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

ABSTRACT

La tecnología ***Ethernet*** provee comunicación entre PCs y dispositivos que funcionen en forma autónoma, en ámbitos locales o a través de Internet. En este trabajo presentamos un ***core*** que implementa la capa **MAC *Ethernet***, de uso sencillo, con diversas configuraciones, que ocupa pocos recursos de una **FPGA**. El diseño fue simulado con herramientas de **Software Libre** y verificado en ***hardware*** utilizando una **FPGA** Virtex 4.

1. 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* que surgió de lo aprendido en base al estudio del *core* **GReth**. Es compacto, de fácil utilización y capaz de ser usado en **FPGAs** de cualquier fabricante.

2. CORE GRETH

2.1. Introducción

La **GRLib** es una biblioteca de IP *cores*, distribuida mediante un sistema de doble licenciamiento: comercial y **GPL** [5]. **GReth** provee una interfaz entre un bus **AMBA** y una red *Ethernet* (10/100 Mb/s, *full- and half-duplex*). Implementa el estándar 802.3-2002, sin soporte de la capa opcional de control.

2.2. Arquitectura

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

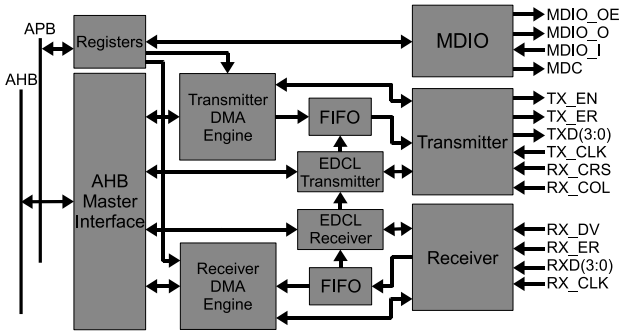

Fig. 1. Diagrama en bloques de **GReth**.

- 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, dado a través de canales **DMA** (*Direct Memory Access*) para transmisión y recepción.
- Se conecta a un **PHY** externo mediante las interfaces **MII** (*Media Independent Interface*) o **RMII** (*Reduced MII*) para el intercambio de datos y **MDIO** (*Management Data Input/Output*) para acceder a la configuración y estado.
- La interfaz **EDCL** (*Ethernet Debug Communication Link*) provee acceso de lectura/escritura al bus **AHB** mediante *Ethernet*.
- 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**.

2.3. Descripción de *hardware*

La **GRLib** está descripta utilizando el llamado **Método de los dos procesos** [6]: usando dos procesos por entidad, uno conteniendo toda la lógica combinacional y el otro toda la secuencial, el algoritmo completo puede ser codificado en el proceso combinacional, mientras que el proceso secuencial sólo contiene asignación de registros. Dicho método abstrae la descripción de *hardware* asimilándola al desarrollo de un *software*.

2.4. Modo de uso

El *core* es controlado mediante **APB** con registros de 32 *bits*:

- Registros 0 y 1: control/estado.

- Registros 2 y 3: dirección MAC.
- Registro 4: control/estado de interfaz **MDIO**.
- Registros 5 y 6: dirección de memoria de la tablas de descriptores de transmisión y recepción.

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

- Descriptor 0: se conforma de *bits* de control y estado. Utiliza 11 *bits* para especificar la cantidad de *bytes* a transferir.
- Descriptor 1: consiste en un puntero de 30 *bits* a la zona de memoria donde se almacenan/extraen los datos.

2.4.1. Transmisión

A través del **AHB** se colocan los datos a partir de la dirección apuntada por el descriptor 1. Los datos deben poseer las direcciones MAC destino y origen, y el campo tipo/tamaño. El **CRC** (*Cyclic redundancy check*) 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.

2.4.2. Recepción

Se especifica la dirección del descriptor 0 en el registro 6. **GReth** lee los descriptores cuando se le indica en el registro 0 y aguarda un paquete entrante. Dicho paquete será aceptado cuando la dirección MAC destino sea la indicada en los registros 2 y 3 o la de *broadcast*, o cuando el *core* tenga habilitado el modo promiscuo. En cualquier otro caso será descartado.

Cuando finaliza, se escribe información de estado en el registro 1 y el descriptor 0, y los datos recibidos son accesibles a partir de la dirección apuntada por el descriptor 1.

2.4.3. MDIO

Esta interfaz permite acceder de 1 a 32 **PHY**, que contengan de 1 a 32 registros de 16 *bits*. Su control y estado es accesible mediante el registro 4.

La escritura se inicia especificando el dato, número de **PHY** y registro, y colocando a '1' el bit de escritura, mientras que la lectura precisa el número de **PHY** y registro, e inicia colocando a '1' el bit de lectura.

3. TESTEO DEL GRETH

Con el objetivo de poder detectar cualquier error introducido al simplificar el *core* se diseñó un *testbench* para el **GReth**. Esto nos permitió tener un mejor conocimiento de su funcionamiento, en particular teniendo en cuenta la utilización del **Método de los dos procesos** en **GReth**.

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.

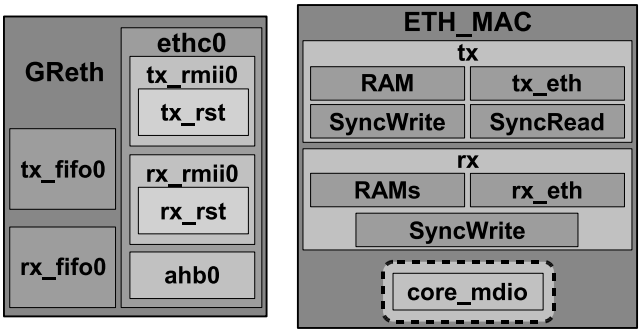

Fig. 2. Esquema de instanciaciones de **GReth** (izq.) y **MAC** (der.).

Para el manejo de **AMBA** se desarrolló una biblioteca denominada AMBA Handler, con propósitos de simulación. En la misma se implementaron ocho procedimientos que representan las combinaciones de escritura o lectura, a un maestro o esclavo, **APB** o **AHB**.

4. EL CORE DESARROLLADO: MAC *Ethernet*

4.1. 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.

4.2. Implementación

El *core* desarrollado fue 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 con memoria **RAM** *dual port*. Además, las mismas pasaron a ser instanciadas

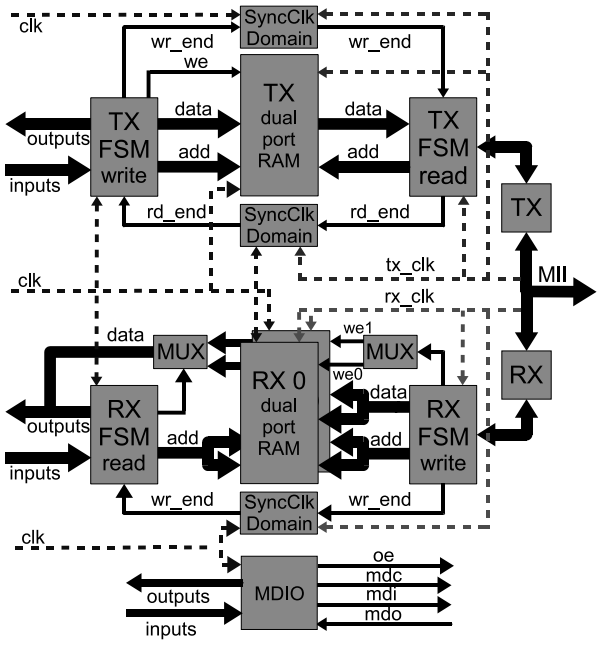

Fig. 3. Diagrama en bloques del *core* **MAC**.

dentro de los nuevos canales de transmisión y recepción, los cuales 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 y la utilización del **Método de los dos procesos**. Sufrieron cambios como: eliminación de soporte para **RMII**; eliminación o simplificación de estados de sus **FSM** (*Finite State Machine*); eliminación o cambios de señales de control y estado; eliminación de componente que filtraba posibles *glitches* en la señal de *reset*; etc.

La sincronización entre distintos dominios de reloj, antes se daba entre las **FIFO** y los componentes de transmisión y recepción, y 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 desarrollado para tal fin.

4.3. Arquitectura

La Fig. 3 muestra un diagrama en bloques *core*, donde se puede apreciar los tres dominios de reloj con los cuáles trabaja el sistema.

La transmisión consiste en una **FSM** que en función de señales de entrada, escribe datos a una **FIFO** implementada con una **RAM** *dual port*. Al terminar de transferir datos a la **FIFO**, se genera la señal *wr_end*, 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 *rd_end*, la **FSM** de escritura vuelve a su estado inicial.

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** quedan disponibles para ser usados. Para evitar la pérdida de paquetes, debido a que la aplicación no haya terminado de retirar los datos recibidos, se implementó un esquema de múltiples **FIFO**s. El número de **FIFO**s es configurable y su manejo depende exclusivamente del *core*.

4.4. Modo de uso

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: cantidad de canales de recepción a utilizar. Cada canal implica el uso de una **FIFO**.
- ENABLE_MDIO: para indicar si se hace uso o no del módulo **MDIO**.

Además, posee líneas de control para:

- Habilitar o deshabilitar los canales de transmisión y recepción.
- Habilitar o deshabilitar señales de interrupción.
- Indicar *half* o *full duplex*.
- Especificar la dirección MAC.
- Activar el modo promiscuo.

4.4.1. Transmisión

se indica el inicio y fin con señales independientes para tal fin. Los datos se confirman mediante una señal de escritura. Posee indicación de ocupado y provee información de errores de *overrun* de la memoria o alcance de límite de reintentos de transmisión en el bus.

4.4.2. Recepción

se informan datos disponibles colocando una señal en estado alto, la cual se mantiene hasta la lectura de todos los datos. Estas lecturas se confirman mediante la señal de lectura o se abortan en caso de decidirse descartar el paquete. Los errores que señaliza son: *Overrun* de la memoria de datos; paquete recibido más corto/largo que el mínimo/máximo soportado por *Ethernet*; alineamiento o **CRC** erróneo; cantidad de datos recibidos no concuerda con los especificados en el campo *length* del paquete recibido.

4.4.3. MDIO

presenta características similares al **GReth**, pero una nueva interfaz. Posee señales para especificar el número de **PHY**, de registro y datos de entrada y salida por separado. Con señales individuales se indica si la operación es una escritura o una lectura. Finalmente, cuenta con una señal de ocupado y otra de falla en la comunicación.

5. VALIDACIÓN DEL CORE DESARROLLADO

5.1. Simulación

Para la simulación se utilizó *GHDL* [8] 0.28.

Table 1. Resultados de la síntesis

core **GReth**

Configuración	LUTs	FFs	Slices	BRAMs
Sin **MDIO**	1814	775	1099	2
Con **MDIO**	2011	834	1220	2

core **MAC**

Configuración	LUTs	FFs	Slices	BRAMs
1 RX sin **MDIO**	823	333	491	2
2 RX sin **MDIO**	872	341	516	3
2 RX con **MDIO**	1016	381	591	3

Se realizó un *testbench*, donde nuevamente se instancia al *core* FakePHY, esta vez junto a nuestro **MAC**, pero a diferencia del testeo del **GReth**, este es más riguroso, incluyendo características tales como:

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

Por otro lado, se desarrolló un *core* denominado *Replies*, el cual contesta peticiones **ARP** (*Address Resolution Protocol*) e **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 para realizar pruebas. Este *core* se utilizó en un *testbench* junto a tramas *Ethernet* reales adquiridas con el *software wireshark* [9], para recrear la ejecución del comando *ping* y poder visualizar las formas de onda y los paquetes de datos intercambiados.

5.2. Validación en *hardware*

Se llevó a cabo utilizando una **FPGA** Virtex 4 de Xilinx y el *software* ISE WebPack 11.3 - L.57. 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 usando una comunicación *full-duplex* de 100 Mb/s .

6. RESULTADOS

En el Cuadro 1 pueden observarse los resultados de la síntesis de los *cores* **GReth** y **MAC**, para una Virtex 4.

En el caso del **GReth**, se sintetizaron las configuraciones más comunes con y sin el uso de la interfaz **MDIO**, en 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.

7. 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**. 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, aunque puede ser fácilmente adaptado al que sea necesario, ya sea **AMBA**, **WISHBONE** [12] u otro. La simplificación del modo de uso 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.

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

8. REFERENCES

[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