UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA
DEPARTAMENTO DE INFORMÁTICA DE SISTEMAS Y COMPUTADORES

Máster en Ingeniería de Computadores y Redes

Trabajo Fin de Máster

Diseño de un interfaz de programación en OpenCL para clústers de alto rendimiento basados en sistemas multi-FPGA

Autor: David Rodríguez Agut

Directores: José Flich Cardo y Rafael Tornero Gavilá

Fecha de presentación: 3 de Septiembre de 2020

Resumen

La popularidad de las FPGAs como unidades de cómputo en sistemas HPC ha aumentado considerablemente. No obstante, aunque son dispositivos muy flexibles, también son complejos y difíciles de programar para los usuarios finales. Por este motivo, hay un esfuerzo por parte de los principales fabricantes de FPGAs, Xilinx e Intel/Altera, para facilitar la programación de estos dispositivos a través de otros paradigmas de programación, como OpenCL. En este documento se presenta una plataforma hardware (DSA) con soporte para OpenCL y comunicación entre FPGAs, habilitado para utilizar reconfiguración parcial expandida. El proyecto consigue la comunicación efectiva entre FPGAs tanto desde un servidor, como desde los kernels de cómputo. Sentando las bases para un uso adecuado de un clúster de 12 FPGAs incluido en el prototipo de MANGO.

Palabras clave

Clústers alto rendimiento; FPGA; OpenCL; HLS; Reconfiguración parcial

Keywords

High performance clusters; FPGA; OpenCL; HLS; Partial reconfiguration

Índice general

Siglas

AI	Artificial Intelligence.
ASIC	Application Specific Integrated Circuit.
AXI	Advanced eXtensible Interface.
BD	Big Data.
BIOS	Basic Input/Output System.
CPU	Central Processing Unit.
DDR	Double Data Rate.
DMA	Direct Memory Access.
DNN	Deep Neural Network.
DSA	Device Support Archive.
FPGA	Field Programmable Gate Arrays.
GPU	Graphics Processing Unit.
HD	High Density.
HDL	Hardware Description Language.
HP	High Performance.
HPC	High-Performance Computing.
HR	High Range.
I/O	Input/Output.
MGT	Multi-Gigabit Transceivers.
MMCM	Mixed-Mode Clock Manager.
OpenCL	Open Computing Language.
PCIe	PCI Express.
PCS	Physical Coding Sublayer.
PLI	Programming Language Interface.
PLL	Phase-Locked Loop.
PMA	Physical Medium Attachment.
PR	Partial Reconfiguration.

RTL	Register-Transfer Level.
SDR	Single Data Rate.
SLR	Super Logic Region.
XPR	Expanded Partial Reconfiguration.

Capítulo 1

Introducción

1.1. Motivación del proyecto

Los sistemas de computación tradicionales dependían del uso de Central Processing Unit (CPU) como unidad principal de cómputo. Sin embargo, a medida que el rendimiento y el tamaño de estos sistemas fue aumentando, la influencia del consumo energético y la disipación de la energía se volvió aparente. Como resultado, las CPUs dejaron de ser consideradas como los únicos componentes de cálculo del sistema.

Esto ha llevado a la aparición de lo que hoy se conoce como computación heterogénea. En estos sistemas, existen diferentes tipos de unidades de cálculo cada uno con un consumo de energía y rendimiento distinto. Estos dispositivos difieren en la arquitectura y, por lo tanto, cada uno se adapta mejor a enfoques de programación y definiciones de problemas específicos y particulares.

El primer impulso que se tuvo en High-Performance Computing (HPC) fue la adopción de Graphics Processing Unit (GPU) como unidades de cómputo, originalmente utilizadas en sistemas de sobremesa para el procesamiento de gráficos. La capacidad de cómputo de las GPUs, dado que disponen de muchos núcleos pequeños, las hacen idóneas para ejecutar aplicaciones paralelas eficientemente. Aunque el consumo de energía es parecido al de las CPUs, las GPUs ofrecen un rendimiento de uno o dos órdenes de magnitud superior. De hecho, hoy en día aplicaciones de Artificial Intelligence (AI) con Deep Neural Network (DNN), y la fusión de Big Data (BD) y HPC han hecho que las GPUs estén consideradas como componentes básicos y claves en este tipo de sistemas.

Aunque la combinación de CPU y GPU es la elección principal para los sistemas HPC, empieza a haber una tendencia para utilizar otros componentes heterogéneos que puedan utilizarse para otros dominios de aplicación específicos, o donde el consumo de energía y su disipación es un factor a tener en cuenta. Este es el caso de los *Manycores* (un claro ejemplo fue el Intel Xeon Phy) y de las Field Programmable Gate Arrays (FPGA). Las FPGAs son dispositivos flexibles cuya arquitectura puede ser programada, y de este modo el hardware de esta puede adaptarse perfectamente al algoritmo utilizado para resolver un problema. Las

FPGAs son ampliamente utilizadas en sistemas empotrados y aplicaciones críticas como por ejemplo automoción, aviónica y misiones espaciales.

Existen algunos esfuerzos para la adopción de FPGAs como unidad de procesamiento en sistemas HPC. Aunque las FPGAs tienen una gran flexibilidad, no son tan potentes como las GPUs en operaciones de coma flotante. Sin embargo, pueden ser muy útiles en dominios de aplicación donde los cálculos en coma flotante no son de vital importancia, y pueden utilizarse algoritmos de precisión reducida en su lugar. En este proyecto se apuesta por el uso de FPGAs como una unidad de procesamiento más y su adopción en los sistemas HPC.

El uso de FPGAs en sistemas HPC se está extendiendo debido a su menor consumo de energía comparado con una GPU. Sin embargo, para adoptar el uso de FPGAs en sistemas HPC, es necesario tener en cuenta la flexibilidad que estos sistemas pueden ofrecer para el programador. Es bien conocida la complejidad y dificultad para programar dispositivos FPGA por parte de los usuarios finales. Para facilitar la programación de las FPGAs por parte de los usuarios finales, existe el paradigma de programación Open Computing Language (OpenCL), aunque también existen otras soluciones. OpenCL define un enfoque de programación facilitando la programación de FPGAs abstrayendo las complejidades de estas.

OpenCL puede utilizarse directamente con la gran mayoría de FPGAs disponibles en el mercado. Los fabricantes, principalmente Xilinx e Intel/Altera, ofrecen sus productos con soporte hardware y software para OpenCL. Aunque los productos ofrecidos por Xilinx e Intel/Altera son diferentes, el flujo de programación es muy similar. En la Figura1.1 se puede observar el flujo de programación para la plataforma SDAccel de Xilinx que ofrece soporte para OpenCL. La idea principal reside en habilitar una plataforma hardware (DSA) y un conjunto de bibliotecas software para reducir la complejidad de programación y facilitar la aceleración de aplicaciones a los desarrolladores de software. Sin embargo, otros sistemas basados en FPGA ofrecidos por compañías de terceros no tienen soporte para OpenCL, dado que se centran en otros campos, principalmente prototipado de Application Specific Integrated Circuit (ASIC). Este es el caso de los productos ofrecidos por la compañía ProDesign [6]. ProDesign ofrece una solución interesante donde un usuario puede construir su propio sistema basado en FPGAs con diferentes componentes como si se tratarán de partes de un juego de LEGO. Módulos FPGA, memorias Double Data Rate (DDR) y componentes de Input/Output (I/O) pueden conectarse de diferentes formas según las necesidades de cómputo.

En este proyecto se ha trabajado con el prototipo de MANGO [5], cuyos componentes del sistema han sido proporcionados por ProDesign. En la Figura1.2 se muestra el diagrama de bloques del sistema. Como se puede observar, el sistema tiene dos tipos de nodos:

- ***Heterogenous Node*** **(HN)**: en estos nodos se instancian diferentes arquitecturas hardware programadas utilizando lenguajes de programación Hardware Description Language (HDL). Cada uno de estos nodos (denominados clústers) está compuesto de 12 FPGAs y 24GB de memoria DDR3 y DDR4 montados encima de cuatro placas base de ProDesign. El clúster está compuesto por diferentes tipos de FPGAs: Xilinx Kintex UltraScale KU115, Xilinx Virtex 7 Series V2000T, Xilinx Zynq 7000 SoC Z100 e Intel Stratix 10 SG280.
- ***General-purpose Node*** **(GN)**: estos nodos se encargan de gestionar tanto las diferentes unidades de computación implementadas en una arquitectura dada, como la arquitectura

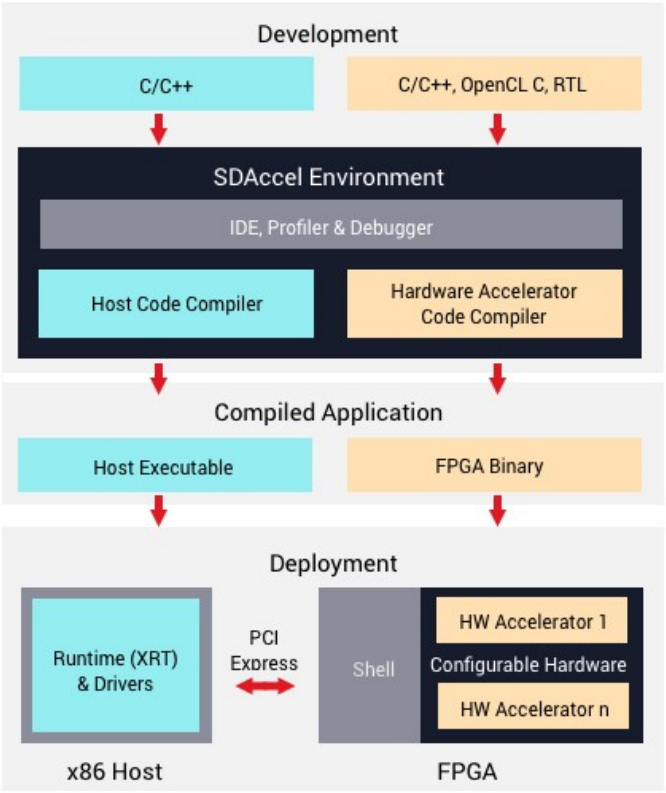

Figura 1.1: Diagrama del flujo de desarrollo para SDAccel [7].

misma. Cada uno de estos nodos consta de un procesador Intel Xeon E5 V3, una GPU, 64GB de memoria DDR4, 1TB SSD y conectividad PCI Express (PCIe).

Cada nodo GN está conectado a dos clústers HN diferentes, a través de ocho líneas PCIe Gen3. De este modo, cada nodo GN tiene acceso a dos clústers HN diferentes, y a un total de 24 FPGAs y 44GB de memoria DDR3/4. En la Figura 1.3 se muestra un nodo HN, donde los módulos FPGA se corresponden con las flechas de color rojo y las interconexiones del clúster se corresponden con las flechas azules y verde. Para conectar las FPGAs dentro de un mismo clúster se utilizan los cables azules que se muestran en la figura, mientras que para la conexión entre dos nodos HN se utilizan módulos QSFP, que se corresponde con el módulo de la imagen que señala la flecha verde.

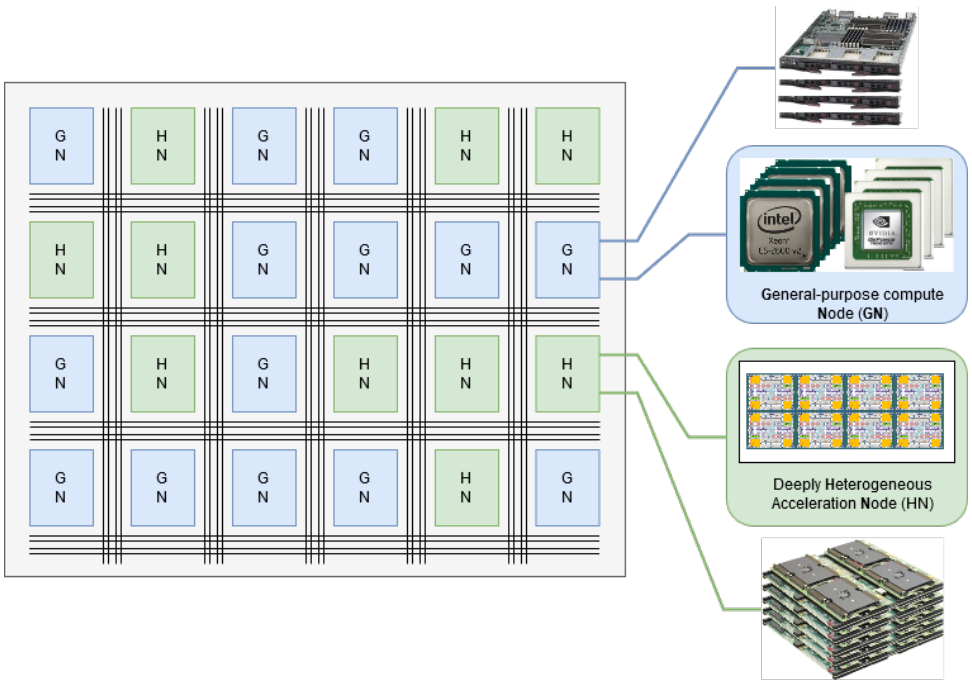

Figura 1.2: Diagrama de bloques del sistema MANGO.

1.2. Objetivos del proyecto

El objetivo del proyecto es proporcionar compatibilidad de uso de OpenCL en el prototipo de MANGO. Los clústers de FPGAs del prototipo se utilizarán como entidades únicas programables mediante un único dispositivo OpenCL. Actualmente no hay posibilidad de este soporte en el prototipo.

Específicamente, el objetivo del proyecto es desarrollar un Device Support Archive (DSA) habilitado para utilizar reconfiguración parcial para la plataforma Xilinx Kintex UltraScale KU115 proporcionada por la empresa ProDesign, que incorpore soporte para la comunicación entre FPGAs a través de los interfaces AXI Memory Mapped y AXI4-Lite. La idea es utilizar los enlaces entre FPGAs dentro de los nodos HN del prototipo de MANGO para proporcionar un enlace de comunicación desde el servidor a todas las FPGAs del nodo, y facilitar la programación de los dispositivos a los desarrolladores de software.

Este objetivo se puede estructurar en los siguientes subobjetivos:

- Habilitar una plataforma hardware con soporte para reconfiguración parcial expandida para utilizar en las FPGAs KU115 de los nodos HN.
- Habilitar el uso de OpenCL en la plataforma hardware.
- Modificar la plataforma hardware para dar soporte a la comunicación entre FPGAs.
- Permitir a un servidor acceder a una FPGA con la que no está conectado directamente, mediante transacciones DMA utilizando los interfaces AXI Memory Mapped y AXI4-Lite.

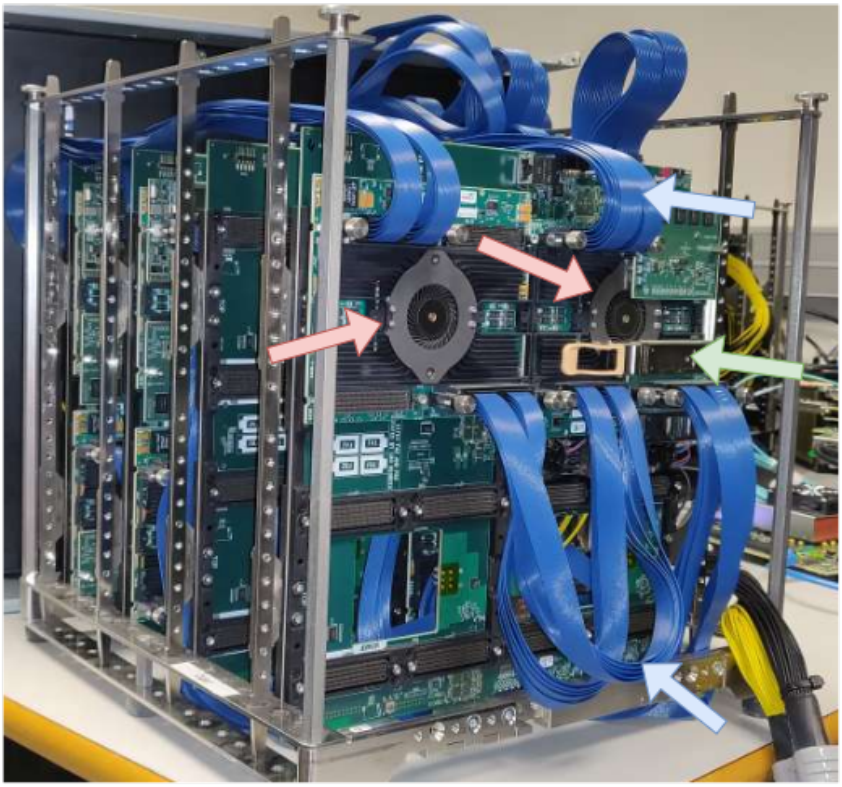

Figura 1.3: Imagen de un nodo HN del prototipo de MANGO.

1.3. Estructura de la memoria

El documento está estructurado en cinco capítulos y un anexo. En el capítulo 2 se describe el proyecto desarrollado así como su finalidad, y se describen los conceptos básicos para entender la naturaleza del proyecto. En el capítulo 3 se presenta el diseño de un DSA para funcionar en una FPGA Xilinx Ultrascale KU115. En el capítulo 4 se comentan las modificaciones realizadas al diseño de referencia descrito en el capítulo anterior, para añadir soporte para la comunicación entre FPGAs. Finalmente, en el capítulo 5 se comentan las conclusiones finales del proyecto. Además, existe un anexo donde se describe el simulador *QuestaSim* que se ha utilizado durante el desarrollo del proyecto para comprobar el correcto funcionamiento del diseño.

Capítulo 2

Descripción del proyecto

En el proyecto descrito en este documento se presenta el funcionamiento de un DSA habilitado para utilizar reconfiguración parcial, para la plataforma de desarrollo Xilinx Ultrascale KU115 de Xilinx. Un DSA es una plataforma hardware que opera en las FPGAs para permitir la integración del dispositivo y el servidor con la lógica del kernel definido por el usuario. Para este proyecto se ha partido de un diseño de referencia inicial desarrollado por Xilinx [31].

Una FPGA es un tipo de circuito integrado cuya lógica puede ser reprogramada por el usuario para ejecutar una aplicación o funcionalidad deseada despues de haber sido fabricada [39]. Originalmente, para programar una FPGA se utilizaban lenguajes de descripción de hardware: Verilog, VHDL o SystemVerilog. No obstante, debido a la popularidad obtenida en entornos de HPC, ha habido un esfuerzo por parte de los fabricantes de FPGAs para permitir el uso de otros lenguajes de programación y paradigmas. En el caso de Xilinx, a través de la plataforma de desarrollo SDAccel, se pueden programar las FPGAs utilizando el paradigma de programación OpenCL, y los lenguajes de programación C y C++. En la Figura1.1 se muestra un diagrama representando el flujo de desarrollo en SDAccel.

OpenCL [3] es un estándar de programación paralela desarrollado por el consorcio Khronos Group [2], del que Xilinx forma parte, para abordar los desafíos que surgen al programar plataformas de computación heterogénea y multinúcleo. La especificación de OpenCL define un único modelo de programación y un conjunto de abstracciones a nivel de sistema que deben ser soportadas por todas aquellas plataformas hardware que cumplan con el estándar. Esto significa que un ingeniero de software puede aprender un único modelo de programación y utilizarlo en múltiples dispositivos de diferentes fabricantes.

El modelo de OpenCL define una representación lógica de todos los dispositivos capaces de ejecutar un programa OpenCL. Las plataformas OpenCL están definidas por la agrupación de un procesador servidor, y uno o mas dispositivos de cómputo OpenCL. En el caso de Xilinx el procesador servidor se corresponde con un procesador x86 que se comunica con los dispositivos a través de PCIe. En la Figura2.1 se muestra un esquema de la plataforma OpenCL en Xilinx. El servidor tiene las siguientes responsabilidades:

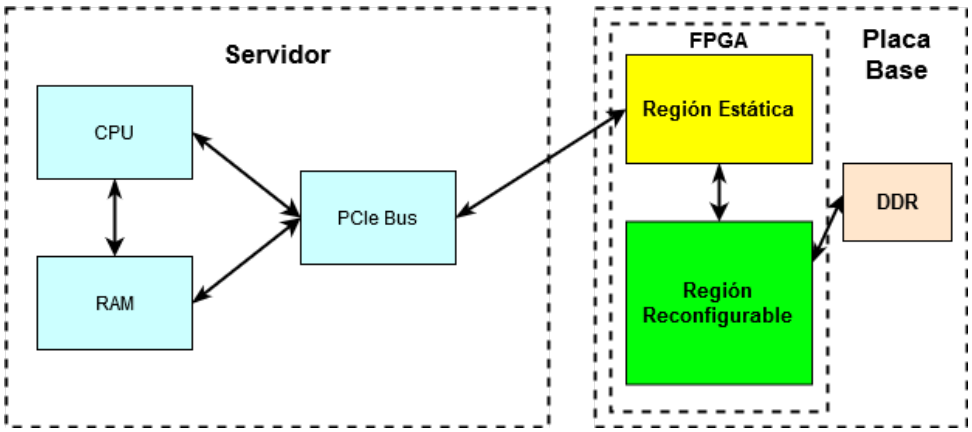

Figura 2.1: Esquema plataforma OpenCL en Xilinx.

- Gestionar el sistema operativo y habilitar los controladores para todos los dispositivos.
- Ejecutar el programa de aplicación del servidor.
- Establecer todos los *buffers* de memoria global, y gestionar las transferencias de datos entre el servidor y los dispositivos.
- Monitorizar el estado de todas las unidades de cómputo del sistema.

Por otro lado, un dispositivo, en el contexto de OpenCL, es un conjunto de recursos hardware donde se ejecutan los kernels de usuario.

OpenCL puede utilizarse en las FPGAs de Xilinx a través del entorno de desarrollo SDAccel [30]. Desde la versión 2019.2, SDAccel está integrado dentro de la plataforma software Vitis [36] de Xilinx. En SDAccel una aplicación se encuentra dividida entre la aplicación de servidor y los kernels que se ejecutan en el hardware, con canales de comunicación entre ellos. La comunicación entre un servidor y un dispositivo se realiza a través del bus PCIe. Mientras que la información de control se transfiere entre regiones de memoria especificas en el dispositivo, la memoria global se utiliza para transferir datos entre la aplicación servidor y los kernels.

En el servidor se ubica la parte de la aplicación que se encarga de habilitar la comunicación con las interfaces físicas del dispositivo, y con la región reconfigurable. Esto es lo que se conoce como plataforma software. La plataforma software está compuesta por los controladores para acceder a los dispositivos, y de todas las herramientas de síntesis y compiladores necesarias para generar los kernels de cómputo. En la Figura2.2 la plataforma software se corresponde con la parte de *SDAccel Development Environment*. La parte del servidor se compila utilizando un compilador estándar de C, por ejemplo GCC, haciendo uso del *runtime* de Xilinx.

Por otro lado, en la Figura2.2, las partes de *Vivado Design Suite* y *Active Board* se corresponden con la denominada plataforma hardware. *Vivado Design Suite* [38] es un entorno de desarrollo para la síntesis y análisis de diseños HDL desarrollado por Xilinx. SDAccel lleva integrada una versión optimizada de Vivado.

En la Figura2.2 la parte de Vivado representa el diseño de un DSA con los interfaces de todos los módulos configurados y conectados con los pines I/O del dispositivo, junto con la

región reconfigurable. El DSA también contiene la representación de la interfaz de la región reconfigurable. En la parte de *Active Board* se representa el dispositivo destino donde se ubica el DSA, y donde se ejecutan los kernels de cómputo. Los kernels de cómputo se compilan utilizando un compilador especifico que genera una representación binaria del diseño para un determinado dispositivo.

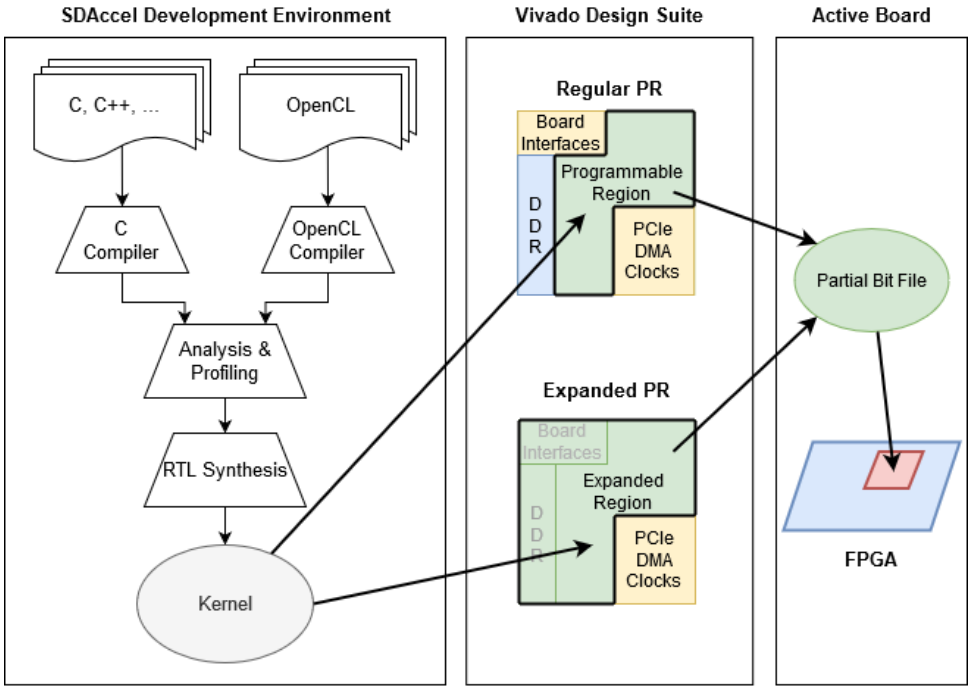

Figura 2.2: Diagrama de una aplicación en SDAccel.

Como se puede observar en la Figura2.2 existen dos tipos de reconfiguración parcial: Partial Reconfiguration (PR) y Expanded Partial Reconfiguration (XPR). El proceso de desarrollo del DSA varia según el tipo de reconfiguración parcial utilizado. El DSA descrito en este documento utiliza reconfiguración parcial expandida para cargar binarios compilados en el dispositivo mientras este continua activo y conectado con el servidor a través de PCIe. La reconfiguración parcial permite el cambio de forma dinámica de los módulos dentro de un diseño activo. Algunas de las ventajas de utilizar reconfiguración parcial son [28]:

- Reducción del tamaño de la FPGA utilizado para implementar una determinada arquitectura. Reducción de coste y consumo de potencia.
- Flexibilidad en la elección de los algoritmos o protocolos disponibles para una aplicación.
- Permite nuevas técnicas de diseño más seguras.
- Mejora la tolerancia a fallos de la FPGA.

Como se puede observar en la Figura2.3, cuando se utiliza PR se define una región reconfigurable correspondiente a la región lógica definida para los kernels de usuario, y otra región estática donde se ubica la lógica del diseño, como por ejemplo: controlador de memoria, relojes y resets, periféricos y controlador XDMA. Esto conlleva una ineficiencia en el uso de

parte del área del dispositivo, potencia y tiempo de ejecución cuando este no se está utilizando.

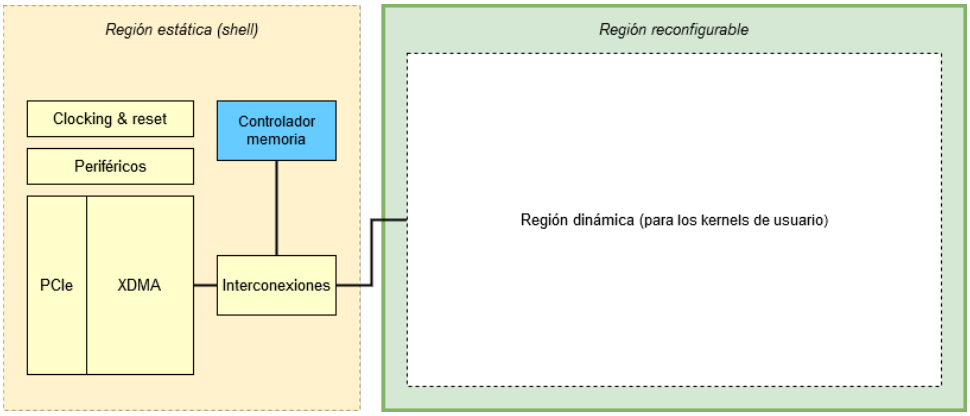

Figura 2.3: Diagrama de un DSA utilizando PR

.

Para solucionar las limitaciones de PR, surgió lo que se conoce como XPR [27]. En XPR la lógica del controlador de memoria y las interconexiones se ubican dentro de la región reconfigurable. Esta lógica, aunque esté ubicada en la región reconfigurable, es fija, es decir, no se ve modificada por el proceso de reconfiguración parcial. Sin embargo, las herramientas de implementación pueden elegir donde es más conveniente ubicar esta lógica, dentro de la región reconfigurable, para mejorar el rendimiento del kernel a ejecutar. De esta forma, se pueden aprovechar mejor los recursos disponibles para implementar funciones más complejas y con mayor flexibilidad. Por tanto, en lugar de sintetizar solo la región lógica definida para los kernels de cómputo, se sintetiza toda la región reconfigurable incluyendo la lógica del controlador de memoria y de las interconexiones. En la Figura2.4 se muestra el esquema de un DSA utilizando XPR.

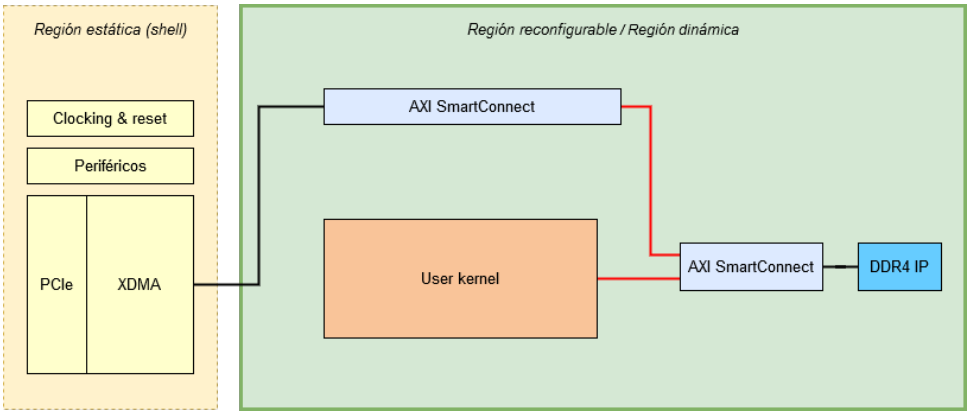

Figura 2.4: Diagrama de un DSA utilizando XPR.

En este proyecto, se quiere poder utilizar el DSA en las FPGAs de los nodos HN del sistema representado en la Figura1.2. En el capítulo 3 se describe el diseño del DSA para funcionar en una FPGA KU115 conectada a través de PCIe con un servidor, y con un módulo de memoria DDR4 de 2GB.

Los nodos HN del prototipo de MANGO, son clústers de 12 FPGAs conectados entre sí a

través de cables de interconexión. Las FPGAs se organizan en cuatro placas base. Las placas base proporcionan la infraestructura del sistema y la mayoría de hardware a nivel de sistema. En las placas base se incluyen las siguientes funcionalidades:

- Fuentes de alimentación.
- Generadores de reloj configurables y distribución de los relojes.
- Gestión del sistema mediante un interfaz I^2C.
- Soporte para JTAG.

En la Figura2.5 se muestra una placa base con cuatro ranuras para módulos FPGA. Al lado de las ranuras para las FPGAs hay varios conectores donde se puede conectar módulos adicionales. Estos módulos proporcionan funcionalidad hardware a los diseños que se estén ejecutando en las FPGAs. En estos conectores pueden conectarse, desde módulos de memoria, hasta módulos de PCIe y de interconexión.

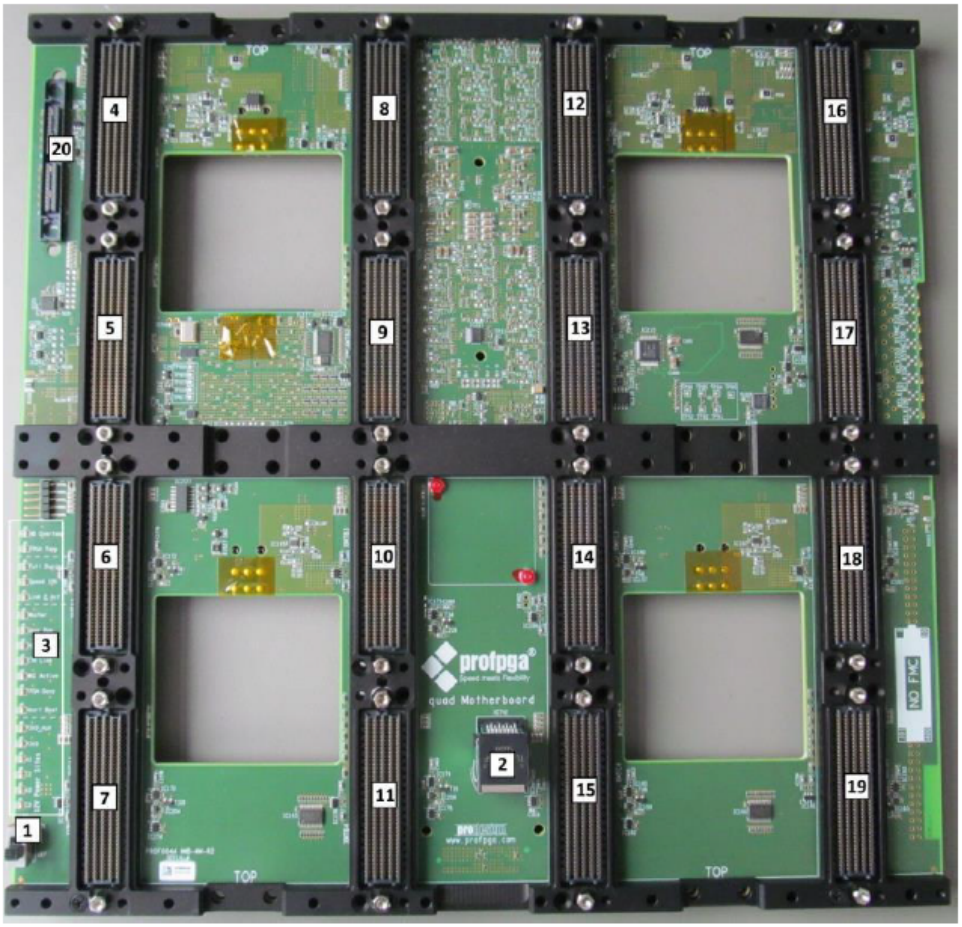

Figura 2.5: Placa base proporcionada por ProDesign con cuatro ranuras para FPGAs.

Para las conexiones entre las FPGAs de un clúster, se utilizan los pines I/O disponibles en los recursos de las FPGAs. Mientras que para conectar dos nodos HN entre sí, se utilizan los Multi-Gigabit Transceivers (MGT). En el capítulo 4 se describen estos recursos más en detalle.

Un clúster HN cuenta con una conexión PCIe Gen3 con un servidor. PCI Express (PCIe) [9] es un bus serie de alta velocidad de uso general utilizado para la interconexión de periféricos, interfaces chip a chip y como puente para otros protocolos. La versión 1.0 del bus se presentó en 2003, a fecha de hoy la última versión presentada es la 5.0, y se espera que la versión 6.0 este disponible entre 2021 y 2022. El servidor puede acceder a la FPGA con la que está conectado mediante Direct Memory Access (DMA) a través de PCIe. DMA [8], o en español acceso directo a memoria, permite que un cierto componente, en este caso el servidor, pueda acceder a la memoria del sistema, la FPGA, para leer o escribir en ella independientemente de la CPU. Para realizar transacciones DMA, es necesario que el servidor disponga del controlador apropiado. Xilinx proporciona varios controladores, para este proyecto se ha utilizado *xcldma* que permite, además de realizar transacciones DMA con una FPGA, controlar la región programable para configurar los kernels de OpenCL y controlar los periféricos de la región estática.

Además del controlador en la parte del servidor, también es necesario disponer de lógica en la FPGA para permitir DMA a través de PCIe con un servidor. En las Figuras 2.3 y 2.4 esta lógica se representa mediante el componente XDMA de la región estática y las interconexiones desde este hacia el controlador de memoria. Para estas interconexiones, y el resto de interconexiones del diseño, se utilizan interfaces AXI Memory Mapped y AXI4-Lite.

Advanced eXtensible Interface (AXI) es una interfaz de comunicación paralela de alto rendimiento, síncrona, de alta frecuencia, multi-máster, multi-slave, principalmente diseñada para las comunicaciones dentro de un chip [19]. AXI forma parte del ARM AMBA, una familia de buses introducida en 1996 por ARM [4]. La primera versión de AXI se incluyó en AMBA 3.0 en 2003. En la versión AMBA 4.0, de 2010, se incluyen las dos versiones principales de AXI, AXI4.

Hay tres tipos de interfaces AXI4:

- AXI4/AXI Memory Mapped: Para requisitos de asignación de memoria de alto rendimiento.
- AXI4-Lite: Utilizada para comunicaciones DMA simples y de bajo rendimiento. Por ejemplo, para registros de estado.
- AXI4-Stream: Para transmisión de datos de alta velocidad.

En el DSA se utiliza el interfaz AXI Memory Mapped como interfaz de datos, para acceder a la memoria de la FPGA desde el servidor, y el interfaz AXI4-Lite para acceder a los registros de estado y de control.

Cada clúster solo tiene una conexión PCIe, por tanto, dado que el resto de los dispositivos del clúster no tienen conexión PCIe con el servidor, y también para aprovechar las interconexiones entre los dispositivos, se ha modificado el diseño para soportar la comunicación entre FPGAs. De esta forma el servidor puede acceder a todas las FPGAs aunque no esté conectado a ellas directamente. El kernel de una FPGA maestro también puede acceder a la memoria de otra FPGA esclavo, de esta forma se aprovecha las interconexiones disponibles en el sistema y se puede mejorar el rendimiento de los kernels ejecutados en este. El soporte para la comunicación entre FPGAs se comenta en el capítulo 4.

Capítulo 3

Diseño de referencia

En este capítulo se describe el funcionamiento de un DSA para la plataforma XCKU115 de Xilinx [35]. Se ha partido de un diseño de referencia inicial desarrollado por Xilinx [31] que actualmente está descontinuado y sin soporte. El diseño original está desarrollado para las versiones de Vivado 2017.1, 2016.4 y 2016.3. En este proyecto se ha utilizado la versión de Vivado 2017.4, por tanto ha sido necesario portar el proyecto desde la versión 2017.1, a la 2017.4. La versión de Vivado utilizada corresponde con una versión optimizada bajo el entorno de desarrollo SDAccel. El diseño divide el área del dispositivo en dos regiones: una estática y otra reconfigurable. En la Figura3.1 se puede observar la estructura del diseño.

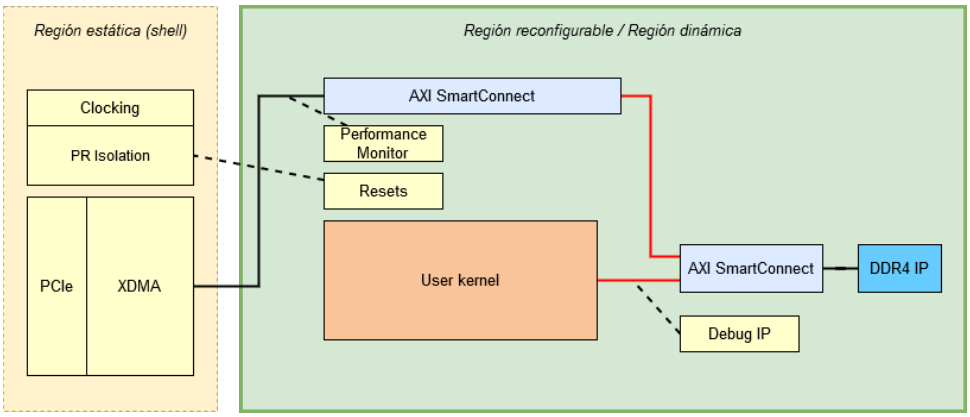

Figura 3.1: Esquema del DSA para la FPGA KU115.

El diseño original estaba pensado para funcionar con una configuración con cuatro controladores de memoria y soporte para cuatro kernels, entre otras características. Por tanto, para adaptar el diseño original al dispositivo descrito en el capítulo 2, utilizando la versión 2017.4 de Vivado bajo el entorno SDAccel, se han realizado las siguientes modificaciones:

- Portar el diseño desde la versión de Vivado 2017.1 a la 2017.4.
- Modificar el número de kernels que pueden ejecutarse simultáneamente de cuatro a uno.
- Disminuir el número de controladores de memoria de cuatro a uno, ya que solo se tiene

un módulo de DDR4 de 2GB. El diseño original tiene cuatro controladores de memoria para acceder a un total de 16GB.

- Eliminar la lógica encargada de programar la memoria flash a través de SPI.
- Eliminar la lógica encargada de controlar la velocidad del ventilador a través de I^2C.
- Modificar el módulo de depuración "Debug Bridge" para utilizar la versión disponible en Vivado 2017.4, y seguir las recomendaciones de uso descritas por Xilinx.
- Solucionar violaciones de tiempo al implementar con Vivado 2017.4.

El capítulo consta de dos secciones, cada una con varias subsecciones. En la sección 3.1 se comentan aquellas funciones del diseño que se ubican en la región estática y que no se ven afectadas cuando se realiza una reconfiguración parcial en el dispositivo. En la sección 3.2 se describen los bloques funcionales ubicados dentro de la región reconfigurable. En estas secciones se describen los diferentes bloques funcionales que forman el diseño del DSA teniendo en cuenta las modificaciones anteriormente mencionadas.

3.1. Región estática

El proyecto cuenta con una región estática donde se ubica la lógica necesaria para mantener el enlace entre el dispositivo y el servidor activo, y también el dispositivo operacional cuando se realiza una reconfiguración parcial. Las partes funcionales ubicadas en esta región son:

- Subsistema DMA para PCIe.
- Generación de relojes.
- Infraestructura para reconfiguración.

En la Figura3.2 se puede observar el esquema de la región estática, donde la línea continua que va desde el módulo XDMA a la región reconfigurable representa el interfaz AXI Memory Mapped, y la línea de puntos representa el interfaz AXI4-Lite.

Como se puede observar en la Figura3.3, que se corresponde con el diseño implementado, de toda el área del dispositivo la región estática (en amarillo) representa aproximadamente 8 % del total. Este área no puede utilizarse por los kernels de usuario. El resto del área del dispositivo corresponde con la región reconfigurable, y es donde se implementan los kernels de cómputo. En este ejemplo la lógica resaltada en verde se corresponde con el controlador de memoria, las interconexiones AXI y el resto de lógica de la región reconfigurable.

En los siguientes apartados se describen las unidades funcionales ubicadas en esta región.

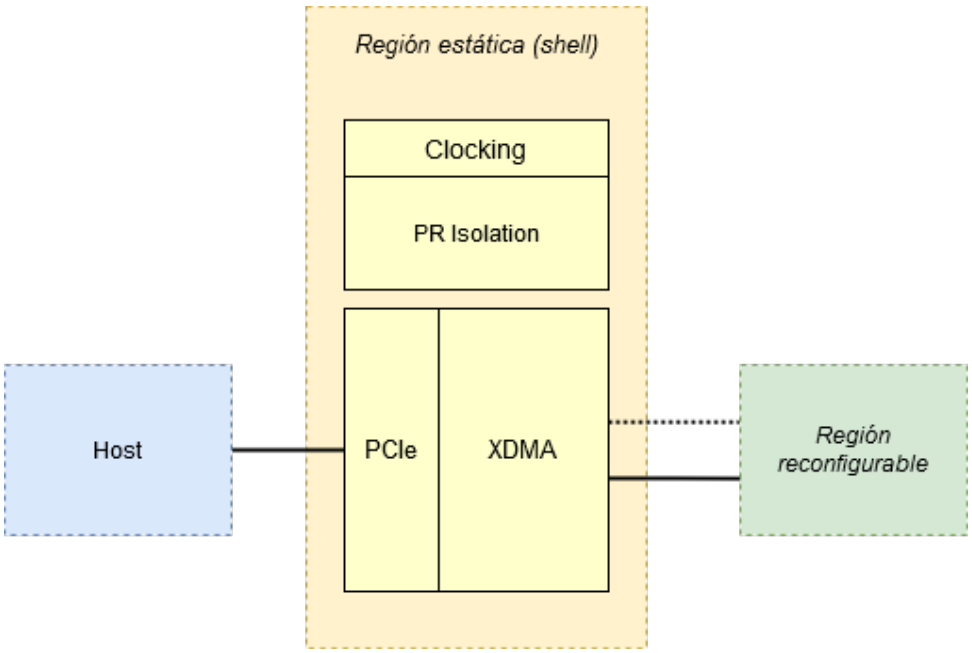

Figura 3.2: Esquema de la región estática.

3.1.1. Subsistema DMA para PCIe

El diseño tiene soporte para comunicación DMA con un servidor a través de PCIe mediante el módulo "DMA Subsystem for PCI Express" [24] proporcionado por Xilinx. Se ha decidido ubicar este módulo en la región estática para que el dispositivo no se vea afectado por el proceso de reconfiguración parcial, y permanezca activo y conectado al bus PCIe del servidor. De lo contrario después de cada proceso de reconfiguración del dispositivo, sería necesario reiniciar el servidor ya que en general es la Basic Input/Output System (BIOS) la lleva a cabo la enumeración de todos los dispositivos PCIe encontrados en el sistema en el inicio del mismo, no siendo posible una re-enumeración, en la mayoría de los casos, cuando el sistema ya se ha iniciado.

El módulo está configurado para permitir un ancho de banda de hasta 8GB/s teóricos. A través del enlace PCIe se obtiene una señal de reloj a una frecuencia de 100MHz de la que se derivan el resto de relojes que se utilizan para alimentar los diferentes módulos del diseño. En la sección 3.1.2 se describe en detalle la generación de relojes en el diseño.

Para la conectividad entre el dispositivo y el servidor se utiliza una interfaz AXI Memory Mapped funcionando a una frecuencia de 250MHz, un tamaño de datos de 256 bits y un tamaño de direcciones de 64 bits. Además, el módulo también está configurado para habilitar una interfaz AXI4-Lite a una frecuencia de 50 MHz. De este modo el servidor puede acceder hasta 4MB de recursos mapeados en memoria mediante el interfaz de control AXI4-Lite. El módulo está configurado para utilizar dos canales DMA de lectura y otros dos canales de escritura.

El servidor puede acceder a los recursos mapeados en memoria a través del interfaz AXI4-Lite, y a los 2GB de memoria a través del interfaz AXI Memory Mapped. En la Tabla 3.1 se muestran los rangos de direcciones que son accesibles desde el servidor.

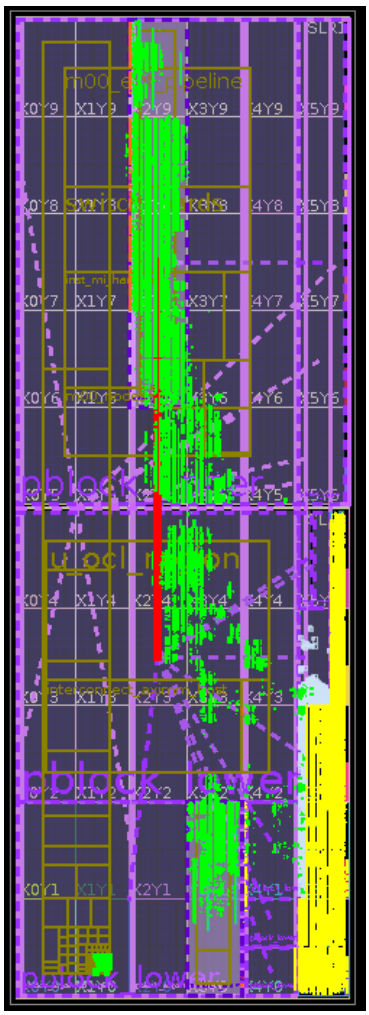

Figura 3.3: Vista del dispositivo con el diseño implementado, en amarillo la lógica de la región estática y en verde la lógica de la región reconfigurable.

Los rangos de direcciones de la Tabla 3.1 se comentan más en detalle durante el resto del capítulo.

3.1.2. Relojes

Como se ha mencionado en la sección 3.1.1, el diseño tiene una entrada de reloj a una frecuencia de 100MHz proporcionada por el módulo de PCIe conectado a la placa base. Esta señal de reloj se utiliza como referencia para derivar los dominios de reloj: AXI4-Lite Control, Kernel clock, Kernel clock 2 y AXI Performance Monitor.

Los dominios de reloj AXI4-Lite Control y AXI Performance Monitor son generados por una instancia de "Clocking Wizard IP" [22] utilizando Phase-Locked Loop (PLL). Estas señales funcionan a una frecuencia de 50 y 250 MHz, respectivamente. AXI4-Lite Control se utiliza en las rutas de control AXI4-Lite del proyecto, mientras que AXI Performance Monitor se utiliza para alimentar la lógica de *Profiling* disponible dentro de la región reconfigurable.

Interfaz AXI	Slave IP Core	Offset Address	Range	High Address
AXI Memory Mapped Data Interface	DDR4 Controller	0x0000_0000_0000_0000	2G	0x0000_0000_7FFF_FFFF
	Trace offload FIFO for Application profiling	0x0000_0020_0000_0000	2G	0x0000_0020_7FFF_FFFF
AXI4-Lite Control Interface	Controlador DDR4	0x0006_0000	128K	0x0007_FFFF
	Clock Wizard Kernel clock source	0x0005_1000	4K	0x0005_1FFF
	Clock Wizard Kernel 2 clock source	0x0005_0000	4K	0x0005_0FFF
	DDR4 calibration status	0x0003_2000	4K	0x0003_2FFF
	GPIO for PR Isolation	0x0003_0000	4K	0x0003_0FFF
	GPIO for Feature ID	0x0003_1000	4K	0x0003_1FFF
	OpenCL Region	0x0000_0000	128K	0x0001_FFFF
	AXI Performance Monitor	0x0010_0000	64K	0x0010_FFFF
	Trace offload FIFO for application profiling	0x0011_0000	4K	0x0011_0FFF

Tabla 3.1: Rango de direcciones accesibles desde el servidor.

Por otro lado, están las señales de reloj para alimentar los kernels de usuario: Kernel a 250 MHz y Kernel2 a 500 MHz. Estas señales son generadas por dos instancias de "Clocking Wizard IP" diferentes utilizando Mixed-Mode Clock Manager (MMCM). Al igual que PLL, MMCM es un recurso de la FPGA para la gestión de relojes, la principal diferencia es que el proceso de cambio de fase en PLL se realiza de forma analógica, mientras que en MMCM se hace de manera digital. La razón para utilizar MMCM en los relojes que alimentan los kernels, es para hacer uso de la característica disponible en la plataforma *SDAccel* que permite ajustar las frecuencias de reloj según los resultados de tiempos obtenidos durante la implementación del diseño. Los módulos cuentan con una interfaz AXI4-Lite por la que la plataforma SDAccel puede ajustar la frecuencia de los relojes a través del rango de direcciones descrito en la Tabla 3.1.

Por ejemplo, si la frecuencia de la lógica de un kernel está restringida al valor predeterminado de 250 MHz (período de 4 ns) pero finaliza el proceso de implementación con una violación de tiempo de -235ps WNS en rutas limitadas al dominio de reloj del kernel, SDAccel en tiempo de ejecución ajustará el generador de relojes MMCM para generar un reloj de 236MHz (4,235 ns) en su lugar, siempre y cuando no existan otras violaciones de tiempo en el resto de dominios de reloj.

3.1.3. Infraestructura para reconfiguración

Para aislar la región estática de la región reconfigurable cuando se realiza una reconfiguración parcial, la lógica de la región reconfigurable, así como la lógica en el límite de la región estática, deben estar en modo *reset*. Para conseguirlo se hace uso de una instancia del módulo "GPIO IP" [16]. Cuando se va a realizar una reconfiguración parcial, el controlador del dispositivo escribe en el registro *gate_pr* a través de AXI4-Lite utilizando el rango de direcciones indicado en la

Tabla 3.1. La escritura del registro *gate_pr* tiene los siguientes efectos:

- Los módulos de la región estática *psreset_regslice_data_pr* y *psreset_regslice_ctrl_pr* se mantiene en *reset*. El resultado es que las rutas XDMA de datos y de control en el limite de la regiones estática y reconfigurable se ponen en modo *reset*.
- Pone a *reset* la lógica de la región reconfigurable.

Una vez finalizado el proceso de reconfiguración parcial, el controlador borra el valor del registro *gate_pr* causando que las señales de *reset* que se habían activado en la región reconfigurable, y en la frontera de la región estática, se desactiven de forma sincronizada.

3.2. Región reconfigurable expandida

En la región reconfigurable se ubican los recursos disponibles para los kernels de cómputo. Como se puede observar en la Figura3.3, esta región representa la mayor parte del área del dispositivo. En esta región se encuentran los siguientes bloques funcionales:

- Soporte para ejecución de kernels de cálculo basados en OpenCL o HLS.
- Controlador de memoria.
- Lógica de *Profiling* y depuración.

En la Figura3.4 se muestra el esquema de esta región. En los siguientes apartados se describe en detalle cada uno de los bloques funcionales que componen esta región.

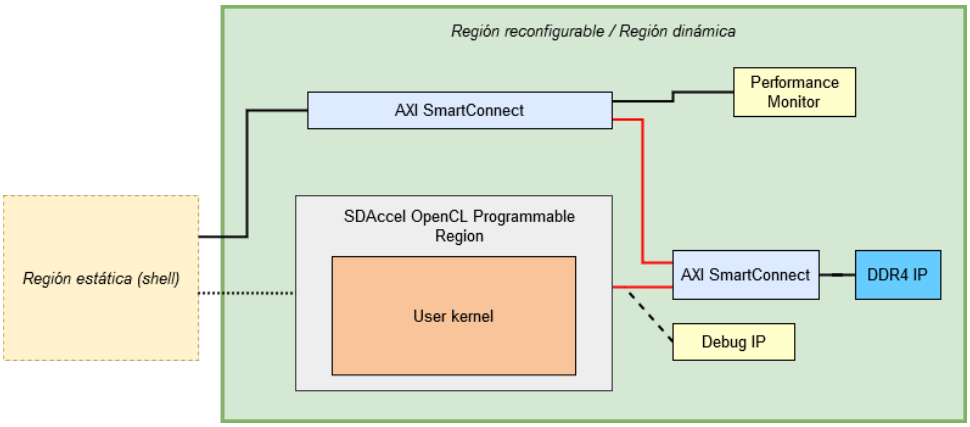

Figura 3.4: Esquema región reconfigurable.

3.2.1. Soporte para kernels OpenCL y HLS

Dentro de la región reconfigurable se encuentra la instancia del módulo "SDAccel OpenCL Programmable Region" que se corresponde con la región donde se programan los kernels de OpenCL o HLS a ejecutar. Este módulo establece una región programable lógica en el DSA resultante que el compilador de SDAccel reemplaza con los kernels definidos por el usuario y con las conexiones AXI correspondientes con el resto de la plataforma.

El módulo está configurado para soportar un kernel, sin embargo puede soportar hasta 16 en una sola FPGA. Dispone de una interfaz AXI4-Lite en modo esclavo para recibir comandos de control desde el servidor con el rango de direcciones que aparece en la Tabla 3.1. Además, dispone de un interfaz AXI Memory Mapped con un tamaño de datos de 512 bits que se conecta a través de un módulo "AXI SmartConnect" con el controlador de memoria DDR4 utilizando el rango de direcciones de la Tabla 3.2. El módulo dispone de una señal de reloj y una señal de reset síncronas al interfaz AXI4-Lite de control. Mientras que para los kernels cuenta con otras dos señales de reloj y reset. También se puede configurar el módulo para que disponga de un segundo dominio de reloj opcional el cual se puede utilizar solo con los kernels Register-Transfer Level (RTL) definidos por el usuario. Este segundo dominio ofrece a los usuarios la flexibilidad de ejecutar la lógica de los kernels RTL a una frecuencia distinta a la que funciona el primer dominio de reloj del kernel. Sin embargo, las rutas de datos deben ser síncronas al primer dominio de reloj tanto en los interfaces de entrada, como en los interfaces de salida.

Interfaz AXI	**Slave IP Core**	**Offset Address**	**Range**	**High Address**
AXI Memory Mapped Data Interface	Controlador DDR4	0x0000_0000_ 0000_0000	2G	0x0000_0000_ 7FFF_FFFF

Tabla 3.2: Rango de direcciones accesible por los kernels.

3.2.2. Controlador de memoria

El DSA tiene soporte para un único canal de memoria de 2GB. Como controlador de memoria se utiliza una instancia del módulo de Xilinx "DDR4 SDRAM (MIG)" [34].

El módulo está configurado para utilizar el componente EDY4016AABG-DR-F con un tamaño de datos de 512 bits y direcciones de 31 bits, permitiendo acceso a los 2GB del canal a través de una interfaz AXI Memory Mapped. Como se ha comentado en las secciones 3.1.1 y 3.2.1, tanto el servidor como el kernel tienen acceso a la memoria siguiendo la estructura de interconexiones AXI de la Figura3.5, y utilizando los rangos de direcciones indicados en las Tablas 3.1 y 3.2.

El controlador de memoria indica el estado de calibración a través del puerto de salida *c0_init_calib_complete*. De esta forma, el servidor puede saber cuando la memoria se encuentra operativa. El estado de esta señal es accesible por el controlador del dispositivo accediendo al módulo GPIO *ddr_calib_status* de la región estática a través del interfaz AXI4-Lite de control utilizando el rango de direcciones indicado en la Tabla 3.1.

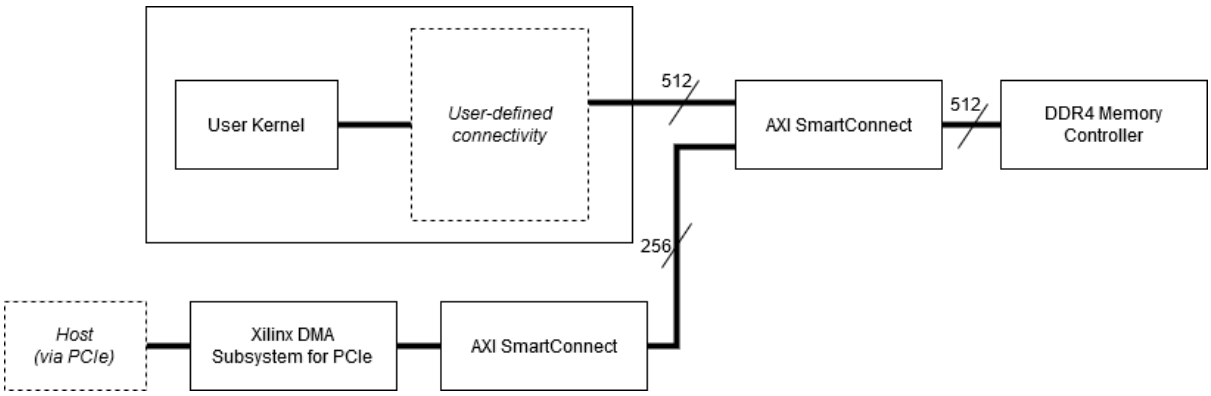

Figura 3.5: Interconexiones AXI Memory Mapped del sistema.

El controlador de memoria está ubicado en el Super Logic Region (SLR) superior del dispositivo KU115, alrededor de los pines de entrada y salida de alto rendimiento que utiliza. Dado que el controlador de memoria está ubicado en el SLR superior, es necesario una instancia del módulo "AXI Clock Converter" [17] para facilitar que las señales de control de baja velocidad crucen de un SLR a otro, convirtiendo de un dominio de reloj de 50MHz en el SLR inferior, a un dominio de reloj de 250MHz para el controlador de memoria en el SLR superior. El tiempo de penalización para aquellas rutas que cruzan de un SLR a otro puede ser significativo, más aún cuando se tienen rutas de datos AXI Memory Mapped por todo el diseño con tamaños de datos relativamente grandes y, en muchos casos, incluyendo rutas combinacionales. Por tanto, es imprescindible que aquellas rutas entre SLRs estén bien controladas y contengan el mínimo nivel de lógica posible. Para minimizar la penalización se ha seguido el esquema de particionado de la Figura3.6.

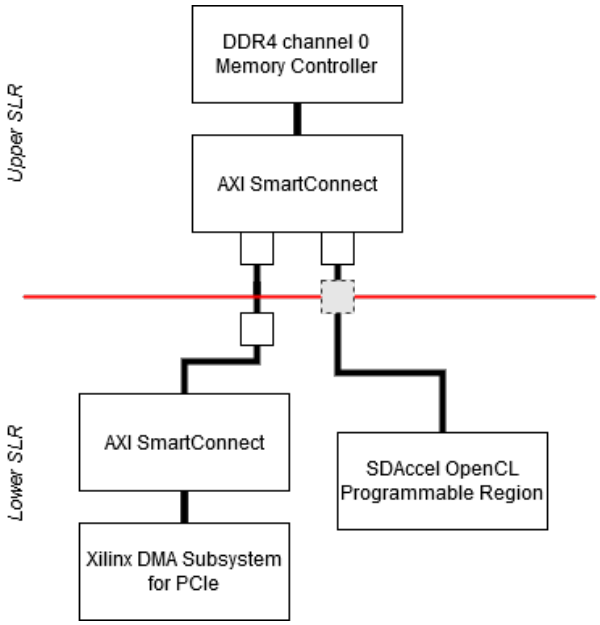

Figura 3.6: Particionado para controlar el cruce entre SLRs.

Como se puede observar en la Figura3.6, la instancia de "AXI SmartConnect" [20] que conecta la región programable y el servidor con el controlador de memoria está ubicada en el

SLR superior. Sin embargo, las etapas de *pipeline* de entrada de las interfaz maestro conectadas a la región programable (cuadrados grises de la figura), pueden ubicarse en cualquiera de los dos SLR según decidan las herramientas de implementación de Vivado que es más conveniente. Por otro lado, las etapas de *pipeline* conectadas al servidor (cuadrados blancos) están ubicadas en ambos SLR para facilitar el cruce entre SLRs.

La consecuencia de ubicar el controlador de memoria en la región reconfigurable, es que el contenido global de la memoria se pierde cuando se realiza una reconfiguración parcial.

3.2.3. *Profiling* y Depuración

El DSA ofrece soporte para *profiling*, permitiendo la monitorización y caracterización de las transacciones AXI realizadas. Para llevar a cabo el *profiling* se utiliza el módulo "AXI Performance Monitor" [18] configurado con dos interfaces de monitorización AXI Memory Mapped, una conectada al interfaz máster del módulo "SDAccel OpenCL Programmable Region" para la monitorización del kernel, y otra conectada al módulo de XDMA a través de un módulo "AXI Register Slice" para la monitorización del servidor. Los datos de perfíl recopilados se almacenan en una "FIFO AXI-Stream" [21] para su interpretación por la función de *profiling* del entorno SDAccel. Estos datos son accesibles a través del rango de direcciones indicado en la Tabla 3.1.

El diseño también tiene soporte para depuración a través del módulo "Debug Bridge" [23]. Este módulo establece el canal de comunicación entre la máquina servidor y los módulos de depuración dentro de la región reconfigurable, a través de la región estática. El módulo se tiene que instanciar dentro de la región reconfigurable con un interfaz BSCAN definido en el borde de esta región.

El módulo está configurado para utilizar el modo de funcionamiento *From BSCAN to DebugHub* utilizado para crear una instancia de "Debug Bridge" que debe ubicarse en cada módulo reconfigurable siguiendo el esquema de la Figura3.7. Las conexiones entre los componentes de depuración de la región estática y los de la región reconfigurable las realizan de forma automática las herramientas de Vivado.

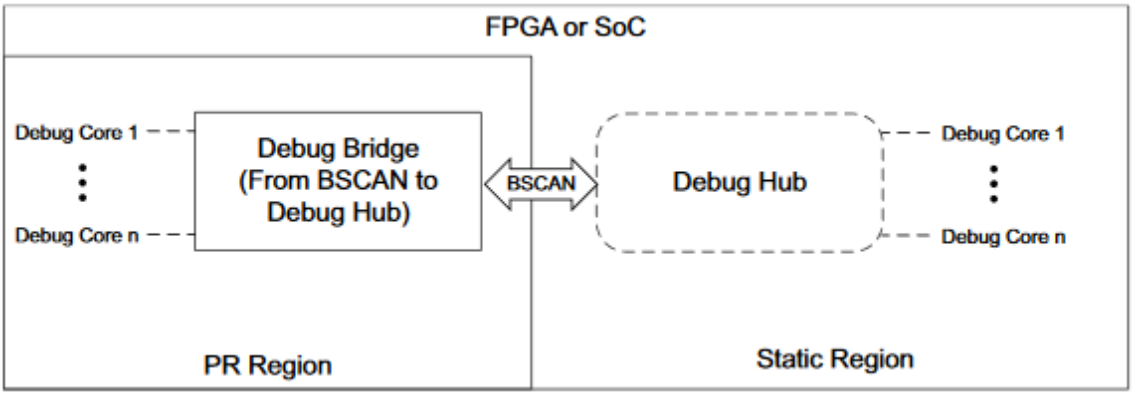

Figura 3.7: Debug Bridge configurado con el modo *From BSCAN to DebugHub*.

Capítulo 4

Soporte para comunicación entre FPGAs

En el capítulo 3 se ha descrito el funcionamiento de un DSA para la plataforma KU115 de Xilinx. Este DSA está pensado para funcionar en una FPGA conectada a través de PCIe con un servidor. Por tanto, para poder aprovechar el potencial que ofrece el sistema descrito en el capítulo 2, donde hay varias FPGAs interconectadas entre sí y solo una de ellas está conectada con el servidor a través de PCIe, es necesario modificar el diseño para añadir soporte para la comunicación entre FPGAs. Como resultado, también es necesario crear otro DSA para aquellas FPGAs del sistema sin conexión a través de PCIe con el servidor. En este capítulo se van a describir las modificaciones realizadas para permitir la comunicación entre dos FPGAs, una configurada en modo maestro (FPGA local) y otra configurada en modo esclavo (FPGA remota), cumpliendo con los objetivos indicados a continuación:

- Permitir transacciones AXI Memory Mapped desde un servidor hacia la memoria ubicada en la FPGA remota.
- Permitir el acceso mediante AXI4-Lite a los registros de estado y control de la FPGA remota.
- Permitir que el kernel de la FPGA maestro pueda realizar transacciones a la memoria de la FPGA remota a través de una interfaz AXI Memory Mapped.

En la Figura4.1 se muestra un diagrama del conexionado de las dos FPGAs, y las modificaciones realizadas en las regiones estáticas de cada una de ellas.

Para la comunicación entre FPGAs a través de una interfaz AXI Memory Mapped, Xilinx proporciona un módulo denominado Chip2Chip [15]. Este módulo tiene soporte para los protocolos: SelectIO y Aurora. En el apartado 4.1 se describe el módulo de Chip2Chip y los protocolos soportados por este. En el apartado 4.2 se comentan las modificaciones realizadas al proyecto descrito en el capítulo 3 para cumplir con los objetivos mencionados anteriormente. Por último, en el apartado 4.3 se comentan los resultados obtenidos de la

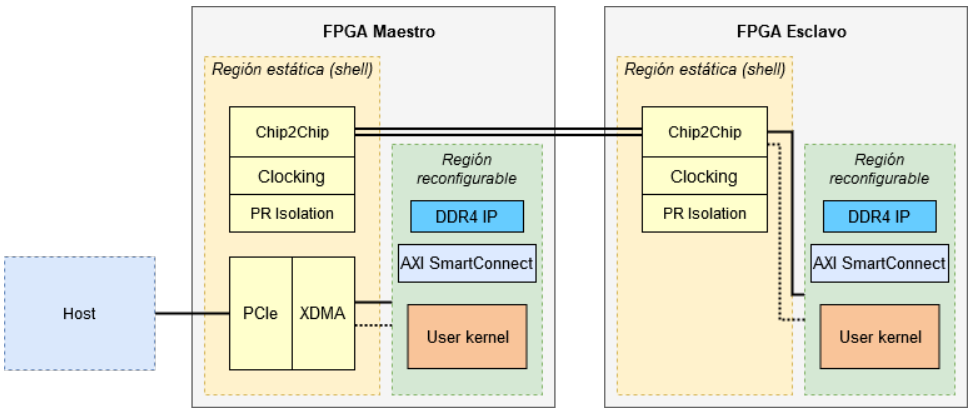

Figura 4.1: Diagrama de dos FPGAs conectadas a través de Chip2Chip.

validación del diseño y de la evaluación de prestaciones realizando transacciones DMA a las memorias de las dos FPGAs.

4.1. Chip2Chip

El módulo de Chip2Chip funciona como un puente conectando dos dispositivos a través de una interfaz AXI Memory Mapped, intercambiando transacciones de forma transparente según la especificación del protocolo AXI [15].

Además del interfaz AXI Memory Mapped, el módulo también tiene soporte para AXI4-Lite. Ambos interfaces pueden configurarse como Máster o Esclavo. El módulo permite seleccionar el tamaño de los datos, que puede ser de 32, 64 o 128 bits, este último solo en el caso de utilizar Aurora como protocolo de comunicación. El tamaño de las direcciones puede ser de 32 o 64 bits. También se puede configurar el tamaño del AXI ID y el tamaño del AXI WUSER. Los protocolos de comunicación soportados por el módulo son los siguientes:

- SelectIO SDR
- SelectIO DDR
- Aurora 8B/10B
- Aurora 64B/66B

Es importante que todos los parámetros de configuración sean iguales en el máster y en el esclavo. El módulo automáticamente determina el número de pines I/O, en el caso de SelectIO, o el número de *lanes*, en el caso de Aurora, que se van a utilizar según la configuración seleccionada. En la Tabla 4.1 se muestra el número de pines I/O que utiliza el módulo de Chip2Chip según la configuración. En el caso de Aurora el número de *lanes* depende del tamaño de datos: para un tamaño de datos de 32 bits se utiliza un *lane*, para un tamaño de datos de 64 bits se utilizan dos *lanes* y para un tamaño de datos de 128 bits se utilizan tres *lanes*.

AXI Data Width	AXI Address Width	Chip2Chip PHY Type	Chip2Chip PHY Width	Number of I/Os
32	32	SelectIO SDR	Compact 1-1	112
			Compact 2-1	62
			Compact 4-1	38
		SelectIO DDR	Compact 1-1	58
			Compact 2-1	32
			Compact 4-1	20
	64	SelectIO SDR	Compact 1-1	176
			Compact 2-1	94
			Compact 4-1	54
		SelectIO DDR	Compact 1-1	90
			Compact 2-1	48
			Compact 4-1	28
64	32	SelectIO SDR	Compact 2-1	90
			Compact 4-1	52
		SelectIO DDR	Compact 1-1	84
			Compact 2-1	46
			Compact 4-1	28
	64	SelectIO SDR	Compact 2-1	94
			Compact 4-1	54
		SelectIO DDR	Compact 1-1	90
			Compact 2-1	48
			Compact 4-1	28

Tabla 4.1: Número de pines I/O utilizados por Chip2Chip según la configuración.

En los siguientes apartados de esta sección se describen los protocolos soportados por el módulo de Chip2Chip.

4.1.1. SelectIO

Hoy en día las FPGAs tienen una gran cantidad de recursos para satisfacer un amplio abanico de casos de uso. Entre estos componentes se encuentran los pines de entrada y salida (pines I/O). Estos pines permiten la comunicación entre dispositivos con una latencia mínima. Los pines se agrupan en bancos de pines (en inglés *I/O banks*) de 52 pines cada uno, aunque algunos pocos que están etiquetados como bancos parciales, contienen solo 26 pines. De los 52 pines, 48 pueden utilizarse tanto como pares diferenciales como *single-ended*. Existen tres tipos de bancos de pines I/O con características y finalidades distintas. A continuación, se describen los tres tipos [33].

- Los grupos de pines High Performance (HP) están diseñados para cumplir con los requisitos de rendimiento de memorias de alta velocidad y otros interfaces chip a chip con voltajes de hasta 1.8V.
- Los grupos de pines High Range (HR) están diseñados para soportar un amplio rango de estándares I/O con voltajes de hasta 3.3V.

- Los grupos de pines High Density (HD) están diseñados para soportar interfaces de baja velocidad.

Cada dispositivo tiene una combinación diferente de bancos de pines, en el caso de la gama Ultrascale solo disponen de bancos HP y HR. El dispositivo XCKU115 tiene un total de 24 bancos de pines, de los cuales 20 son de alto rendimiento y 4 de alto rango. En la Figura4.2 se puede observar la distribución de los bancos en el área del dispositivo.

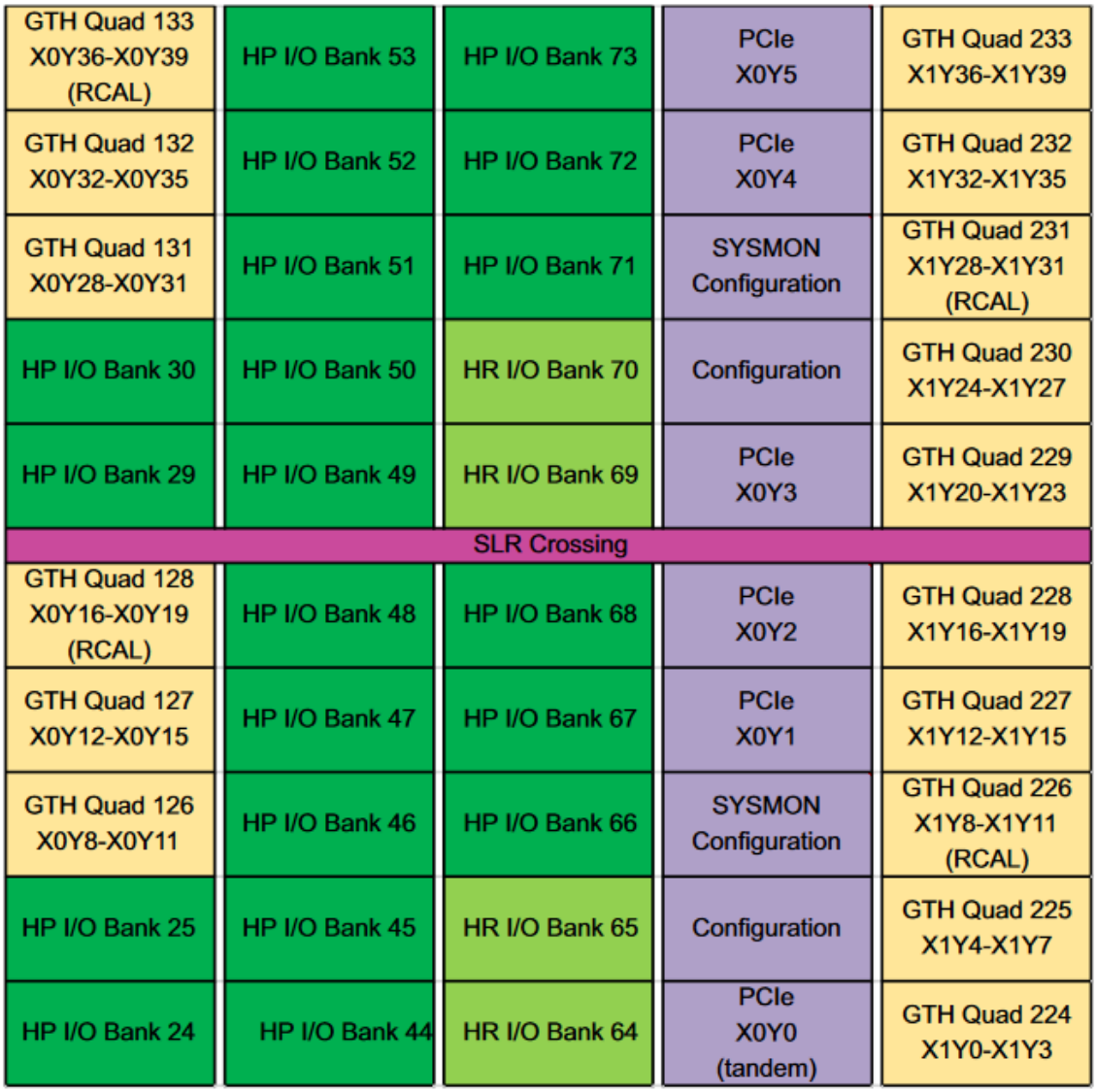

Figura 4.2: XCKU115 Banks [32].

SelectIO tiene dos modos de funcionamiento: Single Data Rate (SDR) y DDR. El funcionamiento de ambos es el mismo, la principal diferencia es que al seleccionar la opción DDR permite doblar la velocidad a la que funcionan los pines, sin afectar a la latencia ni al rendimiento.

El proceso de calibración de AXI Chip2Chip cuando se utiliza SelectIO es el siguiente:

- Cuando empieza la inicialización, el máster envía un número fijo de mensajes de ACK con un patrón concreto seguido por un patrón de calibración y espera la respuesta del esclavo. Al mismo tiempo, el esclavo envía mensajes con patrones de INIT. Una vez que el esclavo ve los mensajes de ACK desde el máster, confirma todos los mensajes de ACK y empieza el proceso de autocalibración. Si la autocalibración es satisfactoria, el esclavo envía patrones de ACK de confirmación.

- Después de recibir los patrones de ACK desde el esclavo, el máster envía patrones de NACK. Después de recibir los mensajes de NACK desde el máster, el esclavo envía mensajes de CALIBRATION.

- El máster continua enviando mensajes de NACK y entra en el proceso de autocalibración. Si la autocalibración es satisfactoria, envía mensajes de ACK para confirmarlo. El esclavo responde con mensajes de ACK, pasa al estado de transferencia de datos y pone a HIGH la señal de *link_status*

- Después de recibir los mensajes de ACK desde el esclavo, el máster pasa al estado de transferencia de datos y pone la señal de *Link_status* a HIGH.

Si durante el proceso de calibración para establecer el enlace ocurre algún error, tanto el máster como el esclavo lo indican a través de la señal llamada *multi_bit_error*.

SelectIO permite reducir el número de pines a utilizar, reduciendo también el ancho de banda. Esta opción se corresponde con la columna *Chip2Chip PHY Width* que aparece en la Tabla 4.1. En este diseño dado que se tiene una región estática con un número limitado de recursos, pero también se quiere obtener un ancho de banda aceptable, se ha optado por un tamaño de datos y direcciones de 64 bits, SelectIO DDR y Compact 2-1. Con esta configuración, acorde a lo mencionado en la Tabla 4.1 se necesitan un total de 48 pines.

4.1.2. Aurora

Aurora es un protocolo ligero de capa de enlace utilizado para transferencia de datos punto a punto a través de una o más líneas serie de alta velocidad. Existen dos variantes del protocolo: Aurora 8B/10B [13] y Aurora 64B/66B [11]. Las dos principales diferencias entre las dos variantes son:

- El tipo de codificación utilizado. Aurora 8B/10B como su nombre indica utiliza codificación 8B/10B, mientras que la otra variante utiliza codificación 64B/66B.

- La otra diferencia es el ancho de banda. Con Aurora 8B/10B el ancho de banda puede ir desde 480Mb/s hasta 84,48Gb/s, mientras que con Aurora 64B/66B el ancho de banda puede ir desde los 500Mb/s hasta 400Gb/s.

A diferencia de SelectIO, Aurora cuenta con una especificación donde se definen las características del protocolo. La especificación de Aurora define lo siguiente:

- Requisitos eléctricos.
- Capa Physical Medium Attachment (PMA).
- Subcapa física de codificación Physical Coding Sublayer (PCS).
- Control de flujo.
- Mecanismos de detección de errores.

Aurora hace uso de los llamados MGT, que son transceptores capaces de funcionar a velocidaddes superiores a 1Gb/s. Al igual que con los pines de I/O en SelectIO, existen varios tipos de transceptores. De todos los tipos de transceptores, Aurora hace uso de los GTX/GTH en los modelos Series 7 y de los GTH/GTY en los modelos Ultrascale y Ultrascale+. En el caso de los modelos Kintex Ultrascale, solo están disponibles los transceptores GTH. Como se puede observar en la figura 4.2 la FPGA KU115 dispone de 16 transceptores GTH. La principal diferencia entre los tipos de transceptores es el ancho de banda máximo que soportan. En la Tabla 4.2 se muestra el ancho de banda máximo de los transceptores soportados por Aurora.

	GTX	GTH	GTY
7 Series	12.5Gb/s	13.1Gb/s	X
UltraScale	X	16.3Gb/s	30.5Gb/s
UltraScale+	X	16.3Gb/s	32.75Gb/s

Tabla 4.2: Ancho de banda transceptores utilizados por Aurora.

Para utilizar Aurora con Chip2Chip es necesario un módulo adicional llamado Aurora 8B/10B [14], o Aurora 64B/66B [12] según la variante del protocolo seleccionado. Este módulo se encarga de toda la gestión del enlace cumpliendo con los requisitos descritos en la especificación del protocolo.

4.2. Modificaciones realizadas al diseño

Como ya se comentó al inicio del capítulo, se quiere añadir soporte para la comunicación entre FPGAs en el DSA. Para ello, además de modificar el diseño original, es necesario crear un nuevo proyecto sin módulo XDMA para aquellas FPGAs que no estén conectadas mediante PCIe a ningún servidor.

Para conseguir el objetivo de poder realizar transacciones de memoria entre FPGA a través de una interfaz AXI Memory Mapped, se ha utilizado el módulo de Xilinx Chip2Chip utilizando el protocolo SelectIO DDR. Para utilizar C2C junto con SelectIO DDR no es necesario ningún otro módulo adicional, no obstante si se quiere utilizar Aurora 64B/66B como protocolo de comunicación, es necesario un módulo adicional también proporcionado por Xilinx denominado Aurora 64B/66B. A continuación, se describen las modificaciones que se han realizado al diseño.

4.2.1. *Floorplanning*

Como se ha comentado en el capítulo 3, cuando se realiza una reconfiguración parcial la región reconfigurable está en modo *reset*. Por tanto, para evitar tener que restablecer el enlace entre dispositivos después de cada reconfiguración parcial, la lógica encargada de la comunicación entre FPGAs se ha ubicado dentro de la región estática. De este modo el enlace entre FPGAs no se ve afectado cuando se realiza el proceso de reconfiguración parcial. Como se puede observar en la Figura3.3, en el diseño original la región estática ocupa una parte pequeña de toda el área del dispositivo. Dado que se va añadir lógica adicional en la región estática, es necesario aumentar su tamaño no solo para aumentar el número de recursos de reloj necesarios para alimentar la lógica de comunicación, sino también para disponer de pines I/O de alto rendimiento en la región estática que puedan utilizarse por los módulos de comunicación. Modificar el tamaño de la región estática, y en consecuencia de la región reconfigurable, implica modificar el *floorplaning* del diseño original.

Floorplaning [26] es el proceso de elegir la mejor agrupación y conectividad para la lógica de un diseño, ubicando de forma manual los bloques lógicos en el área de la FPGA. El objetivo es aumentar la densidad, el rutado o el rendimiento, con la intención de reducir el retraso (*delay*) de las rutas al sugerir una mejor ubicación para la lógica seleccionada.

El proceso de *floorplaning* en el diseño original se realiza mediante el fichero XDC (fichero de *constraints*) con la creación de *pblocks* donde aquellos elementos del diseño con lógica relacionada se ubican es una región concreta del dispositivo destino. Una vez se crea un *pblock*, utilizando la orden *create_pblock*, este se rellena con los recursos necesarios y los elementos del diseño seleccionados. Para ubicar recursos dentro de un *pblock* se utiliza la orden *resize_pblock*, mientras que para añadir elementos del diseño se utiliza la orden *add_cells_to_pblock*. A continuación se muestra una parte del fichero de *constraints* donde se puede observar el uso de estas ordenes.

```
create_pblock pblock_expanded_region
add_cells_to_pblock [get_pblocks pblock_expanded_region]
    [get_cells [list xcl_design_i/expanded_region]]
resize_pblock [get_pblocks pblock_expanded_region] -add
    {PCIE_3_1_X0Y1:PCIE_3_1_X0Y5}
```

En la Figura4.3 se muestra el *floorplaning* modificado del diseño. Como se puede observar, comparando con las Figuras 3.3 y 4.2, los grupos de pines de alto rendimiento 44 y 45 ahora se encuentran dentro de la región estática. De este modo la región estática cuenta con bastantes recursos para poder utilizar tanto SelectIO como Aurora 64B/66B.

4.2.2. Relojes

El módulo de Chip2Chip se ha configurado con dominios de reloj independientes, es decir el interfaz AXI y los pines de entrada y salida de la FPGA están cada uno en un dominio de reloj único. El módulo se encarga de gestionar la sincronización entre los dominios de reloj. Los requisitos de reloj cuando se utiliza el modo independiente se pueden observar en la Figura4.4.

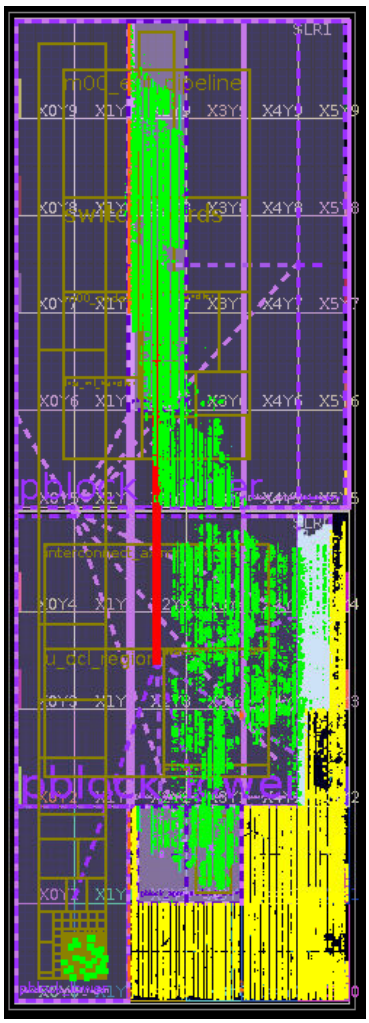

Figura 4.3: Diseño implementado con el *floorplaning* modificado, en amarillo la región estática y en verde la región reconfigurable.

Utilizando dominios de reloj independientes, el módulo de Chip2Chip configurado en modo maestro necesita cuatro entradas de reloj, en modo esclavo solo necesita tres. Las frecuencias a las que se han configurado las señales de reloj para alimentar el módulo se muestran a continuación:

- Señal de reloj del interfaz AXI Memory Mapped funcionando a una frecuencia de 240 MHz. Esta es la máxima frecuencia a la que puede funcionar este interfaz en una FPGA Kintex UltraScale. En el maestro el nombre de la señal es *s_aclk* mientras que en el esclavo es *m_aclk*.
- *axi_c2c_pgy_clk* funcionando a una frecuencia de 150MHz.
- *idelay_ref_clk* funcionando a una frecuencia de 200 MHz.
- Señal de reloj del interfaz AXI4-Lite funcionando a una frecuencia de 50MHz. En el maestro el nombre de la señal es *m_axi_lite_aclk* mientras que en el esclavo es *s_axi_lite_aclk*.

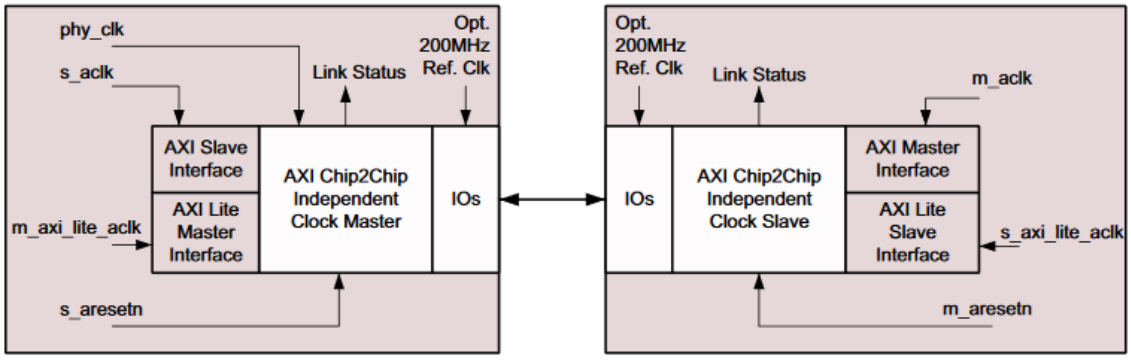

Figura 4.4: Relojes para SelectIO utilizando dominios de reloj independientes.

En el caso de Aurora, Chip2Chip solo se puede configurar con dominios de reloj independientes. Si se utiliza una FPGA de la gama UltraScale o UltraScale+, como es el caso de este proyecto, se debe de tener en cuenta lo siguiente:

- La señal *do_cc* debe dejarse desconectada.
- La señal de reloj *aurora_init_clk* debe ser la misma que la señal de reloj del módulo de Aurora *init_clk*.

En la Figura4.5 se muestran los requisitos de reloj si se utiliza Aurora como protocolo de comunicación.

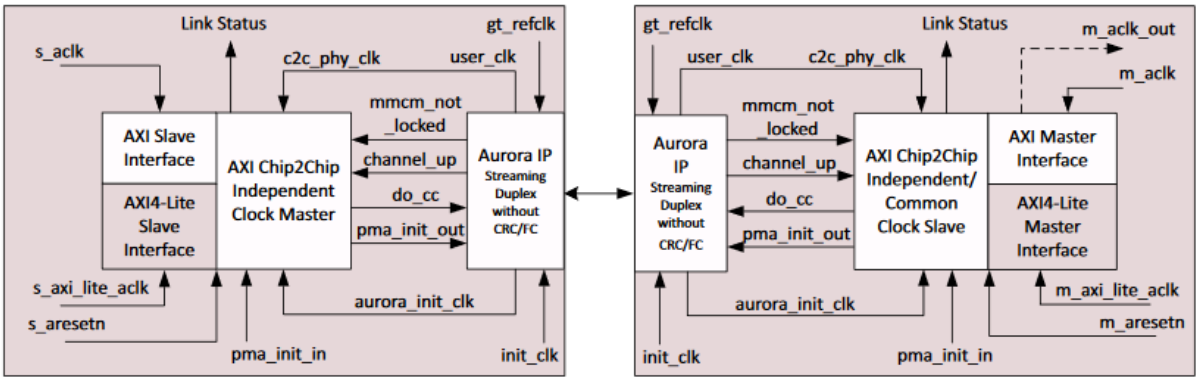

Figura 4.5: Requisitos de reloj de Aurora.

Además de las señales de reloj de las interfaces AXI Memory Mapped y AXI4-Lite del módulo de Chip2Chip, son necesarias otras dos señales de reloj para alimentar el módulo de Aurora: *init_clk* y *gt_refclk*.

Para alimentar los módulos de Chip2Chip tanto en el máster como en el esclavo, se ha modificado la lógica encargada de generar los relojes. El diseño original dispone de tres módulos "Clock Wizard" uno configurado para utilizar PLL y los otros dos configurados para utilizar MMCM. En el diseño modificado también se tienen tres módulos pero en lugar de dedicar los dos módulos MMCM para generar una señal de reloj cada uno para alimentar los kernels, se

tiene un único módulo MMCM para generar las dos señales de reloj “Kernel” y “Kernel2”. El otro módulo es el encargado de generar las señales de reloj necesarias para alimentar los módulos de comunicación entre FPGAs. Esto implica que el módulo encargado de generar las señales de reloj para alimentar los kernels es el encargado de ajustar las frecuencias dinámicamente según lo comentado en el apartado 3.1.2. Como resultado, ahora solo se tiene un rango de direcciones con el que la plataforma *SDAccel* puede ajustar la frecuencia de los relojes a través del interfaz AXI4-Lite del módulo. Este rango de direcciones se muestra en la Tabla 4.3.

4.2.3. Interconexiones AXI

Para cumplir el objetivo de que tanto el servidor, a través de XDMA, como el kernel en ejecución puedan realizar transacciones AXI con otra FPGA, se ha modificado el esquema de interconexiones AXI del diseño original descrito en la sección 3.2.2. Se ha añadido una instancia del IP core de Xilinx “AXI SmartConnect”. De está forma los dos interfaces AXI del host y del kernel están conectadas con el interfaz del módulo de Chip2Chip. El esquema de conexiones AXI con el módulo Chip2Chip configurado como maestro, y utilizando SelectIO como protocolo de comunicación, se muestra en la Figura4.6.

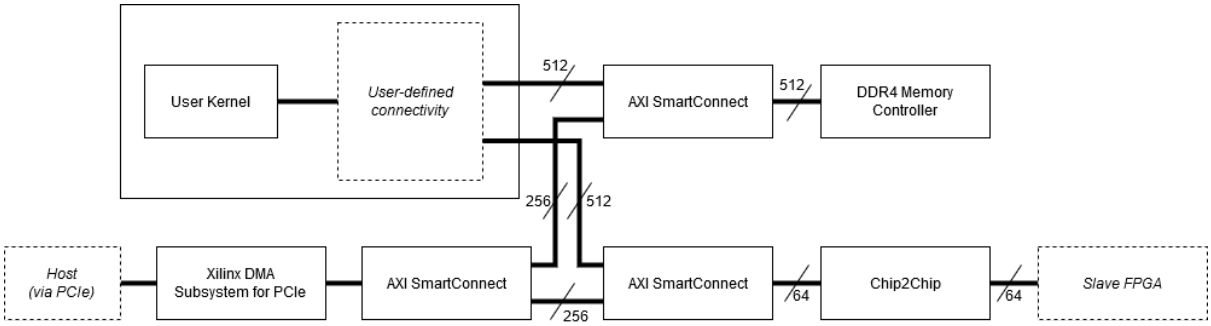

Figura 4.6: Interconexiones AXI con Chip2Chip como maestro y utilizando SelectIO.

En el caso de utilizar Aurora 64B/66B como protocolo de comunicación, el esquema de conexiones AXI es prácticamente igual, la diferencia es que se necesita el módulo Aurora 64B/66B. El esquema de interconexiones AXI si se utiliza Aurora se muestra en la Figura4.7.

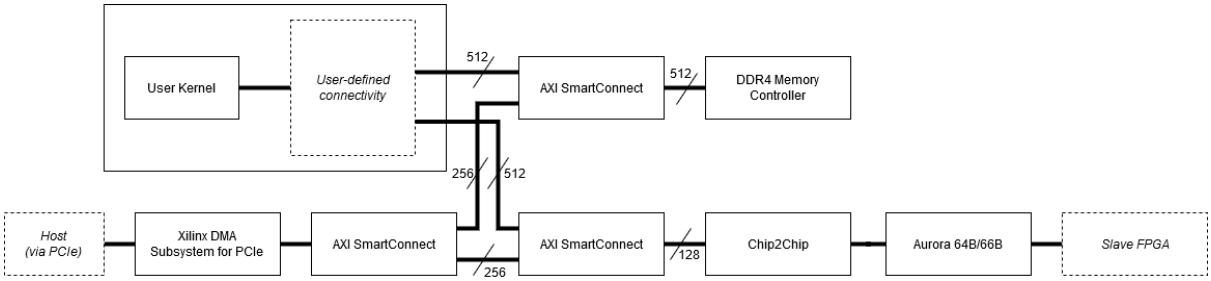

Figura 4.7: Interconexiones AXI con Chip2Chip como maestro y utilizando Aurora 64B/66B.

Teniendo en cuenta lo mencionado en la sección 3.1.3, para aislar la ruta de datos de la región estática del proceso de reconfiguración parcial se utiliza una instancia del módulo “Register Slice” ubicada en la frontera de la región estática. Este módulo cuando se realiza un proceso de reconfiguración parcial está en modo *reset*, de modo que el servidor no puede

acceder al controlador de memoria. Para que el proceso de reconfiguración parcial en la FPGA maestro no afecte el acceso desde el servidor a la memoria de la FPGA esclavo, el módulo "AXI SmartConnect" que conecta el interfaz AXI del módulo XDMA con el controlador de memoria y con el módulo de Chip2Chip, no debe verse afectado por el valor del registro "GPIO PR isolation". Sin embargo, el interfaz AXI del módulo que se conecta con el controlador de memoria, continua conectada al módulo "Register Slice" para mantener el aislamiento entre las dos regiones.

Para permitir que el kernel pueda acceder a la memoria de la FPGA esclavo, es necesario en el módulo "SDAccel OpenCL Programmable Region IP" modificar el campo *Master address width* de 31 bits a 32 bits indicando de está forma que se pueden direccionar hasta 4GB.

En el caso del diseño de la FPGA esclavo con el módulo de Chip2Chip en modo esclavo y sin conexión con el servidor a través de PCIe, el conexionado AXI es igual que en el diseño original, salvo que en lugar de tener el módulo de XDMA se tiene el módulo de Chip2Chip. En la Figura4.8 se pueden observar las interconexiones AXI de la FPGA esclavo cuando se utiliza SelectIO, mientras que en la Figura4.9 se muestran las interconexiones AXI utilizando Aurora 64B/66B.

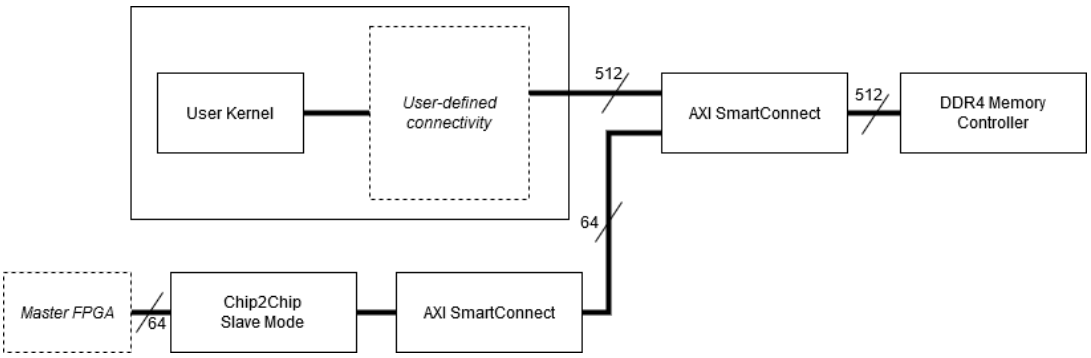

Figura 4.8: Interconexiones AXI en la FPGA esclavo utilizando SelectIO.

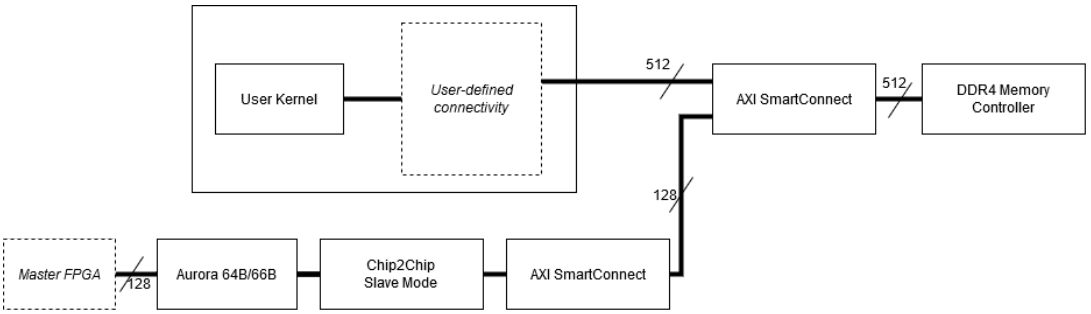

Figura 4.9: Interconexiones AXI en la FPGA esclavo utilizando Aurora 64B/66B.

Dado que ahora el servidor y el kernel pueden acceder a la memoria de otra FPGA, es necesario modificar el rango de direcciones del diseño original. En la Tabla 4.3 se muestran los rangos de direcciones accesibles desde el servidor y en la Tabla 4.4 los rangos de direcciones accesible desde el kernel.

Además, el servidor también puede acceder a los registros de control de la FPGA remota a través del rango de direcciones que se muestra en la Figura4.3. En este caso dado que los rangos

de direcciones en la FPGA remota son iguales, y se tienen los mismos registros de control, se le ha dado a Chip2Chip un rango de direcciones de 512KB para poder acceder a todos los registros de control.

Interfaz AXI	**Slave IP Core**	**Offset Address**	**Range**	**High Address**
AXI Memory Mapped Data Interface	Controlador DDR4	0x0000_0000_ 0000_0000	2G	0x0000_0000_ 7FFF_FFFF
	Chip2Chip	0x0000_0000_ 8000_0000	2G	0x0000_0000_ FFFF_FFFF
	Trace offload FIFO for Application profiling	0x0000_0020_ 0000_0000	2G	0x0000_0020_ 7FFF_FFFF
AXI4-Lite Control Interface	Controlador DDR4	0x0006_0000	128K	0x0007_FFFF
	Chip2Chip	0x8000_0000	512K	0x8007_FFFF
	Clock Wizard Kernel clock source	0x0005_1000	4K	0x0005_1FFF
	DDR4 calibration status	0x0003_2000	4K	0x0003_2FFF
	GPIO for PR Isolation	0x0003_0000	4K	0x0003_0FFF
	GPIO for Feature ID	0x0003_1000	4K	0x0003_1FFF
	Region OpenCL	0x0000_0000	128K	0x0001_FFFF
	AXI Performance Monitor	0x0010_0000	64K	0x0010_FFFF
	Trace offload FIFO for application profiling	0x0011_0000	4K	0x0011_0FFF

Tabla 4.3: Rango de direcciones accesibles desde el servidor.

Interfaz AXI	**Slave IP Core**	**Offset Address**	**Range**	**High Address**
AXI Memory Mapped Data Interface	Controlador DDR4	0x0000_0000_ 0000_0000	2G	0x0000_0000_ 7FFF_FFFF
	Chip2Chip	0x0000_0000_ 8000_0000	2G	0x0000_0000_ FFFF_FFFF

Tabla 4.4: Rango de direcciones accesibles por el kernel de cómputo.

En la FPGA esclavo el rango de direcciones para acceder al controlador de memoria es el mismo que el indicado en la FPGA maestro. De lo contrario, si se recibiera una dirección dentro del rango 0x80000000-0xFFFFFFFF el módulo de Chip2Chip no la reconocería y no llevaría a cabo la operación. Como el controlador de memoria está configurado con un tamaño de direcciones de 31 bits, cuando se recibe una transacción desde el servidor o desde el kernel de la FPGA maestro con dirección 0x80000000, al soportar direcciones de 31 bits, en realidad se está accediendo a la dirección 0x0000000 de la memoria de la FPGA esclavo.

4.3. Validación del diseño

4.3.1. Metodología

Para la validación y la evaluación de prestaciones se han utilizado las herramientas proporcionadas por Xilinx [40]: *dma_to_device* y *dma_from_device*. Como controlador PCIe se ha utilizado *xcldma*, tambien proporcionado por Xilinx junto al entorno de desarrollo SDAccel. Estas herramientas permiten realizar transacciones DMA de escritura en el caso de *dma_to_device*, y de lectura en el caso de *dma_from_device*. A continuación se muestra un ejemplo de como utilizar estas herramientas para escribir y leer a la memoria de la FPGA local.

```
./dma_to_device -d /dev/xcldma0_h2c_0 -f test1.bin -s 1024 -a 0000 -c 1 -v

./dma_from_device -d /dev/xcldma0_c2h_1 -f output1.bin -s 1024 -a 0000 -c 1 -v
```

Y para realizar las mismas transacciones pero con la memoria de la FPGA remota, se utilizan las mismas ordenes cambiando la dirección.

```
./dma_to_device -d /dev/xcldma0_h2c_0 -f test1.bin -s 1024 -a 2147483648 -c 1 -v

./dma_from_device -d /dev/xcldma0_c2h_1 -f output1.bin -s 1024 -a 2147483648 -c 1 -v
```

Con la opción "-d" se indica el canal que se quiere utilizar. Tal como se ha mencionado en el capítulo 3 existen dos canales de escritura y dos canales de lectura. Con la opción "-f" se indica el fichero con los datos que se escribirán en memoria, en el caso de realizar una escritura, o el fichero donde se guardarán los datos leídos de memoria, en el caso de realizar una lectura. La opción "-s" permite especificar el tamaño de los datos, mientras que la opción "-a" se indica a partir de que dirección de memoria se escribirán o leerán los datos, estos dos valores hay que indicarlos en decimal. La opción "-c" permite repetir la misma transacción el número de veces indicado. Y, finalmente, la opción "-v" muestra información sobre la ejecución de la herramienta.

Estas herramientas una vez finalizada su ejecución devuelven una estimación del ancho de banda. El ancho de banda se calcula como la división entre el número de bytes de la transacción y el tiempo que ha transcurrido hasta completarla. Si se indica que la transacción debe de repetirse varias veces, el resultado del ancho de banda es la media de todas las transacciones realizadas.

En este caso se ha obtenido el ancho de banda de escritura y de lectura tanto en la FPGA local (conectada con el servidor a través de PCIe) como de la FPGA remota (conectada a la FPGA local a través de C2C). Se han utilizado distintas tamaños de datos para observar a partir de que tamaño de datos se obtiene el ancho de banda máximo. Para cada tamaño se han realizado 10 transacciones y se ha obtenido el ancho de banda medio de todos ellos.

4.3.2. Resultados

Para realizar las pruebas con la FPGA remota se ha utilizado SelectIO como protocolo de comunicación, configurado con un tamaño de datos de 64 bits y una frecuencia de enlace de 150MHz. En el caso del módulo de PCIe, está configurado para ofrecer un ancho de banda máximo teórico de 8GB/s. Sin embargo, en el momento de realizar las pruebas, en lugar de negociar las ocho líneas PCIe, solo se han podido negociar cuatro. Por tanto, el ancho de banda máximo teórico a la hora de hacer las pruebas es de 4GB/s. Las pruebas se han realizado utilizando el primer canal de lectura y el primer canal de escritura. La razón de realizar las pruebas con solo un canal es que los resultados van a ser prácticamente los mismos, es decir, no hay ninguna diferencia entre utilizar un canal u otro.

En las Figura4.10 se muestra el ancho de banda de lectura obtenido para la FPGA local, conectada a través de PCIe con el servidor (línea azul), y la FPGA remota (línea naranja). En la Figura4.11 se muestra el ancho de banda de escritura para las dos FPGAs.

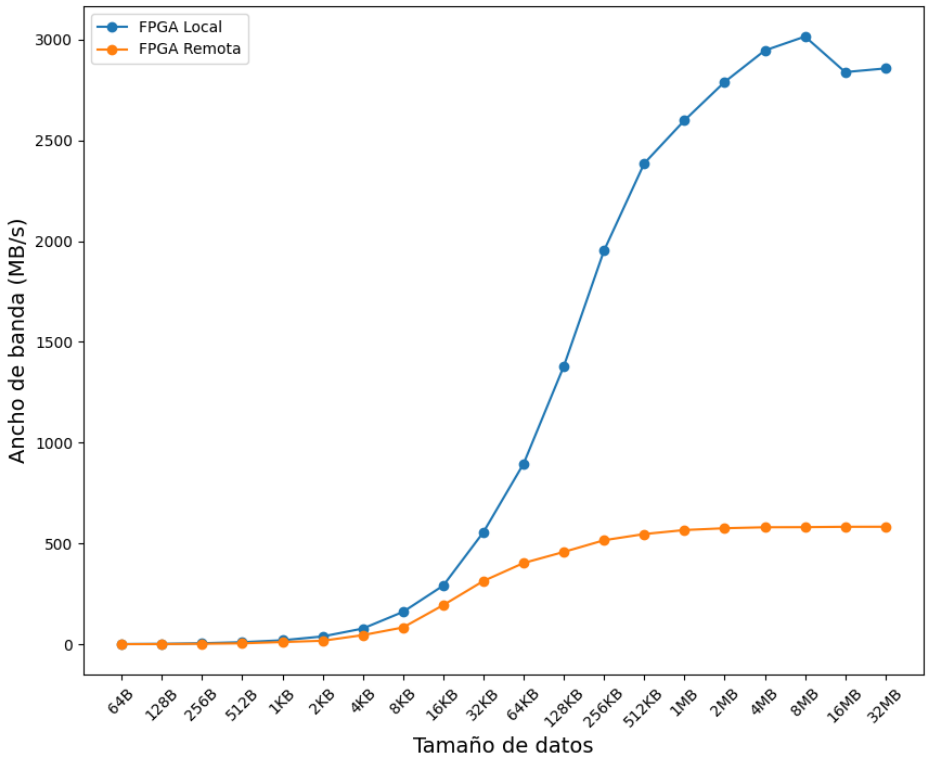

Figura 4.10: Ancho de banda de lectura.

Como se puede observar, en el caso de la FPGA local el ancho de banda máximo obtenido está alrededor de los 3000MB/s. La razón de que el ancho de banda obtenido sea menor del ancho de banda teórico puede deberse a alguna de las siguientes motivos:

- El controlador funciona utilizando interrupciones. Xilinx recomienda utilizar *polling* para

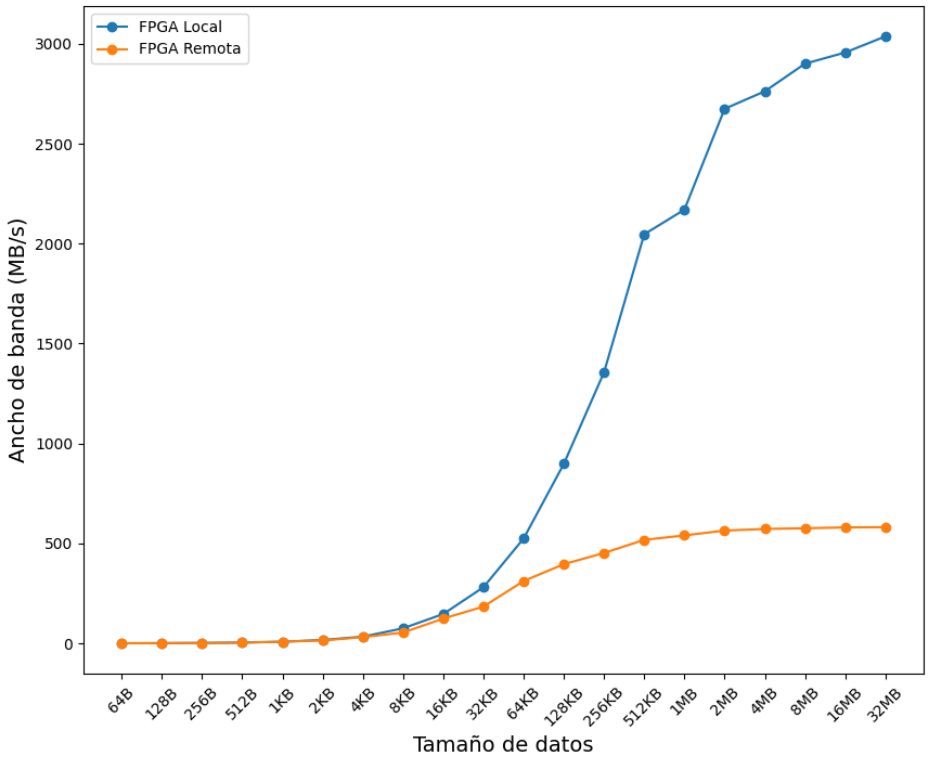

Figura 4.11: Ancho de banda de escritura.

aumentar el rendimiento.

- Un número limitado de canales DMA.
- El *Maximum Payload Size* es de 512 bytes. Este tamaño debería ser lo más grande posible.

En el caso de la FPGA remota el ancho de banda máximo obtenido está alrededor de los 580MB/s, en este caso el enlace entre FPGAs es el factor que limita el ancho de banda. Para aumentar este ancho de banda se puede aumentar la frecuencia a la que funciona el enlace, el máximo permitido por el módulo de Chip2Chip es de 400MHz, o se puede aumentar el número de pines utilizados. En este proyecto, debido al limitado número de recursos disponibles dentro de la región estática, lo más interesante sería estudiar cual es la mejora de rendimiento a medida que se va aumentando la frecuencia del enlace. Esto se deja como trabajo futuro.

Capítulo 5

Conclusiones

Durante los últimos años la popularidad de las FPGAs en entornos de HPC ha ido aumentando, y cada vez es más común ver estos dispositivos en estos sistemas. Las principales razones que han propiciado este fenómeno, son el alto rendimiento que ofrecen en aplicaciones paralelas junto con un consumo de energía reducido comparado al de una GPU. No obstante, aunque las FPGAs ofrecen muchas ventajas, también son muy complejas de programar y exigen un conocimiento muy elevado sobre la arquitectura. Como resultado, hay un esfuerzo por parte de los fabricantes de FPGAs, Xilinx e Intel/Altera, para facilitar el uso de estos dispositivos, habilitando el uso de paradigmas de programación como OpenCL y lenguajes de programación como C o C++. En el caso de Xilinx a través de la plataforma de desarrollo SDAccel, hoy en día integrada dentro del entorno de desarrollo Vitis.

En este documento se ha descrito un DSA para la FPGA Kintex UltraScale KU115 de Xilinx, habilitado para utilizar XPR y para dar soporte para OpenCL. Este proyecto está pensado para utilizarse con los dispositivos de un nodo HN del prototipo de MANGO. Para aprovechar las interconexiones entre los dispositivos dentro de un nodo, se ha modificado el diseño del DSA para dar soporte para la comunicación entre FPGAs. De este modo el servidor puede acceder a través de DMA a aquellos dispositivos con los que no está conectado directamente, y también ofrece la posibilidad de que los kernels de cómputo en ejecución en dos dispositivos distintos puedan comunicarse entre sí, mejorando de está forma el rendimiento.

El diseño presentado en este documento puede ampliarse para aumentar el número de conexiones entre FPGAs siguiendo una topología dada, mejorar el ancho de banda de los enlaces, e incluso habilitar el proceso de reconfiguración parcial a través de los enlaces de interconexión utilizando el módulo “Dynamic Function eXchange Controller” [25].

Bibliografía

[1] Mentor Graphics. *Questa Advanced Simulator*. URL: https://www.mentor.com/products/fv/questa/. (última consulta: 13/5/2020).

[2] Khronos Group. *Khronos Group web page*. URL: https://www.khronos.org.

[3] Khronos Group. *OpenCL*. URL: https://www.khronos.org/opencl/.

[4] ARM Holding. *ARM Official web page*. URL: https://www.arm.com/. (última consulta: 27/6/2020).

[5] *MANGO project*. URL: http://www.mango-project.eu/.

[6] *ProDesign*. URL: https://www.prodesign-europe.com/.

[7] *SDAccel product page*. URL: https://www.xilinx.com/products/design-tools/software-zone/sdaccel.html. (última consulta: 27/8/2020).

[8] Wikipedia. *DMA*. URL: https://es.wikipedia.org/wiki/Acceso_directo_a_memoria. (última consulta: 22/8/2020).

[9] Wikipedia. *PCI Express*. URL: https://es.wikipedia.org/wiki/PCI_Express. (última consulta: 22/8/2020).

[10] Asic Worls. *PLI - How it Works*. URL: http://www.asic-world.com/verilog/pli1.html#How_it_Works. (última consulta: 2/5/2020).

[11] Xilinx. *Aurora 64B/66B Protocol Specification*. URL: https://www.xilinx.com/support/documentation/ip_documentation/aurora_64b66b_protocol_spec_sp011.pdf. (última consulta: 20/8/2020).

[12] Xilinx. *Aurora 64B/66B v11.2*. URL: https://www.xilinx.com/support/documentation/ip_documentation/aurora_64b66b/v11_2/pg074-aurora-64b66b.pdf. (última consulta: 25/8/2020).

[13] Xilinx. *Aurora 8B/10B Protocol Specification*. URL: https://www.xilinx.com/support/documentation/ip_documentation/aurora_8b10b_protocol_spec_sp002.pdf. (última consulta: 20/5/2020).

[14] Xilinx. *Aurora 8B/10B v11.1*. URL: https://www.xilinx.com/support/documentation/ip_documentation/aurora_8b10b/v11_1/pg046-aurora-8b10b.pdf. (última consulta: 20/5/2020).

[15] Xilinx. *AXI Chip2Chip v5.0*. URL: https://www.xilinx.com/support/documentation/ip_documentation/axi_chip2chip/v5_0/pg067-axi-chip2chip.pdf. (última consulta: 27/8/2020).

[16] Xilinx. *AXI GPIO*. URL: https://www.xilinx.com/support/documentation/ip_documentation/axi_gpio/v2_0/pg144-axi-gpio.pdf. (última consulta: 2/7/2020).

[17] Xilinx. *AXI Interconnect v2.1*. El módulo AXI Clock Converter se documenta en está guía. URL: https://www.xilinx.com/support/documentation/ip_documentation/axi_interconnect/v2_1/pg059-axi-interconnect.pdf. (última consulta: 17/7/2020).

[18] Xilinx. *AXI Performance Monitor v5.0*. URL: https://www.xilinx.com/support/documentation/ip_documentation/axi_perf_mon/v5_0/pg037_axi_perf_mon.pdf. (última consulta: 3/7/2020).

[19] Xilinx. *AXI reference guide*. URL: https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug1037-vivado-axi-reference-guide.pdf. (última consulta: 22/8/2020).

[20] Xilinx. *AXI SmartConnect v1.0*. URL: https://www.xilinx.com/support/documentation/ip_documentation/smartconnect/v1_0/pg247-smartconnect.pdf. (última consulta: 17/7/2020).

[21] Xilinx. *AXI4-Stream FIFO v4.1*. URL: https://www.xilinx.com/support/documentation/ip_documentation/axi_fifo_mm_s/v4_1/pg080-axi-fifo-mm-s.pdf. (última consulta: 3/7/2020).

[22] Xilinx. *Clock Wizard*. URL: https://www.xilinx.com/support/documentation/ip_documentation/clk_wiz/v5_4/pg065-clk-wiz.pdf. (última consulta: 18/7/2020).

[23] Xilinx. *Debug Bridge v3.1*. URL: https://www.xilinx.com/support/documentation/ip_documentation/debug_bridge/v3_0/pg245-debug-bridge.pdf. (última consulta: 12/7/2020).

[24] Xilinx. *DMA/Bridge Subsystem for PCI Express*. URL: https://www.xilinx.com/support/documentation/ip_documentation/xdma/v4_1/pg195-pcie-dma.pdf. (última consulta: 20/7/2020).

[25] Xilinx. *Dynamic Function eXchange Controller v1.0*. URL: https://www.xilinx.com/support/documentation/ip_documentation/dfx_controller/v1_0/pg374-dfx-controller.pdf. (última consulta: 28/8/2020).

[26] Xilinx. *Floorplanning Methodology Guide*. URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx12_4/Floorplanning_Methodology_Guide.pdf. (última consulta: 21/7/2020).

[27] Xilinx. *Methodologies for creating reconfigurable application*. URL: https://www.xilinx.com/publications/events/developer-forum/2018-frankfurt/designing-for-acceleration-methodologies-for-creating-reconfigurable-applications.pdf. (última consulta: 22/8/2020).

[28] Xilinx. *Partial Reconfiguration*. URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_4/ug909-vivado-partial-reconfiguration.pdf. (última consulta: 22/8/2020).

[29] Xilinx. *Release Notes Vivado 2017.4*. URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_4/ug973-vivado-release-notes-install-license.pdf. (última consulta: 15/7/2020).

[30] Xilinx. *SDAccel documentation*. URL: https://www.xilinx.com/html_docs/xilinx2017_4/sdaccel_doc/. (última consulta: 27/6/2020).

[31] Xilinx. *SDAccel KU115 project*. URL: https://github.com/zakinder/SDAccel/tree/master/KU115BOARD_SDAccel. (última consulta: 27/6/2020).

[32] Xilinx. *UltraScale and UltraScale+ FPGAs Packaging and Pinouts*. Pag. 76. URL: https://www.xilinx.com/support/documentation/user_guides/ug575-ultrascale-pkg-pinout.pdf. (última consulta: 27/8/2020).

[33] Xilinx. *UltraScale Architecture SelectIO Resources*. URL: https://www.xilinx.com/support/documentation/user_guides/ug571-ultrascale-selectio.pdf. (última consulta: 20/7/2020).

[34] Xilinx. *UltraScale Architecture-Based FPGAs Memory IP*. URL: https://www.xilinx.com/support/documentation/ip_documentation/ultrascale_memory_ip/v1_4/pg150-ultrascale-memory-ip.pdf. (última consulta: 15/7/2020).

[35] Xilinx. *UltraScale Datasheet*. URL: https://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf. (última consulta: 22/8/2020).

[36] Xilinx. *Vitis web page*. URL: https://www.xilinx.com/products/design-tools/vitis/vitis-platform.html. (última consulta: 27/6/2020).

[37] Xilinx. *Vivado Design Suite User Guide*. URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_4/ug900-vivado-logic-simulation.pdf. (última consulta: 15/7/2020).

[38] Xilinx. *Vivado Design Suite web page*. URL: https://www.xilinx.com/products/design-tools/vivado.html. (última consulta: 27/6/2020).

[39] Xilinx. *What is an FPGA?* URL: https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html. (última consulta: 27/6/2020).

[40] Xilinx. *Xilinx PCIe DMA drivers*. URL: https://github.com/Xilinx/dma_ip_drivers/tree/master/XDMA/linux-kernel. (última consulta: 28/8/2020).

Capítulo 6

Anexos

6.1. QuestaSim

QuestaSim es un simulador hardware desarrollado por Mentor Graphics [1] que combina alto rendimiento con avanzadas capacidades de depuración y cobertura funcional para el mejor soporte nativo de Verilog, SystemVerilog, VHDL, SystemC, SVA, UPF y UVM.

La simulación hardware es el proceso de emular el comportamiento de un diseño real en un entorno software. Este proceso ayuda a verificar la funcionalidad de un diseño mediante la inyección de estímulos y observando el resultado del diseño.

El entorno de desarrollo Vivado de Xilinx tiene soporte para varios simuladores desarrollados por otras compañías. Entre los simuladores soportados por Vivado, están QuestaSim y ModelSim, ambos desarrollados por Mentor Graphics. Por defecto Vivado está configurado para utilizar el simulador que viene integrado con el entorno, llamado Vivado Simulator. Sin embargo, este simulador es menos potente y tiene menos funcionalidades que los simuladores mencionados anteriormente. Por ejemplo, Vivado Simulator no tiene soporte para Verilog Programming Language Interface (PLI). PLI es un mecanismo para llamar a funciones de C o C++ desde un código Verilog [10]. Por tanto, para proyectos de mayor tamaño y complejidad es recomendable utilizar algunos de los simuladores de terceros soportados por Vivado.

La simulación utilizando simuladores de terceros se puede realizar tanto desde el propio entorno de Vivado, como utilizando un entorno de simulación externo. En este anexo se describe el proceso para simular el comportamiento de un diseño desarrollado con Vivado, utilizando el entorno de simulación QuestaSim. Para integrar QuestaSim, u otro de los simuladores soportados, en Vivado, Xilinx proporciona toda la información pertinente en el siguiente documento [37].

Antes de hacer nada, es imprescindible comprobar la versión de QuestaSim soportada por la versión de Vivado que se está utilizando. Por ejemplo, en el caso de la versión de Vivado 2017.4, según la documentación proporcionada por Xilinx [29], la versión de QuestaSim soportada

corresponde con la **10.6b**. Además, también es importante el sistema operativo que se está utilizando, así como su versión. Si no se utilizan las versiones recomendadas por Xilinx, es probable que a la hora de compilar las bibliotecas de simulación aparezcan errores.

Una vez se han comprobado las versiones de Vivado, el sistema operativo indicado y de QuestaSim, es necesario compilar las bibliotecas de simulación de Vivado. Estas bibliotecas contienen los dispositivos, y los modelos de comportamiento y temporización de cada IP core. Las bibliotecas de simulación se pueden compilar dentro del entorno de Vivado seleccionado la opción **Tools** → **Compile Simulation Libraries**. Aparecerá un diálogo como el de la Figura6.1. En el diálogo es necesario rellenar los campos según la configuración que se esté utilizando, en el documento [37] se describen cada una de las opciones.

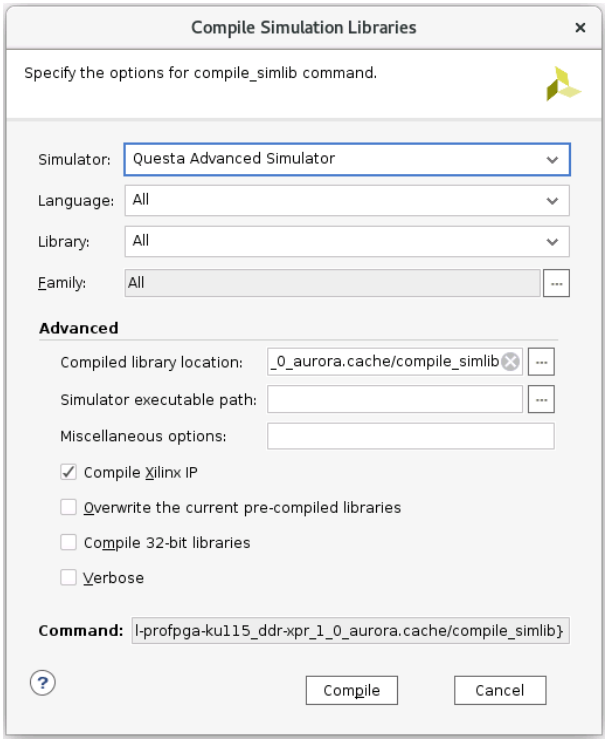

Figura 6.1: Diálogo para compilar las bibliotecas de simulación.

Finalizado el proceso de compilación de las bibliotecas, es necesario exportar el proyecto para su simulación a través de la opción de Vivado **File** → **Export** → **Export Simulation**. Se abrirá un diálogo como el de la Figura6.2 donde se puede seleccionar para que simulador se quiere exportar el proyecto.

Una vez se tienen las bibliotecas de simulación compiladas y el proyecto exportado para QuestaSim, es recomendable crear un script para automatizar la simulación del proyecto. A continuación se muestra el ejemplo de un script utilizado para lanzar la simulación de un proyecto en QuestaSim.

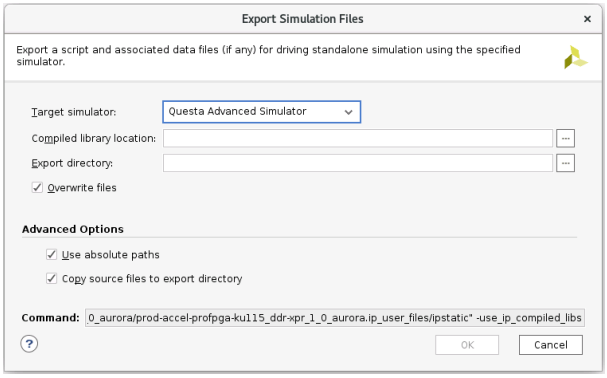

Figura 6.2: Diálogo para exportar el proyecto como simulación.

```
#!/bin/bash

# break on error
set -e

# setting path
export XILINX_LIB_PATH='/opt/xilinx_lib'

echo "Launching simulation with following path"
echo "XILINX_LIB_PATH=$XILINX_LIB_PATH"
echo ""
sleep 2

# check if path to xilinx Vivado installation is set
if  [[ -z "$XILINX_VIVADO" ]]; then
    echo "Path to Xilinx Vivado installation (\$XILINX_VIVADO) not set
                - terminating!"
    exit 1
else
    echo "Found Xilinx Vivado installation at $XILINX_VIVADO"
    export XILINX_PATH=$XILINX_VIVADO/data
fi

# check if path to Mentor QuestaSim installation is set
if [[ -z "$QUESTASIM_PATH" ]]; then
        if which vsim ; then
                echo "Found Modelsim/Questasim installation"
                export QUESTASIM_PATH=`which vsim | sed -e "s#/bin/vsim##"`
        else
                echo "Path to Mentor Questasim installation (\$QUESTASIM_PATH) not set
                - terminating!"
    exit 1
        fi
fi
```

```
# check if path to precompiled Xilinx libraries is set
if [[ -z "$XILINX_LIB_PATH" ]]; then
        echo "Path to precompiled Xilinx libraries (\$XILINX_LIB_PATH) not set
                - terminating!"
        exit 1
fi

# default settings: Virtex7 architecture and verilog language
export FPGA_ARCHITECTURE='XV7S'
export FPGA_LANGUAGE='verilog'
export FPGA_SIM='gui'


#####
# Create folder structure and map libraries
#####

# create directory for questasim libraries
mkdir -p libs

# remove local modelsim.ini file
rm -f modelsim.ini

# create questasim work library
vlib libs/work
vlib libs/msim

# map work library
vmap work libs/work

# map xilinx libraries
vmap unisim          $XILINX_LIB_PATH/unisim
vmap unisims_ver     $XILINX_LIB_PATH/unisims_ver
vmap secureip        $XILINX_LIB_PATH/secureip
vmap unimacro_ver    $XILINX_LIB_PATH/unimacro_ver

vlib libs/msim/xil_defaultlib
vlib libs/msim/xpm
vlib libs/msim/xlconstant_v1_1_3
vlib libs/msim/util_ds_buf_v2_01_a

# map libraries
vmap xil_defaultlib       libs/msim/xil_defaultlib
vmap xpm                  libs/msim/xpm
vmap xlconstant_v1_1_3           libs/msim/xlconstant_v1_1_3
vmap util_ds_buf_v2_01_a  libs/msim/util_ds_buf_v2_01_a


vlog -work xil_defaultlib -ccflags "-DHDL_SIM -DMMI64_HAS_NOW
```

```
        -D_SHORT_BURST_ -I /opt/prodesign/profpga/proFPGA-2018B/include"
        ~/demo_designs/mmi64_upstream/source/sw/mmi64_upstream_test.c


## common files
#
vlog -work xil_defaultlib -64 -sv "+incdir+srcs/incl" "+incdir+srcs/incl" \
"/nfs/drodagu/tfm/questa/srcs/xpm_cdc.sv" \
"/nfs/drodagu/tfm/questa/srcs/xpm_memory.sv" \

vcom -work xpm -64 -93 \
"/nfs/drodagu/tfm/questa/srcs/xpm_VCOMP.vhd" \

vlog -work xil_defaultlib -64 "+incdir+srcs/incl" "+incdir+srcs/incl" \
"srcs/pattern_checker.v" \
"srcs/pattern_generator.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_aurora_lane.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_rx_startup_fsm.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_tx_startup_fsm.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_support.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_gt_common_wrapper.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_support_reset_logic.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_clock_module.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_axi_to_drp.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_standard_cc_module.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_reset_logic.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_cdc_sync.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_core.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_axi_to_ll.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_block_sync_sm.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_common_reset_cbcc.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_common_logic_cbcc.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_cbcc_gtx_6466.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_channel_err_detect.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_channel_init_sm.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_ch_bond_code_gen.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_64b66b_descrambler.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_err_detect.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_global_logic.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_wrapper.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_lane_init_sm.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_ll_to_axi.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_multi_wrapper.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_rx_stream_datapath.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_rx_stream.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_width_conversion.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_64b66b_scrambler.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_sym_dec.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_sym_gen.v" \
```

```
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_gtx.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_tx_stream_control_sm.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_tx_stream_datapath.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0_tx_stream.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_aurora_64b66b_0_0.v" \

vlog -work xlconstant_v1_1_3 -64 "+incdir+srcs/incl" "+incdir+srcs/incl" \
"srcs/ip/design_1/xlconstant_v1_1_3/xlconstant_v1_1_vl_rfs.v" \

vlog -work xil_defaultlib -64 "+incdir+srcs/incl" "+incdir+srcs/incl" \
"srcs/ip/design_1/xil_defaultlib/design_1_xlconstant_0_0.v" \

vcom -work util_ds_buf_v2_01_a -64 -93 \
"srcs/ip/design_1/util_ds_buf_v2_01_a/util_ds_buf.vhd" \

vcom -work xil_defaultlib -64 -93 \
"srcs/ip/design_1/xil_defaultlib/design_1_util_ds_buf_0_0.vhd" \
"srcs/ip/design_1/xil_defaultlib/design_1_util_ds_buf_1_0.vhd" \

vlog -work xil_defaultlib -64 "+incdir+srcs/incl" "+incdir+srcs/incl" \
"srcs/ip/design_1/xil_defaultlib/design_1_pattern_checker_0_0.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_xlconstant_1_0.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_xlconstant_2_0.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_xlconstant_3_0.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_xlconstant_4_0.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_xlconstant_5_0.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_pattern_generator_0_0.v" \
"srcs/ip/design_1/xil_defaultlib/ethernet_crc.v" \
"srcs/ip/design_1/xil_defaultlib/generic_dpram_separated_ports.v" \
"srcs/ip/design_1/xil_defaultlib/mmi64_buffer.v" \
"srcs/ip/design_1/xil_defaultlib/mmi64_buffer_uni.v" \
"srcs/ip/design_1/xil_defaultlib/mmi64_deserializer.v" \
"srcs/ip/design_1/xil_defaultlib/mmi64_identify.v" \
"srcs/ip/design_1/xil_defaultlib/mmi64_m_devzero.v" \
"srcs/ip/design_1/xil_defaultlib/mmi64_m_regif.v" \
"srcs/ip/design_1/xil_defaultlib/mmi64_p_muxdemux.v" \
"srcs/ip/design_1/xil_defaultlib/mmi64_p_muxdemux_ctrl_fsm.v" \
"srcs/ip/design_1/xil_defaultlib/mmi64_p_muxdemux_demux.v" \
"srcs/ip/design_1/xil_defaultlib/mmi64_p_muxdemux_mux.v" \
"srcs/ip/design_1/xil_defaultlib/mmi64_regfifo.v" \
"srcs/ip/design_1/xil_defaultlib/mmi64_router.v" \
"srcs/ip/design_1/xil_defaultlib/mmi64_router_core.v" \
"srcs/ip/design_1/xil_defaultlib/mmi64_router_upstream.v" \
"srcs/ip/design_1/xil_defaultlib/mmi64_rt.v" \
"srcs/ip/design_1/xil_defaultlib/profpga_sync_ipad.v" \
"srcs/ip/design_1/xil_defaultlib/profpga_sync_opad.v" \
"srcs/ip/design_1/xil_defaultlib/profpga_sync_rx2.v" \
"srcs/ip/design_1/xil_defaultlib/profpga_sync_tx.v" \
"srcs/ip/design_1/xil_defaultlib/timer.v" \
```

```
"srcs/ip/design_1/xil_defaultlib/profpga_ctrl.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_profpga_ctrl_1_0.v" \
"srcs/ip/design_1/xil_defaultlib/afifo_core.v" \
"srcs/ip/design_1/xil_defaultlib/generic_dpram.v" \
"srcs/ip/design_1/xil_defaultlib/mmi64_identify.v" \
"srcs/ip/design_1/xil_defaultlib/mmi64_m_upstreamif.v" \
"srcs/ip/design_1/xil_defaultlib/design_1_mmi64_m_upstreamif_1_0.v" \
"srcs/ip/design_1/xil_defaultlib/design_1.v" \
"srcs/design_1_wrapper.v" \
"srcs/ethernet_crc.v" \
"srcs/generic_dpram_separated_ports.v" \
"srcs/mmi64_buffer.v" \
"srcs/mmi64_buffer_uni.v" \
"srcs/mmi64_deserializer.v" \
"srcs/mmi64_identify.v" \
"srcs/mmi64_m_devzero.v" \
"srcs/mmi64_m_regif.v" \
"srcs/mmi64_p_muxdemux.v" \
"srcs/mmi64_p_muxdemux_ctrl_fsm.v" \
"srcs/mmi64_p_muxdemux_demux.v" \
"srcs/mmi64_p_muxdemux_mux.v" \
"srcs/mmi64_p_pli.v" \
"srcs/mmi64_regfifo.v" \
"srcs/mmi64_router.v" \
"srcs/mmi64_router_core.v" \
"srcs/mmi64_router_upstream.v" \
"srcs/mmi64_rt.v" \
"srcs/profpga_mb.v" \
"srcs/profpga_sync_opad.v" \
"srcs/profpga_sync_tx.v" \
"srcs/timer.v" \
"srcs/aurora_tb.v" \

vlog -work xil_defaultlib \
"glbl.v"

vopt -64 +acc -l elaborate.log -L xil_defaultlib -L xpm -L xlconstant_v1_1_3
        -L util_ds_buf_v2_01_a -L unisims_ver -L unimacro_ver -L secureip
        -work xil_defaultlib xil_defaultlib.aurora_tb xil_defaultlib.glbl
        -o aurora_tb_opt

echo "Starting simulation for $FPGA_ARCHITECTURE architecture"

vsim -64 -voptargs="+acc" -gDEVICE=${FPGA_ARCHITECTURE} -t 1ps
        -L unisims_ver -L secureip -L xil_defaultlib
        -ldflags "-Wl,--whole-archive ${PROFPGA}/lib/linux_x86_64/libmmi64pli.a
        -Wl,--no-whole-archive ~/lib/linux_x86_64/libprofpga.a
        ~/lib/linux_x86_64/libconfig.a" -lib xil_defaultlib aurora_tb_opt
        -do "set NumericStdNoWarnings 1; set StdArithNoWarnings 1;
```

```
do wave.do; run -all" -l simulate.log
```

Para más información respecto a QuestaSim revisar la documentación proporcionada por Mentor.