Diseño e implementación de un sistema de verificación funcional utilizando la técnica de aceleración por hardware para optimizar el diseño de sistemas electrónicos digitales

Design and Implementation of an Open Source Functional Verification System Using the Hardware Acceleration Technique to Optimize the Digital Systems Design

Recibido: agosto de 2012
Arbitrado: septiembre de 2012

Johan Sebastián Eslava Garzón*, Héctor Andrés González Díaz**

Resumen

Este artículo describe el diseño y la implementación de un sistema de verificación funcional basado en la técnica de la aceleración de hardware, «Aceleración de la simulación», utilizando herramientas libres (hardware y software libre) que acercan la solución a desarrolladores pequeños y a espacios académicos que incentivan el desarrollo de soluciones electrónicas locales. Este sistema permite realizar la verificación funcional de sistemas digitales, ejecutando el lenguaje de descripción de hardware (HDL) directamente en un dispositivo emulador (hardware) y visualizando su comportamiento en una estación de trabajo controlada por el usuario o desarrollador. El tiempo de verificación, y por ende el tiempo de producción de un diseño digital de complejidad considerable, se disminuye con el uso de la verificación por emulación de hardware.

Palabras clave

Aceleración de la simulación, emulación de hardware, hardware libre, sistemas digitales, verificación funcional.

Abstract

This paper describes the implementation and design of a functional verification system based on the hardware acceleration technique, «Simulation Acceleration». Utilizing open source tools (Open Software and Hardware) brings the solution closer to the small developers and to the academic spaces, which motivates the development of local electronic

* Ingeniero Electricista, Magíster y Doctor en Ingeniería Eléctrica. Profesor Asistente, Universidad Nacional de Colombia, e-mail: jseslavag@unal.edu.co

** B.Sc. Ingeniería Electrónica, Universidad Nacional de Colombia. E-mail hagonzalezd@unal.edu.co

solutions. This system makes it posible to realize the functional verification of digital systems, executing the hardware description language (HDL) directly into an emulating device (Hardware) and visualizing its behavior in a workstation controlled by the user or developer. The verification time, and the time to market of a complex digital design is decreasing with the use of hardware emulation verification.

Keywords

Digital systems, functional verification, hardware emulation, open hardware, simulation acceleration.

I. Introducción

Cuando se habla de desarrollos tecnológicos, se debe hacer inmediata referencia a la electrónica presente en cada uno de los diversos desarrollos de hoy en día. La evolución de tales sistemas electrónicos, los cuales en su mayoría son digitales, es debida a los avances obtenidos principalmente en dos aspectos: el primero, los procesos de fabricación de circuitos integrados (que permiten integrar más transistores, con menor consumo de energía y mayor velocidad de operación) y el segundo, los avances en las metodologías y herramientas de diseño de los mismos. Sin embargo, este segundo factor no ha evolucionado con la misma velocidad que los procesos de fabricación. Esta situación ha ocasionado lo que se denomina el «gap de productividad» [14]. Esta situación indica que las metodologías y herramientas actuales no permiten aprovechar todo el poder de integración ofrecido por los procesos de fabricación.

Para solucionar esta problemática, uno de los enfoques es optimizar los procesos del diseño que mas exijan recursos, como la verificación funcional. La misión de esta tarea es comprobar que el diseño está de acuerdo a la especificación, es decir completo (realiza lo que se estableció) y correcto (lo realiza como se estableció). A través del tiempo la verificación de sistemas digitales se ha ganado su lugar en la producción de dispositivos electrónicos. Cada vez resulta más importante implementar procedimientos que garanticen la calidad de los dispositivos que se lanzan al mercado, puesto que es más fácil corregir defectos antes que después. Imagine que se lanza al mercado un nuevo microprocesador que posee un elemento de hardware (IP core) que presenta fallas en la actividad para la cual fue diseñado, esto podría implicar retirar por completo estos dispositivos del mercado, o peor aún si el microprocesador está destinado a una aplicación crítica de control en donde una falla podría implicar pérdidas millonarias o la pérdida de vidas humanas.

Situaciones como estas han ocasionado que las compañías que trabajan con dispositivos electrónicos dediquen un gran porcentaje de los recursos (alrededor del 70%), en realizar procesos intensos de verificación que le otorguen niveles altos de confiabilidad a sus productos. Existen diversas técnicas para realizar verificación de sistemas digitales, entre las cuales se tiene la técnica de verificación por simulación, en donde existen unos procesos estándar que se deben seguir para verificar el producto.

La técnica de verificación por simulación es muy popular dentro de los procesos de las compañías fabricantes de SoCs (System On Chip). Esta técnica otorga una gran flexibilidad a la hora de realizar la verificación funcional, puesto que el entorno de verificación ha sido implementado en software. Pero posee un gran problema con su tiempo de prueba. Generalmente, los IP cores que se someten a verificación por simulación, son bastante grandes (compuestos de millones de puertas lógicas), para simular un sistema de este porte es necesario disponer de ambientes de simulación robustos y de gran capacidad de procesamiento, en otras palabras, las simulaciones son muy demoradas. La simulación de un controlador cualquiera puede tardar semanas. Para ser competitivo en la industria de la electrónica se debe producir rápido, puesto que el tiempo de innovación en electrónica es muy pequeño, así como los tiempos de migración de un producto antiguo a otro relativamente nuevo.

Estos inconvenientes han hecho que se exploren alternativas para reducir el tiempo de la verificación (Tverif). Una es la verificación utilizando aceleración por hardware (HW), en la que se utiliza un hardware programable para implementar el IP core, permitiendo que todas o parte de las tareas sean ejecutadas directamente en HW, antes de la fabricación.

La alternativa de la Aceleración por hardware es bastante prometedora en la industria por su corto Tverif, aunque posee limitaciones del tamaño del hardware que podría ser verificado, pues un gran tamaño requiere físicamente más capacidad en el hardware programable utilizado. Fabricar una plataforma para cada desarrollo es poco práctico (Prototipado).

Es por esto que surge la necesidad de diseñar plataformas dedicadas a este tipo de procesos, que intenten abarcar la mayoría de módulos, sin tener que realizar una fabricación extra en cada proceso de verificación. En la actualidad, es posible encontrar soluciones comerciales a estas necesidades, cuyos costos varían dependiendo del nivel de complejidad que posean, siendo inaccesible para pequeños desarrolladores o para la investigación académica dentro de la sociedad colombiana.

Actualmente en el grupo de microelectrónica de la Universidad Nacional de Colombia (GMUN) se están realizando trabajos importantes [11] que están llevando a un pensamiento tecnológicamente revolucionario de los nuevos profesionales de dicha institución. Se efectúan proyectos que involucran el uso de aplicaciones con procesadores de 32 bits, implementando tareas hardware (HW) en FPGAs (Field Programmable Gate Array), involucrando sistemas operativos, todo implementado en placas de circuito impreso con componentes de montaje superficial. Dado que se tiene un auge en la producción de sistemas digitales en la UN, y que se está expandiendo a otras universidades importantes del país, se incursionó por generar una solución igualmente seria, de alta calidad y complejidad en el campo de verificación de sistemas digitales, garantizando la correcta funcionalidad de los diseños producidos, evitando costos adicionales después de la fecha de la implementación en el ambiente real del dispositivo, y contribuyendo a disminuir el gap de productividad que se enunció.

Es por esto que se diseñó e implementó un sistema usando HW y SW libre que permita realizar verificación funcional en sistemas digitales aprovechando las velocidades de ejecución del hardware y dando una primera gran aproximación al diseño de soluciones de verificación de sistemas digitales de características libres.

II. Marco teórico

A continuación se enuncian algunos conceptos y definiciones que ayudarán a contextualizar el contenido del artículo.

2.1 Verificación funcional

La verificación funcional de sistemas digitales es el proceso contrario al proceso de diseño de sistemas digitales. En el diseño se parte de una especificación y se quiere llegar a una implementación. En la verificación funcional se parte de una implementación y se quiere llegar a la especificación inicial, consiguiendo la validación de la implementación que el diseñador realizó. Existen varias alternativas para realizar la verificación funcional de diseños digitales, tales como la verificación formal, la simulación y la aceleración de hardware.

2.1.1 Verificación formal: está basada en algoritmos matemáticos que verifican la funcionalidad del sistema, este modelo realiza un seguimiento de la totalidad de los estados del diseño a través de preposiciones matemáticas. Esta técnica es bastante exhaustiva y por tanto muy precisa, pero implica un alto consumo de recursos computacionales.

2.1.2 Simulación: la Verificación por Simulación está basada en la realización de un modelo usando lenguajes de descripción de hardware

(HDL), esta técnica es muy flexible y ha sido ampliamente usada, sin embargo, cuando el modelo a implementar es demasiado grande o demasiado complejo, su esfuerzo computacional (requerido para simular) toma mucho tiempo. Esto afecta el tiempo de producción y directamente el tiempo en que sale el diseño al mercado (Time to market).

La verificación por simulación está basada en la realización de un modelo usando lenguajes de descripción de hardware (HDL), esta técnica es muy flexible y ha sido ampliamente usada, sin embargo, cuando el modelo a implementar es demasiado grande o demasiado complejo, su esfuerzo computacional (requerido para simular) toma mucho tiempo. Esto afecta el tiempo de producción y directamente el tiempo en que sale el diseño al mercado (Time to market).

2.1.3 **Aceleración de Hardware:** la verificación por aceleración de hardware está basada en una implementación de hardware del diseño bajo verificación (DUV), esto no es un método flexible pero es muy rápido para ejecutar IP cores grandes. Esta técnica reduce el tiempo de verificación requerido en grandes diseños [9] debido a que implementa en hardware el modelo que es simulado en la verificación por simulación.

Existen diversas técnicas para realizar verificación por aceleración de hardware: Targetless Logic Emulation (TLE), In-Circuit Emulation (ICE) y la Aceleración de la Simulación (Simulation Acceleration) [7]. La técnica TLE es usada para verificar un diseño con condiciones aisladas, esto significa que el IP core bajo verificación es ejecutado sin ninguna intervención externa, la intervención externa son los estímulos, los cuales no provienen de adentro del emulador. La técnica ICE es similar a la técnica TLE pero con la diferencia de que en ICE los estímulos provienen del «mundo real». Esto significa que otra placa (tarjeta) u otra entidad del mundo real proveen el conjunto de estímulos (testbench) al diseño. La aceleración de la simulación es una técnica donde el diseño es implementado en un emulador de hardware y el resto del ambiente de verificación es implementado en una estación de trabajo externa que controla el flujo de verificación. La visualización y los estímulos suministrados son suministrados por la estación de trabajo.

En este artículo se desarrollará la técnica de la simulación de la aceleración.

- **Ambiente de verificación**

La elaboración de un ambiente de verificación para emulación de hardware (para cualquiera de las alternativas) es una tarea compleja. Es necesario establecer una comunicación coherente entre los diferentes elementos durante la ejecución, es muy importante garantizar que los resultados obtenidos sean correctos al final de la emulación. Los emuladores basados en FPGA tienen observabilidad limitada porque para visualizar señales internas del diseño es necesario rutear las señales a un pin externo, lo cual requiere un esfuerzo extra e implica que el diseñador necesita conocer detalles técnicos del emulador para depurar el DUV. Otra tarea difícil es la construcción de las herramientas software que deben ser asociadas con la ejecución del emulador (Con la arquitectura del emulador).

III. Desarrollo

La información está relacionada con el diseño del sistema de verificación de sistemas digitales.

3.1 Emulador de hardware

3.1.1 **Definición:** un emulador es un sistema que imita una cierta plataforma computacional o programa de computadora en otra plataforma u otro programa. Esto causa que el comportamiento del segundo sistema sea casi el mismo que el del primero [4]. El emulador de hardware para verificación debe tener algunas especificaciones básicas para implementar y verificar de forma adecuada el diseño. Este debe incluir un elemento conocido como «logic box» que es

donde el diseño es implementado, un sistema de ruteo y una interfaz de circuito [6].

Para emular el diseño, este tiene que ser implementado en la «logic box» por un computador externo, el cual es conocido como estación de trabajo o «host». La caja lógica o «logic box» puede ser una FPGA o un conjunto de FPGA donde el diseño es particionado para obtener una mejor ejecución paralela consiguiendo velocidades más grandes. El sistema de ruteo es para comunicar las unidades de la caja lógica entre ellas y para comunicarlas con el resto del ambiente. Este además comunica la caja lógica con la interfaz de circuito que es una unidad usada para traducir todas las señales que provienen de la caja lógica. A continuación podemos ver la estructura de un emulador basado en múltiples FPGA.

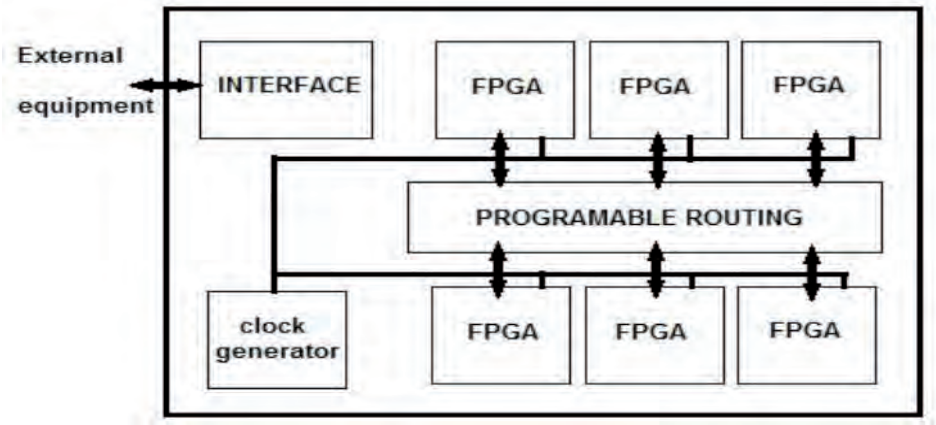

Fig. 1. Estructura de un Emulador de Hardware.

Las características del emulador varían ligeramente dependiendo de la técnica de aceleración de hardware que se utilice.

3.1.2 Estado del arte: actualmente existen diversas plataformas para realizar verificación usando técnicas de aceleración de hardware. Estas plataformas son suministradas por alguna marca que ofrece servicios de verificación como es el caso de la línea Palladium de Cadence [15], o pueden ser desarrolladas por compañías para suplir sus necesidades internas de verificación de sus diseños digitales como es el caso de Cambridge Consultants con su plataforma XEMU emulator [14]. Los Emuladores comerciales van desde plataformas sencillas como el XEMU emulator que contiene un procesador XAP y una FPGA Spartan 3, pasando por plataformas como ZEBU (ZEro BUgs) de la compañía EVE, las cuales poseen diversas FPGA que son usadas para dividir y mapear las partes del DUV en las FPGA que contiene, haciendo que la velocidad de comunicación y la observabilidad del sistema se incremente, llegando hasta plataformas computacionales que reúnen las tres técnicas de aceleración de hardware en un ambiente híbrido como el Palladium XP de Cadence. El palladium XP es usado por compañías como NVIDIA para la verificación de sus tarjetas de vídeo. Estas plataformas no están al alcance de desarrolladores pequeños o al alcance de espacios académicos cuya intención de adquisición es netamente educativa.

3.1.3 Emulador utilizado (Plataforma SIE): la plataforma SIE es un proyecto de hardware libre desarrollado por el profesor Carlos Camargo de la Universidad Nacional de Colombia [8]. Enlistamos algunas de las características de SIE:

- → Procesador JZ4725 Xburst 400 MHZ
- → FPGA Spartan 3e XC3S500E-VQ100
- → Memoria NAND 2 GB

- SDRAM 64 MB
- USB Device
- Micro SD
- UART

En el ambiente a implementar, se plantea que el procesador jz4725 se utilice para controlar la ejecución dentro del emulador y el intercambio con la estación de trabajo. La memoria nand de 2 GB es utilizada para almacenar los estímulos temporalmente mientras son procesados desde y hacia la FPGA. También se usa para alojar la aplicación de espacio de usuario. La FPGA naturalmente es usada para implementar el diseño bajo verificación (DUV).

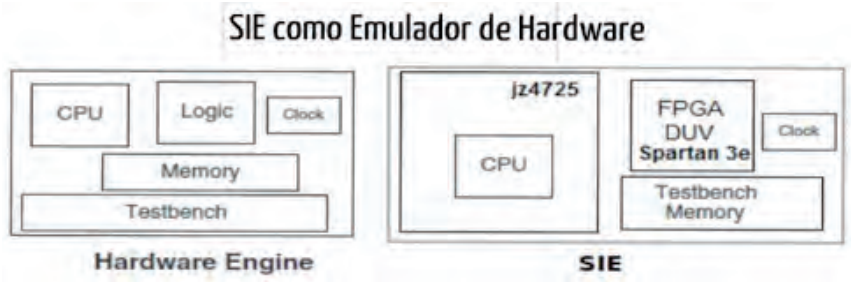

Fig. 2. SIE como emulador de hardware.

La principal razón por la que se escogió SIE es porque se estaba buscando elaborar un trabajo que fuera de licencia abierta y sin restricciones de distribución académica. SIE cumple con estos requisitos por ser una plataforma de hardware libre, y además viene consigo con un historial de que fue diseñada en la Universidad Nacional de Colombia, y que cuenta con un equipo de trabajo a nivel mundial a través de una wiki, en donde es posible resolver dudas y recibir contribuciones de distintas fuentes, esto es un gran valor adicional que se tiene por trabajar con hardware y software libre. Además las características de SIE dan un respaldo de rendimiento por tratarse de una plataforma embebida con un procesador de 32 bits, así como con otras características físicas que brindaban una infinidad de posibilidades a la hora de intentar implementar una técnica de verificación por emulación de hardware en esta. Dado que SIE contaba con un sistema operativo con kernel de Linux, era posible igualmente implementar software que soportara las actividades de verificación que se iban a contemplar durante la ejecución del proyecto.

3.2 Aceleración de la simulación

Es una de las tres técnicas existentes para realizar verificación funcional usando aceleración de hardware. Esta técnica es la técnica que presenta un menor rendimiento temporal de las tres, puesto que en esta la ejecución del emulador posee una dependencia con la estación de trabajo, el emulador de hardware no puede ejecutarse a su mayor velocidad puesto que está regulado por el reloj generado por la estación de trabajo. El emulador intercambia información constantemente con el ambiente de verificación. El ambiente de verificación está implementado en la estación de trabajo y el diseño bajo verificación se mapea dentro del emulador.

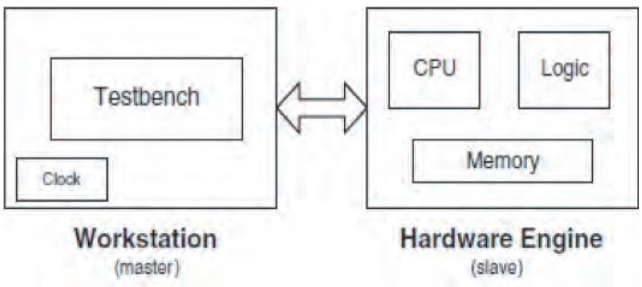

Fig. 3. Esquema de la aceleración de la simulación.

Como se ve en la figura 3, en la estación de trabajo se generan los estímulos y dentro del emulador se puede ver claramente que se tiene una unidad central de procesos que regula la comunicación entre la estación de trabajo y el diseño bajo verificación.

A diferencia de la técnica de aceleración de hardware «Targetless logic emulation» en la cual el emulador se ejecuta en condiciones aisladas, en la aceleración de la simulación se deben tener en cuenta aspectos de comunicación con la estación de trabajo durante la ejecución del diseño bajo verificación (DUV) lo cual afecta su desempeño.

En la estación de trabajo es posible implementar un ambiente de verificación con las facilidades de SystemC y su adicional SystemC Verification (SCV). Esto permite una adecuada observabilidad de la ejecución del emulador, manteniendo un control similar al control que se tiene cuando se ejecuta una simulación en software, pero con la diferencia de que en este caso el núcleo (core) de la simulación se ejecuta en una plataforma Hardware.

3.2.1. Características de aceleración de la simulación: la técnica que se escogió es la técnica de la Aceleración de la Simulación, debido a que presenta una observabilidad interesante y comparable con la verificación que normalmente se utiliza siempre, la verificación por simulación. Las principales características de esta técnica son:

- → Permite obtener cierta observabilidad del sistema, pues todo el monitoreo de la ejecución se hace desde una estación de trabajo controlada por el usuario.
- → Aunque no permite velocidades tan altas como las otras dos técnicas (ICE y TLE), permite que el usuario sienta que al trabajar con el ambiente de verificación sobre la estación de trabajo, pareciera como si estuviera trabajando con una simulación, pero en realidad el diseño bajo verificación (DUV) está siendo ejecutado en HW, obteniendo ventajas de la emulación y de la simulación.
- → Debido a que la estación de trabajo está en constante comunicación con el emulador, existen retrasos inducidos por el sistema de comunicación implementado entre la estación de trabajo y el emulador.
- → En el emulador no son generados los estímulos que se requieren para la verificación del DUV, estos son generados en la estación de trabajo y son transferidos al emulador durante la ejecución.

Hasta ahora se mencionó qué técnica de emulación de hardware se iba a utilizar, la aceleración de la simulación, y qué emulador de hardware iba a utilizarse para el desarrollo del sistema de verificación de sistemas digitales. En este punto, teniendo el emulador y la técnica deben establecerse las características del ambiente de verificación.

3.3 Ambiente de verificación

Un ambiente básico de verificación consta de un bloque generador de estímulos, que representa la entidad que suministra los patrones de prueba al diseño bajo verificación (duv), un comparador (checker) que es quien registra la actividad del duv, un modelo de referencia que posee el comportamiento ideal del duv, suele estar programado en un lenguaje de alto nivel, y finalmente el módulo que contiene al diseño bajo verificación. La interconexión de estos módulos es como se presenta en la figura 4, y es a este ambiente de verificación al que se quería llegar, este es el sistema que permite realizar las diversas pruebas sobre el duv.

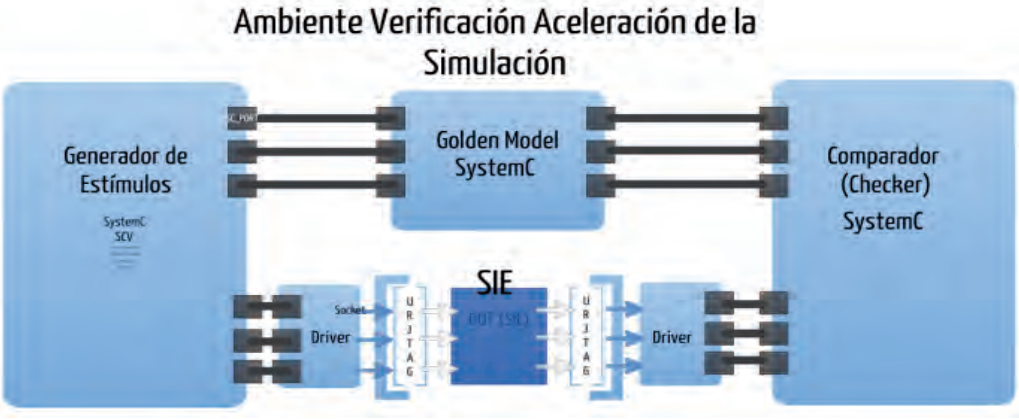

Figura 4. Ambiente de verificación objetivo.

Gran parte de este ambiente está implementado en SystemC, que es una librería que le otorga a C++ concurrencia y otra serie de funciones para describir hardware a un alto nivel de abstracción, a nivel de sistema. Los módulos de SystemC están sobre la estación de trabajo, y se puede apreciar en la figura 4, la forma en que debe ir conectada la plataforma SIE, se aprecia que está rodeada de dos drivers o traductores, que permiten la interpretación de las señales que se obtienen de SIE y que permiten la escritura sobre las señales en SIE. El bloque generador de estímulos utiliza funciones de SystemC Verification (SCV), que es una librería adicional de SystemC que permite generar datos aleatorios, permite realizar generaciones de estímulos restringidos con gran flexibilidad. Para definir por completo el ambiente de verificación, se debe definir cómo se conecta el diseño bajo verificación con el exterior de SIE y a su vez como se conecta SIE con la estación de trabajo que contiene el resto del ambiente de verificación (Los modelos azules de la figura 4 que están en SystemC y en SCV).

3.3.1 Comunicación PROCESADOR+FPGA: en el interior de la SIE como ya se describió se cuenta con una serie de características que deben ser utilizadas para la respectiva implementación del duv y su respectiva comunicación con el exterior de SIE. El diagrama de bloques de SIE asociado a su papel de emulador se encuentra en la figura 2. Para la comunicación entre el procesador y la FPGA se tenían las alternativas del puerto serial, JTAG o una comunicación sobre un bus virtual que se genera entre el procesador y la FPGA, pero que no es más que un determinado número de posiciones en memoria que se comparten entre los dos dispositivos, sobre escribiendo en estas posiciones cada vez que se quiere establecer una comunicación entre el procesador y la FPGA.

3.3.1.1 JTAG: el estándar IEEE 1149.1 desarrollado por el grupo JTAG (Joint Test Action Group), define la arquitectura Boundary-Scan implementada para ICs digitales, la cual surgió en respuesta a la necesidad de realizar pruebas sobre circuitos impresos con una elevada densidad de componentes. La arquitectura Boundary-Scan permite la excitación y la captura del estado de los pines de entrada y salida de cada módulo a través de un registro de corrimiento. Este registro está compuesto por celdas denominadas BSC (Boundary-Scan Cell), las conectadas a los pines de entrada y salida de cada uno de los componentes. Dependiendo de la funcionalidad, la BSC está en capacidad de brindar información sobre el estado de la señal asociada a cada pin.

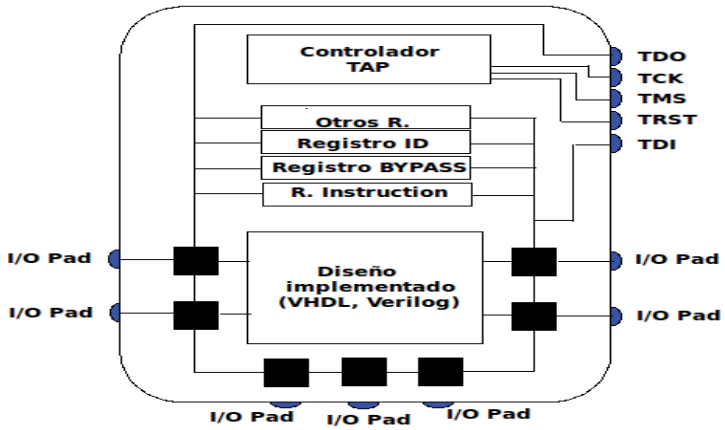

Fig. 5. Estructura de JTAG sobre un dispositivo.

Existen diversos comandos que indican la forma de operación de JTAG, pero la forma de operación que interesa ahora es la relacionada con la instrucción INTEST que permite suministrar patrones de prueba para testear la lógica interna de un dispositivo, capturando del mismo modo las respuestas provenientes de la lógica interna de este dispositivo. Los comandos para inicializar la instrucción INTEST en el módulo URJTAG son:

INTEST INSTRUCTION

```
instruction intest
shift ir
dr 10101001111010111010100111101011101...//vector de prueba de 588 posic...
shift dr
shift dr
dr out
```

Fig. 6. Comandos de la instrucción INTEST en URJTAG.

Como se ve en la figura 6, se le suministra al módulo URJTAG un vector de 588 posiciones, que es el patrón de pruebas que va a introducir en la FPGA. Es de 588 posiciones debido a que el BSR (Boundary Scan Register) de la FPGA de SIE posee 588 pines, y cada pin tiene una celda del BSR asociada.

Se escogió el puerto JTAG porque se observó la oportunidad de sincronizarlo con el espacio de usuario de una forma más rápida y eficiente que las otras dos alternativas. Es decir, a través de URJTAG era posible vincular el tránsito desde la FPGA y hacia esta para realizar la intervención del código implementado y de esta forma estimular y capturar las salidas del diseño bajo verificación usando INTEST. URJTAG es un módulo de software libre utilizado para controlar el protocolo JTAG. El repositorio SVN de URJTAG que se encontraba disponible tuvo que ser modificado para generar un reporte de los movimientos en la cadena BSR durante la instrucción INTEST, también para darle soporte al BSR de la FPGA de SIE que es de 588 posiciones y no de 100 como viene por defecto en URJTAG. Fue además necesario darle soporte a la FPGA de SIE, la spartan 3e porque no estaba contenida dentro de los dispositivos soportados por URJTAG. El procedimiento en detalle de los cambios y el soporte se encuentra en esta wiki [3]

En la figura 7 se aprecia la metodología adoptada con el protocolo JTAG que se utilizó.

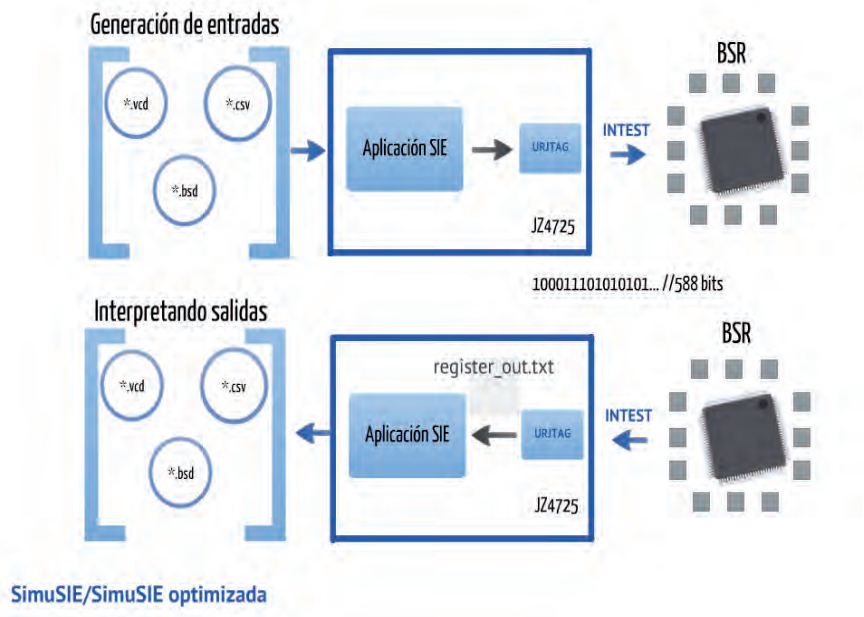

Fig. 7. Instrucción INTEST en URJTAG.

Se utilizan una serie de archivos para realizar la comunicación con el diseño implementado en la FPGA a través de la instrucción INTEST. Los estímulos deben ser entregados a la FPGA en forma de un vector de 588 posiciones como se observa en la Fig. 6. Para poder traducir los estímulos escritos por el usuario desde la estación de trabajo se realiza una aplicación en espacio de usuario que extrae los estímulos de un archivo .vcd (Value Change Dump) y los convierte en vectores de 588 posiciones que puedan ser leidos por URJTAG, considerando que cada señal del diseño está mapeada a un pin de la FPGA, y que ese pin respectivamente está referenciado a una posición en el BSR. La información relacionada con la ubicación de los pines y las señales se encuentra en el archivo .csv (Comma Separated Values) que surge después de la síntesis del archivo .bin a través de Xilinx.

3.3.2. Comunicación PROCESADOR+FPGA: el otro aspecto a considerar para definir el ambiente de verificación es cómo se va a conectar SIE con la estación de trabajo que contiene el resto de módulos en SystemC.

- Las alternativas que se tenían para realizar esta comunicación eran:
- Comunicación SERIAL entre HOST y SIE
- Comunicación usando «Ethernet over usb»

Se escogió la comunicación «Ethernet over usb» porque es más rápida que la comunicación serial, y porque permitía facilidad en el intercambio de información entre la aplicación ejecutándose sobre SIE y la aplicación en la estación de trabajo a través de Sockets. La comunicación se hace utilizando Ethernet sobre el cable USB, la comunicación se basa en el principio de Cliente y Servidor, en donde es el servidor quien espera por la conexión de un determinado cliente. Se seleccionó la SIE como servidor buscando que no fuera necesario re inicializar la aplicación de SIE para una nueva ejecución. Es decir, la aplicación sobre SIE se ejecuta cada vez que el cliente se conecta, y el cliente es la aplicación en la estación de trabajo. Esto significa que podemos recompilar la aplicación de la estación de trabajo para adicionar estímulos o para realizar pruebas específicas sin tener que realizar nada sobre SIE, una vez esta ya se ha inicializado y está esperando por conexión.

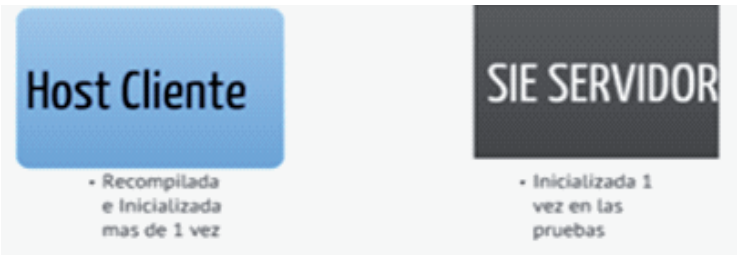

Fig. 8. Comunicación entre el emulador y la estación de trabajo (HOST).

3.4 Control de la ejecución y programación de la FPGA

Una vez definido el ambiente de verificación en detalle, se determina quién y de qué manera tendrá el control de la ejecución dentro de SIE. Este es realizado por la aplicación en espacio de usuario, que está escrita en C++. Esta se encarga de generar los respectivos comandos de URJTAG y de recibir y enviar los estímulos desde y hacia la estación de trabajo.

El control de la ejecución en la estación de trabajo se realiza por el ambiente escrito en C++ con las extensiones de SystemC, este se encarga de realizar los respectivos envíos y las respectivas recepciones, así como se encarga de la comparación y la entrega de resultados. Para la programación de la FPG se generó una plantilla de restricciones (*.UCF) con todos los pines libres de la FPGA de SIE que pueden ser utilizados para las pruebas. Esto permite que la persona que va a realizar las pruebas no necesite acceder al manual de SIE para poder observar qué pines puede utilizar, simplemente quita el carácter «#» que es usado para comentar las líneas, y se modifica según se necesite. La programación de la FPGA se realiza a través de JTAG, y también es realizada por la aplicación en espacio de usuario de SIE.

3.5 Modos de ejecución

Durante el diseño fue necesario elaborar dos tipos de ejecución, buscando alternativas de optimización del rendimiento de la verificación funcional de sistemas digitales. El sistema generado puede ejecutarse de dos formas:

3.5.1 Ejecución por ciclos: el ambiente generado para este modo de ejecución es el de la figura 4. En este ambiente por cada estímulo generado se obtiene una respuesta en los demás módulos de forma inmediata hasta generar una respectiva salida, y luego continúa para el siguiente estímulo o ciclo. Esta aplicación es llamada SimAccel Ciclos.

3.5.2 Ejecución por bloques: el ambiente generado para este modo de ejecución es presentado en la figura 9:

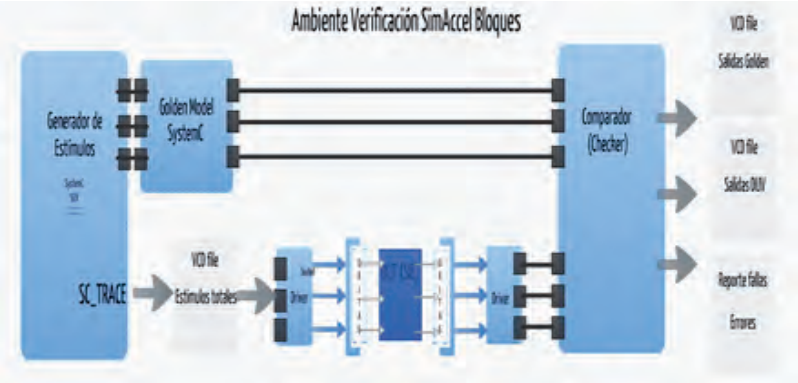

Fig. 9. Ambiente de verificación SimAccel Bloques.

En esta figura se observa un alargamiento y una contracción en la parte gráfica del Golden Model, modelo de referencia, y se hizo intencionalmente para mencionar que la ejecución sobre SIE no se presenta hasta que no se han generado la totalidad de los estímulos. Se genera un archivo en formato VCD en systemC, y es suministrado a SIE, la ejecución sobre SIE se hace en bloque de todo este conjunto de estímulos. Finalmente se entregan los resultados igualmente en bloque, y son comparados con los resultados del Golden Model. El módulo Comparador genera dos archivos en formato VCD y automáticamente los grafica en GTKWave, que es un visualizador de forma de onda. Este módulo también genera un reporte de errores donde se indica en qué señal y en qué estímulo se presenta error, el reporte de errores indica en qué señal se presenta el error, cual es su valor esperado y cuál es el valor obtenido, e indica también el estímulo en el que se presentó el error. Esta aplicación es llamada SimAccel Bloques.

3.6 Metodología utilizada

Una vez se ha definido todo el sistema y se conoce tanto la estructura como los conceptos involucrados en el ambiente de verificación, se enunciará la metodología que se utilizó para el flujo de verificación. El principio manual de esta metodología tiene fundamento en el trabajo realizado por el profesor Carlos Camargo y sus estudiantes en el proyecto «Control de protocolo JTAG usando URJTAG» [2].

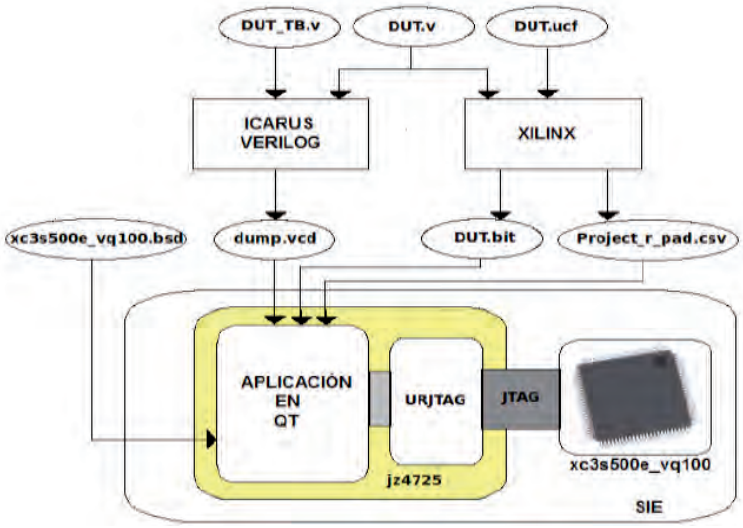

Fig. 10. Flujo de diseño. Principio conceptual para la metodología planteada.

1. Se inicia teniendo tres archivos:
   - → DUT.v: Archivo en HDL del diseño que se quiere verificar.
   - → DUT_TB.v: Archivo que contiene el test-bench con los estímulos
   - → DUT.ucf: Archivo que contiene las asignaciones de pines para la FPGA de SIE.
2. Con el archivo DUT.v y con DUT_TB.v se genera una especie de simulación temporal a través de ICARUS verilog que guarda todos los estímulos en un archivo de forma de onda. VCD (Value Change Dump).
3. Con el archivo DUT.v y con DUT.ucf se realiza la síntesis en Xilinx, obteniendo dos archivos de nuestro interés:
   - → **DUT.bit:** Es el archivo con el que se programa la FPGA, y se obtiene como producto de la síntesis en Xilinx.

→ **Project_r_pad.csv:** Es un archivo que contiene un registro de a qué señal se le ha asignado qué pin de la FPGA.

4. En este punto se debe tener el archivo xc-3s500e_vq100.bsd que es un archivo relacionado directamente en cómo está implementado JTAG sobre la FPGA de SIE.

5. La aplicación en QT, que se está ejecutando en SIE, toma el DUT.bit y programa la FPGA.

6. Luego toma el archivo con extensión .bsd y el archivo con extensión .csv e identifica qué posición del registro BSR de JTAG está asociada con qué señal del diseño implementado en la FPGA.

7. Finalmente, extrae los estímulos del archivo .vcd y se los suministra a URJTAG en forma de un registro de 588 posiciones, haciendo que correspondan los valores de interés con las posiciones de interés en los pines en que están mapeadas las señales del diseño bajo verificación.

8. Los valores de salida del DUV son capturados en un archivo de texto plano, debido a las modificaciones que se le realizaron a URJTAG.

9. Finalmente se toma este archivo con las salidas, se interpreta en las respectivas señales del mismo modo en que se suministraron, y se almacenan los resultados en otro archivo .vcd para visualizarse en un visualizador de onda como GTKWave.

Con el fin de automatizar el procedimiento del flujo de archivos, se le dio el control a la aplicación en SystemC que está siendo ejecutada en la estación de trabajo para que con tan solo entregarle los archivos DUT.v y DUT.ucf realizara todo el procedimiento mencionado y mostrara resultados a nivel de verificación, con comparaciones incluidas.

La aplicación desarrollada en SystemC suministra los estímulos aleatorios sin necesidad de suministrarlos a través del archivo DUT_TB.v, sino que utiliza funciones de la librería SCV (SystemC Verification) para proveer estímulos aleatorios y restringidos según se desee. Además de eso realiza la comunicación con SIE de forma automática y captura del mismo modo los resultados e inicializa GTKWave al finalizar la verificación para que el usuario pueda ver las formas de onda de las señales del diseño bajo verificación.

Finalmente, se han definido las características del ambiente de verificación, la entidad de control durante la ejecución, los modos de ejecución del sistema y la metodología. A continuación se mostrarán los resultados obtenidos durante las pruebas.

3.7 Resultados

Ahora se enuncian las pruebas que se realizaron con el sistema y la metodología planteada en este artículo. La primera sección contiene las pruebas iniciales que se realizan a los dos modos de ejecución (Por ciclos y por bloques), usando una compuerta XOR. La segunda sección contiene la verificación de un IP core más complejo, un I2C, usando la aplicación SimAccel Bloques.

3.7.1 Verificación XOR: en esta sección se utilizan las dos aplicaciones, SimAccel Ciclos y SimAccel Bloques. En la aplicación de SimAccel ciclos se utiliza un reloj conocido como CLK_SC, cuya función es regular las operaciones en el sistema, buscando que se genere un estímulo en el instante correcto y que posteriormente pueda ser capturado por el resto del sistema antes de que se finalice el ciclo y se envíe el siguiente estímulo. La sensibilidad de todos los módulos está ajustada buscando que esto se cumpla, en el flanco de subida de dicho clock se generan los estímulos, mientras que en el flanco de bajada funcionan los demás bloques, garantizando que todos tengan acceso al mismo grupo de estímulos en el mismo tiempo.

El período del reloj de la señal generada en SimAccel Ciclos no debe ser menor al tiempo que

tarda en efectuar un ciclo dentro de SIE, que es de alrededor de 155 ms, luego el clock del sistema no puede ser mayor a este. En la tabla 1 se observan los resultados de estas pruebas:

TABLA 1. Resultados de la prueba

	SimAccel Ciclos	SimAccel Bloques
T. ejecución INTEST+Lógica diseño	137 ms	1.78 ms
T. ciclo	153 ms	12.26 ms
. captura salidas	3 ms	7 ms
T. Transferencia de salida	2 ms	0.84 ms
T. programar FPGA	1883 ms	1666 ms

De estas pruebas podemos ver que la aplicación SimAccel Bloques posee un mejor desempeño en el tiempo que la aplicación SimAccel Ciclos. Esto se debe a que en la ejecución por ciclos se debe inicializar la instrucción INTEST que permite modificar las entradas del diseño cada vez que se ingresa a un ciclo dentro de SIE, si vemos en la tabla de resultados correspondiente para este caso, el tiempo que tarda el ciclo, casi todo este se atribuye al tiempo de la ejecución de la instrucción intest. El tiempo de ejecución de la instrucción intest es de alrededor de 137 ms y el tiempo del ciclo es de 155 ms.

En la aplicación SimAccel Bloques solamente es necesario inicializar la instrucción INTEST una sola vez, al igual que la comunicación debe ejecutarse en bloque y una sola vez, lo que explica que se obtenga un mejor rendimiento. Las formas de ondas generadas para las pruebas tantas en SimAccel ciclos como en SimAccel bloques se encuentran en la figura 11. En esta figura se puede apreciar el reloj que se menciona en SimAccel Ciclos y se aprecia que los estímulos se generan en el flanco de subida de este reloj, y son procesados por el resto de los módulos del ambiente de verificación en el flanco de bajada.

3.7.2 Verificación módulo I2C: para la verificación del módulo I2C se utilizó solamente la aplicación de SimAccel Bloques, aunque hubieran podido ser usadas las dos, se optó por esta debido a que fue la que presentó mejor desempeño temporal en la sección anterior.

3.7.2.1 Funcionamiento: para la verificación del módulo I2C se utilizó solamente la aplicación de SimAccel Bloques, aunque hubieran podido ser usadas las dos se optó por esta debido a que fue la que presentó mejor desempeño temporal en la sección anterior.

Fig. 11. Resultados de la verificación de la compuerta xor.

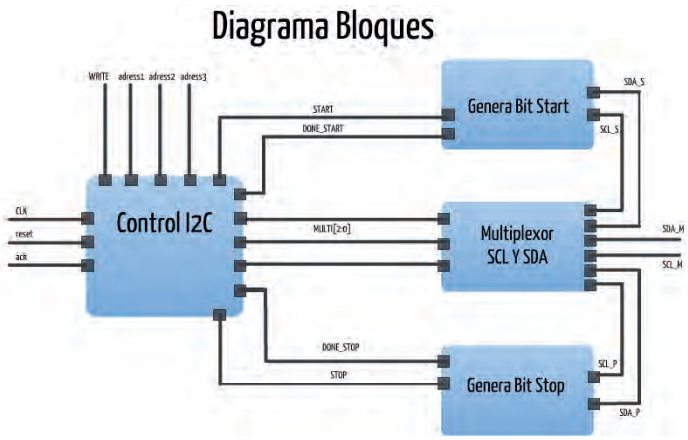

Figura 12. Diagrama de bloques de un I2C.

El IP core implementado consta de 4 bloques:

- **Control I2C:** se encarga de generar la secuencia de la máquina de estados de una comunicación I2C, enviando las señales de Bitstart, de escritura, de Bitstop en los debidos tiempos, garantizando que la comunicación se realice.

- **Genera Bit Start:** este módulo recibe una señal proveniente del módulo Control I2C cada vez que hay una condición de start, y posteriormente genera la secuencia de start para el bus I2C, que es poner en bajo SDA mientras SCL continua en alto.

- **Genera Bit Stop:** este módulo recibe una señal proveniente del módulo Control I2C cada vez que hay una condición de stop, y esta es poner SDA en alto mientras el SCL continúa en alto.

- **Multiplexor SCL Y SDA:** el multiplexor se encarga de asignar sobre el bus, que señal de SCL y SDA deben estar activas, por ejemplo para la condición del bitstart, el multiplexor debe garantizar que sobre el bus I2C se encuentren las señales SCL y SDA provenientes del módulo Bitstart, que son las que poseen la secuencia de inicialización del Bus. La máquina de estados de este I2C implementado está en presentado en la figura 13.

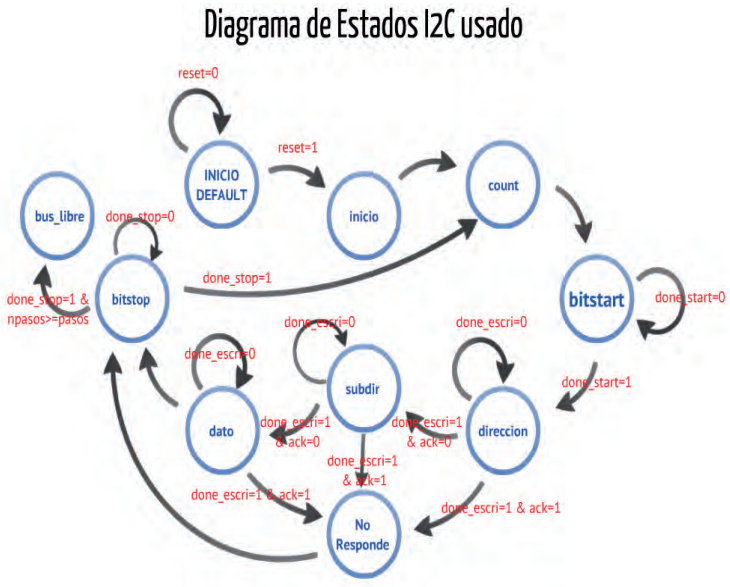

Figura 13. Diagrama de estados del I2C implementado.

Básicamente la información que contiene esta máquina de estados indica que el controlador I2C implementado lo que realiza es el control de las operaciones de un maestro con un dispositivo esclavo, en donde el maestro accede periódicamente al bus I2C, y escribe sobre un dispositivo, una dirección, una subdirección y un dato, y luego de realizar 4 de estos conjuntos de escrituras, se deja libre el bus.

3.7.2.2 Verificación

- **Funcionamiento sin errores:**

TABLA 2.

T. programar FPGA	1666 ms
T. lógica diseño	750 ms
T. interpretar y enviar estímulos	2234 ms
T. recibir salidas	4527
T. recibir salidas	4527 ms
T. Interpretar salidas	77 ms
T. ciclo total (Dentro de SIE)	5744 ms

Vemos que hay que pagar siempre unos costos fijos en tiempo, como el tiempo de ejecutar la instrucción INTEST, o como el tiempo de programar la FPGA, aun cuando este en la ejecución por bloques se haga antes de iniciar la conexión con SIE. En cuanto al reloj del diseño bajo verificación, hay que decir que este no puede ser generado por uno de los cristales dentro del emulador cuando se utiliza el protocolo JTAG, específicamente con este módulo URJTAG, puesto que este aísla todos los pines de la FPGA, forzando al sistema a generar el reloj a través de un pin de la FPGA como si fuera otra señal más del diseño. Esto no resulta conveniente a nivel de desempeño puesto que hay que realizar un corrimiento del registro BSR por cada transición del reloj, y esto se torna más complicado cuando se trabaja con un clock de alta frecuencia, puesto que es necesario realizar muchos corrimientos del BSR para poder abarcar todos los cambios del reloj de alta frecuencia.

Para un clock de período 5US (El concepto de las unidades en realidad no aplica en sim_accel bloques, debido a que simplemente pasa el archivo VCD como transiciones), con un tiempo de simulación de 10 000US el sistema funciona aún, aunque con tiempos grandes, de alrededor de 45 segundos, siendo un total de 4000 estímulos. Para un período menor a 5 bajo este tiempo de simulación, el sistema no funciona puesto que ocurre una desincronización a causa de los extensos tiempos que tienen que esperar los módulos para registrar el alto número de estímulos. Luego más que una restricción en frecuencia del reloj, es una restricción en el número alto de estímulos que conlleva trabajar con un reloj de alta frecuencia. En la gráfica 20 se presentan los resultados de la simulación.

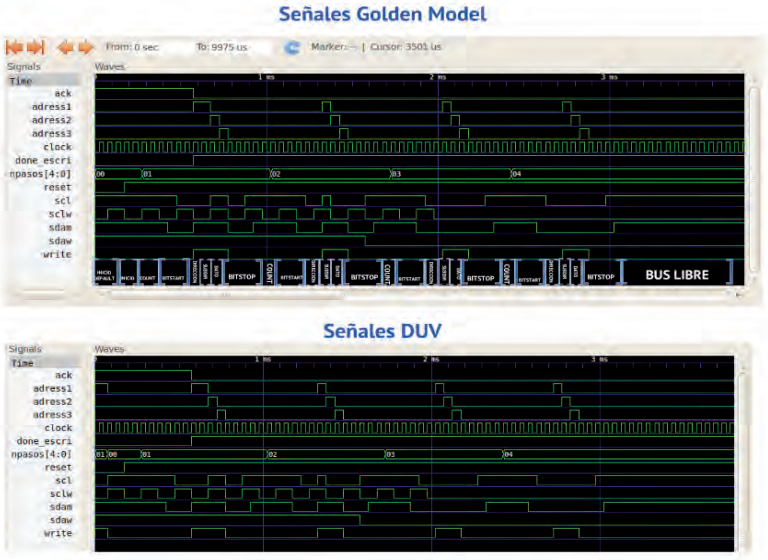

Figura 14. Resultados I2C GtkWave

En la figura 14 es posible observar el comportamiento del I2C, en donde las escaleras de la parte de arriba de la simulación son las tres señales que se emiten para escribir en una dirección, subdirección y en un dato, luego vemos que se realizan 4 de estas instrucciones y luego se deja libre el bus. En este punto vale la pena observar que el comportamiento de inicio del diseño bajo verificación no coincide con el comportamiento del modelo de referencia, y es debido a que el sistema en la FPGA debe ingresar a un estado válido luego de efectuar la señal Reset para poder iniciar la comparación.

- **Funcionamiento con errores:** buscando validar la aplicación diseñada, se decidió realizar la verificación del mismo I2C pero con errores, esperando que el sistema detecte el error e indique algún tipo de mensaje de depuración.

- **Error 1:** el primer error inducido es en la condición BITSTART, que es la condición en la que el SDA se pone en bajo mientras SCL está en alto, buscando inicializar la comunicación sobre el bus I2C.

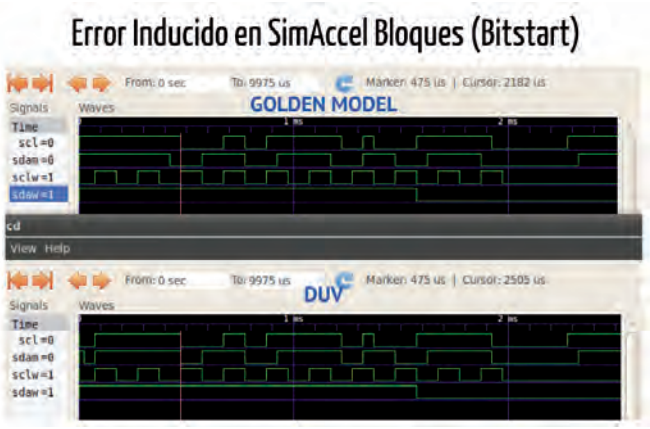

Figura 15. Formas de onda para el error inducido.

En la figura se ve con la línea roja de señalador. Se observa que en el golden model o modelo de referencia si se genera la condición BITSTART, mientras que en el diseño bajo verificación, no se genera la condición de start, correctamente por el error inducido.

```
--------------------------------------------------------------------------
-- Error Encontrado
--------------------------------------------------------------------------
No estímulo | 103
SeÃ±al | sdam
Valor esperado | 0
Valor obtenido | 1
--------------------------------------------------------------------------
-- Error encontrado
--------------------------------------------------------------------------
No estímulo | 104
SeÃ±al | sdam
Valor esperado | 0
Valor obtenido | 1
--------------------------------------------------------------------------
---- Número de errores de inicio
12
---- Número total de errores
8
================================
```

Figura 16. Reporte de errores.

Del reporte de errores se observa que si se está señalando la respectiva señal que presenta el error, indicando el valor que se esperaba e indicando el número de estímulo.

Error 2: el segundo error inducido es en uno de los bits de address, en donde se genera una señal de adress1 con el error en el código de verilog.

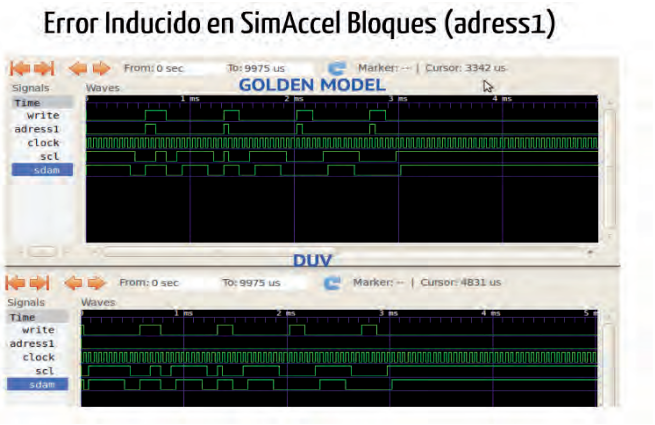

Figura 17. Formas de onda para el error inducido.

Se observó que el diseño bajo verificación presenta un error en adress1.

```
--------------------------------------------------------------------------------
-- Error encontrado
--------------------------------------------------------------------------------
 No estímulo | 109
 SeÃ±al | adress1
 Valor esperado | 1
 Valor obtenido | 0
--------------------------------------------------------------------------------
-- Error encontrado
--------------------------------------------------------------------------------
 No estímulo | 110
 SeÃ±al | adress1
 Valor esperado | 1
 Valor obtenido | 0
--------------------------------------------------------------------------------
---- Número de errores de inicio
 9
---- Número total de errores
 10
================================
```

Figura 18. Reporte de errores.

El error inducido fue adress1, y efectivamente el reporte de errores lo determina y envía mensajes con fines de depuración.

3.7.3 Propuesta de mejora: luego de realizar un trabajo extenso sobre SIE se definen unas características que debe tener una plataforma con base en SIE, para realizar exclusivamente esta técnica de aceleración de la simulación planteada.

3.7.3.1 Características del Emulador propuesto

Tabla 3.

FPGA	Xc3s500e_vq100
PROCESADOR	JZ4725
MÓDULO SERIAL	NO
MICRO SD	NO
LCD	NO

Los criterios para definir las características anteriores fueron los siguientes:

Explorando otras alternativas de procesador, teniendo en cuenta que el proyecto SIE soporta la familia de los procesadores JZ47XX, se contempló la posibilidad de reemplazar el JZ4725 de SIE por un JZ4730 que poseía soporte para un puerto Ethernet, el cual llegó a contemplarse como una alternativa de comunicación rápida para el intercambio de datos de las aplicaciones. Luego de la elaboración del ambiente entero fue evidente que no se poseían inconvenientes de comunicación y que la velocidad que estaba entregando el puerto usb era suficiente, por lo que no se justificaba la reasignación de pines para un nuevo procesador. Dado que el tamaño de la FPGA implica directamente el tamaño del diseño que puede ser verificado, se realizó una búsqueda de otro dispositivo que fuera PIN COMPATIBLE con la Spartan 3E que se poseía, para poder realizar el cambio dentro de los tiempos de diseño establecidos. Pero vimos que la FPGA de SIE era la de mayor tamaño que presentaba el empaquetado VQ100. Realizamos un análisis de los códigos que iban a ser implementados en las pruebas iniciales y nos dimos cuenta de que la FPGA que se tenía era más que suficiente, puesto que disponía de 500 mil compuertas, y rediseñar para otra FPGA más compleja no iba a tener ningún impacto en el desempeño, pues con los dos dispositivos se podrían implementar igualmente las pruebas.

A continuación se procedió a hacer un análisis de qué características de SIE no estaban siendo empleadas en la aplicación para poder disminuir costos suprimiéndolas. De esta forma se suprimió el LCD, ya que no se estaba graficando, se suprimió el módulo de la MICRO SD y el módulo SERIAL. Con la nueva SIE HE (SIE Hardware Emulation), que en realidad resulta ser una derivación de SIE, pues los cambios no la hacen considerablemente diferente de SIE se obtuvo una reducción en costo de fabricación del PCB de alrededor del 25% con respecto a la SIE V2.

Área SIE V2: 8,2 cm X 9,1 cm
Área SIE HE: 8,1 cm X 8,2 cm

3.7.3.2 Fabricación del Émulador propuesto: se realizaron las modificaciones sobre el PCB de SIE y se fabricó el emulador propuesto:

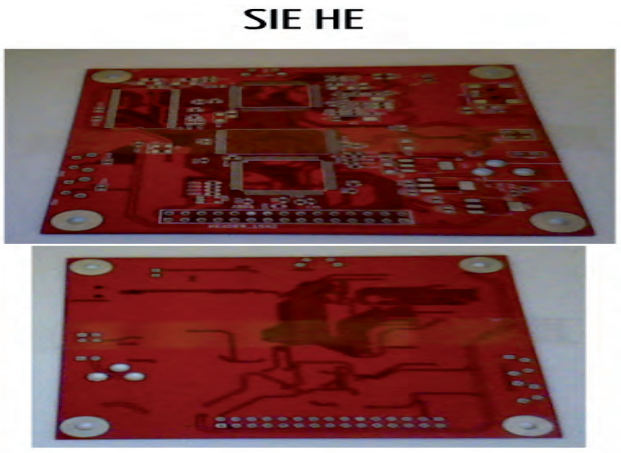

Figura 19. Plataforma SIE HE (Hardware Emulation).

IV. Conclusiones

Este artículo presentó el diseño y la implementación de un sistema de verificación funcional basado en la técnica de la aceleración de hardware, «Aceleración de la simulación», utilizando herramientas libres (hardware y software libre). El sistema utilizando el protocolo JTAG posee restricciones. El reloj que puede generarse no puede ser de alta frecuencia.

La Tarjeta SIE resulta ser un emulador bastante robusto para interactuar con el diseño bajo verificación implementado en su FPGA.

Se manejaron dos tipos de implementaciones, SimAccel Bloques y SimAccel Ciclos, y se observó que la aplicación por bloques presenta un mejor desempeño temporal.

Los resultados indican que el sistema tendría su mejor desempeño en IP cores cuya ejecución desprecie los costos fijos de tiempo que se tienen. Con este trabajo se contribuye a la estructura del curso

de verificación de sistemas digitales de la Universidad Nacional de Colombia y a los desarrollos en microelectrónica que se están llevando a cabo en el grupo de investigación GMUN.

V. Trabajos futuros

Se deben explorar nuevas técnicas de comunicación FPGA+Procesador que no estén incluidas en SIE y que permitan la utilización del reloj generado por el cristal de la plataforma, o bien que den una alternativa para incrementar la velocidad y continuar disminuyendo los tiempos de verificación en una mayor magnitud. Realizar un sistema de emulación cuyo componente hardware posea múltiples FPGA para adquirir mayor velocidad, y mayor ejecución paralela.

Referencias bibliográficas

[1] C.I. Camargo, L.A. de Oro, E. M. Abello and J.R. Cuarán, «Control de Protocolo JTAG usando URJTAG» , Networks (Online), Available: http://en.qi-hardware.com/wiki/Control_de_protocolo_JTAG_usando_URJTAG

[2] C.I. Camargo, «SIE», Networks (Online), Available: http://en.qi-hardware.com/wiki/SIE

[3] H.A. González, Section: «Control de Protocolo JTAG usando URJTAG», Networks (Online), Available: http://linuxencaja.net/wiki/Analizador_y_Generador_de_Patrones

[4] Koninklijke Bibliotheek, «What is Emulation?», Networks (Online), Available: www.kb.nl/hrd/dd/dd_projecten/projecten_emulatiewatis-en.html

[5] Bailey, «To Emulate or Prototype», Networks (Online), Available: www.eetimes.com/electronics-blogs/other/4209290/To-emulate-or-prototype-

[6] L.T. Wang, Y.W. Chang, and K.T. Cheng, Electronic Design Automation (Book style). Morgan Kauffman, 2009, ch 8, section 8.4.

[7] J. Andrews, Co-Verification of Hardware and Software for ARM SoC Designs. Newnes, 2005, ch 2.

[8] C.I. Camargo, «SIE: Plataforma Hardware copyleft para la Enseñanza de Sistemas Digitales», IBERCHIP 2011.

[9] J. Kumar, «HW Emulators: Does it belong in your Verification tool chest?», Sun Microsystems Inc. DV Club May 23rd 2007.

[10] S. Banerjee, T. Gupta, «Automatic Error Recovery in Targetless Logic Emulation», Mentor Graphics Pvt. Ltd. India, IEEE.

[11] [XJTAG, Section: «JTAG technical Guide», Networks (Online), Available: www.xjtag.com/support-jtag/jtag-technical-guide.php

[12] IEEE, Section: «JTAG 1149-1990», Networks (Online), Available: http://standards.ieee.org/findstds/standard/1149.1-1990.html

[13] IEEE, Section: «JTAG standard», Networks (Online), Available: http://es.scribd.com/doc/63310993/33/The-RUNBIST-instruction

[14] Cambridge Consultants, Section: «XEMU boards», Networks (Online), Available: www.cambridgeconsultants.com/cs_xemu.html

[15] Cadence, Section: «PALLADIUM XP», Networks (Online), Available: www.cadence.com/products/sd/palladium_xp/Pages/default.aspx

[16] EVE, «Design and Implementation of an OCP-IP Compliant 64-Node Butterfly Network on Chip on Multi-FPGA», Networks (Online), Available: www.design-reuse.com/articles/27711/ocp-ip-compliant-64-node-butterfly-network-on-chip-on-multi-fpga.html

Agradecimientos

Los autores reconocen las contribuciones del profesor Carlos Camargo de la Universidad Nacional de Colombia quien brindó soporte al trabajo que se realizó con la plataforma SIE.