Verificación de Funcionamiento de una Arquitectura de Procesador Basada en RISC-V Utilizando UVM

Óscar Mauricio Díaz Silva

Trabajo de Grado para Optar el Título de ingeniero electrónico

Director

Héctor Iván Gómez Ortiz

MSc en Ciencias

Codirector

Élkim Felipe Roa Fuentes

PhD en Ingeniería Eléctrica y de Computadores

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecanicas

Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones

Bucaramanga

2017

Contenido

Pág.

Lista de Figuras

Pág.

Resumen

TÍTULO: VERIFICACIÓN DE FUNCIONAMIENTO DE UNA ARQUITECTURA DE PROCESADOR BASADA EN RISC-V UTILIZANDO UVM[*]

AUTOR: OSCAR MAURICIO DIAZ SILVA[**]

PALABRAS CLAVE: SISTEMA DE VERIFICACIÓN, ARQUITECTURA DE PROCESADOR, METODOLOGÍA UNIVERSAL DE VERIFICACIÓN.

DESCRIPCIÓN:

La verificación de cada etapa del proceso de desarrollo de un sistema digital tiene un papel fundamental en la industria electrónica actual, debido a que la fabricación de los prototipos de estos sistemas es un proceso altamente costoso y la presencia de fallas representa pérdidas económicas considerables.

Este documento expone un sistema de verificación funcional diseñado utilizando la Metodología Universal de Verificación (UVM) para verificar una arquitectura de procesador de 32 bits basada en RISC-V. Esta metodología tiene capacidades de reutilización y permite automatizar las tareas de estimulación y análisis del procesador, además de que se encarga de gran parte de las tareas triviales propias de la simulación. El sistema propuesto cuenta con elementos definidos utilizando programación orientada a objetos, los cuales comunican el sistema de verificación con el procesador y un modelo de memoria, generan instrucciones en lenguaje de máquina y comparan los resultados con los obtenidos mediante la estimulación de un modelo de ejecución del procesador. Estos elementos verifican la correcta ejecución de parte del set de instrucciones RV32I y permiten verificar los resultados de las interacciones de estas instrucciones dentro del esquema de *pipeline* de 3 etapas para descubrir posibles fallos de tipo estructural presentes en la arquitectura implementada. Como resultado de la verificación realizada se presentan los porcentajes de cobertura obtenidos para las interacciones del total de las instrucciones seleccionadas y para cada familia de instrucciones.

[*] Trabajo de grado

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingeniería Eléctrica, Electrónica y de Telecomunicaciones. Director: Héctor Iván Gómez Ortiz, Maestro en ciencias.

Abstract

TITLE: A BEHAVIORAL VERIFICATION USING UVM FOR A PROCESSOR ARCHITECTURE BASED ON RISC-V[*]

AUTHOR: OSCAR MAURICIO DIAZ SILVA[**]

KEYWORDS: VERIFICATION SYSTEM, PROCESSOR ARCHITECTURE, UNIVERSAL VERIFICATION METHODOLOGY.

DESCRIPTION:

Verification of each stage in the development process of a digital system has a fundamental role in nowadays electronics industry, since manufacturing prototypes of these systems is a highly expensive process and presence of failures represents significant economic losses.

This document exposes a behavioral verification system designed using the Universal Verification Methodology (UVM) to verify a 32-bit processor architecture based on RISC-V. This methodology has reuse capabilities, allows to automate stimulation and analysis tasks of the processor and is responsible for many mundane simulation tasks. The proposed system has elements defined using object-oriented programming, which communicate the verification system with the processor and a memory model. They also stimulate the processor by generating machine code instructions and compare results data with the values obtained through stimulation of a processor execution model. These elements verify the correct execution of part of the RV32I instruction set and allow to verify the results of the interactions of these instructions within the 3-stage pipeline scheme to discover possible structural failures present in the implemented architecture. As a result of the verification carried out, the percentages of coverage obtained for the interactions of the total of the selected instructions and for each family of instructions are presented.

[*] Bachelor Thesis

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingeniería Eléctrica, Electrónica y de Telecomunicaciones. Director: Héctor Iván Gómez Ortiz, Maestro en ciencias

Introducción

El avance de gran parte de la ciencia en las últimas décadas se ha estimulado gracias al desarrollo de las computadoras y al aumento constante de su capacidad de cómputo, permitiendo la creación de software cada vez más potente y abriendo un mundo de nuevas posibilidades. El aumento de la capacidad de cómputo ha sido posible en parte gracias al desarrollo de las tecnologías de fabricación y al mejoramiento de las microarquitecturas utilizadas en los procesadores. Hoy en día los procesadores están presentes en muchos aspectos de la vida diaria de las personas, no sólo en la ciencia sino también en la industria y en los dispositivos de consumo, desde los que se encuentran en los electrodomésticos hasta los presentes en los sistemas de conducción autónoma en los automóviles. Dada la cantidad de procesos que son controlados por estos sistemas digitales, un error en su funcionamiento podría causar danos humanos y materiales, por lo tanto, antes de su salida al mercado deben ser probados y verificados en todo su espacio funcional.

Los lenguajes de descripción de hardware (HDL) son ampliamente utilizados en la industria para el modelado y verificación de sistemas digitales, sin embargo, las capacidades de verificación de estos lenguajes son muy limitadas. Estas limitaciones representan un reto en la verificación de sistemas complejos como los procesadores y a menudo su resultado es incierto, además de ser una tarea altamente ineficiente. Para solucionar este problema, la industria ha creado con el paso de los años varios lenguajes de verificación de hardware (HVL) como Vera, SystemC y e. A pesar de las soluciones que aportan estos lenguajes propietarios, su utilización requiere tiempo de capacitación lo cual representa costos para la industria y retrasos en el desarrollo de los entornos y en la verificación misma. Además, se presentan como una barrera entre los equipos de diseño y

verificación debido a sus diferencias con los HDLs [Sutherland & Mills, 2003]. La respuesta a estos problemas es una extensión de Verilog con capacidades de HVL llamada SystemVerilog. Gracias a la popularidad de Verilog, la adopción de este nuevo lenguaje es más rápida y debido a que también es un HDL elimina las barreras entre los equipos de diseño y verificación. Es considerado el primer lenguaje de descripción y verificación de hardware (HDVL).

Este documento presenta un esquema de verificación para la segunda versión del procesador Turpial, el primer procesador con set de instrucciones RISC-V [Waterman & Asanovic, 2017] fabricado en el mundo. A pesar del funcionamiento de este procesador, la verificación efectuada a nivel de arquitectura antes de su fabricación se realizó utilizando programas de prueba escritos manualmente, lo cual no permite cuantificar el espacio funcional cubierto y asegurar un funcionamiento correcto en todo momento. La segunda versión de este procesador implementa un esquema de *pipeline* de 3 etapas, esto motiva el desarrollo de este sistema de verificación para mejorar la eficiencia y fiabilidad del proceso, añadiendo la flexibilidad de que permite verificar posteriores versiones con base en el esquema general propuesto. Esta versión del procesador tiene como nombre Olinguito y será el núcleo de la segunda versión del microcontrolador Open-V [Duran *et al.*, 2016] desarrollado por el grupo de investigación *OnChip*. Adicionalmente, este proyecto busca extender las capacidades de verificación con las que cuenta el grupo de investigación, integrándose a la base de datos de verificación donde se encuentra el sistema desarrollado previamente para una memoria LPDDR3 [Ramírez, 2016].

1. Metodología universal de verificación (UVM)

La Metodología Universal de Verificación es un estándar para la verificación de diseños de circuitos integrados y es el resultado de la unión de diferentes metodologías de verificación (eRM, OVM, VMM) creadas por las distintas compañías en la última década. Este estándar es presentado como una librería de clases base, una referencia de clases [Accellera, 2014] y una guía de usuario [Accellera, 2015].

Esta librería es descrita en SystemVerilog y por lo tanto hereda todas sus características, algunas de estas son: clases, interfaces, aserciones, colectores de cobertura y restricciones de aleatoriedad entre otras. Estas herramientas proveen un mejor control de los objetivos de verificación y aumentan la eficiencia del proceso. Gracias a las aserciones es posible detectar fallos funcionales en etapas tempranas del proceso. Los colectores de cobertura permiten analizar el espacio funcional verificado y realizar pruebas directas para los casos no cubiertos de manera aleatoria. Además, este lenguaje posee técnicas orientadas a objetos tales como extensión, polimorfismo, clases parametrizadas y patrones de fábrica.

A pesar de contar con todas estas herramientas, SystemVerilog no conduce por sí mismo a la adopción de las mejores técnicas de verificación [Bromley, 2013]. El lenguaje provee todo lo necesario para elaborar un entorno de verificación complejo, sin embargo, para alguien poco experimentado esto será todo un desafío. Esta dificultad de adopción genera una necesidad de guía para aprovechar todas estas ventajas. La Metodología Universal de Verificación aborda este problema proporcionando una jerarquía de objetos enfocada a la reutilización y el fácil mantenimiento de los mismos. Estos objetos están organizados de tal manera que cada uno realice

una única función de la mejor manera posible [Salemi, 2013], esto permite una depuración efectiva ya que la detección de la funcionalidad que presenta fallas indica inmediatamente su objeto correspondiente. Gracias a esta organización cada nuevo objeto se convierte en una herramienta para el desarrollo del entorno de verificación lo que hace que a medida que el entorno crece también se hace más potente. Además de la guía que nos ofrece esta metodología en la generación eficiente y adaptable de un entorno de verificación, también se encarga de la mayoría de las tareas mundanas asociadas a la simulación de un banco de pruebas.

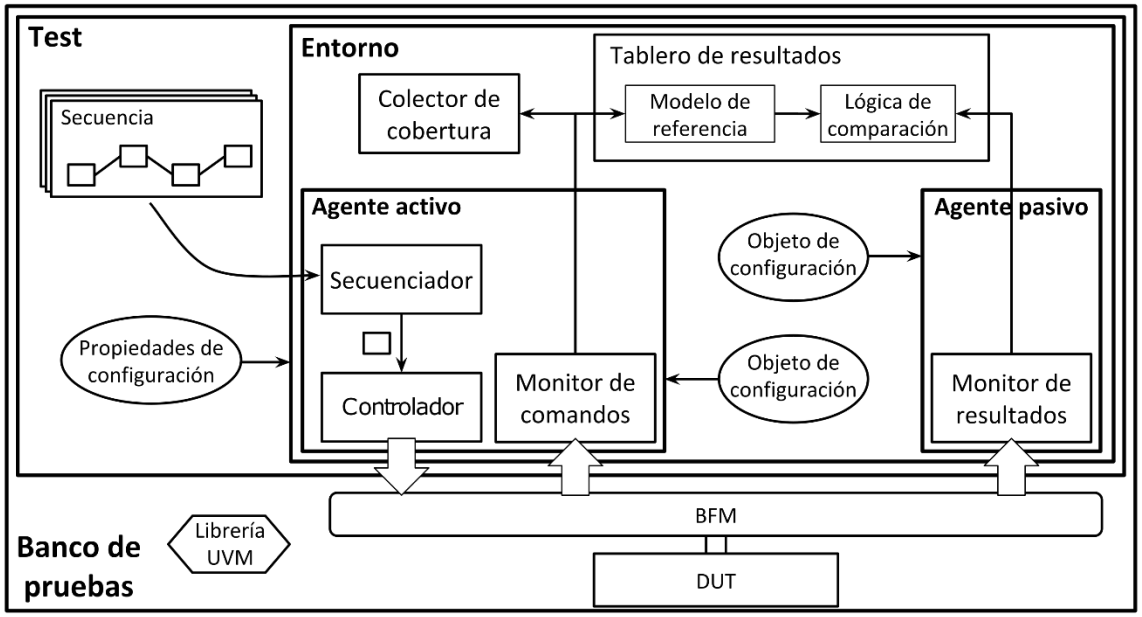

Figura 1. Banco de pruebas UVM

Para utilizar la librería UVM basta con extender las clases base con las que cuenta y formar el banco de pruebas deseado. La estructura de un banco de pruebas general se muestra en la Fig. 1, los niveles de jerarquía se muestran en negrita y a continuación se describen los elementos mostrados y su función:

Los ítems de secuencia son los objetos encargados de encapsular la estructura de datos necesaria para definir cada estímulo del DUT (Device Under Test) y facilitar el transporte de estos estímulos a través del banco de pruebas. Las secuencias se encargan de definir la información transmitida por los ítems de secuencia y el orden de transmisión de estos. Las secuencias y sus ítems no hacen parte de la jerarquía UVM. Un BFM (Bus Functional Model) es una interfaz de SystemVerilog que se encarga de comunicar el DUT con los objetos del sistema de verificación. Para esto contiene un modelo del protocolo de comunicación utilizado por el DUT, además proporciona el reloj y en algunos casos contiene tareas de control como el reinicio.

Los agentes representan el primer nivel en la jerarquía UVM, estos objetos cuentan con 2 modos de configuración, como agentes activos o pasivos. Cada agente pasivo encapsula únicamente un monitor, el cual se encarga de obtener la información de respuesta del DUT a través de las señales del BFM y la adapta para transmitirla a los elementos de análisis. Cada agente activo encapsula un secuenciador, un controlador y un monitor. El secuenciador se encarga de iniciar la ejecución de las secuencias y transmite los estímulos de cada secuencia al controlador. Una vez el controlador recibe un estímulo, lo transmite al DUT a través de las señales del BFM. El monitor obtiene la información de los estímulos de las señales del BFM y la adapta para transmitirla a los elementos de análisis.

El siguiente nivel en la jerarquía es el entorno, el cual contiene los elementos de análisis, los agentes y sus objetos de configuración necesarios. Los elementos encargados del análisis generalmente son un colector de cobertura y un tablero de resultados. El colector de cobertura se encarga de almacenar y organizar la información de estímulos obtenida de los monitores de los agentes activos de acuerdo al modelo de cobertura deseado, usualmente este modelo representa el espacio funcional objetivo de verificación. El tablero de resultados obtiene la información de

estímulos y respuestas del DUT a través de los monitores y se encarga de comparar los datos de respuesta con los obtenidos como resultado de la estimulación del modelo de referencia.

El objeto de mayor jerarquía en UVM es el test, este se encarga de controlar el inicio y la finalización de la simulación, lanza las secuencias utilizando su respectivo secuenciador y contiene el entorno y sus propiedades de configuración. Finalmente, el banco de pruebas se encarga de lanzar los test y contiene el DUT, el BFM y el paquete que contiene la librería UVM necesaria para realizar la simulación.

2. Esquema de verificación

La verificación funcional es el proceso inverso del diseño, en el diseño se da una especificación y se desea desarrollar una implementación que cumpla con esta. Para el caso de la verificación funcional se parte de una implementación y se quiere comprobar la especificación inicial. En la arquitectura de un procesador, la especificación inicial es el set de instrucciones con el que debe funcionar que para nuestro procesador objetivo es el set RV32I.

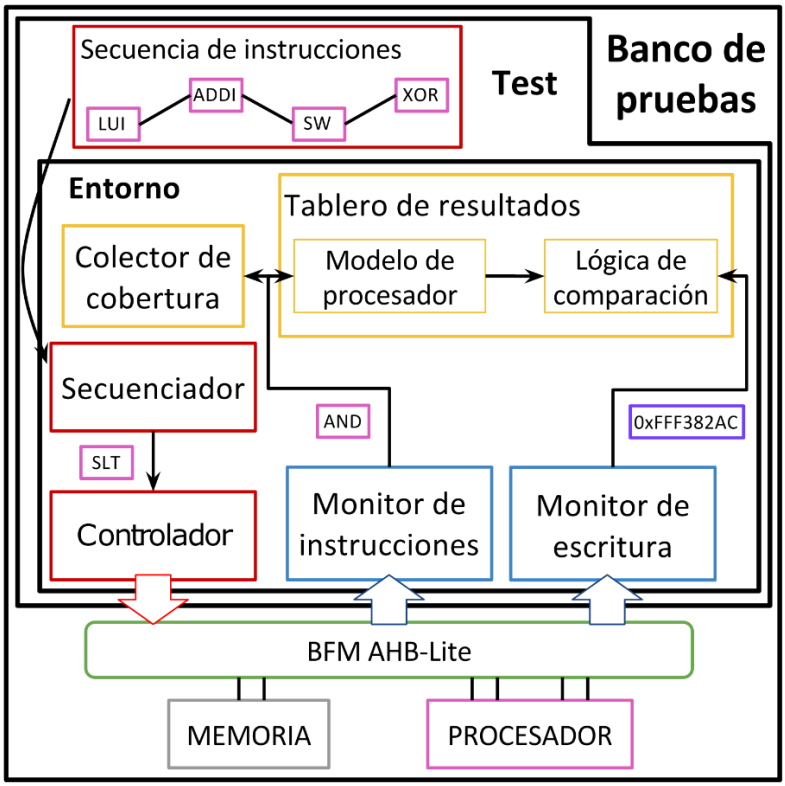

Figura 2. Banco de pruebas del procesador

Con el fin de comprobar la correcta implementación de este set de instrucciones, se requiere evaluar el resultado de la ejecución de cada instrucción. Sin embargo, el resultado de la estimulación del procesador no siempre es visible ya que muchas de estas instrucciones modifican el estado interno del mismo, es decir, los registros o el contador de programa. Debido a esto el proceso de evaluación se condiciona a la necesidad de una instrucción de escritura que permita comparar el resultado en memoria con el deseado. Esta característica se hace significativa si se realiza la verificación individual de cada instrucción, ya que se modifica la naturaleza aleatoria que debe tener el proceso y no permite la variación del contexto que se presenta en la ejecución cotidiana del procesador. Para superar este inconveniente se debe realizar la verificación durante un ciclo normal de ejecución, es decir, la comparación de los datos escritos a memoria se realiza cuando de manera aleatoria aparece la instrucción de escritura. En dirección a implementar esta idea, se hace necesario el diseño de un modelo de ejecución del procesador para comparar los resultados.

El esquema de verificación desarrollado se basa en el modelo básico de UVM presentado anteriormente. Para adaptar este modelo a un procesador se realizaron las modificaciones siguientes: se agregó un módulo extra como modelo de memoria y se eliminó el nivel de jerarquía en el que estaba el agente para simplificar el entorno. En la Fig. 2 se muestra la estructura del banco de pruebas del procesador, el funcionamiento de éste es descrito a continuación

2.1. Comunicación

La comunicación con el procesador se realiza a través del BFM demarcado en verde en la Fig. 2, éste contiene 2 protocolos de comunicación AHB-Lite [ARM, 2006], uno para instrucciones y otro

para datos. Además, cuenta con un protocolo de comunicación simple para la memoria, contiene la tarea de reinicio del procesador y las 2 tareas de escritura de datos a los monitores de instrucciones y de escritura. La comunicación con la memoria es un proceso controlado por tareas independientes al protocolo AHB-Lite, esto con el fin de recibir los datos del controlador sin necesidad de ocupar las tareas del bus del procesador. La operación de las tareas de comunicación con la memoria durante un ciclo de reloj se muestra en la Fig. 3.

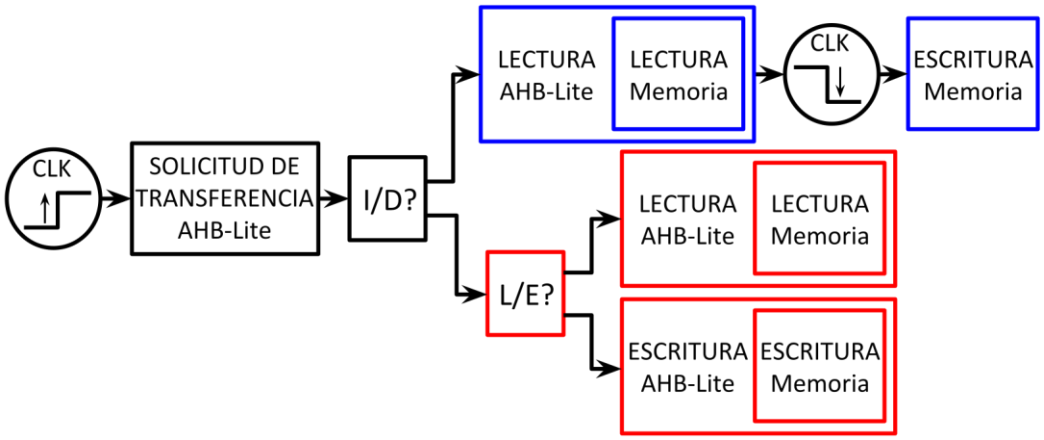

Figura 3. Tareas de comunicación durante un ciclo de reloj

En esta figura se presentan en azul y rojo las tareas involucradas en el flujo de instrucciones y datos respectivamente. El procesador sólo opera un bus por ciclo de reloj, si la transferencia es de instrucciones el bus únicamente realiza lecturas. Una vez finalizada la operación, en el flanco de bajada del reloj el controlador escribe nuevas instrucciones a memoria. Si la transferencia es de datos, puede ser de lectura o escritura y una vez realizada espera al siguiente ciclo.

2.2 Estímulos

Los elementos de la capa de estímulos demarcados en rojo se encargan de transmitir las instrucciones hasta el procesador y los campos que forman cada instrucción son estructurados en los ítems de secuencia demarcados en rosa como se muestra en la Fig. 2. Estos campos son: valor inmediato (imm), registro fuente 1 (rs1), registro fuente 2 (rs2), registro destino (rd), código de operación (opcode) y selectores de operación (funct3 y funct7). El programa cargado a la memoria es definido en la secuencia mediante los ítems de secuencia. Una vez se transmite un estímulo hasta el controlador, éste llama un método propio del estímulo que utiliza los parámetros de cada objeto previamente definidos y obtiene el código binario que representa cada instrucción utilizando el modo de direccionamiento apropiado. Luego, el controlador escribe esta instrucción en la memoria mediante la tarea que proporciona el BFM. El flujo de estímulos es controlado por el avance en la dirección de lectura de instrucciones del procesador.

2.3. Análisis

Los elementos de la capa de análisis demarcados en amarillo operan utilizando los datos transmitidos por los monitores demarcados en azul como se muestra en la Fig. 2. Esta transmisión de información está controlada por el avance en la dirección de lectura de instrucciones para el monitor de instrucciones y por la solicitud de escritura del bus de datos para el monitor de escritura. Dentro del colector de cobertura se define el espacio funcional de cada familia de instrucciones y para las combinaciones de instrucciones posibles dentro del *pipeline* de 3 etapas. Este colector recibe la información de los estímulos ejercitados mediante el monitor de instrucciones.

El modelo de ejecución del procesador diseñado no posee la capacidad de modificar el contador de programa, por esta razón se excluyen de la verificación las instrucciones que realizan esta tarea. El tablero de resultados contiene este modelo y lo estimula a medida que el monitor de instrucciones entrega datos. Los datos proporcionados por el monitor de escritura demarcados en púrpura en la Fig. 2, se comparan con los datos obtenidos del modelo y se reportan los resultados.

3. Resultados

Para asegurar el correcto funcionamiento del sistema de verificación se realizó una prueba de comunicación utilizando instrucciones no aleatorias y se compararon sus resultados con los esperados observando los datos escritos en la dirección predefinida. La Fig. 4 muestra las pruebas de comunicación del sistema y se muestran las señales relevantes de cada bus. Las señales que inician con I pertenecen al bus de instrucciones y las que inician con D al bus de datos. Para estas pruebas se precargo el número 0x99999999 a la memoria en la dirección 0x190. Las instrucciones de prueba son las siguientes: La primera instrucción sombreada en azul, es una lectura de palabra de la dirección 0x190 de la memoria y su resultado se almacena en el registro número 1. La segunda instrucción sombreada en verde, es una adición con valor inmediato, al registro 1 se le agrega el valor 0x111 y el resultado se almacena en el mismo registro. La tercera instrucción en este caso sombreada en amarillo, es una escritura de palabra a memoria en la dirección 0x190, se escribe el valor del registro 1. La línea punteada púrpura encierra las señales que ejecutan la lectura y la línea punteada azul encierra las señales que ejecutan la escritura. En la figura se muestra la generación

de instrucciones y el resultado del proceso de prueba, éste es el dato 0X99999AAA en la dirección 0x190 que concuerda con el esperado, comprobando así la correcta comunicación con el procesador.

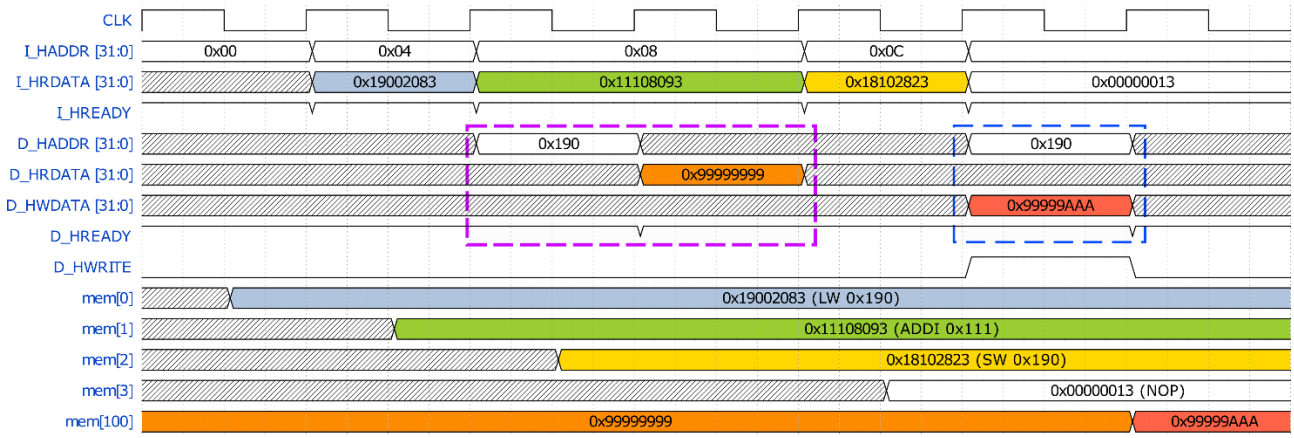

Figura 4. Pruebas de comunicación y generación de instrucciones

El proceso de verificación realizado es el siguiente: Antes de iniciar la generación aleatoria de instrucciones es necesario establecer los registros del procesador a un estado conocido, para lo cual se utilizan 62 instrucciones de carga de datos (lui y addi para los 31 registros disponibles) como inicialización del sistema. Luego, se restringe la aleatoriedad a las instrucciones de cada familia y a las 3 instrucciones de escritura y se envía un número fijo de instrucciones aleatorias, las instrucciones de lectura y escritura se ejecutan a una dirección fija para evitar una posible excepción del procesador si la dirección no está alineada. Finalmente, se escriben los datos de los 31 registros a memoria como una verificación extra del estado completo del procesador. Al ejecutar la escritura, el modelo de ejecución compara el estado de sus 31 registros mostrando algún error escondido en el estado interno del procesador.

La Tabla I presenta los resultados del proceso de verificación y las familias de instrucciones utilizadas. Para la verificación de la interacción de las instrucciones fueron necesarias 15 millones de instrucciones debido a la alta cantidad de escenarios posibles para las 3 etapas del *pipeline* con las 28 instrucciones utilizadas. La tabla muestra que se logra en el peor de los casos un porcentaje de cobertura de 99.1% garantizando la correcta operación de las funcionalidades analizadas.

Tabla 1.

Resultados de Verificación

Familia de instrucciones	**Número de errores encontrados**	**Porcentaje de cobertura**	**Numero de instrucciones por programa**	**Instrucciones**
Operaciones ALU con valor inmediato	0	100	123	LUI - ADDI - ANDI - ORI XORI - SLLI - SLTI SLTIU - SRLI - SRAI
Operaciones ALU entre registros	0	100	113	ADD - SUB - AND - OR XOR - SLL - SLT SLTU - SRL - SRA
Operaciones de lectura y escritura	0	100	123	LB - LH - LW - LBU LHU - SB - SH - SW
Interacción de 3 instrucciones (*pipeline*)	0	99.1	15e6	Todas las anteriores

4. Conclusiones y trabajo futuro

El aprendizaje de las capacidades de SystemVerilog que ofrece la Metodología Universal de Verificación, no sólo ofrece nuevas herramientas para el diseño de entornos de verificación sino también para un mejor diseño de sistemas digitales. Además de este nuevo conocimiento, la elaboración de este proyecto entrega un entorno de verificación completamente reusable para procesadores basados en RISC-V y un modelo de interfaz para buses AHB-Lite que permite su reutilización en procesos de simulación y verificación de sistemas digitales que utilicen este protocolo.

Los resultados de verificación para las instrucciones seleccionadas, muestran que el porcentaje de cobertura es de más del 99% para la interacción de las familias de instrucciones. Esto asegura el correcto funcionamiento del procesador en casi todo el espacio funcional verificado, y permite la fácil extensión si se desea verificar otras funcionalidades.

Para el desarrollo de nuevas versiones del procesador Olinguito, se completará el modelo de ejecución del procesador con instrucciones que permitan la modificación del contador de programa y se modificarán las restricciones de aleatoriedad para una verificación más profunda.

Referencias Bibliográficas

Accellera. (2014). *Universal Verification Methodology (UVM) 1.2 Class Reference*. Recuperado de http://www.accellera.org/images/downloads/standards/uvm/UVM_Class_Reference_Manual_1.2.pdf

Accellera. (2015). *Universal Verification Methodology (UVM) 1.2 User's Guide*. Recuperado de http://www.accellera.org/images//downloads/standards/uvm/uvm_users_guide_1.2.pdf

ARM. (2006). *AMBA 3 AHB-Lite Protocol v1.0 Specification*. Recuperado de https://goo.gl/5ZFdCV

Bromley, J. (2013, Septiembre). *If SystemVerilog Is So Good, Why Do We Need the UVM? Sharing Responsibilities between Libraries and the Core Language*. Documento presentado en 2013 Forum on Specification & Design Languages (FDL), Paris. IEEE.

Duran, C., Rueda, D. L., Castillo, G., Agudelo, A., Rojas, C., Chaparro, L., Hurtado, H., Romero, J., Ramirez, W., Gomez, H., Ardila, J., Rueda, L., Hernandez, H., Amaya, J., y Roa, E. (2016, Febrero). *A 32-bit RISC-V AXI4-lite bus-based microcontroller with 10-bit SAR ADC*. Documento presentado en 2016 IEEE 7th Latin American Symposium on Circuits Systems (LASCAS), Florianopolis. doi: 10.1109/LASCAS.2016.7451073

Ramírez, W. (2016). *A universal verification methodology for an lpddr3 memory* (tesis de pregrado). Recuperado de http://tangara.uis.edu.co/biblioweb/tesis/2016/165645.pdf

Salemi, R. (2013). *The UVM Primer: A Step-by-Step Introduction to the Universal Verification Methodology*. Estados Unidos: Boston Light Press.

Sutherland, S., y Mills, D. (2003, Marzo). *HDVL+=(HDL & HVL) SystemVerilog 3.1 The Hardware Description AND Verification Language*. Documento presentado en 2003 Synopsys Users Group (SNUG) Conference, San Jose. Recuperado de http://www.sutherland-hdl.com/papers/2003-SNUG-paper_SystemVerilog.pdf

Waterman, A. y Asanovic, K. (Eds.). (2017). *The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 2.2*. Recuperado de https://riscv.org/specifications/