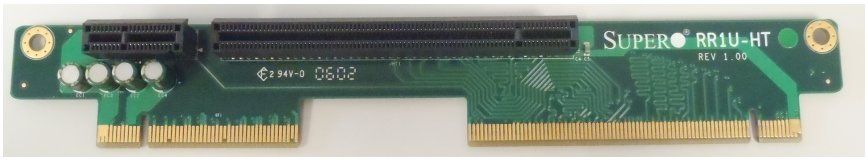


**Figura 3.6:** Raiser card utilizado en las tarjetas HyperTransport

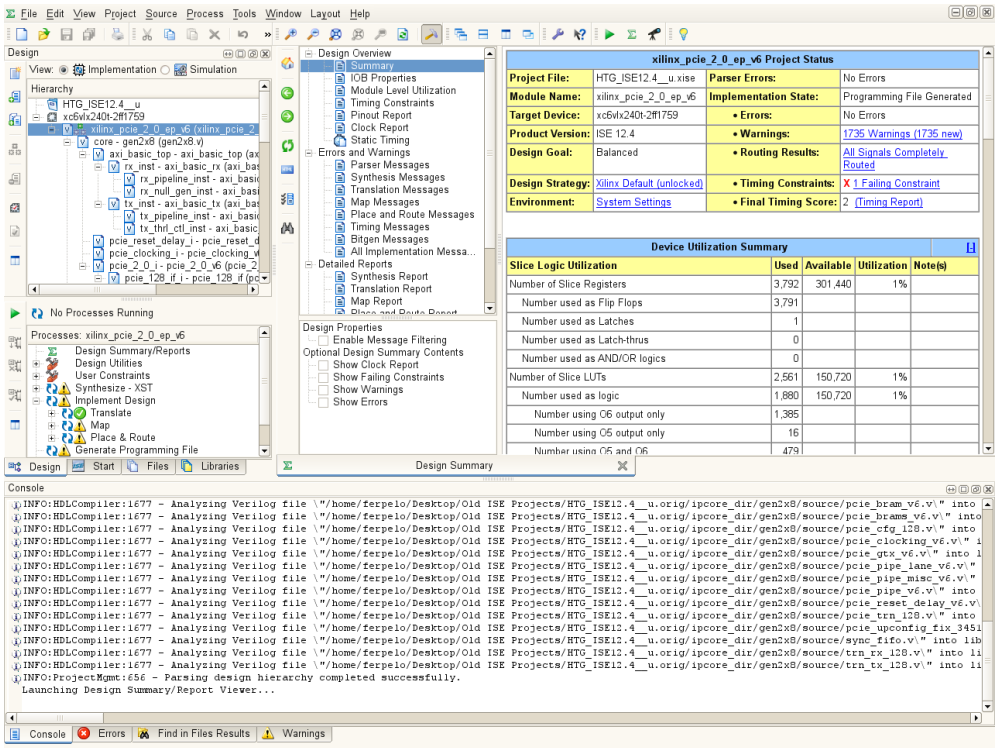


**Figura 3.7:** Pantalla principal ISE Project Navigator

En la siguiente etapa se ha intentado generar los ficheros de configuración desde el ISE Project Navigator, se ha añadido el fichero .xco del core PCIe y los ficheros Verilog de la memoria de ejemplo, pero como se producen errores de ficheros desaparecidos, se ha decidido incluir los ficheros de código fuente en Verilog que se generan junto al core y eliminar el .xco. Este cambio también tiene la ventaja de que permite modificar el código del core si fuera necesario. De este modo se ha conseguido generar un fichero de configuración desde el ISE Project Navigator que funciona.

## 3.3. ChipScope

El siguiente paso ha sido familiarizarse con la herramienta de depuración ChipScope. ChipScope es una herramienta que inserta analizadores lógicos en los diseños de FPGAs de Xilinx, permitiendo ver las señales del circuito mientras éste está funcionado. El número de señales y los ciclos consecutivos que pueden verse están limitados y deben de definirse antes de generar el fichero de configuración de la FPGA. Además, también han de definirse triggers para configurar cuando se empieza a guardar el valor de las señales a analizar.

En la figura 3.8 se muestra la pantalla principal de ChipScope. En la columna de la izquierda, se pueden seleccionar los analizadores lógicos insertados y las señales de los mismos. La ventana superior permite configurar los triggers: cuales están seleccionados, con qué valor se activan y cuantos ciclos deben de almacenarse cada vez que se activen. La ventana central muestra el valor de las señales analizadas y la inferior los mensajes de información y error.

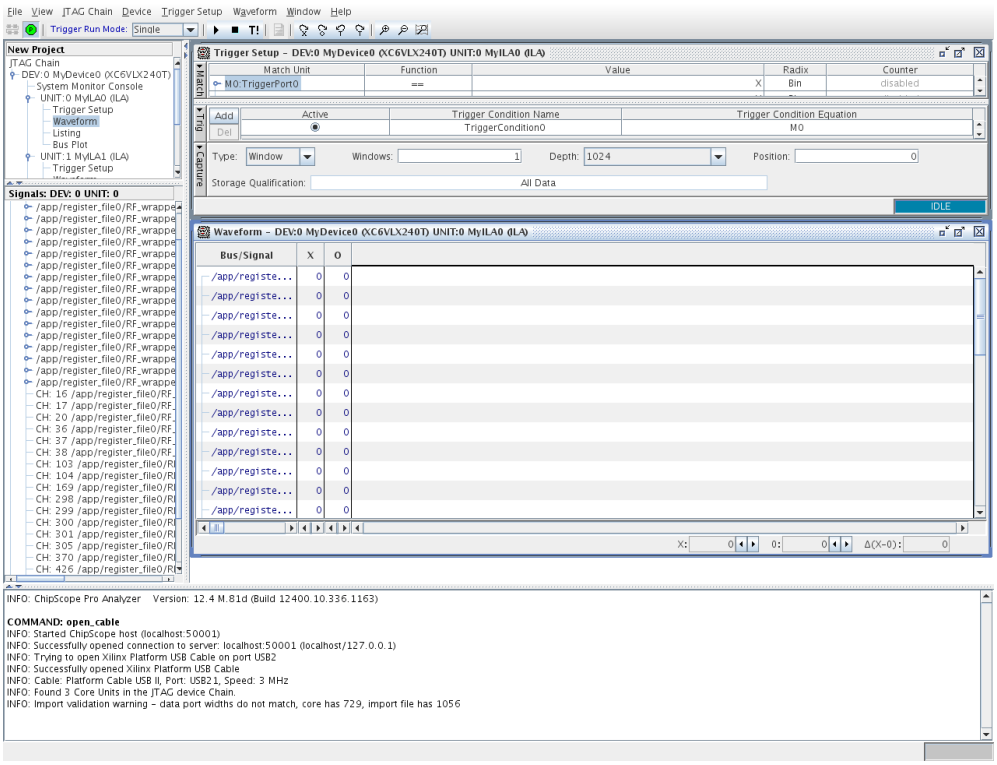


**Figura 3.8:** Pantalla principal del ChipScope

Como primer contacto, se ha seguido la guía “ML605 MIG Design Creation” (XTP047)[20]. Esta guía explica como utilizar el ChipScope Pro Analyzer con un fichero de configuración en el que ya se ha insertado el analizador lógico, configurar los triggers y visualizar las señales.

El ChipScope Pro Core Inserter es el programa que inserta los analizadores lógicos en los diseños. La figura 3.9 muestra la pestaña Net Connections con las señales seleccionadas para ser analizadas (data port) y los triggers para obtener valores de las señales cuando sean relevantes.

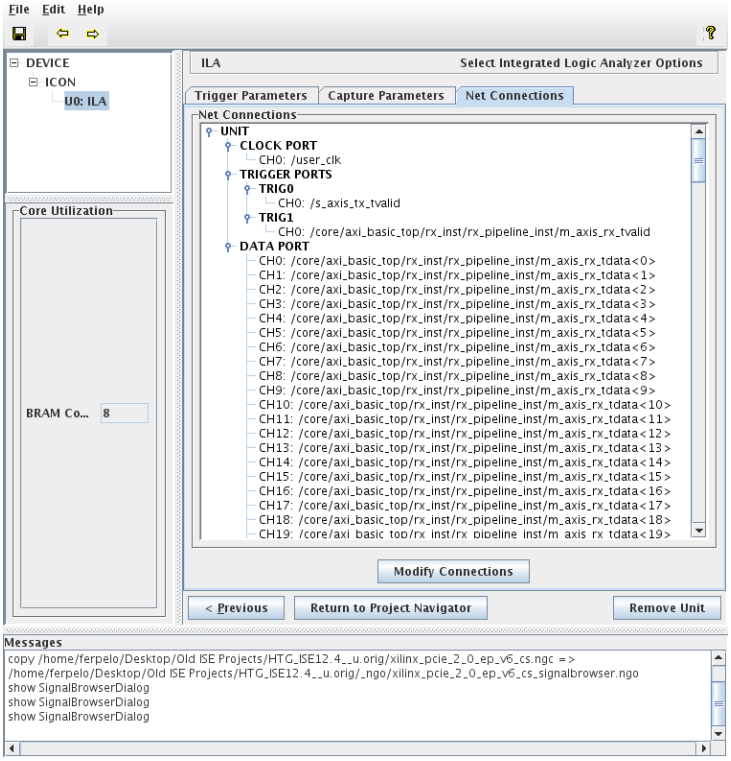


**Figura 3.9:** ChipScope Pro Core Inserter

Una vez se ha adquirido confianza con el uso del ChipScope, se ha añadido un analizador lógico al diseño con el core PCIe que se había creado anteriormente. Después de añadir señales para conocer mejor el funcionamiento del diseño de ejemplo (como los buses AXI o los puertos de la memoria), se han analizado las señales de la tarjeta creando tráfico con el PCITree

Observar el funcionamiento del core con el ChipScope ha sido una gran ayuda para entender el protocolo AXI4-Stream de comunicación entre el core y la memoria y el protocolo de paquetes PCI Express con el que la memoria se comunica con el ordenador.

En la figura 3.10 se muestra una captura de ChipScope con una lectura realizada con el PCITree, en el puntero X (azul), pueden verse los primeros 8 bytes del paquete con la petición de lectura de 4 bytes con dirección de 32 bits (0xDFF00000, la dirección base del BAR0). El puntero O (verde) muestra los primeros 8 bytes de la respuesta.

Se han repetido parte de las pruebas con la tarjeta HiTech Global y el PC Asus. Las pruebas más relevantes han sido las que han servido para validar el correcto funcionamiento de los diseños creados desde cero sobre las tarjetas en todos los modos PCIe: generación 1 y 2 y número de lanes x1, x2, x4 y x8. Un problema destacable que se ha encontrado ha sido que la configuración

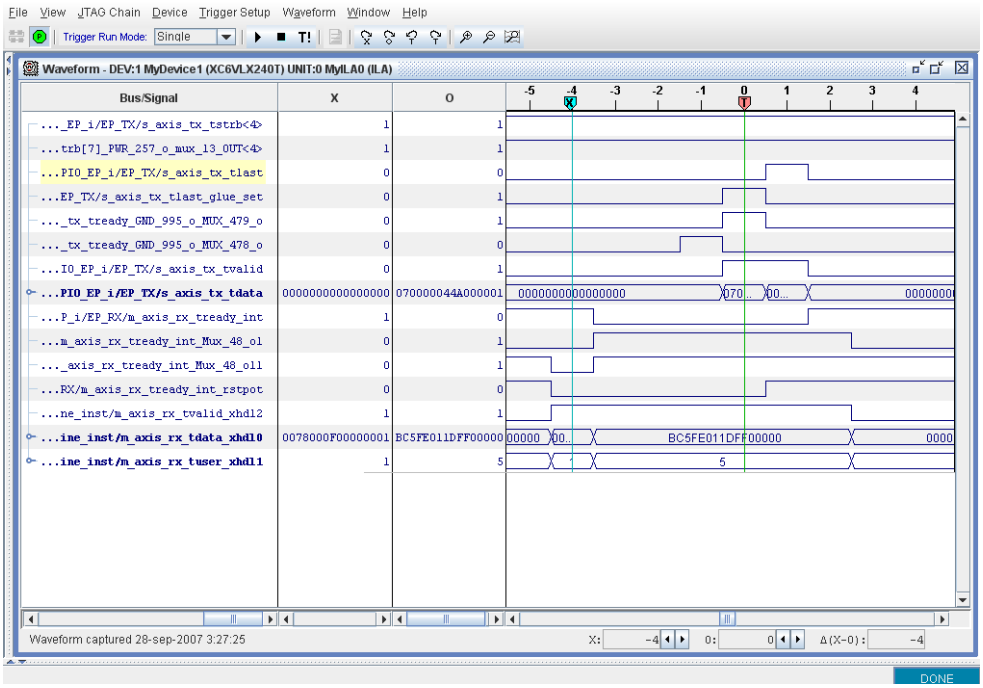


**Figura 3.10:** Captura en ChipScope de la memoria de ejemplo

por defecto del ISE 12.4 genera bitfiles que funcionan programando la FPGA pero no funcionan programando la Flash. Para resolver el problema, se ha copiado la configuración del script para generar el bitfile que se incluye en el ejemplo con código fuente de la documentación de la tarjeta.

## 3.4. lspci

Por último se han probado la tarjeta HiTech Global y el PC Asus con SuSE Linux 11.1 de 64 bits. Para acceder a la tarjeta se ha utilizado `lspci`, esta utilidad por consola forma parte del paquete `pciutils`. `lspci` permite listar las tarjetas PCI , acceder al sus espacios de configuración y mostrar sus registros en hexadecimal o decodificar los mismos en un formato más entendible para las personas. A continuación se muestran partes de una ejecución de `lspci` en una tarjeta HTG.

Orden con parámetros e identificadores de la tarjeta:

```
# lspci -d 0007:0008 -vvv -nn -xxx -H1
02:00.0 RAM memory [0500]: Device [0007:0008] (rev ef)
```

BARs:

```
Region 0: Memory at d0000000 (64-bit, non-prefetchable) [disabled]
Region 2: Memory at c0000000 (64-bit, non-prefetchable) [disabled]
Region 4: Memory at <unassigned> (64-bit, prefetchable) [disabled]
```

Capacidades y estado:

```
LnkCap:         Port #0, Speed 2.5GT/s, Width x8
LnkSta:         Speed 2.5GT/s, Width x8
```

`lspci` permite elegir entre varios métodos para acceder la configuración de la tarjeta. Aunque el volcado de los registros en hexadecimal (opción `-xxx`) siempre accede a los registros del dispositivo, la información decodificada puede que se obtenga del kernel sin acceder a los dispositivos, Esto supone un problema ya que cuando se cargan los drivers de las tarjetas, se realizan cambios en los espacios de configuración PCI que el kernel no registra. Para que `lspci` siempre acceda a los registros para decodificar la información se le pasa la opción `-H1`

Una de las principales ventajas de `lspci` sobre PCITree es que el primero decodifica mucha más información, lo que permite saber con mucha más facilidad el estado en el que encuentra la tarjeta.

Un inconveniente de `lspci` es que no permite acceder a la memoria de los BARs. Pero esto no ha supuesto un gran problema ya que en GNU/Linux se dispone de los drivers para las tarjetas HTX que con cambios mínimos se ha podido utilizar para acceder a la memoria apuntada por los BARs .

Como resultado de este primer estudio se han comprobado las capacidades de los equipos que se utilizarán en el resto del proyecto, se han adquirido los conocimientos para poder programar y depurar los diseños en las tarjetas, se ha entendido la interfaz AXI4-Stream del core PCIe y se ha familiarizado con el protocolo de paquetes de PCI Express.

# Capítulo 4
# Diseño Hardware

En este capítulo se dará una visión general del diseño que se ha implementado en la FPGA para hacer posible MEMSCALE sobre PCI Express y se explicarán más en detalle los módulos que forman parte del trabajo de este proyecto final de carrera.

## 4.1. Descripción general

Durante el proceso de adaptación de las tarjetas de HyperTransport a PCI Express se ha intentado mantener la mayor parte posible de los módulos para garantizar la compatibilidad con los drivers y disminuir el tiempo de desarrollo.

El diseño HyperTransport del que se ha partido se muestra en la figura 4.1. El HT-Core se utiliza para comunicarse por el bus HT. El módulos HTAX se encarga de interconectar los módulos principales de la tarjeta. Los módulos en gris oscuro no se incluyeron en la FPGA debido a restricciones de espacio. El Register File (RF) se encarga de configurar y monitorizar el resto de los módulos. La Shared Memory Functional Unit (SMFU) se encarga de la gestionar los accesos a memoria remota. Por último el Network Port, el Crossbar (XBAR) y los Link Port (LP) se encargan de gestionar las comunicaciones por la red de fibra óptica.

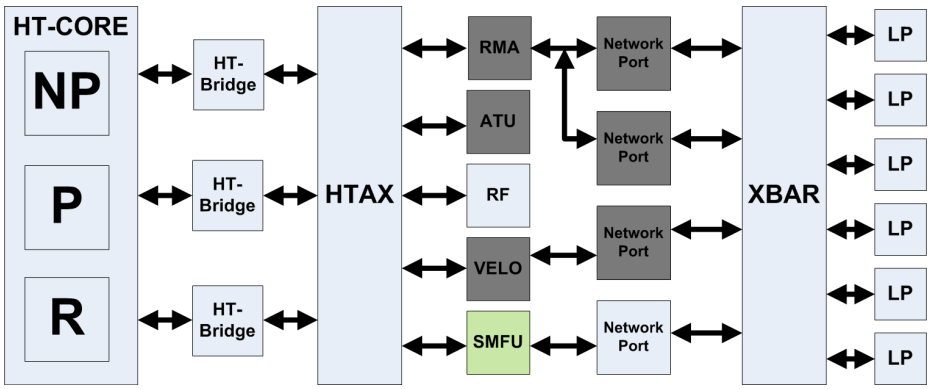


**Figura 4.1:** Diagrama de bloques del diseño HyperTansport [15]

Los módulos del diseño en las tarjetas HTX que han necesitado menos cambios han sido el switch y el banco de registros (RF), el cual se explicará en la sección 4.8. En la misma linea de mantener la compatibilidad con el diseño ya existente, el lenguaje de descripción de hardware

que se ha seleccionado para la versión de PCIe es Verilog.

Todos los módulos del diseño original han sido modificados y/o re-escritos de tal forma que soporten el protocolo de comunicación interno del sistema AMBA 4 AXI4-Stream [9] con un ancho de 128 bits de datos (`tdata`) y 32 bits de datos auxiliares (`tuser`). La motivación para la selección de estas opciones se discutirá en detalle durante la descripción de los módulos correspondientes.

En la figura 4.2 se muestra la estructura general del nuevo diseño. La única parte que se comunica con el exterior de la FPGA es el core PCIe. En el diseño definitivo el módulo de red también estará conectado a los puertos de entrada/salida, pero como aun no está disponible se ha creado el modelo de red (capítulo 6). El bridge 64/128 solo estará presente si la interfaz del core es un AXI de 64 bits, que son los casos en los que se empleen cores PCIe de generación 1. En los casos de los cores PCIe generación 2 x8 que tiene la interfaz de 128 bits, se conectará el core directamente a las async FIFOs (sin bridge). Las async FIFOs se han generadas con el Core Generator de Xilinx, tienen una interfaz AXI4- Stream y relojes independientes para las lecturas y escrituras. Su principal función es permitir que la SMFU funcione a menor velocidad que la interfaz del core PCIe.

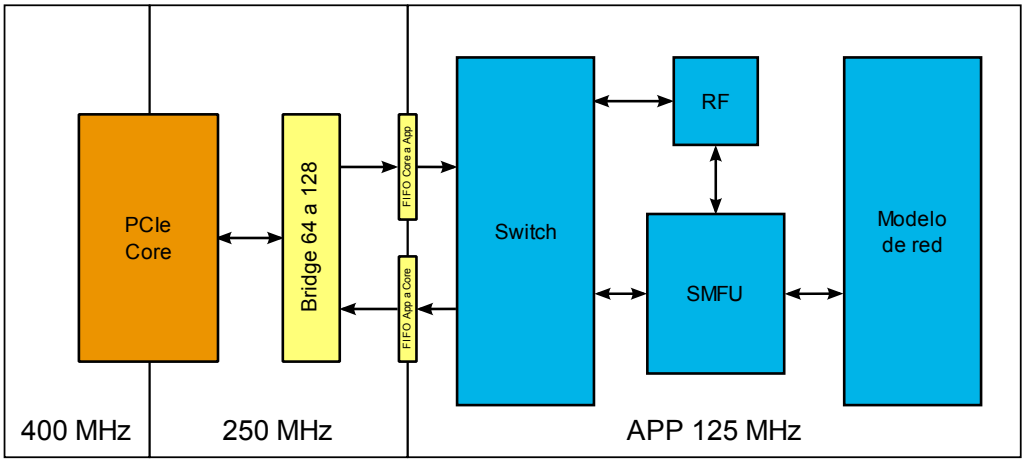


**Figura 4.2:** Diagrama de bloques del diseño

Al conjunto del switch, el banco de registros, la SMFU y el modelo de red se le llamará Remote Memory Controller (RMC) .

La adaptación del diseño a PCIe se ha realizado conjuntamente por un grupo de tres personas dentro del proyecto MEMSCALE (sección 1.1). La adaptación de la SMFU la ha realizado Santiago Mislata y ha sido su proyecto final de carrera [12]. El bridge y la adaptación del switch ha corrido a cargo de Yana Krasteva, la codirectora de este proyecto. La adaptación del banco de registros, la configuración del Core PCIe, el modelo de red y la integración de todos los módulos se ha realizado en este proyecto final de carrera.

## 4.2. AMBA 4 AXI4-Stream

El protocolo AXI4-Stream [9] permite conectar un componente maestro, que genera información, a un esclavo. Aunque también puede utilizarse para conectar varios maestros y esclavos,

en este documento sólo se describirán los aspectos del protocolo utilizados en el diseño. Señales del protocolo:

**tdata** Contiene los datos a transferir. Tiene un tamaño C_DATA_WIDTH.

**tvalid** Indica si los datos son válidos

**tready** Esta señal la activa el esclavo para indicar que puede recibir datos. Una transferencia se considera válida cuando `tvalid` y `tready` están activos en el mismo ciclo.

**tlast** Indica el final de la transferencia de un paquete

**tuser** Se usa para incluir información adicional de `tdata`.

**tstrobe** Esta señal tiene un tamaño de C_DATA_WIDTH/8 y indica qué bytes de `tdata` son válidos. Por ejemplo, para un `tdata` de 64 bits y un `tstrobe` con valor 0x0F, significa que los bits 32 a 63 de `tdata` deben descartarse.

Todas las señales del protocolo son síncronas. Para que se realice una transmisión, el maestro debe de activar `tvalid`, cuando el esclavo active `tready` (si no lo estaba ya), la transferencia se considerará válida y el maestro deberá desactivar `tvalid` o reemplazar los datos por los siguientes a transmitir.

En la figura 4.3 se muestra una transacción AXI4-Stream donde `tready` se ha activado después de `tvalid` y el maestro solo tenía que transmitir un ciclo de información.

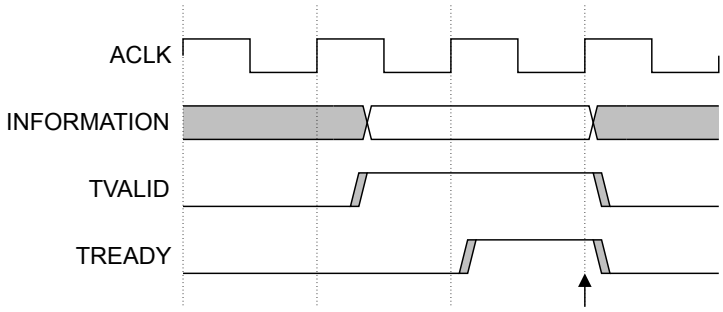


**Figura 4.3:** Transacción AXI4-Stream [9]

La utilización de este protocolo para los módulos del diseño presenta varias ventajas:

- Permite controlar el flujo de paquetes dentro del diseño. Si un módulo se satura, los módulos conectados al mismo pararán de enviarle información hasta que indique que vuelve a estar disponible, incluso en medio de un paquete y sin perdida de datos.
- El diseño resultante es mucho más modular que el actual de las tarjetas HTX. Permite quitar, mover y añadir módulos al diseño con cambios mínimos o nulos al resto de módulos. Como se explicará en el capítulo 7, se han podido añadir los módulos de forma incremental gracias a la interfaz común de los módulos.

## 4.3. PCI Express Core

Este módulo[18] es un Intellectual Property (IP) Core proporcionado por Xilinx para facilitar el uso de los bloques PCIe que llevan las Virtex 5 y 6. Este core es de tipo hard core, lo que significa que está fundido en el silicio de la FPGA. A pesar de ello, el core permite realizar una serie de configuraciones que luego son implementadas en la lógica de la FPGA. Todas las configuraciones del core son accesibles por medio de un asistente gráfico del IP Core Generator. Para incluir en el diseño el core resultante, puede añadirse un fichero .xco o los ficheros en Verilog (o VHDL) que implementan un wrapper y la lógica para configurar el hard core. La opción elegida ha sido la de los ficheros en Verilog, ya que como se ha visto en el capítulo 3, el fichero .xco provoca errores en algunos casos y los ficheros en Verilog pueden modificarse, como ha sido necesario para poder conectar simultáneamente a los relojes de 250 Mhz y 125 Mhz, ya que desde la interfaz de configuración generada por defecto sólo se permite acceder al reloj de 250 Mhz.

La interfaz que utiliza el core para comunicarse con los demás módulos es AXI4-Stream. Se utilizan dos interfaces, una `m_axis_rx` en la que el core es el master y el RMC es el esclavo. Por este canal se transmiten los paquetes PCIe que llegan por el bus. El otro canal es `s_axis_tx`, el master es el RMC, el esclavo el core y se utiliza para enviar paquetes por el bus PCIe.

Estas dos interfaces (`m_axis_rx` y `s_axis_tx`) se diferencian únicamente en las señales de tuser.

**s_axis_tx_tuser** de el RMC al core.

- **s_axis_tx_tuser[1] (terr_fwd)** indica al core que el paquete que se está enviando contiene errores (*error-poisoned*).
- **s_axis_tx_tuser[2] (str)** indica que todos los ciclos del paquete se enviarán consecutivamente, por lo que el core puede empezar a enviar el paquete por el bus PCIe antes de tener todo el paquete en los buffers.
- **s_axis_tx_tuser[3] (src_dsc)** si se activa, el core descarta el paquete que se está transmitiendo. Es incompatible con la señal `s_axis_tx_tuser[2]`.

**m_axis_rx_tuser** del core a el RMC.

- **m_axis_rx_tuser[1] (rerr_fwd)** indica a el RMC que el paquete que se está enviando contiene errores (*error-poisoned*).
- **m_axis_rx_tuser[9:2] (bar_hit[6:0])** indica a que BAR va dirigido el paquete actual. Si dos BARs están configurados como uno solo de 64 bits, se activarán las señales correspondientes a los dos BARs. Las correspondencias son las siguientes.

  **m_axis_rx_tuser[2]** BAR0
  **m_axis_rx_tuser[3]** BAR1
  **m_axis_rx_tuser[4]** BAR2
  **m_axis_rx_tuser[5]** BAR3
  **m_axis_rx_tuser[6]** BAR4
  **m_axis_rx_tuser[7]** BAR5
  **m_axis_rx_tuser[8]** Expansion ROM Address

**m_axis_rx_tuser[9]** está reservada para futuros usos.

En la figura 4.4 se muestra una transacción de un paquete de 12 bytes por la interfaz de 64 bits con destino al core. Se puede observar como en el segundo ciclo, `s_axis_tx_tstrobe` vale 0x0F para marcar que los últimos 32 bits deben descartarse porque no forman parte del paquete.

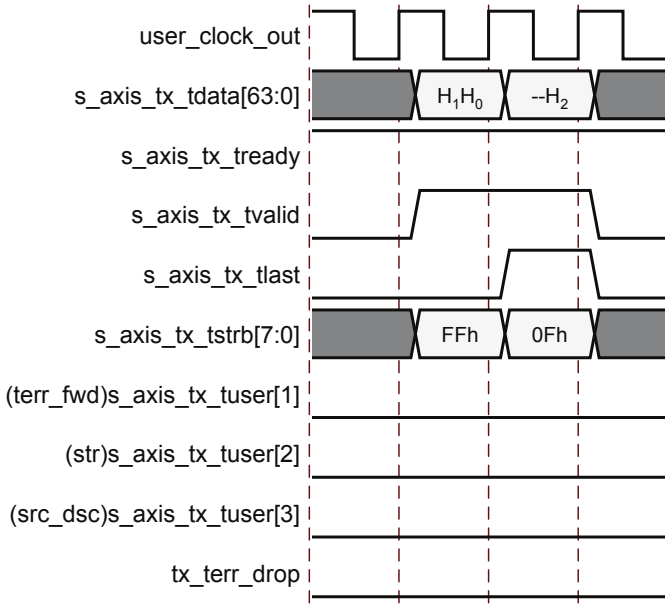


**Figura 4.4:** Transmisión de un paquete al core [18]

La única diferencia entre las interfaces de 64 y 128 bits es debida a que los paquetes PCIe tienen una granularidad de 32 bits. Para poder transmitirlos por los buses de 64 o 128 bits sin que se alteren, el core de Xilinx emplea dos estrategias diferentes según el ancho de la interfaz:

**64 bits** se utiliza el bus de `tstrobe` para marcar si los últimos bytes del ciclo son válidos.

**128 bits** en esta interfaz usa el bus `tstrobe` en la interfaz de el RMC al core y las señales `is_eof` y `is_sop` de `tuser` en la interfaz del core a el RMC.

Las señales `is_eof` (End Of Frame) y `is_sof` (Start Of Frame) tienen cinco bits cada una. No marcan explicitamente los bytes inválidos sino que indican si en el ciclo actual empieza y/o termina un paquete y en qué byte ocurre. El significado de los bits de las señales es el siguiente:

**is_sof[4] (m_axis_rx_tuser[14])** Se activa a uno cuando en el ciclo actual empieza un paquete.

**is_sof[3:0] (m_axis_rx_tuser[13:10])** Marca en qué byte del ciclo actual empieza el paquete. Solo son significativos si `is_sof[4]` está activo. Los únicos valores válidos son 0000b y 1100b.

**is_eof[4] (m_axis_rx_tuser[21])** Se activa a uno cuando en el ciclo actual termina un paquete.

**is_eof[3:0] (m_axis_rx_tuser[20:17])** Marca en qué byte del ciclo actual termina el paquete. Solo son significativos si `is_eof[4]` está activo. Los únicos valores válidos son 0011b, 0111b, 1011b y 1111b.

El motivo para cambiar las señales de `tstrobe` por las `is_sof`/`is_eof` es que las segundas permiten que en el mismo ciclo termine un paquete y empiece el siguiente, obteniendo el doble de rendimiento que la interfaz que 64 bits, lo que no sería posible con las señales de `tstrobe`.

En la figura 4.5 se muestra un diagrama de tiempos en el que el core envía dos paquetes, transmitiendo en el segundo ciclo el final del primer paquete (D1) y el principio del segundo (H1H2). En el primer ciclo se observa que `is_eof` tiene el valor 00011b, pero como `is_eof[4]` es 0 el resto de los bits se ignoran.

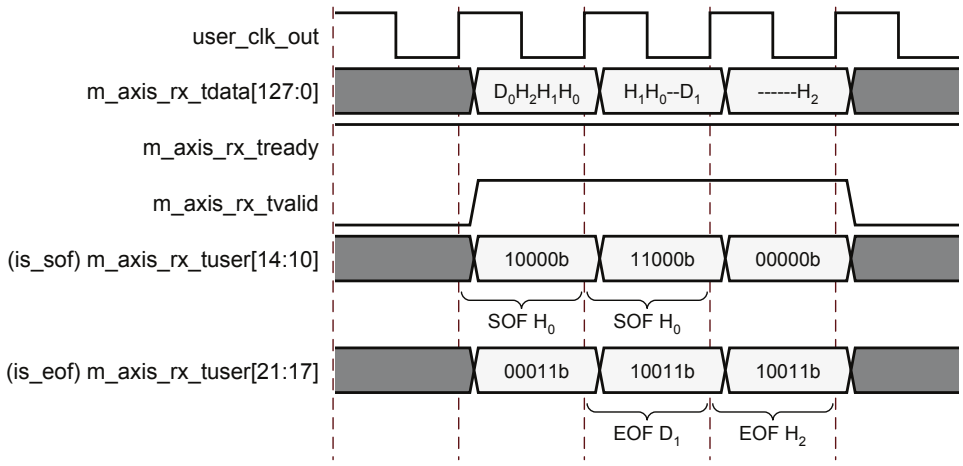


**Figura 4.5:** Transacciones con parte de dos paquetes en el mismo ciclo [18]

Además de las interfaces principales de AXI4-Stream, el core utiliza otras señales para comunicarse con el RMC:

**pl_*** Estado y configuración de la capa física. Permite conocer y cambiar el estado del enlace entre la tarjeta y el ordenador, el número de lanes activos y la velocidad del enlace o generación.

**cfg_*** Estas señales agrupan diversas funcionalidades:

- Acceso al espacio de configuración PCI de la tarjeta. Algunos registros tienen cables dedicados que permiten acceder a ellos sin direccionarlos, al resto se accede mediante su dirección en registros de 4 bytes.
- Gestión de los estados de energía. Permite conocer y controlar el estado de energía de la tarjeta.
- Gestión de interrupciones (`cfg_interrupt_*`). Se utilizan para enviar interrupciones y conocer que tipo de interrupciones pueden enviarse.
- Información de errores (`cfg_err_*`). Permiten a el RMC informar al core que se han recibido paquetes erróneos.

## 4.4. PCI Express Core para MEMSCALE

La interfaz del core PCIe es AMBA AXI4-Stream. Esta ha sido la razón principal por la que este protocolo se ha seleccionado como bus interno del sistema para la comunicación entre

todos los módulos del diseño. El ancho de la señal `tdata` depende de la configuración con la que se genere el core: 128 bits para generación 2 x8 y 64 para el resto (gen1 x1-x8 y gen2 x1-x4). Como las tarjetas de HiTech Global soportan gen2 x8, se ha decidido que el resto de los módulos utilicen la interfaz con 128 de ancho de datos. Cuando se utilice el diseño con un core de 64 bits (las placas SuperMicro no soportan PCIe gen2) se añadirá un bridge para traducir el AXI entre la interfaz de 64 bits que usará el core y la interfaz de 128 bits que utilizarán el resto de los módulos.

Se ha utilizado la versión 2.2 del core, que se ha generado para que tenga una configuración PCI lo más parecida posible a las tarjetas HTX:

- Identificadores de dispositivo (0x8) y fabricante (0x7)
- BAR0-1: direcciones de 64 bits y 32MB de tamaño, conectado al banco de registros.
- BAR 2-3: direcciones de 64 bits y 256MB de tamaño, en el diseño actual no se utiliza.
- BAR 4-5: direcciones de 64 bits y 256GB de tamaño, conectado a la SMFU.
- Los identificadores de dispositivo y fabricante de subsistema, a los que no se accede en el driver, se han utilizado para distinguir fácilmente entre los distintos ficheros de configuración de las tarjetas.

Ha sido necesario marcar como prefetchable el BAR 4-5 debido a su gran tamaño. Los puentes PCI-PCI de los dos ordenadores de pruebas solo permiten asignar rangos de direcciones de 64 bits a zonas marcadas como memoria prefetchable. Este problema no se da en los otros dos BARs porque se les asigna en el arranque del sistema un rango de direcciones menor que 0x100000000 (4GB). Al último BAR, la dirección base la asigna el driver, que también ha de configurar el puente PCI-PCI.

Aunque el objetivo es utilizar las tarjetas en modo generación 2 x8, debido a las limitaciones de los servidores Supermicro el core se ha configurado con gen1 x8. En esta configuración, la única frecuencia a la que puede funcionar la interfaz del core PCIe con el RMC para que se puedan transmitir los paquetes sin cuellos de botella es 250MHz.

Aunque las tarjetas HTX tenían habilitado el envío de interrupciones, los módulos que envían interrupciones no se utilizan para MEMSCALE. Por tanto las interrupciones se han deshabilitado en el core debido a que ninguno de los módulos que se ha incluido en el diseño las envía.

## 4.5. Bridge

El bridge es el módulo que convierte la interfaz de AXI4-Stream de 64 bits a 128 bits para poder utilizar el diseño con un core PCI Express que no sea generación 2 x8. Funciona fusionando dos ciclos de 64 bits procedentes del core PCIe en uno de 128 con destino al switch o generando dos ciclos de 64 por cada uno de 128 en sentido inverso. También debe de adaptar las señales de `tuser` y `tstrobe`, ya que la interfaz de 128 bits del core a el RMC utiliza los bits de `is_sof` y `is_eof` y la interfaz de 64 bits utiliza `tstrobe`.

## 4.6. Switch

El switch se encarga de interconectar los principales módulos del diseño. Actualmente solo hay conectados 3 módulos: el banco de registros, la Shared Memory Functional Unit (SMFU) y el core PCI Express.

El switch original de las tarjetas HTX permite la comunicación entre cualquiera de la módulos, no solo del core HyperTransport a las unidades funcionales. Aunque no se utilice en este diseño, para permitir este comportamiento en el switch PCIe, se ha establecido una interfaz simétrica para los puertos maestro y eslavo de las unidades funcionales. Se ha elegido la interfaz `m_axis_rx_*` con las señales `is_sof` y `is_eof` para poder recibir paquetes montados (ver figura 4.5), ya que con `s_axis_tx_*` no se pueden recibir partes de dos paquetes distintos en el mismo ciclo. A esta interfaz simétrica se le ha incrementado el tamaño del `tuser` de 22 a 32 bits para poder utilizar los nuevos bits para indicar el puerto de salida en los paquetes que entran en el switch. El puerto cero del bridge, al que está asignado el core PCIe, ha tenido que ser modificado para que su interfaz si sea la nativa del core, que es asimétrica y utiliza las señales `m_axis_rx_tuser[9:2]` para indicar a que unidad funcional van los paquetes. El puerto cero también ha sido modificado para que en la interfaz `s_axis_tx_*`, le marque el inicio y fin de los paquetes al core con la señal `tstrobe`, ya que el resto de los módulos utilizan `is_sof` y `is_eof`.

El switch se ha modificado para que su interfaz sea AXI4, este cambio ha supuesto una gran ventaja sobre el switch HTX ya que permite a los módulos esclavo pausar la recepción de información si se saturan poniendo a cero el `tready`. Este comportamiento no estaba soportado por el switch HTX, en el que los módulos debían de aceptar paquetes completos sin interrupciones.

El switch provoca un retardo de 3 ciclos en los datos que pasan por él, por lo que debe de tenerse en cuenta cuando se diseñan los módulos, ya que las señales del protocolo AXI4-Stream llegarán retardadas (principalmente el `tready`). Para solucionar este problema, se han puesto colas First In First Out (FIFO) en la entrada de los módulos conectados al switch para poder guardar los ciclos que llegan después de la desactivación del `tready`. Para evitar que lleguen ciclos duplicados cuando el `tready` se activa después de `tvalid`, `tvalid` siempre se pone a cero cuando se desactiva el `tready` y no vuelve a activarse hasta que no recibe un `tready` a uno.

Para enviar un paquete, el módulo emisor tiene que pedir el canal de salida activando el bit correspondiente de la señal `s_axis_tx_tuser[22:29]`, cuando switch esté libre abrirá el canal y se propagará el `tready` del destino, empezando la transacción AXI4-Stream cuando `tready` esté activo.

## 4.7. Shared Memory Functional Unit

La SMFU es el módulo principal de la tarjeta, su función es modificar y reenviar las peticiones y respuestas a memoria remota. El funcionamiento de la SMFU es el siguiente (en la explicación se referirá al nodo en el que se origina la operación como nodo local y al nodo en el que está la memoria física a la que se quiere acceder como nodo remoto):

1. El nodo local recibe una petición a memoria desde el bus PCIe.
2. Calcula la dirección física en el nodo remoto y la sustituye en el paquete.
3. Envía la petición al nodo remoto por la red.

4. La SMFU del nodo remoto recibe la petición de la red y la envía a la controladora de memoria del nodo por el bus PCIe.

5. Si la petición es una escritura, la operación termina. Si es una lectura, cuando la SMFU remota reciba la respuesta, la enviará al nodo que envió la petición por la red.

6. La SMFU local recibe la repuesta y la envía al dispositivo que la originó (normalmente un procesador).

Este módulo ha sido completamente rediseñado durante el proceso de adaptación a PCIe, la descripción completa de los cambios se puede encontrar en [12].

## 4.8. Banco de registros

Este módulo se encarga de configurar los parámetros y recoger estadísticas del resto de los módulos. En el diseño actual sólo está conectado a la SMFU, configurando valores como el identificador del nodo al que está conectado la tarjeta o la dirección base del BAR4 y guardando estadísticas como el número de paquetes de cada tipo que pasan por la SMFU.

El banco de registros tiene tres partes distinguibles:

1. Por un lado los registros en sí, con cada registro directamente conectado a la parte del diseño donde se utiliza.

2. Otra parte se encarga de hacer accesibles todos los registros con una interfaz única similar a la de una memoria, con una dirección única para cada registro del banco.

3. La última parte se encarga de ofrecer una interfaz PCI Express para poder acceder a los registros desde el driver.

De las partes 1 y 2 se ha podido reutilizar todo el código sin cambios. Pero la parte 3, que en el diseño anterior tenia una interfaz HyperTransport, ha tenido que ser reescrita por completo, el nuevo módulo se ha llamado PCIe_RF_wrapper.

La función del PCIe_RF_wrapper es convertir peticiones de escrituras de 8 bytes y de lecturas de 4 y 8 bytes en operaciones sobre la interfaz de la parte 2. Si llega una petición de otro tipo, se ignora si es un non-posted (paquetes que no requieren respuestas, como las escrituras) o se envía una respuesta con un código de error si el paquete es posted (paquetes que requieren respuesta, como las lecturas).

En la figura 4.6 se muestra la máquina de estados del PCIe_RF_wrapper. La parte más compleja es la de decodificación de los paquetes debido la forma en la que los paquetes son enviados por el core PCIe: aunque el ancho del bus AXI4-Stream es de 128 bits, es posible que en el primer ciclo de una transacción sólo se reciban los primeros 64 bits del paquete. Este comportamiento y la necesidad de aceptar paquetes con direcciones de 32 y 64 bits (las cabeceras tienen tamaños distintos) ha causado los múltiples estados para decodificar los paquetes.

El estado inicial es IDLE, en este estado se espera a que llegue la cabecera de un paquete por el bus AXI (es fácilmente reconocible debido a que se activa la señal `if_sof[4]`). Cuando llega, se comprueba si es de un tipo y tamaño permitidos. Si el tipo del paquete es desconocido o es una escritura de tamaño diferente a 8 bytes, se ignora el paquete. Si es una lectura con un tamaño

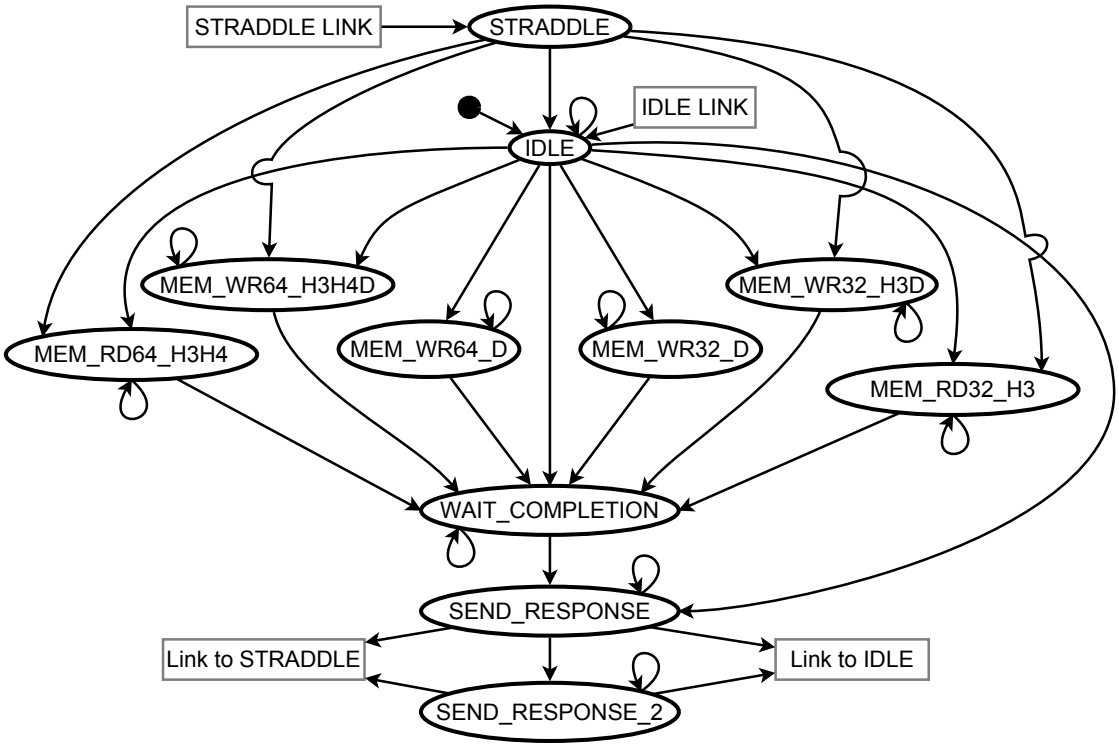


**Figura 4.6:** Máquina de estados del PCIe_RF_wrapper

no soportado o cero, se va al estado SEND_RESPONSE para enviar una respuesta sin datos con el estado de finalización a *Petición no soportada* o *Completada con éxito* si el tamaño es cero. Si la petición es una lectura compatible y se he recibido completa, se inicia una lectura en el banco de registros de la dirección pedida y se va al estado WAIT_COMPLETION. Si es una lectura y sólo se dispone de los primeros 8 bytes, se va al estado MEM_RD32_H3_STATE si la dirección del paquete es de 32 bits o a MEM_RD64_H3H4_STATE si la dirección es de 64 bits. En estos estados se leerá la dirección del bus AXI cuando esté disponible (normalmente en el siguiente ciclo), se iniciará una lectura en el banco de registros y se va al estado WAIT_COMPLETION. Las escrituras siempre ocupan más de un ciclo ya que en 128 bits no cabe una cabecera y 8 bytes de datos. Cuando se recibe una escritura con la cabecera completa, se va al estado MEM_WR32_D o MEM_WR64_D dependiendo del tamaño de la dirección. Si no se recibe la cabecera completa se va al estado MEM_WR32_H3D o MEM_WR64_H3H4D. En estos últimos 4 estados se inicia una escritura con la dirección y datos recibidos y se va al estado WAIT_COMPLETION. El estado WAIT_COMPLETION se encarga de esperar a que termine la operación sobre el banco de registros, guarda el estado de finalización (puede terminar con éxito o con una señal de error) y va al estado de SEND_RESPONSE. SEND_RESPONSE envía las respuestas cuando es necesario con los datos y estado de finalización correspondientes. Si la respuesta contiene 8 bytes de datos es necesario otro ciclo de datos para terminar de enviarlos (la cabecera de las respuestas ocupa 96 bits), por lo que se va al estado SEND_RESPONSE_2. En este último estado se envían los últimos 32 bits de datos. Cuando los estados de SEND_RESPONSE y SEND_RESPONSE_2 terminan de enviar un paquete, pueden ir a IDLE o a STRADDLED. STRADDLED es un estado

para poder procesar una cabecera si llega junto al final de otro paquete. Cuando esto ocurre, se guardan los 4 bytes de la cabecera en un registro y se decodifican en el estado STRADDLED cuando se ha terminado la transacción anterior. El estado STRADDLED decodifica la cabecera parcial de la misma forma que lo hace IDLE.

Los datos en los Transaction Layer Packet (TLP) de PCI Express se representan en little endian (byte menos significativo primero), a diferencia de HyperTransport donde datos se transfieren el big endian (byte más significativo primero). Como el banco de registros guarda la información en big endian, el nuevo módulo tiene que convertir entre las dos ordenaciones en todas la operaciones reordenando los bytes.

En la figura 4.7 se muestra una escritura y una lectura sobre el banco de registros. Como puede observarse, el módulos recibe y decodifica las peticiones, desactiva `m_axis_rx_tready` mientras realiza la operación, si es necesario envía una respuesta y vuelve a activar `m_axis_rx_tready`. La información que se lee del banco de registros es inválida debido a que como solo se ha simulado en banco de registros sin la SMFU, los registros no están conectados al resto de módulos y no pueden acceder los valores correctos.

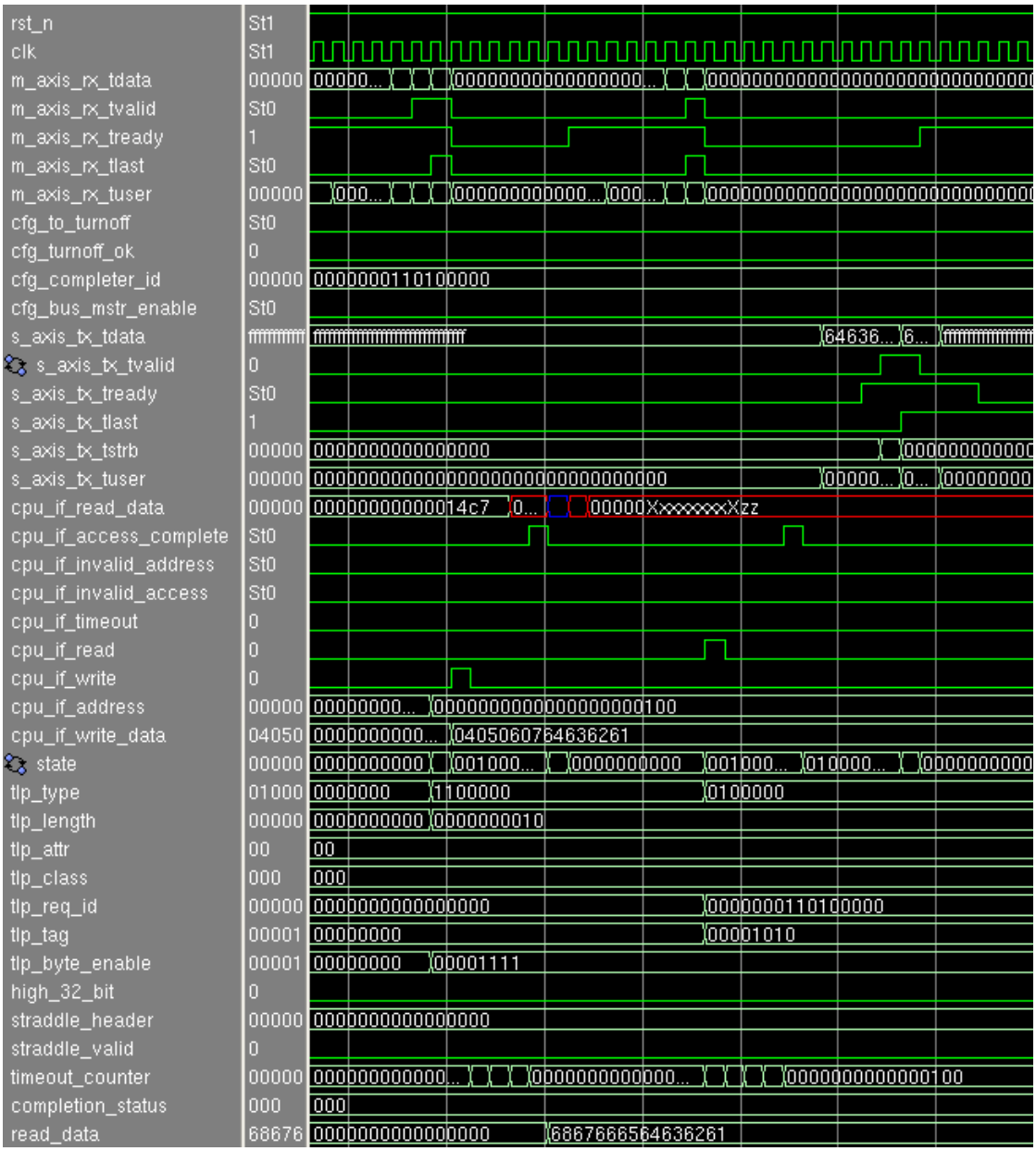


**Figura 4.7:** Simulación de operaciones sobre el banco de registros

Capítulo 5

# Adaptación del Software

En este capítulo se explicarán los cambios que se han realizado al software y a los drivers existentes para HyperTransport para adaptarlos al nuevo diseño en PCI Express. Los cambios realizados no han sido significativos debido a que uno de los objetivos en el diseño hardware ha sido mantener la compatibilidad con los drivers, pero sí han sido complejos de identificar debido a la complejidad del propio driver de HyperTransport y a la necesidad de configuración de los chipsets combinado con la falta de documentación de las placas base de AMD.

## 5.1. Introducción

El espacio de configuración PCI (figura 5.1) es un conjunto de registros que utilizan algunos dispositivos para ser detectados en el arranque del sistema y configurase para poder funcionar sin interferir con el resto de dispositivos (asignación de puertos de entrada/salida, rangos de direcciones y lineas de interrupción entre otros). Es utilizado por los dispositivos PCI convencionales, PCI-X y PCI Express. HyperTransport utiliza una variante de este espacio de configuración, pero los primeros 64 bytes son comunes.

Debido a la gran similitud entre los espacios de configuración, en Linux se utilizan las funciones de gestión de dispositivos PCI para gestionar los dispositivos HyperTransport. Esta es la principal razón para que los cambios en los drivers no sean significativos.

La principal diferencia entre las dos tarjetas desde el punto de vista del software es la presencia del chipset en las tarjetas PCIe, que no es necesario en las tarjetas HTX. El chipset es un componente de las placas base que permite al procesador comunicarse con los dispositivos conectados al sistema.

En la figura 5.2 se muestra como una tarjeta HTX se conecta directamente al procesador sin necesidad de ningún otro elemento de interconexión. Esto es posible porque los procesadores utilizados (AMD de la familia 10h) implementan el protocolo HyperTransport de forma nativa.

La figura 5.3 ilustra el uso del chipset para que el procesador pueda comunicarse con una tarjeta PCIe. A diferencia de HyperTransport, los procesadores utilizados no implementan PCI Express de forma nativa. Por eso necesitan el chipset, que utiliza HyperTransport para comunicarse con el procesador y también implementa PCI Express. La presencia del chipset, no modifica el código de los módulos del driver, ya que este utiliza las funciones proporcionadas por el kernel que abstraen al driver de estos detalles. Pero deberá de configurarse el chipset antes de cargar los módulos en el kernel

Los BARs (Base Address Register) representan las diferentes funciones de un dispositivo.

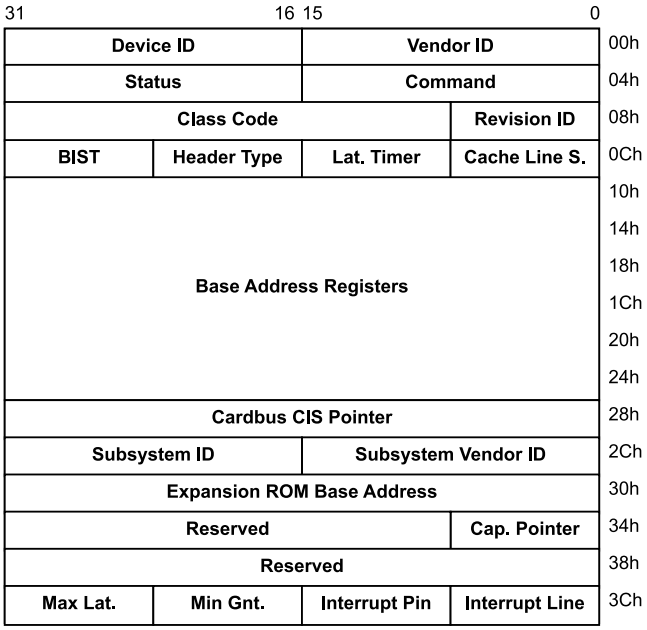


**Figura 5.1:** Espacio de configuración PCI

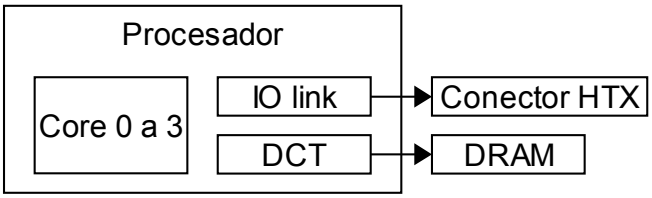


**Figura 5.2:** Procesador con tarjeta HTX (DCT: DRAM controller)

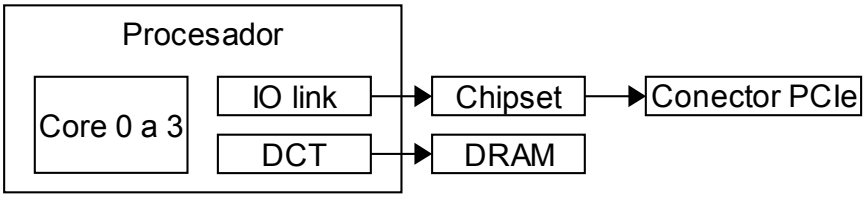


**Figura 5.3:** Procesador con tarjeta PCIe (DCT: DRAM controller)

Tienen un tamaño que indica la memoria disponible en cada función. En el arranque del sistema se debe asignar un rango de direcciones único a cada BAR del sistema. Cada dispositivo PCI tiene 6 registros BAR de 32 bits que pueden combinarse para crear BARs de 64 bits. En las tarjetas utilizadas en este proyecto el BAR0-1(64 bits) tiene un tamaño de 32MB y está asignado al banco de registros, el BAR4- 5 de 256GB está asignado a la SMFU. Tener un BAR tan grande (256GB) crea problemas debido a que la BIOS no le puede asignar un rango de direcciones tan grande. Para poder utilizar el BAR4, es necesario asignar manualmente una dirección base en los registros del BAR y configurar la CPU y el chipset para que envíen las peticiones de memoria al rango seleccionado de forma que lleguen a la tarjeta.

## 5.2. Opciones del kernel

Para que las tarjetas puedan utilizar parte de la memoria del sistema de forma exclusiva se le debe de pasar al kernel un parámetro al arranque del sistema indicándole la memoria que puede utilizar, el resto será la que podrá utilizar la tarjeta. En los servidores Supermicro no ha sido necesario modificar este parámetro porque ya estaba configurado para las tarjetas HTX con 8GB para el kernel y 8GB para MEMSCALE.

En el PC de pruebas si se ha tenido que configurar el argumento, repartiendo 4GB para cada parte. Para cambiar las opciones de arranque de Linux es necesario modificar la configuración del gestor de arranque, en nuestro caso el GRUB. En el fichero de configuración del GRUB (`/boot/grub/menu.lst`) se ha copiado la entrada por defecto y se ha añadido `mem=5632M` al final de la linea del kernel. El valor es mayor que 4GB porque este parámetro también contabiliza los espacios de direcciones asignados a los dispositivos, no solo la memoria RAM.

## 5.3. Módulos del kernel

Para utilizar la SMFU es necesario cargar tres módulos del driver en el kernel de Linux:

**extoll_rf** crea un directorio de ficheros virtuales (sysfs) que permite el acceso al banco de registros de la tarjeta. `extoll_rf` no puede utilizarse hasta que no se ha cargado el modulo `extoll`, pero debe de cargarse antes que `extoll` ya que éste depende de `extoll_rf`.

**extoll** este módulo registra la tarjeta y proporciona funciones de ayuda a los módulos que utilizan las unidades funcionales, en nuestro caso sólo la SMFU. También envía a `extoll_rf` la dirección base del BAR0 para que pueda acceder al banco de registros.

**smfu** este módulo genera un fichero de dispositivo en `/dev/smfu1` que permiten acceder a la SMFU desde una aplicación de usuario. Este fichero es similar a `/dev/mem`, lo que permite mapear el fichero en memoria mediante `mmap()` para poder acceder a la SMFU sin que se ejecute ningún driver.

Ha sido necesario modificar el fichero `extolldrv.c` que forma parte del módulo `extoll`:

- Comentar el código de gestión de la interrupciones, ya que éstas no se han implementado en el la tarjeta PCIe.
- Añadir una llamada a la función `pci_set_master(devp)` para habilitar el envío de peticiones a memoria local desde la tarjeta.

## 5.4. Scripts de configuración

Para utilizar las tarjetas es necesario ejecutar varios shell scripts para cargar los drivers, configurar el banco de registros de la tarjeta y varios registros del procesador y el chipset.

En las tarjetas HTX, debido al gran tamaño del BAR4, la BIOS no puede asignar una dirección base a esta región de memoria [10]. Para poder utilizar el BAR hay que configurar manualmente registros del procesador y del espacio de configuración PCI de la tarjeta HTX. Para adaptar esta configuración a las tarjetas PCIe, se han cambiado los valores de los registros modificados del procesador y la tarjeta y se han modificado otros del chipset.

Antes de acceder a la tarjeta es necesario cambiar el valor por defecto del BAR4 (SMFU). Esto es necesario debido al funcionamiento del Core PCIe. La BIOS asigna una dirección base a los dos primeros BARs, dejando la dirección base del BAR4 a 0. Cuando un paquete llega al BAR0 (Banco de registros), al que se le asigna una direcciones base menor de 4GB, como el BAR4 tiene un gran tamaño (256GB), los dos espacios de direcciones se solapan y el core activa las señales de BAR destino del BAR0 y del BAR4. Este comportamiento crea paquetes no deseados cuando la SMFU recibe peticiones dirigidas al Banco de registros.

Para cambiar el BAR4 al valor deseado(0x8000000000) hay que modificar los registros 0x20 y 0x24 a los valores 0x0000000C y 0x00000080 respectivamente. El registro 0x20 se ha modificado respecto a las tarjetas HTX para activar el bit de prefetch (bit 3).

Comandos ejecutados en el PC para establecer la dirección base del BAR4:

```
setpci -s 02:00.0 20.l=0000000c
setpci -s 02:00.0 24.l=00000080
```

Ha sido necesario modificar una pareja de registros en el procesador Memory Mapped IO Base/Limit Address Registers (Direcciones de los registros: Base:F1x[B8, B0, A8, A0, 98, 90, 88, 80], Limit:F1x[ BC, B4, AC, A4, 9C, 94, 8C, 84]) [4] [6]. Estos registros contienen rangos de direcciones de periféricos y el IO links (ver figuras 5.2 y 5.3) por el que deben de enviarse las peticiones a esas direcciones. Se ha configurado el rango del BAR4 (0x8000000000-BFFFFFFFFF) con destino al chipset marcándolo como lectura/escritura. Estos registros de configuración del procesador pueden modificarse mediante una interfaz PCI.

En el PC de pruebas, la dirección PCI de los registros es 00:18.1, se ha utilizado el par 0x98, 0x9C que se encontraba libre y los valores escritos han sido los siguientes:

- MMIO Base Addres register (98) = 0x8000003
  - MMIO Base Address = 0x8000000000
  - Write enable and read enable set to 1
  - `setpci -s 00:18.1 98.l=8000003`
- MMIO Limit Addres register = 0xBFFFFF00
  - MMIO Limit Address (9C) = 0xBFFFFFFFFF
  - Destination Node = 0
  - Destination Link = 0
  - `setpci -s 00:18.1 9C.l=BFFFFF00`

En los servidores Supermicro se han configurado los 4 procesadores, que tiene las siguientes direcciones PCI: 00:18.1, 00:19.1, 00:1A.1 y 00:1B.1. Se ha utilizado el par libre 0xA8, 0xAC y los registros se han configurado a:

- MMIO Base Addres register (A8) = 0x8000003
  - MMIO Base Address = 0x8000000000
  - Write enable and read enable set to 1
  - `setpci -s 00:18.1 98.l=8000003`
  - `setpci -s 00:19.1 98.l=8000003`
  - `setpci -s 00:1A.1 98.l=8000003`
  - `setpci -s 00:1B.1 98.l=8000003`
- MMIO Limit Addres register = 0xBFFFFF20
  - MMIO Limit Address (AC) = 0xBFFFFFFFFF
  - Destination Node = 0
  - Destination Link = 2
  - `setpci -s 00:18.1 9C.l=BFFFFF20`
  - `setpci -s 00:19.1 9C.l=BFFFFF20`
  - `setpci -s 00:1A.1 9C.l=BFFFFF20`
  - `setpci -s 00:1B.1 9C.l=BFFFFF20`

El enlace 2 del nodo 0 es donde está conectado el chipset MCP55Pro.

Para las tarjetas PCIe ha sido necesario modificar algunos registros en el chipset [5]. Esto no era necesario en las tarjetas HTX porque los procesadores utilizados soportan nativamente HyperTransport. En el chipset se ha modificado el bridge PCI-PCI asignándole el rango del BAR4 al puerto PCIe correspondiente. Se ha asignado el rango de direcciones a los registros de memoria prefetch porque son los únicos registros que permiten mapear rangos de direcciones de 64 bits. Se han modificado tres registros: el primero (0x24) contiene la parte baja de las dos direcciones (bits 31:20) y si las direcciones son de 32 ó 64 bits, en los otros dos (0x28 y 0x2C) se configura la parte alta (bits 63:32) de cada una las direcciones.

En el PC la dirección del bridge PCI-PCI es 00:03.0 y los valores de los registros:

- PCIe prefetchable memory base limit (0x24) = 0xFFF10001
  - Las dos direcciones son de 64 bit
  - `setpci -s 00:03.0 24.l=fff10001`
- PCIe prefetchable memory base upper (0x28) = 0x00000080
  - Base Address = 0x8000000000
  - `setpci -s 00:03.0 28.l=00000080`

- PCIe prefetchable memory limit upper (0x2C) = 0x000000BF
  - Limit Address = 0xBFFFFFFFFF
  - `setpci -s 00:03.0 2c.l=000000FA`

En el servidor, la dirección PCI del bridge es 00:0F.0 y los valores de los registros son los mismos.

```
setpci -s 00:0F.0 24.l=fff10001
setpci -s 00:0F.0 28.l=00000080
setpci -s 00:0F.0 2c.l=000000FA
```

Después de configurar el acceso al BAR4 y cargar los módulos del kernel, es necesario configurar varios valores del banco de registros desde los ficheros creados por el módulo extoll_rf:

Para que el script de configuración sea independiente del ordenador donde se ejecuta, la dirección PCI de la tarjeta se calcula partir de los identificadores de dispositivo(0x0008) y fabricante(0x0007):

- `bdf=`lspci -d 0007:0008 | awk '{print $1}'``
- `SYSPATH=/sys/bus/pci/devices/0000:$bdf/extoll`

Una vez se conoce el directorio donde se encuentran los ficheros del banco de registros (`SYSPATH`) se configuran varios registros de la tarjeta:

**smfu_general** dirección base del BAR4

- `echo 0x8000000000 > $SYSPATH/smfu_general`

**smfu_balt** direcciones donde empieza la zona de memoria compartida por la SMFU en cada nodo

- `echo 0 0x0200000000 > $SYSPATH/smfu_balt`
- `echo 1 0x0200000000 > $SYSPATH/smfu_balt`

**smfu_mask_calc** mascara que aplica la SMFU a las direcciones para calcular el nodo donde se encuentra la memoria a la que va dirigida la petición.

- `echo    0x3F00000000 > $SYSPATH/smfu_mask_calc`

**ids** configura el identificador del nodo

- `echo 0x000000000 > $SYSPATH/ids`

## 5.5. Aplicaciones de prueba

Se han probado varias aplicaciones escritas para las tarjetas HTX y utilizadas anteriormente como banco de pruebas de los nodos del supercomputador. Todas las aplicaciones han funcionado sin necesidad de ser modificadas o recompiladas. Esto ha sido posible gracias no sólo a la compatibilidad a nivel de diseño HW, sino también a que las aplicaciones acceden a las tarjetas por el fichero de dispositivo /dev/smful, interfaz que no ha sido modificada.

Con todo esto se ha conseguido uno de los objetivos principales del proyecto relacionado a la compatibilidad entre la versión de HTX y la de PCIe.

Capítulo 6

# Modelado del supercomputador

En este capítulo se describirá el prototipo basado en HyperTransport, se explicarán las dos aproximaciones que se han utilizado para modelarlo y se hará una comparativa entre ellas.

La motivación para crear un módulo que modele el supercomputador ha sido que para realizar comparativas entre el nuevo diseño basado en PCIe y las tarjetas HyperTransport, es necesario un módulo de red para conectar varias tarjetas. El problema es que las tarjetas HyperTransport están basadas en FPGAs Xilinx Virtex 4 y las PCIe en FPGAs Xilinx Virtex 6 y el módulo de red utiliza los serializadores/deserializadoes integrados en la FPGA y su interfaz es distinta en las dos generaciones de FPGAs. En el Computer Architecture Group de la Universidad de Heidelberg ya están trabajando en portar el módulo de red a las Virtex 6. Mientras no esté disponible la nueva versión del módulo, se ha decidido crear otro módulo que tenga la misma interfaz que el módulo de red, pero que conecte los puertos de entrada y salida de la SMFU de una sola tarjeta.

Otra ventaja muy importante de tener el modelo es que se podrán probar los nuevos diseños de una forma rápida y cómoda sin tener que utilizar varios nodos, sobre todo teniendo en cuenta que actualmente para este PFC solo hay dos tarjetas PCIe de prototipado disponibles.

La interfaz de los módulos es AXI4-Stream, pero se ha intentado minimizar el código que depende del protocolo para que sea fácil portar los módulos a otra interfaz.

## 6.1. Modelado del supercomputador

El supercomputador que se quiere modelar es el prototipo que se ha montado en el GAP. Este cluster dispone de 64 nodos con la misma configuración que los servidores SuperMicro H8QM8E utilizados en este proyecto. Además, cada nodo está equipados con una tarjeta HTX.

Cada tarjeta HTX dispone de una FPGA Virtex 4 XC4VFX100 donde se implementa la lógica de enrutado y 6 conectores SFP (Small Form-factor Pluggable) que permiten una gran flexibilidad para configurar la topología de la red: malla 2D, malla 3D, hipercubo, etc. El cableado es por fibra óptica.

Para abordar el diseño del modelo de un sistema tan complejo como el supercomputador, se ha optado por la solución más sencilla, El modelo basado en retardos de transacciones. El módulo del modelo deberá generar retardos en los paquetes según el número de enlaces que debería atravesar en el supercomputador.

Haciendo pruebas con una malla 2D 4x4, en [13] se ha calculado un retardo de las peticiones de 1.9µs entre nodos conectados directamente, cada enlace adicional añade 600ns de retardo (300 ns en la petición y 300ns en la respuesta). Para dar una solución versátil a la tarea del modelado

del supercomputador, se ha optado por diseñar un módulo de red, que es la base del modelo del supercomputador, que introduce retardos correspondientes al número de enlaces que la petición debería de atravesar.

Para modelar la red del supercomputador se ha creado un módulo que sustituye el Network Port (NP) existente en las tarjetas HTX e introduce retardos en los paquetes de 600ns por enlace. La razón para introducir todo el retardo en las peticiones y no repartirlo entre la petición y la respuesta es que como se quiere mantener la misma interfaz que el Network Port, cuando las respuestas entran en el módulo de red no se conoce el nodo que envía la respuesta, solo el destino, por lo que no puede calcularse el retardo que le correspondería. Introducir todo el retardo en las peticiones (las de lectura y las de escritura) no afecta al funcionamiento del modelo. Las lecturas y las escrituras llegarán en el mismo orden en el que se generan ya que cada nodo, desde el punto de vista del flujo de datos, se comporta como una FIFO.

El modelo se ha planteado lo más genérico posible. Los parámetros de la red, como la topología de la red, el retardo de cada enlace y el número de nodos, pueden ser modificados fácilmente. El retardo se configura cambiando un entero que representa el retardo de una petición en nanosegundos (en el modelo actual 600). La topología se configura cambiando una matriz que contiene el número de enlaces que debe atravesar un paquete para ir del nodo en el que está la tarjeta al resto de los nodos modelados. Si se cambia la topología puede que también sea necesario modificar el número máximo de enlaces que un paquete podría atravesar. Estos valores se han puesto en un fichero de cabecera (network_model.h) para facilitar su localización. A continuación se muestra como ejemplo el contenido de uno de estos ficheros para el modelo de un supercomputador de 16 nodos conectados en una red de tipo MESH de 4x4 con un retardo de salto de 300 ns

```
parameter DELAY_HOP = (2*300); //ns Double of a single link delay
parameter T_CYCLE = 8; //ns (1/125MHz)
parameter T_PROC_NET_MODEL = 3; //Extra cycles used by the model to
                                //process the the packets. Substracted
                                //from the total timeout to compensate
parameter COUNTER_WIDTH = 16;
parameter MAX_HOPS = (6+1); //One extra node is needed for the packets
                            //without delay
parameter HOPS = {6'd6,6'd5,6'd4,6'd3,
                  6'd5,6'd4,6'd3,6'd2,
                  6'd4,6'd3,6'd2,6'd1,
                  6'd5,6'd2,6'd1,6'd0};
```

Para que el modelo pueda generarse automáticamente con las distintas opciones, ha sido necesario diseñarlo con un módulo básico que se genere múltiples veces dependiendo del tamaño de la red a modelar. Este nodo básico se explicará en la siguiente sección.

Debido a que el modelo del supercomputador se ha basado en los retardos de los saltos en la red, a lo largo del documento se hará referencia a este modelo también como modelo de red.

## 6.2. Modelado de un nodo

Como ya se ha mencionado anteriormente, la base del modelo del supercomputador es el modelo de un nodo. La característica común de las dos versiones del modelo es la unidad básica que se encarga de introducir los retardos a la que llamaremos Nodo de Retardo (NR) . En la figura 6.1 se muestra el diagrama de bloques de un NR. Se ha utilizado una cola FIFO con el ancho de datos suficiente para almacenar los datos de cada ciclo del paquete y una marca de tiempo que indica cuando puede salir el paquete de la cola. La marca de tiempo se calcula a partir de un contador de 16 bits que se incrementa en cada ciclo de reloj. A este contador se le suman los ciclos equivalentes al retardo que se quiere incluir. A la salida de las FIFOs la marca de tiempo se compara con el contador, cuando el contador es mayor o igual a la marca, se saca el dato de la FIFO.

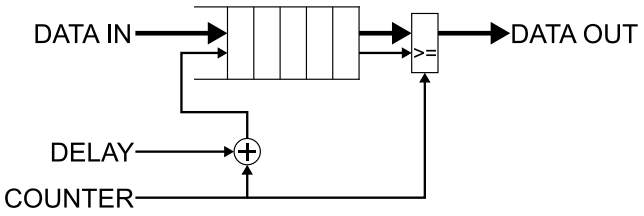


**Figura 6.1:** Esquema de un Nodo de Retardo

El contador se desbordará continuamente. Para que el retardo de los paquetes no cambie cuando esto ocurre, el desbordamiento se ha tenido en cuenta para el cálculo de la marca de tiempo que condiciona la salida de los datos de la FIFO.

La interfaz del módulo es la misma que la de la FIFO para hacerlo más portable y simplificar su diseño.

Las FIFOs utilizadas han sido generadas con el IP Core Generator de Xilinx [16]. Estas FIFOs son muy configurables, lo que permite adaptar la profundidad de las colas a las más adecuadas en cada caso. Las FIFOs tienen un retardo entre la escritura y la lectura de 5 ciclos, aunque esto no presenta ningún problema ya que todos los retardos introducidos han sido mayores.

En el diseño actual funcionando a 125MHz se pueden modelar retardos desde 40ns hasta 262µs que cubren ampliamente el rango de retardos existentes en el supercomputador. Para modelar retardos menores seria necesario cambiar las FIFOs por unas con menor retardo o aumentar la frecuencia de funcionamiento del modelo. Para retardados mayores bastaría con aumentar el parámetro COUNTER_WIDTH del fichero network_model.h. Cuando se modelan retardos grandes, hay que tener en cuenta que las FIFOs deben de almacenar los datos durante todo el tiempo que dura el retardo. Si éstas se llenan, el modelo no funciona correctamente por lo que debe de ajustarse el tamaño de las FIFOs dependiendo del retardo modelado y del tráfico con el que se vayan a utilizar los Nodos de Retardo.

Con una FIFO de 64 posiciones, un Nodo de retardo ocupa 263 Slice Registers, 93 Slice LUTs (LookUp Tables) y 3 bloques RAM.

La aproximación al modelado del supercomputador ha tenido dos vertientes: una paralela, en la que los NRs (Nodos Retardo) necesarios son conectados en paralelo y la otra serie, en la que todos los NRs son concatenados. Cada una de las versiones se describirá en detalle en las siguientes secciones.

## 6.3. Modelo paralelo

La versión paralela del modelo del supercomputador consiste en conectar en paralelo tantos NRs como nodos se pretenden modelar. Además consta de un multiplexor en la entrada que distribuye los paquetes entrantes al nodo correspondiente y un demultiplexor que hace el trabajo inverso.

El problema principal de la versión paralela del modelo es que con un número moderado de nodos (32), el multiplexor de la entrada ya no puede funcionar a la frecuencia requerida de 125MHz debido a su gran tamaño, que repercute en la profundidad de la lógica necesaria.

Para reducir el tamaño del multiplexor se ha decidido reducir el número de NRs, generando uno por cada número de enlaces que debería de atravesar un paquete: un NR para todos los paquetes en los que el nodo destino está directamente conectado al nodo origen, otro para los paquetes cuyo destino esté a dos enlaces de distancia, etc. Así, serán necesarios tantos NRs como el diámetro de la red, consiguiendo reducir drásticamente la cantidad de nodos necesarios en el modelo.

Por ejemplo, para el caso anterior de una red de 16 nodos MESH de 4x4 serían necesarios únicamente 6 módulos NR.

En el multiplexor se ha introducido un árbitro Round-Robin para evitar que a algunos NRs se les llenen las FIFOs si no se lee de los mismos. El multiplexor y el demultiplexor también cumplen la función de cambiar del AXI4-Stream a la interfaz de las FIFO del los NRs.

Para que los retardos sean los más exactos posible, se han compensado los tres ciclos de retardo que introducen el multiplexor y el demultiplexor restándolos a los retardos de los NRs.

Aunque en un uso real de las tarjetas no se accederá a la memoria local del propio sistema a través de la tarjeta PCIe (es más rápido acceder directamente a la memoria), es posible que se generen estas peticiones. Por ello, se ha creado un nodo extra con retardo nulo como se puede ver en la figura 6.2.

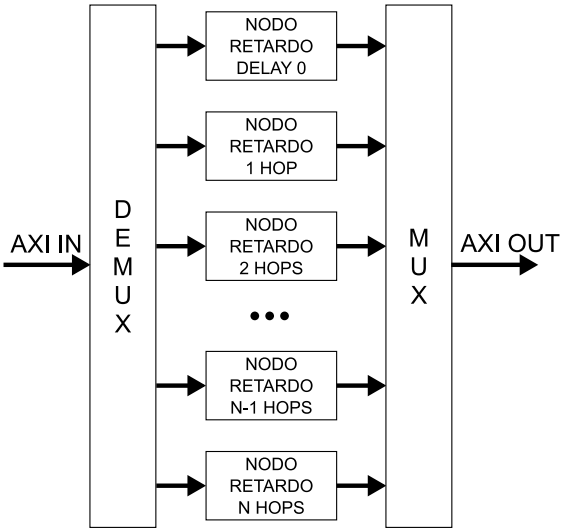


**Figura 6.2:** Modelo paralelo

## 6.4. Modelo serie

Aunque el modelo paralelo, descrito en el punto anterior, cubre plenamente los requisitos de modelado de un supercomputador, el límite de la frecuencia de multiplexor y el cuello de botella que puede suponer el mismo si el número de NRs es elevado, han dado lugar a la búsqueda de una alternativa serie. En la figura 6.3 se puede ver el esquema del modelo serie del supercomputador donde los NRs están conectados uno después del otro.

La característica principal de este modelo es que tiene una única entrada y una única salida para todos los paquetes. Además, en este modelo todos los NRs tendrán un retardo igual al de un enlace y los paquetes pasarán por tantos NRs como enlaces pasaría en la red modelada. Así, si un paquete en la red real pasa por 5 enlaces, en el modelo pasará por 5 NRs en los cuales esperara el tiempo equivalente al retardo de un salto en la red.

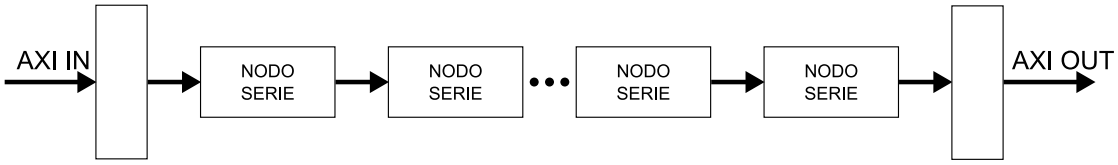


**Figura 6.3:** Modelo serie

Además, a los nodos en serie se ha añadido otra FIFO sin retardo en paralelo a cada NR como se ve en la figura 6.4. En cada nodo, los paquetes sólo pasarán por una de las dos FIFOs: mientras no hayan completado el retardo, pasarán por el NR y cuando lo hayan completado seguirán por las FIFOs normales. Todos los paquetes atravesaran todos los nodos. Según su retardo pasarán por más nodos de retardo o FIFOs normales. Para reducir el retardo introducido por las nuevas FIFOs estas no han sido generadas con el Core Generator, que introducen un retardo de 5 ciclos. Se han utilizado otras FIFOs basadas en Shift Register LUT (SLR) con una profundidad fija de 16 posiciones y un ciclo de retardo.

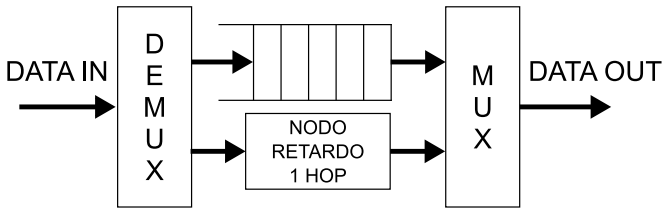


**Figura 6.4:** Nodo del modelo serie

Para compensar al retardo extra que sufren los paquetes al pasar por más FIFOs después de completar su retardo, al valor del retardo del primer NR se le ha restado el retardo que introducen las FIFOs. Se ha restado el doble del retardo que introducen las FIFOs para compensar también el retardo que sufren las respuestas.

Un inconveniente de este modelo es que a los paquetes que van de un nodo al mismo nodo, a los que no se debería introducir ningún retardo, al pasar por todos los nodos en serie sufren un retardo considerable no deseado. De todas formas este caso, tal como ya se ha mencionado

anteriormente, no es común en el funcionamiento real del supercomputador, ya que los paquetes que van de un nodo a sí mismo no pasan por la tarjeta PCIe.

Con este modelo no ha sido posible generar un fichero de configuración (bitfile) del diseño completo con más de 6 nodos debido a que el proceso de síntesis falla si se intenta con más nodos, aunque el modelo separado del resto del diseño si que sintetiza correctamente.

## 6.5. Comparativa de los modelos

Para comparar las prestaciones de los dos modelos, se han implementado separados del resto del diseño. Se ha variado el diámetro de la red entre 1 y 10 para analizar como cambia la utilización de los recursos de la FPGA y la frecuencia máxima de funcionamiento.

En el gráfico de la figura 6.5 podemos observar que el modelo serie utiliza más slices (unidad básica de la lógica interna de la FPGA) que el paralelo y la diferencia se incrementa con el aumento del diámetro de la red. Esto de debe a que los nodos serie son más complejos que los paralelos, ya que tiene dos FIFOs en vez de una y los multiplexores/demultiplexores para gestionarlas. El modelo paralelo tiene un multiplexor y un demultiplexor que aumentan de tamaño con el número de nodos, pero la cantidad total de lógica que requieren es menor que la necesaria para los nodos serie, ya que el grado de optimización de los recursos que puede alcanzar el sintetizador es mayor.

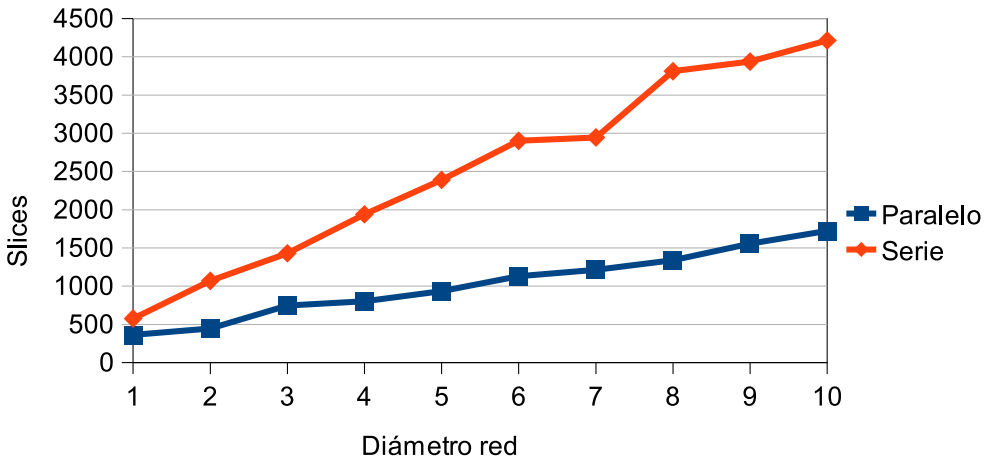


**Figura 6.5:** Slices ocupados

El gráfico de la figura 6.6 muestra que el modelo paralelo utiliza un 67 % más de bloques RAM que el serie. Esto es debido a que las FIFOs del modelo paralelo han de ser más grandes para poder almacenar la información que pasaría por varios enlaces, no como en el modelo serie donde cada FIFO solo ha de almacenar los datos de un enlace. Para simplificar el diseño y hacerlo parametrizable se han utilizado FIFOs del mismo tamaño para todos los nodos paralelos, el tamaño necesario para el nodo con más retardo. Si fuera necesario, se podrían generar FIFOs para cada nodo, pero el módulo serie dejaría de ser parametrizable.

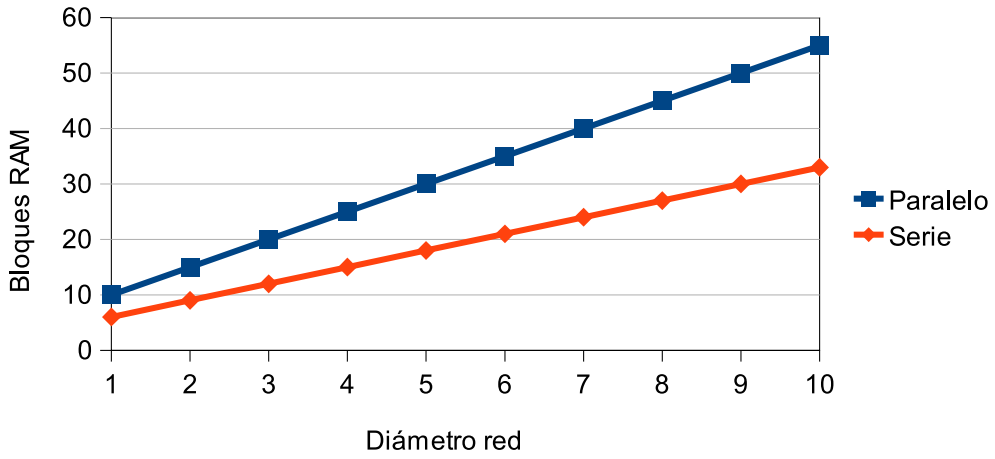


**Figura 6.6:** Bloques RAM ocupados

En la figura 6.7 se muestra una gráfica con las frecuencias máximas calculadas por los procesos de síntesis y Place and Route (posicionamiento y enrutado) del ISE. Las frecuencias máximas de la síntesis son las esperadas: la frecuencia se mantiene constante en el modelo serie y disminuye de forma inversamente proporcional al diámetro de la red en el modelo paralelo. Esto es debido a que en el modelo paralelo los multiplexores aumentan de complejidad con el tamaño de la red y hacen disminuir la frecuencia. En el modelo paralelo se aumenta el número de nodos pero no la complejidad de los mismos, por lo que la frecuencia continua siendo la misma. La similitud entre las frecuencias máximas después del Place and Route entre los dos modelos puede ser debida a que la mayor cantidad de lógica en el modelo serie no puede ser organizada en la FPGA para que funcione a mayor frecuencia.

En la figura 6.8 se han comparado los retardos introducidos por los dos modelos generándolos con un diámetro de la red de 6 enlaces y haciendo lecturas a nodos a distintas distancias. Los resultados han sido los esperados, 600ns de retardo por enlace atravesado, en todos los casos menos en el modelo serie en accesos al mismo nodo, en este caso, se introduce un retardo extra de 360ns. Este retraso se introduce cuando la petición y la respuesta de una lectura pasan por todos los nodos serie, aunque no pasa por ningún nodo de retardo, cada nodo genera 3 ciclos de retardo en cada paquete procesado, que se acumula y produce este retardo significativo.

Los retardos totales de los paquetes son mayores en los modelos que en el supercomputador: 3.1µs frente a 1.9µs en nodos directamente conectados. Como se explicará en la sección 7.3, este retardo constante de 1.2µs no se introduce por el modelo sino por el chipset de PCI Express, por lo que no hay que tener en cuenta este retardo base para valorar la exactitud de los modelos (serie y paralelo) al reproducir los retardos de la red.

Comparando los modelos con el funcionamiento real del supercomputador, el modelo paralelo reproduce fielmente los retardos de la red del supercomputador. El modelo serie, en cambio, no reproduce exactamente los retardos que se esperan, como se ha explicado anteriormente, tener todos los nodos en serie crea un pequeño retardo en los paquetes al nodo cero, aunque en el resto de paquetes si que se genera el retardo correcto y exacto.

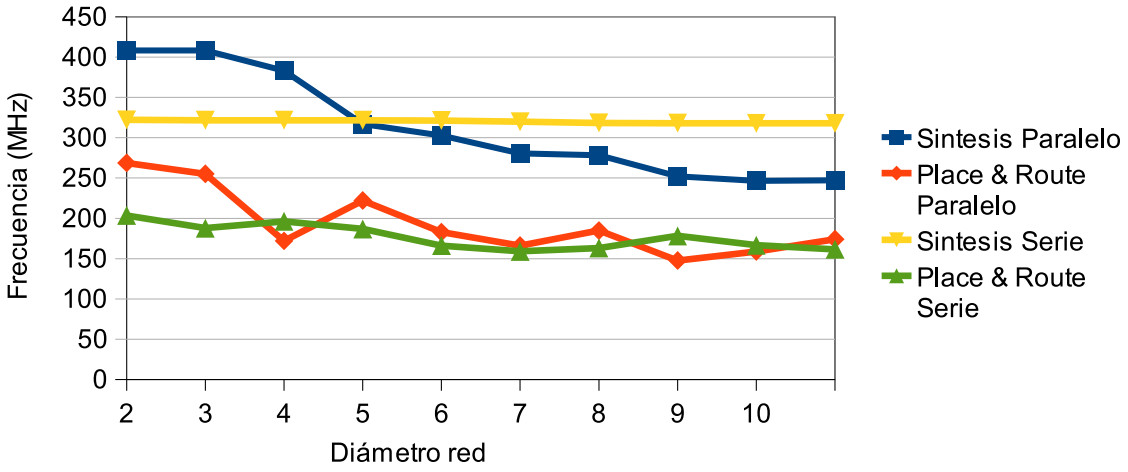


**Figura 6.7:** Frecuencia máxima

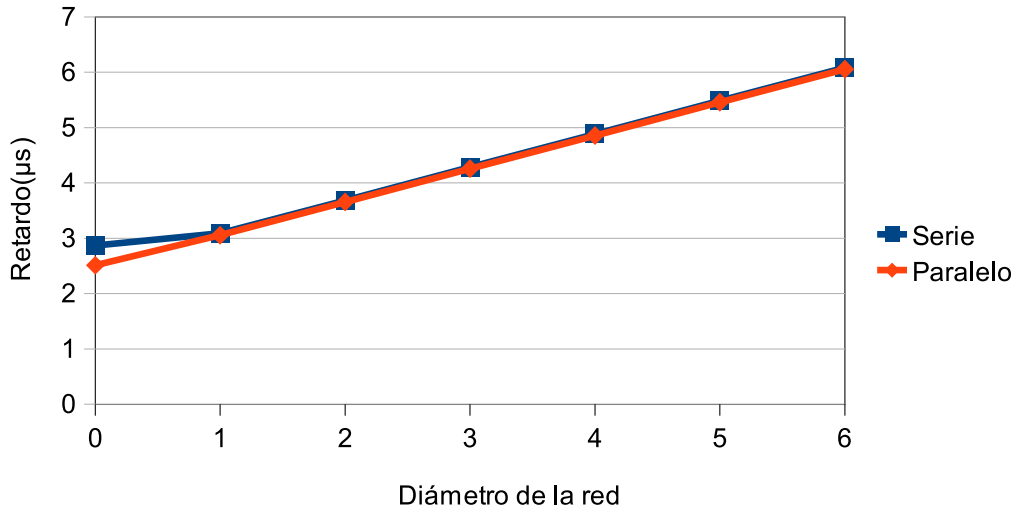


**Figura 6.8:** Retardos de las tarjetas PCIe con los modelos de red

Después de comparar los modelos se ha decidido utilizar el modelo paralelo en el diseño debido a que el modelo serie introduce retardos extra y no puede utilizarse para generar modelos con más de 6 NRs debido a limitaciones del ISE.

Capítulo 7

# Pruebas realizadas con el diseño

En este capítulo se describirán las pruebas realizadas con el nuevo diseño para memoria compartida basado en PCIe para verificar su correcto funcionamiento y comparar sus prestaciones con las del diseño original para HyperTransport y las tarjetas HTX.

## 7.1. Depuración del diseño

En esta sección se describirán las pruebas realizadas durante el desarrollo del presente proyecto final de carrera para comprobar el correcto funcionamiento de los módulos y depurar los mismos.

El proceso de pruebas que se ha seguido es ir integrando y depurando uno a uno cada módulos de los que se compone el sistema completo de memoria compartida para PCIe.

El sistema completo se puede ver en la figura 7.1 (ya mostrada en la figura 4.2), mientras que el orden en el que se ha realizado la integración ha sido el siguiente: primero se ha probado el PCIe Core con el RF. Luego se han añadido el bridge y se ha probado el sistema parcial PCIeCore+bridge+RF, así sucesivamente con el switch, la SMFU y por último con el modelo del supercomputador. Cada sistema parcial ha sido simulado, depurado, implementado en la FPGA y probado conjuntamente con los drivers usando las aplicaciones de test. Para realizar la depuración en hardware se han monitorizado señales internas de la FPGA con ChipScope que es un programa que permite configurar y utilizar analizadores lógicos embebidos en las FPGAs de Xilinx (descrito en la sección 3.3). Cada uno de estos pasos se comentará en los siguientes párrafos.

La simulación de los diseños antes de probarlos en hardware supone una gran ventaja ya que permite tener un gran control sobre las entradas del módulo y obtener información detallada de todas las señales, lo que hace la depuración mucho más sencilla que si se realiza sobre el propio hadware con analizadores lógicos (como el proporcionado por Xilinx: ChipScope). Por otro lado, el mayor inconveniente es que hay situaciones que se dan en hardware que son complejas de replicar en simulación, sobre todo en las simulaciones comportamentales empleadas en el presente trabajo que no incluyen retardos de tiempo.

El programa utilizado para las simulaciones ha sido ModelSim SE-64 6.5. En la figura 7.2 se muestra la pantalla principal de ModelSim que está dividida en las siguientes partes:

- A la izquierda puede ver la lista de los módulos del diseño
- En el centro se encuentra la lista de las señales del módulo seleccionado en la lista

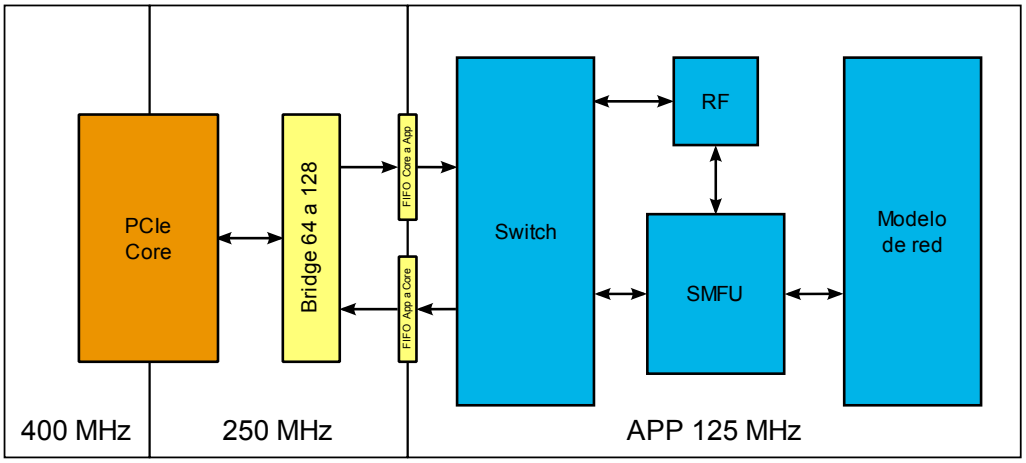


**Figura 7.1:** Diagrama de bloques del diseño

- A la derecha están las señales a monitorizar
- En la parte inferior está la consola, donde se muestran los mensajes de compilación y simulación.

Las primeras pruebas realizadas han sido para depurar el PCIe_RF_wrapper, el módulo escrito para conectar el banco de registros a la interfaz AXI4-Stream. Para no tener que escribir vectores de simulación (ficheros con los valores que se introducen en las entradas del modulo durante la simulación) a mano, tarea costosa que, además puede dar lugar a numerosos fallos. se ha simulado el PCIe_RF_wrapper junto a los módulos a los que estará conectado en la tarjeta: el banco de registros y un core PCIe de Xilinx configurado en PCIe x8 generación 2 para que la interfaz AXI sea de 128 bits. La utilización del core PCIe aporta una gran ventaja debido a que incluye unos vectores de simulación configurables que permiten simular el envío de gran variedad de paquetes y comprobar si las respuestas se reciben correctamente.

En la figura 7.3 se puede ver parte de una simulación de acceso al banco de registros. Concretamente una lectura a la dirección 0x3800C2. Los datos leídos del banco de registros son indeterminados (x) debido a que no hay ningún módulo conectado al banco de registros que suministre la información.

Después de depurar todos los errores detectados en el PCIe_RF_wrapper en simulación, se han implementado los módulos simulados en la tarjeta PCIe HTG y se ha probado el funcionamiento del módulo en la FPGA con los drivers. Se ha usado ChipScope para monitorizar las señales del diseño dentro de la FPGA mientras se realizaban lecturas y escrituras en el banco de registros desde el driver a través de los ficheros virtuales del módulo `extoll_rf`.

En la figura 7.4 se muestra una lectura al banco de registros. Se han incluido los cursores para mostrar los paquetes PCIe. La primera columna corresponde con el primer cursor (azul) y la segunda columna con el segundo (verde).

Una vez el PCIe_RF_wrapper ha sido depurado y todos los tests han sido superados satisfactoriamente, se ha pasado a integrar en el diseño el axi_64_128_bridge y se ha cambiado el core PCIe por uno con generación 1 x8 para que tenga una interfaz AXI de 64 bits. Se ha simulado el nuevo diseño con los mismos vectores utilizados en las anteriores simulaciones del core

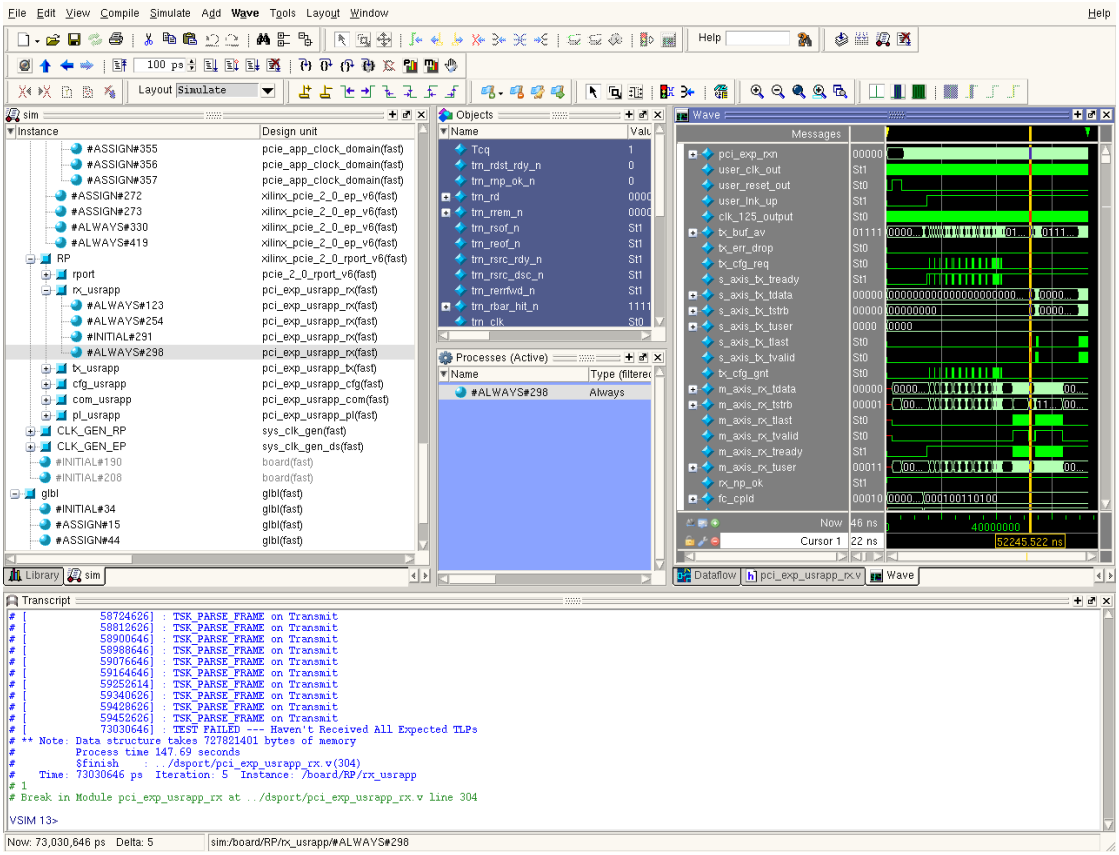


**Figura 7.2:** Pantalla principal de ModelSim

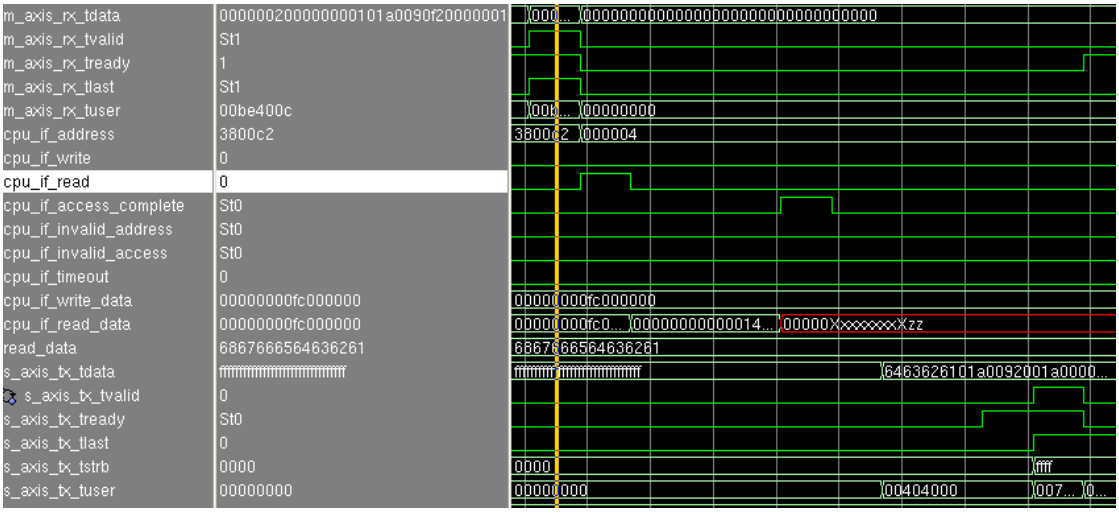


**Figura 7.3:** Simulación del banco de registros

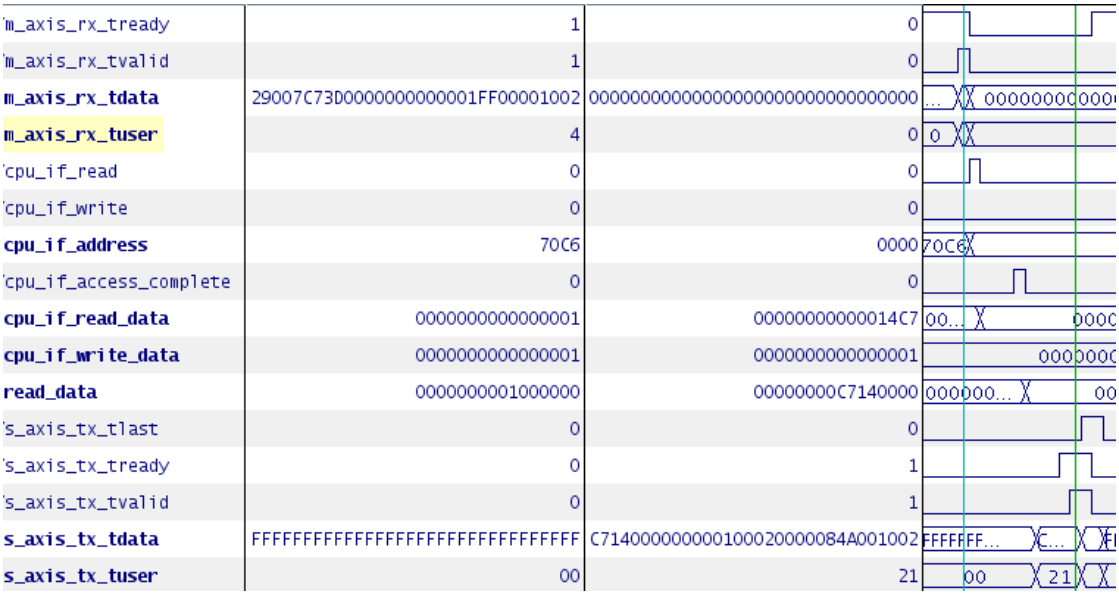


**Figura 7.4:** Depuración del banco de registros

2x8. Esta es una de las ventajas de utilizar los vectores de simulación incluidos en el core PCIe: pueden utilizarse los mismos vectores de simulación aunque cambie la interfaz del core con el resto del diseño ya que en los vectores solo hace falta configurar los paquetes PCIe y el orden en el que se envían y se reciben.

En la figura 7.5 se muestra la traducción de una escritura de 8 bytes a una dirección de 64 bits (un paquete de 24 bytes), de la interfaz de AXI 64 bits a la de 128. Se puede observar como el diseño junta los dos primeros ciclos y establece las señales `is_eof` de `tuser` a 10111b para indicar que sólo los primeros 64 bits del segundo ciclo son válidos.

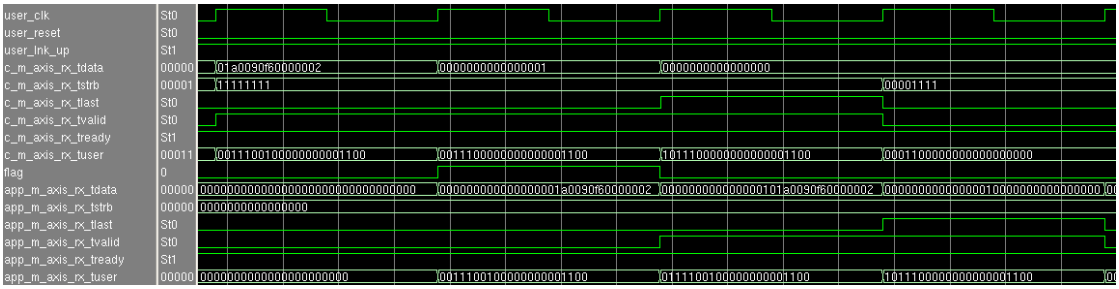


**Figura 7.5:** Simulacion del Bridge

Una vez depurado el bridge en simulación se ha implementado en la tarjeta y se han repetido los tests con el banco de registros y el driver para comprobar que el nuevo módulo funciona correctamente en la tarjeta.

La figura 7.6 muestra una captura de ChipScope del bridge con una petición de lectura a una dirección de 32 bits (paquete de 12 bytes). En este caso, el bridge une dos ciclos de 64 bits en uno de 128 bits.

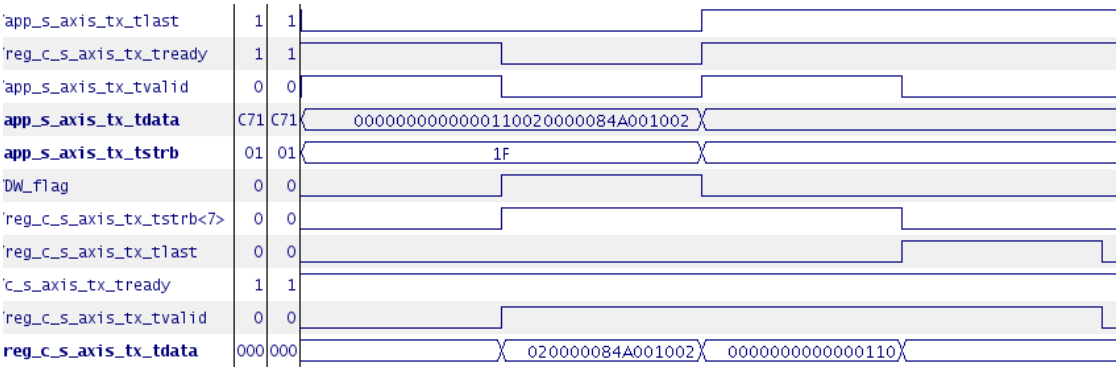


**Figura 7.6:** Depuración del bridge (transmisión de datos del core al RMC)

En la figura 7.7 se puede ver la traducción de 128 bits a 64 de la respuesta a la lectura anterior. El paquete es de 20 bytes por lo que ocupa 2 ciclos de 128 bits y 3 de 64 bits. Entre la recepción de los dos ciclos del RMC, el bridge pone a cero el `tready` para no aceptar el segundo ciclo hasta que no se haya enviado completo el primero.

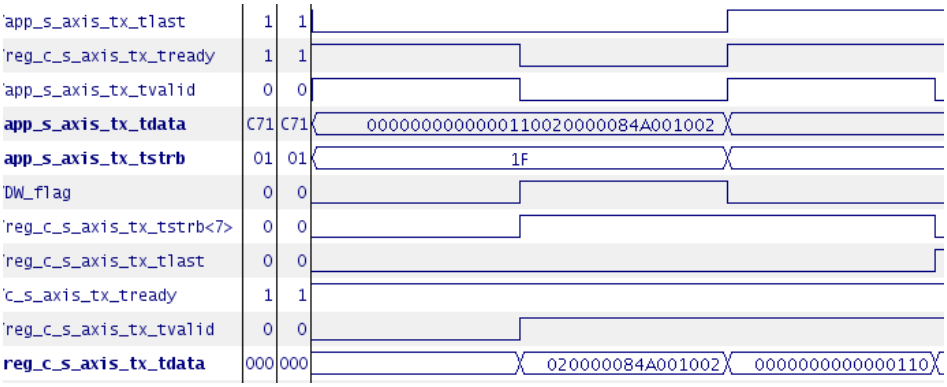


**Figura 7.7:** Depuración del bridge (transmisión de datos del core al RMC)

El siguiente módulo en integrarse dentro del proyecto ha sido el AXI_128_switch, este módulo ha sido uno de los más problemáticos debido a que introduce 3 ciclos de retardo en las señales que crean problemas de sincronización entre las señales de `tready` y `tvalid` en el protocolo AXI4-Stream. Para evitar la perdida o envío duplicado de información en el bus, se han tenido que añadir FIFOs en varios módulos conectados al switch con un umbral en la señal de casi lleno de 3, con esto se consigue que no se pierdan paquetes si un módulo no puede aceptar más ciclos de datos durante una transmisión y pone a cero el `tready`. En el banco de registros no ha sido necesario introducir la FIFO porque nunca pone a cero el `tready` en medio de una transmisión, si que se han puesto en la SMFU y el entre bridge (o el core PCIe si no se utiliza el bridge) y el switch. Las FIFOs que se han conectado entre el switch y el bridge tienen dos relojes distintos en cada lado: uno para el core PCIe y el bridge y otro para el resto del diseño (switch, banco de registros y SMFU). El objetivo de los dos relojes es poder bajar la frecuencia de SMFU si no se consigue que funcione a 250 Mhz, frecuencia a la que funcionará la parte del core PCIe. Para evitar que reciban ciclos duplicados, se ha modificado el protocolo AXI4S para

que no active la señal de `tvalid` hasta que no reciba un `tready`. Por lo que los módulos deberán pedir un canal del switch sin poner a 1 el `tvalid` y cuando reciban el `tready` a 1, activar el `tvalid`.

Para comprobar que el switch permite que el core se comunique con varias unidades funcionales, se han conectado tres bancos de registros (uno para cada BAR configurado en la tarjeta) a los puertos 1, 2 y 3 del switch, el core está conectado en el puerto 0.

En la figura 7.8 se muestra una simulación del PCIe Core, con el bridge, el switch y con tres bancos de registros, realizando una escritura y una lectura en cada banco. Esta simulación se realizó antes de integrar la SMFU, por lo que aun no se habían modificado las señales del protocolo AXI para que se activara antes `tready` que `tvalid`. Se puede observar como dependiendo del valor de las señales `tuser[25:22]` el switch abre el canal correspondiente. El core (fu0) pide el canal uno (`tuser[23]`), dos (`tuser[24]`) o tres (`tuser[25]`) donde están conectados los bancos de registros y los bancos piden el canal cero (`tuser[22]`) para enviar las respuestas al core.

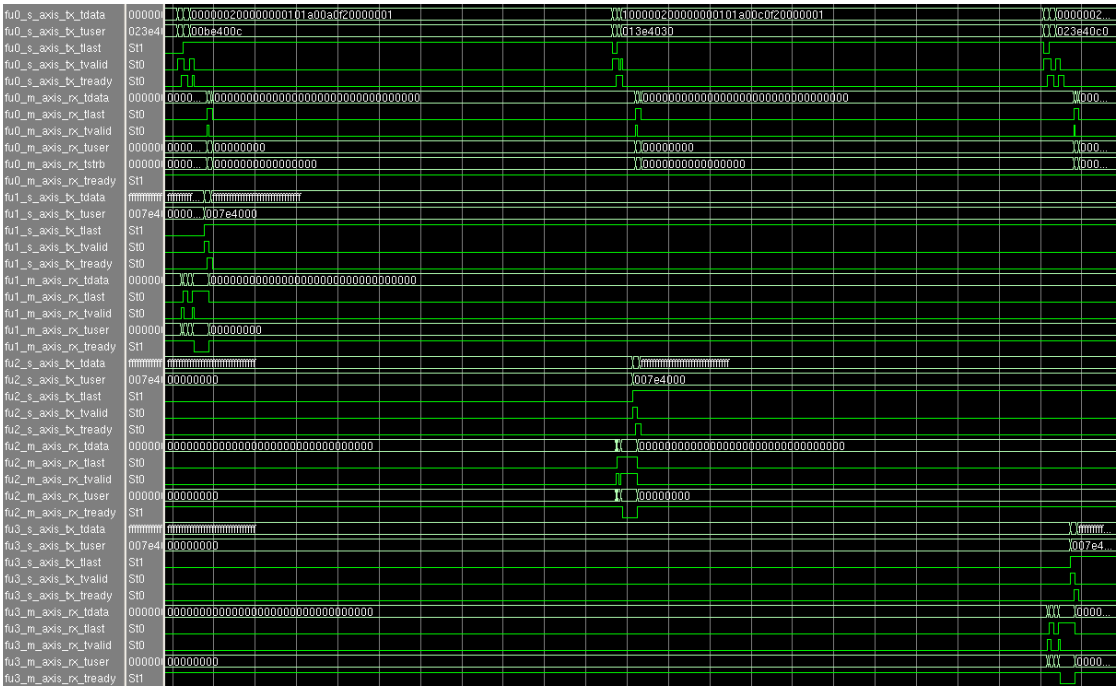


**Figura 7.8:** Simulación del Switch

Este diseño parcial ha sido implementado en la tarjeta. Como los ficheros de sysfs del driver sólo permiten utilizar un banco de registros, para poder acceder a todos los bancos se han programado accesos a las direcciones de los BARs directamente desde el driver. En la captura de ChipScope de la figura 7.9 se muestran las señales del switch durante una lectura al banco de registros. En esta captura ya se ha modificado el banco de registros para que espere la activación de `tready` antes de poner a uno `tvalid`.

El siguiente paso ha sido substituir los dos bancos de registros extra por la SMFU. La SMFU ya había sido simulada junto a un core PCIe [12], por lo que se ha simulado directamente junto al switch y al banco de registros. Para no introducir más de un módulo a la vez, no se ha incluido un

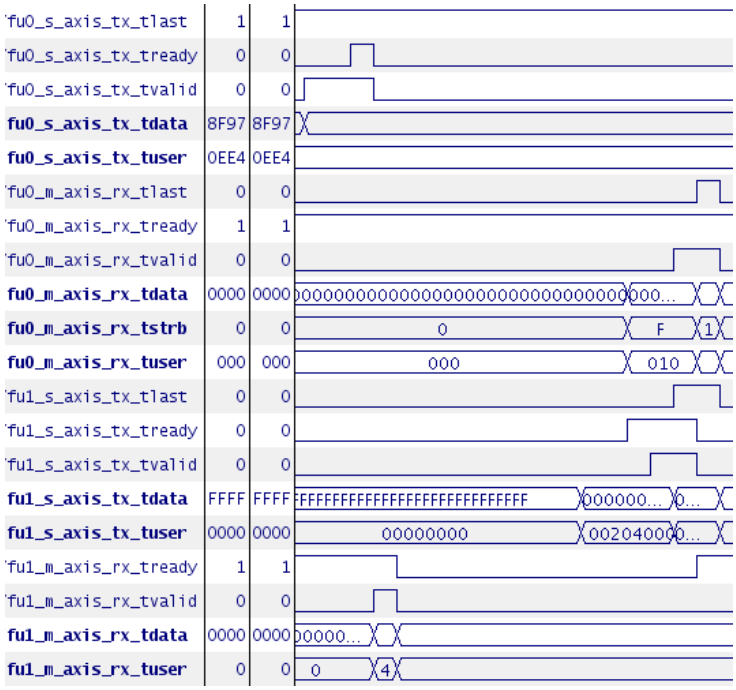


**Figura 7.9:** Depuración del switch

módulo de red, se han conectado el puerto de red de salida de la SMFU con el de entrada. Esto ha sido posible gracias al uso de AXI4S como interfaz común en todo el diseño. Esta conexión no habría sido posible en el diseño de las tarjetas HTX sin modificar los módulos porque la interfaz de envío de paquetes del puerto de red es distinto al de recepción. En las primeras pruebas, los parámetros que recibe la SMFU del banco de registros no se han conectado y se han dejado a constantes, como se había simulado en [12].

En la figura 7.10 se muestra una simulación de la SMFU con el switch, se puede ver que la SMFU ya ha sido modificada para funcionar con el switch y que los buses AXI de la red están cortocircuitados.

Una vez ha funcionado sin la conexión al banco de registros, se han conectado las señales de configuración y estadísticas entre la SMFU y el banco de registros. Para poder simular este caso se han escrito los parámetros en el banco, se han añadido al principio de los vectores de simulación escrituras a las direcciones apropiadas del BAR0, que está conectado al banco de registros.

Cuando se ha implementado en la tarjeta, se ha comprobado que la SMFU recibe correctamente los parámetros escritos en el banco de registros desde el driver y que las estadísticas leídas corresponden con el tráfico que se ha producido en la tarjeta.

En la figura 7.11 se muestra una captura de ChipScope con escrituras al banco de registros y los registros de la SMFU que se modifican. Como puede verse, en cada escritura se modifica el valor de un registro de configuración de la SMFU.

En la figura 7.12 se muestra como la SMFU activa las señales de envió de estadísticas. En la captura se muestran las interfaces AXI de la SMFU con el switch y el módulo de red, las señales de envío de estadísticas y los registros que guardan las estadísticas. Puede verse como

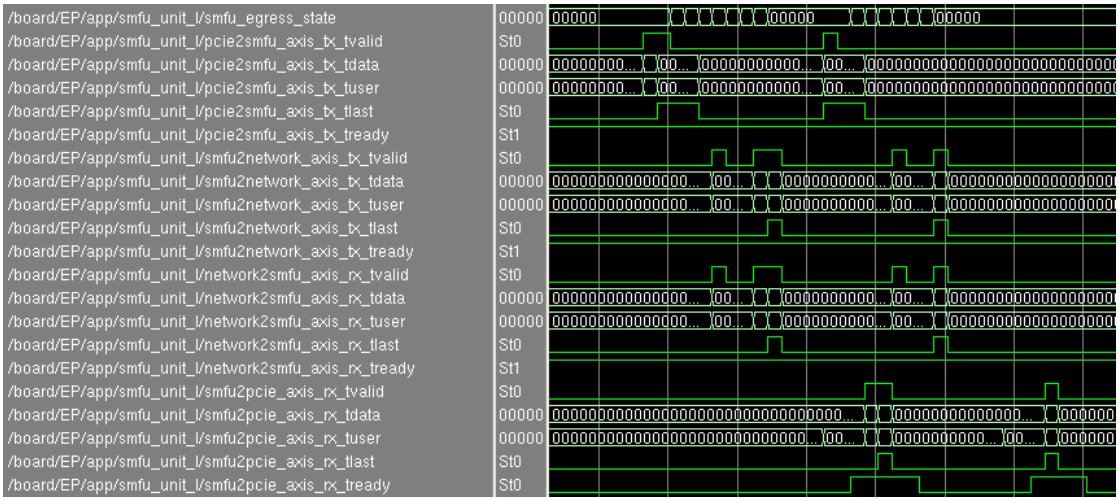


**Figura 7.10:** Simulación con la SMFU sin modelo de red

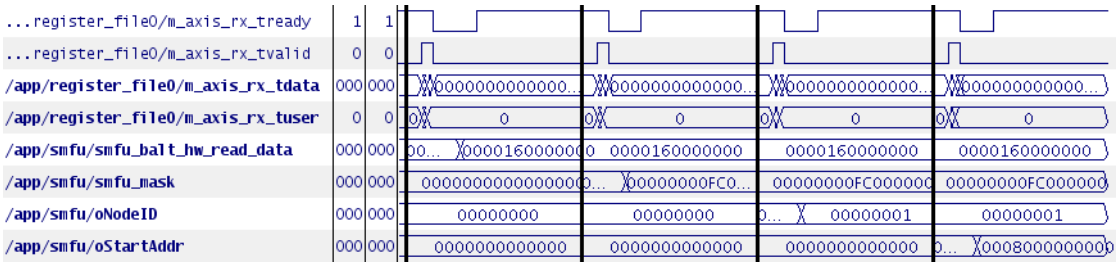


**Figura 7.11:** Captura de la configuración de los registros de la SMFU

la SMFU activa las señales dependiendo del tipo del paquete y si se envía por la red o se recibe. Estas señales incrementan los valores del banco de registros a los que puede accederse desde los ficheros de sysfs.

Por último se han integrado y probado los modelos de red. Para estas pruebas se han configurado los modelos con una red virtual Hipercubo 6D de 64 nodos con un diámetro de 6. En el fichero de configuración del banco de registros del driver se ha añadido código para configurar las direcciones base de cada nodo, como la red es virtual y solo existe un nodo físico, se han asignado direcciones incrementales a cada nodo creando 64 rangos de direcciones consecutivos de 64MB para evitar que una escritura a un nodo modifique la memoria asignada a otro.

En la figura 7.13 se muestra una captura del modelo de red paralelo gestionando paquetes a distintos nodos. El modelo recibe cinco paquetes a los que tiene que aplicar los retardos: 2, 6, 1, 4, 4 respectivamente (señal `n_hops`). Los paquetes se envían a los nodos de retardo correspondientes y cuando termina su tiempo de retardo, el árbitro les concede el canal de salida y se envían por el bus AXI. El árbitro round-robin utiliza la señal `mask` (que se desplaza un bit a la izquierda cada vez que se envía un paquete) para priorizar los canales.

En la figura 7.14 se muestra una captura del modelo de red serie. En esta captura llegan seis paquetes a los que se les debe de aplicar los siguientes retardos: 1, 6, 6, 3, 5, 5. Se puede ver como los paquetes pasan por todos los nodos serie (buses `din`), pero cuando termina su retardo

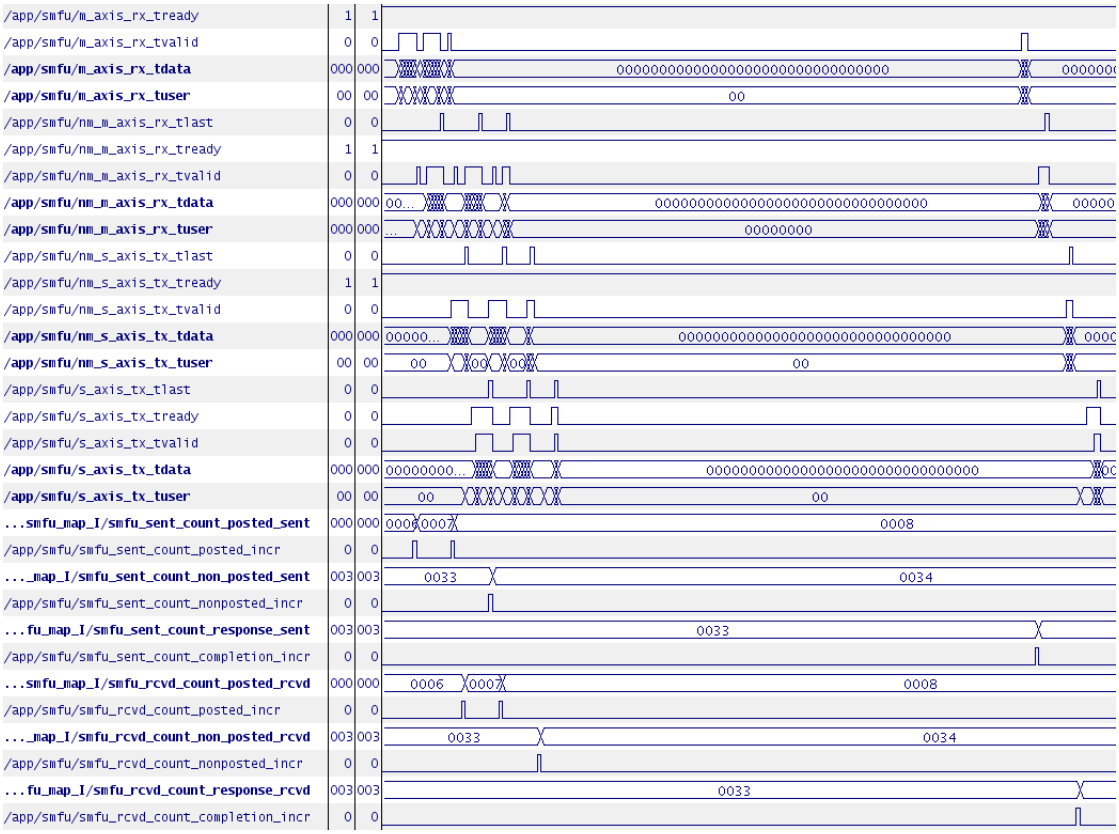


**Figura 7.12:** Captura del envio de estadisticas por parte de la SMFU

pasan por el resto de nodos sin esperas.

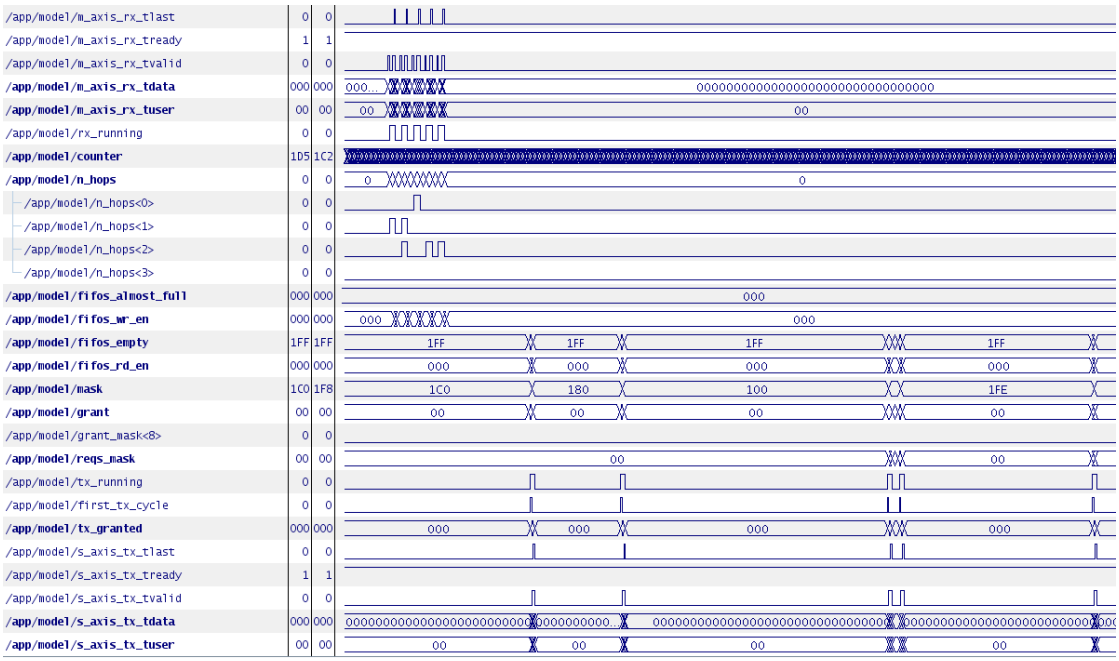


**Figura 7.13:** Captura del modelo de red paralelo

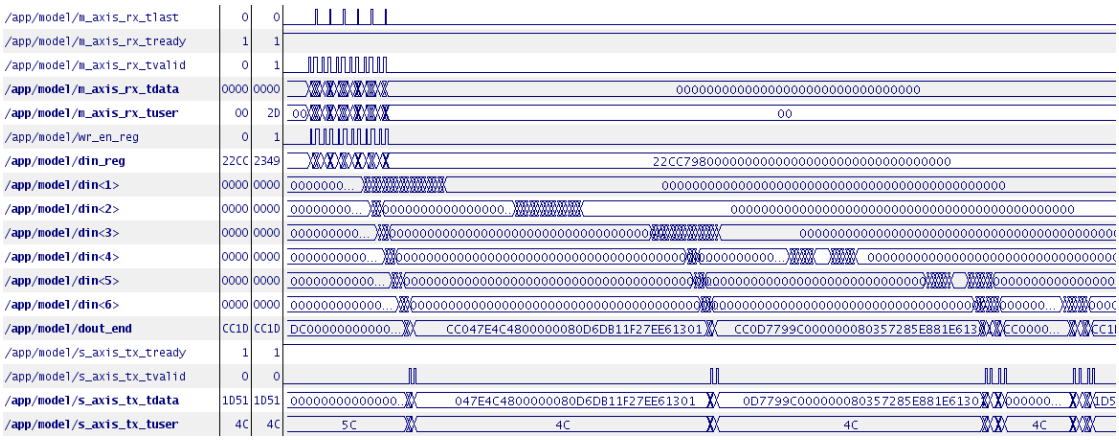


**Figura 7.14:** Captura del modelo de red serie

## 7.2. Pruebas software

Para facilitar la depuración del diseño, se ha modificado un programa creado para las tarjetas HTX llamado `smfu_dump`. El programa original configura el acceso a la SMFU, escribe secuencialmente en una región de la memoria y lee la misma región mostrando los datos leídos por pantalla.

Al `smfu_dump2` se le han realizado las siguientes modificaciones:

- El tamaño de la memoria a utilizar y el desplazamiento de la misma se pueden cambiar con parámetros por la línea de comandos.
- El programa comprueba automáticamente si los datos leídos son los mismos que se han escrito. Si hay errores puede abortar la ejecución o mostrar un error y continuar leyendo.
- El tamaño de las operaciones se puede cambiar entre 4 y 8 bytes.
- Se calcula la latencia de las operaciones de lectura.
- Se calcula el ancho de banda de las escrituras.

Se ha creado otra versión llamada `smfu_dumpRand` que realiza operaciones de lectura y escritura con direcciones, tamaños y orden aleatorios.

Estas aplicaciones han sido de gran ayuda para detectar y localizar fallos en los módulos de diseño durante la etapa de integración descrita en la sección anterior, ya que permiten crear cargas muy diversas para poder comprobar el correcto funcionamiento del diseño en un gran número de escenarios. Para depurar los módulos permiten tener un entorno controlado y reproducible para realizar pruebas.

## 7.3. Pruebas sintéticas

Para comprobar el correcto funcionamiento del diseño integrado se ha ejecutado el `smfu_dump2` y el `smfu_dumpRand` en los 4GB disponibles en el PC con unos resultados satisfactorios en todos los casos.

Para comparar las prestaciones del nuevo diseño basado en PCIe con respecto al diseño original, se han realizado varias pruebas con dos tarjetas HTX conectadas con un cable de fibra óptica y con una tarjeta PCIe HTG con el modelo del supercomputador paralelo. Los resultados obtenidos han sido los siguientes:

En la figura 7.15 se muestra la comparativa de las latencias de lectura en accesos a la memoria del mismo nodo (loopback) y a memoria de otro nodo a un enlace de distancia (1 hop). Los resultados se han obtenido con `smfu_dump2` realizando lecturas secuenciales a un rango de 1GB. Se han ejecutado varias varias veces todos los tests y se han descartado los que se desviaban demasiado de la media. En la gráfica, el eje de las abscisas representa la distancia entre los nodos (cero o uno) y eje de las ordenadas la latencia en microsegundos. Cada línea de datos representa una tecnología (PCIe o HT) y el tamaño de los datos leídos. El tamaño de las lectura no en significativo en las latencias debido a que la mayor parte de los retardos se corresponde a la propagación de los paquetes, en la que el tamaño no es relevante. La diferencia entre las latencias en loopback y a un nodo directamente conectado son de 0.6µs. Este retardo ya se ha

analizado en profundidad en el capítulo 6. En la figura 7.15 además se puede observar que la latencia de la tarjeta PCIe es un 50 % más grande que la de las HTX. Se ha analizado el tiempo que utilizan las tarjetas PCIe para procesar los paquetes y se ha comparado con el análisis realizado en [15] de las tarjetas HTX (de los 1.9µs de latencia en las lecturas a un nodo directamente conectado, el 20 %, 0.4µs, corresponde a la FPGA). Para calcular el retardo que corresponde a la FPGA en PCIe, se ha medido con el ChipScope los ciclos que pasan los paquetes de una lectura en la FPGA (157 ciclos) que multiplicado por el tiempo de ciclo (4ns), da 0.6µs de retardo para las FPGA en PCIe. La diferencia entre los tiempos de procesado (0.6-0.4=0.2µs), es mucho menor que la diferencia entre los resultados obtenidos en las pruebas (2.5-1.3=1.2µs y 3-2=1µs), por lo que el aumento de la latencia es principalmente debido a factores externos a la tarjeta, probablemente al tiempo que necesita el chipset para enrutar los paquetes, que no es necesario para las tarjetas HTX.

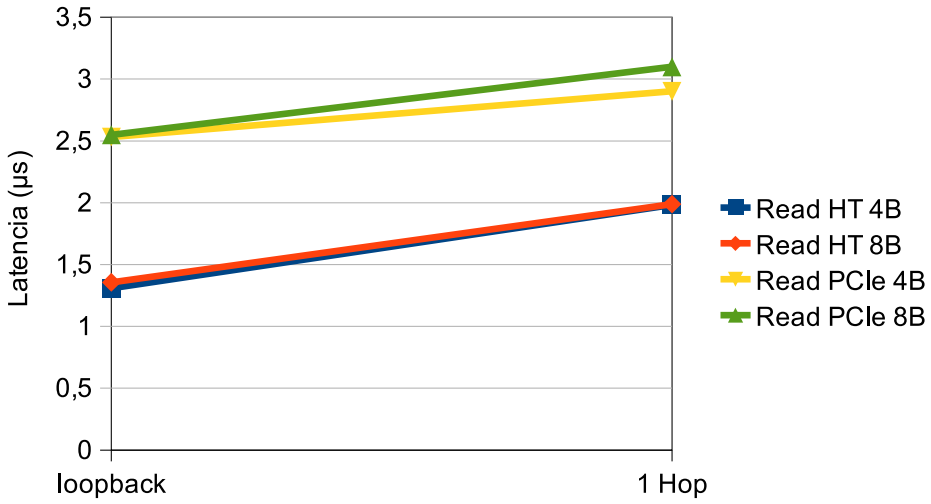


**Figura 7.15:** Latencia de las lecturas

En la figura 7.16 se muestra el ancho de banda de las tarjetas al realizar escrituras de distinto tamaño en modo loopback y a un nodo directamente conectado (1 hop en la figura). El eje de las abscisas representa el tamaño de los datos a escribir en bytes y el eje de las ordenadas el ancho de banda conseguido en cada caso. Cada línea de datos representa una tecnología (PCIe o HT) y la distancia del nodo al que se escribe (cero: loopback y uno : 1 Hop). Puede verse que el ancho de banda, calculado con `smfu_dump2` con escrituras secuenciales de 1GB, es superior en las tarjetas PCIe.

En HT el ancho de banda varía entre 28 y 60 MB/s mientras que en PCIe este valor alcanza 60-80 MB/s. Esto es debido al mayor ancho del bus de datos interno en el diseño PCIe (128 bits frente a 64 bits) que permite un mayor rendimiento de la tarjeta cuando debe procesar paquetes ininterrumpidamente. Esta diferencia no se aprecia en las lecturas, cuyo cuello de botella es la latencia debido a que el procesador únicamente puede tener una lectura pendiente a las tarjetas, pero si es relevante en las escrituras, que el procesador puede enviar ininterrumpidamente.

Además, se puede observar que en HyperTransport el ancho de banda disminuye cuando los paquetes pasan por la red, hecho que se puede deber a que el puerto de red tenga menos

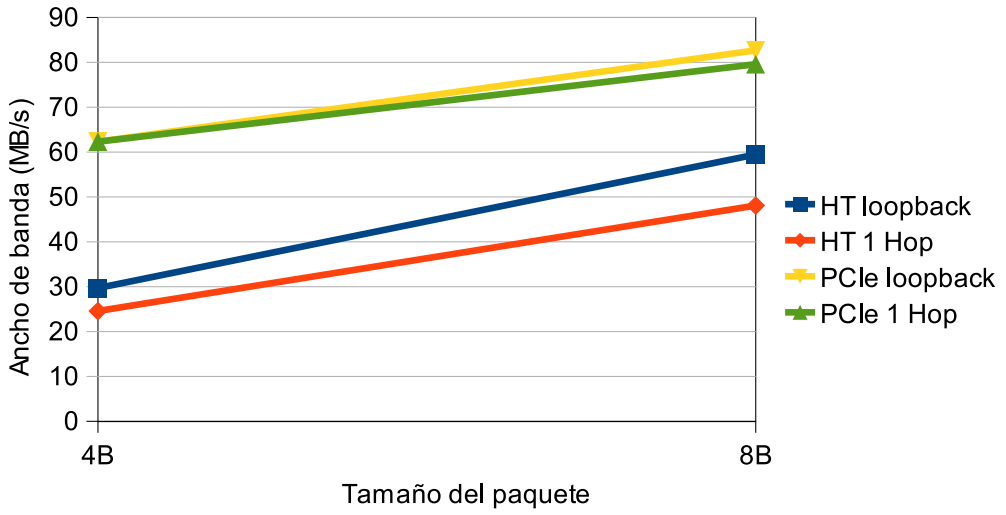


**Figura 7.16:** Ancho de banda de las escritura

ancho de banda que el resto del diseño y cree un cuello de botella. En PCIe no hay diferencias significativas porque el modelo de red ha sido diseñado con FIFOs suficientemente grandes para que no se llenen con las pruebas utilizadas en este proyecto y no se cree un cuello de botella.

## 7.4. Pruebas en la base de datos

Se ha configurado una base de datos MySQL con el plugin `rm_heap` desarrollado en el GAP para el prototipo HTX. Con este plugin es posible almacenar las tablas de la base de datos en memoria remota usando las tarjetas con FPGAs, lo que permite usar la memoria de otros nodos en el prototipo. Como el diseño objeto de este proyecto final de carrera no dispone por el momento de una red funcional, se ha configurado el modelo de red para simular una red hipercubo de 64 nodos y se ha configurado el driver para que asigne 64MB a cada nodo, utilizando los 4GB disponibles para la tarjeta en el PC de pruebas.

Se ha ejecutado sobre la base datos el mismo banco de pruebas utilizado para medir las prestaciones del prototipo HTX en [13]. Este banco de pruebas consiste en una base de datos que imita una red social y sobre ésta se realizan consultas de lectura. Se ha obtenido una latencia de 1.9ms de media en la ejecución de las consultas. Este resultado es mayor al obtenido en el prototipo, que concuerda con los resultados obtenidos en las comparativas anteriores. Aunque el ancho de banda es mayor en PCIe, el cuello de botella se encuentra en la latencia de las lecturas, que son un 50 % superiores en PCIe y hacen que incremente la latencia de la base de datos.

Capítulo 8

# Conclusiones y líneas futuras

En este proyecto final de carrera se ha conseguido implementar y probar en hardware una versión de MEMSCALE sobre PCIe con un modelo en hardware de la red del prototipo del supercomputador. Con esto se ha logrado portar la arquitectura desarrollada para las tarjetas HTX a PCI-Express. Esta versión presenta muchas ventajas a pesar de su mayor latencia (50 % más) debido a que las peticiones de la CPU a la tarjeta deben de pasar por el chipset:

- Permite ampliar el abanico de los procesadores que se puede utilizar con MEMSCALE, como por ejemplo toda la gama de Intel.
- No requiere placas base con el conector HTX.
- La utilización del bus AMBA AXI4-Stream permite controlar el flujo de información en todos los módulos del diseño, lo que no es posible en el diseño de las tarjetas HTX.

## 8.1. Líneas futuras

La principal tarea necesaria para obtener una versión de MEMSCALE completamente funcional sobre PCIe es añadir un módulo de red que permita a distintas tarjetas comunicarse entre sí para poder compartir la memoria entre distintos nodos reales. Este trabajo ya está en las pruebas finales de desarrollo en la Universidad de Heidelberg y se incorporará al diseño en cuando esté disponible.

Otro aspecto mejorable del sistema actual es la latencia, que podría disminuirse con placas base más sofisticadas o compensándolo con aceleraciones por hardware como podría ser el prefetching de los datos. Linea de trabajo que ya está planificada.

## 8.2. Formación y conocimientos adquiridos

Antes de empezar este proyecto, se disponía de conocimientos básicos de desarrollo sobre FPGAs y sólo se habían realizado diseños simples. En la elaboración de este PFC se han mejorado considerablemente estos conocimientos, se ha aprendido a plantear y desarrollar proyectos hardware de media y alta complejidad, así como a depurar diseños de FPGAs con analizadores lógicos.

Por otra parte, se ha formado parte de un grupo de investigación con el que se ha desarrollado el diseño, por lo que se han mejorado los habilidades de trabajo en equipo y se han adquirido conocimientos de como se trabaja dentro de un proyecto de investigación de gran envergadura.

# Bibliografía


[1] http://www.pcitree.de/.

[2] http://www.xilinx.com/products/design-tools/ise-design-suite/.

[3] PCI Express base specification revision 2.0. PCI-SIG, Beaverton, OR, 2006. Rev 2.0.

[4] AMD64 Architecture Programmer's Manual Volume 2: System Programming. Advanced Micro Devices, Inc., 2007. Revision 3.14.

[5] AMD 780G Family Register Reference Guide. Advanced Micro Devices, Inc., 2009. Revision 1.01.

[6] BIOS and Kernel Developer's Guide (BKDG) For AMD Family 10h Processors. Advanced Micro Devices, Inc., 2009. Revision 3.34.

[7] H8QME-2+ USER'S MANUAL. Super Micro Computer, Inc, 2009. Revision 1.0b.

[8] M4A78T-E ASUS Motherboard E4465. ASUSTeK COMPUTER INC, 2009. Version 2.

[9] AMBA 4 AXI4-Stream Protocol Specification. ARM Limited, 2010. Version 1.0.

[10] BACHMANN, PHILIPP: Increased MMIO Space and Read Performance of HT Units on AMD Family 10h Processor Plataforms. Computer Architecture Group, University of Heidelberg, 2009.

[11] HTG-DOC-937: HTG-V6-PCIE-xxxx User Manual. HiTech Global, 2010. Version 3.2.

[12] MISLATA VALERO, SANTIAGO: Adaptación a PCI-Express de un diseño de memoria compartida distribuida basado en FPGAs. BSc Thesis, Escuela Técnica Superior de Ingeniería Informática, Universitat Politècnica de València, Spain, 2011.
http://hdl.handle.net/10251/11981

[13] MONTANER, HÉCTOR; SILLA, FEDERICO; FRÖNING, HOLGER y DUATO, JOSÉ: «A new degree of freedom for memory allocation in clusters». Cluster Computing, 2011, pp. 1 – 23. ISSN 13867857.
http://dx.doi.org/10.1007/s10586-010-0150-7

[14] MONTANER, HÉCTOR; SILLA, FEDERICO; FRÖNING, HOLGER y DUATO, JOSÉ: «MEMSCALE™: A Scalable Environment for Databases». En: High Performance

Computing and Communications (HPCC), 2011 IEEE 13th International Conference on, pp. 339 – 346, 2011. doi: 10.1109/HPCC.2011.51.
http://dx.doi.org/10.1109/HPCC.2011.51

[15] SCHINKE, JANUSZ: Design and Verification of a Low Latency Functional Unit for Direct Access to Remote Memory. Diploma Thesis, University of Mannheim, 2009.

[16] UG175: LogiCORE™ IP FIFO Generator v7.2. Xilinx, Inc., 2010. Version 1.1.

[17] UG534: ML605 Hardware User Guide. Xilinx, Inc., 2010. Version 1.4.

[18] UG671: Virtex-6 FPGA Integrated Block for PCI Express, User Guide. Xilinx, Inc., 2011. Version 1.1.

[19] XTP044: ML605 PCIe x8 Gen1 Design Creation. Xilinx, Inc., 2010.

[20] XTP047: ML605 MIG Design Creation. Xilinx, Inc., 2010.

# Lista de Abreviaturas

| | |
|---|---|
| BARs | Base Address Registers |
| DCT | DRAM controller |
| DDR3 SDRAM | Double Data Rate 3 Synchronous Dynamic Random Access Memory |
| FIFO | First In First Out |
| FPGA | Field-Programmable Gate Array |
| GAP | Grupo de Arquitecturas Paralelas |
| HT | HyperTransport |
| HTG | HiTech Global |
| HTX | HyperTransport eXpansion |
| IP | Intellectual Property |
| LP | Link Port |
| LUT | LookUp Tables |
| NP | Network Port |
| NR | Nodo de Retardo |
| PCI | Peripheral Component Interconnect |
| PCIe | PCI Express |
| PFC | Proyecto Final de Carrera |
| RF | Register File |
| RMC | Remote Memory Controller |
| SFP | Small form-factor pluggable |
| SLR | Shift Register LUT |

| | |
|---|---|
| SMFU | Shared Memory Functional Unit |
| SO-DIMM | Small Outline Dual In-line Memory Module |
| SSD | Solid State Disk |
| TLP | Transaction Layer Packet |
| UART | Universal Asynchronous Receiver/Transmitter |
| UPV | Universitat Politècnica de València |
| USB | Universal Serial Bus |