Implementación en FPGA de un mecanismo de control remoto mediante interfaz web

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

***Resumen*—Para visualización de estado y configuración de dispositivos mediante una implementación sencilla, hasta hace pocos años, se precisaba una PC dedicada. En la actualidad, gracias a la proliferación de redes informáticas y dispositivos capaces de conectarse a las mismas, se puede optar por un control descentralizado utilizando una interfaz web. En este trabajo, presentamos la implementación de un *core* microcontrolador AVR que incorpora un *Media Access Controller* (MAC) Ethernet, conectados utilizando *Memory mapped I/O* (MMIO), capaz de correr el *stack* TCP/IP uIP. Sobre el ejemplo de un pequeño *web server*, se hicieron modificaciones para dotarlo de un mecanismo capaz de leer y actualizar el estado de registros del AVR. Este diseño fue simulado con herramientas de Software Libre y verificado en *hardware* utilizando una FPGA Virtex 4 de Xilinx. Adicionalmente, se discuten optimizaciones realizadas sobre la implementación, para reducir la cantidad de memoria BRAM utilizada. Los resultados de la síntesis dan una ocupación de recursos de la FPGA por debajo al de otras implementaciones comparadas. Se obtuvo una interacción mediante *web* con el AVR, para controlar el *core* dentro de la FPGA, simple y versátil, fácilmente manejable mediante Javascript. El stack uIP se adaptó y utilizó de manera muy sencilla. La metodología MMIO resultó una forma natural para la interconexión de *cores* en la clase de dispositivos utilizados.**

***Index Terms*—AVR, MAC, Ethernet, MMIO, FPGA, uIP, Web**

I. Introducción

En los últimos años, la cantidad y variedad de dispositivos electrónicos capaces de conectarse a una red, ya sea cableada o inalámbrica, se ha incrementado notablemente. Algunos de dichos dispositivos, permiten configuraciones y pueden proveer información de estado a través de una interfaz *web*.

En nuestro laboratorio, se desarrollan instrumentos de medición y control, que frecuentemente precisan estar acompañados de una PC fija para su operación. Dotarlos de la capacidad de ser controlados y configurados a distancia, incluso a través de Internet, independientemente del *software* que se ejecute en la PC, y de manera descentralizada, resulta una alternativa sumamente atractiva. La manera sencilla de proveer un acceso *web* en dispositivos con altas prestaciones, es utilizando un sistema operativo embebido que ejecute un servidor. Esta solución puede resultar sobredimensionada para equipos con usos muy específicos y acotados.

En este trabajo, se presenta la implementación sobre FPGA del *core* ATeth, el cual surge de la unión de dos *cores* previamente desarrollados en nuestro laboratorio, un microcontrolador AVR [1] y un controlador MAC Ethernet [2], capaz de correr el *stack* TCP/IP de Software Libre denominado uIP [3]. Sobre dicho *stack*, utilizando el código del servidor *web* de ejemplo, se desarrolló un sencillo mecanismo para leer y actualizar el estado de registros del procesador, el cual puede ser combinado con técnicas como *Asynchronous JavaScript And XML* (AJAX) para evitar recargar la página tras cada operación. Adicionalmente, se realizaron optimizaciones tanto en *hardware* como en *firmware* para reducir la cantidad de memorias BRAM utilizadas por la implementación.

Este trabajo se estructura de la siguiente manera: en la sección II se describe el *core* desarrollado mientras que la sección III contiene información acerca del *stack* TCP/IP utilizado, las adaptaciones realizadas para que funcione sobre el microcontrolador AVR, el mecanismo desarrollado para interactuar con los registros del microcontrolador y algunas herramientas de Software Libre utilizadas. Las optimizaciones en el uso de BRAMs son detalladas en la sección IV. Las secciones V y VI describen las pruebas realizadas y los resultados obtenidos respectivamente. Finalmente, las conclusiones son expuestas en la sección VII mientras que la sección VIII propone trabajo futuro a realizar.

II. IP core ATeth

II-A. Implementación

Se implementó un *core* microcontrolador AVR el cual incluye un *core* controlador MAC Ethernet. Ambos *cores* se interconectaron utilizando la metodología MMIO [4] (ver sección II-B).

El *core* fue implementado utilizando el estándar VHDL 93, y se desarrolló con herramientas y bajo los lineamientos recomendados por el proyecto FPGALibre [5] [6]. El diagrama en bloques del microcontrolador obtenido, puede apreciarse en la Fig. 1.

El *core* AVR fue instanciado con una configuración equivalente a un ATmega32. Todos sus periféricos se habilitan mediante *generics* y sus características son similares a [1].

El *core* MAC Ethernet fue modificado respecto a [2] donde se usaban FIFOs y registros de control y estado. Los *buffers* de recepción y transmisión fueron mapeados en el espacio de memoria y los registros en una interfaz WISHBONE [7]

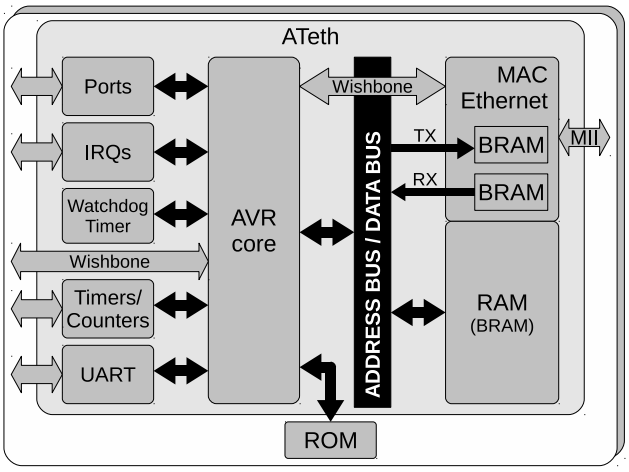

Figura 1. Diagrama en bloques de ATeth

esclava. El WISHBONE maestro fue implementado como bus interno en lugar de utilizar el externo ya disponible, para evitar limitaciones de direccionamiento y mantener compatibilidad con desarrollos previos. Por lo tanto se tienen dos maestros WISHBONE, mapeados en registros de I/O que comparten el registro de direcciones, pero poseen registros independientes para los datos.

Para una decodificación más sencilla de la memoria de sistema y los periféricos mapeados, se propuso el uso de los *bits* más significativos del bus de direcciones. Se consideraron dos posibles casos, uno utilizando los dos *bits* MSBs (cuadro I(a)) y el otro utilizando los tres MSBs (cuadro I(b)). La primer opción tiene un problema: en la arquitectura AVR, los primeros 96 *bytes* se utilizan para los *General Purpose Registers File* y los *I/O Registers*, y a continuación se ubica la RAM interna. El *stack pointer* se inicializa apuntando a la última posición de memoria, la cual por ejemplo en un ATmega32 es 2K+96. Si el bit 15 es utilizado para seleccionar la RAM, habrá disponibles 32KB, pero esto puede provocar el solapamiento entre el *stack pointer* y el canal de recepción que se ubica a partir de la dirección 32K. Para evitar este problema, se adoptó la segunda alternativa.

II-B. Memory-mapped I/O

MMIO es un método para llevar a cabo transferencias de entrada/salida entre un procesador y sus periféricos. El mismo bus de direcciones se utiliza tanto para las memorias como para los dispositivos, por lo cual las instrucciones utilizadas por el procesador son las mismas para acceder a uno u otro.

Cuadro I
MAPEO EN MEMORIA

(a) Opción 1

15	14	Device
0	0	RAM
1	0	RX
1	1	TX

(b) Opción 2

15	14	13	Device
0	0	0	RAM
1	1	0	RX
1	1	1	TX

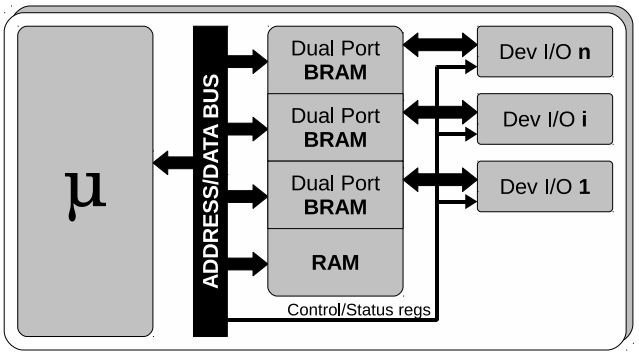

Figura 2. Arquitectura propuesta

Para su implementación, cada dispositivo precisa una memoria de doble puerto, disponibles en FPGAs actuales.

La Fig. 2 muestra un esquema general de la metodología utilizada. En la misma, el procesador se encuentra únicamente conectado al espacio de memoria donde la RAM del sistema y un puerto de cada BRAM están mapeados. El segundo puerto de cada BRAM es controlado por el periférico asociado. El control y lectura de estado se implementó utilizando registros ubicados en el mismo espacio de memoria.

II-C. Modo de uso del core MAC Ethernet

Los registros WISHBONE del *core* MAC Ethernet pueden apreciarse en la Fig. 3. El registro 0 se utiliza para configuración y control. Si los *bits* TXerr o RXerr están en 1 tras una transferencia, significa que hubo un error y el detalle del mismo se puede obtener del registro 1. Los registros 2 y 3 se utilizan en conjunto para indicar el número de *bytes* recibidos (lectura) o a transmitir (escritura). Sólo se utilizan 11 *bits* (2KB) debido a que el tamaño máximo de un *frame* Ethernet es 1514 *bytes*.

Un *frame* Ethernet consiste de 6 *bytes* para la dirección MAC destino, 6 *bytes* para la dirección MAC origen, 2 *bytes* para el campo *length/type*, la información contenida y finalmente el CRC, el cual es automáticamente agregado en transmisión y descartado en recepción.

- Configuración: los *bits* TXen, RXen, Full y Prom del registro de Control son respectivamente utilizados para

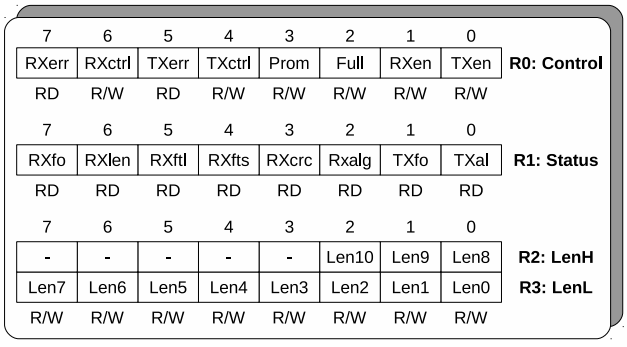

Figura 3. Registros del core MAC Ethernet

habilitar/deshabilitar los canales de transmisión y recepción, el modo *full duplex* y el modo promiscuo.
- Transmisión: para determinar si el canal esta listo para una nueva transmisión, el usuario debe asegurarse que el *bit* TXctrl esté en 0. Los datos a ser transmitidos deben copiarse a la memoria de transmisión y el número de *bytes* involucrados debe ser especificado en los registros 2 y 3. La transmisión comienza cuando se coloca un 1 en el *bit* TXctrl. Si ocurre un error, el *bit* TXerr se coloca en 1 y puede obtenerse información detallada de los *bits* 0 y 1 del registro de Estado.
- Recepción: habrá datos disponibles cuando el *bit* RXctrl esté en 0. La cantidad de datos recibidos se obtiene de los registros 2 y 3. Si ocurrió un error, el *bit* RXerr estará en 1 y los detalles del mismo pueden obtenerse de los *bits* 2 a 7 del registro de Estado. Finalmente, es necesario indicar que se procesaron los datos, colocando un 1 en el *bit* RXctrl.

III. Firmware

III-A. El stack TCP/IP

Se realizó una búsqueda de *stacks* TCP/IP que tuvieran bajos requerimientos de *hardware* y una licencia libre. Se seleccionó el *stack* denominado uIP versión 1.0, el cual en la actualidad forma parte del sistema operativo Contiki [8]. Este *stack* está escrito en lenguaje C, provee conectividad TCP/IP a microcontroladores de 8 *bits* y es compatible con los documentos RFC correspondientes. Como aplicación de pruebas se seleccionó un *web server* sencillo, utilizado para servir 4 páginas diferentes.

Las siguientes características hicieron a uIP adecuado para nuestro proyecto:

- Posee muy pocos requerimientos de *hardware*.
- Es utilizado ampliamente en sistemas embebidos.
- Había una versión implementada con un AVR, por lo cual se sabía con antelación que probablemente funcionaría en ATeth.
- Es Software Libre, con lo cual se puede usar, modificar y distribuir sin la necesidad de pagar regalías.
- El código fuente es estructurado y modular.
- Está bien documentado.
- Hay ejemplos disponibles, incluido el *web server* seleccionado.

III-B. Modificaciones del stack

Para utilizar uIP con ATeth, fue necesario escribir un controlador para el *core* MAC Ethernet. Básicamente, se implementaron funciones para inicializar, enviar y recibir datos, siguiendo los pasos descriptos en la sección II-C.

El ejemplo utilizado para el *web server* fue diseñado simulándolo con interfaces de red virtuales bajo GNU[9]/Linux. Fue necesario modificar dicho ejemplo para que funcione con una interfaz Ethernet real.

Por otra parte, los 2KB de memoria de datos del ATmega32 no eran suficientes para contener las páginas *web* embebidas. Nuestra implementación sobre FPGAs soporta la configuración del tamaño de la memoria. El único cuidado especial a tener en cuenta es mover la dirección de inicio del *stack pointer* en nuestro caso a la dirección 0x405F (16K+96 *bytes*).

III-C. Web Server de ejemplo

El *web server* de ejemplo de uIP consiste en cuatro páginas navegables, donde una de ellas muestra información estática, mientras que las otras tres varían con cada recarga. Esto lo logra a partir de la implementación de un pequeño sistema de *scripts*, que soporta tres comandos estadísticos (muestra uno en cada página):

- Acceso a archivos del servidor.
- Procesos corriendo.
- Datos de conexión

Adicionalmente, el sistema de *scripts* permite la inclusión de una página dentro de otra.

Si bien el contenido a servir es y se desarrolla como el típico contenido de una *web* (imágenes, archivos HTML, CSS y Javascript, etc), es convertido a archivos con cadenas constantes y estructuras simplemente enlazadas en lenguaje C, y luego compilado junto al *web server* y uIP.

III-D. Desarrollo sobre el Web Server

Al dotar a un dispositivo de una interfaz *web*, además de poder visualizar datos del sistema, se necesita poder interactuar desde dicha interfaz. La manera típica de resolverlo en una aplicación *web* real, es mediante formularios, pero implementar el soporte en el pequeño *web server*, dejaba de hacerlo pequeño y excedía las prestaciones buscadas. Se desarrolló entonces un sencillo mecanismo mediante el cual se pueden leer registros del microcontrolador y actualizar sus valores a través de una *URL*, de la siguiente manera:

- Lectura: *http://URLBASE/cgi/XX*
- Escritura: *http://URLBASE/cgi/XX/YY*

Donde XX e YY son números hexadecimales de dos cifras: XX representa el número de registro a leer/escribir mientras que YY es el valor a escribir sobre el registro XX.

En la Fig. 4 se aprecia una página de prueba del mecanismo. Tanto los *leds*, como los pulsadores y los *dip switches* reflejan el estado de sus equivalentes en el *kit* utilizado. Al *clickear* sobre un *led*, se altera su estado en la placa y por ende en la página. Todo esto se logra sin necesidad de recargar la página, haciendo uso de AJAX.

Al programa que convierte los archivos *web* a archivos en C, se le modificó para no estar atado al ejemplo de uIP, sino poder especificar el directorio con archivos a convertir y de salida de datos, para permitir su reutilización en diferentes proyectos.

III-E. Herramientas de Software

El *core* implementado soporta el juego de instrucciones completo del procesador original y una configuración equivalente a un microcontrolador existente, por lo cual es posible utilizar herramientas de *software* disponibles, tales como el avr-gcc, una versión del compilador de C de GNU [10].

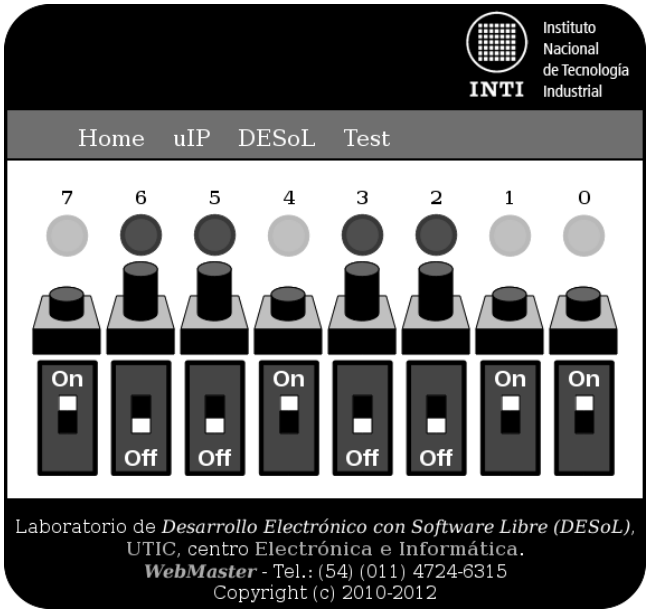

Figura 4. Página web de testeo

La Fig. 5 muestra la cadena de herramientas utilizada para obtener la descripción de *hardware* de la memoria de programa en base al código fuente del *web server*. Comenzando por las fuentes en C (y sus encabezados .h) se obtienen el ejecutable (elf) utilizando el avr-gcc. Luego, la memoria de programa (bin) es extraída mediante el avr-objcopy. Las herramientas bin2hex y vhdlspp fueron desarrolladas por nuestro laboratorio como parte del proyecto FPGALibre: bin2hex convierte un archivo bin en valores hexadecimales para asignaciones VHDL (dat) y luego vhdlspp (*VHDL Simple Pre Processor*) los combina con el esqueleto de una ROM, dando como resultado el archivo a utilizar junto a ATeth.

Se utilizó la herramienta avr-size en la determinación de la cantidad de memoria de programa necesaria para la aplicación resultante, y con esta información se dimensionó correctamente la memoria del sistema. La herramienta muestra la cantidad de *bytes* utilizados por las secciones *text*, *data* y *bss*. La suma de las secciones *text* (código ejecutable) y *data* (valores de las variables inicializadas) es igual al número de *bytes* utilizados por la memoria de programa, mientras que la suma de *data* y *bss* (variables no inicializadas) es la cantidad de *bytes* de la memoria de datos.

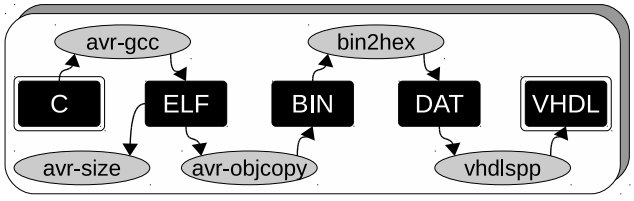

Figura 5. Obtención de la memoria de programa

IV. Reducción del uso de BRAMs

Las BRAMs son recursos frecuentes en una FPGA, pero finitos, por lo cual deben ser utilizados con moderación. Para optimizar su utilización, se analizaron diferentes alternativas, realizando cambios tanto en el *hardware* como en el *software*.

IV-A. Colocación de la ROM en una memoria flash paralela

Se observó que gran parte de las BRAMs eran utilizadas como memoria ROM por el AVR, debido a que el código del *stack* TCP/IP era de aproximadamente 20KB. Se decidió mover la ROM del microcontrolador a una memoria *flash* paralela externa, preexistente en el *kit* de desarrollo utilizado. La misma contaba con interfaz CFI [11], un estándar abierto desarrollado por AMD, Fujitsu, Intel y Sharp, el cual puede proveer información a través de secuencias predefinidas de comandos. Esta información incluye detalles tales como el tamaño de la memoria, tiempos de borrado, voltajes utilizados, etc. Existen muchos tipos de comandos para dialogar con las memorias CFI, cada uno definido por su fabricante. Los mas comunes son *AMD/Fujitsu Command Set* e *Intel/Sharp Command Set*. Nuestra memoria CFI utilizaba el segundo.

Para trabajar con esta clase de memorias, nuestro laboratorio desarrolló un *core* controlador CFI. En resumen, es una *Finite State Machine* (FSM) en *hardware*, que permite leer y escribir la memoria *flash* con comandos enviados a través del USB de una PC. Los pasos necesarios para cambiar el programa de ATEth son: primero, la FPGA tiene que ser configurada con el controlador CFI. Luego, la memoria *flash* es grabada mediante USB. Finalmente, la FPGA se configura nuevamente, pero esta vez con ATeth.

La memoria utilizada tiene un tiempo de lectura de página mayor al tiempo de acceso a datos. Dado que ATeth no puede insertar estados de espera por sí mismo, debía leer la memoria a una velocidad compatible a los cambios de página, lo cual implicaba trabajar a menor frecuencia que en el caso previo a utilizar la *flash* externa. Para mejorar el rendimiento durante operaciones de lectura, se utilizó el *core Flash ROM Interface* [12]. El mismo, inserta estados de espera al microcontrolador cuando se está efectuando un cambio de página, y el resto del tiempo permite extraer los datos a la máxima velocidad.

IV-B. Optimizaciones del firmware

El *firmware* del uIP fue desarrollado para ser independiente del dispositivo donde se ejecuta, por lo cual, no posee consideraciones de *hardware* particulares.

El AVR usa una arquitectura Harvard modificada donde el programa y los datos son almacenados en espacios separados de memoria, los cuales utilizan diferentes espacios de direcciones. Por defecto, las instrucciones toman los datos desde la RAM.

Las cadenas constantes de datos, son almacenadas en la ROM y copiadas a la RAM antes de pasar el control a la función *main()*. Esta redundancia de datos requiere mas memoria de la realmente necesaria.

El compilador gcc provee un mecanismo especial para evitar esta duplicación. Esto se logra mediante el uso de atributos,

una extensión del estándar ANSI C. Como efecto secundario, el código debe copiar la constante en la memoria RAM antes de llamar a la función que la utiliza. Un *buffer* es utilizado para mantener sólo una de las constantes necesarias en RAM.

Se aplicaron estos atributos principalmente sobre las constantes que representan las páginas *web*, las cuales son una gran cantidad de datos, y se realizaron algunas modificaciones al código del uIP, basadas en una adaptación a un AVR [13].

V. Testeo y validación

V-A. Banco de pruebas

Para la simulación del *core* se utilizó GHDL [14] 0.29. Se realizó un banco de pruebas muy sencillo para probar la interconexión entre el AVR y el MAC Ethernet:

- El programa responde paquetes de tipo ARP (*Address Resolution Protocol*) e ICMP (*Internet Control Message Protocol*) a través de la interfase MAC Ethernet. Llamamos a este programa loopback_eth.
- Para generar estímulos, se usaron programas originalmente desarrollados para el proyecto del MAC Ethernet. Se generaron dos archivos en base a datos aleatorios: el primero con datos de recepción y el segundo con los correspondientes a la transmisión.
- El banco de prueba inyecta datos del archivo de recepción y compara la salida obtenida con los datos del archivo de transmisión, abortando en caso de detectarse diferencias.

V-B. Validación en hardware

Este desarrollo se validó utilizando una FPGA Virtex 4 (XC4VLX25) de Xilinx y el *software* ISE WebPack 11.3 - L.57. La memoria CFI externa es una Intel Strata Flash TE28F640. Se utilizaron computadoras personales corriendo el sistema operativo Debian [15] GNU/Linux.

La primer prueba consistió en utilizar el programa loopback_eth sobre el AVR. Se comprobó el funcionamiento del sistema mediante el envío y recepción de paquetes ICMP (comando *ping*), y luego se analizaron los paquetes intercambiados entre la PC y el *core* con el programa *wireshark* [16].

Las siguientes pruebas incluían el *stack* uIP. Se realizó un programa que descarga las páginas *web* servidas y las compara con las versiones de desarrollo. Nuevamente, se analizó el comportamiento frente a la ejecución de comandos *ping*, pero esta vez en *hardware*. Se navegaron las páginas *web* de ejemplo y las desarrolladas, donde se encuentra la página enseñada en la sección III-D, que permite el testeo del mecanismo de lectura/escritura de registros implementado.

VI. Resultados

VI-A. Síntesis

El cuadro II(a) muestra los resultados de la síntesis de ATeth para cada modificación y mejora descriptas en la sección IV, y con la versión propia del *web server*. Adicionalmente, se muestra en el mismo cuadro la cantidad de memoria utilizada en cada caso. En el caso 1, la ROM está implementada sobre BRAMs, mientras que en el caso 2 y los siguientes, se encuentra en la memoria *flash* externa (CFI). En los casos 1, 3, 4 y 5, la frecuencia de operación fue 50 MHz, la cual corresponde a uno de los relojes disponibles en la placa de evaluación de Virtex 4 utilizada. En el caso 2, el sistema corre a 8 MHz utilizando un *Digital Clock Manager* (DCM), debido a que la memoria *flash* tiene una demora de 120 ns cuando realiza un cambio de página.

Comparando los casos 1 y 2, el consumo de BRAMs disminuyó en un 60 % y los *slices* tuvieron una ligero decremento del 6 % cuando la ROM se movió a la *flash*. Por otro lado, el tiempo de respuesta frente al comando *ping* se triplicó. Por lo tanto, en el caso 2 se consume menos *hardware* pero el tiempo de respuesta es peor.

En el caso 3 se utilizó el *Flash ROM Interface* para mejorar los tiempos de acceso a la *flash*. En comparación al caso 1, los resultados dan un área y tiempo de respuesta similares, pero las BRAMs utilizadas se mantienen estables respecto al caso 2.

El caso 4 corresponde a optimizaciones en el *firmware*, con la intención de reducir las redundancias en RAM. 5.5K *bytes* de las páginas *web* se movieron a la memoria de programa. En este caso la RAM se reduce 90 % y consecuentemente las BRAMs utilizadas un 55 % respecto al caso 3 y un 80 % respecto al caso 1. La cantidad de *bytes* en ROM aumentó ligeramente un 15 % debido al uso de funciones especiales para leer las constantes desde esta memoria en lugar de la RAM.

Finalmente, el caso 5 corresponde a la implementación con el *web server* modificado corriendo nuestra propia aplicación web. El aumento de área en la FPGA se debe a la habilitación de tres puertos del AVR para interactuar con los *leds*, *switches* y *dip switches* del *kit*. El aumento de tamaño de la ROM, se debe en gran parte a que la página *web* desarrollada es más elaborada (más imágenes, utiliza JavaScript para actualizar el estado de la placa, etc), y en menor medida, a los cambios en el *web server* para soportar el mecanismo de lectura y actualización de registros.

VI-B. Otras implementaciones

Se encontraron 2 notas de aplicación [17] [18] de Xilinx, las cuales pueden ser utilizadas para contrastar algunos resultados. El sistema diseñado en estas notas de aplicación tiene las siguientes características

- Se utiliza una FPGA Virtex 4 (XC4VFX12).
- Un procesador (*softcore* o *hardcore*) es conectado a 16 Kb de BRAM.
- Se utiliza una memoria SRAM externa para contener los 3.8KB de páginas *web*.
- Se utiliza una memoria DDR SDRAM externa para el código ejecutable.
- Se conectan *cores* UARTLite, Ethernet 10/100 MAC y General Purpose I/O.
- Se utiliza el *On-chip Peripheral Bus* (OPB) para las interconexiones.
- Se utiliza el *stack* lwIP [19].

Los resultados extraídos de las notas, como así también los de este trabajo, se encuentran en el cuadro II(b). La misma

Cuadro II
RESULTADOS DE LA SÍNTESIS

(a) El IP core desarrollado

Caso	FFs	LUTs	Slices	Fmax (MHz)	BRAMs	PING (ms)	Text	Data	bss	ROM	RAM
1	739	2558	1486	62	27	0,25	13640	6774	7045	20414	13819
2	736	2411	1417	56	11	0,74					
3	754	2567	1499	61	11	0,32					
4	727	2612	1488	61	5	0,32	23946	30	1224	23976	1254
5	822	2539	1524	61	5	0,32	45748	52	1208	45800	1260

(b) Comparación con otros cores

Caso	Procesador	Tipo	Slices	GCLKs	BRAMs
xapp433	MicroBlaze	Softcore	3737	5	8
xapp434	PowerPC	Hardcore	4383	5	12
ATeth	AVR	Softcore	1524	3	5

muestra que nuestra implementación utiliza aproximadamente 2,5 veces menos *slices*, ocupa menos BRAMs y usa 3 relojes globales en lugar de 5.

VII. CONCLUSIONES

La metodología MMIO, resultó ser de muy sencilla implementación. Una vez adaptado el *core* MAC *Ethernet* y mapeado sus registros a *WISHBONE*, la interconexión con el AVR y manejo desde el *firmware* fueron triviales.

Si bien no es posible realizar una comparación totalmente justa con otras implementaciones, es posible apreciar que nuestro desarrollo es más compacto. Ocupa menos de la mitad del área de la FPGA que los ejemplos de las notas de aplicación con similares características, así como también menor cantidad de BRAMs.

La estrategia de mover la memoria de programa de las BRAM internas a *flash* externa, incidió notablemente sobre el número de BRAMs utilizadas, pero acarreó el degradamiento de la frecuencia de operación del sistema, que se reflejó en la velocidad de respuesta del *stack*, como se puede observar en las contestaciones al comando *ping*. La utilización del *core Flash ROM Interface* mejoró la velocidad de respuesta. Utilizar los *string* de datos directamente desde la ROM, disminuye el tamaño de la sección *data*, liberando así más BRAMs.

La adaptación y utilización del *stack* TCP/IP uIP y el *web server* de ejemplo sobre ATeth fueron sencillas y demostraron la versatilidad del mismo para su utilización en sistemas embebidos.

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.

VIII. TRABAJO FUTURO

Como trabajo futuro, sería recomendable implementar un *file system* sobre una memoria y ahí colocar los archivos *web*, evitando así su compilación junto a uIP y el *web server*, facilitando e independizando el desarrollo de la interfaz de visualización y configuración del dispositivo.

REFERENCIAS

[1] S. E. Tropea and D. M. Caruso, "Microcontrolador compatible con AVR, interfaz de depuración y bus wishbone," in *Proceedings of the FPGA Designer Forum 2010*, Ipojuca, Brazil, 2010, pp. 1–6.

[2] R. A. Melo and S. E. Tropea, "IP core MAC Ethernet," in *Proceedings of the FPGA Designer Forum 2011*, Córdoba, Argentina, 2011, pp. 1–4.

[3] A. Dunkels. (2011, Oct.) uIP TCP/IP stack. Networked Embedded Systems group / Swedish Institute of Computer Science. [Online]. Available: "http://www.sics.se/˜ adam/old-uip/"

[4] R. Melo, D. Caruso, and S. Tropea, "Memory-mapped I/O over Dual Port BRAM on FPGA," in *2012 VIII Southern Conference on Programmable Logic*, Bento Gonçalves, Rio Grande do Sul, Brasil, 2012, pp. 99–104.

[5] 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.

[6] INTI Electrónica e Informática *et al.*, "Proyecto FPGA Libre," http://fpgalibre.sourceforge.net/.

[7] Silicore and OpenCores.Org. (2011, Oct.) WISHBONE System-on-Chip (SoC) interconnection architecture for portable IP cores. [Online]. Available: http://prdownloads.sf.net/fpgalibre/wbspec_b3-2.pdf?download

[8] (2011, Oct.) The operating system for the internet of things. Cisco, Redwire LLC, SAP, SICS, and others. [Online]. Available: http://www.contiki-os.org/

[9] "GNU project," http://www.gnu.org/, Jun. 2010.

[10] (2009, Nov.) GCC, the GNU compiler collection. [Online]. Available: http://gcc.gnu.org/

[11] JEDEC Solid State Technology Association. Common Flash Interface (CFI). [Online]. Available: http://www.jedec.org/download/search-/jesd68-01.pdf

[12] D. M. Caruso and S. E. Tropea, "Comparación del desempeño de microcontroladores AVR de 4ta generación," in *Congreso Argentino de Sistemas Embebidos - Libro de Trabajos*, Buenos Aires, Argentina, 2011, p. 179.

[13] T. Harbaum. (2012, Feb.) Wlan for avr. [Online]. Available: "http://www.harbaum.org/till/spi2cf/index.shtml"

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

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

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

[17] (2011, Oct.) Embedded system example: Web server design using microblaze soft processor. Xilinx. [Online]. Available: http://-www.xilinx.com/support/documentation/application_notes/xapp433.pdf

[18] (2011, Oct.) Web server reference design using a powerpc-based embedded system. Xilinx. [Online]. Available: http://www.xilinx.com/-support/documentation/application_notes/xapp434.pdf

[19] L. W. Adam Dunkels *et al.* (2011, Oct.) The lwIP TCP/IP Stack. Swedish Institute of Computer Science. [Online]. Available: "http://www.sics.se/ adam/lwip/"