IP core MAC Ethernet

Ing. Rodrigo A. Melo, Ing. Salvador E. Tropea
Instituto Nacional de Tecnología Industrial
Centro de Electrónica e Informática
Laboratorio de Desarrollo Electrónico con Software Libre
Email: {rmelo,salvador}@inti.gob.ar

***Resumen*—La tecnología *Ethernet*, presente en la mayoría de los dispositivos dotados de conexión a una LAN, provee comunicación entre PCs y dispositivos que funcionen en forma autónoma, tanto en ámbitos locales como a través de Internet. En este trabajo presentamos un *core* que implementa la capa MAC (*Media Access Controller*) *Ethernet*, de uso sencillo, que ofrece diversas configuraciones y ocupa pocos recursos de una FPGA. Este diseño fue simulado con herramientas de *Software Libre* y verificado en *hardware* utilizando una FPGA Virtex 4 de Xilinx.**

I. Introducción

Nuestro equipo de trabajo desarrolla sistemas embebidos que en la mayoría de los casos precisan estar comunicados con una PC. Si bien hemos desarrollado *cores* que cubran esta necesidad, como el *core* **USB** [1], en la actualidad, esta conexión deja de ser suficiente para incontables aplicaciones que precisan de un funcionamiento autónomo, que vaya más allá de un ámbito local. La tecnología *Ethernet*, presente en sus diversas variantes en la mayoría de los dispositivos dotados de conexión a una **LAN** (*Local Area Network*), sumado al uso de Internet, provee la solución más conveniente a este problema.

Se realizó una búsqueda de *cores Ethernet* disponibles, de uso libre y descriptos en **VHDL**, ya que estas condiciones forman parte de la línea de trabajo de nuestro laboratorio. Los resultados fueron pocos, siendo el más destacable el *core* **GReth** [2], perteneciente a la **GRLib** [3]. Sin embargo, el área ocupada de la **FPGA**, el complejo modo de uso y la única opción de utilización mediante un bus **AMBA** [4], excedían las características deseadas.

En este trabajo presentamos un *core* **MAC** (*Media Access Controller*) *Ethernet* compacto, y de fácil utilización, que pueda usarse en una **FPGA** de cualquier fabricante. El mismo, surgió de lo aprendido en base al estudio del *core* **GReth**. Dadas las complicaciones a nivel eléctrico que presenta el **PHY** (*PHYsical Interface*) *Ethernet*, se abordó solamente la realización de la capa **MAC** utilizando para su interconexión la interfaz **MII** (*Media Independent Interface*).

II. Core GReth

II-A. Introducción

La **GRLib** es una biblioteca de IP *cores*, distribuida mediante un sistema de doble licenciamiento: comercial y **GPL** [5]. Esta última, permitió tomar el código fuente del *core* **GReth**, estudiarlo y en base a lo aprendido desarrollar uno nuevo.

GReth provee una interfaz entre un bus **AMBA** y una red *Ethernet* (10/100 Mbit, *full- and half-duplex*). Implementa el standard 802.3-2002, sin soporte de la capa opcional de control.

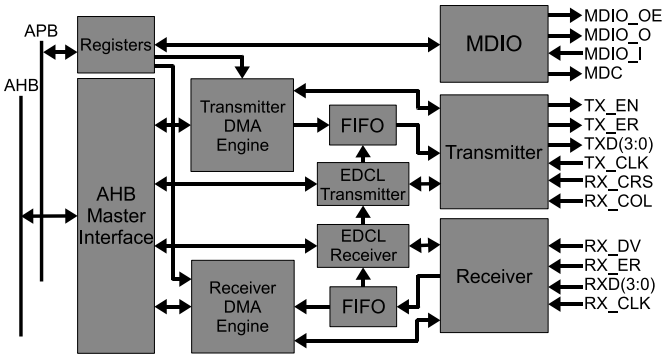

Figura 1. Diagrama en bloques de **GReth**.

II-B. Arquitectura

El diagrama en bloques de **GReth** se encuentra en la Fig. 1.

- Los buses **AMBA** utilizados son el **APB** (*Advanced Peripheral Bus*) para el manejo de registros de configuración y control, y el **AHB** (*Advanced High-performance Bus*) para flujo de datos, el cual está dado a través de canales **DMA** (*Direct Memory Access*), uno para transmisión y otro para recepción.
- La interfaz *Ethernet* se conecta a un **PHY** externo mediante las interfaces **MII** o **RMII** (*Reduced MII*) para el intercambio de datos y se utiliza la interfaz **MDIO** (*Management Data Input/Output*) para acceder a la configuración y estado.
- La interfaz **EDCL** (*Ethernet Debug Communication Link*) es opcional y provee acceso de lectura/escritura al bus **AHB** mediante Ethernet. No posee registros accesibles para el usuario y corre siempre en paralelo a los canales **DMA**.
- El *core* posee tres dominios de reloj: los de transmisión y recepción, provistos por el **PHY** externo, y el del resto de componentes y buses **AMBA**.

II-C. Descripción de hardware

El core **GReth**, y la **GRLib** en general, presentan la particularidad de estar descriptos utilizando el llamado ***Método de los dos procesos*** [6]. El mismo es aplicable a cualquier diseño sincrónico con un solo reloj y sus objetivos son:

- Proveer un algoritmo unificado de codificación.
- Incrementar el nivel de abstracción.
- Incrementar la facilidad de lectura.
- Identificar claramente secuencias lógicas.
- Simplificar el procesos de *debugging*.
- Incrementar la velocidad de simulación.
- Proveer un modelo tanto simulable como sintetizable.

Dichos objetivos, se alcanzan utilizando:

- Dos procesos por entidad.
- *Records* como puertos de las entidades.
- Codificación de algoritmos de alto nivel.

Usando dos procesos por entidad, uno conteniendo toda la lógica combinacional (asincrónico) y el otro toda la lógica secuencial (sincrónico), el algoritmo completo puede ser codificado de manera secuencial (no concurrente) en el proceso combinacional, mientras que el proceso secuencial sólo contiene asignación de *records*.

Utilizando *records* como puertos de las entidades:

- La lista de puertos se hace lo más corta posible y legible.
- Las señales se agrupan de acuerdo a su funcionalidad y dirección (entrada/salida).
- Se pueden declarar en paquetes y así agregar o quitar una señal implica modificar un sólo lugar.

La codificación de algoritmos de alto nivel, esta dada por técnicas tales como:

- Describir un sumador con un '+' en lugar de una red de compuertas.
- Utilizar la biblioteca IEEE.numeric_std, que provee muchas operaciones aritméticas y hace portable al código.
- Usar la sentencia *loop* para implementar algoritmos con prioridad e indexación.
- Multiplexar usando conversión de enteros (integer).
- Utilizar máquinas finitas de estado.
- Utilizar subprogramas (procedimientos y funciones), que pueden agruparse en bibliotecas y ser reutilizados.

II-D. Modo de uso

El *core* es controlado mediante registros de 32 *bits* mapeados en el bus **APB**:

- Registro 0: Control.
- Registro 1: Estado.
- Registro 2: *bytes* más significativos dirección MAC.
- Registro 3: *bytes* menos significativos dirección MAC.
- Registro 4: Control/Estado de interfaz **MDIO**.
- Registro 5: Dirección de memoria de la tabla de descriptores de transmisión.
- Registro 6: Dirección de memoria de la tabla de descriptores de recepción.

Los descriptores son datos de 32 *bits* transmitidos mediante el bus **AHB**. Tanto para la transmisión como para la recepción se tienen dos descriptores contiguos:

- Descriptor 0: si bien su formato varía según sea para transmisión o recepción, se conforma de *bits* de control, *bits* de estado y 11 *bits* para especificar la cantidad de *bytes* a transmitir o recibidos.
- Descriptor 1: es igual tanto para transmisión como para recepción y consiste en un puntero de 30 *bits* a la zona de memoria donde se almacena o de donde se extraen los datos.

II-D1. Transmisión: A través del bus **AHB** se colocan los datos a transmitir a partir de la dirección apuntada por el descriptor 1. A los datos se les debe añadir las direcciones MAC destino y origen, y el campo tipo/tamaño. El **CRC** (*Cyclic redundancy check*) *Ethernet* de 4 *bytes* es añadido automáticamente.

A continuación, se especifica la dirección del descriptor 0 en el registro 5. **GReth** comienza la transmisión cuando se le indica en el registro 0.

Cuando la transmisión finaliza, **GReth** escribe información de estado en el registro 1 y el descriptor 0. Finalmente apunta al siguiente par de descriptores y queda listo para la próxima operación.

II-D2. Recepción: En primer lugar, se especifica la dirección del descriptor 0 en el registro 6. **GReth** lee los descriptores cuando se le indica en el registro 0 y se queda aguardando un paquete entrante. Dicho paquete será aceptado cuando:

- está dirigido a la dirección MAC indicada en los registros 2 y 3.
- está dirigido a la dirección de *broadcast*.
- el *core* tiene habilitado el modo promiscuo.

En cualquier otro caso será descartado.

Cuando la recepción finaliza, **GReth** escribe información de estado en el registro 1 y el descriptor 0. Finalmente apunta al siguiente par de descriptores y queda listo para la próxima operación.

Los datos recibidos son accesibles a partir de la dirección apuntada por el descriptor 1.

II-D3. MDIO: Esta interfaz es opcional, seleccionable mediante un *generic* y se puede utilizar para acceder de 1 a 32 **PHY**, que contengan de 1 a 32 registros de 16 *bits*. Se maneja a través del registro 4.

Una operación de escritura se inicia escribiendo el dato de 16 *bits*, el número de **PHY** y de registro a escribir, y colocando a '1' el bit de escritura.

Una operación de lectura se inicia escribiendo el número de **PHY** y de registro a leer, y colocando a '1' el bit de lectura.

En ambas operaciones, la lectura del registro 4 permite saber si el **MDIO** está ocupado y si la operación se pudo realizar o hubo un problema.

III. Testeo del GReth

III-A. Introducción

Debido a que nuestro *core* surgiría en base a modificaciones del **GReth**, dada las complicaciones atribuibles a no ser un *core* escrito por nosotros, y a que la lectura de una descripción utilizando el ***Método de los dos procesos*** no es sencilla si no se esta acostumbrados al mismo, se decidió realizar un *testbench* para el **GReth**. El mismo nos permitió entender el modo de uso del *core*. Además, una ejecución periódica del

testeo brindaba detección de posibles errores introducidos con los cambios efectuado.

III-B. Testbench

El testeo consistió en instanciar el **GReth**, junto a una descripción denominada FakePHY, que simula ser un **PHY** y desde las interfaces **AMBA**, realizar escrituras y lecturas **MDIO**, transmisiones y recepciones mediante **MII**, y verificar que lo enviado y lo recibido coincidiera, o abortar en caso contrario.

Dado que no se pretendía mantener ni el modo de uso ni su señalización, no se abarcó toda la complejidad del **GReth**, sino lo mínimo indispensable.

Además, se desarrolló una biblioteca de procedimientos, funciones, constantes y declaraciones de tipos, para realizar operaciones sobre los buses **AMBA**, denominada AMBA Handler.

III-C. Core FakePHY

Este *core*, no sintetizable sino solamente usado para simulación, fue desarrollado para testear el funcionamiento de las interfaces **MDIO** y **MII**, simulando ser un **PHY**.

Mediante la interfaz **MDIO**, se puede escribir a diferentes direcciones y registros, para después leerlos y corroborar que los datos sean iguales.

A través de **MII**, se realizan transmisiones, que dentro del FakePHY son almacenadas, se intercambian las direcciones MAC destino y origen, se recalcula el **CRC** y el resultado se retransmite por **MII**.

Finalmente, no para el testeo del **GReth** sino para el del *core* desarrollado, se agregó la posibilidad de provocar errores intencionales mediante una señal de control, para poder así testear la respuesta a los mismos.

III-D. AMBA Handler

Para el testeo del **GReth**, se necesitaba realizar operaciones sobre los buses **AMBA**. Además, nuestro equipo de desarrollo tiene decidido utilizar la **GRLib** en en futuros desarrollos. Estas dos consideraciones nos llevaron a afrontar el desarrollo de esta biblioteca, con propósitos de simulación. No se abarcó por completo, pero si gran parte de ella, dejando de lado características avanzadas normalmente no utilizadas en desarrollos de pequeña o mediana envergadura.

Así, por un lado se implementaron ocho procedimientos que representan los casos posibles de escritura o lectura, a un maestro o esclavo, **APB** o **AHB**. Todos comparten la misma interfaz y hacen uso de tipos de datos especiales.

Por otro lado, se desarrolló un *core* muy sencillo que simula ser un arbitro **AMBA**, manejando las señales de acceso al bus de los maestros y selección de los esclavos.

Por último, varios procedimientos y funciones sirven de soporte optativo para tareas como por ejemplo:

- Carga de datos desde archivos al bus.
- Copiado, comparación y borrado de datos en el bus.
- Seteo de opciones o señales particulares.
- Muestra de información por pantalla.

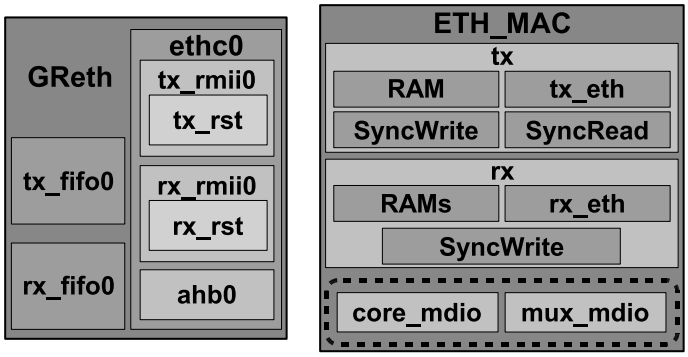

Figura 2. Esquema de instanciaciones de **GReth** (izq.) y **MAC** (der.)

IV. El Core desarrollado: **MAC** *Ethernet*

IV-A. Introducción

En la Fig. 2, se pueden ver dos esquemas resumidos de la instanciación de componentes del *core* del cual se partió (izquierda) y del *core* que se obtuvo (derecha).

El nivel superior del **GReth**, instancia las **FIFO** de transmisión y recepción, el componente ethc0 e implementa el manejo del *core* mediante descriptores, la comunicación **MDIO** y parte de **AMBA**, la interfaz **EDCL** y la sincronización entre distintos dominios de reloj. El componente ethc0, instancia a los componentes que resuelven la transmisión y recepción a través de **MII/RMII** y a un componente que resuelve la otra parte de la comunicación **AMBA**.

El *core* **MAC** desarrollado, presenta un nivel superior netamente estructural, que solamente instancia a los llamados canales de transmisión y recepción, y opcionalmente la interfaz **MDIO**. Los canales nombrados, instancian en su interior memorias **RAM** *dual port*, los componentes que resuelven la transmisión y recepción a través de **MII** y componentes para la sincronización entre distintos dominios de reloj.

IV-B. Implementación

El *core* desarrollado fue íntegramente escrito en lenguaje **VHDL** 93 estándar. Para su desarrollo se utilizaron las herramientas y lineamientos recomendadas por el proyecto FPGALibre [7].

Con respecto a **GReth** se eliminaron ciertas características, se remplazaron descripciones y se modificaron en parte o totalmente otras.

Se eliminaron las siguientes características:

- Utilización de buses **AMBA**.
- Manejo mediante descriptores.
- Interfaz **EDCL**.
- Soporte de **RMII**.

Las **FIFO** genéricas utilizadas en **GReth** fueron remplazadas por unas propias del laboratorio, implementadas sobre memoria **RAM** *dual port*. Además, las mismas pasaron a ser instanciadas dentro de los nuevos canales de transmisión y recepción, los cuales son descripciones totalmente nuevas que

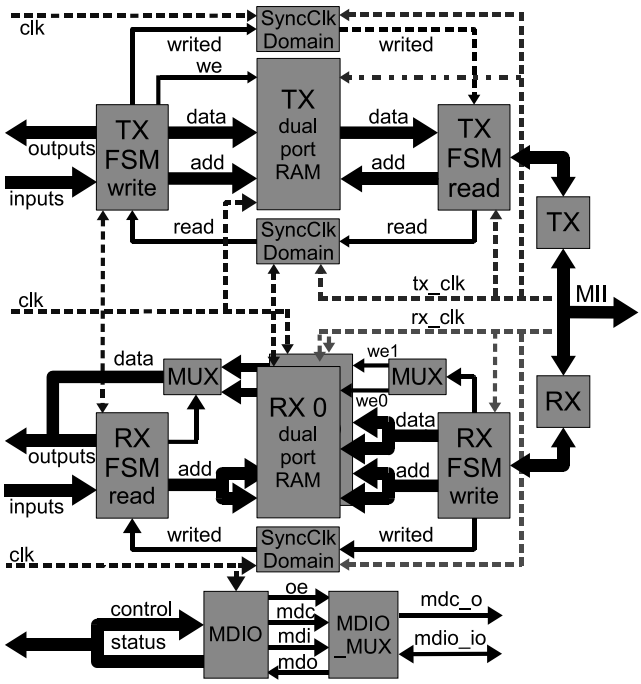

Figura 3. Diagrama en bloques del *core* **MAC**.

implementan la comunicación del **MAC** con una aplicación superior, de manera mucho más sencilla.

La funcionalidad **MDIO** se extrajo de la compleja descripción donde se encontraba para pasar a ser un componente independiente.

Los componentes que resolvían la transmisión y recepción a través de **MII**, son junto al **MDIO**, los únicos que mantienen parte de la descripción original. Esto es en parte, dado que sufrieron además de la eliminación de soporte para **RMII**, la eliminación o simplificación de estados de sus **FSM** (*Finite State Machine*), cambios o eliminación de señales de su interfaz, tanto de control como de estado o señalización de errores, eliminación de componente que filtraba posibles *glitches* en la señal de *reset*, etc. Además, estos componentes son los únicos que mantienen la utilización del ***Método de los dos procesos***.

La sincronización entre distintos dominios de reloj, que antes se daba entre las **FIFO** y los componentes de transmisión y recepción, ahora se da entre los puertos de escritura y lectura de las **RAM** *dual port*. Además, antes eran una funcionalidad esparcida por diversas zonas de la descripción, mientras que ahora utiliza un nuevo componente que desarrollamos, denominado SyncClkDomain.

IV-C. Arquitectura

La Fig. 3 muestra un diagrama en bloques resumido del *core*. En la misma se puede apreciar los tres dominios de reloj con los cuáles trabaja el sistema, así como también la independencia en cuanto a funcionamiento que hay entre los canales de transmisión, recepción y el módulo **MDIO**.

La transmisión consiste en una **FSM** que en función a las señales que comparte con una aplicación, escribe datos a una **FIFO** implementada con una **RAM** *dual port*. Al terminar de transferir datos a la **FIFO**, se genera la señal *writed*, que luego de ser sincronizada, es identificada por la **FSM** que lee los datos de la **FIFO** y los transmite a través de **MII**. Una vez leídos todos los datos, mediante la señal *read*, la **FSM** de escritura vuelve a su estado inicial y el *core* queda listo para la próxima transmisión.

La recepción es similar a la transmisión, con la diferencia que los datos escritos a la **FIFO** son los obtenidos de **MII** y los leídos de la **FIFO** son enviados a la aplicación. Además, la **FSM** de escritura no depende de la de lectura haya terminado para escribir nuevos datos sino que en su lugar se utilizan múltiples **FIFO**. Esto se debe a que depender de lecturas por parte de una aplicación superior, dado que el flujo de datos provenientes del **PHY** no se puede detener, acarrea posibles perdidas de paquetes cuando estos vienen contiguos en el tiempo.

IV-D. Configuración

El *core* presenta diversas configuraciones en base a *generics*, de las cuales se pueden destacar:

- TXFIFOSIZE y RXFIFOSIZE: utilizadas para especificar la capacidad de almacenamiento en *bytes* de las **FIFO**.
- RX_CHANNELS: permite especificar la cantidad de canales de recepción. Cada canal implica el uso de una **FIFO**.
- ENABLE_MDIO: para indicar si se hace uso o no del módulo **MDIO**.

IV-E. Modo de uso

Para el manejo de la **MAC** se tienen señales de control para:

- Habilitar y deshabilitar los canales de transmisión y recepción.
- Habilitar y deshabilitar señales de interrupción correspondientes a transmisión y recepción.
- Indicar si se transmite en *half* o *full duplex*.
- Especificar la dirección MAC del *core*, a fin de no aceptar datos entrantes indeseados.
- Activar el modo promiscuo, para capturar así todas las tramas recibidas.

IV-E1. Transmisión: se lleva a cabo indicando el inicio de la misma con una señal para tal fin. Los datos se confirman mediante una señal de escritura y se cuenta con otra señal que informa la finalización del suministro de datos. Esta interfaz posee indicación de ocupada mientras finaliza la transmisión a través de la interfaz **MII**. Además, provee información de estado para conocer la ocurrencia de errores, tales como *overrun* de la memoria de datos o alcance de límite de reintentos de transmisión en el bus.

IV-E2. Recepción: al tener datos disponibles, esta interfaz lo informa colocando una señal en estado alto, la cual se mantiene hasta la lectura de todos los datos. La lectura de dichos datos se confirma mediante la señal correspondiente o se aborta en caso de decidirse descartar el paquete. Al igual

que la interfaz de transmisión, provee información de estado. Los errores que señaliza son:

- *Overrun* de la memoria de datos.
- Paquete recibido más corto que el mínimo soportado por *Ethernet*.
- Paquete recibido más largo que el máximo soportado por *Ethernet*.
- Alineamiento erróneo.
- **CRC** erróneo.
- Cantidad de datos recibidos no concuerda con los especificados en el campo *length* del paquete recibido.

IV-E3. MDIO: presenta las mismas características del **GReth**, pero un nuevo manejo. Posee señales para especificar el número de **PHY**, de registro a escribir y datos de entrada y salida por separado. Con señales individuales se índica si la operación es una escritura o una lectura. Finalmente, cuenta con una señal de ocupado y otra de falla en la comunicación.

V. Validación del Core desarrollado

V-A. Simulación

Para la simulación se utilizó *GHDL* [8] 0.28, así como otras herramientas recomendadas por el proyecto FPGALibre.

Se realizó un *testbench*, donde nuevamente se instancia al *core* FakePHY, esta vez junto a nuestro **MAC**.

A diferencia del testeo del **GReth**, este es más riguroso, por características tales como:

- Implementa procesos separados para transmisión y recepción, en lugar de utilizar uno solo de forma secuencial.
- Verifica el funcionamiento de las indicaciones de estado, esto es, la indicación de errores.
- Los tres relojes que utiliza, ahora no son múltiplos exactos entre ellos, lo que permite una mejor simulación de la sincronización entre señales.

Por otro lado, se desarrollo un *core* que fuera una aplicación utilizada por el **MAC**. El mismo fue denominado *Replies* y lo que hace es contestar tanto peticiones **ARP** (*Address Resolution Protocol*) como **ICMP** (*Internet Control Message Protocol*). Cabe aclarar que los mecanismos que utiliza para tal fin no reflejan los especificado para estos dos protocolos, sino artilugios aptos para realizar pruebas. Este *core* se utilizó en un *testbench* para a partir de tramas *Ethernet* reales adquiridas con el *software Wireshark* [9], recrear la ejecución del comando *Ping* y poder visualizar las formas de onda y los paquetes de datos intercambiados.

V-B. Validación en hardware

Se llevó a cabo utilizando una **FPGA** Virtex 4 de Xilinx y el *software* ISE WebPack 10.1.02 - K.37. El *host* utilizado fue una computadora personal corriendo el sistema operativo Debian [10] GNU [11] /Linux.

Como aplicación se utilizó el *core Replies*, el cual es sintetizable. Una vez que el *core* superó el *testbench* sin reportar ningún error, se hicieron múltiples pruebas utilizando el comando *Ping*, que fueron desde horas hasta más de una semana de ejecución, presentando en todos los casos cero paquetes perdidos. Nuevamente, se utilizó el *software Wireshark*, en este caso para verificar la correcta conformación de los paquetes recibidos.

El **PHY** externo utilizado, fue el DP83847 de National Semiconductor. Las pruebas se realizaron a 100 Mbit *full-duplex*.

Cuadro I
Resultados de la síntesis

core **GReth**

Configuración	LUTs	FFs	Slices	BRAMs
Sin **MDIO**	1811	780	1104	2
Con **MDIO**	2013	831	1221	2

core **GReth**

Configuración	LUTs	FFs	Slices	BRAMs
1 canal RX sin **MDIO**	813	344	493	2
2 canales RX sin **MDIO**	836	341	508	5
2 canales RX con **MDIO**	994	377	588	5

VI. Resultados

En el Cuadro I pueden observarse los resultados de la síntesis de los *cores* **GReth** y **MAC**, sobre una Virtex 4.

En el caso del **GReth**, se sintetizaron las configuraciones más comunes con y sin el uso de la interfaz **MDIO**, ambos casos con la interfaz **EDCL** deshabilitada. Para el **MAC** se sintetizaron las mismas opciones, siendo dos canales de recepción el caso más común de uso, y además el caso de utilizar un sólo canal de recepción, lo cual puede ser suficiente en numerosas aplicaciones que no requieran un flujo de datos continuo.

Por otro lado, el *core* desarrollado presentó un correcto funcionamiento tras días de validación en *hardware*, haciendo uso del *core Replies* y enviando paquetes mediante el comando *Ping*. Se puede apreciar un ejemplo de 9 días de continua ejecución a continuación:

```
64 bytes from 192.168.1.99: icmp_seq=17731 ttl=64
 time=0.435 ms
64 bytes from 192.168.1.99: icmp_seq=17732 ttl=64
 time=0.438 ms
^C
--- 192.168.1.99 ping statistics ---
804164 packets transmitted, 804164 received, 0% packet loss,
 time 804166937ms
rtt min/avg/max/mdev = 0.044/0.432/0.488/0.025 ms
```

VII. Conclusiones

De la comparación de los resultados de la síntesis, puede apreciarse que se obtuvo una implementación más compacta de la que se partió. Para configuraciones de uso equivalentes, nuestro *core* utiliza menos del 50 % de área de la **FPGA** que el **GReth**, aunque en ciertos casos hace uso de más memoria. Debe considerarse también que el *core* **GReth** precisa la disponibilidad de memoria accesible mediante **AMBA**, además de todo el soporte para el manejo de descriptores, mientras que nuestro *core* cuenta con todo lo necesario para ser directamente utilizado.

En cuanto al modo de uso, el *core* desarrollado es más simple y no depende de un cierto bus, sino que puede ser fácilmente adaptado al que sea necesario, ya sea **AMBA**, ***WISHBONE*** [12] u otro. Está simplificación en el modo en que se utiliza y el cambio de arquitectura, son las principales razones de la menor ocupación de recursos de la **FPGA**.

La utilización de lenguaje **VHDL** 93 estándar, permite que el *core* sea sintetizable en una **FPGA** de cualquier fabricante.

La utilización de las herramientas propuestas por el proyecto FPGALibre demostró ser adecuada para un proyecto de estas características.

En cuanto a trabajo a futuro sobre este desarrollo, podrían reescribirse las interfaces de transmisión, recepción y **MDIO**, para dejar de utilizar el ***Método de los dos procesos*** y eliminar cualquier remanente del *core* original que puediera quedar. También puede ser interesante evitar la aparición de memoria extra utilizada cuando se configura más de un canal de recepción.

Tareas futuras sobre este trabajo, podrían implicar tanto capas de menor nivel, como la implementación del **PHY** *Ethernet*, como aplicaciones de un nivel superior, que provea manejo del protocolo **IP** (*Internet Protocol*), etc.

Referencias

[1] S. E. Tropea and R. A. Melo, "USB framework - IP core and related software," in *XV Workshop Iberchip*, vol. 1, Buenos Aires, 2009, pp. 309–313.

[2] *GRLIB IP Core User's Manual*, 1.0.19 ed. Gaisler Research, 2008, pp. 324–336.

[3] J. Gaisler, "An open-source VHDL IP library with plug&play configuration," in *IFIP Congress Topical Sessions*, R. Jacquart, Ed. Kluwer, 2004, pp. 711–718.

[4] ARM. (2010, Jun.) AMBA - Advanced Microcontroller Bus Architecture. [Online]. Available: http://www.arm.com/products/system-ip/amba/amba-open-specifications.php

[5] Free Software Foundation, Inc., "GNU General Public License," http://www.gnu.org/copyleft/gpl.html.

[6] J. Gaisler, "A structured VHDL design method," http://www.gaisler.-com/doc/vhdl2proc.pdf, Jun. 2010.

[7] S. E. Tropea, D. J. Brengi, and J. P. D. Borgna, "FPGAlibre: Herramientas de software libre para diseño con FPGAs," in *FPGA Based Systems*. Mar del Plata: Surlabs Project, II SPL, 2006, pp. 173–180.

[8] T. Gingold. (2010, Jun.) A complete VHDL simulator. [Online]. Available: http://ghdl.free.fr/

[9] G. Combs and contributors. (2010, Jun.) Network protocol analyzer. [Online]. Available: http://www.wireshark.org/

[10] I. Murdock *et al.* (2010, Jun.) Debian GNU/Linux operating system. [Online]. Available: http://www.debian.org/

[11] R. M. Stallman *et al.* (2010, Jun.) The GNU project. [Online]. Available: http://www.gnu.org/

[12] Silicore and OpenCores.Org. (2010, Jun.) WISHBONE System-on-Chip (SoC) interconnection architecture for portable IP cores. [Online]. Available: http://prdownloads.sf.net/fpgalibre/wbspec_b3-2.pdf?download