Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Electrónica

Desarrollo de un Ambiente de Verificación para una Unidad Lógica Aritmética mediante la Metodología Universal de Verificación.

Informe de Proyecto de Graduación para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura

Pablo André Valenciano Blanco

Cartago, Junio de 2018

INSTITUTO TECNOLÓGICO DE COSTA RICA

ESCUELA DE INGENIERÍA ELECTRÓNICA

PROYECTO DE GRADUACIÓN

ACTA DE APROBACIÓN

Defensa de Proyecto de Graduación
Requisito para optar por el título de Ingeniero en Electrónica
Grado Académico de Licenciatura
Instituto Tecnológico de Costa Rica

El Tribunal Evaluador aprueba la defensa del proyecto de graduación denominado Desarrollo de un Ambiente de Verificación para una Unidad Lógica Aritmética(ALU) mediante la Metodología Universal de Verificación (UVM), realizado por el señor Pablo André Valenciano Blanco y, hace constar que cumple con las normas establecidas por la Escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal Evaluador

Ing. Aníbal Ruiz Barquero

Profesor lector

Ing. Esteban Baradín Méndez

Profesor lector

Ing. Roberto Molina Robles

Profesor asesor

18 de Junio, 2018. Cartago

Declaratoria de autenticidad:

Yo Pablo Valenciano Blanco, cédula 1-1572-0043, estudiante activo de ingeniería en Electrónica del Instituto Tecnológico de Costa Rica, carné 201242320 me comprometo a realizar este proyecto de graduación de manera íntegra con conocimientos propios, denotando claramente en la bibliografía de donde se obtuvo la información cuando sea necesario indicarlo.

Pablo André Valenciano Blanco

Resumen

Se desarrolla el presente proyecto en el Instituto tecnológico de Costa Rica con el objetivo de colocar en el mercado un producto que proporcione un mejor control a los dispositivos médicos implantables activos (AIMD por sus siglas en ingles), con los profesores a cargo Roberto Molina y Alfonso Chacón del Departamento de Electrónica.

Iniciando la realización del proyecto, ya habiendo superado las etapas de investigación y costos de presupuesto, los encargados se decidieron en colocar dicho proyecto, en la tabla de posibles proyectos de graduación para la obtención del título de Licenciatura en Ingeniera de Electrónica.

Ya con los estudiantes necesarios para dar inicio del proyecto, se explican los conceptos generales del proyecto y lo que se desea terminar en estas primeras 16 semanas, de parte de los estudiantes. Siendo el objetivo primordial la terminación de una arquitectura RISC-V, y esta que cumpla exhaustivas pruebas de funcionalidad, para corroborar el buen funcionamiento de todas las partes que la componen.

Para ello los profesores dividen al grupo de estudiantes en tres partes de suma importancia cada una de ellas: Diseñadores, Verificadores de Funcionalidad y Diseñadores de Pruebas Físicas. Donde cada diseñador tendrá un bloque de trabajo y un verificador que corrobore que las especificaciones de diseño se cumplan.

Cuya función asignada a mi persona es la verificación funcional del bloque denominado ALU por sus siglas en inglés o Unidad Lógica Aritmética, encargada de realizar distintas funcionalidades aritméticas, lógicas y comparativas.

Palabras Clave: RISC-V, Metodología Universal de Verificación (UVM), System-Verilog, Unidad Lógica Aritmética (ALU)

Abstract

With the aim of the Technological Institute of Costa Rica to place on the market a product that provides better control to the implantable medical implantable devices (AIMD for its acronym in English), with the professors in charge Roberto Molina and Alfonso Chacón of the department of electronics.

Starting the realization of the project, having already passed the stages of research and budget costs, the managers decided to put this project, in the table of possible graduation projects for obtaining the title of Bachelor in Electronics Engineering.

With the students needed to start the project, the general concepts of the project and what you want to finish in these first 16 weeks, from the students are explained. Being the primary objective the completion of a RISC-V architecture, and this one that fulfills exhaustive tests of functionality, to corroborate the good functioning of all the parts that compose it.

For this, the teachers divide the group of students into three parts of great importance, each one of them: Designers, Functional Verifiers and Physical Test Designers. Where each designer will have a work block and a verifier that corroborates that the design specifications are met.

Whose function assigned to my person is the functional verification of the block called ALU by its acronym in English or Arithmetic Logical Unit, responsible for performing various arithmetic, logical and comparative functions.

Keywords: RISC-V, Universal Verification Methodology (UVM), SystemVerilog, Arithmetic Logic Unit (ALU).

Dedicatoria

Le dedico este trabajo de graduación a mis seres queridos más cercanos como lo son mi familia en especial a mi madre Laura Gabriela Blanco y a mi padre Manuel Francisco Valenciano, que además de heredarme el apellido me otorgaron las herramientas para avanzar en el camino de la vida.

En parte de mi dedicatoria se la otorgó a mi tío, padrino y ángel que me ve desde del cielo Frank Blanco, que estuvo a mi lado desde el inicio de mi vida y hasta el fin del suyo, otorgándome la confianza, las ganas de trabajar y sobre todo el valor de no rendirse ante la adversidad. Enseñándome que de nada sirve el talento si no se hace un esfuerzo máximo que lo acompañe.

También le dedico este trabajo realizado con sangre y sudor a aquellas personas que tuvieron la paciencia de enseñar, evaluar y comprender a todos los compañeros que hoy se gradúan, en especial a los de la Escuela de Ingeniería en Electrónica. Y con especial cariño de mi parte a los profesores e ingenieros: Aníbal Coto, Néstor Hernández y Roberto Molina, que me mostraron que la electrónica además de su importancia puede llegar a ser divertida y ser llevada como beneficiaos a muchas más personas.

Agradecimiento

Me gustaría agradecer a mis familiares, desde que entré al Tecnológico de Costa Rica me preguntan mi estado en la carrera, que tal me va y si voy pasando los cursos. Y con el fin de hacerlos sonreír, con la frente en alto les respondo: “Estos han sido los mejores años de mi vida, pues obvio que estoy bien.”

También quiero dar mi aprecio a todos los compañeros de carrera que me acompañaron durante mi camino, no solo a aquellos que están entregando esperanzados sus proyectos de graduación, sino también a aquellos compañeros de carrera que compartimos cursos, trabajos y proyectos.

Para finalizar doy gracias a esta hermosa institución que desde el día que ingresé me abrieron sus puertas y me invitaron a ser cada día mejor, aceptando dicho reto llegué a ser lo que soy ahora, un profesional de calidad y una persona con la ética para hacer lo que es debido y devolverle todo lo brindado a este hermoso país, y porque no, al resto del mundo.

ÍNDICE GENERAL

Contenido

INDICE DE FIGURAS

INDICE DE TABLAS

Capítulo 1: Problema y Solución seleccionada

1.1 Problema existente e importancia de solución

El proceso de diseño de un chip es complejo, primero se deben conocer los requerimientos para definir la arquitectura y las especificaciones técnicas. Una vez que se cuente con esta especificación se procede a crear los componentes necesarios utilizando un lenguaje de descripción de hardware (HDL), cuando el código está listo se somete a una verificación funcional para garantizar que se comporte de la manera esperada. Este proceso es sumamente importante ya que cualquier error que se encuentre puede ser corregido a nivel de código antes de la fabricación del chip lo cual implica ahorros importantes en su implementación.

Cualquier fallo no encontrado en esta etapa puede llevar a la creación de un producto defectuoso que puede generar pérdidas muy importantes a la empresa, además dependiendo de la aplicación, puede representar un riesgo para la salud de las personas.

De ahí la relevancia que adquiere el verificar cada uno de los componentes de un chip, no solo para garantizar su correcto funcionamiento, sino también la seguridad de sus usuarios. En el caso de una Unidad Lógica Aritmética (ALU), se debe comprobar que realice todas las operaciones de manera correcta y que siempre se obtenga el resultado esperado.

En resumen, el problema se define como la inexistencia de un ambiente de pruebas que permita verificar el funcionamiento correcto de una Unidad Lógica Aritmética -ALU- para su ulterior uso como parte de los nuevos dispositivos médicos implantables activos.

Para crear el ambiente de verificación se utilizará la metodología de verificación universal, ya que es un estándar en la industria. La ventaja de este método es que su estructura puede ser reutilizable, lo cual ahorra tiempo cuando se requiere verificar más de un diseño. En las figuras 1 y 2 se observa los elementos que lo conforman.

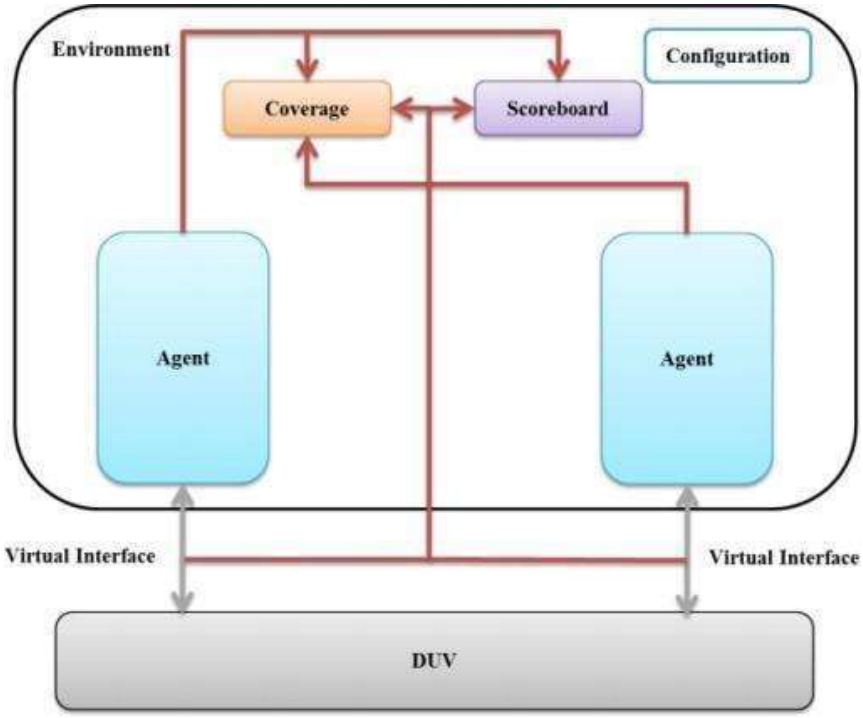

Figura 1 Estructura general de un ambiente de verificación [4]

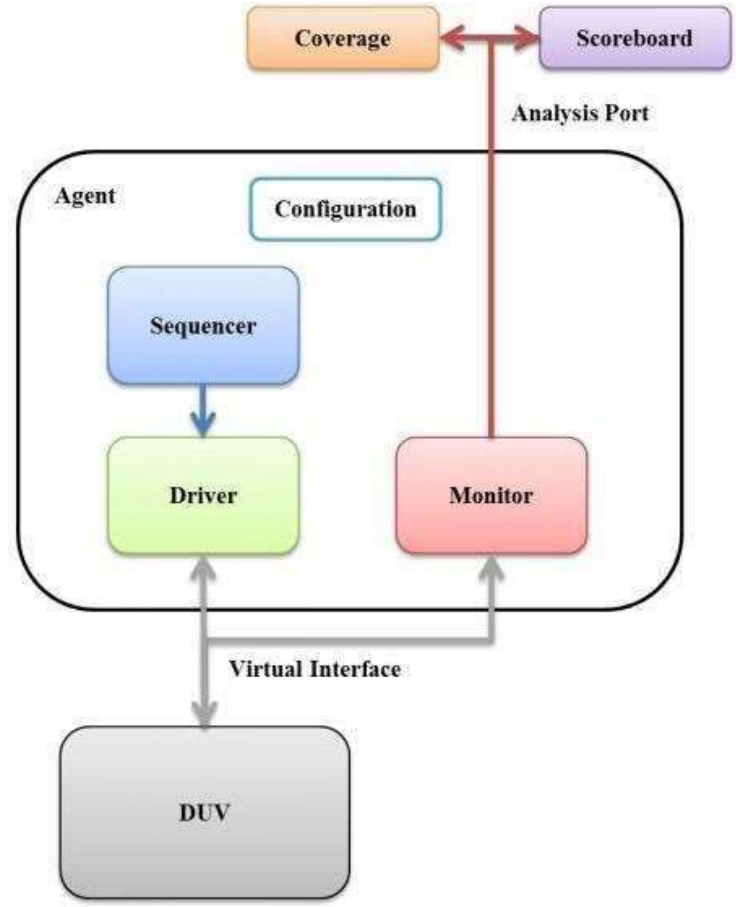

Figura 2 Estructura general de un Agente [4]

Tabla 1 Descripción de los bloques funcionales del ambiente de Verificación

Bloque	Función
Sequencer	Controlar el envío de paquetes de datos hacia el driver, genera estímulos de datos según el tipo de paquete que maneje y su funcionamiento depende de la secuencia que el agente le configure
Driver	Recibir los paquetes de datos provenientes del Sequencer para inyectarlos hacia el DUV

Monitor	Extraer información y el estado de la interfaz virtual a base de eventos, también puede recolectar y enviar información para análisis de cobertura y comprobación por medio de los puertos de análisis hacia otros componentes como el Coverage y el Scoreboard
Coverage	Contar y organizar los eventos que recibe mientras se está ejecutando una prueba, así como de generar las bases de datos de donde se analiza la cobertura funcional del dispositivo bajo prueba
Scoreboard	Cotejar el comportamiento deseado del DUV mientras una prueba está en ejecución e informa si ha ocurrido algún error o comportamiento inesperado

Cada bloque deberá ser adecuado para verificar el comportamiento de una ALU. También se deberá diseñar una interfaz virtual que permita establecer las conexiones de los puertos del diseño bajo pruebas (DUT) y el ambiente de pruebas. Esta interfaz deberá garantizar que los estímulos generados por el Sequencer sean transmitidos al puerto deseado, de lo contrario el scoreboard podría generar informes incorrectos de errores no asociados al funcionamiento de la ALU.

1.2 Solución seleccionada

La solución seleccionada plantea transmitir una señal de sincronización entre subestaciones basada en los sistemas de comunicación PDH y OP/AT los cuales son utilizados para comunicar disparos de protecciones.

Para garantizar que el diseño funciona correctamente se deberá comparar contra un

modelo de referencia conocido, el cual también será diseñado, dicho modelo deberá tener el mismo comportamiento que el DUT, para verificar que el DUT funciona según las especificaciones de diseño.

También se diseña un plan de verificación el cual presenta una descripción detallada del proceso de verificación, además debe responder las preguntas ¿qué se está verificando? y ¿cómo se va a verificar? Este documento debe presentar las siguientes secciones:

Tabla 2 Contenidos del plan de Verificación

Sección	Descripción
Herramientas requeridas	Contiene la especificación y lista del conjunto de herramientas de verificación, y describe el software necesario para realizar el procedimiento.
Riesgos y dependencias	Identifica amenazas críticas que pueden impedir el desarrollo de la prueba, por ejemplo, la dependencia de una nueva herramienta o la actualización de la actual que podría generar problemas de incompatibilidad.
Funciones por verificar	Lista de las funciones específicas que se van a evaluar, también se especifica las condiciones bajo las cuales son verificadas.

Pruebas y métodos específicos	Describe si el ambiente trata al DUT como una caja negra, y provee aspectos específicos sobre la estrategia de verificación, incluyendo la cantidad de aleatoriedad o determinismo usado en la simulación
Requerimientos de cobertura	Describe las metas de los estímulos utilizados, estas deberían cubrir todas las especificaciones funcionales del diseño y asegurar que los componentes de estímulo del ambiente hagan lo que deberían
Escenario de casos de prueba	Contiene la matriz de casos de prueba que consiste en una lista de todos los casos de prueba que servirán para verificar el diseño
Requerimiento de recursos	Incluye no sólo el recurso humano, sino también el computacional. Acá de igual manera se estima la cantidad de pruebas que se van a realizar durante todo el ciclo de verificación
Detalles del programa	Se crea y se documenta una línea del tiempo para cada una de las actividades de verificación, para esto se toman en cuenta los recursos disponibles

Capítulo 2: Metas y objetivos

2.1 Meta

Crear un ambiente de verificación que permita a los investigadores académicos garantizar el funcionamiento esperado de cualquier diseño bajo prueba al cual se someta.

2.2 Objetivo general

Desarrollar un ambiente de verificación para una Unidad Lógica Aritmética, que ejecute las operaciones más sencillas en una arquitectura RISC-V de 32 bits.

- Indicador: El ambiente realiza las pruebas necesarias que permiten determinar si la ALU se comporta de acuerdo con sus especificaciones funcionales.

2.3 Objetivos específicos

Diseñar los bloques funcionales del ambiente de verificación

- Indicador: Los bloques cumplen con las funciones descritas en la tabla 1.

Crear la interfaz virtual entre el ambiente de verificación y el diseño bajo prueba.

- Indicador: La interfaz conecta cada uno de los puertos del diseño con su respectivo driver y permite una conexión ininterrumpida entre ellos, además garantiza que las señales enviadas y recibidas lleguen a su destino correspondiente.

Diseñar el modelo de referencia de la ALU.

- Indicador: El modelo de referencia se comporta de la misma manera que la ALU de acuerdo con sus especificaciones funcionales por medio de la scoreboard

Capítulo 3: Marco teórico

En el presente capitulo se muestra de forma introductoria ciertos conceptos que introducirán al lector de lo aprendido durante el desarrollo del proyecto. Estos conceptos van desde el lenguaje de descripción de hardware, el estándar empresarial a lo que corresponde al ambiente de verificación y hasta una breve explicación del funcionamiento de la ALU en la arquitectura RISC-V.

3.1 System-Verilog

El lenguaje System-Verilog [1] ha sido llamado el primer descriptor de hardware y lenguaje de verificación a nivel de la industria, esta combina las características del descriptor de hardware utilizado en Verilog y VHDL, junto a las características especializadas de los lenguajes de verificación y junto con las herramientas de programación C y C++.

El descriptor de hardware System-Verilog se convirtió en un estándar de la IEEE en el año 2005 (IE 1800), siendo actualizado en el 2009 y ahora su desarrollo se encuentra a cargo del grupo Acelera, generando en sus usuarios la implementación de prácticas y aplicaciones del sistema. System-Verilog es llamada a ser la sucesora de Verilog y competir a nivel de mercado contra VHDL y SystemC.

El lenguaje System-Verilog está basado en orientación de objetos por lo que, hay que comprender ciertos componentes básicos como: Clases, Heredación de clases, Interfaces, Métodos variables y estáticos, parametrización de clases y estructuras. A modo de repaso se consideró ver los videos en YouTube del usuario Theo UVM Primer y cuyo primer video se titula Chapter 1. Introduction and Device Under Test.

3.2 Metodología Universal Verificación (UVM)

La metodología universal de verificación [4] (UVM) es definida como el estándar para los procesos de verificación funcional. Varias empresas como Intel, HP y universidades

internacional como la católica de Uruguay, utilizan el estándar UVM para la revisión y verificación de muchos diseños bajo prueba.

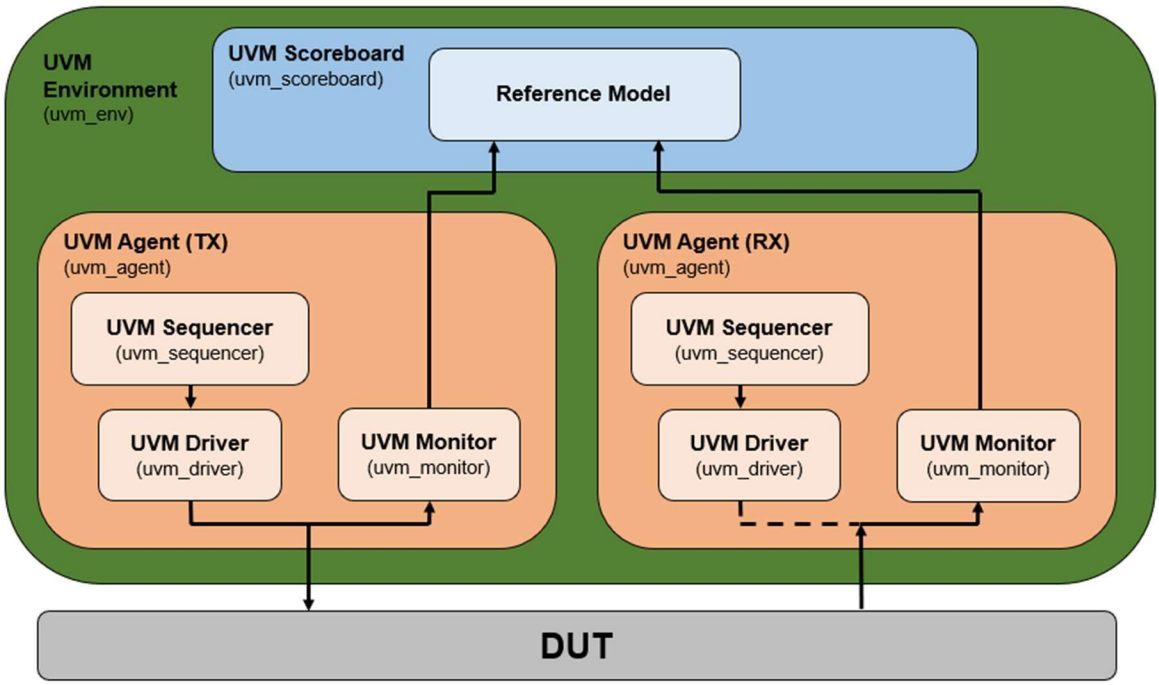

Figura 3 Esquema del ambiente UVM típico de 2 agentes. [7]

Siendo el diagrama estructural de UVM como una unión de elementos más simples de una única función cada una. Este diagrama se muestra en la Figura 3, lo cual detalla tanto el UVM Environment como al DUT, y su unión se da por medio del UVM Testbench.

3.2.1 UVM Testbench

Es el nivel más alto de la arquitectura UVM, esta se divide en 2 instancias: DUT y UVM Environment. Siendo el DUT el elemento al que se le realizarán las pruebas y el UVM Environment se configura de manera que pueda generar la mayor cantidad de pruebas posibles y reportar cualquier error en el DUT.

3.2.2 UVM Environment

Es desarrollado bajo el lenguaje System-Verilog explicado anteriormente, es configurado con respecto al DUT. Este archivo se conforma por diferentes bloques:

- UVM Agents
- UVM Scoreboard
- Interfaces virtuales

Las salidas del UVM Environment son las entradas del DUT. A su vez, las salidas del DUT son las entradas del UVM Environment.

3.2.3 UVM Agent

Los UVM Agent se encuentran ubicados dentro del UVM Environment, usualmente se utilizan dos agentes con el fin de que uno controle las entradas al DUT y el otro que sea para monitorizar las salidas del DUT. El encargado de estimular las entradas del DUT recibe el nombre de TX UVM Agente o Active UVM Agent. Dado el estímulo generado por el Active Agent, el DUT se excita y genera salidas que el otro UVM Agent recibe y se encarga de monitorizarla, a este agente se le domina comúnmente como RX UVM Agent o Passive UVM Agent. Ambos agentes envían información al UVM Scoreboard.

Ambos UVM Agents están compuesto de tres bloques: UVM Driver, UVM Monitor, UVM Sequencer. Los UVM Agents tienen 3 parámetros esenciales: El identificador de prueba, Bandera de TX y Bandera de RX. El identificador indica la secuencia o prueba a realizar. Y las banderas se utilizan siguiendo la Tabla 3.

Tabla 3 Modos del Agente [7]

Modo	Bandera RX	Bandera TX	Funcionalidad
Apagado	0	0	Driver: Apagado Monitor: Apagado Sequencer: Apagado

Transmisión	0	1	Driver: Encendido Monitor: Encendido Sequencer: Encendido
Recepción	1	0	Driver: Apagado Monitor: Encendido Sequencer: Apagado
Estado Inválido	1	1	Driver: Apagado Monitor: Apagado Sequencer: Apagado

3.2.4 UVM Driver

Es el encargado de manejar y estimular y estimular las entradas del DUT, para eso es necesario implementar una interfaz virtual hacia el DUT. En palabras más sencillas es el encargado de traducir cualquier cosa a algo que el DUT entienda y pueda ser monitoreado.

Éste requiere de 2 entradas: La primera es la secuencia enviada por el secuenciador, y la segunda es la bandera TX que solo en caso de estar activa, realiza su función de traducir hacia el DUT. En los demás casos se considera que su conexión con el DUT está en abierto.

3.2.5 UVM Monitor

Es un bloque pasivo que se encuentra en ambos agentes. Su propósito es observar y monitorizar la información que recibe de las interfaces virtuales que se conectan al DUT.

En ambos modos (TX y RX), la única diferencia en el monitor es de donde recibe la información y hacía que parte del scoreboard se transmite la información monitoreada. Para el caso TX Mode recibirá del driver y enviará a la izquierda del scoreboard, y en RX Mode recibirá de las salidas del DUT y transportará dicha información a la parte derecha del scoreboard. El monitor solo se apagará cuando ambas banderas se encuentren apagadas.

Es importante mencionar que otra función importante del monitor es reportar información para propósitos de verificación, Con el objetivo de indicar lo que se ha obtenido y lo que aún no. Se deben colocar los puntos de cobertura en las interfaces virtuales.

3.2.6 UVM Sequencer

En este bloque, se implementan las diferentes pruebas con el fin de buscar bugs en el DUT. Generalmente contiene un banco con diferentes pruebas. El secuenciador, como su nombre señala, genera secuencias de señales al driver, donde son transformadas como se explicó anteriormente.

Sus dos entradas son: El identificador de prueba y la bandera TX. El identificador proviene del UVM Environment. El identificador busca en el banco de secuencias y genera el patrón seleccionado. Y el elemento solo se encuentra activo cuando la bandera TX se encuentra activa.

3.2.7 UVM Scoreboard

Es el bloque diseñado para realizar el análisis de datos. Dicho bloque adquiere la información de ambos monitores, y compara la respuesta esperada contra la respuesta recibida del DUT, la respuesta esperada se obtiene del modelo de referencia.

En caso de lograr que en la cadena de información de una secuencia no se cometa ni un solo fallo, entonces dicha secuencia será un éxito y se probará la siguiente, en caso de encontrar un fallo en alguna de las secuencias se deberá señalar que secuencia lo generó y reportar el bug, con el propósito de que el diseñador lo repare y se vuelva a hacer una revisión de éste. En otras palabras, el scoreboard revisa que las especificaciones descritas por el arquitecto en el DUT sean revisadas por distintas pruebas. El scoreboard debe generar un reporte que detalle las diferentes pruebas y detallar los casos de éxito.

3.2.8 Reference Model

Es el bloque que simula al DUT, donde las salidas del monitor proveniente del agente activo serán transformadas y genera la respuesta que se espera que el DUT realice.
Este modelo es único para cada DUT.

3.3 Arquitectura RISC-V

La meta del proyecto de investigación para el año 2018 es completar la arquitectura mostrada en la figura 4. Para ello varios grupos de trabajo se dividen en diseñadores y verificadores para cada uno de los elementos complejos que componen la arquitectura RISC-V, específicamente aquel con el set de instrucciones 32I. [9]

Este set implica la construcción de bloques funcionales que puedan realizar operaciones con registros, accesos a memoria, ejecuciones de saltos y la inserción de valores inmediatos. Cuyo tamaño por instrucción debe ser de 32 bits.

Cada instrucción se categoriza por el tipo de codificación que tiene, y según los elementos que cada una utiliza al ejecutar la operación. Las instrucciones se dividen según la tabla 4. En este proyecto dará énfasis al elemento con el nombre: ALU, que utiliza los grupos de instrucciones: B-type, R-type, r-type, I-type, Machine Mode I-type.

Tabla 4 Diferentes tipos de instrucciones agrupadas por codificación

CODIF[6:0]
0XXX000 –> U-type Instruction
1XXX000 –> J-type Instruction
XXXX010 –> B-type Instruction
XXXX011 –> S-type Instruction
XXXX100 –> I-type Instruction
XXXX101 –> R-type Instruction (SLLI,SRLI,SRAI)
XXXX110 –> I-type jalr
XXXX111 –> r-type instructions (add,sll.slt,sltu,xor,srl,or,and,sub,sra)
XXXX001 –> Machine-Mode-I-type Instruction

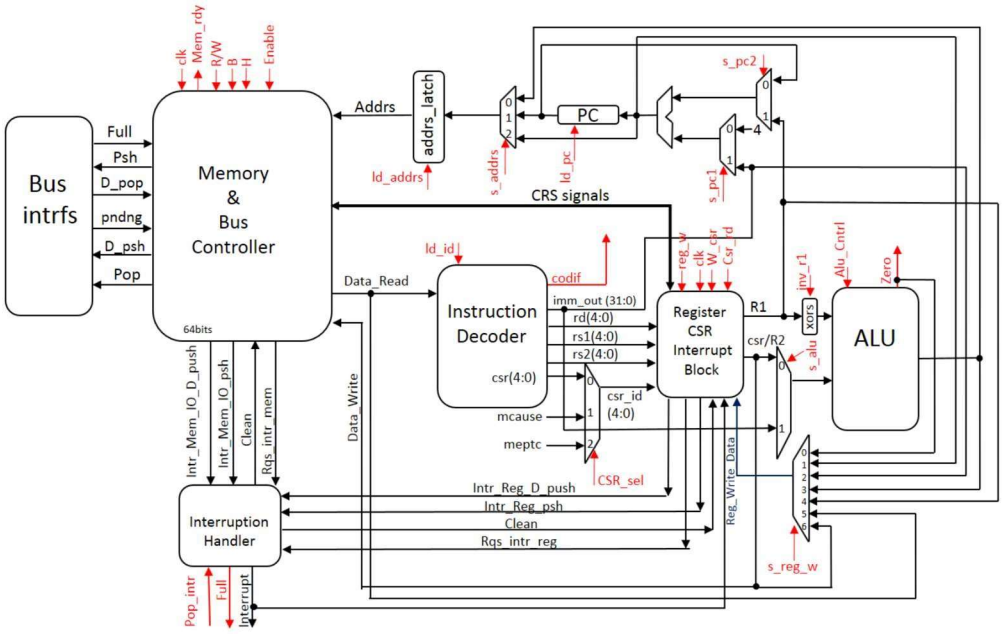

Figura 4 Diagrama Básico de la micro arquitectura RISCV 32I

3.4 Descripción de la ALU

El Diseño Bajo Prueba (DUT por sus siglas en inglés) para este proyecto será, el mencionado en el título, la Unidad Lógica Aritmética [2] (ALU por sus siglas en inglés). Siendo este diseño completamente combinacional, es decir independiente a una sincronización y Lineal Invariante en el Tiempo (LTI), dado que sus respuestas siempre son las mismas para las mismas entradas sin depender del tiempo que éste produzca.

La ALU es un banco de operaciones, estas pueden ser de tipo lógicas, aritméticas o comparativas. Las operaciones lógicas son en sí un banco de compuertas lógicas aplicando su función lógica, véase en la tabla 4 las principales funciones lógicas. Las funciones aritméticas se definen como aquellas que realizan algún tipo de operación más compleja que conlleva la utilización de más de un componente lógico. Y, por último, las operaciones comparativas son aquellas que activan una bandera en caso de ser cierta la condición de comparación elegida.

Tabla 5 Especificaciones Generales de la ALU. [2]

Tipo de Función	Nombre de la Función	Código de Función	Result	Zero
Comparativas	EQUAL	0000	0	¿A==B?
	LESS THAN SIG	0001	0	¿A<B?
	LESS THAN UNS	0010	0	¿A<B?
	GREATER THAN SIG	0011	0	¿A>B?
	GREATER THAN UNS	0100	0	¿A>B?
Aritméticas	ADD SIG	0101	A+B	0
	ADD UNS	0110	A+B	0
	SUB UNS	0111	A-B	0
Desplazamientos	SHIFT LEFT	1000	A>B	0
	SHIFT RIGHT	1001	A<B	0
	SHIFT RIGHT ARITM	1010	A<<B	0
Lógicas	OR	1011	A|B	0
	XOR	1100	A^B	0
	AND	1101	A&B	0

3.4.1 Datos con signo o sin signo

Como se habrá visto en la tabla anterior no se nota una diferencia en los resultados cuando el nombre de la operación se distingue por ser sin signo (UNS) o con signo (SIG). Para ello se creó esta parte, con el fin de evitar confusiones a futuro. Para el proyecto se utilizará una ALU de 32 bits, pero para facilitar su comprensión se utilizarán 4 bits para el ejemplo.

De estos 4 bits, el bit más significativo será el del extremo izquierdo, este servirá para distinguir el signo del dato; si es requerido. A qué se refiere cuando es requerido, es el caso de las operaciones con signo, siendo de suma importancia para la obtención del resultado.

Para ello se utiliza un análisis denominado complemento a 2, este tomará el dato al cual se le requiera realizar el complemento a 2, este proceso negará todos los bits del dato y sumará 1, cambiando el signo del dato. El siguiente ejemplo ayudará su comprensión.

Tabla 6 Comparativa entre 2 datos con signo, sin signo y su complemento a 2

Tipo de Dato	Binario	Decimal
Dato Sin Signo	1101	13
Complemento a 2	0011	3
Dato Con Signo	1101	-3
Dato Sin Signo	0101	5
Complemento a 2	1011	-5
Dato Con Signo	0101	5

3.4.2 Operaciones Comparativas

Las funciones comparativas tienen como único propósito alzar la bandera Zero cuando la condición dictada por la función a utilizar sea cierta, y en caso de no serlo esta bandera será bajada. Punto importante para destacar es que las demás salidas deben mantenerse en estado bajo o cero según el tipo de salida.

Las operaciones comparativas se dividen en 2 tipos según la forma que se utilice el dato. Estas 2 formas son tomando los datos con signo o sin signo. Generando cambios en la bandera según la operación que se utilice. Las operaciones comparativas pueden ser mayor que, menor que o iguala, tomando el Dato A como referencia y el Dato B al que se le va a comparar.

3.4.3 Operaciones Aritméticas

Estas operaciones son las más complejas debido que son las que utilizan la mayor cantidad de recursos en el diseño, y genera la mayor cantidad de transiciones en las banderas y cambio en los bits de un resultado.

Entre las operaciones aritméticas a trabajar en la ALU son la suma con signo, sin signo y la resta sin signo. Estas pueden llegar a activar las banderas de Acarreo (Carry), Desbordamiento (Overflow) y Negativo (Negative). Son las únicas que utilizan la entrada Acarreo de Entrada (Cin), que es un elemento booleano que se utiliza para aumentar el tamaño de la ALU si el arquitecto desea realizar operaciones con mayor longitud numérica.

La bandera Carry para una arquitectura de 32 bits requiere en primera instancia los bits menos significativos de ambos datos y el Cin generando un acarreo de salida; a esto se le domina sumador completo (Full Adder), dada la ecuación de la figura 4, se continuará colocando en cascada sumadores completos hasta llegar a la cantidad de 32, donde sus entradas son el bit correspondiente de cada dato, y el acarreo de salida del sumador anterior; es decir para el sumador 7 sus entradas son el 7° bit de Data_A, el 7°bit de Data_B y el acarreo de salida del 6° sumador. Por lo tanto, la bandera de Carry corresponde al acarreo de salida del sumador 32. Para el caso de la resta es necesario usar la información del complemento a 2 del Data_B, en lugar del Data_B.

$$C_{\text{out}} = (A \cdot B) + (C_{\text{in}} \cdot (A \oplus B))$$

Figura 5. Ecuación de Acarreo [8]

Mientras, que la bandera Overflow es definida por los bits más significativos de las entradas como del resultado. Existe Overflow si los bits más significativos de ambas entradas son diferentes y a la vez se da, que el bit más significativo de la primera entrada es diferente al bit más significativo del resultado. Si se cumplen ambas condiciones existirá Overflow, e igual que el acarreo, si se da una resta el segundo dato de entrada sería el complemento de éste.

La bandera Negative solo indica el valor más significativo del resultado, sin importar si el valor sea positivo o negativo. Y en caso de que la operación sobrepase los 32 bits el valor será recortado a 32 bits.

3.4.4 Operaciones Desplazamiento

Estas operaciones son en sí aritméticas, pero más sencillas que las anteriores debido a que los efectos en los resultados son de forma más directa.

En este grupo se encuentran las funciones: Desplazamiento a la Izquierda, a la Derecha, y Derecha Aritmética. Las primeras dos colocan ceros según el lado que corresponda, y la última según cual sea el valor del bit más significativo se completará con dicho valor hacia la derecha. En el siguiente ejemplo mostrará la función de las tres operaciones.

Tabla 7 Demostración de Operaciones de Desplazamiento [6]

Nombre de la Función	Entradas	Salida
Desplazamiento Izquierda	A: 1001, B: 0001	0010
Desplazamiento Derecha	A: 1001, B: 0001	0100
Desplazamiento Der. Aritmética	A: 1001, B: 0001	1100

3.4.5 Operaciones Lógicas

Las operaciones lógicas son expresiones matemáticas cuyo resultado genera un valor booleano, pero en caso de datos tan grandes, como los que se van a trabajar son de 32 bits, se toma el primer bit de cada dato y se aplica la compuerta lógica designada obteniendo un valor booleano, se coloca este valor en la primera posición, y se aplica lo mismo para los 32 casos, obteniendo como resultado un bus de 32 bits de valores booleanos.

Las operaciones por trabajar en la ALU son: O (OR), Y (AND) y O Exclusivo (XOR). La operación OR genera un resultado de verdadero si alguna de sus entradas es verdadera, la operación AND genera verdadero si todas sus entradas son verdaderas, y para la operación XOR es verdadero si existe un número impar de verdaderos; en caso contrario los resultados de las operaciones serán falso. Los resultados para cada operación en un ejemplo de dos datos se observan en la tabla 8.

Tabla 8 Tabla Lógica Binaria para las operaciones OR, XOR, AND [6]

Dato A[i]	Dato B[i]	Result (OR)	Result (XOR)	Result (AND)
0	0	0	0	0
0	1	1	1	0
1	0	1	1	0
1	1	1	0	1

Es importante decir que una ALU se encuentra completa cuando se pueden obtener resultados de otros tipos de operaciones a partir de las que se tienen. Como por ejemplo se puede realizar una función NOT que niega la entrada a partir de una XOR cuya una de las

entradas sea una cadera de puros unos. Y con la combinación de la NOT con la AND u OR se pueden diseñar las operaciones lógicas NAND y NOR respectivamente.

3.5 Pseudocódigo

Este punto va a tratar una explicación de alto nivel de los componentes de la UVM, procurando utilizar conceptos sencillos. Esto con el objetivo de que personas ajenas del área de ingeniería sean capaces de comprender lo que hace cada uno de los principales componentes.

3.5.1 UVM Sequencer

El Sequencer o Secuenciador en español, tiene como trabajo diseñar secuencias; de ser posible aleatorias, de un tamaño igual a la cantidad de entradas del dispositivo al que se va a realizar la prueba, y tantas repeticiones posibles que generen la mayor cantidad de combinaciones en las entradas.

```
Pseudocódigo Secuenciador
Crea el bloque físico del Secuenciador
Repetir más de 5000 veces
Inicio
    Crea un elemento que posea las variables a aleatorizar
    Espera a que se conceda el permiso
    Se Aleatorizan las variables
    Envía hacia el Driver
    Espera que todos los elementos sean enviados
Fin: De la Repetición
```

3.5.2 UVM Driver

La función del driver es recolectar la información del Secuenciador y colocarla en un interfaz virtual, en si son conexiones que el diseño del dispositivo es capaz de comprender como cables y registros. Este cambio se realiza debido a que el secuenciador genera la información en lenguaje de programación; denominado C, y el diseño se crea por medio de descripción de Hardware, pudiendo ser Verilog o VHDL; de ahí la razón de usar System-Verilog para este proyecto.

Pseudocódigo del Driver
Crea el elemento físico Driver
Espera a recibir información del secuenciador
Transforma la información recibida
La coloca en el interfaz virtual
Notifica que el ítem llegó

3.5.3 UVM Monitor

El monitor es el encargado de recolectar la información de ambos lados del diseño, es decir las entradas y las salidas, y como función extra es el encargado de reportar el porcentaje de puntos cubiertos (Coverage) en la interfaz virtual.

Pseudocódigo Monitor
Diseña el bloque físico
Recolecta tanto las entradas como las salidas del diseño Reporta
el Coverage de las entradas y salidas.

3.5.4 UVM Reference Model

El modelo de referencia se encarga de imitar el comportamiento ideal del diseño, siguiendo las exigencias del arquitecto. Éste compara las salidas según las entradas en el

modelo de referencia con las salidas recolectadas por el monitor. Se aplica un sistema lógico de manera que, si sus cinco salidas son iguales se evitará informarlo y solo mostrará los resultados discrepantes, para reportar los bugs del diseño o del ambiente de verificación.

Pseudocódigo Modelo de Referencia
Recolecta la información del monitor (Entradas y Salidas)
Coloca las entradas en un modelo de referencia
Obtienen salidas del modelo de referencia
Compara las salidas del modelo de referencia contra las salidas del DUT
Si son diferentes alguna de sus salidas
    Muestra la información del bug, las entradas que las produjeron y las discrepancias de las salidas.

3.6 Conexiones entre módulos

Los módulos anteriormente mencionados, por si solos no pueden diseñar el ambiente de verificación UVM, necesita de todas sus partes conectadas entre sí para que la función de todos los dispositivos trabaje de manera correcta. En el UVM existen 4 bloques que lo componen diferentes elementos: UVM Agent, UVM Scoreboard, UVM Environment y UVM Testbench.

3.6.1 UVM Agent

El agente interconecta los componentes del Secuenciador, Driver y Monitor, y los coloca de la forma como se muestra en la figura 2, con el fin de entregar dicha información a los puntos de cobertura y scoreboard.

Según el modo de operación que se encuentre el agente, éste puede llegar a comportarse de manera diferente. Para el proyecto hay 2 modos de operación explicados en

los conceptos básicos, éstos pueden ser activos o pasivos, para el caso del activo todos los elementos se encuentran conectados como se ve en la figura 2, y en modo pasivo el secuenciador y el driver se desconectan del Monitor y del DUT.

3.6.2 UVM Scoreboard

El Scoreboard por otra parte está compuesta de un solo elemento, este elemento es el modelo de referencia como se mencionó en la sección correspondiente su misión es imitar el comportamiento del DUT, siendo excitada por las mismas entradas y generando la misma cantidad de salidas. El Scoreboard recibe información tanto del agente activo como del pasivo a través de sus monitores. Comparando así las salidas del DUT con los recibidos del modelo de referencia. Si ambas salidas discrepan, se deberá recibir aquellos datos, entradas y salidas, que generaron dicha discrepancia, y con ello revisar si hay un problema en el DUT o más bien en el ambiente de verificación. Esto será reportado en una lista de bugs que tanto el verificador como el diseñador tendrá a su disposición, para la revisión y corrección necesaria.

3.6.3 UVM Environment

El Environment o ambiente, en si es el ambiente de verificación que se va a trabajar, para este caso se elaborará con el método de verificación universal, UVM.

UVM tradicionalmente trabaja con dos agentes, puede llegar a ser tres, uno de ellos activo; secuenciador, driver y monitor activos, y el otro pasivo, solo el monitor activo. Siendo estos dos agentes comunicados por medio del scoreboard, y revisando los procesos de aleatoriedad del proceso, se revisarán los puntos de cobertura o reporte de Coverage, siendo formado así el ambiente de pruebas.

3.6.4 UVM Coverage

El Coverage [3] o porcentaje de Cobertura se define como la proporción de resultados que pasan por la interfaz con respecto a la totalidad de las posibilidades, se puede calcular el porcentaje de cobertura tanto a las entradas como a las salidas del DUT. A este tipo de Coverage se le define como Functional Coverage.

Mientras que el Code Coverage es aquel que reporta la cantidad de líneas dentro del DUT, que se instanciaron con el fin de revisar que todos los recursos diseñados por el encargado general de la ALU sean realmente utilizados, y no se presenten desperdicios de recursos.

Muchos verificadores utilizan este Coverage, como punto final, para que no se realicen repeticiones de pruebas o secuencias innecesarias. Para su diseño en el plan de verificación UVM se crea un covergroup o grupo de cobertura. Ahí mismo se instancian los coverpoint o puntos de cobertura, que definen a que entrada o salida corresponde cada uno de los puntos de cobertura. Dentro de los puntos de cobertura se definen bins o contenedores que almacenan la totalidad de las posibilidades, cada contenedor puede poseer un dato o un rango de datos, y en el momento que una secuencia atraviese la interfaz, busca en la pila de contenedores aquel que contenga dicho valor, y este contenedor se constituye como cubierto, y así para cada secuencia, con la meta de que gran parte de los contenedores sean al fin cubiertos.

El porcentaje de cobertura para cada bus de datos seria con este concepto la cantidad de contenedores cubiertos contra la totalidad de contenedores. Los coverpoint también se pueden definir de forma cruzada, esto implica que la totalidad de posibilidades de ser cubiertos es la multiplicación de ambas posibilidades de forma individual, ya que se crean automáticamente todos los contenedores posibles, donde cado uno de estos contendrá dos contenedores: uno de un coverpoint y otro del segundo coverpoint.

3.6.5 UVM Testbench

Finalmente, se verá la conexión del ambiente UVM, con el DUT, será con la creación de interfaces virtuales que se conecten a las entradas y salidas del DUT, y en el ambiente se observará primero en el agente activo por medio del driver hacia las entradas del DUT y al monitor, y luego se tendrán estas interfaces en el agente pasivo como la conexión de las salidas del DUT hacia su monitor.

El testbench tiene la responsabilidad de seleccionar la prueba que se desea testear, con el fin de revisar ciertas condiciones específicas. Esto genera un estímulo en el secuenciador y entregara resultados para dicha secuencia.

Capítulo 4: Implementación

4.1 Diagrama a implementar UVM

El siguiente diagrama de flujo presentado en la figura 6, muestra el flujo de la información en el programa, color naranja. Siendo dominada por las señales de control, principalmente los relojes del driver y el monitor, color azul.

Se inicia creando todos los elementos físicos en el aplicativo y la interfaz virtual que conecta hacia el DUT. Ya verificado que todo se encuentra conectado se envía una secuencia de información, es decir una de todas las posibilidades de entrada, a través del secuenciador hacia el driver, transformando la información a señales eléctricas que comprenda el DUT y puedan ser recibidas por el monitor.

En el monitor también se recolectan las señales de salidas del DUT, y éstas corroboran que el DUT funciona parcial o totalmente, con la ayuda del modelo de referencia, encontrado inserto en el scoreboard. El scoreboard revisará la equidad de dicha secuencia y mostrará, en caso de alguna discrepancia, los datos que provocaron dicho error. En casos de generarse un error o no, se reiniciará el proceso de diseñar una nueva secuencia aleatoria

siguiendo los pasos nuevamente, esto hasta un cumplir un número de repeticiones o cumplir un porcentaje de cobertura.

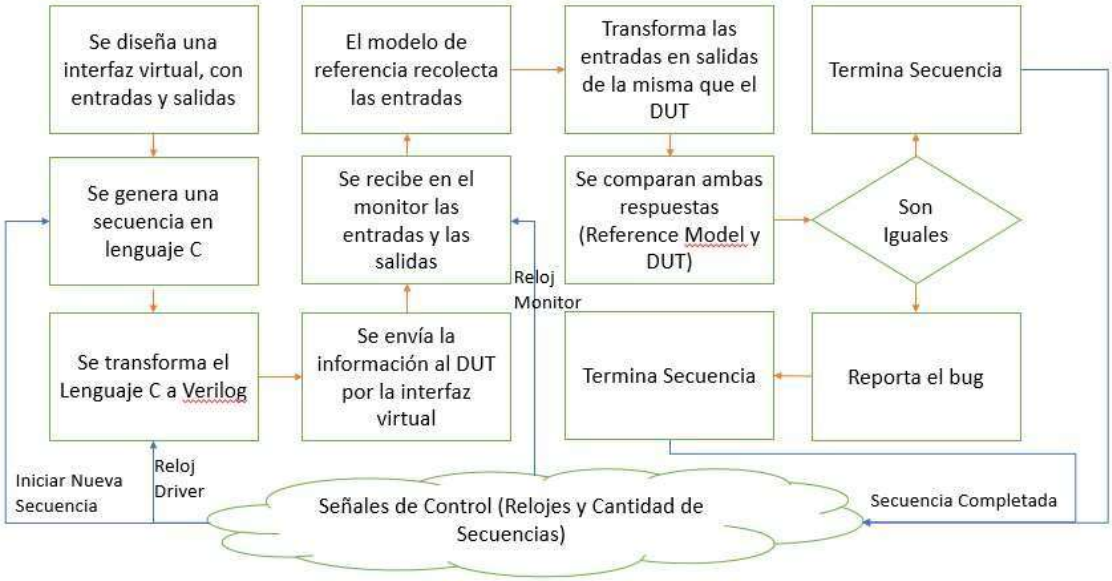

Figura 6 Diagrama de Flujo del ambiente de Verificación UVM

4.2 Modelo de Referencia

El modelo de referencia es aquel elemento que debe imitar lo más posible al DUT al cual se ejecutan las diferentes pruebas, que se encuentran en el apéndice del plan de verificación en la sección de escenario de casos de pruebas.

Esté se moldea a partir de diferentes casos, según la señal de control (ALU_CTRL) definida. Se realizó la división de los casos con el uso de los condicionales “if” y “else”. Cada caso contiene la asignación de las banderas y del resultado. Y se crean variables enteras de los tipos: sin signo “unsigned” y con signo “signed de ambas entradas de datos y el complemento a 2 del segundo dato (-Data_B).

Para las operaciones comparativas se tiene que en caso de que se necesiten los valores con signo o sin signo para su comparativa se utilicen dependiente de la operación a

ejecutar. Para el caso de las comparaciones con el código 2 y 4 se utilizan sin signo y para los casos con código 1 y 3 se utilicen los valores con el signo, y para la igualdad es independiente la forma en que se muestren los valores, por lo que se utiliza de forma directa los adquiridos por el monitor del agente activo. Con el fin de decidir el valor que se le asigne a la bandera Zero, en caso de cumplir la condición asignada se coloca un uno y en caso contrario un 0. Tanto el resultado como las demás banderas se les asigna el valor de cero. Para el resto de las operaciones la bandera Zero se mantiene en cero.

Para las operaciones aritméticas se tiene que la obtención del resultado depende del tipo de dato que este tome. Para la suma con signo, se requieren de ambos valores con signo, para la suma sin signo se deben de usar los valores sin signo, y para la resta sin signo se utiliza el primer valor sin signo y el segundo como el complemento a 2 del segundo valor con signo. Y para las banderas se utilizan los bits que corresponden a los valores de entrada y salida de las operaciones. La bandera Negative es el bit más significativo del resultado, el Carry es el acarreo del ultimo sumador, el Overflow depende del bit más significativo de ambos datos entrantes y del bit 32 del resultado.

En el caso de las operaciones de desplazamiento, el resultado es el fin de desplazar el primer dato tanto como el segundo dato lo indique, para los desplazamientos lógico se insertan ceros, mientras para los desplazamientos aritméticos se insertan el valor del bit que se encuentre en el bit más significativo. En caso de que el segundo dato sea mayor a 32 se traslada tanto como el valor del residuo de la división del segundo con dato con el entero 32. La bandera Negative es la extensión del bit 32 del resultado.

Las operaciones lógicas no son más que colocar la compuerta correspondiente para cada caso entre ambos datos de forma directa. Para la OR se utiliza el símbolo |, para la XOR se hace uso de ^, y para la AND se coloca &.

Capítulo 5: Resultados y Análisis de Resultados

5.1 Bugs Encontrados

Los bugs o errores del sistema permiten tanto al diseñador del DUT como al verificador funcional, corregir las partes que uno u otro encontraron a la hora de realizar diferentes pruebas. Incluso, en algunas ocasiones se condicionan los datos entrantes, con el fin de tener una comunicación directa entre ambas partes. Para este proyecto se trabajó con un drive compartido bajo el nombre de: Verification Plan: CR_TEC_RISCV. Este plan se encuentra en el anexo A.3 Plan de Verificación.

Debido a que uno de los bugs encontrados se encuentra en el ambiente de verificación, es parte del trabajo del verificador solucionarlo, este bug se denomina: Flag Incomprehension. En español, Incomprensión de banderas, esto fue debido, en un principio, solo se tenía la tabla de la sección 3.4, y esta contiene solamente los valores de la bandera Zero y el resultado para las operaciones, siendo escaza la especificación del resto de elementos. La solución para cada una de las banderas se discutirá a continuación.

Para la bandera Negative se predijo que era una bandera exclusiva de las funciones aritméticas y solo cuando el resultado generaba un complemento a 2, es decir un signo negativo como se describió con anterioridad. Siendo esto erróneo y se configuró como una extensión del bit 32 del resultado, como el arquitecto tenía previsto.

Para la bandera Carry tanto el verificador como el diseñador tenían la forma de obtener la bandera Carry de forma incorrecta, por parte del verificador se pensó como el elemento sobrante de las sumas es decir el bit 33, de forma equivocada. Esto debido a que la bandera Carry depende de todos de los bits de entrada y realiza la suma de sus bits uno por uno, en caso de las restas se hace la realiza la mismas, pero con el complemento a 2 del segundo dato o Data_B.

El Overflow no se había podido trabajar debido a la falta de comprensión de éste, sino hasta que, con el diseñador, se encontró una fórmula lógica que toma los bits más

significativos tanto de las señales de entrada como del resultado final, y obtiene dicha bandera.

5.2 Resultados para cada operación

Se diseñaron secuencias específicas para cada operación donde se obligó a la señal de control a mantener un valor constante y los datos variantes. Para cada operación se mostrarán 3 ejemplos, que entregará el ambiente de verificación por medio del scoreboard. Se tendrá el valor esperado como aquel valor del modelo de referencia que tiene como entradas a las entradas del DUT, y el valor obtenido como las salidas recibidas del DUT dada la secuencia de entradas.

Las operaciones se definen como la tabla descrita en la sección 3.4: Descripción de la ALU. A modo resumen el código de las operaciones es:

0: Igualdad.
1: Menor que, sin signo.
2: Menor que, con signo.
3: Mayor que, sin signo.
4: Mayor que, con signo.
5: Suma con signo.
6. Suma sin signo.
7. Resta sin signo.
8: Desplazamiento izquierdo lógico.
9: Desplazamiento derecho lógico.
10. Desplazamiento derecho aritmético.
11: Operación Lógica OR.
12: Operación Lógica XOR.
13: Operación Lógica AND.

5.2.1 Comparativas

Estas operaciones como su nombre lo indica sirven para comparar entre datos. La ALU tiene 2 entradas de datos: Data_A, y Data_B, el Data_A se compara con respecto al Data_B. Y si se cumple la condición, la bandera Zero se levanta. Estas funciones por su característica de solo usar una salida, generalmente se utilizan para saltos en el controlador.

Tabla 9 Ejemplos de Operaciones Comparativas [6]

Entradas / Salidas	Dato AS: -329011843 Dato BS: -39842810 Dato AU: 3965955453 Dato BU: 4255124486	Dato AS: -835321783 Dato BS: 1234113092 Dato AU: 3459645513 Dato BU: 1234113092	Dato AS: 1990550577 Dato BS: 967648330 Dato AU: 1990550577 Dato BU: 967648330
Igual OP Code: 0.	Expected Zero: 0 Actual Zero: 0	Expected Zero: 0 Actual Zero: 0	Expected Zero: 0 Actual Zero: 0
Menor que con signo. OP Code: 1.	Expected Zero: 1 Actual Zero: 1	Expected Zero: 1 Actual Zero: 1	Expected Zero: 0 Actual Zero: 0
Menor que sin signo. OP Code: 2.	Expected Zero: 1 Actual Zero: 1	Expected Zero: 0 Actual Zero: 0	Expected Zero: 0 Actual Zero: 0
Mayor que con signo. OP Code: 3.	Expected Zero: 0 Actual Zero: 0	Expected Zero: 0 Actual Zero: 0	Expected Zero: 1 Actual Zero: 1
Mayor que sin signo. OP Code: 4.	Expected Zero: 0 Actual Zero: 0	Expected Zero: 1 Actual Zero: 1	Expected Zero: 1 Actual Zero: 1

Como se observó en la tabla 9 se logra la diferenciación entre los datos recolectados con signo y sin signo, donde aquellos elementos que comparten el mismo signo, ejemplo 1 y 3, la bandera tiene el mismo valor para la misma comparativa sin importar si es con signo o

sin signo la operación, sin embargo, aquellos que tengan valores de diferente signo, como en el ejemplo 2, el valor de la bandera es diferente.

5.2.2 Aritméticas

Las funciones aritméticas son las operaciones más complejas y las que más tiempo llevan realizar dentro de la ALU, generalmente las rutas críticas de las arquitecturas se miden realizando alguna de estas operaciones.

La complejidad de éstas se debe a dos principales situaciones: una entrada que solo afecta a estas operaciones: Acarreo de Entrada, Carry IN o Cin para abreviar y su segunda complejidad son las banderas de Overflow y Carry que toman importancia, en estas operaciones.

Siendo el Cin, el acarreo de una operación del pasado y se necesita para la operación a presente. La bandera Overflow como el desbordamiento, generan un resultado erróneo y advierte un posible error el usuario sobre las limitaciones de la arquitectura. Por último, el Carry es el acarreo obtenido de la operación que se está ejecutando, y posible de insertar en la Carry IN a futuro.

Tabla 10 Ejemplos de Operaciones Aritméticas [6]

Entradas / Salidas	Cin: 0 Dato AS: -329011843 Dato BS: -39842810 Dato AU: 3965955453 Dato BU: 4255124486	Cin: 1 Dato AS: -835321783 Dato BS: 1234113092 Dato AU: 3459645513 Dato BU: 1234113092	Cin: 0 Dato AS: 1990550577 Dato BS: 967648330 Dato AU: 1990550577 Dato BU: 967648330

Suma con Signo OP Code: 5.	Exp Result: 3926112643 Act Result: 3926112643 Negative: 1 Carry: 1 Overflow: 0	Exp Result: 398791310 Act Result: 398791310 Negative: 0 Carry: 1 Overflow: 1	Exp Result: 2958198907 Act Result: 2958198907 Negative: 1 Carry: 0 Overflow: 0
Suma sin signo OP Code: 6.	Exp Result: 396112643 Act Result: 396112643 Negative: 1 Carry: 1 Overflow: 0	Exp Result: 398791310 Act Result: 398791310 Negative: 0 Carry: 1 Overflow: 1	Exp Result: 2958198907 Act Result: 2958198907 Negative: 1 Carry: 0 Overflow: 0
Resta sin signo OP Code: 7.	Exp Result: 4005798263 Act Result: 4005798263 Negative: 1 Carry: 0 Overflow: 0	Exp Result 2225532422 Act Result:2225532422 Negative: 1 Carry: 1 Overflow: 0	Exp Result: 1022902247 Act Result: 1022902247 Negative: 0 Carry: 1 Overflow: 0

Como se denota en la tabla 10 los valores obtenidos del DUT coinciden con los valores esperados en el modelo de referencia, y las banderas coinciden en todos los casos. La bandera Negative aun cuando el resultado esperado sea positivo, dado el ejemplo 1 en el caso de suma sin singo, esta solo se refiere al bit 32 del resultado siendo coherente la obtención de dicho valor. El Carry viene de la suma de bits y el acarreo de entrada, por lo que no tiene importancia el signo de los datos, y estos comparten valores tanto en la suma con signo como la suma sin signo. De la misma forma el Overflow solo depende del bit más significativo de las entradas y del resultado.

5.2.3 Desplazamientos

Las funciones de desplazamiento necesitan de dos entradas, donde el Data_A será el que se desplaza, y el que defina cuánto se desplaza ese dato, es el Data_B debido a que el valor de Data_B puede llegar a ser mayor a la cantidad de bits del Data_A. Por ello al Data_B se le aplica un módulo que es el residuo de la división de éste como divisor, y cuyo

dividendo es la cantidad de bits del Data_A. Para el caso de la arquitectura RISC-V se trabaja con una cantidad de bits de 32 bits.

Tabla 11 Ejemplos de Operaciones de Desplazamiento [6]

Entradas / Salidas	Dato AH: EC63AD7D Dato BH: FDA00C06 %32: 6	Dato AH: CE360049 Dato BH: 498F1244 %32: 4	Dato AH: 76A56431 Dato BH: 39AD244A %32: 10
Izquierdo Lógico. OP Code: 8.	Exp Result: 18EB5F40 Act Result: 18EB5F40 Negative: 0	Exp Result: E3600490 Act Result: E3600490 Negative: 1	Exp Result: 9590C400 Act Result: 9590C400 Negative: 1
Derecho Lógico. OP Code: 9.	Exp Result: 03B18EB5 Act Result: 03B18EB5 Negative: 0	Exp Result: 0CE36004 Act Result: 0CE36004 Negative: 0	Exp Result: 001DA959 Act Result: 001DA959 Negative: 0
Derecho Aritmético. OP Code: 10.	Exp Result: FFB18EB5 Act Result: FFB18EB5 Negative: 1	Exp Result: FCE36004 Act Result: FCE36004 Negative: 1	Exp Result: 1DA959 Act Result: 1DA959 Negative: 0

Para los desplazamientos lógicos se logró verificar que se colocaron en el lado designado por el código de operación fueron ceros, y para el desplazamiento aritmético derecho se dispuso del valor del bit más significativo, es decir, si el último valor hexadecimal se encuentra entre 8 y F, se colocarían unos, y en caso de ser menor a 8 se colocan ceros. En todos los ejemplos la cantidad de translación fue la misma que el valor Data_B modulado a 32, cantidad de bits de la arquitectura. Se cumple así con los requisitos del sistema para las funciones de desplazamiento. Otro punto por considerar es la bandera Negative, que implica el valor del bit más significativo del resultado. A pesar de no ser importante para la operación, puede utilizarse para tomas de decisiones dentro del controlador.

5.2.4 Operaciones Lógicas

Las operaciones lógicas no son más que las compuertas con dicho nombre, donde la función de cada compuerta para señales de forma hexadecimal se facilita en el apéndice A.2 de este mismo documento, con el de su revisión, para la creación del modelo de referencia, y próximo análisis del DUT.

Tabla 12 Ejemplos de Operaciones Lógicas [6]

Entradas / Salidas	Dato AH: EC63AD7D Dato BH: FDA00C06	Dato AH: CE360049 Dato BH: 498F1244	Dato AH: 76A56431 Dato BH: 39AD244A
OR. OP Code: 11.	Exp Result: FDE3AD7F Act Result: FDE34D7F Negative: 1	Exp Result: CFBF124D Act Result: CFBF124D Negative: 1	Exp Result: 7FAD647B Act Result: 7FAD647B Negative: 0
XOR. OP Code: 12.	Exp Result: 11C3A17B Act Result: 11C3A17B Negative: 0	Exp Result: 87B9120D Act Result: 8989120D Negative: 1	Exp Result: 4F08407B Act Result: 4F08407B Negative: 0
AND. OP Code: 13.	Exp Result: EC200C04 Act Result: EC200C04 Negative: 1	Exp Result: 48060040 Act Result: 48060040 Negative: 0	Exp Result: 30A52400 Act Result: 30A52400 Negative: 0

Con la tabla 12 se comprende la función de las operaciones lógicas en una ALU, fijándose en el bit más significativo; bandera Negative, para la compuerta OR siempre que alguna de sus entradas tenga su bit más significativo en 1, su salida lo tendrá también. Pero por otro lado en la compuerta XOR solo se llega a levantar la bandera Negative cuando son de signo contrario, es decir sus bits más significativos son opuestos. Y para finalizar la compuerta AND logra alzar la bandera, cuando en ambas entradas su bit más significativo es el mismo y su valor uno.

5.3 Porcentajes de Cobertura

Con 200,000 secuencias diferentes alternando el código de operación se obtuvo la siguiente información:

Coverage Data_A: 88.304916 %

Coverage Data_B: 88.412819 %

Coverage Result: 56.402676 %

Coverage ALU_CTRL: 100

Y algunos Coverage cruzados que se hicieron fue la de cruzar la señal de Control con las banderas. Esto presentó los siguientes resultados:

Cruzado con Zero: 64.285713

Cruzado con Carry: 60.714287

Cruzado con Negative: 82.142860

Cruzado con Overflow: 60.714287

Otras pruebas realizando 200,000 secuencias, ejecutadas consistieron en separar según el tipo de operación en cuatro grupos: Comparativas, Aritméticas, Desplazamientos y Lógicas. Los resultados se muestran en la tabla 13. Los datos de entrada tienen un Coverage de Data_A: 88.412819, y Data_B: 88.476479.

Tabla 13 Porcentaje de Coverage según el tipo de operación [6]

Coverage	Comparativas	Aritméticas	Desplazamientos	Lógicas
Result	0.001079	88.435478	48.384766	71.337936
ALU_CTRL	35.714287	21.428572	21.428572	21.428572

Teniendo en cuenta todo lo anterior se llega a concluir que se obtienen valores aceptables de Coverage, con una cantidad finita de 200000 secuencias. Donde las entradas de información se redondean a un 88% de puntos de contenedores cubiertos, por lo menos en una ocasión, dándose una gran cantidad de posibilidades.

En la prueba que realiza variantes en la señal de control, se obtiene un porcentaje de cobertura de un 100%, implicando que todas las operaciones se ejecutan y no se produjeron

errores en los resultados dada su variación. De forma opcional se probaron los puntos de cobertura de forma cruzada con las banderas y la señal de control. Cuyos detalles por destacar es que la bandera Zero aparece a menor medida debido a que solo se utiliza en las operaciones comparativas, igual que las banderas de Carry y Overflow que se dan únicamente en las señales aritméticas, y la que se obtiene con mayor frecuencia es la bandera Negative, debido a que es utilizada en todas las operaciones excepto para las comparativas.

Por último, se obtuvieron los Coverage del bus de datos Result, donde para las comparativas se obtiene el valor mínimo, lo anterior debido a que siempre aparece un 0, y dicho valor es un posible contenedor, y no se llegan a completar los demás contenedores, mientras que en los demás su valor es mucho mayor, producto de que el resultado varió. En las funciones aritméticas se denotó la mayor variación debido a la complejidad de sus funciones. Como punto importante el Coverage de la señal ALU_CTRL, solo denota la cantidad de operaciones que contenía cada tipo de función: Comparativas, 5; Aritméticas, 3; Desplazamiento, 3; Lógicas, 3.

Capítulo 6: Conclusiones y recomendaciones

6.1 Conclusiones

Los bloques descritos en la tabla 1 fueron diseñados a partir de una planilla y modificándolos hasta lograr un funcionamiento sin errores de sintaxis y que todos sus elementos se encuentren interconectados para la transferencia correcta de datos dentro del ambiente de verificación.

Se creó la interfaz que conecta el ambiente de verificación con el DUT, por medio del testbench, y se generaron una gran cantidad de pruebas; con ciertas limitaciones de la herramienta tales como los mostrados en el apéndice A.3.2 Riesgos y Dependencias, con el propósito de dar una revisión completa al DUT.

Dentro del concepto de Scoreboard se diseñó un modelo de referencia que asemeja el comportamiento de la ALU, tanto lo que respecto a sus entradas de información y control como a las salidas que corresponden a resultados de operaciones y banderas que permiten tomar decisiones a futuro en el controlador.

Se encontraron 3 errores; dos de ellos a nivel del diseño y uno de ellos a nivel de verificación. Los 3 errores fueron corregidos y se logró trabajar de forma continua sin interrupciones.

Se lograron los requerimientos de cobertura para 250,000 secuencias en la prueba alu_crazy_test, como se denota en el apéndice de verificación de la sección A.3.5, cumpliendo así el objetivo opcional para este trimestre.

A nivel empresarial se logró un aprendizaje del ambiente laboral, dentro de la institución, donde se crearon grupos de trabajo, a pesar de que se tenían labores específicas para cada estudiante. Lo anterior permitió mejorar las habilidades blandas profesionales, tales como: la comunicación, el compañerismo y valores de respeto y ayuda.

6.2 Recomendaciones

Se tenía previsto realizar una migración hacia un servidor, con el objetivo de realizar una regresión, permitiendo en si realizar la totalidad (100%) de las pruebas. Dicha migración no se realizó debido a la urgencia de tener todos los ambientes de verificación listos para el próximo semestre. Para próximos proyectos se recomienda realizar la migración para realizar la regresión bajo el sistema operativo de Ubuntu.

Se hizo búsqueda del reporte de cobertura dentro de los archivos generados por las diferentes pruebas, pero no se logró ubicar. En su lugar se utilizaron ciertas funciones para obtener dichas especificaciones en el plan de verificación. Se recomienda obtener el reporte de cobertura.

Generar pruebas específicas utilizando corner tests, con el fin de depurar la verificación de la ALU. Estos corner tests se pueden definir con restricciones en el alu_seq_item.

Por ultimo se recomienda a los estudiantes posteriores de este proyecto, que hereden el ambiente de verificación desarrollado este semestre. Compartido por este link: https://www.edaplayground.com/x/5iZ9 . Lo copien y lo agreguen a sus trabajos, para su posterior modificación y regresión.

Bibliografía

[1] Doulos. What is System Verilog? United Kingdom. 2005-2018. Link: https://www.doulos.com/knowhow/sysverilog//whatissv/

[2] Harris, S., & Harris, D. Digital Design and Computer Architecture: Arm Edition. 2015.

[3] Kumar Tala, Deepak. Functional Coverage. ASIC World. 2014. Link: http://www.asic-world.com/systemverilog/coverage.html

[4] Molina Robles, Roberto. CR_TEC_RISCV. Verification Plan. 2017.

[5] Ureña Calderón, Carlos. Diseño de un sistema para sincronizar el reloj interno de las protecciones de línea y grabadores de eventos presentes en las subestaciones de ICE. Cartago. 2005. Planilla de Informe.

[6] Valenciano, P. Resultados obtenidos por medio de simulaciones, en el browser edaplayground.com, cuyo link es: https://www.edaplayground.com/x/5iZ9

[7] Verification Guide. UVM Tutorial. Introduction to UVM. 2016. Link: http://www.verificationguide.com/p/uvm-tutorial.html

[8] https://en.wikipedia.org/wiki/Adder_(electronics)

[9] Rimolo, R. Chacón, A., García, R. TEC-Risc-V. Micro-Architectural Specification. 2018. Tecnologico de Costa Rica.

[10] Wile B. Comprehensive Functional Verification. 2005.

[11] Gonalez, R. Olvera J. ALU Unidad Aritmética Lógica. 2012. Universidad del Valle de México. Campus: San Rafael.

Apéndices

Apéndice A.1: Abreviaturas

Tabla A.1.1 Abreviaturas

UVM	Metodología Universal de Verificación
ALU	Unidad Lógica Aritmética
AH	Primer dato Hexadecimal
AS	Primer dato Decimal con signo
AU	Primer dato Decimal sin signo
BH	Segundo dato en Hexadecimal
BS	Segundo dato Decimal con signo
BU	Segundo dato Decimal sin signo
OP Code	Código de Operación
TEC o ITCR	Instituto Tecnológico de Costa Rica
DUT	Diseño bajo Prueba
ALU_CTRL	Señal de control de la ALU
Exp	Valor Esperado
Act	Valor Obtenido del DUT

Apéndice A.2: Tabla Compuertas Lógicas en Hexadecimal

Tabla A.2.1 Funciones Lógicas OR, XOR y AND dada entradas hexadecimales

	Logical Function	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	OR	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	XOR	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	AND	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	OR	1	1	3	3	5	5	7	7	9	9	B	B	D	D	F	F
	XOR	1	0	3	2	5	4	7	6	9	8	B	A	D	C	F	E
	AND	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
2	OR	2	3	2	3	6	7	6	7	A	B	A	B	E	F	E	F
	XOR	2	3	0	1	6	7	4	5	A	B	8	9	E	F	C	D
	AND	0	0	2	2	0	0	2	2	0	0	2	2	0	0	2	2
3	OR	3	3	3	3	7	7	7	7	B	B	B	B	F	F	F	F
	XOR	3	2	1	0	7	6	5	4	B	A	9	8	F	E	D	C
	AND	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
4	OR	4	5	6	7	4	5	6	7	C	D	E	F	C	D	E	F
	XOR	4	5	6	7	0	1	2	3	C	D	E	F	8	9	A	B
	AND	0	0	0	0	4	4	4	4	0	0	0	0	4	4	4	4
5	OR	5	5	7	7	5	5	7	7	D	D	F	F	D	D	F	F
	XOR	5	4	7	6	1	0	3	2	D	C	F	E	9	8	B	A

	AND	0	1	0	1	4	5	4	5	0	1	0	1	4	5	4	5
	Logical Function	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
6	OR	6	7	6	7	6	7	6	7	E	F	E	F	E	F	E	F
	XOR	6	7	4	5	2	3	0	1	E	F	C	D	A	B	8	9
	AND	0	0	2	2	4	4	6	6	0	0	2	2	4	4	6	6
7	OR	7	7	7	7	7	7	7	7	F	F	F	F	F	F	F	F
	XOR	7	6	5	4	3	2	1	0	F	E	D	C	B	A	9	8
	AND	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
8	OR	8	9	A	B	C	D	E	F	8	9	A	B	C	D	E	F
	XOR	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7
	AND	0	0	0	0	0	0	0	0	8	8	8	8	8	8	8	8
9	OR	9	9	B	B	D	D	F	F	9	9	B	B	D	D	F	F
	XOR	9	8	B	A	D	C	F	E	1	0	3	2	5	4	7	6
	AND	0	1	0	1	0	1	0	1	8	9	8	9	8	9	8	9
A	OR	A	B	A	B	E	F	E	F	A	B	A	B	E	F	E	F
	XOR	A	B	8	9	E	F	C	D	2	3	0	1	6	7	4	5
	AND	0	0	2	2	0	0	2	2	8	8	A	A	8	8	A	A
B	OR	B	B	B	B	F	F	F	F	B	B	B	B	F	F	F	F
	XOR	B	A	9	8	F	E	D	C	3	2	1	0	7	6	5	4

	AND	0	1	2	3	0	1	2	3	8	9	A	B	8	9	A	B
C	OR	C	D	E	F	C	D	E	F	C	D	E	F	C	D	E	F
	XOR	C	D	E	F	8	9	A	B	4	5	6	7	0	1	2	3
	AND	0	0	0	0	4	4	4	4	8	8	8	8	C	C	C	C
	Logical Function	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
D	OR	D	D	F	F	D	D	F	F	D	D	F	F	D	D	F	F
	XOR	D	C	F	E	9	8	B	A	5	4	7	6	1	0	3	2
	AND	0	1	0	1	4	5	4	5	8	9	8	9	C	D	C	D
E	OR	E	F	E	F	E	F	E	F	E	F	E	F	E	F	E	F
	XOR	E	F	C	D	A	B	8	9	6	7	4	5	2	3	0	1
	AND	0	0	2	2	4	4	6	6	8	8	A	A	C	C	E	E
F	OR	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
	XOR	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
	AND	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Apéndice A.3: Plan de Verificación

Apéndice A.3.1: Herramientas Requeridas

- Computador con internet.
- Explorador de preferencia Google Chrome.
- Un usuario en el servidor de EDA Playground, de preferencia un correo empresarial.
- Instalación de la herramienta de simulación Synopsys VCS. En el proyecto se utiliza la versión 2014.10.
- Incluir las librerías del UVM en los archivos System-Verilog (.sv).
- El diseño bajo prueba otorgado por el diseñador.

Apéndice A.3.2: Riesgos y Dependencias

Las amenazas más influyentes durante el proyecto corresponden a: 1) Convertir un ambiente de verificación para un sistema secuencial, controlado por un reloj de entrada, a un ambiente de verificación para DUTs combinacionales, es decir diseños independientes de una señal de reloj y generen sus salidas de forma independiente. 2) La creación de un modelo de referencia que se equipare a la ALU, a partir de SystemC. 3) Y por último tomar la decisión de dónde colocar el Coverage en el ambiente de verificación debido a que se tiene 4 posibles opciones. 4) La herramienta cuenta con un tiempo máximo de simulación de 10s, siendo imposibles pruebas con un 100% de Coverage en todas sus entradas.

Las dependencias que se trabajan para el proyecto son del tipo de aprendizaje del lenguaje System-Verilog y el estándar UVM empresarial. Y debido a que la herramienta EDAPLAYGRPUND se encuentra solamente en línea es necesario ancho de banda, para cuando se trabaja en el proyecto. No hay problemas de incompatibilidad ni de posibles actualizaciones debido a que la herramienta EDAPLAYGROUND contaba con todas las herramientas necesarias.

Apéndice A.3.3: Funciones por Verificar

Para controlar cuales funciones de la ALU se iban a testear se colocaba una restricción como se muestra en la figura 4, para este ejemplo son las señales comparativas que son aquellas que van desde el código de operación a 0 hasta el código 5, incluyéndolos.

Otra manera de generar una restricción de la señal de control fue crear diferentes secuencias donde después de aleatorizar todas sus entradas y antes de enviar la información, se fuerza a tener un valor especifico en esa señal. Con ello se hizo pruebas diversas con resultados de Coverage diversos.

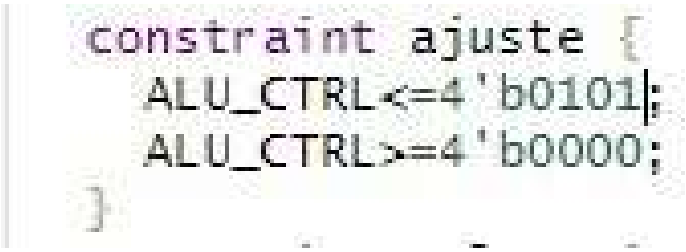

Figura A.3.3.1 Restricción de la señal de Control

A la hora de realizar pruebas se generaron incoherencias en los resultados y en las banderas, pero no en todos los casos, solo se daba en ciertas combinaciones de valores en funciones específicas, para ello se crea otra restricción, con el objetivo de dar énfasis y encontrar la causa de las incoherencias. Esta restricción se diseña como se observa en la figura A.3.3.1.

Durante el desarrollo de las pruebas hubo 2 generadores de errores. El primero fueron los valores con signo negativos y los valores con signo contrario.

```
constraint Valores{
  Data_A>2147483647;
  Data_B<2147483647;
}
```

Figura A.3.3.2 Restricción de las variables numéricas

Ambas restricciones están en el módulo alu_seq_item debido a que ahí se definen el rango de valores al momento de aleatorizar las variables de entrada.

Apéndice A.3.4: Pruebas y Métodos Específicos

Debido a que la ALU es un diseño cuyo objetivo es generar un resultado o activar banderas y no es de suma importancia las señales internas que en ella se generan, no es necesario realizar el proceso back-door; proceso que imita los registros internos del diseño en el modelo de referencia. Por esto se tomó la decisión de tomar a la ALU como una caja negra, donde se insertan estimulas en las entradas y se obtienen respuestas en las salidas.

Debido a que la ALU es un sistema combinacional se decide eliminar del ejemplo original, encontrado de la guía de verificación [4] para una memoria, las señales de reloj y de reset. Y se coloca un reloj dentro de la interfaz virtual, fuera del testbench, que proporcione el control del driver y del monitor, ya que estos dos componentes la requieren para hacer el envió de la información a su siguiente etapa. Ambas escrituras se dan en el flanco positivo del reloj de 20MHz, requerimiento mínimo.

Ya con el ambiente de verificación completamente combinacional se diseñan secuencias específicas para cada una de las funciones, es decir mantener la señal de control constante y variar los datos numéricos, con el fin de encontrar los valores que generan

discrepancias en sus resultados y su comportamiento ante distintos valores. Y para terminar se diseña su bloque físico para la conexión con otros módulos.

Se seguiría a revisar el driver, como un elemento que recolecta la información obtenida del secuenciador, y este componente espera al flanco positivo para transmitirlo. De la misma forma se diseña el driver que transmite los datos al Scoreboard. Se crean ambos bloques físicos, y se combinan con el secuenciador para crear el agente y según si el agente es pasivo desconecta al driver y al secuenciador, dejando con la interfaz al monitor. En caso de ser el agente activo, los tres elementos estaría en funcionamiento.

Por último, se desarrolla el modelo de referencia de la ALU utilizando condicionales "if" controlados por la señal de control; ALU_CTRL, donde para cada caso se generan los resultados esperados y en caso de no existir dicha condición, "default", todos sus valores de salida se manejan en cero.

Apéndice A.3.5: Requerimientos de Cobertura

Describe las metas de los estímulos utilizados, estas deberían cubrir todas las especificaciones funcionales del diseño y asegurar que los componentes de estímulo del ambiente hagan lo que deberían de realizar. La cobertura indicada en la tabla A.3.5.1 se logra con un estímulo de 250000 pruebas.

Tabla 3.5.1 Porcentajes de cobertura meta y obtenidos

Coverage Point	What does it cover?	Goal	Actual Percentage
ALU Input A	This coverage point verifies that most of Input A's possible combinations are stimulated.	90%	91.06%

ALU Input B	This coverage point verifies that most of Input B's possible combinations are stimulated.	90%	91.17%
ALU Output	This coverage point verifies that most of the ALU output's possible combinations are generated.	60%	60.48%
ALU Selection	This coverage point verifies that all the functions of the ALU are being tests.	100%	100%

Apéndice A.3.6: Escenario de casos de pruebas

Tabla 3.6.1 Descripción de las pruebas

Test Name	What does it test?	Description
Name: alu_equ_test	Genera diferentes secuencias con entradas de datos distintas, con la señal de control de 0000(0 Hex).	Verifica si ambos datos son iguales, en caso de serlo se activa la bandera zero, en caso contrario se coloca con un 0. Todas las demás salidas son 0
Name: alu_le_test	Genera diferentes secuencias con entradas de datos distintas, con la señal de control de 0001(1 Hex).	Verifica si el dato A (Signed) es menor al dato B (Signed), en caso de serlo se activa la bandera zero, en caso contrario se coloca con un 0. Todas las demás salidas son 0
Name: alu_leu_test	Genera diferentes secuencias con entradas de datos distintas, con la señal de control de 0010(2 Hex).	Verifica si el dato A (Unsigned) es menor al dato B (Unsigned), en caso de serlo se activa la bandera zero, en caso contrario se coloca con un 0. Todas las demás salidas son 0

Name: alu_gre_test	Genera diferentes secuencias con entradas de	Verifica si el dato A (Signed) es mayor al dato B (Signed), en caso de serlo se activa
	datos distintas, con la señal de control de 0011(3 Hex).	la bandera zero, en caso contrario se coloca con un 0. Todas las demás salidas son 0
Name: alu_greu_test	Genera diferentes secuencias con entradas de datos distintas, con la señal de control de 0100(4 Hex).	Verifica si el dato A (Unsigned) es mayor al dato B (Unsigned), en caso de serlo se activa la bandera zero, en caso contrario se coloca con un 0. Todas las demás salidas son 0
Name: alu_add_test	Genera diferentes secuencias con entradas de datos distintas, con la señal de control de 0101(5 Hex).	Suma el Dato A (Signed) con el Dato B (Signed), en caso de que el resultado sea negativo se activa la bandera Negative, y si sobrepasa los 32 bits, trunca el resultado activa tanto el Carry como el Overflow. La bandera Zero se mantiene en 0.
Name: alu_addu_test	Genera diferentes secuencias con entradas de datos distintas, con la señal de control de 0110(6 Hex).	Suma el Dato A (Unsigned) con el Dato B (Unsigned), si sobrepasa los 32 bits, trunca el resultado activa tanto el Carry como el Overflow. La bandera Zero se mantiene en 0.
Name: alu_subu_test	Genera diferentes secuencias con entradas de datos distintas, con la señal de control de 0111(7 Hex).	Al Dato A (Unsigned) se le resta el Dato B (Unsigned), en caso de que el resultado tenga en su bit 32 un valor de uno se activa la bandera Negative. La bandera Zero se mantiene en 0. Las banderas de Carry y Overflow se pueden activar.

Name: alu_left_test	Genera diferentes secuencias con entradas de datos distintas, con la señal de control de 1000(8 Hex).	Realiza tantos desplazamientos hacia la izquierda del Dato A según el Dato B (Unsigned), completándolos con ceros. Todas las demás banderas se desactivan.

		Excepto la bandera Negative que depende del valor del bit 32.
Name: alu_ri_test	Genera diferentes secuencias con entradas de datos distintas, con la señal de control de 1001(9 Hex).	Realiza tantos desplazamientos hacia la derecha del Dato A según el Dato B (Unsigned), completándolos con ceros. Todas las demás banderas se desactivan. Excepto la bandera Negative que depende del valor del bit 32.
Name: alu_ria_test	Genera diferentes secuencias con entradas de datos distintas, con la señal de control de 1010(A Hex).	Realiza tantos desplazamientos hacia la izquierda del Dato A según el Dato B (Unsigned) completándolos con el valor del bit 32 del Dato A. Todas las demás banderas se desactivan. Excepto la bandera Negative que depende del valor del bit 32.
Name: alu_or_test	Genera diferentes secuencias con entradas de datos distintas, con la señal de control de 1011(B Hex).	Realiza la operación lógica OR entre el dato A y el dato B (Bit por Bit), y sus salidas se mostrarán en el bus de resultado. Todas las demás banderas se mantienen en 0. Excepto la bandera Negative que depende del valor del bit 32.

Name: alu_xor_test	Genera diferentes secuencias con entradas de datos distintas, con la señal de control de 1100(C Hex).	Realiza la operación lógica XOR entre el dato A y el dato B (Bit por Bit), y sus salidas se mostrarán en el bus de resultado. Todas las demás banderas se mantienen en 0. Excepto la bandera Negative que depende del valor del bit 32.
Name: alu_and_test	Genera diferentes secuencias con entradas de	Realiza la operación lógica AND entre el dato A y el dato B (Bit por Bit), y sus salidas
	datos distintas, con la señal de control de 1101(D Hex).	se mostrarán en el bus de resultado. Todas las demás banderas se mantienen en 0. Excepto la bandera Negative que depende del valor del bit 32.
Name: alu_crazy_test	Genera diferentes secuencias con entradas de datos distintas, con la señal de control aleatoria pero restringida entre 0 y D en hexadecimal.	Se verifica que la operación corresponde al código de operación, que se muestra en la forma de onda.

Apéndice A.3.7: Requerimiento de Recursos

El recurso computacional es de suma importancia, debido a que la herramienta en un browser que requiere de un tiempo en un servidor externo del ITCR, siendo necesaria una conexión de media a alta velocidad para elaborar una cantidad mínima de secuencias para cada prueba.

Se elaboran diferentes pruebas según lo que se vaya obteniendo. Las primeras pruebas se realizan con una sola función ejecutándose y variando los valores de entrada,

comparamos las respuestas del modelo de referencia y se buscan los valores que generan incoherencias en la comparativas, con el fin de darle una lista de bugs al diseñador, esta lista se encuentra en el Anexo B.1.

Ya habiendo revisado cada una de las funciones por separado y ser corregidas por el diseñador, se dará un reporte de cobertura, que contenga entradas y salidas, según el tipo de operación limitando a la función de control para cada rango. Y por último se dará un reporte completo de cobertura, donde todas las operaciones se realizan en la misma prueba.

Apéndice A.3.8: Detalles del Programa.

Week	Date	Activities
0	January 15th - February 2nd	• Draft (anteproyecto) adjustments. • Reading about UVM and the Verification Plan. • Tool Familiarization.
1	February 5th - February 9th	• Monitor construction. • Driver construction. • Sequencer construction. • Agent interconnection.
2	February 12th - February 16th	• Scoreboard construction. • Basic Reference Model construction.
3	February 19th - February 23rd	• UVM Environment Bring-up. • Connect the Environment with the DUT. • Generate a Basic Test.
4	February 26th - March 2nd	• UVM Environment adjustments. • Interfaces adjustments. • Components adjustments.
5	March 5th - March 9th	• UVM Environment adjustments. • Interfaces adjustments. • Components adjustments.
6	March 12th - March 16th	
7	March 19th - March 23rd	• Inclusion of Coverage Points. • Tests Design. • Development of the Reference Model.

---	March 26th - March 30th	HOLY WEEK
8	April 2nd - April 6th	• Migration of the UVM Environments to the server, to be able to use Modelsim.
9	April 9th - April 13th	• Develop new tests. • Debug simulations and design. • Develop the Reference Model even further. • Apply fixes.
10	April 16th - April 20th	• Build a Regression with all the tests.
11	April 23rd - April 27th	• Develop new tests. • Debug simulations and design. • Develop the Reference Model even further. • Apply fixes. • Update the Regression.
12	April 30th - May 4th	
13	May 7th - May 11th	
14	May 14th - May 18th	
15	May 21st - May 25th	
16	May 28th - June 1st	

Anexos

Anexo B.1: Bugs y Errores encontrados.

Name of bug	Bad Shift Function
Bug ID	alu_b001

Where is it located? (Design Block or UVM Environment)	Design Block
Priority (Low, Medium, High)	High
Which test triggered the bug?	Alu_left_test, alu_ri_test, alu_ria_test
Seed test	10
Scenario description	Las pruebas realizan desplazamientos ineficientes, los resultados daban en muchos casos 00000000 o FFFFFFFF.
Has it been resolved?	Yes

Name of bug	Bad Carry Function
Bug ID	alu_b002
Where is it located? (Design Block or UVM Environment)	Design Block
Priority (Low, Medium, High)	Medium
Which test triggered the bug?	alu_add_test, alu_addu_test, alu_subu_test
Seed test	10

Scenario description	El Carry está definido de forma muy simple y solo utiliza los bits más significativos, despreciando los demás datos.
Has ir been resolved?	Yes

Name of bug	Flag Incomprehension
Bug ID	alu_b003
Where is it located? (Design Block or UVM Environment)	UVM Enviroment
Priority (Low, Medium, High)	Medium
Which test triggered the bug?	alu_crazy_test
Seed test	10
Scenario description	La mayoría de las banderas no coinciden con el valor que debe dar según las entradas que se tienen. Algunos ejemplos: 1. La bandera Negativa se encuentra en muchos casos baja en la mayoría de las funciones. Siendo este el bit 32 del resultado. 2. El Carry no realiza la función correspondiente. 3. El Overflow no se comprende en su totalidad.
Has ir been resolved?	Yes

Anexo B.2 Hoja de Información

Información del estudiante:

Nombre: Pablo André Valenciano Blanco

Cédula o No. Pasaporte: 115720043

Carné ITCR: 201242320

Dirección de su residencia en época lectiva: Cartago, Cartago. Gonzales Angulo Etapa II

Dirección de su residencia en época no lectiva: Heredia, Heredia. San Francisco, La Lilliana

Teléfono en época lectiva: 87143350

Teléfono época no lectiva: 22610759

Email: pvalencianocr@gmail.com

Fax: NC

Información del Proyecto:

Nombre del Proyecto: Desarrollo de un Ambiente de Verificación para una Unidad Lógica Aritmética mediante la Metodología Universal de Verificación.

Profesor Asesor: Roberto Molina Robles

Horario de trabajo del estudiante: L: 8:00am a 2:00 pm, K-V de 2:00 pm a 6:00 pm

Información de la Empresa:

Nombre: Esc. De Ingeniería en Electrónica.

Zona: NC

Dirección: Instituto Tecnológico de Costa Rica. Sede Central Cartago.

Teléfono: 25509318

Fax: NC

Apartado: NC

Actividad Principal: Verificador Funcional.