Universidad Autónoma de Madrid

Escuela politécnica superior

Proyecto fin de carrera

Sistema basado en FPGA para la reproducción de tráfico en redes 10 Gigabit Ethernet.

Miguel Cubillo Llanes

Julio 2014

Sistema basado en FPGA para la reproducción de tráfico en redes 10 Gigabit Ethernet

AUTOR: Miguel Cubillo Llanes
TUTOR: Jaime Jesús Garnica Bertrán
PONENTE: Sergio López Buedo

High Performance Computing and Networking Research Group
Dpto. de Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio de 2014

Resumen

Resumen

El aumento de la penetración del uso de Internet, del uso de las redes sociales y el streaming de video y audio, ha hecho necesario aumentar el ancho de banda del que disponen los usuarios finales, provocando un incremento considerable del tráfico tcp/ip que tienen que soportar los primeros niveles de agregación. A la par que se aumenta el ancho de banda de las redes, es necesario desarrollar e investigar nuevos sistemas de procesamiento de paquetes que sean capaces de trabajar a estas tasas de línea.

Generar tráfico que refleje adecuadamente diferentes condiciones y topologías de red, es crítico para realizar experimentos válidos sobre testbeds de red, por lo que la generación de tráfico real es necesaria para poder comprobar el funcionamiento real de cualquier sistema de procesamiento de paquetes como por ejemplo routers, firewalls, sistemas IDS, etc. La manera más fiable de generar tráfico real es mediante la reproducción de tráfico previamente capturado.

El objetivo de este proyecto es el diseño, desarrollo e implementación de una arquitectura de reproducción de tráfico. Esta arquitectura permitirá retransmitir tráfico previamente capturado y almacenado en un fichero PCAP. Posibilitará reproducir dicha captura de tráfico con exactamente el mismo tiempo entre paquetes con el que fue capturado, así como aplicar distribuciones de tiempos entre paquetes según el caso o escenario que se quiera probar.

Para su diseño, se ha planteado una solución híbrida basada en la plataforma NetCOPE. Se va a implementar un software de control y un firmware que se cargará en la FPGA Virtex-5 que proporciona la tarjeta COMBO v2. El software de control proporcionará al usuario una serie de opciones que le permitirán cargar un fichero de trazas, cargar unos tiempos entre paquetes y modificar de manera interactiva los tiempos entre paquetes cargados.

Palabras Clave

FPGA, generador de tráfico, NetCOPE, Ethernet, 10 GbE, testbeds de red, replicación.

Abstract

Abstract

The increasing use of the Internet, social networking and new video and audio streaming applications, has led to an increment in the end users bandwidth demand. This has meant a considerable increment in the TCP/IP traffic that the distribution network has to handle. Due to this increasing bandwidth demand, it is necessary to develop new packet processing systems that are capable of working at higher bit rates.

It is critical to perform valid experiments on network testbeds, the use of a traffic generation tool that adequately reflects different network topologies and conditions. In order to test how the packet processing systems (routers, firewalls, IDS systems, etc) would operate in a real environment, it is required to generate traffic as similar as possible to the traffic generated on a real environment. The easiest way to achieve this is by replicating previously captured real traffic.

The aim of this project is to design, develop and implement a traffic replication architecture. This architecture will be able to replicate previously captured traffic stored in a PCAP file. It will allow the replication of the network trace exactly with the same interpacket times as when it was captured, as well as other interpacket time distributions depending on the scenario under evaluation.

In this project we propose a hybrid solution based on the NetCOPE platform. It consist on an administration software and a firmware that will be loaded into a Virtex-5 FPGA provided by the COMBOv2 network card. The administration software will provide different options to the user. Using these options, the end user will be able to: load a previously captured pcap file, load the original interpacket time model and interactively modify the loaded interpackets.

Key words:

FPGA, traffic generator, NetCOPE, Ethernet, 10 GbE,network testbeds, replication.

Agradecimientos

En primer lugar me gustaría agradecerle a Rosely el haber estado a mi lado todo este tiempo tanto en lo personal como en lo académico. Acompañándome durante la carrera, haciendo más llevaderas esas interminables prácticas y las duras épocas de exámenes, ayudándome durante las partes más difíciles del proyecto , y motivándome cuando parecía que el proyecto no iba a acabarse nunca. Sin su inestimable ayuda y esfuerzo este proyecto no sería lo que es.

Quiero agradecer a todos los que me han ayudado y orientado durante el proyecto.

En especial quiero agradecer a Sergio López Buedo, que siempre me ha animado y apoyado en mi vocación por el diseño de hardware. He podido aprender mucho de él cuando era mi tutor durante la beca de colaboración y también durante el proyecto. Y sobre todo, siempre supo sugerir la idea adecuada, cuando parecía que no había salida. A Jaime Garnica, mi tutor, que me guió por el camino adecuado para poder realizar este proyecto, y que ha estado ahí hasta el final, incluso cuando tenía que compaginarlo con su trabajo fuera de la universidad. Y a Víctor Moreno, que siempre ha estado disponible para echarme una mano y explicarme lo que hiciera falta, aunque para ello tuviese que quedarse hasta más tarde en el laboratorio. Siempre he podido contar con él.

También quiero mostrar mi agradecimiento al resto de profesores y compañeros del laboratorio: A Gustavo Sutter, Francisco Javier Gómez, Javier Aracil, Jorge López de Vergara, Marco Forconesi, David Madrigal y Javier Ramos, que me han prestado siempre su ayuda incondicional, y con los que he compartido grandes momentos y experiencias durante los años que he compartido con ellos.

No puedo no pensar en estos momentos en mi familia. En mis padres y mis hermanos, que durante todos estos años me han guiado y apoyado.

Por último, quiero agradecer también a mis compañeros durante la carrera. A Javi y Ajito, por todo su apoyo y afecto en los momentos que más lo necesitaba tanto dentro como fuera de la universidad. A Antón, por esas escapadas de última hora al starbucks. A Lore, porque pese a todo, esos primeros años los recuerdo con nostalgia. A Laura, por transmitirnos a todos tu ilusión y optimismo. Y a ti

Luis, por recordarme que la vida está para disfrutarla.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Motivación

En los últimos años se ha producido un incremento considerable del tráfico tcp/ip que se genera a nivel internacional[1]. El aumento de la penetración del uso de Internet, las redes sociales, la compartición de medios audiovisuales, el streaming de vídeo y audio, las aplicaciones cloud, la competencia comercial entre proveedores de Internet y la tendencia a trasladar todas las comunicaciones a un modelo basado en tcp/ip ha hecho necesario aumentar el ancho de banda del que disponen los usuarios finales, tendiendo actualmente a un modelo de aprovisionamiento de Internet de tipo FTTH (fibra hasta el hogar).

Esta evolución del uso de Internet ha incrementado de manera exponencial la capacidad necesaria en las redes troncales y en los niveles de agregación de las redes de distribución de datos (ver Figura1.1). Para poder atender a esta demanda creciente de ancho de banda, ha sido necesario introducir en las redes de datos nuevos dispositivos de procesado y enrutado de tráfico adaptados a redes de 10 y 40 Gbps.

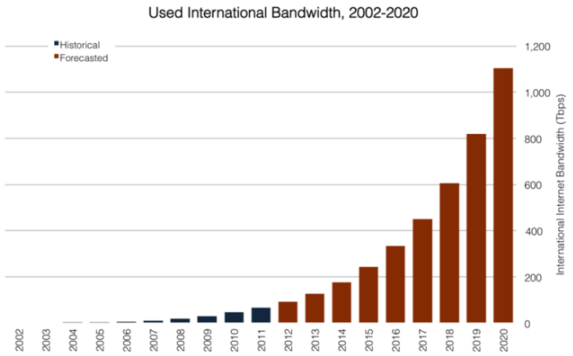

Figura 1.1: Evolución del uso de ancho de banda

Generar tráfico que refleje adecuadamente diferentes condiciones y topologías de red, es crítico para realizar experimentos válidos sobre testbeds de red, por lo que la generación de tráfico real es necesaria para poder comprobar el funcionamiento real de cualquier sistema de procesamiento de paquetes como por ejemplo routers, firewalls, sistemas IDS, etc. La manera más fiable de generar tráfico real es mediante la reproducción de tráfico previamente capturado.

Para poder realizar pruebas fiables sobre sistemas de procesamiento de paquetes es necesario, no sólo simular un tráfico real, sino también ser capaz de simular los peores escenarios de funcionamiento posibles. Estos casos son difíciles de encontrar en el tráfico real y son necesarios para poder asegurar el funcionamiento de los sistemas de procesamiento de paquetes en cualquier escenario, por lo que es necesario disponer de un sistema de replicación capaz de modificar las trazas para adaptarlas de tal manera que simulen estos casos.

Por estos motivos, en este proyecto se va desarrollar e implementar una solución capaz de simular un entorno de trabajo real de una red de 10Gbps, utilizando, tanto tiempos entre paquetes reales, como tiempos entre paquetes especificados manualmente para generar tasas binarias superiores.

1.2. Objetivos

El objetivo de este proyecto, como se ha mencionado anteriormente, es desarrollar e implementar una solución capaz de simular el entorno de trabajo real de una red de 10Gbps para utilizarla como banco de pruebas en el desarrollo de sistemas de procesado de paquetes (PPS). Los objetivos a tener en cuenta durante el desarrollo del aplicativo son los siguientes:

- Se quiere implementar una solución económica.
- Se quiere implementar una solución que sea fácilmente modificable. El entorno de la investigación y el desarrollo está en constante evolución, por lo que es necesario que nuestra solución se pueda adaptar fácilmente a los nuevos escenarios de trabajo.
- Se quiere implementar para un entorno de 10GbE por lo que se pretende que sea capaz de transmitir a la máxima tasa binaria que permita la línea (10 Gbps).
- Se quiere implementar una solución modular, portable a otras plataformas y escalable a entornos con mayores tasas de línea. La modularidad facilitará

la reutilización de módulos del código en futuros diseños y la adaptación del diseño a otras plataformas, ya que sólo será necesario cambiar los módulos que no sean compatibles con la nueva plataforma. Para que el diseño pueda ser utilizado en plataformas diferentes, se va a intentar utilizar parámetros e interfaces estándar. Esto facilitara en un futuro la adaptación del diseño al estándar 100GbE.

- Se quiere poder enviar a cualquier tasa binaria dentro del rango permitido por la línea. Por ello, es necesario que de manera parametrizable, se puedan establecer o modificar los tiempos entre paquetes de envío.
- Se quiere desarrollar una solución que trabaje en el nivel de enlace, para ser capaz de replicar tráfico de cualquier protocolo y encapsulamiento.
- Se quiere obtener la mayor precisión posible en los tiempos entre paquete de la traza que se quiere replicar. Durante la medida de ciertas características del tráfico, como por ejemplo el jitter, es importante que el generador o replicador envíe con una alta precisión en los tiempos entre paquetes para que el modelo de distribución de tiempos entre paquete sea lo más fiel posible al del tráfico que se quiere replicar.

1.3. Organización de la memoria

Esta memoria se ha organizado en los siguientes capítulos:

Introducción: En esta sección se pretende justificar el desarrollo de este proyecto: su motivación y objetivos.

Estado del arte: El objetivo de esta sección es proporcionar una visión general de algunas de las soluciones disponibles actualmente para generar tráfico, así como de las tecnologías disponibles para el desarrollo del proyecto y la evaluación de los resultados.

Diseño: En esta sección se evalúan primeramente distintas alternativas para la implementación del generador: plataformas de desarrollo, memorias y protocolos. A continuación, se describe el diseño realizado para cumplir con los requisitos y objetivos del proyecto.

Implementación: La finalidad de este apartado es describir en detalle la implementación del generador de tráfico y su funcionamiento. Adicionalmente, se explicarán las soluciones elegidas para los distintos problemas que se presentan durante su desarrollo, importantes a tener en cuenta si se desea modificar el diseño actual.

Tests y resultados: Esta sección presenta las pruebas, mediciones y simulaciones que se han realizado durante el desarrollo y tras finalizar la implementación del generador.

Conclusiones y trabajo futuro: Finalmente, se realiza un análisis de los resultados obtenidos durante las pruebas y se proponen nuevas ideas orientadas a mejorar el diseño actual.

Capítulo 2

Estado del arte

2.1. Tecnología de red

Desde sus orígenes, hace mas de 25 años, Ethernet ha evolucionado para dar soporte a la creciente demanda de tráfico de paquetes. Debido a los bajos costes de implementación, la fiabilidad y la relativa facilidad de instalación y mantenimiento, la popularidad de Ethernet ha crecido hasta el punto de que casi todo el tráfico de Internet se genera o termina en una red Ethernet. A medida que han sido necesarias redes cada vez más rápidas, se ha ido adaptando el protocolo Ethernet para que sea capaz de soportar éstas velocidades.

Con el protocolo IEEE 802.3ae* 2002 (estándar 10 Gigabit Ethernet), Ethernet puede dar soporte a velocidades de hasta 10 Gigabits por segundo manteniendo sus propiedades principales. El estándar 10 Gigabit Ethernet no sólo incrementa la velocidad a 10Gbps, sino que además aumenta la distancia máxima de interconexión hasta los 40 km, lo cual permite ampliar su utilización a redes WAN favoreciendo así su despliegue en redes metropolitanas. La implantación de la tecnología 10GbE permite aumentar significativamente el ancho de banda manteniendo la compatibilidad con las interfaces del estándar 802.3 ya instaladas, protegiendo de esta manera las inversiones previas realizadas.

Dentro del modelo OSI, Ethernet es básicamente un protocolo de nivel 1 y 2. El protocolo 10 GbE mantiene las características principales de la arquitectura Ethernet, incluyendo el protocolo de acceso al medio (MAC), el formato de las tramas y el tamaño máximo y mínimo.

La tecnología Ethernet es actualmente la tecnología más extendida para entornos LAN de alta velocidad. Empresas de todo el mundo han invertido grandes cantidades en cableado, equipamiento, procesos y formación en esta tecnología, por lo

que la tecnología 10GbE parece la mejor opción a la hora de expandir y actualizar sus redes Ethernet existentes.

El despliegue de redes 10GbE presenta las siguientes ventajas:

- La infraestructura Ethernet existente es fácilmente interoperable con la 10GbE.
- Presenta menores costes tanto de implantación como de mantenimiento, comparado con las alternativas existentes actualmente.
- Permite reutilizar procesos, protocolos y herramientas de administración ya desplegados en la infraestructura de administración.
- Existen múltiples vendedores que proporcionan productos que garantizan la interoperabilidad con este estándar.

A medida que el estándar 10GbE se introduce en el mercado y los vendedores desarrollan dispositivos 10GbE, se hace necesaria la utilización de herramientas de análisis de tráfico y de estrés para éstas redes. Por todas las razones presentadas anteriormente, se ha decidido utilizar una interfaz de tipo 10GbE en el replicador de tráfico.

2.2. Generadores de tráfico

Como se ha comentado anteriormente, para realizar pruebas fiables sobre sistemas de procesamiento de tráfico, es necesario disponer de dispositivos capaces de generar tráfico con la precisión y la tasa binaria requerida por el sistema que se desea evaluar. A la hora de analizar las diferentes herramientas existentes para generar tráfico, nos encontramos con una gran variedad de opciones, cada una planteada de una manera distinta en función de los objetivos a los que quiere dar más importancia. Por ello, para evaluar las distintas soluciones existentes, se van a analizar los siguientes aspectos del diseño y su rendimiento: tasa binaria, coste, flexibilidad y precisión en los tiempos entre paquetes.

Existen tres tipos de soluciones en función de la forma en la que se han implementado:

1. En primer lugar, están las soluciones software. Se trata de herramientas implementadas como programas que normalmente se ejecutan sobre ordenadores de propósito general. Alcanzan tasas binarias bajas debido a que la carga del sistema operativo influye directamente en su rendimiento. Para mitigar esto, algunas soluciones optan por implementar la aplicación como un módulo del kernel y así tratar directamente con el dispositivo de red, o

bien, por utilizar sistemas operativos en tiempo real que les aseguren una latencia máxima conocida. Las ventajas principales de este tipo de generadores es que son muy configurables, fácilmente modificables y más económicos que el resto.

2. En segundo lugar, están las soluciones hardware. Estas soluciones consisten en un equipo completo de hardware dedicado que proporciona múltiples herramientas de generación de tráfico y alcanza unas tasas binarias muy elevadas. Además, hay tarjetas de expansión (disponibles a la venta) que permiten aumentar su rendimiento. El problema principal que presentan estas herramientas, es que son poco flexibles y costosas.

3. Por último, hay soluciones híbridas. Estas soluciones están compuestas de un software que aporta flexibilidad y opciones de configuración a la herramienta, y una tarjeta de aceleración hardware que es la que realiza todas las tareas que requieren un rendimiento mayor. De esta manera, consiguen alcanzar tasas elevadas y precisión en el envío, permitiendo al usuario múltiples opciones de configuración. Además, las tarjetas de aceleración hardware suelen estar compuestas de chips reconfigurables, por lo que permiten la modificación tanto de la parte software como hardware en caso de que hiciera falta adaptarlas a un caso particular. Como desventaja, presentan recursos de almacenamiento más limitados y su coste es mayor que el de las soluciones software.

2.2.1. Soluciones software

La mayoría de los generadores de tráfico que se han desarrollado consisten en un software que se ejecuta en componentes hardware de propósito general. Estas herramientas, pese a presentar un rendimiento mucho peor que las soluciones hardware o híbridas, son muy configurables , baratas y fácilmente modificables.

El problema principal que presentan es la incertidumbre que introduce el sistema operativo sobre el cual se ejecutan. La carga de la CPU, las operaciones I/O a los discos o el stack TCP/IP, suponen un reto a la hora de diseñar un sistema software capaz de garantizar el rendimiento deseado de manera fiable. Además, estas herramientas normalmente se ejecutan sobre componentes de hardware genérico, lo cual puede hacer que el comportamiento varíe en función del vendedor y modelo del hardware que se utilice. A continuación se presentan distintos generadores de tráfico implementados en software que mediante distintas aproximaciones, intentan mitigar estos problemas, sin perder la flexibilidad que proporciona una solución software. Según su naturaleza propietaria, podemos distinguir entre las soluciones comerciales como puede ser ZTI[2] y los proyectos de código abierto.

Proyectos de código abierto

En el grupo de generadores que se ejecutan como procesos de usuario nos encontramos muchas opciones como pueden ser MGEN[3], BRUTE[4],TCPivo,tcpreplay, packETH, D-ITG[5] u Ostinato. De las herramientas anteriores, se van a comparar a continuación aquellas que soportan la generación de tráfico en enlaces 10GbE y que proporcionan mejores resultados[6]. Todas ellas permiten enviar tanto paquetes TCP como UDP.

packETH PackETH[7] es un generador de tráfico desarrollado para Linux, capaz de enviar tráfico IPv4, IPv6 y ARP. Soporta los protocolos RTP e ICMP y el envío de tramas de tipo *jumbo*. La característica más significativa de esta herramienta es que se han realizado pruebas de envío en enlaces de 40Gbps, consiguiendo alcanzar la tasa de 37,5 Gbps para paquetes UDP de 9000 bytes de payload. En el caso del envío de paquetes TCP de 64 bytes la tasa máxima que alcanza es 150 Mbps. La tasa máxima que puede alcanzar para tramas de tipo *jumbo* en un enlace 10GbE es de 7,8 Gbps.

D-ITG D-ITG[5] es una plataforma capaz de generar tráfico IPv4 e IPv6 así como de otros protocolos de nivel de transporte como DCCP o SCTP. Su característica más significativa es que dispone de patrones predefinidos para el tamaño de los paquetes y para los tiempos entre paquetes de acuerdo a distintas distribuciones probabilísticas: constante, uniforme, cauchy, normal, poisson y gamma. Además, permite realizar medidas sobre métricas QoS: pérdida de paquetes, jitter, RTT y medida de retardos de un sentido. Estas medidas las puede tomar gracias a que dispone de un módulo emisor y de un módulo receptor que recibe el tráfico generado.
En cuanto a rendimiento, tiene un rendimiento inferior a los otros 2 generadores analizados alcanzando tasas de entre 62 y 6200 Mbps para el peor y mejor caso respectivamente.

Ostinato Ostinato[8] es un generador que al igual que los 2 anteriores, permite el envío de paquetes IPv4 e IPv6. Su principal ventaja con respecto al resto, es su flexibilidad. Permite simular gran cantidad de protocolos con diferentes tiempos entre paquetes. Además, permite añadir mediante scripts cualquier protocolo que no esté implementado en la herramienta. En las medidas de rendimiento es el generador software que mejores tasas binarias alcanza, llegando a los 9 Gbps para el envío de paquetes de 8950 bytes.

Comparativa de Soluciones software

Packet size (bytes)	packETH	Ostinato	D-ITG
64	150 Mbps	135 Mbps	62 Mbps
1408	1745 Mbps	2850 Mbps	1950 Mbps
8950	7810 Mbps	9000 Mbps	6200 Mbps

Cuadro 2.1: Comparativa de Soluciones software

Para mejorar el rendimiento de las soluciones software (ver Tabla 2.1), una opción es implementar el generador de tráfico como un módulo del kernel que se comunique directamente con el driver de la tarjeta de red. Un ejemplo de este tipo de solución es KUTE[9]. En el caso de los generadores que se ejecutan como procesos de usuario, se opta por utilizar kernels de baja latencia como propone TCPivo[10], o sistemas operativos de tiempo real como es el caso de RUDE/CRUDE[11].

KUTE KUTE[9] es un generador de tráfico UDP que se ejecuta enteramente en un kernel de Linux 2.6. Gracias a ello, KUTE es capaz de enviar tráfico directamente en el nivel de enlace. Para las medidas de tiempos entre paquetes, KUTE utiliza las marcas de tiempo presentes en el buffer del socket de salida, por lo que la precisión que es capaz de proporcionar depende de si las marcas de tiempo han sido generadas por un hardware externo o por el kernel. El problema principal que presenta esta solución es que es poco configurable ya que no se puede controlar ni modificar desde el espacio de usuario. En cuanto a rendimiento, KUTE es capaz de alcanzar tasas de 200 Mbps para paquetes de 64 bytes (415 Kpps), y una precisión de unos 6 microsegundos.

RUDE/CRUDE RUDE/CRUDE[11] es un generador de tráfico que se ejecuta sobre un sistema operativo de tiempo real. Estos generadores proporcionan mejores resultados que los que se ejecutan en sistemas operativos convencionales ya que el sistema en tiempo real garantiza unos tiempos mínimos de respuesta acotando así la latencia. En concreto, RUDE consigue una precisión en los tiempos entre paquetes de 14 microsegundos, que supone una gran mejora con respecto a los generadores software estándar, que tienen precisiones de cientos de microsegundos. El mayor inconveniente de esos sistemas es que pese a existir soluciones software no comerciales, la mayoría de los sistemas operativos en tiempo real sí que lo son, por lo que no es una solución tan económica como el resto.

Proyectos comerciales

LanTraffic V2 LanTraffic V2 de *ZTI Comunications[2]* es un generador de tráfico comercial implementada en software. Esta solución es altamente configurable y está disponible para los sistemas Windows XP y Windows 7. Según los datos de rendimiento proporcionados por ZTI, el generador alcanza tasas de hasta 940.7 Mb/s en enlaces de 1GbE y de hasta 4.7 Gb/s en enlaces de 10 GbE, aunque este segundo dato de rendimiento se extrapola a partir del rendimiento de los enlaces de 1Gb y en ninguno de los 2 casos se especifica el tamaño de paquete utilizado para las pruebas. La precisión en los tiempos de envío está limitada al orden de los microsegundos ya que es la precisión del reloj del sistema en windows. Además, presenta otro problema común a todas las soluciones comerciales, y es que debido a su naturaleza propietaria es casi imposible realizar modificaciones sobre la herramienta, limitando así su utilidad en escenarios de investigación o de pruebas de nuevos protocolos.

Las soluciones software actuales presentan muchas opciones de configuración, y se adaptan fácilmente a distintos entornos de trabajo. Además, su facilidad de implementación ha permitido que haya una gran variedad de soluciones disponibles que cubren la mayoría de protocolos y entornos existentes. Sin embargo, su bajo rendimiento para paquetes de tamaño pequeño y medio (ver Tabla 2.1) sigue limitando su uso a redes de baja velocidad, o a escenarios en los que la tasa binaria no es un factor importante.

2.2.2. Soluciones hardware

Estas soluciones permiten realizar pruebas sobre sistemas de procesamiento de paquetes a las tasas máximas de línea, simular multitud de escenarios y aumentar su funcionalidad fácilmente mediante tarjetas de expansión con el fin de aumentar la cantidad de tráfico que pueden generar. Sin embargo, a parte de ser muy caros (el precio de un IXIA 1600[12] parte de los $41,000 y según la configuración sobrepasan los $100,000), estos sistemas no siempre son precisos a la hora de replicar un tráfico real determinado, debido a que su funcionamiento suele estar basado en una simulación estadística de tráfico y no en la replicación de unas trazas capturadas previamente. Este comportamiento hace que no sean la mejor opción a la hora de replicar características propias de aplicaciones concretas. Además, su naturaleza propietaria los hace demasiado poco flexibles para ser utilizados en la investigación de nuevos protocolos y técnicas de red.

Los principales fabricantes de equipos hardware de generación de tráfico son IXIA, Spirenet, Smartbits o XENA[13, 14, 12].

2.2.3. Soluciones híbridas

Las soluciones híbridas se encuentra entre medias de las soluciones software y las hardware. Constan de una parte hardware que se encarga de realizar las tareas que requieren un mayor rendimiento, y de una parte software que gestiona el flujo de funcionamiento del generador y las comunicaciones con la parte hardware. Como núcleo de aceleración hardware, se suelen utilizar GPUs[15] o FPGAs[16]. En el caso particular de los generadores de tráfico, la mayoría de los proyectos existentes optan por las soluciones basadas en FPGAs debido a que existen varias plataformas de desarrollo de aplicaciones de red basadas en esta tecnología. Además, el uso de chips FPGA[17] facilita la modificación del firmware aportando mucha flexibilidad a los diseños.

Estos generadores tienen en común que utilizan chips FPGA integrados en tarjetas de red como núcleo de aceleración hardware. Este núcleo de procesamiento además de mejorar el rendimiento de la capa software, permite disponer de recursos exclusivos que no estén gestionados por el sistema operativo. De esta manera, se tiene un mayor control sobre los tiempos de latencia y se puede aumentar la precisión en los tiempos entre paquetes durante el envío por encima de la precisión que proporciona el reloj del sistema operativo. Como ejemplo se puede ver que en un envío de paquetes con tcpreplay, se producen desvíos en los tiempos entre paquetes de entre -2 y -18 microsegundos[18] debidos a la latencia que introduce el stack TCP/IP y a la limitación del reloj de sistema que tiene precisión de microsegundos.

En el campo de las aplicaciones de red aceleradas por hardware destacan dos plataformas: NetFPGA y NetCOPE.

NetFPGA

NetFPGA[19] empezó como un proyecto de investigación de la universidad de Stanford, cuyo objetivo era diseñar una herramienta educativa que permitiese enseñar acerca del diseño de hardware de red. NetFPGA es una tarjeta de aceleración hardware que se puede utilizar en servidores con hardware de propósito general para aumentar sus funcionalidades.

Actualmente, existen dos modelos de NetFPGA: uno que utiliza interfaces 1GbE y otro que utiliza interfaces 10GbE. Estos dos modelos presentan las siguientes características técnicas(ver Tabla 2.2):

Especificaciones NetFPGA

Modelo	NetFPGA-1G	NetFPGA-10G
FPGA	Virtex II Pro	Virtex 5
Interfaces	4 x 1 Gigabit	4 x 10 Gigabit
SRAM	4.5 MB	27 MB
DRAM	64 MB	288 MB
Bus	PCI	PCIex8

Cuadro 2.2: Especificaciones NetFPGA

La plataforma de desarrollo proporcionada junto con la tarjeta se compone de 3 partes. La primera es un módulo del kernel que se utiliza para comunicarse con la tarjeta hardware. Estas comunicaciones se realizan a través de DMA para las transferencias de datos y a través de registros en el caso de señales de control y de estado.

La segunda parte consiste en las herramientas necesarias para comunicarse con la tarjeta (carga de ficheros .bit, lectura y escritura de registros, etc).

La tercera parte es un firmware que facilita la interacción de los módulos que se quieren desarrollar con las interfaces de red. Consiste en un pipeline de 16 colas que controlan las transmisiones y recepciones de las 4 interfaces, y de un módulo de gestión de éstas que hace de interfaz con los módulos del usuario.

Sobre ésta plataforma se han desarrollado muchas aplicaciones de red en el ámbito de la investigación y muchos de estos trabajos se han publicado como proyectos de código abierto y están disponibles como modelos de referencia[20]. En lo que respecta a generadores de tráfico destacan 2 proyectos que plantean el problema desde 2 puntos de vista totalmente diferente:

SPG El generador de paquetes SPG[18] ha sido desarrollado por la universidad de Stanford como ejemplo de aplicación para la NetFPGA de 1GbE. Permite el envío de tráfico a 1 Gbps por hasta cuatro interfaces de red simultáneamente. Los datos transmitidos se obtienen de un fichero PCAP y se transfieren a la memoria local de la tarjeta NetFPGA para, posteriormente, ser enviadas a través de las interfaces de red con la tasa binaria, el tiempo entre paquetes, y el número de iteraciones especificado por el usuario.
El diseño, también permite capturar simultáneamente tráfico a través de las cuatro interfaces de red. Los paquetes capturados se almacenan en la memoria local de la tarjeta antes de escribirlos a un fichero con formato estándar PCAP. El módulo de captura de paquetes, además, se encarga de generar estadísticas tanto de los paquetes capturados como de los generados.

Aunque el SPG permite generar tráfico de una manera mucho más realista que otros generadores que utilizan trazas sintéticas, presenta varias limitaciones. La más importante es que el tamaño de la memoria local de la tarjeta NetFPGA-1G está limitado a 64 MB, lo cual equivale a unos 0,5 segundos de tráfico a 1Gbps. Esta memoria, además, se comparte con el módulo de captura por lo que si se quieren utilizar los dos simultáneamente, la limitación es aún mayor para ambos.

Otra limitación que presenta esta solución es que los tiempos entre paquetes utilizados durante el envío tienen que ser fijos. Esto limita su utilidad en caso de escenarios en los que el sistema de procesamiento de paquetes que se quiere verificar es sensible al jitter o requiere de un dimensionado preciso de los buffers.

Por último, presenta la desventaja de ser un diseño para la tarjeta de 1GbE por lo que su aplicación está limitada a anchos de banda de hasta 1Gbps.

Caliper Caliper[21] plantea una solución diferente al problema de replicar tráfico de la manera más fiel posible a la real. No es un generador de tráfico como tal, sino que pretende introducir una capa intermedia entre un generador de tráfico cualquiera y la interfaz de red, con el fin de mejorar la precisión en los tiempos de envío. El objetivo de esta herramienta es controlar de manera precisa los tiempos de transmisión de paquetes generados dinámicamente en el host y enviarlos de manera continua (no requiere de precarga de datos) a la tarjeta NetFPGA.

Está implementado utilizando la plataforma NetThreads[22], que permite compilar y ejecutar programas en C multihilo en la tarjeta NetFPGA. Caliper recibe paquetes generados en el host y los transmite a través de una interfaz 1GbE con una precisión en los tiempos entre paquetes de 8 ns. Esta aproximación al problema de la generación de tráfico presenta dos ventajas principales con respecto a las soluciones que parten de una traza capturada previamente: En primer lugar, al generar el tráfico de manera dinámica puede simular enlaces reales con TCP u otros protocolos reaccionando a las respuestas del servidor bajo estudio (si el generador software los soporta), y en segundo lugar, permite establecer funciones de distribución reales en los tiempos entre paquetes independientemente de los datos generados.

El flujo que siguen los datos durante el envío de paquetes con Caliper se divide en 3 partes. En primer lugar, los paquetes se generan en un simulador de red en el espacio de usuario y son enviados al driver de la NetFPGA. Después, se envían los paquetes a la NetFPGA a través de PCI utilizando el DMA de la tarjeta. Como el DMA no soporta transferencias de 1Gbps, los paquetes enviados a través del DMA no contienen toda la información, sino que se envía sólo el tamaño del paquete y la información mínima nece-

saria para reconstruir la cabecera de los paquetes. El payload del paquete se rellena con ceros durante la transmisión. Por último, el software que se está ejecutando en la plataforma NetThreads recibe los datos, construye los paquetes y los envía de acuerdo a los tiempos especificados.
Aunque la solución que propone Caliper permite realizar tests que no se pueden realizar con otros generadores de tráfico, presenta muchas limitaciones debido al cuello de botella que supone el DMA, y a que depende del envío de paquetes de una aplicación software de terceros con sus propias limitaciones e influenciada por la incertidumbre que introduce el sistema operativo.

OSNT Recientemente, se ha creado un proyecto de código abierto que implementa un capturador y un generador de tráfico para la tarjeta NetFPGA-10G[23]. El proyecto ha sido publicado en mayo de 2014 y aún no se han realizado estudios sobre su rendimiento o arquitectura pero según la documentación proporcionada hasta ahora, el sistema sería capaz de generar tráfico a tasas de 10Gbps y obtener errores en los tiempos entre paquetes menores de 6.25 ns mediante el uso de una señal gps externa. El proyecto está en una fase inicial pero de llegar a su objetivo de enviar a la tasa máxima de línea en las 4 interfaces de red, podría generar tráfico agregado de hasta 40 Gbps.

NetCOPE

NetCOPE[24] es una plataforma de desarrollo de aplicaciones de red aceleradas por hardware. De manera similar a NetFPGA, NetCOPE utiliza una tarjeta de aceleración hardware que integra un chip FPGA que es el responsable de procesar los paquetes capturados o generados. La tarjeta de red que utiliza la plataforma es la COMBOv2. Dispone de 2 interfaces de red 10GbE y de hasta 2 GB de memoria. El firmware y las herramientas proporcionadas con NetCOPE permiten desarrollar fácilmente aplicaciones de red sin tener que preocuparse de desarrollar los módulos que controlan las distintas interfaces de la tarjeta. A diferencia de NetFPGA, proporciona ya implementadas todas las herramientas de comunicación con el módulo DMA de la tarjeta, y unas librerías en C para facilitar el desarrollo de los programas software que interactúan con ella. Actualmente, hay trabajos que demuestran que es posible implementar un generador de tráfico capaz de transmitir a las tasas binarias máximas en líneas de 10GbE con esta plataforma[25] utilizando una arquitectura similar a la utilizada en el generador SPG.

2.3. Conclusión

A lo largo de este capítulo, se han analizado las distintas soluciones que existen actualmente para generar tráfico tanto real como sintético. Las soluciones software pese a ser muy configurables no cumplen los requisitos de este proyecto porque obtienen tasas demasiado bajas para paquetes pequeños (ver Tabla 2.1) y tienen una precisión en los tiempos entre paquetes del orden de los microsegundos. En cuanto a las soluciones hardware, obtienen muy buenos rendimientos pero no son lo suficientemente flexibles. Su mercado son los entornos empresariales en los que no se trabaja con protocolos en desarrollo y disponen de un presupuesto menos ajustado. Por último se han analizado las soluciones híbridas. Éstas han demostrado ser capaces de alcanzar las tasas binarias y la precisión requeridas por este proyecto. Además, las plataformas de desarrollo basadas en FPGAs presentadas, proporcionan muchas facilidades a la hora de desarrollar e implementar cualquier tipo de aplicación de red.

En concreto la plataforma NetCOPE, nos proporciona ya implementado, todo el entorno necesario para poder desarrollar el replicador de tráfico, y la FPGA que integra aporta gran flexibilidad al diseño y facilita la modificación del mismo. Además, se han realizado trabajos[26] que demuestran que es posible modificar el firmware y las herramientas de la plataforma NetCOPE para portarlas a NetFPGA, lo cual proporcionaría en un futuro la posibilidad de adaptar el diseño que se va a realizar a esta plataforma. Por estas razones se va a utilizar la plataforma NetCOPE durante el desarrollo de este proyecto, por lo que se describirá con más detalle a lo largo de esta memoria.

Capítulo 3

Diseño

3.1. Solución propuesta

El objetivo de este proyecto es el diseño, desarrollo e implementación de una arquitectura de reproducción de tráfico. Esta arquitectura permitirá retransmitir tráfico previamente capturado y almacenado en un fichero PCAP. Posibilitará reproducir dicha captura de tráfico con exactamente el mismo tiempo entre paquetes con el que fue capturado, así como aplicar distribuciones de tiempos entre paquetes según el caso o escenario que se quiera probar.

Para su diseño se ha planteado una solución híbrida basada en la plataforma NetCOPE: se va a implementar un software de control y un firmware que se cargará en la FPGA Virtex-5 que proporciona la tarjeta COMBO v2. El software de control proporcionará al usuario una serie de opciones que le permitirán cargar un fichero de trazas, cargar unos tiempos entre paquetes y modificar de manera interactiva los tiempos entre paquetes cargados.

Una vez cargados los datos en la memoria, el usuario podrá iniciar la replicación de los datos comunicándose con la tarjeta a través de unos registros. En la tarjeta se detectará el valor del registro modificado, y se iniciará el proceso de reconstrucción de la traza cargada. Para ello, se reconstruirá la traza cargada a partir de los datos y los tiempos entre paquetes cargados previamente, y un módulo de envío, mediante un contador con precisión de ns, replicará la traza manteniendo los tiempos entre envíos especificados por el usuario si fuese posible. Este módulo de envío se comunicará con los controladores de las interfaces de red para, finalmente, enviar los paquetes a la red.

3.2. Plataforma de desarrollo NetCOPE

NetCOPE es una plataforma configurable que permite el desarrollo rápido de aplicaciones de red en tarjetas de aceleración FPGA (aplicaciones aceleradas por hardware), como por ejemplo la Combo v2. Este tipo de aplicaciones se compone de dos partes: una parte software y una parte hardware que se implementa como firmware en una FPGA (ver Figura 3.1).

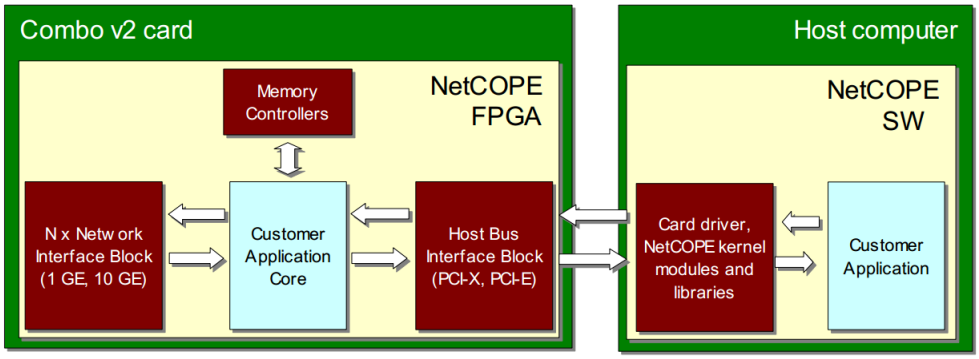

Figura 3.1: Entorno de desarrollo de aplicaciones NetCOPE[27]

El objetivo de la plataforma es definir una interfaz universal entre el software y el hardware e implementar los distintos bloques comunes a la mayoría de las aplicaciones de red, como son:

1. Bloque de entrada/salida para la transmisión y recepción de paquetes a través de las interfaces de red.

2. Acceso a la tarjeta desde la aplicación software. La tarjeta se conecta al host utilizando un bus PCIe y proporciona los drivers para la comunicación con el mismo.

3. Transferencias de datos a alta velocidad entre la aplicación software y el núcleo de aceleración mediante DMA. La aplicación de usuario accede a esta interfaz como si fuese una interfaz de red estándar.

En general el diseño de una aplicación de red acelerada por hardware se puede dividir en 5 capas (ver figura 3.2): La aplicación del usuario está compuesta de una capa software y una capa de un núcleo de aceleración, la interfaz universal está compuesta por las capas de drivers y por los bloques de interconexión que proporciona la plataforma, y por último la última capa consiste en la tarjeta hardware en cuestión, que integra el chip FPGA.

Esta estructura en capas, facilita la portabilidad de las aplicaciones de usuario, ya que si cambiases la tarjeta de aceleración hardware, sólo sería necesario modificar los bloques de la capa de interfaz que dependen de ésta.

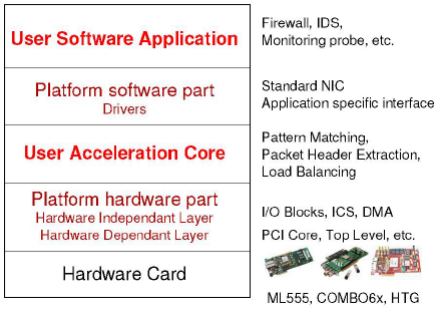

Figura 3.2: Estructura de capas de la plataforma NetCOPE[24]

3.2.1. Arquitectura hardware

Los bloques de la arquitectura hardware que proporciona la plataforma NetCOPE se muestra en la figura 3.3. Los bloques básicos de la plataforma son: las interfaces de red, los buffers DMA, las interfaces local bus e internal bus para transferencias de datos y el controlador PCI. Los paquetes vienen de las interfaces de red y continúan a través del módulo de red hasta el bloque de procesamiento de usuario (Application). El resultado de este procesamiento se puede enviar por el PCI a través del DMA a la aplicación software. De manera similar, en la dirección opuesta la aplicación software se comunica con el bloque de procesamiento de usuario a través del DMA y los buses (local bus e internal bus). Los datos procesados son enviados a través del módulo de red a las interfaces de red. Adicionalmente, la tarjeta Combo v2 permite la conexión del bloque de procesamiento de usuario con un módulo de memoria para almacenamiento temporal de datos.

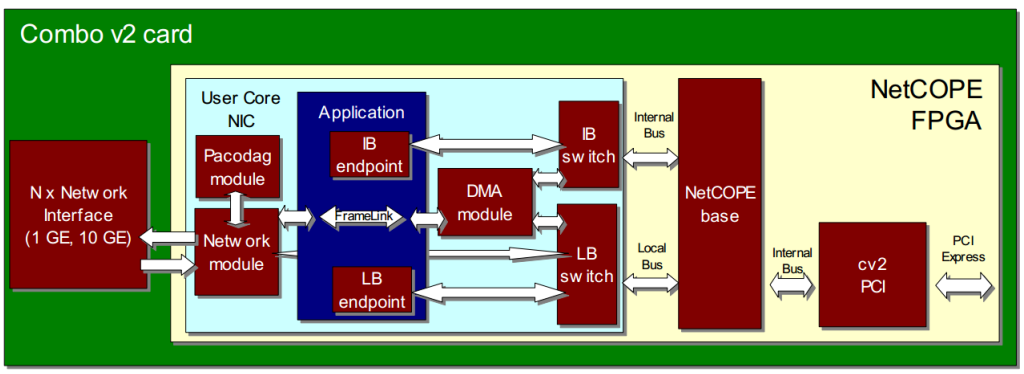

Figura 3.3: Arquitectura de la plataforma NetCOPE[27]

3.2.2. Tarjeta Combo v2

La plataforma NetCOPE actualmente utiliza la segunda generación de tarjetas Combo llamada Combo v2 y es utilizable con cualquier host que soporte PCIe x8. La tarjeta Combo v2 es capaz de procesar flujos de datos en el nivel de enlace del modelo OSI, donde los anchos de banda pueden alcanzar magnitudes de centenas de Gbps. Además, puede soportar tasas de transmisión hacia el entorno software de decenas de Gbps. Una de sus principales ventajas es que presenta una configuración muy flexible ya que está compuesta de una placa base principal a la que se conecta una tarjeta de expansión que puede contener distintas configuraciones de interfaces de red.

Las características más importantes de Combo v2, por las que se ha elegido esta tarjeta para la implementación del replicador de tráfico, son las siguientes:

PCI Express: Las transferencias de datos entre la placa y el host se realizan a través de un PCIe de 8 canales (PCI x8), lo cual permite un ancho de banda máximo teórico de 32 Gbps.

Virtex FPGA: Combo v2 incluye una FPGA Virtex-5, lo que aporta gran flexibilidad a la plataforma al poder utilizar una amplia variedad de bloques embebidos.

JTAG: Todos los elementos electrónicos de la tarjeta que soportan interfaces JTAG están conectados a una única cadena de comunicación. La tarjeta de expansión también se incluye automáticamente en la cadena a través de un circuito auxiliar. El punto de entrada de la cadena JTAG puede ser tanto el bus PCIe como un conector dedicado.

Sistema de alimentación: La parte central del sistema de alimentación de la tarjeta Combo v2 se encuentra en la placa principal y está dividido en varios raíles de 5 y 3.3 V generados a partir de los 12 V proporcionados por el bus PCIe. La inicialización de estos raíles está controlada por un microcontrolador independiente.

Sistema de refrigeración: La parte más crítica en términos de refrigeración es la FPGA, que se refrigera mediante un refrigerador activo.

Sistema de arranque: El sistema de arranque de la tarjeta permite reconfigurar la FPGA a través del PCIe sin la necesidad de apagar y encender el host.

Sistema de almacenamiento en memoria: La tarjeta COMBO v2 está equi-

pada con 2 memorias QDRII y un socket para una memoria DDRII SO-DIMM. La arquitectura del sistema de memorias está diseñada con el objetivo de conseguir un rendimiento de 20Gbps en lecturas y escrituras independientes (10 Gbps en cada una si se realizan de manera concurrente). Junto con la documentación de la tarjeta se proporcionan 2 modelos de referencia que muestran un ejemplo de instanciación de cada uno de los tipos de memoria.

- **QDR**
  - Dispone de dos módulos de memoria QDRII pensados para ser usados como cache en aplicaciones. Funcionan a una frecuencia de 250MHz y permiten almacenar hasta 72 Mb.
- **Socket DIMM**
  - Permite conectar una memoria de alta capacidad DDRII. Funciona a una frecuencia de 250 MHz y puede ser de hasta 2GB de capacidad.

Relojes: Dispone de dos circuitos programables de síntesis de relojes y de múltiples osciladores con frecuencias fijas. Los osciladores alimentan los árboles de reloj de la FPGA y dentro de la FPGA hay instanciados 2 PLL concatenados que proporcionan a la lógica de procesamiento un sistema de relojes muy flexible.

Interfaces de red: Para la conexión por red se utiliza una tarjeta auxiliar que se conecta a la principal. Esto ofrece la posibilidad de elegir entre varias configuraciones de interfaces de red:

- **COMBOI-1G4:** Proporciona 4 interfaces de 1 Gbps de tipo SFP.
- **COMBOI-10G2:** Proporciona 2 interfaces de red de 10 Gbps de tipo XFP.
- **COMBOI-10G4TXT:** Proporciona 4 interfaces de red de 10Gbps de tipo SFP+ con un preprocesado por FPGA independiente.

Para el diseño del replicador de tráfico, el único requisito necesario en la tarjeta de expansión es que tenga una interfaz de 10Gbps, por lo que se ha utilizado la tarjeta de expansión COMBOI-10G2.

3.2.3. Drivers y librerías sze

La parte software de la plataforma NetCOPE está diseñada para proporcionar todas las herramientas necesarias para comunicar el software del usuario con el firmware de la tarjeta COMBO v2. El software de la plataforma NetCOPE se divide en 3 capas: drivers, librerías y herramientas. Además, desde un punto de vista funcional, el software proporcionado está dividido en 2 grupos: el primero proporciona funcionalidades relacionadas con la configuración e interacción con los componentes hardware y el segundo grupo, llamado "SZE", proporciona las funcionalidades para realizar transferencias de datos por DMA.

Drivers

Se proporcionan 2 drivers: el driver combo6 que implementa las operaciones de entrada/salida básicas para interactuar con la tarjeta, y el driver szedata2 que da soporte a las transferencias de datos por DMA.

Librerías

La plataforma NetCOPE proporciona 4 librerías de funciones que se han utilizado en el proyecto durante el desarrollo del programa de control.

Las dos primeras son la librería *libcombo* y la librería *libcommlbr*. Estas librerías proporcionan una serie de funciones en C que permiten comunicarse con los buses de la tarjeta COMBO v2 a bajo nivel. Con estas librerías se ha implementado el acceso a la interfaz de registros de nuestro diseño.

Las otras dos librerías son la *libsze2* y la *libpcap-sze2*. Estas librerías permiten utilizar las herramientas de red estándar, como pueden ser tcpreplay, tcpdump o ethereal, para acceder a las interfaces DMA de la plataforma de manera transparente.

Herramientas

Para facilitar la interacción básica con la tarjeta desde el host, la plataforma viene con una serie de herramientas, que utilizando las librerías anteriormente descritas, permiten realizar las operaciones más comunes como pueden ser: el arranque de la tarjeta, la configuración de la FPGA, acceso a los registros, configuración de las interfaces de red, etc.

3.3. Entorno de desarrollo

Para el desarrollo e implementación del proyecto, se ha utilizado el programa Xilinx ISE. Es un programa que integra todas las herramientas necesarias para para sintetizar, implementar y programar chips FPGA de Xilinx.

Para realizar las simulaciones necesarias durante el desarrollo de los módulos se ha utilizado el programa ModelSim. Este programa permite simular el comportamiento teórico de un módulo introduciendo distintas señales de test a través de un fichero de testbench. También detecta errores sintácticos y de lógica del diseño, facilitando enormemente la detección de errores en las etapas iniciales de desarrollo.

3.4. Almacenamiento en memoria

Uno de los principales problemas a la hora de diseñar cualquier generador de tráfico se encuentra en el almacenamiento de los datos. Los medios de almacenamiento físico normalmente no soportan unas velocidades de lectura tan altas como las velocidades de envío de las interfaces de red, por lo que limitan la velocidad máxima a la que se pueden enviar los datos. Esta limitación es cada vez menor debido a la evolución de los discos de estado sólido (soportan velocidades de lectura de hasta 4Gbps sin configuraciones de raid). Sin embargo, aunque se consiguiera leer los datos de la traza suficientemente rápido, no se podría asegurar que se fuese a mantener constantemente esa velocidad, ya que un pico de carga del procesador u otro proceso del sistema compitiendo por el acceso a disco, podría provocar un descenso en la tasa de lectura, que a su vez, ocasionaría que la tasa de envío se redujese o que los tiempos entre paquetes de la traza enviada no fuesen fieles a los originales.

Por esta razón, para evitar depender de las condiciones externas a nuestro sistema a la hora de leer los datos de la traza que se quiere replicar, se ha optado por una solución que propone cargar los datos inicialmente en una memoria de las proporcionadas por la tarjeta COMBO v2, para posteriormente replicar y enviar la traza almacenada. De esta manera, se obtiene un control mucho mayor de las lecturas de datos, y se pueden conseguir precisiones mayores en la replicación de los tiempos entre paquetes.

Para facilitar la portabilidad del diseño a otras plataformas y para que se pueda cambiar fácilmente de tipo de memoria utilizada, se ha decidido diseñar la interfaz con la memoria utilizando un protocolo sencillo basado en el protocolo localBus. Esta interfaz estará conectada a la memoria a través de un bloque traductor que

transforme el protocolo localBus al protocolo de la memoria. De esta manera se puede adaptar fácilmente el diseño para ser utilizado con otros tipos de memoria con tan solo cambiar el bloque traductor.

A la hora de elegir el tipo de memoria que se va a utilizar, se han tenido en cuenta los 2 tipos de memoria que soporta la tarjeta COMBO v2. A continuación se va a realizar un análisis de los 2 tipos de memoria disponibles y de cómo se podrían utilizar en el contexto de este diseño.

QDRII

La memoria QDR tiene unos tiempos de latencia bajos y soporta un ancho de banda de 17Mbps de lectura. Esta memoria tiene el inconveniente de tener muy poca capacidad. Cada una de las 2 memorias que tiene la tarjeta COMBO v2 es de 4Mb.

Inicialmente se pensó en utilizar estas 2 memorias para almacenar los tiempos entre paquetes, pero su tamaño era demasiado pequeño como para poder almacenar los 20000000 de tiempos entre paquetes necesarios.

DDRII

La memoria DDR tiene unos tiempos de latencia más variables y en general mayores. Soporta un ancho de banda de hasta 32 Gbps y puede tener una capacidad de hasta 2 GB.

Esta memoria, al ser de mayor capacidad, está pensada para ser utilizada como buffer de paquetes. En nuestro diseño, en un principio, se pensó en utilizarla para almacenar sólo los paquetes de la traza que se iba a replicar, pero finalmente ante la imposibilidad de utilizar la memoria QDR se tuvo que utilizar también para almacenar los tiempos entre paquetes.

3.5. Protocolos de comunicación

Para comunicar los diferentes bloques del diseño se han utilizado diversos protocolos de comunicación en función del tipo de datos que se quieren transmitir y de la complejidad requerida en cuanto a señales de control.

En este diseño se utilizan los siguientes protocolos:

localBus El protocolo localBus es un protocolo sencillo que permite una comunicación bidireccional y direccionamiento tanto en lectura como en escritura.

Este protocolo se ha utilizado para las comunicaciones del usuario con la interfaz de registros, ya que estas requieren de información de direccionamiento para poder seleccionar los distintos registros. También se utiliza para transmitir, a través de la interfaz de registros, los tiempos entre paquetes al módulo que hace de interfaz con la memoria. En la figura 3.4, se muestra una operación de lectura estándar utilizando éste protocolo.

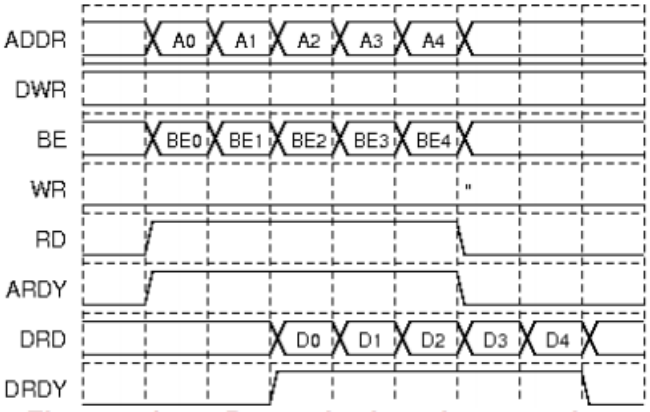

Figura 3.4: Operación de lectura de LocalBus[27]

FrameLink El protocolo FrameLink es un protocolo orientado al envío de datos. Es unidireccional, y permite diferenciar diferentes partes dentro de los datos que envía: header, payload y footer (Ver figura 3.5). Además, este protocolo no contempla el uso de direcciones. Consiste en un flujo de datos con las señales necesarias para indicar el inicio y el fin del flujo. Por estas razones, este protocolo se ha utilizado para la comunicación con las interfaces de red y para las transferencias DMA.

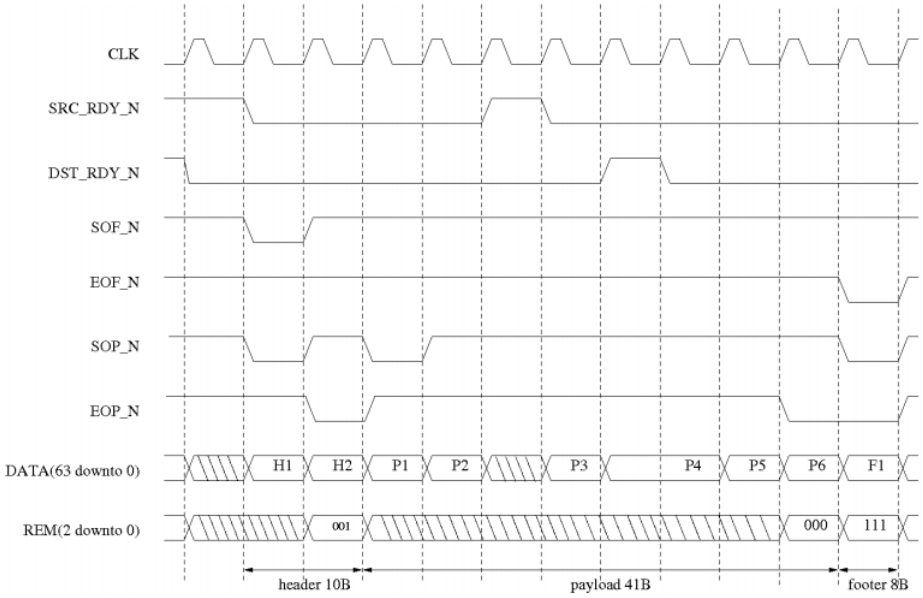

Figura 3.5: Envío de trama con FrameLink[27]

LB_MIG Para comunicarse con el controlador de memoria, como se ha explicado anteriormente, se quiere utilizar una interfaz genérica en lugar de la interfaz propia del controlador de memoria DDR. Para ello, se ha modificado el protocolo LocalBus para que soporte ráfagas de escritura y lectura, es decir, que por cada dirección que se proporcione al bus, éste esperará 2 datos de 128 bits cada uno en lugar de 1 como esperaría el protocolo LocalBus estándar. En la figura 3.6 se puede ver un ejemplo de funcionamiento de este protocolo.

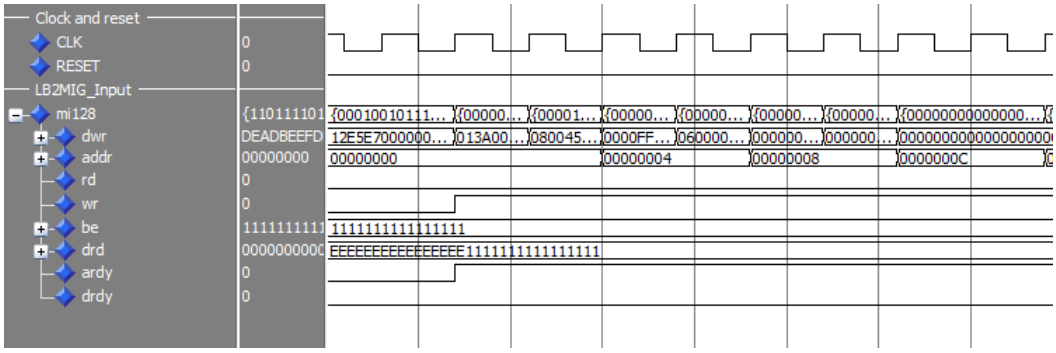

Figura 3.6: Operación de escritura LB_MIG

MIG El protocolo MIG es el protocolo que utiliza la interfaz de usuario del controlador de memoria. Este protocolo es similar al localBus en sus características, pero además tiene señales de control específicas para el tipo de memoria para el que esté diseñado el controlador. El protocolo genérico

LB_MIG tendrá que traducirse a este protocolo para poder acceder a la memoria.

FIFOS Por último, se han utilizado múltiples fifos para interconectar los distintos módulos de la arquitectura. Estas fifos permiten solventar el problema que supone que los distintos módulos procesen los datos a distinta velocidad o que la interfaz de red deje de enviar momentáneamente por saturación.

3.6. Arquitectura del sistema

Como se ha comentado anteriormente, el desarrollo de este proyecto se ha planteado como una solución híbrida. Un programa software que se ejecutara como un proceso de usuario en el sistema que hará de interfaz con el usuario y un firmware que realizara el envío de la traza que se desea replicar. El proceso de replicación de la traza se va a hacer en dos partes.

- La primera parte consiste en enviar la traza almacenada en un fichero con formato PCAP a una memoria local de la tarjeta COMBOv2. Con esta precarga de los datos conseguimos que el envío de los paquetes a la interfaz de red, no se vea influenciado por la carga de trabajo del sistema operativo, ni por las latencias del stack TCP/IP. Además, el módulo DMA proporcionado por la plataforma NetCOPE alcanza como máximo tasas de alrededor de los 600Mbps, y de esta manera ésta limitación solo afecta a la velocidad de carga de los paquetes pero no a la velocidad de envío.

- La segunda parte consiste en el envío de los paquetes cargados previamente. Los paquetes se leerán de la memoria de la tarjeta y un módulo de envío los reconstruirá manteniendo los tiempos entre paquetes especificados por el usuario con una precisión de 8 ns (La máxima permitida por el reloj de 125Mhz de la tarjeta).

El software de control hará de interfaz con el usuario y le proporcionara las opciones necesarias para: cargar la traza a la memoria, cargar los tiempos entre paquetes de la traza, establecer un tiempo entre paquetes fijo, modificar los tiempos entre paquetes originales de la traza en tiempo real y distintas opciones de envío. El software se comunicará a través de dos caminos de datos diferentes con la parte firmware (ver Figura 3.7).

Camino de control: Este camino se utilizará para controlar y monitorizar el estado del hardware e interactuar con él. Este camino se va a implementar utilizando el bus de interconexión LocalBus y una interfaz de registro. La interfaz de registros proporcionará una serie de registros de control en los

que el programa de control podrá escribir para controlar el funcionamiento de los módulos, y otro grupo de registros que podrá leer en los que los distintos módulos irán almacenando información relevante para el usuario. Este tipo de información no requiere de tasas de transferencia elevadas ni supone un alto volumen de datos por lo que lo más adecuado es utilizar el LocalBus para este camino de datos.

A través de este camino también se van a cargar los tiempos entre paquetes de la traza a replicar. Cuando el registro de control que indica que se van a cargar tiempos entre paquetes se active, el módulo de interfaz de memoria empezará a escuchar en una dirección concreta del LocalBus a la que el software de control enviará los tiempos entre paquetes leídos del fichero PCAP y almacenará éstos datos en el espacio de direcciones asignado a los tiempos entre paquetes.

Camino de datos: Este camino comunica el software con el hardware mediante el módulo DMA, y su función es la de transferir los datos de la traza que se quiere replicar a la memoria DDR2 de la tarjeta COMBOv2. El software se comunica con el DMA de manera transparente, ya que la interfaz que éste proporciona al software es la de una interfaz de red virtual. El DMA transmite todo el tráfico enviado a la interfaz virtual a un módulo que hace de interfaz con la memoria DDRII, y éste lo almacena en las direcciones adecuadas con un formato determinado. En este caso se utiliza el protocolo FrameLink.

Una vez que todos los datos necesarios para la reconstrucción de la traza se han cargado en la memoria de la tarjeta, el usuario podrá indicar a través del programa de control que se inicie uno de los tipos de envío disponibles. En ese momento, la interfaz de memoria utilizara los datos almacenados en la memoria DDRII para reconstruir los paquetes y enviárselos al módulo de envío. Éste se encargará de interactuar con las interfaces de red y de asegurarse de que los paquetes se envían a la red con el modelo de tiempos adecuado.

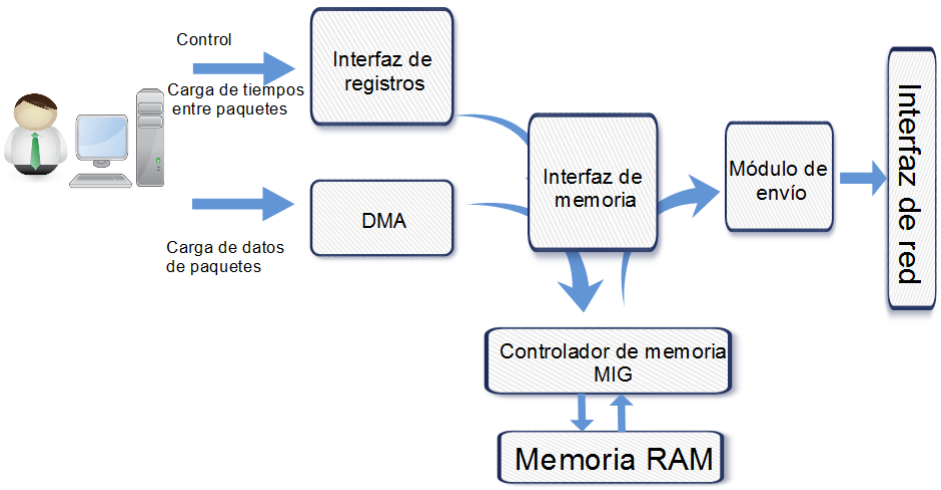

Figura 3.7: Flujo de datos

3.7. Diseño del software

Para proporcionar al usuario una interfaz fácil de utilizar que le abstraiga de los registros internos del módulo de replicación, se va a diseñar una aplicación software sencilla que permite al usuario realizar los pasos necesarios para replicar una traza de red. Su desarrollo se va a realizar en C y se van a utilizar las funciones proporcionadas por la librería *libcombo* para implementar la comunicación con los registros de control del módulo de replicación y la librería *libpcap* para la extracción de los datos del fichero PCAP.

La aplicación presenta al usuario las siguientes opciones:

1. **Load data**
   La opción 1 permite al usuario seleccionar un fichero pcap para cargar los datos de paquetes en la memoria ram. Para ello, se van a utilizar las librerías implementadas por INVEATech, que se integran con la aplicación tcpreplay permitiendo el envío de manera transparente a través de las interfaces virtuales de la plataforma NetCOPE..

2. **Load interpackets**
   La opción 2 permite al usuario seleccionar un fichero pcap para cargar los datos de tiempos entre paquetes en la memoria ram.

3. **Set interpacket offset (also used for uniform mode)**
   La opción 3 permite al usuario establecer un valor numérico en nanosegun-

dos que se utilizará como tiempo entre paquetes cuando se envía en modo uniforme o como offset si se quiere modificar un modelo de tiempos cargado en memoria.

4. **One time replication**
   La opción 4 inicia la replicación de los datos cargados con las opciones 1 y 2. Cuando se llega al final de la traza, el envío se detiene y el módulo espera a que el usuario introduzca una nueva opción. Si no se han cargado tiempos entre paquetes no se iniciará el envío.

5. **Loop replication**
   La opción 5 inicia la replicación de los datos cargados con las opciones 1 y 2. Cuando se llega al final de la traza, vuelve a empezar a enviar la traza desde el principio. El módulo continuará replicando el tráfico hasta que el usuario introduzca la opción 8. Si no se han cargado tiempos entre paquetes no se iniciará el envío.

6. **One time uniform mode**
   La opción 6 inicia la replicación de los datos cargados con la opción 1. Cuando se llega al final de la traza, el envío se detiene y el módulo espera a que el usuario introduzca una nueva opción.

7. **Loop uniform mode**
   La opción 7 inicia la replicación de los datos cargados con la opción 1. Cuando se llega al final de la traza, vuelve a empezar a enviar la traza desde el principio. El módulo continuará replicando el tráfico hasta que el usuario introduzca la opción 8.

8. **Stop Loop**
   La opción 8 detiene el envío cuando se ha seleccionado previamente la opción 5 u 8.

9. **Exit**
   La opción 9 finaliza el programa.

10. **Read regs(debug)**
    La opción 10 muestra los registros de estado del módulo de replicación. Esta opción se ocultará en el programa final, pues tiene como objetivo facilitar la depuración de errores.

Capítulo 4

Implementación

4.1. Implementación hardware

A continuación se va a detallar la implementación de los distintos módulos hardware de los que se compone la aplicación desarrollada.

4.1.1. User_core

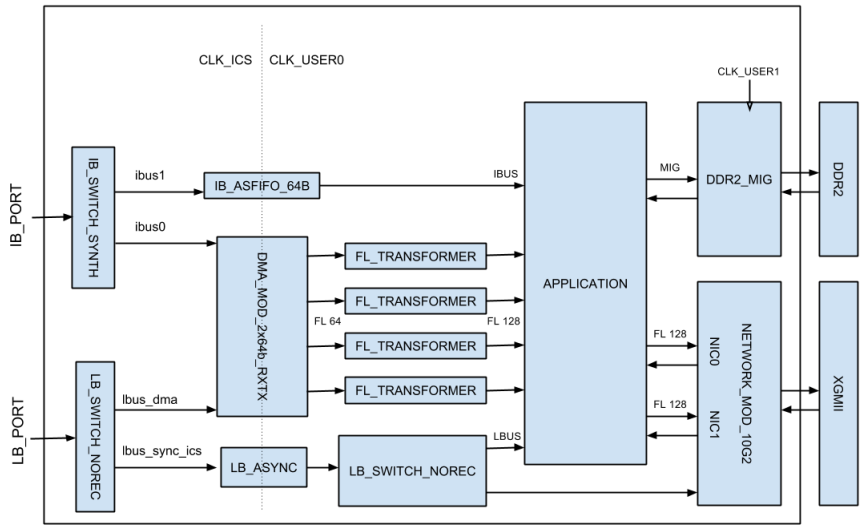

Figura 4.1: Diagrama UserCore

El módulo *user_core* está proporcionado por INVEATech como parte del entorno de desarrollo NetCOPE. En él se realizan los cambios de dominio de reloj y se instancian los controladores de las memorias DDRII y QDR. Contiene también el módulo *application*, un módulo contenedor que proporciona todas las interfaces necesarias para la implementación de aplicaciones de red. En él estará encapsulado el módulo superior de nuestra aplicación. En la figura 4.1se presenta un diagrama con todos los módulos y protocolos de interconexión de los mismos que contiene el módulo *user_core*.

4.1.2. Relojes y resets

La generación de relojes se realiza en un módulo superior a *user_core* llamado *clk_gen_cv2*. Este módulo recibe el reloj de 250Mhz procedente del controlador de PCIe y, utilizando 2 PLL, genera dos conjuntos de relojes: uno con valores

fijos, y otro con valores variables. En nuestro diseño se han utilizado 3 relojes generados por el segundo PLL: uno de 125Mhz (clk_ics) que se utiliza para los buses de interconexión con el resto de módulos proporcionados por INVEATech, otro reloj de 125Mhz (clk_user0) que se utiliza como reloj de sistema de nuestra aplicación y que es común a todos los bloques, y un tercero de 200Mhz (clk_user1) que necesita el controlador de memoria DDR MIG para generar las señales DQS (ver figura 4.2).

4.1.3. Controlador de memoria MIG

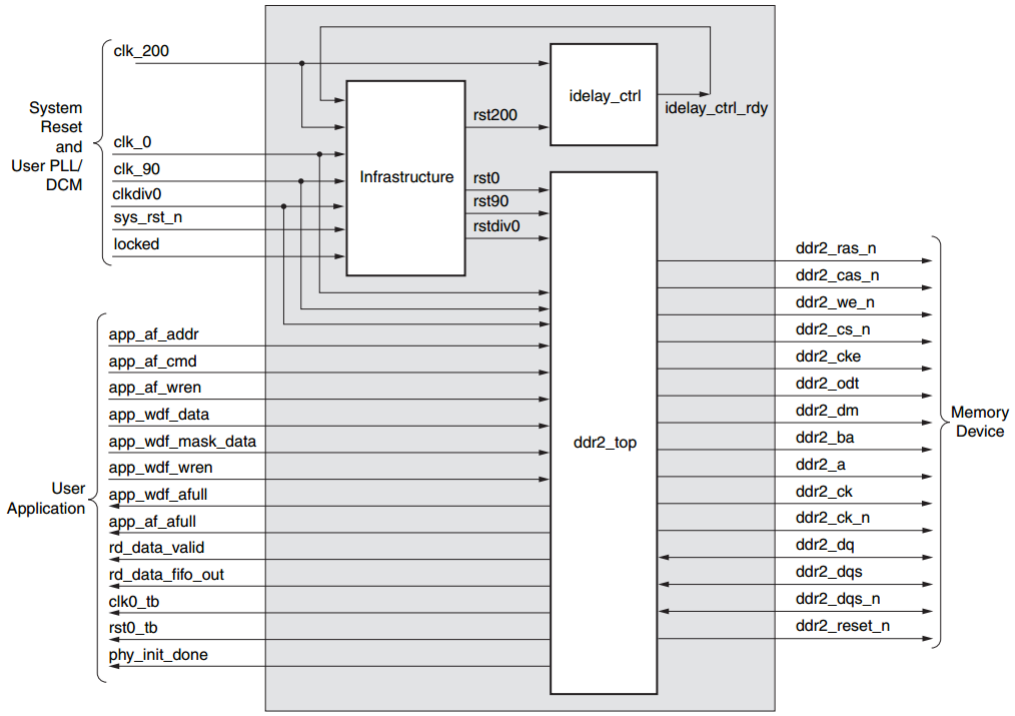

Figura 4.2: Diagrama controlador MIG

Para implementar el controlador de la memoria DDR se ha utilizado la herramienta CoreGen de Xilinx modificando el fichero de constrains generado para adaptarlo al entorno de trabajo ComboV2. Se ha utilizado una memoria de 2GB de Crucial, que utiliza el chip MT16HTF25664HZ-667 de Micron, con part number MT47H128M8. Los parámetros de configuración del controlador MIG se han extraído de la documentación técnica del chip de la memoria [28], proporcionada por Micron, y del modelo de referencia para la integración del controlador MIG en la plataforma NetCOPE proporcionado por el vendedor de Combo V2 (INVEATech). A continuación se presentan todos los parámetros de configuración

empleados en el controlador (Tabla 4.2).

Parámetros de configuración del MIG

Parámetro	Valor	Descripción
BANK_WIDTH	3	Número de bits para direccionar bancos.
CKE_WIDTH	2	Número de enables de reloj. Se ha seleccionado 2 porque la memoria utilizada se compone de 2 chips.
CLK_WIDTH	2	Número de relojes. Se ha seleccionado 2 porque la memoria utilizada se compone de 2 chips.
COL_WIDTH	10	Número de bits para direccionar columnas.
CS_NUM	2	Número de bits de selección de chip diferentes. Se ha seleccionado 2 porque la memoria utilizada se compone de 2 chips.
CS_WIDTH	2	Número de bits para seleccionar chip.
CS_BITS	1	log2(CS_NUM)
DQ_WIDTH	64	Ancho de datos de la memoria.
DQ_PER_DQS	8	Número bits por cada señal de strobe.
DQS_WIDTH	8	Número de señales de strobe.
DQ_BITS	6	log2(DQS_WIDTH*DQ_PER_DQS)
DQS_BITS	3	log2(DQS_WIDTH)
ODT_WIDTH	2	Número de señales de enable para terminaciones dentro de un mismo chip.
ROW_WIDTH	14	Número de bits para direccionar filas.
ADDITIVE_LAT	0	Parámetro que permite aumentar la latencia del controlador.
BURST_LEN	4	Número de palabras consecutivas de 64 bits a las que se accede por cada dirección.
BURST_TYPE	0	Tipo de BURST_LEN. 0: secuencial 1: entrelazada
CAS_LAT	3	Latencia de acceso a columna.
ECC_ENABLE	0	Enable del código de corrección de errores.
APPDATA_WIDTH	128	Número de bits del bus de datos de usuario.
MULTI_BANK_EN	1	Enable que permite tener abiertos varios bancos simultáneamente.
TWO_T_TIME_EN	1	Valor por defecto.

Parámetro	Valor	Descripción
ODT_TYPE	1	Impedancia de las terminaciones ODT. 1(75), 2(150), 3(50).
REDUCE_DRV	0	Valor por defecto.
REG_ENABLE	0	Registro de las señales addr/ctrl.
TREFI_NS	7800	Intervalo de refresco (ns).
TRAS	40000	active->precharge delay
TRCD	15000	active->read/write delay
TRFC	127500	refresh->refresh, refresh->active delay
TRP	15000	precharge->command delay
TRTP	7500	read->precharge delay
TWR	15000	used to determine write->precharge
TWTR	7500	write->read delay
HIGH_PERFORMANCE_MODE	TRUE	TRUE, IODELAY de alto rendimiento FALSE, IODELAY de bajo rendimiento
SIM_ONLY	0	Si vale 1 se salta el tiempo de espera de encendido de la SDRAM para reducir el tiempo de simulación.
DEBUG_EN	0	Habilita las señales de depuración.
CLK_PERIOD	8000	Período del reloj de la memoria (ps).
DLL_FREQ_MODE	HIGH	Rango de frecuencias del DCM.
CLK_TYPE	SINGLE EN-DED	Tipo de reloj de entrada: DIFFERENTIAL/SINGLE_ENDED.
NOCLK200	FALSE	Habilita la entrada de reloj de 200 MHz.
RST_ACT_LOW	1	1= reset activo bajo 0= reset activo alto

Cuadro 4.2: Parámetros de configuración del MIG

A la hora de utilizar, modificar y verificar la implementación del controlador de memoria DDRII es necesario entender la estructura interna de la memoria y cómo se direccionan los datos con el bus de direcciones (señal app_af_addr) que proporciona la interfaz de usuario del controlador (interfaz user application de la figura 4.2).

La memoria utilizada se compone de 2 chips de 1 GB. Cada chip tiene un ancho de datos (DQ_WIDTH) de 64 bits, o lo que es lo mismo 2^3Bytes. Éste será el tamaño mínimo de palabra y por lo tanto la cantidad de bytes que almacena cada

dirección de memoria. Sabiendo que serían necesarios 31 bits para direccionar 2GB con palabras de un byte, con palabras de 8 bytes necesitamos $31-3=28$ bits. Por esta razón, sólo se van a utilizar los bits del 0 al 27 del bus de direcciones de los 31 bits que nos proporciona el controlador de memoria (ver figura 4.3). Cada chip se selecciona utilizando el bit 27 del bus de direcciones. Por lo tanto, las primeras direcciones corresponden a un chip y las últimas a otro.

A la hora de utilizar la memoria hay que tener en cuenta dos puntos importantes:

1. Los bits del 24 al 26 se utilizan para direccionar bancos dentro de un mismo chip. Acceder por primera vez a una dirección de un banco requiere una operación de apertura de banco y en esta tarjeta de memoria pueden estar abiertos hasta 4 bancos simultáneamente.

2. El acceso a una nueva fila requiere también de un comando de apertura de fila pero en este caso no se pueden mantener abiertas varias filas.

Por estas razones es conveniente que los accesos a la memoria se hagan de una manera secuencial, intentando siempre acceder a posiciones de memoria consecutivas y con pocos cambios de bancos.

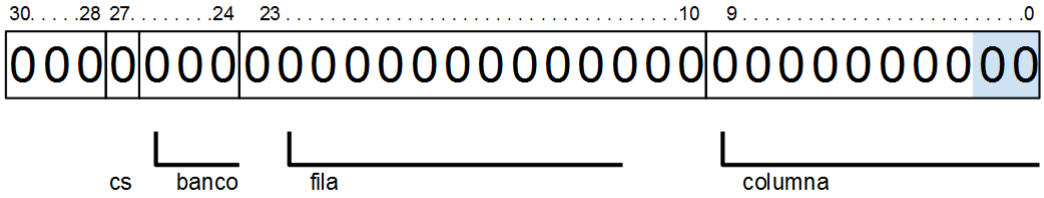

Figura 4.3: Direccionamiento MIG

En el caso de este diseño, se ha configurado el controlador de memoria para que acceda con ráfagas de 4 direcciones escribiendo o leyendo bloques de 256 bits en 2 ciclos de reloj (128 bits por ciclo). Esta configuración hace que no se utilicen los 2 bits menos significativos del bus de direcciones para direccionar y que haya que incrementar las direcciones de 4 en 4 para acceder a direcciones consecutivas.

Estas características se han tenido en cuenta a la hora de decidir la manera en la que se han almacenado los paquetes, longitudes y tiempos entre paquetes en las direcciones de memoria disponibles. Se ha decidido dividir la memoria en dos espacios de direcciones: uno en las direcciones inferiores para almacenar los datos y su longitud de manera consecutiva (cada longitud seguida de los datos), y otro espacio en las direcciones superiores para almacenar los tiempos entre paquetes.

De esta manera se accederá siempre a direcciones consecutivas salvo cuando sea necesario pedir tiempos entre paquetes que se almacenarán en un buffer de 8 en 8. Esta división maximiza los accesos a direcciones consecutivas y limita la mayor parte del tiempo el número de bancos abiertos a 2.

Para calcular la dirección máxima que se puede utilizar para el almacenamiento de datos (paquetes y longitudes) se deben tener en cuenta varias consideraciones:

- En el peor caso deben caber todos los tiempos entre paquetes en el espacio de memoria reservado. El peor caso es cuando todos los paquetes de la traza tienen el mínimo tamaño posible (64 bytes), lo que supone almacenar mayor cantidad de paquetes y por tanto mayor cantidad de tiempos entre paquetes.
- Debido al diseño del controlador DDRII para almacenar los datos de un paquete de 64bytes hacen falta realmente 96 bytes en memoria. Esto se debe a que las escrituras se hacen en ráfagas de 32 bytes, y al tener que almacenar la longitud en la misma dirección que la primera parte de los datos del paquete, realmente se almacena en memoria 16 bytes de longitud, 64 bytes de datos y 16 bytes de ceros para completar la escritura de la última ráfaga. Teniendo en cuenta esta limitación de la memoria y que el tamaño de los tiempos entre paquetes es de 8 bytes, el cálculo del número máximo de paquetes que se puede almacenar en la memoria de 2 GB es el siguiente:

$$x * (8 + 64 + 16 + 16) = (1024)^3 * 2 \implies x = 20648881\, paquetes$$

Con este cálculo, la asignación de espacio de memoria para datos, longitudes de paquetes y tiempos entre paquetes es la siguiente:

```
Datos -> 1651 MB
Longitudes -> 330MB
Datos+Longitudes -> 1982 MB
ITPs -> 165 MB
```

A la hora de crear la traza para cargarla en la memoria, sin embargo, hay que tener en cuenta los espacios de memoria que no se utilizan debido a la escritura por ráfagas. En el caso de paquetes de 64 bytes la eficiencia en el almacenamiento de datos es del 80 % por lo que realmente sólo se podrán almacenar 1321 MB de datos.

Con los datos calculados anteriormente y sabiendo que el tamaño de palabra de la memoria es de 64 bits, la dirección máxima que se puede utilizar para almacenar datos es la “0EC4EC4C”. Para reducir la complejidad de las comparaciones en la lógica y para evitar posibles errores se va a utilizar durante la implementación la dirección “0EC40000”.

Para integrar el controlador DDRII generado con la herramienta *coreGen* en el entorno de la plataforma NetCOPE se han realizado ciertas modificaciones sobre el modelo de referencia proporcionado por INVEATech.

1. En primer lugar, se han añadido tanto en el diseño VHDL como en el fichero de constrains las señales de dataMask. Estas señales en el modelo de referencia no se utilizaban, mientras que en el diseño realizado deben permanecer activas para el correcto funcionamiento de la memoria.

2. Se han añadido algunos emplazamientos manuales de FlipFlops y excepciones para caminos críticos del controlador para calibrar correctamente la memoria. Para ello se han añadido las siguientes líneas en el fichero UCF

```
INST "*/gen_dq[*].u_iob_dq/gen*.u_iddr_dq" TIG ;
INST "*/u_mem_if_top/u_phy_top/u_phy_io/u_phy_calib/gen_gate[*].
    u_en_dqs_ff" TNM = EN_DQS_FF;
TIMESPEC TS_FROM_EN_DQS_FF_TO_DQ_CE_FF = FROM EN_DQS_FF TO
    TNM_DQ_CE_IDDR 3.85 ns DATAPATHONLY;

INST "*/u_phy_calib/gen_gate[0].u_en_dqs_ff" LOC = SLICE_X0Y151;
INST "*/u_phy_calib/gen_gate[1].u_en_dqs_ff" LOC = SLICE_X0Y150;
INST "*/u_phy_calib/gen_gate[2].u_en_dqs_ff" LOC = SLICE_X0Y149;
INST "*/u_phy_calib/gen_gate[3].u_en_dqs_ff" LOC = SLICE_X0Y131;
INST "*/u_phy_calib/gen_gate[4].u_en_dqs_ff" LOC = SLICE_X0Y130;
INST "*/u_phy_calib/gen_gate[5].u_en_dqs_ff" LOC = SLICE_X0Y129;
INST "*/u_phy_calib/gen_gate[6].u_en_dqs_ff" LOC = SLICE_X0Y111;
INST "*/u_phy_calib/gen_gate[7].u_en_dqs_ff" LOC = SLICE_X0Y110;

NET "*/u_phy_io/gen_dqs*.u_iob_dqs/en_dqs_sync" MAXDELAY = 800 ps;

INST "*/gen_dqs[*].u_iob_dqs/u_iddr_dq_ce" TNM = "TNM_DQ_CE_IDDR";
INST "*/gen_dq[*].u_iob_dq/gen_stg2_*.u_iddr_dq" TNM = "TNM_DQS_FLOPS
    ";
TIMESPEC "TS_DQ_CE" = FROM "TNM_DQ_CE_IDDR" TO "TNM_DQS_FLOPS" 3.6 ns
    ;
```

3. Se han modificado las definiciones de las constrains de reloj para añadir un INPUT_JITTER de 100 ps. Estas constrains se han propagado manualmente a través de los distintos PLL para poder añadir al fichero UCF las definiciones de constrains multiciclo generadas por *coreGen,* pero referenciadas a los relojes de nuestro sistema.

4. Por la manera en la que se generan los relojes de la aplicación, ha sido necesario modificar el código del DCM que se incluye en el código generado por CoreGen. El cambio que se ha realizado es la eliminación de los buffers de entrada al DCM que ya están instanciados en la salida del PLL que genera el reloj que alimenta la memoria.

4.1.4. Generador_trafico

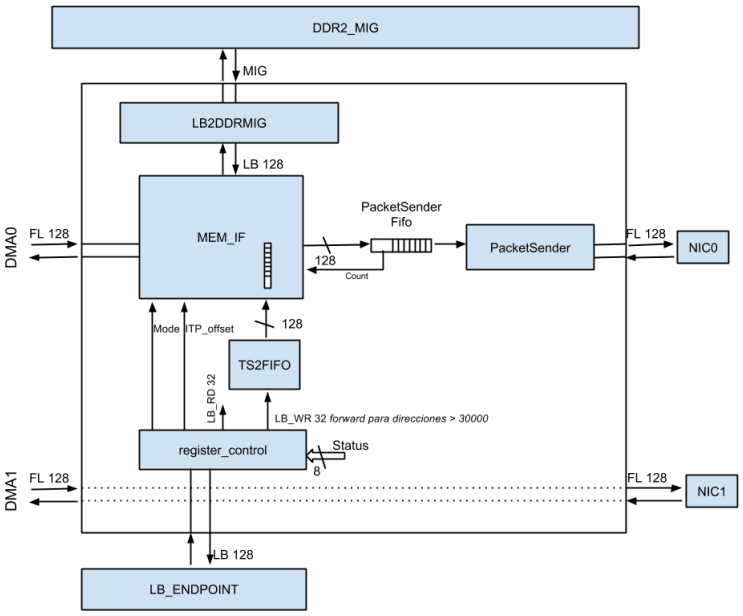

Figura 4.4: Diagrama TraficGenerator

El módulo *Generador_trafico* es el bloque superior del diseño. Su única función es interconectar los distintos módulos que componen el diseño. Funcionalmente se pueden dividir en 3(ver figura 4.4): interfaz de registros (*register_control*), interfaz con la memoria (*mem_if)* y módulo de envío de paquetes (*PacketSender)*.

El módulo *Generador_trafico* presenta 6 interfaces:

- Dos interfaces con protocolo FrameLink para comunicarse con los dos DMAs que gestionan la comunicación con el servidor.
- Dos interfaces con protocolo FrameLink para comunicarse con los PHY de las interfaces de red.
- Una interfaz con protocolo MIG para la comunicación con el controlador de la memoria DDRII.
- Una interfaz local bus para comunicación con el usuario.

De las interfaces anteriores no se han utilizado en el diseño las de conexión con el DMA1 y con la NIC1(interfaz de red 1). En lugar de utilizarlas, se han conectado la una a la otra para que funcionen como una interfaz estándar con el fin de utilizarla como interfaz de captura durante las pruebas iniciales.

Tampoco se ha utilizado el enlace de recepción del DMA0 y de la NIC0 ya que sólo se contempla el envío de tráfico, no la captura.

4.1.5. register_control

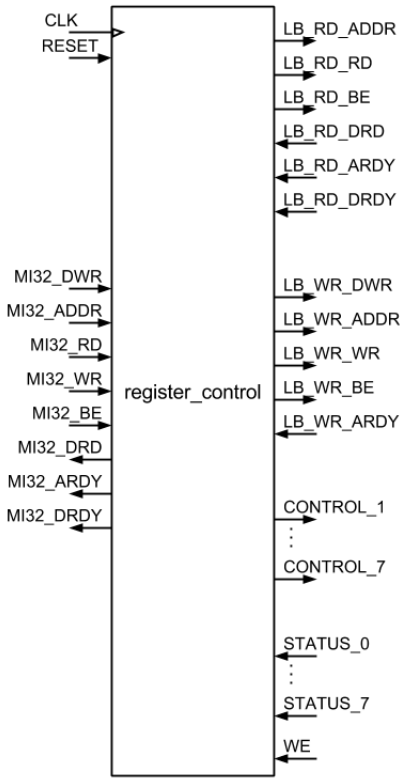

Figura 4.5: Diagrama RegisterControl

Puertos del módulo register_control

Nombre de señal	Dirección	Descripción
CLK	Input	Reloj de 125Mhz.
RESET	Input	Reset asíncrono de sistema.
MI32_LB	-	Interfaz localBus con el usuario.
LB_WR_DWR[31:0]	Output	Redirección de señales de escritura del localBus.
LB_WR_ADDR[31:0]	Output	
LB_WR_WR	Output	
LB_WR_BE[3:0]	Output	

Nombre de señal	Dirección	Descripción
LB_WR_ARDY	Input	
LB_RD_ADDR[31:0]	Output	Redirección de señales de lectura del localBus. Como se pensaba en un principio leer de una memoria con un bus de datos de 64 bits se ha adaptado la lectura para que una dirección de memoria sean 2 direcciones de localBus.
LB_RD_RD	Output	
LB_RD_BE[7:0]	Output	
LB_RD_ARDY	Input	
LB_RD_DRD[63:0]	Input	
LB_RD_DRDY	Input	
CONTROL 1-7[31:0]	Output	7 registros a través de los cuales el usuario envía información a la aplicación. El octavo registro es de uso interno para el módulo register_control y es usado como registro de paginación.
STATUS 0-7[31:0]	Input	8 registros de sólo lectura para el usuario que se utilizan principalmente para tareas de depuración.
WE	Input	Señal que permite la actualización de los registros de estado cuando está activa.

Cuadro 4.4: Puertos del módulo register_control

El módulo *register_ control* (figura 4.5) se encarga de proporcionar al usuario una interfaz de registros a través de la cual comunicarse con la aplicación. Ésta comunicación se realiza a través de la interfaz localBus proporcionada por NetCOPE (ver tabla 4.4).

El entorno de trabajo de NetCOPE tiene reservadas las direcciones 20000 - 3FFFF para la comunicación por localBus. Este espacio de direcciones se ha dividido de tal manera que las primeras n direcciones se utilicen como registros de comunicación con el usuario y el resto se redireccionen a otros módulos en caso de que sean necesarios. Para aumentar el espacio de direccionamiento de las direcciones redireccionadas, se ha utilizado uno de los registros de usuario como registro de paginación.

En un principio, se pretendió almacenar los tiempos entre paquetes en las memorias QDR proporcionadas por la plataforma NetCOPE, por lo que se utilizó parte de las direcciones de localBus para direccionar la memoria QDR. Posteriormente, se determinó que para almacenar todos los tiempos entre paquetes en la memoria QDR sólo se podían utilizar 16 bits para cada tiempo. Este valor limita el tiempo entre paquetes a un máximo de 65535 ns (lo que supone una tasa binaria mínima

de 7.8Mbps) lo cual no satisface los objetivos del diseño que pretende ser capaz de replicar trazas con tasas binarias inferiores. Por esa razón, se decidió abandonar esta opción y en vez de utilizar la memoria QDR para tiempos entre paquetes y la memoria DDRII para datos, repartir el espacio de direcciones de la memoria DDRII para almacenar en ella tanto datos como tiempos entre paquetes, aunque se reduzca con ello la cantidad de datos que se pueden replicar en unos 200MB. Por lo tanto, la redirección, aunque implementada en el módulo, no se va a utilizar para acceder a la QDR. Solo se va a utilizar la redirección en escritura para enviar los tiempos entre paquetes a una fifo de la que posteriormente se leerán para ser almacenados en memoria.

Como registros de usuarios se han reservado las primeras 16 direcciones, las primeras 8 como registros de estado con fines de depuración, y las otras 8 como registros de control a través de los cuales el usuario puede comunicarse con la aplicación.

La asignación de registros es la siguiente:

```
STATUS_0   --Última dirección de datos
STATUS_1   --Última dirección de ITPs
STATUS_2   --Estado de PacketSender
STATUS_3   --Estado de mem_if
STATUS_4   --Estado de ddrReader
STATUS_5   --No utilizado
STATUS_6   --No utilizado
STATUS_7   --No utilizado
CONTROL_1  --Control de modo de funcionamiento
CONTROL_2  --ITP fijo establecido por el usuario
CONTROL_3  --No utilizado
CONTROL_4  --No utilizado
CONTROL_5  --No utilizado
CONTROL_6  --No utilizado
CONTROL_7  --No utilizado
```

Figura 4.6: Asignación de registros

4.1.6. ts2fifo

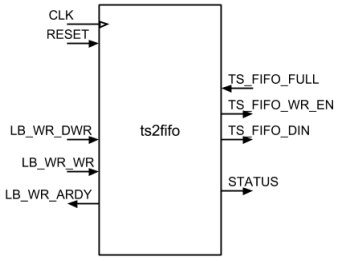

Figura 4.7: Diagrama ts2fifo

Puertos del módulo ts2fifo

Nombre de señal	Dirección	Descripción
CLK	Input	Reloj de 125Mhz.
RESET	Input	Reset asíncrono de sistema.
LB_WR_DWR[31:0]	Input	Señal de datos del protocolo LocalBus.
LB_WR_WR	Input	Señal de petición de escritura por parte del bus.
LB_WR_ARDY	Output	Señal de confirmación de la recepción de un dato.
TS_FIFO_DIN[127:0]	Output	Entrada de datos de la fifo de tiempos entre paquetes.
TS_FIFO_WR_EN	Output	Write enable de la fifo de tiempos entre paquetes. Siempre que esta señal esté a uno, si la fifo no esta llena, se captura el dato que se encuentra en FIFO_DIN.
TS_FIFO_FULL	Input	La señal FIFO_FULL se activa cuando la fifo de tiempos entre paquetes está llena. Esta señal detiene la recepción de datos a través del local bus desactivando la señal LB_WR_ARDY.
STATUS[31:0]	Output	Salida a los registros de depuración.

Cuadro 4.6: Puertos del módulo ts2fifo

El módulo *ts2fifo* (figura 4.7) es el responsable de almacenar los tiempos entre paquetes recibidos a través de la interfaz local bus de 32 bits, acumularlos hasta que se tengan 128 bits, e introducirlos en la fifo de tiempos entre paquetes para que la interfaz de memoria que lee de la fifo los almacene en la memoria DDRII.

El reset se activa cuando no se está en el modo de escritura de tiempos entre paquetes para limpiar cualquier dato acumulado anterior no utilizado.

4.1.7. PacketSender

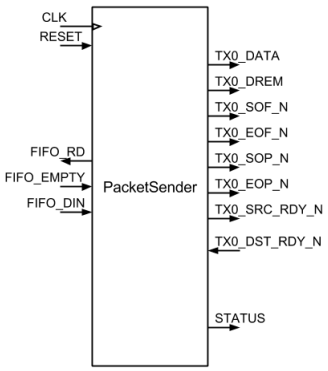

Figura 4.8: Diagrama PacketSender

Puertos del módulo PacketSender

Nombre de señal	Dirección	Descripción
CLK	Input	Reloj de 125Mhz.
RESET	Input	Reset asíncrono de sistema.
FIFO_DIN[127:0]	Input	Salida de datos de la fifo de envío.
FIFO_RD	Output	Read enable de la fifo de envío, siempre que esta señal esté a uno, si la fifo no esta vacía, se recibe un dato en FIFO_DIN.
FIFO_EMPTY	Input	La señal FIFO_EMPTY se activa cuando no hay ningún dato en la fifo de envío. Esta señal es la única señal de control del módulo PacketSender. Siempre que haya algún dato en la fifo el módulo la leerá y la intentará enviar.

Nombre de señal	Dirección	Descripción
TX0_FL_128	-	Conjunto de puertos que implementan el protocolo frame link de 128 bits para comunicarse con la interfaz de red.
STATUS[31:0]	Output	Salida a los registros de depuración.

Cuadro 4.8: Puertos del módulo PacketSender

El módulo *PacketSender* (figura 4.8) es el responsable de comunicarse con las interfaces de red para enviar los datos que lee de la fifo de envío. Además, se encarga de controlar que se cumplan los tiempos entre paquetes durante el envío.

El flujo de funcionamiento está controlado por la máquina de estados que se muestra en la figura 4.9.

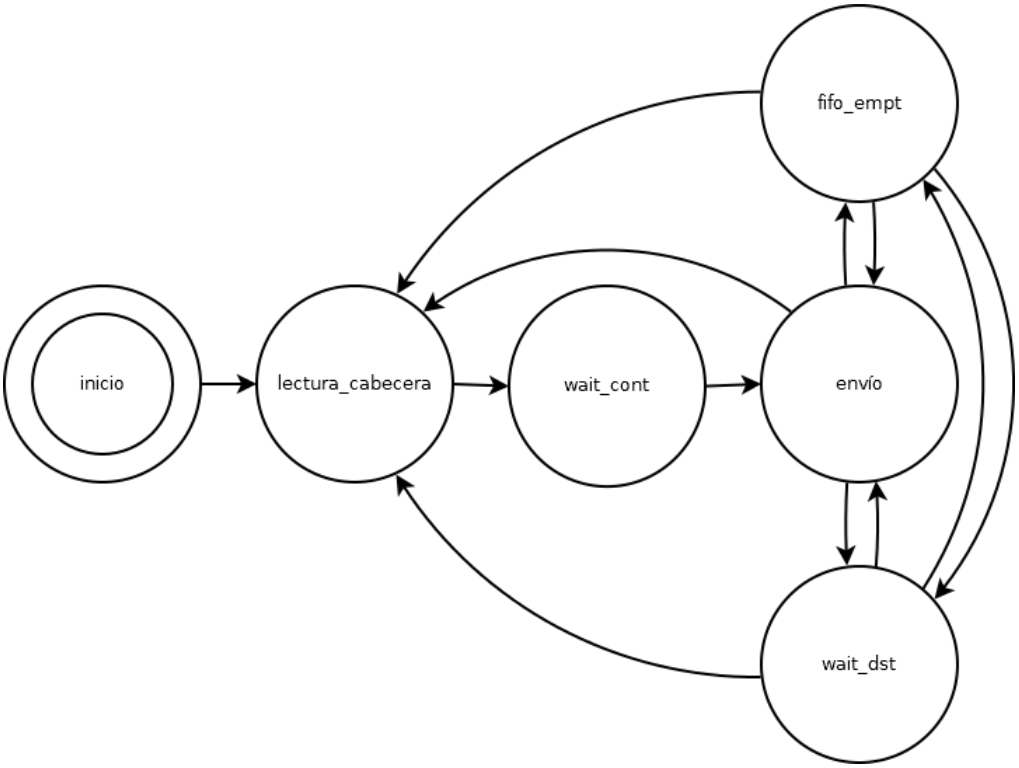

Figura 4.9: Diagrama de estados PacketSender

El flujo normal de funcionamiento sería el siguiente: Tras salir del reset se permanece en un estado inicial hasta que exista algún dato en la fifo de envíos. El primer dato que se lea se va a considerar una cabecera de envío (dato que contiene

la información de longitud y tiempo entre paquetes entre el paquete previo y el que se va a leer a continuación). Cuando se pide la cabecera a la fifo se cambia al estado *lectura_ cabecera,* que se encargará de procesar la cabecera.

A continuación, se cambiará al estado *wait_ cont,* en el que se espera hasta que el contador de tiempos entre paquetes alcance el dato de tiempo leído de la cabecera. Si al cambiar al estado *wait_ cont* el contador ya es superior al tiempo entre paquetes especificado en la cabecera, se activará un bit de estado que indica que no se ha conseguido cumplir con las restricciones de tiempos entre paquetes especificadas por el usuario.

El tiempo máximo permitido que se puede utilizar en la lectura de cabeceras, requerido para alcanzar la tasa binaria de 10GbE, viene determinado por el tiempo entre paquetes de 64 bytes enviados a 10 Gbps. Para alcanzar dicha tasa hay que enviar un paquete de 64 bytes cada 51,2 ns, que en ciclos de 125Mhz equivale a 6 ciclos. Teniendo en cuenta que el envío de los 64 bytes a través del bus de 128 bits requiere de 4 ciclos, disponemos de 2 ciclos como máximo para comenzar a enviar el siguiente paquete. En el diseño de la máquina de estados, teniendo en cuenta esta restricción, se han utilizado 2 estados para procesar la cabecera y verificar el tiempo entre paquetes, transición que, siempre que haya datos en la fifo disponibles, podrá realizarse en 2 ciclos si el tiempo entre paquetes lo requiere. Como posible mejora, si fuese necesario, se podría plantear unificar los 2 estados reduciendo la secuencialidad y por lo tanto empeorando los path delays del módulo.

Una vez esperado el tiempo necesario en el estado *wait_ cont* se comienza con el envío. Para ello, se dispone de 3 estados: uno de envío y otros 2 que contemplan los distintos eventos que pueden provocar que se detenga el envío (que deje de haber datos disponibles o que el PHY necesite ciclos de espera por superar la tasa de 10Gbps).

Tras finalizar el envío de una traza, la fifo quedará vacía y, al no detectarse ningún dato que se pueda leer como cabecera, la máquina de estados permanecerá en el estado *fifo_ empt* hasta que se introduzcan nuevos datos en la fifo.

4.1.8. lb2ddrmig

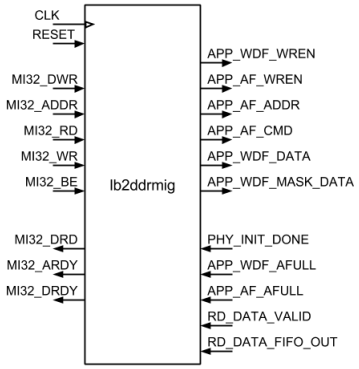

Figura 4.10: Diagrama lb2ddrmig

El módulo *lb2ddrmig* (figura 4.10) es el encargado de traducir el protocolo localBus que utiliza el módulo de interfaz de memoria, al protocolo MIG que utiliza el controlador de memoria DDR de Xilinx. Este módulo se ha introducido con el objetivo de hacer el diseño más reutilizable. Si se migrase el diseño a otra plataforma que utilice otro tipo de memoria, se podría reutilizar la interfaz con la memoria, cambiando simplemente el traductor de protocolo por uno que traduzca de un protocolo localBus modificado para que soporte envíos de ráfagas al protocolo que utilice el nuevo controlador de memoria.

4.1.9. mem_if

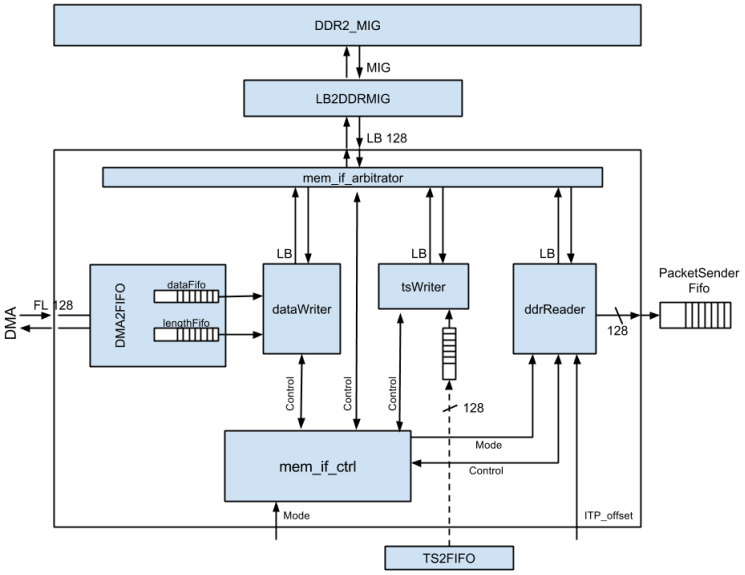

Figura 4.11: Diagrama mem_if

El módulo *mem_if* (figura 4.11) es el responsable de gestionar toda la comunicación con la memoria DDRII. Su función es controlar los distintos modos de funcionamiento que se proporcionan al usuario, y realizar las lecturas y escrituras en memoria que correspondan en cada caso.

Presenta dos interfaces de entrada: una conexión con protocolo *frame link* con el módulo DMA para la recepción de datos de paquetes, y una conexión con protocolo fifo con una fifo que contiene los tiempos entre paquetes que envía el usuario a través de un registro.

También implementa una interfaz con la memoria DDRII para la lectura y escritura de datos. Esta interfaz implementa un protocolo localBus ligeramente modificado para que acepte ráfagas de dos datos. Se ha utilizado este protocolo por ser sencillo y fácilmente adaptable a cualquier protocolo propio de cada memoria. De esta manera, se pretende maximizar la reutilización, ya que se puede adaptar el diseño fácilmente para trabajar con otro tipo de memorias redefiniendo los espacios de direcciones y sustituyendo el traductor de localBus a MIG por otro adecuado para la nueva memoria.

Por último, el módulo presenta una interfaz de salida a una fifo que almacena los paquetes para que posteriormente el módulo de envío los envíe a la interfaz de red. Los paquetes se han de introducir en la fifo de salida siguiendo una estructura

concreta para que el módulo de envío sea capaz de leerlos adecuadamente. Dicha estructura se presenta en la figura 4.12.

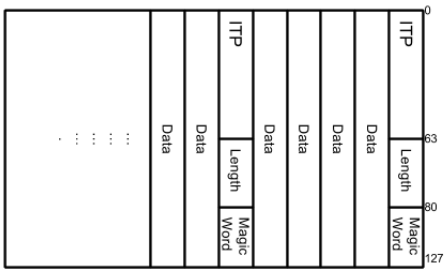

Figura 4.12: Estructura fifo de envío

Para simplificar el diseño y maximizar la modularidad, se ha dividido la lógica del módulo en 3 bloques funcionales:

- La escritura de los datos en memoria se va a implementar en los módulos **dma2fifo** y **dataWriter.**
- La escritura de los tiempos entre paquetes (ITPs) en memoria se va a implementar en el módulo **tsWriter**.
- La lectura de paquetes de memoria y el envío a la fifo de salida de estos se va a implementar en el módulo **ddrReader.**

Estos 3 bloques van a ser gestionados por 2 módulos de control:

mem_if_ctrl: Se encarga de controlar el flujo de funcionamiento de los distintos bloques.

mem_if_arbitrator: Se encarga de gestionar el acceso a la memoria entre los 3 módulos que necesitan acceder a ella.

A continuación, se expone detalladamente la implementación de los distintos submódulos que componen la interfaz con la memoria.

mem_if_ctrl

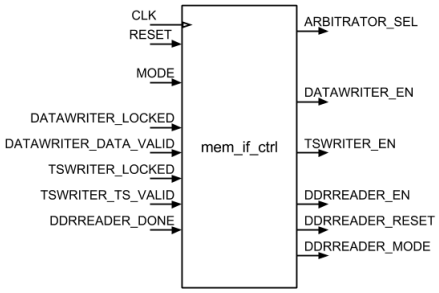

Figura 4.13: Diagrama mem_if_ctrl

El módulo *mem_if_ctrl* (figura 4.13) es responsable de gestionar todos los módulos que comprenden la interfaz con la memoria. Implementa una máquina de estados controlada por las señales procedentes de los distintos módulos, y por el registro de usuario MODO, a través del cual el usuario interactúa con ella cambiando entre los distintos modos de funcionamiento, que se corresponden en este caso con los estados de la máquina (ver figura 4.14).

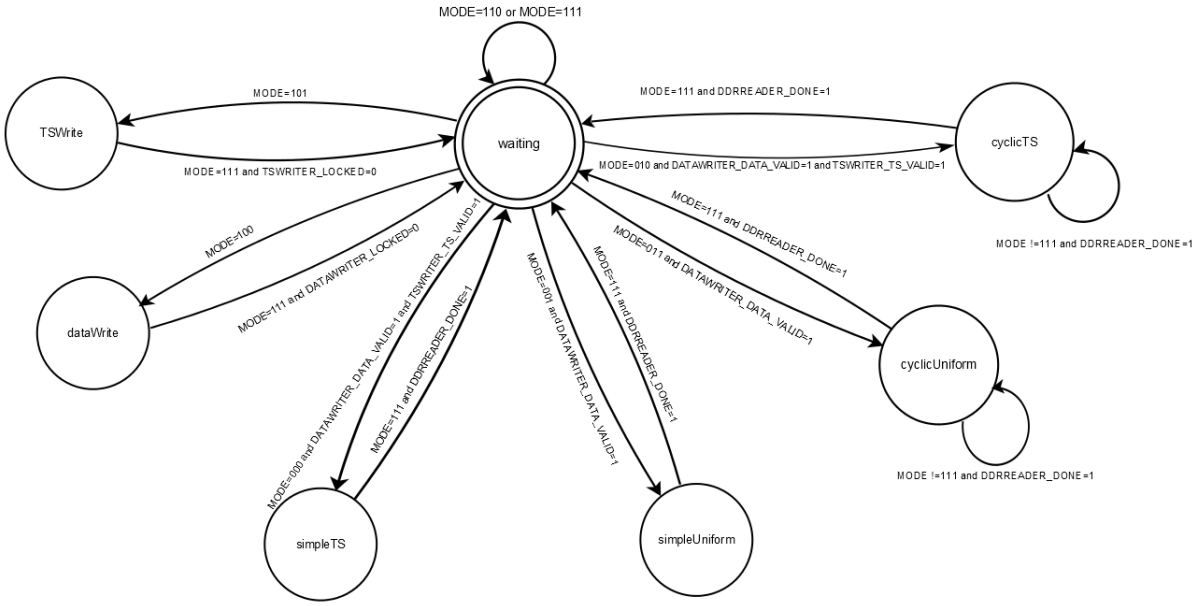

Figura 4.14: Máquina de estados mem_if_ctrl

La máquina de estados se ha diseñado con la intención de que no se pueda cambiar entre estados sin pasar por el estado inicial. De esta manera, se intenta que siempre se produzca un reset de las fifos entre los cambios de estados, para evitar por ejemplo, que en dos escrituras de datos consecutivas se pueda producir algún

error y se mezcle algún dato. Además, esta topología simplifica el diseño y no penaliza el rendimiento del sistema.

En función del estado en el que se encuentre y las señales de estado que reciba de los módulos, se activará un modo de funcionamiento u otro:

waiting: Estado inicial y de transición que asegura un estado inicial conocido para el resto de estados.

TSwrite: Estado de escritura de tiempos entre paquetes. En este estado se espera a que el usuario proporcione datos a través del registro 30000 y siempre que se reciba alguno, se escribe en la memoria en el espacio de direcciones reservado para tiempos entre paquetes.

dataWriter: Estado de escritura de datos de paquetes. En este estado se espera a que el usuario envíe paquetes por el DMA y, tras contar la longitud de cada paquete, se escriben en memoria en el espacio de direcciones reservado para datos de paquetes.

simpleTS: Estado de envío de paquetes con datos reales de tiempos entre paquetes. Solo se puede pasar a este estado si se han almacenado previamente datos reales de tiempos entre paquetes. Al terminar el envío se para hasta que se vuelva al estado waiting.

simpleUniform: Estado de envío de paquetes con tiempos entre paquetes uniformes especificados por el usuario. Solo se puede pasar a este estado si se han almacenado previamente datos de paquetes. Al terminar el envío se para hasta que se vuelva al estado waiting.

cyclicTS: Estado de envío de paquetes con datos reales de tiempos entre paquetes. Solo se puede pasar a este estado si se han almacenado previamente datos reales de tiempos entre paquetes. Al terminar el envío se activa el reset síncrono durante unos ciclos y se vuelve a comenzar a enviar desde la dirección inicial hasta que se cambie al estado waiting.

cyclicUniform: Estado de envío de paquetes con tiempos entre paquetes uniformes especificados por el usuario. Solo se puede pasar a este estado si se han almacenado previamente datos de paquetes. Al terminar el envío se activa el reset síncrono durante unos ciclos y se vuelve a comenzar a enviar desde la dirección inicial hasta que se cambie al estado waiting.

Las señales de salida de la máquina de estado son las siguientes:

Señales de salida máquina de estados de Mem_if_ctrl

State	ARBITRATOR_SEL	DATAWRITER_EN	TSWRITER_EN	DDRREADER_EN	DDRREADER_RESET	DDRREADER_MODE
TSWrite	10	0	1	0	1	1
dataWrite	01	1	0	0	1	1
simpleTS	00	0	0	1	0	0
simpleUniform	00	0	0	1	0	1
cyclicTS	00	0	0	1	1*	0
cyclicUniform	00	0	0	1	1*	1

Cuadro 4.9: Señales de salida máquina de estados de Mem_if_ctrl

ddrReader

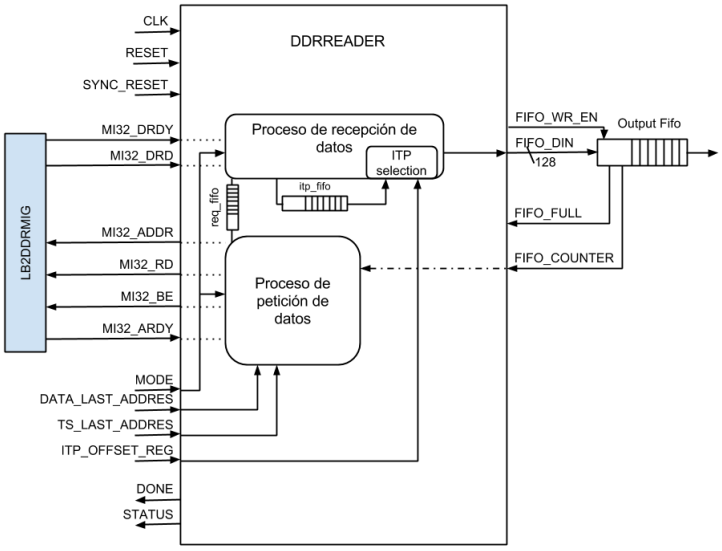

Figura 4.15: Diagrama ddrReader

Puertos del módulo ddrReader

Nombre de señal	Dirección	Descripción
CLK	Input	Reloj de 125Mhz.
RESET	Input	Reset asíncrono de sistema.
SYNC_RESET	Input	Reset síncrono para volver al estado inicial.
MI32_ADDR[31:0]	Output	Dirección de lectura DDRII.
MI32_RD	Output	Señal de control de lectura. Solo se lee la señal MI32_ADDR cuando esta señal es 1.

Nombre de señal	Dirección	Descripción
MI32_BE[15:0]	Output	Máscara de bytes que se quieren leer.
MI32_DRD[127:0]	Input	Datos devueltos por la memoria DDRII. El controlador de memoria está configurado con un BL(burst-length) de 4, por lo que por cada dirección pedida se recibirán 2 datos de 128 bits.
MI32_ARDY	Input	Señal de recepción de dirección. Indica que la petición de un dato se ha procesado y se puede pedir el siguiente.
MI32_DRDY	Input	Esta señal indica cuando vale 1 que hay un nuevo dato en el puerto MI32_DRD.
FIFO_DIN[127:0]	Output	Salida de datos hacia la fifo de envío.
FIFO_WR_EN	Output	Write enable de la fifo de envío. Siempre que esta señal esté a uno, si la fifo no esta llena se almacena el dato de FIFO_DIN.
FIFO_FULL	Input	La señal FIFO_FULL se pone a 1 cuando se ha ocupado completamente la fifo de envío. Esta condición no se controla ya que no existe una señal en la comunicación MIG que permita detener la recepción de datos de la memoria DDR. Si esta señal toma el valor 1 durante el funcionamiento, significará que ha habido un error irrecuperable.
FIFO_COUNTER[9:0]	Input	Contador que almacena el número de entradas utilizadas en la fifo. Esta señal se utiliza para detener la petición de datos a la memoria con el fin de evitar que la fifo se llegue a llenar.
DATA_LAST_ADDRES[31:0]	Input	Registro que se recibe del módulo de escritura de datos que indica cuál es la última dirección de la memoria que contiene datos.
TS_LAST_ADDRES[31:0]	Input	Registro que se recibe del módulo de escritura de interpackets que indica cuál es la última dirección de la memoria que contiene interpackets.

Nombre de señal	Dirección	Descripción
MODE	Input	La señal MODE indica al módulo si tiene que leer los tiempos entre paquetes de la memoria DDR y modificarlos con el registro de offset, o utilizar el registro de offset únicamente como tiempo entre paquetes. Los posibles valores que puede tomar son: 1: Uniforme (tiempos entre paquetes definidos por el usuario). 0: Réplica (tiempos entre paquetes leídos de memoria).
DONE	Output	Esta señal se activa cuando se ha alcanzado la dirección máxima en las lecturas y no queda ningún dato pendiente de recibir de la memoria. Su función es indicar al módulo controlador que se puede activar el reset síncrono para volver al estado inicial.
ITP_OFFSET_REG[31:0]	Input	Registro de usuario que determina el tiempo entre paquetes en modo uniforme, o la modificación de los tiempos entre paquetes reales en el modo réplica.
STATUS[31:0]	Output	Salida a los registros de depuración.

Cuadro 4.11: Puertos del módulo ddrReader

El módulo *ddrReader* (figura 4.15)es el responsable de leer los datos y los tiempos entre paquetes de la memoria DDR, y enviarlos a la fifo de salida en el orden adecuado y con la estructura que espera el módulo PacketSender, responsable del envío de paquetes. Permite 2 modos de funcionamiento: un modo réplica, en el que se leen los tiempos entre paquetes originales de la traza de la memoria, y tras aplicarles un offset se utilizan como tiempos entre paquetes en el envío, y un modo uniforme, en el que sólo se utiliza el valor de offset para establecer el tiempo entre paquetes.

Para ello necesita recibir las siguientes señales (ver tabla 4.11):

- Necesita recibir de los módulos de escritura en la DDR la dirección máxima utilizada, tanto en el espacio de direcciones reservado para tiem-

pos entre paquetes, como en el espacio de direcciones reservado para datos. Estas direcciones se reciben en los puertos DATA_LAST_ADDRES y TS_LAST_ADDRES.

- El offset se obtiene directamente de un registro que puede ser modificado interactivamente por el usuario. El puerto utilizado para este fin es el ITP_OFFSET_REG.
- Como se ha mencionado antes, el módulo puede funcionar en dos modos de funcionamiento: modo réplica y modo uniforme. El bit que se recibe del puerto MODE controla si se trabaja en uno u otro.

El flujo de la lógica del módulo está controlado por una máquina de estados muy simple que consiste en un estado de reposo inicial desde el que se puede pasar al estado réplica o al estado uniforme en función del valor de la señal MODE. A este estado sólo se puede volver si se recibe un reset. De esta manera se evita que se inicie el envío de datos sin que se controle el contenido de las fifos o con datos pendientes de devolver por parte de la memoria DDR. Habitualmente, el reset encargado de volver a situar la máquina de estados en el estado de reposo, será el reset síncrono que está controlado por el controlador de la interfaz de memoria mem_if_ctrl. En un comportamiento normal el reset síncrono se activará sólo cuando la señal DONE esté a uno, indicando así que se ha terminado de enviar todo el contenido de la memoria hasta la dirección máxima utilizada. Por lo tanto no se contempla en el flujo normal de la aplicación el envío parcial de una traza. Si se desea este comportamiento, habrá de controlarse en el momento de la carga de datos, habiendo modificado previamente el fichero pcap.

Como se ha comentado en la descripción de la señal FIFO_FULL, un punto crítico del módulo es el control del desbordamiento de la fifo de salida. Si en algún momento llegase a llenarse se perderían datos y se perdería la estructura necesaria para comunicarse con el módulo de envío(ver figura 4.12). Para asegurarnos de que la fifo de salida no se llena nunca, se va a detener la petición de datos a la memoria con suficiente antelación, de manera que haya espacio suficiente como para almacenar todos los datos pendientes de recibir de la memoria incluso si no se vacía ninguna posición de la fifo en ese tiempo. El valor de llenado de la fifo en el cual se va a parar de pedir nuevos datos a la memoria, se ha calculado como la capacidad máxima de la fifo menos la latencia de la memoria multiplicada por 2.

$$capacidad - (latencia * 2) - margen$$

La problemática de este cálculo reside en que la latencia de la memoria DDRII es variable, y creciente, ya que la memoria necesita ciclos adicionales para comandos internos de refrescos, abrir filas nuevas y cambiar de bancos y a que por cada di-

rección que se pide, el controlador MIG necesita 2 ciclos para devolver los datos. El hecho de que se pidan datos más rápido de lo que se reciben, ocasiona inevitablemente que se encolen peticiones y se aumente la latencia indefinidamente hasta que las fifos internas del controlador se llenen. Como no tenemos control de dichas fifos, se ha reducido la tasa de peticiones de datos dejando un ciclo entre petición y petición. De esta manera se evita el comportamiento creciente de la latencia, a la vez que se dejan ciclos de sobra para refrescos, aperturas de filas y cambios de bancos en la memoria.

La lógica del módulo se divide principalmente en 2 procesos: uno de lectura que controla las peticiones a la memoria, y otro de recepción de datos, que es responsable de componer la estructura de envío de los datos e introducirlos en la fifo de envío.

La lógica de los procesos presenta un flujo diferente en función de si se trabaja en modo réplica o en modo uniforme:

- **Flujo en modo uniforme**: En el modo uniforme el proceso de petición de datos pide datos consecutivos de la zona de memoria de datos siempre que no se llegue a la dirección máxima ni el contador de la fifo de envío pase del valor máximo calculado previamente. El proceso de recepción de datos procesa los datos devueltos por la memoria esperando que los primeros 128 bits indiquen la longitud de los datos almacenados en las direcciones posteriores (la longitud se almacena en los 16 bits más significativos). Cuando un contador de longitud indica que se han recibido todos los datos asociados al primer paquete se esperan otros 128 bits con la longitud del siguiente paquete, y así sucesivamente.
  Cuando se determina que se ha leído una longitud se concatena con el tiempo entre paquetes procedente del registro de offset (estos 128 bits los llamaremos cabecera de envío) y se introduce en la fifo de envío. Dicha concatenación se realiza de la siguiente manera: los bits [31:0] (el ITP es de 64 bits pero al proceder el offset de un registro de 32 bits, se concatena con ceros en los bits [63:32]) almacenan el tiempo entre paquetes y los bits [79:64] almacenan la longitud del paquete. Para controlar errores se ha introducido una constante en los bits no utilizados quedando los 128 bits de la siguiente manera:

$$x\text{"}5E7A00000000\text{"} \,\&\, MI32\_DRD(127 downto 112) \,\&\, x\text{"}00000000\text{"} \,\&\, ITP\_OFFSET\_REG$$

  Cuando se han pedido todos los datos, se espera un número de ciclos igual a la latencia de la memoria y si el contador de longitud de paquete indica

que se ha recibido el último paquete completo, se activa la señal DONE indicando así que se ha finalizado la lectura de paquetes.

- **Flujo en modo réplica:** En el modo réplica el proceso de petición de datos verifica en primer lugar si la fifo de ITP (tiempos entre paquetes) contiene suficientes datos como para que haya siempre ITPs disponibles en el peor caso: el número de paquetes de 64 bytes que puede proporcionar la memoria en el tiempo que tardaría en procesar la petición de nuevos ITPs. Este número es dependiente de la cola de peticiones a la DDR y por lo tanto, de la latencia. Suponiendo que la latencia sea constante se calcularía como:

$$ceil(latencia/((64/16)+1))$$

En este caso, se ha determinado que entre los almacenados en la fifo y los pedidos pendientes de recibir superen los 16 ITPs.
Mientras se disponga de un número suficiente de ITPs se piden datos de la misma forma que en el modo uniforme. Para comunicar al proceso que procesa los datos si el dato devuelto por la DDR es un ITP o un dato de paquete se ha implementado una fifo de peticiones en la que se almacena un 1 cada vez que se pide un ITP o un 0 si lo que se ha pedido es un dato. El proceso de recepción de datos lee de la fifo de peticiones cada vez que recibe un dato de la memoria DDR para decidir si el dato recibido es un ITP o un dato. Si es un ITP lo introduce en la fifo, y si es dato, se procede igual que en modo uniforme pero generando la cabecera de envío a partir de los datos leídos de la fifo de ITPs. Los 128 bits se componen de la siguiente manera:

$$x\text{"}5E7A00000000\text{"} \,\&\, MI32\_DRD(127 downto 112) \,\&\, itp\_fifo\_out + ITP\_OFFSET\_REG$$

Cuando se han pedido todos los datos se espera a que se vacíe la fifo de peticiones y se activa la señal DONE, indicando así que se ha finalizado la lectura de paquetes.

Para mejorar el path delay se han registrado las señales relacionadas con la recepción de datos, lo cual aumenta en un ciclo la latencia de acceso a la memoria. Este registro ha sido necesario también para poder tener un ciclo para decidir si es un ITP o un dato en el modo réplica.

Hay que tener en cuenta también que por la manera en que está implementado el módulo, si se llega antes al límite de direcciones de ITPs que al límite de datos, se dejaran de pedir ITPs y se leerá el último ITP válido utilizado para los paquetes

restantes.

dma2fifo

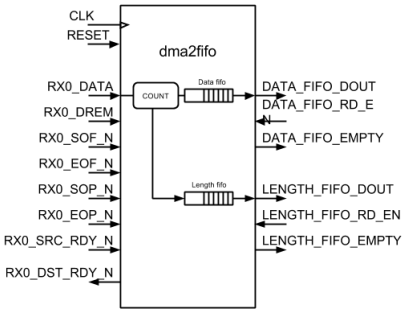

Figura 4.16: Diagrama dma2fifo

El módulo *dma2fifo* (figura 4.16) es el responsable de almacenar los datos de paquetes recibidos a través del DMA en dos fifos, una para los datos de los paquetes, y otra para las longitudes. Siempre que se recibe un dato se almacena en la fifo de datos y se incrementa el contador de la longitud. Cuando se recibe el último dato de un paquete, se guarda el valor de longitud en la fifo de longitudes y se pone el contador otra vez a cero.

El módulo se mantiene en estado de reset siempre que el modo de funcionamiento de mem_if no sea el de escritura de datos, de manera que siempre que se termine de guardar datos en memoria, al cambiar al estado *waiting,* el reset vacíe cualquier dato no utilizado de las fifos. La señal LOCKED del módulo dataWriter impide que se cambie de estado en medio de la escritura de un paquete en memoria.

dataWriter

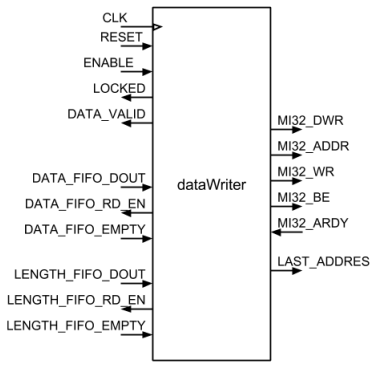

Figura 4.17: Diagrama dataWriter

Puertos del módulo dataWriter

Nombre de señal	Dirección	Descripción
CLK	Input	Reloj de 125Mhz.
RESET	Input	Reset asíncrono de sistema.
ENABLE	Input	Señal que habilita o deshabilita el módulo sin hacer reset de los registros.
LOCKED	Output	Señal que indica si se está escribiendo un tiempo entre paquetes o no. Su objetivo es evitar que el controlador pueda cortar la comunicación con la memoria en medio de una transferencia.
DATA_VALID	Output	Señal que se activa cuando se ha escrito en memoria al menos un dato.
DATA_FIFO_DOUT[127:0]	Input	Entrada de datos de paquetes.
DATA_FIFO_RD_EN	Output	Señal de lectura de la fifo de datos.
DATA_FIFO_EMPTY	Input	Señal que indica si hay algún dato disponible para leer en la fifo de datos.
LENGTH_FIFO_DOUT[15:0]	Input	Entrada de longitudes de datos.
LENGTH_FIFO_RD_EN	Output	Señal de lectura de la fifo de longitudes.
LENGTH_FIFO_EMPTY	Input	Señal que indica si hay algún dato disponible para leer en la fifo de longitudes.
LAST_ADDRES[31:0]	Output	Registro que almacena la dirección máxima utilizada.
MI32_DWR[127:0]	Output	Datos enviados a la memoria DDRII.
MI32_ADDR[31:0]	Output	Dirección de escritura en la DDRII.
MI32_WR	Output	Señal de control de escritura.
MI32_BE[15:0]	Output	Máscara de bytes que se quieren escribir.
MI32_ARDY	Input	Señal de recepción de dirección. Indica que la petición de escritura de un dato se ha procesado y se puede mandar el siguiente.

Cuadro 4.13: Puertos del módulo dataWriter

El módulo *dataWriter* (figura 4.17) es responsable de escribir en memoria los datos de los paquetes enviados por el usuario a través del DMA y previamente almacenados en 2 fifos: una de datos y otra de longitudes. El módulo espera a

que haya una longitud en la fifo de longitudes y la escribe en la memoria, para a continuación almacenar esa misma cantidad de datos leídos de la fifo de datos. Estos datos estarán ya almacenados en la fifo de datos en el momento de leer la longitud, ya que la longitud es calculada por el módulo dma2fifo contando los datos de paquetes introducidos en la fifo de paquetes.

Cada vez que se escribe en una dirección de memoria se actualiza la señal LAST_ADDRES (ver tabla 4.13), que informa al módulo de lectura de la memoria de cuál es la dirección máxima en la que hay datos válidos.

Para controlar que no se escriba fuera del espacio de direcciones de datos de paquetes ni se guarde en memoria un paquete incompleto, se comprueba que hay al menos espacio para 1500 bytes (tamaño máximo del paquete) en el espacio reservado para datos de paquetes en la memoria antes de leer una nueva longitud. Por eso, si en el fichero pcap a replicar hubiera paquetes mayores a 1500 bytes, podría producirse un error al almacenar el último paquete.

Para indicar al módulo de lectura que hay datos válidos que replicar, la señal DATA_VALID se activa si se ha guardado al menos un dato en la memoria.

tsWriter

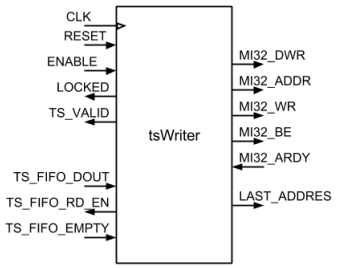

Figura 4.18: Diagrama tsWriter

Puertos del módulo tsWriter

Nombre de señal	Dirección	Descripción
CLK	Input	Reloj de 125Mhz.
RESET	Input	Reset asíncrono de sistema.
ENABLE	Input	Señal que habilita o deshabilita el módulo sin hacer reset de los registros.

Nombre de señal	Dirección	Descripción
LOCKED	Output	Señal que indica si se está escribiendo un tiempo entre paquetes o no. Su objetivo es evitar que el controlador pueda cortar la comunicación con la memoria en medio de una transferencia.
TS_VALID	Output	Señal que se activa cuando se ha escrito en memoria al menos un dato.
TS_FIFO_DOUT[127:0]	Input	Entrada de datos de tiempos entre paquetes.
TS_FIFO_RD_EN	Output	Señal de lectura de la fifo de entrada.
TS_FIFO_EMPTY	Input	Señal que indica si hay algún dato disponible para leer en la fifo de entrada.
LAST_ADDRES[31:0]	Output	Registro que almacena la dirección máxima utilizada.
MI32_DWR[127:0]	Output	Datos enviados a la memoria DDRII.
MI32_ADDR[31:0]	Output	Dirección de escritura DDRII.
MI32_WR	Output	Señal de control de escritura.
MI32_BE[15:0]	Output	Máscara de bytes que se quieren escribir.
MI32_ARDY	Input	Señal de recepción de dirección. Indica que la petición de escritura de un dato se ha procesado y se puede mandar el siguiente.

Cuadro 4.15: Puertos del módulo tsWriter

El módulo *tsWriter* (figura 4.18) es el responsable de recibir los tiempos entre paquetes de la fifo de entrada, y escribirlos en memoria en el espacio de direcciones reservado para tiempos entre paquetes. Cada vez que se escribe una dirección, se actualiza la señal LAST_ADDRES (ver tabla 4.15) para que el módulo de lectura de la DDR sepa qué direcciones contienen datos válidos.

Al recibir los datos de la fifo de entrada, hay que tener en cuenta que no se controla que la cantidad de tiempos entre paquetes recibidos sea múltiplo de 4. Esto es importante ya que la memoria tiene que recibir los datos de 256 en 256 bits y los tiempos entre paquetes son de 64 bits. Si se recibiese un número de ITPs que no fuese múltiplo de 4, el módulo se quedaría esperando los datos restantes hasta completar los 256 bits que espera la memoria, dejando además, la señal de bloqueo (LOCKED) activada e impidiendo así que se pueda pasar a otros modos de funcionamiento. En este caso, sería necesario un reset de sistema para volver

a un estado conocido. Este requerimiento se controla en el programa de gestión, que sólo envía los datos cuando tiene 4 ITPs almacenados.

mem_if_arbitrator

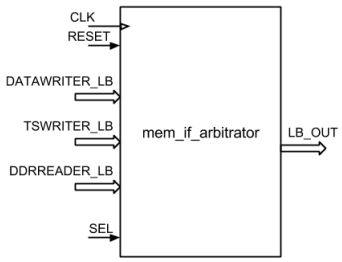

Figura 4.19: Diagrama mem_if_arbitrator

Puertos del módulo mem_if_arbitrator

Nombre de señal	Dirección	Descripción
CLK	Input	Reloj de 125Mhz
RESET	Input	Reset asíncrono de sistema.
DATAWRITER_LB	Input	Entrada de localBus procedente del módulo dataWriter
TSWRITER_LB	Output	Entrada de localBus procedente del módulo tsWriter
DDRREADER_LB	Output	Entrada de localBus procedente del módulo ddrReader
SEL[1:0]	Output	Entrada de control del árbitro. Selecciona la entrada que se conecta con la salida de acuerdo a los siguientes valores: “00”: ddrReader “01”: dataWriter “10”: tsWriter “XX”: ddrReader (opción por defecto)
LB_OUT	Input	Salida de localbus conectada con el traductor a MIG

Cuadro 4.17: Puertos del módulo mem_if_arbitrator

El módulo *mem_if_arbitrator* (figura 4.19) es responsable de arbitrar la conexión de los 3 módulos diferentes que acceden a la memoria DDR. Para ello, multiplexa cada una de las señales del protocolo local bus de la entrada seleccionada con la señal SEL a la salida LB_OUT (ver tabla 4.17). Por defecto la entrada que se selecciona es la del módulo ddrReader para que en caso de un funcionamiento erróneo garantice que no se corrompan datos de la memoria, ya que este módulo mantiene la señal de escritura a DDR siempre desactivada.

4.2. Implementación software

Para proporcionar al usuario una interfaz fácil de utilizar que le abstraiga de los registros internos del módulo de replicación, se ha desarrollado una aplicación software sencilla que permite al usuario realizar los pasos necesarios para replicar una traza de red .

La aplicación presenta al usuario las siguientes opciones:

1. **Load data**
   La opción 1 permite al usuario seleccionar un fichero pcap para cargar los datos de paquetes en la memoria ram. Para cargar los datos se pone el módulo de replicación en modo dataWrite escribiendo un 4 en el registro de modo, para posteriormente utilizar la herramienta tcpreplay para enviar los datos de paquetes a la interfaz de red "sze0:0", que es la que utiliza el dma0 para recibir los paquetes que va a transmitir al módulo principal de la aplicación. Esto es posible gracias a las librerías *libsze* implementadas por INVEATech, que se integran con la aplicación tcpreplay permitiendo el envío de manera transparente a través de las interfaces virtuales de la plataforma NetCOPE. Una vez finalizada la carga, se vuelve al estado de reposo escribiendo un 7 en el registro de modo.

2. **Load interpackets**
   La opción 2 permite al usuario seleccionar un fichero pcap para cargar los datos de tiempos entre paquetes en la memoria ram. Para cargar los datos se pone el módulo de replicación en modo tsWrite escribiendo un 5 en el registro de modo. Una vez el replicador se encuentra en el modo adecuado, escucha en la dirección 30000 del LocalBus esperando el envío de tiempos entre paquetes.Para extraer los tiempos entre paquetes del fichero pcap, se ha utilizado la función "pcap_loop()" de la librería libpcap. Los tiempos entre paquetes se almacenan en una estructura durante la lectura hasta que se han acumulado 4, momento en el cual se realiza el envío de los paquetes almacenados a través del LocalBus. Esto se hace porque, como ya se ha explicado anteriormente, el módulo de replicación no controla que los datos de tiempos entre paquetes sean múltiplos de 4, y al utilizar ráfagas en la escritura en memoria, es necesario verificarlo en el software. Si el número de paquetes del fichero PCAP no es múltiplo de 4, los últimos tiempos entre paquetes serán descartados ya que nunca se podrán acumular los 4 tiempos necesarios para realizar el envío.

3. **Set interpacket offset (also used for uniform mode)**
   La opción 3 permite al usuario establecer el valor del registro ITP_OFFSET_REG. Este registro contiene un valor numérico en nanosegundos que se utiliza co-

mo tiempo entre paquetes en el modo de envío uniforme, y como offset del tiempo entre paquetes real en el modo réplica. En el modo de envío uniforme, si el valor es 0 (valor por defecto), se envían paquetes a la máxima tasa posible.

4. **One time replication**
   La opción 4 inicia la replicación de los datos cargados con las opciones 1 y 2. La implementación de esta función se ha realizado de la siguiente manera. En primer lugar se verifica que el replicador se encuentra en estado de reposo (que el registro de modo contiene un 7), y si es así se inicia el envío escribiendo un 0 en el registro. Finalmente se vuelve a escribir un 7 para que tras finalizar el envío el replicador vuelva al estado inicial.

5. **Loop replication**
   La opción 5 inicia la replicación de los datos cargados con las opciones 1 y 2. En esta opción, en primer lugar se verifica que nos encontramos en el estado inicial comprobando que el registro de modo contiene un 7. Luego se inicia el envío en modo bucle escribiendo un 2 en el registro de modo. El replicador se mantendrá en este modo mientras no se escriba un 7 en el registro, lo cual ocurrirá al ejecutarse la opción 8 del programa.

6. **One time uniform mode**
   La opción 6 inicia la replicación de los datos cargados con la opción 1. La implementación de esta opción se realiza igual que la opción 4 pero escribiendo en el registro un 1.

7. **Loop uniform mode**
   La opción 7 inicia la replicación de los datos cargados con la opción 1. La implementación de esta opción se realiza igual que la opción 4 pero escribiendo en el registro un 3.

8. **Stop Loop**
   La opción 8 detiene el envío cuando se ha seleccionado previamente la opción 5 u 8. Para ello escribe en el registro de modo un 7, que indica al replicador que tras finalizar el siguiente envío vuelva al estado de reposo.

9. **Exit**
   La opción 9 finaliza el programa.

10. **Read regs(debug)**
    La opción 10 lee los registros de estado del módulo de replicación y los muestra al usuario..

Capítulo 5

Tests y resultados

A la hora de plantear la metodología de pruebas que se va a utilizar durante el desarrollo del proyecto, se ha decidido que se va a establecer una serie de hitos o puntos de control durante el desarrollo en los cuales se van a realizar simulaciones y pruebas que garanticen en cierta medida que los módulos desarrollados hasta ese momento están correctamente implementados y que son funcionales independientemente del resto de módulos que se van a implementar posteriormente.

Se han establecido los siguientes puntos de control:

1. Tras la implementación de la interfaz de registros.
2. Tras la implementación del módulo de envío *packetSender* que interactúa con la interfaz de red.
3. Tras la instanciación del controlador de memoria DDRII.
4. Tras la implementación del módulo *memIf* que controla toda la interacción con la memoria.
5. Tras la integración de los módulos.
6. Pruebas de rendimiento.

5.1. Simulaciones

Simulación interfaz de registros

Para simular el comportamiento de la interfaz de registros y del correcto uso del protocolo local bus que utiliza para comunicarse con el usuario, se ha utilizado el testbench proporcionado por INVEA-Tech sin apenas modificaciones. En este

testbench se utilizan las funciones de escritura a localBus e internalBus para simular lo que sería una escritura o lectura del usuario con el comando "cbus". Las únicas modificaciones que se han realizado son la eliminación de los procesos que comprueban el envío por DMA y algunos ajustes en las direcciones del bus que se probaban para adecuarlas a las direcciones relevantes para nuestro diseño. Como el diseño de la interfaz de registros se pensó para que permitiese tanto el acceso a registros como un forwarding de las direcciones superiores del bus, el testbench también verifica que esta redirección funcione correctamente.

Simulación packetSender

Para la verificación del módulo *packetSender* no se ha desarrollado ningún testbench adicional ya que para comprobar la interacción con la interfaz de red haría falta un modelo de simulación de la interfaz que no viene proporcionado por la plataforma de desarrollo. Lo que si se ha hecho ha sido un modulo de generación de paquetes sintéticos controlado por registros y el testbench correspondiente para verificar su funcionamiento antes de implementarlo en la tarjeta.

Simulación MIG

Se ha desarrollado otro testbench para verificar la calibración de la memoria. Éste, es similar al testbench global proporcionado con la plataforma NetCOPE que simula toda la aplicación, pero incluye la instanciación del modelo de la memoria DDRII proporcionado por coreGen al generar el controlador de memoria DDRII MIG. Con este testbench se ha verificado que el DCM que genera los relojes y los resets necesarios para controlar la memoria está correctamente configurado.

Simulación memIf

Sobre este testbench se han hecho posteriores modificaciones para incluir un proceso que generase escrituras y lecturas en la memoria para poder verificar que la lógica del direccionamiento era la correcta y que los datos que se leían eran los mismos que los que se habían escrito. De esta manera, se resolvieron algunos problemas relacionados con el almacenamiento de datos big endian o el tratamiento de los ciclos en los que se escriben ceros debido a la escritura en ráfagas.

Simulación completa

Finalmente, se ha modificado el testbench de simulación del MIG para que simulase unas señales de estímulos similares a las generadas por los flujos más habituales

del programa de control, así como los datos de trazas que posteriormente se han utilizado en las pruebas reales. En la figura5.1 se muestra un ejemplo de funcionamiento de la simulación.

Otras simulaciones

Además de las simulaciones realizadas en los hitos definidos al inicio, durante el desarrollo de los distintos módulos que componen el aplicativo se han realizado simulaciones parciales para comprobar de una manera sencilla el funcionamiento de los distintos módulos desarrollados.

Estas simulaciones, pese a no ser tan exhaustivas como la simulación global, son esenciales para depurar pequeños errores de funcionamiento durante desarrollo de los módulos. Al evaluar sólo el comportamiento de los módulos de manera aislada, y no comprobar todas las posibles combinaciones de entradas y salidas, no garantizan que el funcionamiento sea el correcto al interactuar con el resto de módulos. Sin embargo, establecen un requisito mínimo que cumplir antes de realizar pruebas más costosas. Además, permiten ver más claramente el funcionamiento teórico de los módulos a la hora de depurar y localizar errores detectados en las pruebas y simulaciones globales. Con este fin se ha desarrollado un testbench por cada módulo desarrollado.

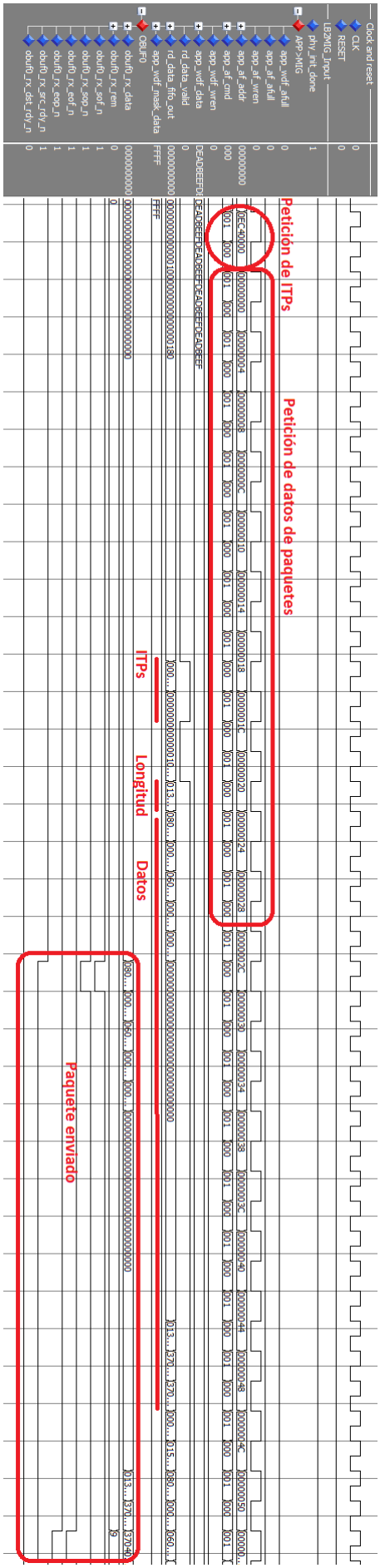

Figura 5.1: Ejemplo de simulación: Lectura en DDR y envío a interfaz de red(FrameLink)

5.2. Pruebas reales

A pesar de todas estas simulaciones mencionadas anteriormente, a la hora de probar el funcionamiento real en los distintos puntos de control durante el desarrollo del proyecto, siempre han aparecido errores que no eran detectables durante la simulación, provocados principalmente por el tiempo de propagación de las señales de reset y las definiciones y constrains de tiempo del fichero UCF.

Pruebas interfaz de registros

Para verificar el funcionamiento de la interfaz de registros se ha desarrollado un script en bash que, utilizando el comando "csbus", escribe datos en los registros y posteriormente los lee verificando que el valor leído es el mismo que el que se escribió.

Pruebas packetSender

A la hora de realizar las pruebas tras implementar el módulo *packetSender* ha habido que desarrollar un módulo capaz de generar datos sintéticos, ya que no existía ninguna funcionalidad de carga ni almacenamiento de datos en esta fase del proyecto. Este generador de datos sintéticos se controla mediante 2 registros: uno para parar o iniciar la generación de datos, y otro para introducir los datos que se quiere que compongan los paquetes que se van a enviar. De esta manera se han detectado errores en la selección de interfaz de red y en el cálculo del valor de la señal DREM (ver en el apartado de implementación del módulo *packetSender*).

Pruebas MIG

Para realizar las pruebas de calibración de la memoria se empezaron a utilizar registros de estado para monitorizar las señales relevantes, entre ellas, la señal phy_init_done que indica si la memoria esta calibrada o no. Conseguir que la memoria calibrase es lo que más tiempo ha supuesto por la dificultad que añade a la hora de depurar errores, el que sea un módulo ngc y no se pueda acceder al código. El principal problema a la hora de instanciar el controlador de memoria MIG para conseguir que calibrase la memoria ha sido que la plataforma NetCOPE no tenía contemplada una señal de dataMask para la memoria DDRII y la tarjeta por defecto hacía un pullDown de las salidas de la FPGA conectadas a esa señal en la tarjeta de memoria. Para conseguir calibrar la memoria hubo que crear esas salidas en el fichero UCF y activarlas en el código. Tras realizar ese cambio se

consiguió calibrar la memoria, pero eso no aseguraba que el acceso a las direcciones de la memoria fuese a funcionar.

Pruebas memIf

Para verificar el funcionamiento de la memoria DDRII, del controlador MIG y del módulo *memIf* encargado de interaccionar con él, se ha desarrollado un módulo independiente *lb2migddr_gen_test* que verifica su funcionamiento. Este módulo escribe en todas direcciones de memoria una secuencia numérica creciente, para posteriormente, leer todas las direcciones y verificar que la secuencia leída es la misma que se almacenó. El módulo se controla mediante un registro de usuario, y escribe en otro el resultado de la comprobación.

Tras realizar varias pruebas se comprobó que varios caminos de datos no cumplían las constrains de tiempo y que otros caminos que si las cumplían presentaban datos erróneos. Esto sólo se podía explicar con un error en la definición de las constrains en el fichero UCF.

Para corregir el error se adaptaron e insertaron las constrains de tiempo generadas por coreGen junto con el modelo de simulación de la memoria.

Pruebas completas

Finalmente, se han realizado pruebas reales del diseño completo. Durante estas pruebas no se ha utilizado ningún analizador de tráfico capaz de capturar tráfico a 10Gbps por lo que los resultados no están orientados a medir el ancho de banda generado, sino a comprobar que los datos se envían correctamente. Por esta razón, el fichero de trazas utilizado no necesita contener muchos paquetes, sino que tenga pocos para poder comprobar rápidamente que no hay ningún bit erróneo.

También se ha intentado verificar que el flujo de datos dentro del diseño funciona correctamente y que los tiempos entre paquetes capturados son similares a los de la traza que se intenta replicar. Por ello, es importante que la traza de pruebas tenga tiempos entre paquetes reales y una tasa binaria baja para que no se pierdan paquetes ni se alteren los timestamps recibidos al capturar con tcpdump, ya que éste no soporta tasas elevadas.

Por estas razones, se ha utilizado una traza de 4 paquetes que contiene una petición dhcp y la respuesta del servidor, con unos tiempos entre paquetes del orden de los microsegundos.

Para poder depurar el diseño se ha implementado un módulo de monitorización, que se sitúa entre la interfaz de la memoria y la fifo de envío. El objetivo de este

módulo es diferenciar si un error está producido en la interfaz de la memoria o en el módulo de envío. Para ello, verifica la estructura de los datos y si detecta algún error modifica un registro de estado, indicando así que el error se ha producido antes de que los datos llegasen al módulo de envío. Este módulo ha sido absolutamente necesario a la hora de corregir los distintos errores que se han producido durante estas pruebas.

Al realizar las pruebas se detectaron principalmente 4 errores:

- Se observó que había bits erróneos en los datos enviados. Cuando uno de estos errores se producía en un bit del dato de longitud o de tiempo entre paquete, se desincronizaban los datos y las cabeceras dejando las máquinas de estado de los módulos en estados no iniciales.
  Este error puede producirse por diversos motivos: problemas de metaestabilidad, o errores en la definición de las constrains de tiempo. Como no se realiza ningún cambio de dominio de reloj, se han restringido más las definiciones de constrains aumentando el input_jitter y se ha aumentado la segmentación del código introduciendo una etapa de registro entre la interfaz de memoria y el controlador MIG.

- En el modo de envío cíclico, al final de una iteración de envío se envían ceros de más, que el módulo de envío detecta como la longitud del siguiente paquete. Cuando detecta una longitud de cero, el módulo lo trata como un error irrecuperable y detiene el envío.
  Este problema se daba debido a que se utilizaban resets asíncronos para poner todo en el estado inicial cuando se terminaba una iteración, para después empezar de nuevo a enviar desde el principio en la siguiente iteración. Al ser asíncronos, las puertas lógicas que lo controlaban introducían retardos no controlados por las constrains de tiempo. Este problema se solucionó añadiendo a los módulos pertinentes un reset síncrono dedicado a este fin independiente del reset de sistema asíncrono.

- Se detectó un error en la implementación del protocolo de comunicación FrameLink con el controlador de red que provocaba la saturación de la interfaz haciendo que nunca estuviese disponible para enviar datos.
  El error se corrigió volviendo a implementar la máquina de estados desde cero.

- Al utilizar paquetes mayores de 64 bytes se hacen más peticiones de datos a la memoria provocando el llenado de la fifo de envío, ya que se llena más rápido de lo que se envían datos a la interfaz de red. Como se ha explicado antes, si esta fifo se llena se pierden datos.
  Este error se producía debido a que algún factor no se consideró en los

cálculos del margen que había que dejar en la fifo de envío para que no se llenase. Para solucionarlo se aumentó este margen hasta que dejó de desbordarse la fifo.

5.3. Pruebas de rendimiento

Para evaluar el rendimiento del replicador de tráfico se han realizado pruebas de envío con varios ficheros de trazas distintos según el tamaño de los paquetes y su procedencia. Los paquetes enviados son capturados por un analizador de tráfico que genera un fichero de trazas de salida con los paquetes capturados. Finalmente, se comparan el fichero de trazas enviado y el generado por el capturador para comprobar que el replicador ha enviado correctamente la traza.

En los siguientes apartados se van a presentar las pruebas realizadas y los resultados obtenidos para cada una de ellas.

5.3.1. Banco de pruebas

Se han realizado, en primer lugar, pruebas con dos ficheros de trazas distintos compuestos por paquetes de 1500 y 64 bytes obtenidos a partir de un generador de trazas. El objetivo de estas pruebas es medir la máxima tasa binaria alcanzable para los distintos tamaños de paquete utilizados, para lo que se ha enviado con tiempo entre paquetes constante de 0 ns. La herramienta utilizada para generar los ficheros de trazas es el programa Genpcap y ha sido desarrollado por el grupo HPCN de la Universidad Autónoma de Madrid. Éste parte de un fichero con un único paquete y genera una traza constituida por el mismo paquete repetido tantas veces como sea necesario para alcanzar el tamaño de traza deseado. A continuación, se muestran los parámetros utilizados para la generación de los ficheros de trazas con esta herramienta.

```
$ ./genpcap -s 1600000000 -i 120000 -o sintetico_1500.pcap paquete_1500.pcap
$ ./genpcap -s 1600000000 -i 12000 -o sintetico_64.pcap paquete_64.pcap
```

Las características de los ficheros pcap generados se han obtenido utilizando la herramienta capinfos y se muestran a continuación:

Traza de paquetes de 1500 bytes:

```
File name:           sintetico_1500.pcap
Number of packets:   1055409
File size:           1600000068 bytes
Data size:           1583113500 bytes
Capture duration:    1 seconds
Data bit rate:       8724027778.69 bits/sec
Average packet size: 1500.00 bytes
Average packet rate: 727002.31 packets/sec
```

Figura 5.2: Traza de 1500 bytes

Traza de paquetes de 64 bytes:

```
File name:           sintetico_64.pcap
Number of packets:   20000000
File size:           1600000024 bytes
Data size:           1280000000 bytes
Capture duration:    1 seconds
Data bit rate:       8619585967.73 bits/sec
Average packet size: 64.00 bytes
Average packet rate: 16835128.84 packets/sec
```

Figura 5.3: Traza de 64 bytes

Nota: El parámetro *Data size* de los ficheros de trazas no debe superar en ningún caso el tamaño máximo que se puede almacenar en el espacio de direcciones de datos de la memoria DDRII (1651 MB).

Con el objetivo de medir la precisión alcanzada para los tiempos entre paquetes se ha utilizado un fichero de trazas capturado en un entorno real. Dado que este fichero presenta una distribución de tamaños de paquetes real, permitirá también analizar la tasa binaria máxima que se podría conseguir en un entorno real. A continuación se muestran las características del fichero pcap empleado.

Traza de tráfico real:

```
File name:           REAL2_tcpdump.pcap
Number of packets:   2890280
File size:           1700000028 bytes
Data size:           1653755524 bytes
Capture duration:    2175 seconds
Data bit rate:       6084108.08 bits/sec
Average packet size: 572.18 bytes
Average packet rate: 1329.15 packets/sec
```

Figura 5.4: Traza de tráfico real

5.3.2. Resultados

El analizador de tráfico empleado para capturar los paquetes enviados por el replicador ha sido desarrollado también por el grupo HPCN[29]. Esta herramienta cuenta con un driver de tarjeta de red mejorado que realiza las funciones de *sniffer* y se comunica con las librerías M^3Omon que proporcionan las herramientas necesarias para el volcado de los paquetes a un fichero de trazas. Utilizando esta herramienta y un sistema de almacenamiento en raid 0 capaz de almacenar los paquetes a velocidades de 10 Gbps, se ha conseguido medir las tasas máximas que se pueden alcanzar con el replicador de tráfico desarrollado.

Al igual que para los ficheros de trazas de test, se han obtenido las características de los ficheros generados por el analizador de tráfico utilizando el programa capinfo. A continuación, se van a presentar los resultados obtenidos y se compararán con las características de los ficheros pcap utilizados durante el envío por el generador para evaluar el rendimiento y la precisión del mismo.

Traza de paquetes de 1500 bytes enviada:

```
File name:           sintetico_1500_constant.pcap
Number of packets:   1055409
File size:           1600000068 bytes
Data size:           1583113500 bytes
Capture duration:    1 seconds
Data bit rate:       9840857103.59 bits/sec
Average packet size: 1500.00 bytes
Average packet rate: 820071.43 packets/sec
```

Figura 5.5: Captura de la traza de 1500 bytes

En primer lugar, se han realizado las mediciones del mejor caso posible, que se da para trazas de paquetes de tamaño máximo, ya que en ese caso la cantidad de

cabeceras que hay que procesar es mínima para una misma cantidad de datos. En las características del fichero capturado observamos que el número de paquetes recibidos y el tamaño de éstos, es el mismo que los de la traza enviada, por lo que no se ha producido pérdida de paquetes. También podemos observar que la tasa binaria conseguida es de 9.84 Gbps que es muy próximo a 10 Gbps, que es el máximo permitido por la interfaz de red.

Traza de paquetes de 64 bytes enviada:

```
File name:           sintetico_64_constant.pcap
Number of packets:   20000000
File size:           1600000024 bytes
Data size:           1280000000 bytes
Capture duration:    1 seconds
Data bit rate:       7271013466.80 bits/sec
Average packet size: 64.00 bytes
Average packet rate: 14201198.18 packets/sec
```

Figura 5.6: Captura de la traza de 64 bytes

La siguiente medida que se ha tomado ha sido la del peor caso, que se da para trazas de 64 bytes, ya que en ese caso la proporción de cabeceras con respecto a la cantidad de datos es máxima. En las características del fichero capturado podemos observar que tampoco se han perdido paquetes, pero observando la tasa binaria obtenida podemos ver que no se acerca al objetivo de los 10 Gbps. Esto se debe a que para el estándar 10GbE 802.3ae se ha definido un CRC de 4 bytes, un preámbulo de 8 bytes y un espacio entre frames de 12 bytes. Teniendo en cuenta estos datos se puede calcular la tasa máxima alcanzable en frames de 64 bytes por segundo de la siguiente manera[30]:

$$\frac{10000000000}{(64+4(CRC)+8(Preamble)+12(InterFrameGap))*8} = 14204545\,packets/s = 7{,}2727\,bits/s$$

Traza de tráfico real enviada:

```
File name:           real_constant.pcap
Number of packets:   2890280
File size:           1700002980 bytes
Data size:           1653758476 bytes
Capture duration:    1 seconds
Data bit rate:       9272694089.15 bits/sec
Average packet size: 572.18 bytes
Average packet rate: 2025740.36 packets/sec
```

Figura 5.7: Traza real enviada a la máxima tasa

En el envío a máxima velocidad podemos observar que se obtienen tasas próximas a los 10 Gbps pese a que el tamaño medio de paquete es bastante bajo. Como en casos anteriores se puede ver también que no se ha producido pérdida de paquetes.

5.4. Pruebas funcionales

5.4.1. Replicación de paquetes

A la hora de realizar pruebas sobre un replicador de tráfico, es indispensable verificar que los paquetes que envía son realmente los de la traza original. Para verificar que los datos enviados son los correctos, se ha comparado con la herramienta Wireshark la traza enviada con la original. En las figuras 5.8 y 5.9 se puede ver que los paquetes originales y los enviados son idénticos excepto por los tiempos entre paquetes, ya que en la traza enviada, éstos han sido generados por la herramienta utilizada durante la captura.

```
 1 0.000000   2001:0:5ef5:79fd:42 2001:0:5ef5:79fd:18 IPv6      98 IPv6 no next header
 2 0.000009   2001:0:5ef5:79fd:42 2001:0:5ef5:79fd:18 IPv6      98 IPv6 no next header
 3 0.000012   2001:0:5ef5:79fd:42 2001:0:9d38:6ab8:85 IPv6      98 IPv6 no next header
 4 0.000014   2001:0:5ef5:79fd:42 2001:0:9d38:6ab8:85 IPv6      98 IPv6 no next header
 5 0.084757   201.95.175.76      192.168.229.67     UDP      114 Source port: 54357  Destination port: 63378
 6 0.084766   201.95.175.76      192.168.229.67     UDP      114 Source port: 54357  Destination port: 63378
 7 0.172799   192.168.229.67     150.244.56.136     TCP       66 64071 > 24800 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
 8 0.172807   192.168.229.67     150.244.56.136     TCP       66 64071 > 24800 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
 9 0.173550   150.244.56.136     192.168.229.67     TCP       60 24800 > 64071 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
10 0.173558   150.244.56.136     192.168.229.67     TCP       60 24800 > 64071 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
11 0.239439   194.28.91.167      192.168.229.67     UDP      110 Source port: 59909  Destination port: 63378
12 0.239447   194.28.91.167      192.168.229.67     UDP      110 Source port: 59909  Destination port: 63378
13 0.318403   2001:0:5ef5:73b8:30 2001:0:5ef5:79fd:42 IPv6      98 IPv6 no next header
14 0.318411   2001:0:5ef5:73b8:30 2001:0:5ef5:79fd:42 IPv6      98 IPv6 no next header
15 0.318754   192.168.229.67     192.168.0.101      UDP       94 Source port: 63378  Destination port: 61109
16 0.318762   192.168.229.67     192.168.0.101      UDP       94 Source port: 63378  Destination port: 61109
```

Figura 5.8: Paquetes que se van a replicar

```
 1 0.000000   2001:0:5ef5:79fd:42 2001:0:5ef5:79fd:18 IPv6        98 IPv6 no next header
 2 0.000008   2001:0:5ef5:79fd:42 2001:0:5ef5:79fd:18 IPv6        98 IPv6 no next header
 3 0.000008   2001:0:5ef5:79fd:42 2001:0:9d38:6ab8:85 IPv6        98 IPv6 no next header
 4 0.000008   2001:0:5ef5:79fd:42 2001:0:9d38:6ab8:85 IPv6        98 IPv6 no next header
 5 0.084759   201.95.175.76       192.168.229.67      UDP        114 Source port: 54357  Destination port: 63378
 6 0.084768   201.95.175.76       192.168.229.67      UDP        114 Source port: 54357  Destination port: 63378
 7 0.172817   192.168.229.67      150.244.56.136      TCP         66 64071 > 24800 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
 8 0.172818   192.168.229.67      150.244.56.136      TCP         66 64071 > 24800 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
 9 0.173561   150.244.56.136      192.168.229.67      TCP         60 24800 > 64071 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
10 0.173569   150.244.56.136      192.168.229.67      TCP         60 24800 > 64071 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
11 0.239464   194.28.91.167       192.168.229.67      UDP        110 Source port: 59909  Destination port: 63378
12 0.239465   194.28.91.167       192.168.229.67      UDP        110 Source port: 59909  Destination port: 63378
13 0.318430   2001:0:5ef5:73b8:30 2001:0:5ef5:79fd:42 IPv6        98 IPv6 no next header
14 0.318437   2001:0:5ef5:73b8:30 2001:0:5ef5:79fd:42 IPv6        98 IPv6 no next header
15 0.318780   192.168.229.67      192.168.0.101       UDP         94 Source port: 63378  Destination port: 61109
16 0.318788   192.168.229.67      192.168.0.101       UDP         94 Source port: 63378  Destination port: 61109
```

Figura 5.9: Paquetes enviados

5.4.2. Replicación de tiempos entre paquetes

Para realizar esta prueba se ha enviado una traza de tráfico real en modo replicación, en el cual se intenta replicar de la manera más fiel posible el modelo de tiempos entre paquetes de la traza original. Las características de la traza enviada se pueden ver en la figura 5.10.

```
File name:           real_realtime.pcap
Number of packets:   2890280
File size:           1700002980 bytes
Data size:           1653758476 bytes
Capture duration:    2175 seconds
Data bit rate:       6083642.63 bits/sec
Average packet size: 572.18 bytes
Average packet rate: 1329.05 packets/sec
```

Figura 5.10: Traza real enviada con el modelo de tiempos original

En las medidas de la traza enviada en modo replicación queremos intentar evaluar la fidelidad del modelo de tiempos capturado con respecto al original y la precisión en los tiempos entre paquetes que puede ofrecer el diseño.

Analizando las trazas original y replicada, se han obtenido las siguientes gráficas (figura 5.11), que muestran la tasa de envío en $n^{o}\,de\,paquetes/0{,}1s$ de ambas.

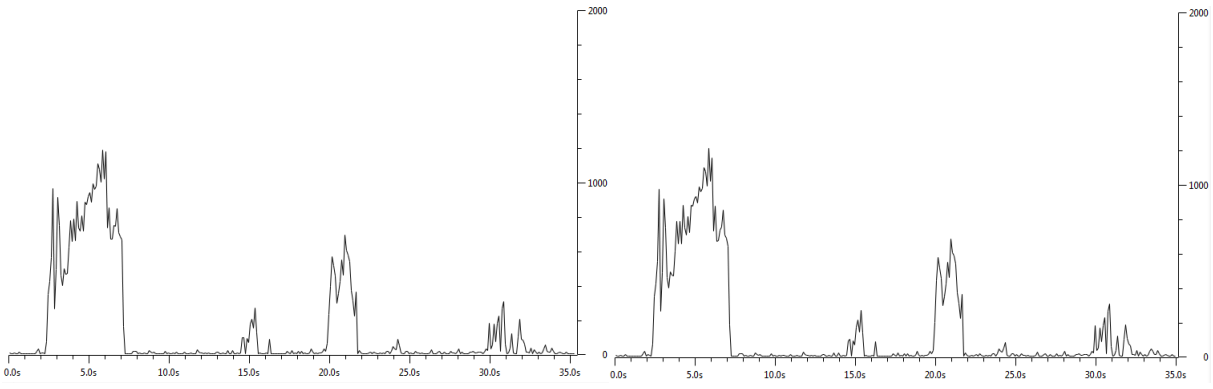

Figura 5.11: Tasa binaria original (izda) vs tasa enviada (dcha)

Para comparar mejor los resultados obtenidos, se han procesado los datos con Matlab y se han superpuesto las gráficas anteriores. Como se puede ver en la figura 5.12, no se puede apreciar apenas la diferencia, entre las tasas binarias de la traza original y las enviadas por el replicador de tráfico.

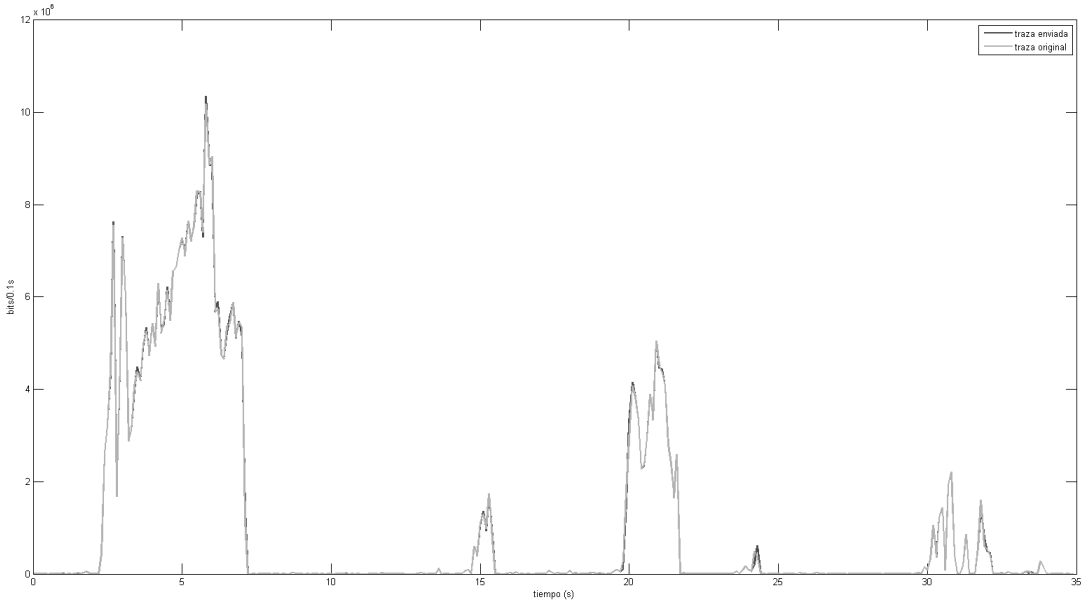

Figura 5.12: Comparativa tasas binarias

Comparando las tasas binarias de la traza original (figura 5.4) con la capturada (figura 5.10), observamos que la tasa binaria media recibida es 466 bits/s más baja que la original. Esto, puesto en términos de precisión en los tiempos entre paquetes, quiere decir que en media se ha incrementado cada tiempo entre paquete en 0.2 ns. Este resultado es correcto de acuerdo a los objetivos predefinidos, y concuerda con lo esperado, ya que de la manera en la que está implementado el contador siempre va a tender a incrementar los tiempos entre paquetes con respecto a el modelo de tiempos original. A continuación se explica el cálculo realizado para obtener el incremento medio mencionado:

Se obtienen 466 bits/s menos de tasa binaria que en la traza replicada. Teniendo en cuenta que se envían 128 bits por ciclo el dato anterior equivale a un retraso de $3{,}64 ciclos/s$. Con un periodo de reloj de 8 ns eso equivale a $29{,}125 ns/s$ de incremento total en los tiempos entre paquetes por segundo. Como se están enviando 1329 paquetes por segundo con 1329 tiempos entre paquetes el incremento medio por tiempo entre paquete es de $29{,}125/1329 = 0{,}2 ns/itp$

Como método de verificación se ha utilizado también un proceso dentro del módulo de envío que verifica, en el momento de empezar a enviar un nuevo paquete, que

el contador de tiempos interno del módulo no difiera en más de 8 ns del valor de tiempo entre paquetes del paquete que se va a enviar. Si en algún momento el módulo de envío no es capaz de cumplir los tiempos de envío, este proceso modifica un registro de estado informando del error al usuario. A parte de esta verificación interna, no ha sido posible realizar verificaciones más exhaustivas ya que no se disponía de un analizador de tráfico capaz de proporcionar suficiente precisión en los tiempos entre paquetes durante la captura.

5.5. Consumo de recursos

Los recursos de la FPGA utilizados por el replicador, como se puede ver en la tabla 5.2, son en su mayoría los utilizados por la propia plataforma de desarrollo.

Se puede apreciar un incremento considerable en el uso de IOBs, BUFG, IDELAYCTRLs y PLLs. Estos recursos se han utilizado principalmente durante la instanciación del controlador de memoria MIG, ya que éste necesita instanciar PLLs internos, y acceder a los pines de salida del socket de memoria DDR.

El aumento del uso de la memoria interna de la FPGA es debido a las distintas fifos instanciadas en el módulo de replicación y en el controlador de memoria. Debido a la latencia tan variable que introduce la memoria DDR, ha sido necesario sobredimensionar las fifos utilizadas en el diseño. Estas fifos se han instanciado utilizando principalmente BlockRAM y registros de desplazamiento. De este modo, se ha repartido de una manera más uniforme el uso de los recursos entre la lógica combinacional, los registros y la memoria interna de la FPGA.

Recursos de la FPGA utilizados

Slice Logic Utilization	Utilizados por la plataforma	Utilizados por el replicador y la plataforma
Slice Registers	23 %	28 %
Slice LUTs	28 %	32 %
Used as logic	24 %	28 %
Used as Memory	15 %	15 %
Occupied Slices	46 %	53 %
Bonded IOBs	76 %	89 %
BlockRAM/FIFO	25 %	33 %
Total Memory used	**24 %**	**33 %**

Slice Logic Utilization	Utilizados por la plataforma	Utilizados por el replicador y la plataforma
BUFG/BUFGCTRLs	37 %	53 %
IDELAYCTRLs	0 %	13 %
PLL_ADVs	50 %	66 %

Cuadro 5.2: Recursos de la FPGA utilizados

Los recursos totales utilizados permiten ampliamente que en un futuro se puedan aumentar las funcionalidades del diseño, y que pueda convivir con otros módulos que necesiten interactuar con el replicador.

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones

En este proyecto se ha realizado un replicador de tráfico capaz de generar entornos de pruebas para redes 10GbE.

En el análisis del estado del arte realizado, se ha observado que en el campo de la replicación de tráfico, pese a haber multitud de soluciones, es difícil encontrar una que sea capaz de proporcionar el rendimiento suficiente para ser utilizada en enlaces 10GbE a un precio asequible y que sea lo suficientemente flexible para ser utilizada en el campo de la investigación. Teniendo en cuenta los objetivos del proyecto, de las soluciones estudiadas, las que mejores resultados ofrecían eran las híbridas.

Por ello se ha implementado una solución híbrida de un sistema de replicación de tráfico. El sistema permite al usuario cargar un fichero PCAP en una memoria de la tarjeta COMBOv2 para luego enviarla respetando el modelo de tiempos entre paquetes original. La solución implementada incluye un software, que hace de interfaz con el usuario proporcionándole varias opciones de configuración y de envío, y un núcleo de aceleración hardware que realiza las tareas que requieren más rendimiento o precisión. El núcleo de aceleración hardware consiste en chips FPGA reprogramables, por lo que aportan más flexibilidad y menor coste que las soluciones puramente hardware, manteniendo las ventajas que éstas aportan. Sin embargo, esta solución en dos fases (una de carga de datos y otra de envío), limita el tamaño de la traza que se puede enviar, ya que ésta depende del tamaño de la memoria utilizada. En la plataforma utilizada, la capacidad máxima de memoria permitida es de 2 GB, lo cual equivale a alrededor de segundo y medio de tráfico a 10 Gbps.

El desarrollo del replicador se ha centrado sobre todo en la parte hardware. Ésta

se ha implementado como una serie de módulos independientes que se interconectan entre sí con protocolos estándar ampliamente conocidos, facilitando así su portabilidad a otras plataformas y sus futuras modificaciones. Cumple tres funciones principalmente: implementa una interfaz de registros para comunicarse con el programa software, un controlador que gestiona la escritura y la lectura en la memoria DDRII de la tarjeta, y un módulo de envío que controla la reconstrucción y el envío de la traza que se quiere replicar.

Mediante la interfaz de registros, se proporciona al usuario la opción de especificar de manera dinámica el tiempo entre paquetes que se quiere utilizar, o modificar el modelo de tiempos original de la traza. De esta manera, el diseño es capaz de generar tráfico a cualquier tasa binaria alcanzando hasta los 10 Gbps.

El módulo encargado del envío es capaz de replicar las trazas en el nivel de enlace, por lo que su funcionamiento es independiente de los protocolos que encapsulen las tramas enviadas. Además, como se interactúa directamente con los controladores de las interfaces de red y el envío está controlado por un reloj de 125Mhz, se obtienen precisiones de 8 ns en los tiempos entre paquetes.

Durante las pruebas y medidas realizadas, se ha podido verificar que el diseño implementado cumple con todos los objetivos de precisión, funcionalidad y rendimiento que se habían establecido al inicio del proyecto.

6.2. Trabajo futuro

Durante el diseño del generador de tráfico se ha contemplado que en un futuro se pueda mejorar fácilmente añadiendo nuevos módulos o modificando los existentes. A continuación se proponen algunas ideas para mejorar el diseño y aportar nuevas funcionalidades.

- Añadir la posibilidad de capturar el tráfico recibido por la interfaz y almacenarlo en la memoria para, posteriormente, replicarlo o almacenarlo en disco en formato PCAP.
- Permitir que el usuario especifique la tasa binaria a la que desea enviar en vez de el tiempo entre paquetes.
- Añadir un módulo que recoja estadísticas del envío y que se de la opción de generar un informe para el usuario.
- Modificar el módulo que lee y escribe de la memoria para que aplique un algoritmo de compresión a los datos. De esta manera se aumentaría el tamaño máximo de la traza que se puede replicar.

- Disponer de una serie de distribuciones de tiempos típicas que el usuario pueda elegir para que sean utilizadas durante el envío en lugar de la almacenada en el fichero de trazas.
- Permitir la generación de datos sintéticos, ya sea cargando datos generados por la parte software o generando directamente el tráfico en el módulo de envío.

Bibliografía

[1] *Cisco Visual Networking Index: Forecast and Methodology*, 2013.

[2] ZTI, "http://www.zti-communications.com/lantrafficv2/."

[3] MGEN, "http://www.nrl.navy.mil/itd/ncs/products/mgen/."

[4] *BRUTE: A High Performance and Extensible Traffic Generator*, 2005.

[5] *D-ITG distributed Internet traffic generator*, 2004.

[6] *Comparative study of various traffic generator tools*, 2014.

[7] packETH, "http://packeth.sourceforge.net/packeth/home/."

[8] Ostinato, "https://code.google.com/p/ostinato/."

[9] *KUTE: a high performance Kernel-based UDP traffic engine*, 2005.

[10] W. chang Feng, A. Goel, A. Bezzaz, W. chi Feng, and J. Walpole, "Tcpivo: A high-performance packet replay engine," in *In Proceedings of the ACM SIGCOMM workshop on Models, methods*, pp. 57–64, 2003.

[11] S. S. J. Laine and R. Prior, "http://rude.sourceforge.net/."

[12] Ixia, "http://www.ixiacom.com/."

[13] SPIRENET, "http://www.spirent.com/."

[14] XENA, "www.xenanetworks.com/html/overview.html."

[15] J. Zhao, X. Zhang, and X. Wang, "An architecture design of gpu-accelerated vod streaming servers with network coding," in *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2010 6th International Conference on*, pp. 1–10, Oct 2010.

[16] *A Configurable FPGA-Based Traffic Generator for High-Performance Tests of Packet Processing Systems*, 2011.

[17] Xilinx, "http://www.xilinx.com/."

[18] *A Packet Generator on the NetFPGA Platform*, 2009.

[19] *NetFPGA–An Open Platform for Gigabit-Rate Network Switching and Routing*, 2007.

[20] netfpga, "http://netfpga.org/."

[21] *Caliper: Precise and Responsive Traffic Generator*, 2012.

[22] M. Labrecque, J. G. Steffan, G. Salmon, M. Ghobadi, and Y. Ganjali, "Netthreads: Programming netfpga with threaded software."

[23] OSNT, "https://github.com/netfpga/osnt-public/wiki."

[24] *NetCOPE: Platform for Rapid Development of Network Applications*, 2008.

[25] *Precise IPv4/IPv6 packet generator based on NetCOPE platform*, 2011.

[26] *Hacking NetCOPE to Run on NetFPGA-10G*, 2011.

[27] NetCOPE, "Fpga platform for rapid development of network applications. user guide. [online]. available: http://www.inveatech. com/data/netcope."

[28] micron, "Mt16htf25664hz-667, http://www.micron.com/parts/modules/ddr2-sdram/mt16htf25664hz-667."

[29] V. Moreno, P. M. Santiago del RÃo, J. Ramos, D. Muelas, J. L. GarcÃa-Dorado, F. J. Gomez-Arribas, and J. Aracil, "Multi-granular, multi-purpose and multi-gb/s monitoring on off-the-shelf systems," *International Journal of Network Management*, vol. 24, no. 4, pp. 221–234, 2014.

[30] V. Moreno, P. del Rio, J. Ramos, J. Garnica, and J. Garcia-Dorado, "Batch to the future: Analyzing timestamp accuracy of high-performance packet i/o engines," *Communications Letters, IEEE*, vol. 16, pp. 1888–1891, November 2012.

Parte I

Presupuesto

1) Ejecución Material:

- Compra de ordenador personal.....1000 €
- Compra de equipos hardware.....20000 €
- Material de oficina.....150 €
- Total ejecución material.....21150 €

2) Gastos generales

- 20 % sobre Ejecución Material.....4230 €

3) Honorarios Proyecto

- 960 horas a 30 € / hora.....28800 €

4) Beneficio Industrial

- 10 % sobre total.....5418 €

5) Presupuesto antes de Impuestos

- Subtotal Presupuesto.....59598 €

6) I.V.A. aplicable

- 21 % Subtotal Presupuesto.....12515 €

7) Total presupuesto

- Total Presupuesto.....72113 €

Madrid, Julio de 2014

El Ingeniero Jefe de Proyecto

Fdo.: Miguel Cubillo Llanes

Parte II

Pliego de condiciones

Este documento contiene las condiciones legales que guiarán la realización, en este proyecto, de un sistema de replicación de tráfico implementado en FPGA para redes 10GbeE. En lo que sigue, se supondrá que el proyecto ha sido encargado por una empresa cliente a una empresa consultora con la finalidad de realizar dicho sistema. Dicha empresa ha debido desarrollar una línea de investigación con objeto de elaborar el proyecto. Esta línea de investigación, junto con el posterior desarrollo del sistema está amparada por las condiciones particulares del siguiente pliego.

Supuesto que la utilización industrial de los métodos recogidos en el presente proyecto ha sido decidida por parte de la empresa cliente o de otras, la obra a realizar se regulará por las siguientes:

Condiciones generales

1. La modalidad de contratación será el concurso. La adjudicación se hará, por tanto, a la proposición más favorable sin atender exclusivamente al valor económico, dependiendo de las mayores garantías ofrecidas. La empresa que somete el proyecto a concurso se reserva el derecho a declararlo desierto.

2. El montaje y mecanización completa de los equipos que intervengan será realizado totalmente por la empresa licitadora.

3. En la oferta, se hará constar el precio total por el que se compromete a realizar la obra y el tanto por ciento de baja que supongo en los casos de rescisión.

4. La obra se realizará bajo la dirección técnica de un Ingeniero Superior de Telecomunicación, auxiliado por el número de Ingenieros Técnicos y Programadores que se estime preciso para el desarrollo de la misma.

5. Aparte del Ingeniero Director, el contratista tendrá derecho a contratar al resto del personal, pudiendo ceder esta prerrogativa a favor del Ingeniero Director, quien no estará obligado a aceptarla.

6. El contratista tiene derecho a sacar copias a su costa de los planos, pliego de condiciones y presupuestos. El Ingeniero autor del proyecto autorizará con su firma las copias solicitadas por el contratista después de confrontarlas.

7. Se abonará al contratista la obra que realmente ejecute con sujeción al proyecto que sirvió de base para la contratación, a las modificaciones autorizadas por la superioridad o a las órdenes que con arreglo a sus facultades le hayan comunicado por escrito al Ingeniero Director de obras siempre que dicha obra se haya ajustado a los preceptos de los pliegos de condiciones, con arreglo a los cuales, se harán

las modificaciones y la valoración de las diversas unidades sin que el importe total pueda exceder de los presupuestos aprobados. Por consiguiente, el número de unidades que se consignan en el proyecto o en el presupuesto, no podrá servirle de fundamento para entablar reclamaciones de ninguna clase, salvo en los casos de rescisión.

8. Tanto en las certificaciones de obras como en la liquidación final, se abonarán los trabajos realizados por el contratista a los precios de ejecución material que figuran en el presupuesto para cada unidad de la obra.

9. Si excepcionalmente se hubiera ejecutado algún trabajo que no se ajustase a las condiciones de la contrata pero que sin embargo es admisible a juicio del Ingeniero Director de obras, se dará conocimiento a la Dirección, proponiendo a la vez la rebaja de precios que el Ingeniero estime justa y si la Dirección resolviera aceptar la obra, quedará el contratista obligado a conformarse con la rebaja acordada.

10. Cuando se juzgue necesario emplear materiales o ejecutar obras que no figuren en el presupuesto de la contrata, se evaluará su importe a los precios asignados a otras obras o materiales análogos si los hubiere y cuando no, se discutirán entre el Ingeniero Director y el contratista, sometiéndolos a la aprobación de la Dirección. Los nuevos precios convenidos por uno u otro procedimiento, se sujetarán siempre al establecido en el punto anterior.

11. Cuando el contratista, con autorización del Ingeniero Director de obras, emplee materiales de calidad más elevada o de mayores dimensiones de lo estipulado en el proyecto, o sustituya una clase de fabricación por otra que tenga asignado mayor precio o ejecute con mayores dimensiones cualquier otra parte de las obras, o en general, introduzca en ellas cualquier modificación que sea beneficiosa a juicio del Ingeniero Director de obras, no tendrá derecho sin embargo, sino a lo que le correspondería si hubiera realizado la obra con estricta sujeción a lo proyectado y contratado.

12. Las cantidades calculadas para obras accesorias, aunque figuren por partida alzada en el presupuesto final (general), no serán abonadas sino a los precios de la contrata, según las condiciones de la misma y los proyectos particulares que para ellas se formen, o en su defecto, por lo que resulte de su medición final.

13. El contratista queda obligado a abonar al Ingeniero autor del proyecto y director de obras así como a los Ingenieros Técnicos, el importe de sus respectivos honorarios facultativos por formación del proyecto, dirección técnica y administración en su caso, con arreglo a las tarifas y honorarios vigentes.

14. Concluida la ejecución de la obra, será reconocida por el Ingeniero Director que a tal efecto designe la empresa.

15. La garantía definitiva será del 4 % del presupuesto y la provisional del 2 %.

16. La forma de pago será por certificaciones mensuales de la obra ejecutada, de acuerdo con los precios del presupuesto, deducida la baja si la hubiera.

17. La fecha de comienzo de las obras será a partir de los 15 días naturales del replanteo oficial de las mismas y la definitiva, al año de haber ejecutado la provisional, procediéndose si no existe reclamación alguna, a la reclamación de la fianza.

18. Si el contratista al efectuar el replanteo, observase algún error en el proyecto, deberá comunicarlo en el plazo de quince días al Ingeniero Director de obras, pues transcurrido ese plazo será responsable de la exactitud del proyecto.

19. El contratista está obligado a designar una persona responsable que se entenderá con el Ingeniero Director de obras, o con el delegado que éste designe, para todo relacionado con ella. Al ser el Ingeniero Director de obras el que interpreta el proyecto, el contratista deberá consultarle cualquier duda que surja en su realización.

20. Durante la realización de la obra, se girarán visitas de inspección por personal facultativo de la empresa cliente, para hacer las comprobaciones que se crean oportunas. Es obligación del contratista, la conservación de la obra ya ejecutada hasta la recepción de la misma, por lo que el deterioro parcial o total de ella, aunque sea por agentes atmosféricos u otras causas, deberá ser reparado o reconstruido por su cuenta.

21. El contratista, deberá realizar la obra en el plazo mencionado a partir de la fecha del contrato, incurriendo en multa, por retraso de la ejecución siempre que éste no sea debido a causas de fuerza mayor. A la terminación de la obra, se hará una recepción provisional previo reconocimiento y examen por la dirección técnica, el depositario de efectos, el interventor y el jefe de servicio o un representante, estampando su conformidad el contratista.

22. Hecha la recepción provisional, se certificará al contratista el resto de la obra, reservándose la administración el importe de los gastos de conservación de la misma hasta su recepción definitiva y la fianza durante el tiempo señalado como plazo de garantía. La recepción definitiva se hará en las mismas condiciones que la provisional, extendiéndose el acta correspondiente. El Director Técnico propondrá a la Junta Económica la devolución de la fianza al contratista de acuerdo con las condiciones económicas legales establecidas.

23. Las tarifas para la determinación de honorarios, reguladas por orden de la Presidencia del Gobierno el 19 de Octubre de 1961, se aplicarán sobre el denominado en la actualidad “Presupuesto de Ejecución de Contrata” y anteriormente

llamado "Presupuesto de Ejecución Material" que hoy designa otro concepto.

Condiciones particulares

La empresa consultora, que ha desarrollado el presente proyecto, lo entregará a la empresa cliente bajo las condiciones generales ya formuladas, debiendo añadirse las siguientes condiciones particulares:

1. La propiedad intelectual de los procesos descritos y analizados en el presente trabajo, pertenece por entero a la empresa consultora representada por el Ingeniero Director del Proyecto.

2. La empresa consultora se reserva el derecho a la utilización total o parcial de los resultados de la investigación realizada para desarrollar el siguiente proyecto, bien para su publicación o bien para su uso en trabajos o proyectos posteriores, para la misma empresa cliente o para otra.

3. Cualquier tipo de reproducción aparte de las reseñadas en las condiciones generales, bien sea para uso particular de la empresa cliente, o para cualquier otra aplicación, contará con autorización expresa y por escrito del Ingeniero Director del Proyecto, que actuará en representación de la empresa consultora.

4. En la autorización se ha de hacer constar la aplicación a que se destinan sus reproducciones así como su cantidad.

5. En todas las reproducciones se indicará su procedencia, explicitando el nombre del proyecto, nombre del Ingeniero Director y de la empresa consultora.

6. Si el proyecto pasa la etapa de desarrollo, cualquier modificación que se realice sobre él, deberá ser notificada al Ingeniero Director del Proyecto y a criterio de éste, la empresa consultora decidirá aceptar o no la modificación propuesta.

7. Si la modificación se acepta, la empresa consultora se hará responsable al mismo nivel que el proyecto inicial del que resulta el añadirla.

8. Si la modificación no es aceptada, por el contrario, la empresa consultora declinará toda responsabilidad que se derive de la aplicación o influencia de la misma.

9. Si la empresa cliente decide desarrollar industrialmente uno o varios productos en los que resulte parcial o totalmente aplicable el estudio de este proyecto, deberá comunicarlo a la empresa consultora.

10. La empresa consultora no se responsabiliza de los efectos laterales que se puedan producir en el momento en que se utilice la herramienta objeto del presente proyecto para la realización de otras aplicaciones.

11. La empresa consultora tendrá prioridad respecto a otras en la elaboración de los proyectos auxiliares que fuese necesario desarrollar para dicha aplicación industrial, siempre que no haga explícita renuncia a este hecho. En este caso, deberá autorizar expresamente los proyectos presentados por otros.

12. El Ingeniero Director del presente proyecto, será el responsable de la dirección de la aplicación industrial siempre que la empresa consultora lo estime oportuno. En caso contrario, la persona designada deberá contar con la autorización del mismo, quien delegará en él las responsabilidades que ostente.