
# Actas de las

# X JORNADAS DE COMPUTACIÓN RECONFIGURABLE Y APLICACIONES

# JCRA 2010

## EDITORES

Javier Valls
Asunción Pérez
Trinidad Sansaloni
María José Canet


**Actas de las X Jornadas de Computación Reconfigurable y Aplicaciones JCRA 2010**

**Editores:** Javier Valls, Asunción Pérez, Trinidad Sansaloni, María José Canet

**ISBN:** 978-84-92812-56-1

**IBERGARCETA PUBLICACIONES, S.L., Madrid, 2010**

**Edición:** 1ª
**Impresión:** 1ª
**Nº de páginas:** 312
**Formato:** 17 x 24
**Materia CDU:** 004 Ciencia y tecnología de los ordenadores. Informática



1ª Edición, 1ª Impresión

ISBN: 978-84-92812-56-1

Depósito legal: M-

**Maquetación**: Los Editores

**Coordinación del proyecto:** @LIBROTEX

**Portada:** Estudio Dixi

**Impresión y encuadernación:**

OI: 19/2010

PRINT HOUSE, S.A.

**IMPRESO EN ESPAÑA -PRINTED IN SPAIN**

## Sesión 9. IP Cores, Test, Educación

# Verificación on-chip de larga duración de sistemas con eventos lógicos dispersos en el tiempo

J. Viejo, J. I. Villar, J. Juan, A. Millan, M. J. Bellido y J. Quiros

Universidad de Sevilla

E. T. S. Ingeniería Informática

Dpto. Tecnología Electrónica-Grupo ID2 (Investigación y Desarrollo Digital)

{julian,jose,jjchico}@dte.us.es, amillan@us.es,{bellido,jquiros}@dte.us.es

## Resumen

Tradicionalmente, en el proceso de desarrollo de sistemas electrónicos digitales la verificación de los diseños supone una de las tareas más costosas en tiempo y dinero. Para mejorar la eficiencia de este proceso se han aportado al mercado diversas soluciones que facilitan la depuración tanto *off-chip* como *on-chip* de sistemas de alta velocidad utilizando una aproximación similar a los analizadores lógicos tradicionales. Sin embargo, existen otro tipo de diseños, tales como los sistemas de sincronización de red, en los que la verificación de su correcta operación requiere el análisis de un gran número de eventos distribuidos sobre un largo periodo de tiempo.

Para verificar este tipo de sistemas se hace necesario desarrollar nuevas herramientas *ad-hoc*. En este sentido, en este trabajo se presenta una herramienta de verificación *on-chip* basada en PicoBlaze adaptable a los sistemas que requieran la captura de eventos dispersos en el tiempo, aunque no limitada exclusivamente a este tipo de diseños. Finalmente, se incluye una comparación de la herramienta propuesta frente a una de las alternativas comerciales más conocidas en la actualidad, en concreto, la herramienta de verificación *ChipScope Pro* de Xilinx.

## 1. Introducción

En la actualidad el desarrollo de sistemas electrónicos digitales se ha popularizado experimentando un gran crecimiento debido fundamentalmente a la reducción de costes que han supuesto para gran cantidad de usuarios (diseñadores y empresas) los dispositivos electrónicos programables tipo FPGA. Esta disminución de costes se basa principalmente en la reducción del tiempo de diseño que ha supuesto la utilización de este tipo de dispositivos en la fase de verificación del diseño.

El alto coste de fabricar un prototipo sobre ASIC impone la necesidad de realizar una etapa de simulación funcional exhaustiva con la finalidad de depurar y comprobar en su totalidad la correcta operación del sistema antes de su fabricación en silicio [1]. En este caso, el principal inconveniente que presenta la simulación funcional reside en la velocidad con la que ésta se realiza. Así, esta opción puede resultar factible en sistemas en los que en un corto espacio de tiempo se puede determinar su buen funcionamiento. Sin embargo, existen otro tipo de sistemas en los que la simulación funcional no resulta viable debido a que para comprobar ciertos eventos se necesitan ejecutar muchos ciclos de reloj, requiriéndose en algunos casos días, semanas o incluso meses hasta completar la verificación. Esto en definitiva supone un enorme esfuerzo convirtiendo la etapa de simulación en una de las fases a las que hay que dedicar mayor tiempo y recursos.

La principal ventaja de utilizar dispositivos programables consiste en que no es necesario realizar una fase de simulación funcional tan exhaustiva como ocurre en el caso del ASIC. En efecto, diseñar sobre FPGA permite un rá-

pido prototipado del diseño y su depuración sobre *hardware* real, lo cual acelera la verificación del diseño y reduce el tiempo de desarrollo. Si a esto unimos la capacidad de reconfiguración de estos dispositivos se puede concluir que esta alternativa supone un ahorro económico importante.

En este sentido, los diferentes fabricantes de FGPA han puesto a disposición de los diseñadores un conjunto de herramientas que permiten la verificación *hardware* de un sistema digital: *ChipScope Pro* [2] de Xilinx o *SignalTap II* [3] de Altera entre los más destacados. En general, estas herramientas se adaptan a las necesidades de la mayoría de los diseños que se pretenden verificar ya que presentan características como altas frecuencia de muestreo y opciones avanzadas de disparo; sin embargo, debido a la gran variedad de sistemas que se pueden implementar sobre estos dispositivos en determinadas ocasiones no es posible realizar una correcta depuración empleando estas herramientas, siendo un ejemplo concreto un sistema de sincronización de red previamente desarrollado por los autores [4] donde para una correcta verificación del sistema es fundamental la adquisición de un gran número de eventos distribuidos sobre un largo periodo de tiempo. El principal problema encontrado viene motivado por el hecho de que las alternativas de verificación existentes emplean recursos propios de la FGPA para almacenar los datos que van adquiriendo de forma que el número máximo de muestras viene limitado por los recursos libres que deje el diseño bajo test.

En estos casos, se hace necesario desarrollar nuevas herramientas *ad-hoc* que permitan depurar cada sistema específico. Así, en este trabajo se presenta el desarrollo de una herramienta que puede ser adaptada a cualquier sistema en el cual la verificación se base en la captura de eventos dispersos en largos periodos de tiempo.

A continuación, se indica como se va a organizar este trabajo: en el apartado 2, se presentará un análisis detallado de las herramientas de verificación existentes desde el punto de vista de su aplicación al tipo de sistema que se está pretendiendo verificar. En el apartado 3, se introducirá la arquitectura de la plataforma de verificación *on-chip* que se ha desarrollado. El apartado 4 presenta la aplicación real que ha servido de base para realizar una comparativa de la herramienta propuesta respecto de la alternativa comercial *ChipScope Pro*. Finalmente, en el apartado 5 se resumen las conclusiones más importantes derivadas de esta experiencia.

## 2. Soluciones actuales para verificación *hardware* sobre FPGA

La alta densidad de integración de los dispositivos programables han propiciado que se puedan implementar diseños cada vez más complejos lo cual dificulta significativamente las tareas de verificación de los sistemas desarrollados sobre ellos [1]. En este aspecto, hasta hace pocos años, la verificación *hardware* se ha realizado mediante la utilización de analizadores lógicos externos. Estos analizadores lógicos presentan principalmente dos inconvenientes a la hora de testear un diseño implementado sobre un dispositivo programable:

1. El acceso a las señales del sistema sólo se puede realizar a través de pines de la FPGA. Esto hace necesario tener que rutar las señales internas que se deseen capturar hacia pines de la FPGA, lo cual presenta a su vez una serie de limitaciones. Por un lado, la conexión de las sondas del analizador lógico con los pines de la FPGA resulta una tarea compleja y en algunas ocasiones imposible debido a que con las tecnologías de empaquetado actuales no se puede asegurar una completa visibilidad de todos los pines de la FPGA. Por otro lado, estas conexiones de señales internas a pines de la FPGA generan nuevos caminos lógicos con temporizaciones diferentes de las originales.

2. El tamaño de la ventana temporal de captura está limitado por el número de muestras máximo que permita almacenar en memoria el analizador lógico que se esté empleando.

Por estas razones se ha hecho imprescindible la adopción de nuevas metodologías y herramientas eficientes para la depuración y verificación de los diseños implementados sobre FPGA [5, 6, 7]. Así, desde hace unos años se han puesto a disposición de los diseñadores un conjunto de herramientas de verificación basadas en la integración de un analizador lógico dentro del propio dispositivo programable [2, 3, 8, 9, 10]. Este tipo de verificación se ha denominado depuración *on-chip* ya que el analizador se añade como un componente más del sistema y se implementa sobre el dispositivo programable junto con el diseño. Con estas herramientas se consigue resolver el problema de poder acceder a las señales internas del sistema, aunque siguen presentando los siguientes inconvenientes:

1. Realizar la conexión de las señales con el analizador requiere un esfuerzo por parte del diseñador, sobre todo si es necesario capturar señales pertenecientes a subsistemas muy alejados del *top level* del sistema.

2. Estas alternativas emplean recursos de la FPGA tanto para la implementación del analizador lógico integrado como para almacenar las muestras capturadas. En este sentido, los recursos utilizados dependen del número de señales y del número de muestras, estando limitado el tamaño de la ventana temporal de captura por la cantidad de memoria RAM que deja libre el diseño bajo test.

3. Al introducir lógica adicional consiste en una técnica intrusiva. Esto significa que el rendimiento total del sistema bajo test será menor e incluso podría presentar características funcionales y temporales diferentes a las del circuito original [6].

4. No son capaces de capturar muestras de determinados tipos de señales como por ejemplo de señales triestado.

Como se puede deducir del análisis realizado el principal problema que presentan estas dos alternativas consiste en que el número de muestras que se pueden capturar del sistema está limitado por la cantidad de memoria de que se disponga. En este aspecto, existen multitud de aplicaciones donde esta limitación no resulta un serio inconveniente para la verificación del sistema. En efecto, en la actualidad el aspecto crítico suele ser la velocidad de captura debido a que en general los circuitos bajo test trabajan a frecuencias de operación elevadas. Por esta razón, las herramientas existentes no están optimizadas para la adquisición de un gran número de muestras sino para capturar datos a alta frecuencia (Fig. 1).

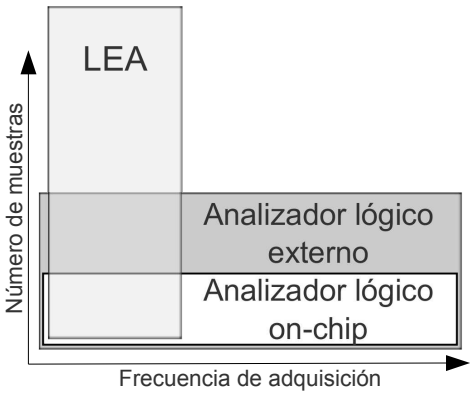


Figura 1: Campo de aplicación de las herramientas de verificación analizadas.

No obstante, existen otro tipo de aplicaciones en los que para realizar una correcta verificación del sistema se hace necesario el análisis de un gran número de eventos distribuidos sobre un largo periodo de tiempo, lo cual requiere a su vez una gran cantidad de recursos de almacenamiento. Un caso concreto de este tipo de diseños es un sistema de sincronización de red previamente desarrollado donde los eventos sobre los que se realiza el test de fiabilidad tienen una separación del orden de los segundos y deben ser analizados durante días, semanas o incluso meses. Así, al no adaptarse las herramientas existentes a las características de los sistemas que se están pretendiendo verificar, esto nos obliga a desarrollar una nueva herramienta que permita abordar la verificación de este tipo de diseños. La alternativa de veri-

ficación *on-chip* propuesta se ha denominado Analizador de Eventos Lógicos o *Logical Event Analyzer* (LEA). Este nombre se debe a que esta herramienta no va a realizar una verificación del diseño ciclo a ciclo de reloj sino que estará controlada por eventos lógicos espaciados en el tiempo; cada vez que se produzca uno de estos eventos se procederá a adquirir el estado del sistema en ese instante y a continuación se transmitirá la información adquirida a través de un bus estándar (por ejemplo RS232), no siendo necesario el almacenamiento de las muestras dentro del dispositivo programable.

En definitiva, con el LEA se pretende cubrir uno de los huecos dejados por los analizadores lógicos existentes tal y como se muestra en la Fig. 1. En concreto nos referimos a la limitación de los analizadores, tanto externos como *on-chip*, de poder capturar únicamente un número finito y no muy elevado de muestras. En efecto, el número de muestras que se pueden adquirir dependerá de la cantidad de recursos de memoria que deje libre el sistema bajo pruebas. Con esta nueva herramienta se ha conseguido que los recursos de la FPGA empleados para realizar la depuración del diseño sean independientes del número de muestras de forma que el sistema pueda ser testado indefinidamente, sólo limitado por los recursos externos.

## 3. Plataforma de verificación para sistemas de sincronización

La plataforma de verificación que se presenta en este apartado ha sido desarrollada para adaptarse a cualquier sistema de sincronización u otros diseños con características similares. Como se ha comentado anteriormente esta plataforma debe tener la capacidad de adquirir un número elevado de eventos del sistema lo cual implica que las muestras capturadas no pueden ser almacenadas empleando recursos propios de la FPGA. De esta forma, la solución que se plantea consiste en transmitir los datos capturados siendo una aplicación *software* externa la que se encargue de procesar y almacenar la información enviada. Esto resulta factible debido a que los eventos a partir de los que se van a tomar muestras del sistema están espaciados en el tiempo, pudiendo oscilar el rango de estos eventos desde unos pocos segundos a muchos minutos.

La arquitectura del analizador propuesto se basa en el microprocesador PicoBlaze de Xilinx [11], aunque también podrían emplearse otros microprocesador de similares características. En la Fig. 2 se presenta el diagrama de bloques del analizador diseñado. Como se observa en esta figura una serie de puertos de entrada de PicoBlaze se reservan para las señales de disparo, reloj y control de la comunicación; los restantes puertos se utilizan para capturar las señales de datos. La versión de PicoBlaze empleada en el LEA permite el direccionamiento de 256 puertos de 8 bits. Así, usando un único módulo de PicoBlaze y dedicando $N$ puertos a las señales de disparo, reloj y control, este analizador permite capturar $(256 - N) * 8$ señales de un bit. Si fuera necesario capturar más señales una solución sencilla consiste en añadir un registro externo de $n$ bits que amplíe la señal de selección de puerto *port_ id*. Con esta alternativa, el número máximo de señales que se pueden muestrear sería $(256-N)*2^n*8$. Sin embargo, como se va a exponer a continuación, el número de señales de datos que se pueden adquirir está también limitado por la frecuencia a la cual se puede transmitir la información capturada fuera del chip.

PicoBlaze emplea uno de sus puertos de salida para comunicarse con la UART encargada de transmitir los datos capturados vía serie. Esta UART permite configurar su *baud rate* mediante la señal *conf_ baud* y además dispone de una señal *half_ full_ buffer* que indica que la FIFO está medio llena. PicoBlaze usará la señal *half_ full_ buffer* para controlar la transmisión de los datos a la UART. Asumiendo la siguiente configuración para la UART: formato fijo del dato de envío "8N1"(8 bits de datos, 1 bit de stop y sin paridad), comunicación a través de los pines RX/TX (sin usar otros adicionales) y control de flujo desactivado, el máximo *bit rate* (bps) que se puede obtener se calcula de acuerdo a (1). En la Tabla 1 se muestran los *bit rates* calculados para algunos

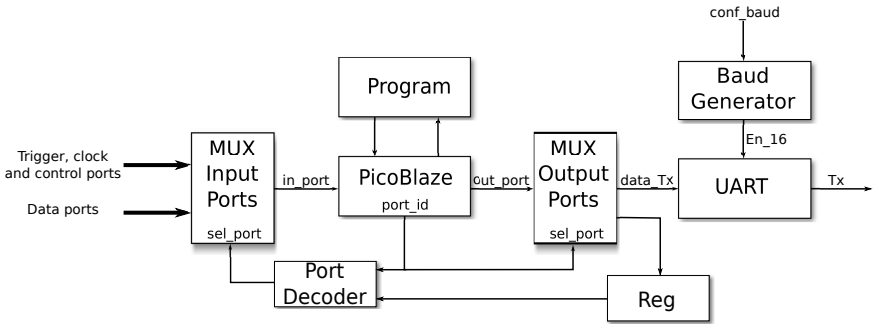


Figura 2: Arquitectura del Analizador de Eventos Lógicos.

Cuadro 1: Bit rates calculados para algunos baud rate típicos.

| Baud rate | Bit rate (bps) |
|---|---|
| 4800 | 3840 |
| 9600 | 7680 |
| 19200 | 15360 |
| 38400 | 30720 |
| 57600 | 46080 |
| 115200 | 92160 |

*baud rates* típicos.

$$bitrate = \frac{baudrate * 8}{10} \quad (1)$$

Finalmente, el módulo de programa se corresponde con el programa que ejecuta PicoBlaze. Este programa realiza las siguientes tareas:

1. Verificación de la condición de disparo. PicoBlaze lee los puertos asignados a las señales de disparo y aplica la función lógica configurada. En caso de verificarse la adquisición de los datos comienza. En este punto es importante comentar que PicoBlaze dispone de un amplio conjunto de instrucciones que permite configurar muchas opciones de disparo.

2. Adquisición de los datos de acuerdo a la señal de reloj. La señal de reloj se corresponde con algún evento del sistema diseñado, de forma que siempre que este evento ocurre se procede a leer los datos conectados a los puertos de entrada de PicoBlaze.

3. Procesamiento y envío de los datos a la UART. Este procesado consiste en calcular un *checksum* de los datos transmitidos de forma que el receptor pueda verificar la correcta recepción de la información. Una vez procesados, los datos serán enviados a la UART.

4. Control de la comunicación con la UART. Periódicamente, el microprocesador comprobará el estado de la señal *half_full_buffer*. Si está activa, PicoBlaze esperará hasta que la FIFO esté medio vacía antes de enviar más datos a la UART.

## 4. Ejemplo de aplicación y resultados

En este apartado se describe la aplicación del LEA para la verificación *on-chip* de un cliente y un servidor SNTP implementado completamente en *hardware*. El protocolo SNTP es una versión simplificada del *Network Time Protocol* (NTP) [12] usado comúnmente para sincronizar los relojes de sistemas de ordenadores sobre redes de datos tales como *Internet*. La operación de este protocolo consiste en enviar peticiones de tiempo a un servidor sincronizado con una fuente de tiempo precisa como por ejemplo un receptor de GPS en intervalos

periódicos que puede variar desde unos pocos segundos a muchos minutos. Cuando se recibe la respuesta del servidor, el cliente usa un conjunto de marcas de tiempo para calcular el retraso de propagación y el desplazamiento de tiempo entre los relojes del cliente y del servidor. Finalmente, el cliente puede ajustar su reloj local en base a estos parámetros calculados.

En un escenario típico, el cliente estará sincronizado de forma muy precisa sólo después de muchos ciclos de petición y respuesta. Desde que el tiempo entre peticiones puede variar desde pocos segundos a muchos minutos el aspecto más importante en la verificación de este tipo de sistemas no es la velocidad a la que se capturan las muestras sino la adquisición de un gran número de eventos del sistema, cubriendo por tanto un amplio intervalo de tiempo.

Las herramientas de verificación *on-chip* como *ChipScope Pro* presentan características como altas frecuencias de muestreo lo que permite la verificación de buses y sistema de alta velocidad, pero muestran ciertas limitaciones en cuanto al número máximo de capturas que se pueden obtener del sistema. Esto se debe fundamentalmente a que emplean recursos internos de la FPGA para almacenar las muestras y a que algunos de estos recursos, dependiendo del tipo de dispositivo programable usado, suelen ser limitados. En las Figuras 3, 4, and 5 se muestran el número de *Look-Up Table* (LUT), *Flip Flop* (FF) y *Block RAM* (BRAM) empleados por *ChipScope Pro* en función del número de señales y del número de muestras. En las Figuras 3 y 4 se observa como el uso de LUT y FF depende principalmente del número de señales. Sin embargo, de la Fig. 5 se desprende que el principal problema cuando se realiza la verificación *on-chip* empleando este tipo de herramientas reside en que el número de BRAM utilizadas es directamente proporcional al producto del número de señales y del número de muestras que se quieren capturar lo cual se debe a que éste es el recurso empleado para almacenar los datos capturados. De esta forma, considerando que las BRAM son el recurso más limitado este tipo de verificación es inviable si el objetivo es capturar un gran

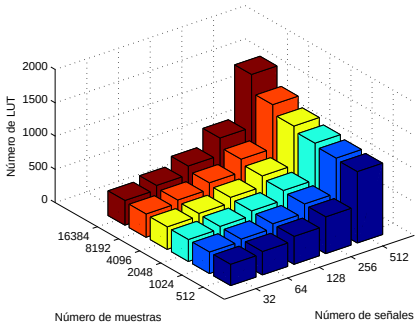


Figura 3: Número de LUTs empleados en función del número de señales y de muestras usando ChipScope Pro.

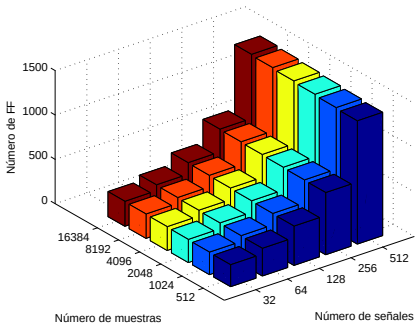


Figura 4: Número de FFs empleados en función del número de señales y de muestras usando ChipScope Pro.

número de muestras incluso para un número reducido de señales a capturar.

Para el caso bajo discusión, el cliente y el servidor SNTP se han implementado sobre un dispositivo FPGA Spartan-3E modelo xc3s500e. Estas FPGA tienen un total de 20 BRAM y cada bloque dedica 16 Kbit al almacenamiento de datos. Para cada diseño se han muestreado un total de 256 señales: marcas de tiempo (parte menos significativa), desplazamiento, retraso de propagación y parámetros de ajuste del reloj local. Con esta configuración, en la Fig. 6 se muestra el porcentaje de recursos necesarios (LUT, FF y BRAM) para

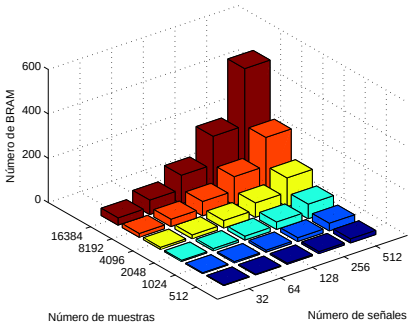


Figura 5: Número de BRAMs empleados en función del número de señales y de muestras usando ChipScope Pro.

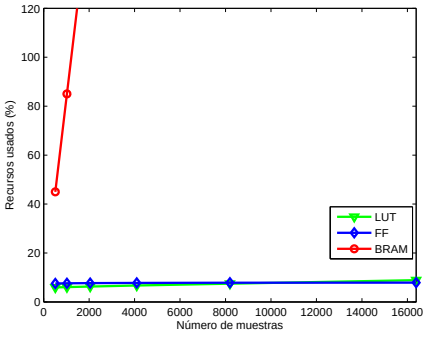


Figura 6: Recursos empleados en función del número de muestras para 256 señales usando ChipScope Pro.

verificar el sistema en función del número de muestras. Como se observa en esta figura, el consumo de LUT y FF no es crítico suponiendo respectivamente el 9 % y 8 % del total de recursos en el peor de los casos. Sin embargo, se observa un uso excesivo de BRAM incluso para un número pequeño de muestras. En este caso, sólo se pueden capturar un máximo de 1024 muestras lo cual resulta insuficiente para el tipo de sistema que se pretende verificar.

El Analizador de Eventos Lógicos desarrollado tiene la ventaja de que no almacena los datos en BRAM sino que los transmite vía serie. Por consiguiente, el número de BRAM em-

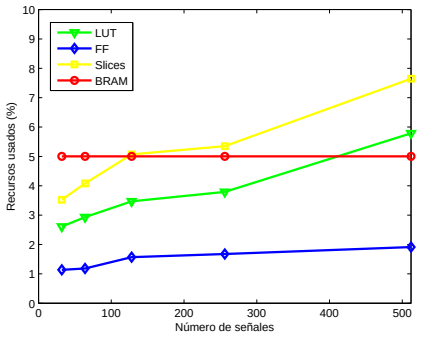


Figura 7: Recursos empleados en función del número de señales usando el LEA.

pleadas para verificar el sistema no depende del número de señales o el número de muestras de forma que el sistema puede testarse de forma indefinida, sólo limitado por los recursos externos. En cuanto al resto de recursos de la FPGA, éstos pasan a ser únicamente dependientes del número de señales (Fig. 7). Para el mismo escenario presentado para *ChipScope Pro* (muestreo de 256 señales), el porcentaje de uso de LUT, FF, Slices y BRAM ha sido del 3,79 %, 1,68 %, 5,35 % y 5 % respectivamente. Destaca la utilización de una única BRAM empleada para almacenar el programa que ejecuta PicoBlaze.

## 5. Conclusiones

En este trabajo se describe una plataforma de verificación *on-chip* basada en Picoblaze que se ha empleado para depurar de forma indefinida sistemas con eventos distribuidos sobre un largo periodo de tiempo. Como se ha discutido previamente, la ausencia de herramientas para verificar estos sistemas particulares fuerza a los diseñadores a crear *cores* de verificación a medida en función de las características de cada uno. En este sentido, la flexibilidad proporcionada por PicoBlaze permite la generación casi automática de estos componentes *hardware* de verificación, permitiendo su reutilización en otro tipo de sistemas.

Además, la plataforma propuesta se ha com-

parado con una de las herramientas de verificación *on-chip* más conocidas y utilizadas en la actualidad: *ChipScope Pro*. De los resultados obtenidos se puede deducir que *ChipScope Pro* no permite una correcta verificación de los sistemas que se están pretendiendo depurar. Esto se debe a que *ChipScope Pro* emplea recursos internos de la FGPA para almacenar los datos capturados y a que éstos resultan insuficientes incluso para un número reducido de muestras a adquirir. Por el contrario, el LEA al transmitir la información vía serie no necesita almacenar internamente ningún tipo de información lo que permite verificar el sistema indefinidamente.

## Acknowledgment

Este trabajo ha sido parcialmente financiado por el Ministerio de Educación y Cultura del Gobierno Español a través de los proyectos TEC2007-61802/MIC (HIPER) y PROFIT-MITC SEPIC TSI-020100-2008-258.

## Referencias

[1] K. Arshak, E. Jafer, and C. Ibala, "Testing FPGA based digital system using XILINX ChipScopeTM logic analyzer," in *29th International Spring Seminar on Electronics Technology (ISSE)*, St. Marienthal (Germany), May 2006, pp. 355–360.

[2] Xilinx, *ChipScope Pro 11.1 Software and Cores User Guide*. Xilinx, Inc., Apr. 2009.

[3] Altera, *Design Debugging Using the SignalTap II Embedded Logic Analyzer*. Altera Corporation, Nov. 2009.

[4] J. Viejo, J. Juan, M. J. Bellido, E. Ostua, A. Millan, P. Ruiz-de-Clavijo, A. Muñoz, D. Guerrero, "Design and implementation of a SNTP client on FPGA," in *Proc. 2008 IEEE International Symposium on Industrial Electronics (ISIE)*, Cambridge (United Kingdom), Jun. 2008, pp. 1971–1975.

[5] P. S. Graham, "Logical Hardware Debuggers for FPGA-Based Systems," Bringham Young University, Department of Electrical and Computer Engineering, PhD Dissertation, 2001.

[6] N. Ohba and K. Takano, "Hardware debugging method based on signal transitions and transactions," in *IEEE Proc Asia and South Pacific conf on Design Automation*, Yokohama (Japan), Jan. 2006, pp. 454–459.

[7] T. Wheeler, P. Graham, B. Nelson, and B. Hutchings, "Using Design-Level Scan to Improve Design Observability and Controllability for Functional Verification of FPGAs," in *2001 Proceedings International Conference on Field-Programmable Logic and Applications (FPL)*, Belfast, Northern Ireland (UK), Aug. 2001, pp. 483–492.

[8] L. Ehrenpreis, P. Ellervee, and K. Tammemae, "Open Source On-Chip Logic Analyzer for FPGAs," in *2006 International Baltic Electronics Conference*, Tallinn (Estonia), Oct. 2006, pp. 1–4.

[9] G. Knittel, S. Mayer, and C. Rothlaender, "Integrating Logic Analyzer Functionality into VHDL Designs," in *2008 International Conference on Reconfigurable Computing and FPGAs*, Cancun (Mexico), Dec. 2008, pp. 127–132.

[10] T.-Y. Lee, Y.-H. Fan, S.-C. Yen, C.-C. Tsai, and R.-S. Hsiao, "An Integrated Functional Verification Tool for FPGA Systems," in *International Conference on Innovative Computing, Information and Control*, Kumamoto (Japan), Sep. 2007, p. 203.

[11] K. Chapman, *PicoBlaze 8-Bit Embedded Microcontroller User Guide for Spartan-3, Virtex-II and Virtex-II PRO FPGAs*. Xilinx, Inc., Nov. 2005.

[12] D. L. Mills, "Network Time Protocol (Version 3) Specification, Implementation and Analysis," RFC 1305 (Draft Standard), Mar. 1992.