Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Programa de Licenciatura en Ingeniería Electrónica

Diseño de un ambiente de verificación usando UVM para un módulo de interconexión basado en el protocolo de comunicación AMBA AXI desarrollado en SystemVerilog

Informe de Trabajo Final de Graduación para optar por el título de

Ingeniero en Electrónica con el grado académico de Licenciatura

Jeremy Córdoba Wright

Cartago, 25 de noviembre del 2024

Dedicatoria

Declaro que el presente documento de tesis ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos y resultados experimentales propios.

En los casos en que he utilizado bibliografía he procedido a indicar las fuentes mediante las respectivas citas bibliográficas. En consecuencia, asumo la responsabilidad total por el trabajo de tesis realizado y por el contenido del presente documento.

Jeremy Córdoba Wright

Cartago, 25 de noviembre del 2024

Céd: 1-1806-0214

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica
Trabajo Final de Graduación
Acta de Aprobación

Defensa de Trabajo Final de Graduación
Requisito para optar por el título de Ingeniero en Electrónica
Grado Académico de Licenciatura

El Tribunal Evaluador aprueba la defensa del trabajo final de graduación denominado *Diseño de un ambiente de verificación usando UVM para un módulo de interconexión basado en el protocolo de comunicación AMBA AXI desarrollado en SystemVerilog*, realizado por el señor Jeremy Córdoba Wright 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

ANIBAL IGNACIO RUIZ BARQUERO (FIRMA)
Firmado digitalmente por ANIBAL IGNACIO RUIZ BARQUERO (FIRMA)
Fecha: 2024.11.28 08:57:17 -06'00'

MSc. Ing. Anibal Ruiz Barquero
Profesor Lector

JAVIER MAURICIO PEREZ RODRIGUEZ (FIRMA)
PERSONA FISICA, CPF-01-1110-0856.
Fecha declarada: 28/11/2024 09:20:33 AM

Ing. Javier Perez Rodriguez
Profesor Lector

RONNY GIOVANNI GARCIA RAMIREZ (FIRMA)
PERSONA FISICA, CPF-01-1137-0229.
Fecha declarada: 28/11/2024 08:25:47 AM
Esta es una representación gráfica únicamente, verifique la validez de la firma.

Dr. Ing. Ronny García Ramirez
Profesor Asesor

Cartago, 25 de noviembre del 2024

Resumen

En el laboratorio de Computación de Alto Rendimiento (HPC) de la Escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica (TEC), se está desarrollando un proyecto de diseño que incluye la implementación de un módulo de interconexión basado en el protocolo de comunicación AMBA AXI, utilizando el lenguaje de descripción de hardware SystemVerilog. Como parte fundamental del proyecto, es imprescindible diseñar una serie de pruebas que validen el correcto funcionamiento de las características del módulo.

Para el desarrollo de dichas pruebas, se implementó un ambiente de verificación en UVM que envía estímulos al módulo y recibe las respuestas correspondientes para evaluar su comportamiento. Los resultados obtenidos se comparan con una referencia que simula el mismo comportamiento del bloque, permitiendo así detectar errores presentes en el módulo.

El ambiente de verificación se desarrolló utilizando el paquete de clases de UVM, el cual estandariza el proceso de verificación de módulos siguiendo una estructura jerárquica y la estructura del ambiente, las pruebas y los resultados serán abarcados posteriormente en el presente documento.

Palabras clave: Agente, Assertions, AXI, Constraints, Crossbar, Cobertura, Driver, Golden reference, Handshake, Monitor, Scoreboard, UVM.

Abstract

In the High-Performance Computing (HPC) laboratory of the School of Electronic Engineering at the Costa Rica Institute of Technology (TEC), a design project is being developed, which includes the implementation of an interconnection module based on the AMBA AXI communication protocol, using the SystemVerilog hardware description language. As a fundamental part of the project, it is essential to design a series of tests to validate the correct functionality of the module's features.

For the development of these tests, a UVM-based verification environment was implemented, which sends stimuli to the module and receives the corresponding responses to evaluate its behavior. The results obtained are compared to a reference that simulates the same behavior of the block, thereby detecting potential errors in the module.

The verification environment was developed using the UVM class library, which standardizes the module verification process following a hierarchical structure. The structure of the environment, the tests, and the results will be addressed later in this document.

Keywords: Agent, Assertions, AXI, Constraints, Crossbar, Coverage, Driver, Golden reference, Handshake, Monitor, Scoreboard, UVM.

a mis queridos padres Robert Córdoba, Bárbara Wright y Andrés Astúa les dedico este gran logro que representa años de esfuerzo, no solo mío, sino también de ustedes, que con amor y dedicación me criaron, inculcándome los valores que me han guiado hasta aquí.

Agradecimientos

Este trabajo no habría sido posible sin el apoyo y la orientación que recibí a lo largo del proceso. Quiero agradecer especialmente al Instituto Tecnológico de Costa Rica (TEC) y al laboratorio de High Performance Computing (HPC), por brindarme la oportunidad de desarrollar este proyecto.

Mi más sincero agradecimiento al profesor Ronny García Ramírez, por su invaluable guía y retroalimentación en cada etapa de este trabajo. Su apoyo fue clave para el éxito de este proyecto.

A todos aquellos que, de una u otra forma, me inspiraron y acompañaron en este camino, les ofrezco mi más profunda gratitud.

Jeremy Córdoba Wright

Cartago, 25 de noviembre del 2024

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La creciente demanda de la tecnología sistema en chip (por sus siglas en inglés, SoC's) está impulsada por la evolución de la electrónica, buscando satisfacer las necesidades actuales del mercado, como dispositivos más pequeños, inteligentes, rápidos y eficientes en el manejo de datos y consumo de energía. Un SoC es un enfoque de diseño que combina diversos componentes, como procesadores, interfaces de red y controladores de entrada o salida, en un único chip [1, 2].

Los SoC's integran diversas funciones, como núcleos digitales, memoria, componentes analógicos y gestión de energía, en un solo chip utilizando tecnología CMOS estándar [3]. Esta integración ofrece ventajas en tamaño, costo, consumo de energía y rendimiento [4]. Los SoC's son circuitos integrados que se encuentran presentes en casi cualquier dispositivo electrónico inteligente en el mercado, como los teléfonos, relojes inteligentes, televisores, consolas de video juegos, computadores, entre otros [5]. Sin embargo, el aumento de la complejidad y los desafíos del escalado tecnológico afectan el tiempo de desarrollo y los costos, lo que requiere mejorar la eficiencia en el diseño y la verificación para ampliar el potencial de mercado de los SoC's [4].

A medida que los SoC's se vuelven más grandes y complejos dependen cada vez más de los módulos de interconexión para conectar eficientemente múltiples bloques funcionales de IP en un solo chip y facilitar la comunicación entre ellos [6]. Estas interconexiones se han vuelto más complejas y sofisticadas para satisfacer las demandas de las nuevas generaciones de SoC's, optimizando al mismo tiempo el área, la eficiencia de procesamiento y el consumo de energía [6]. Los buses en chip son una implementación común para las redes de interconexión en SoC's, facilitando la reutilización de diseño y simplificando la integración de componentes [7, 8]. Entre las redes de interconexión basadas en buses en chip más populares se encuentran AMBA, Avalon, CoreConnect, STBus y Wishbone [8]. Estas interconexiones pueden lograr un alto ancho de banda, baja latencia y consumo de energía escalable, convirtiéndolas en una solución efectiva para los desafíos de interconexión en SoC's [9].

La arquitectura de bus de microcontrolador avanzada (por sus siglas en inglés AMBA) es

un estándar para la conexión y manejo de bloques funcionales en un integrado. AMBA simplifica el desarrollo de diseños con múltiple IP's, procesadores y una amplia variedad de controladores y periféricos, además ofrece un alto ancho de banda y baja latencia en la transmisión de datos, por lo que su popularidad ha aumentado en el tiempo y ahora es comúnmente usado en los sistemas en chip y los circuitos integrados de aplicación específica [10]. El protocolo AMBA AXI ofrece la posibilidad de ser reutilizado en diversos sistemas, lo que permite trabajar con una amplia variedad de circuitos integrados con diferente potencia, rendimiento y especificaciones de área, es decir, es compatible entre diferentes componentes de distintos equipos de diseño o proveedores y además, el protocolo se encuentra en constante evolución, ya que es ampliamente implementado y tiene constante soporte en la industria de los semiconductores.

En el laboratorio HPC (High Performance Computing) de la Escuela de Ingeniería Electrónica en el Instituto Tecnológico de Costa Rica (TEC), en un proyecto de diseño se encontraba en desarrollo un módulo de interconexión basado en el protocolo de comunicación AMBA AXI usando el lenguaje de descripción de hardware SystemVerilog, siguiendo una guía exhaustiva que describía todas sus generalidades ya definidas por el estándar, desde sus características hasta sus modos de operación, variables y arquitectura [10]. El módulo, al ser un sistema grande y complejo, estaba susceptible a presentar algún tipo de error.

Algunos de los errores que el sistema puede presentar son los errores en los modos de operación, datos incorrectos, fallos en la escritura o lectura de transacciones, conexiones defectuosas o errores de seguridad. Resolver estos problemas de manera oportuna en etapas tempranas del desarrollo es crucial, ya que un error no detectado podría desencadenar problemas mayores. Entre los problemas que pueden surgir se incluye la presencia de un sistema defectuoso con errores difíciles de resolver rápidamente debido a la complejidad del sistema provocando interrupciones en las investigaciones de laboratorio, desperdicio de tiempo y pérdida de confianza en los resultados de las investigaciones. Además, sin un método adecuado para verificar futuras versiones del dispositivo, la verificación manual se vuelve extremadamente lenta, lo que puede extender considerablemente el período de desarrollo del dispositivo. Luego, el costo de encontrar y arreglar un error puede ser excesivo si este fue descubierto en una etapa avanzada del desarrollo [11].

El costo asociado a los errores no detectados a tiempo aumenta de manera exponencial a medida que el proyecto progresa a lo largo de sus etapas de desarrollo, como se puede observar en la figura 1.1, por esta razón el principal objetivo de la verificación es detectar fallos en las etapas iniciales del desarrollo, ya que es mucho más económico en comparación con su identificación en etapas posteriores y el tiempo de depuración se ve reducido considerablemente como se puede apreciar en la figura 1.2.

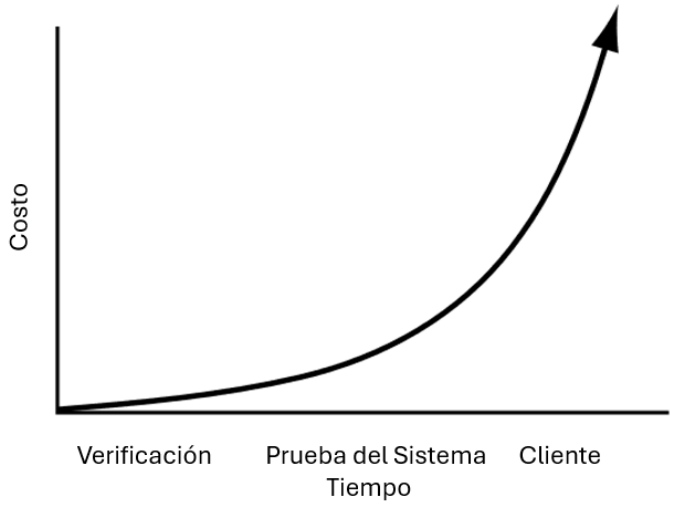


Figura 1.1: Aumento del costo en etapas posteriores de diseño [11].

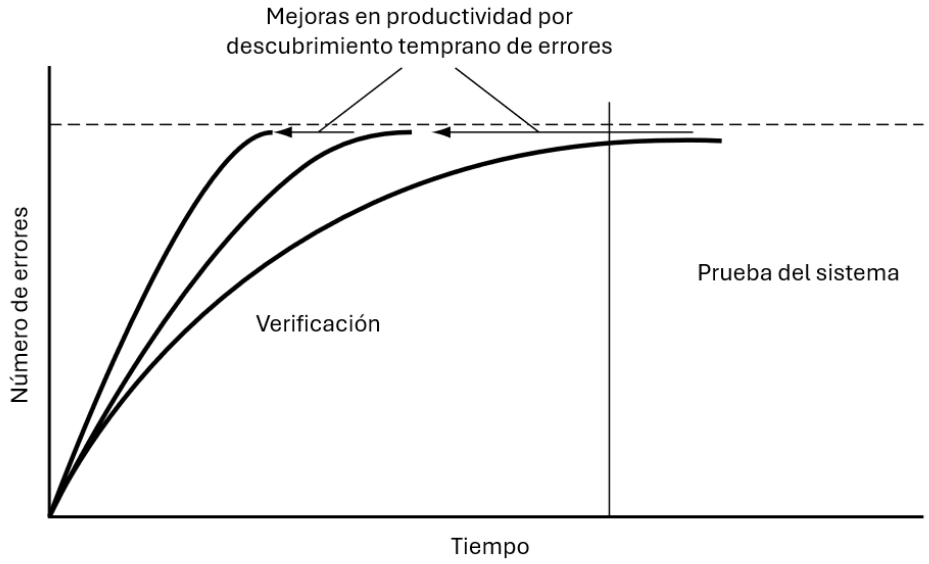


Figura 1.2: Mejora de eficiencia de errores encontrados en el tiempo mediante verificación [11].

Por lo tanto, el principal problema que el proyecto pretende resolver es la falta de un mecanismo para validar o verificar la funcionalidad del dispositivo, lo que impide detectar y corregir errores a tiempo. Esta carencia podría llevar a que el diseño final del módulo de interconexión presente errores difíciles de resolver en su versión final, comprometiendo su calidad y funcionalidad.

Esbozo de la solución

Por ende, se llevó a cabo el diseño y la verificación en paralelo con el fin de lograr la validación del módulo de interconexión AXI. El objetivo fue encontrar la mayor cantidad de errores posibles dentro del diseño actual. Además, debía cumplir con las especificaciones y características del protocolo. Dado que esta fase de desarrollo no involucraba componentes físicos ni herramientas de hardware, los recursos necesarios fueron exclusivamente de software, incluyendo una computadora y las licencias de las herramientas de ingeniería necesarias para el análisis y la validación del protocolo.

Para iniciar con el proceso de verificación, se estableció un plan de pruebas que definiera los casos generales y de esquina del dispositivo, así como la estructura del ambiente y las pruebas que se llevarían a cabo. Esto fue fundamental para mantener una guía en la etapa de desarrollo del ambiente, donde la principal herramienta fue la metodología de verificación universal (UVM, por sus siglas en inglés). UVM ofreció la posibilidad de desarrollar un ambiente modular, reutilizable y robusto para la verificación, lo que permitió el testeo completo del sistema y la reutilización del ambiente para futuras versiones del RTL [12].

Una vez desarrollado el entorno de verificación y sus respectivas pruebas, se ejecutaron simulaciones utilizando aleatorización controlada y regresiones con el objetivo de maximizar el porcentaje de cobertura. Este enfoque garantizó que se evaluara la mayor cantidad posible de escenarios del dispositivo, asegurando así que todas las características principales fueran probadas.

Capítulo 2

Meta y objetivos

2.1. Meta

El laboratorio HPC debe contar con un módulo de interconexión 4x4 basado en el protocolo de comunicación AXI, con todas sus características verificadas mediante un entorno de verificación diseñado en UVM. **Indicadores de cumplimiento:** Disponer de un repositorio centralizado, que incluya el ambiente de verificación y resultados detallados de pruebas anteriores que certifiquen el funcionamiento del dispositivo.

2.2. Objetivo general

a) Verificar que el dispositivo cumpla con todas las funcionalidades conforme al protocolo AXI, garantizando que cada característica descrita en el diseño y en el plan de pruebas funcione correctamente. **Indicadores de cumplimiento:** Análisis de cobertura del 95 % o superior.

2.3. Objetivos específicos

a) Desarrollar un plan de pruebas que cubra las características del DUT. **Indicadores de cumplimiento:** el plan de pruebas debe establecer casos generales y de esquina que involucren el uso de toda la lista de características del dispositivo.

b) Desarrollar un ambiente de verificación modular en UVM que implemente pruebas basadas en los casos generales y de esquina. **Indicadores de cumplimiento:** El ambiente es capaz de simular cada una de las pruebas exitosamente.

c) Realizar un análisis de cobertura funcional, toggle (cambio de estado) y FSM (máquina de estado finita). **Indicadores de cumplimiento:** lograr un 95 % o superior en el análisis de cobertura.

Capítulo 3

Marco teórico

3.1. Universal Verification Methodology (UVM)

UVM es un marco de clases en SystemVerilog que facilita la construcción estandarizada de pruebas funcionales. Este enfoque estandarizado hace que las pruebas sean fáciles de entender, reutilizar, flexibles y fiables en la verificación de diseños de sistemas embebidos complejos [13]. UVM proporciona características avanzadas, como mecanismos de fases, informes, callbacks y bibliotecas de secuencias, en comparación con metodologías anteriores [13]. Además, hay una amplia cantidad de recursos gratuitos disponibles para el aprendizaje y la utilización de esta herramienta. A continuación, se ilustra la jerarquía de clases en UVM [14]

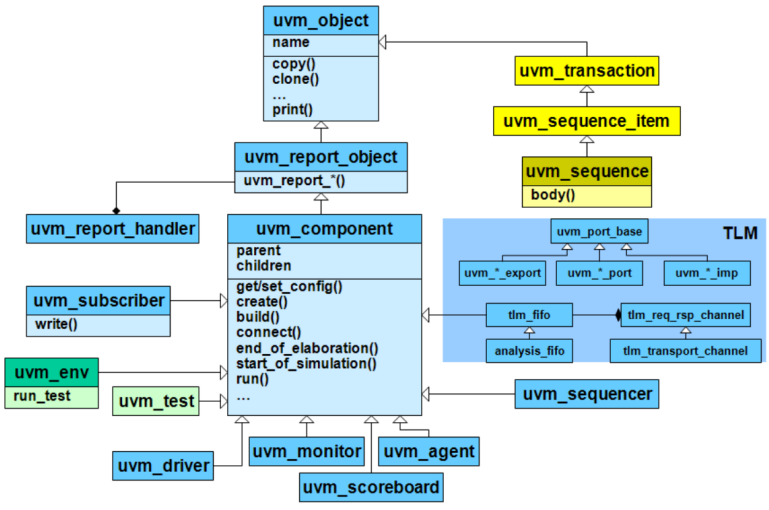


Figura 3.1: Jerarquía de clases de UVM [14].

En la figura 3.1 se muestra cómo las clases de UVM están organizadas de manera jerárquica. Las clases derivadas de uvm_component proporcionan todos los elementos necesarios para desarrollar un entorno de verificación robusto y funcional. Este enfoque orientado a

clases permite aprovechar funcionalidades heredadas de clases predefinidas, lo que facilita la creación de un entorno de verificación eficiente y estructurado. Entre estas funcionalidades se incluyen las fases de verificación, callbacks, generación de informes y bibliotecas de secuencias, todas las cuales fueron mencionadas anteriormente. Estas funciones predefinidas permiten ahorrar tiempo al verificador, quien solo debe centrarse en utilizarlas, sin necesidad de implementarlas desde cero.

En base a lo anterior, un entorno de verificación es una herramienta fundamental para asegurar la corrección funcional de los diseños de hardware y software. Generalmente, este entorno incluye componentes encargados de generar estímulos de prueba, aplicarlos al diseño bajo prueba, recolectar las respuestas y compararlas con los resultados esperados [15], dichos componentes usan el marco de clases de UVM para ser generados, conectados y simulados. En la figura A.1 se puede apreciar la estructura básica de un ambiente de verificación.

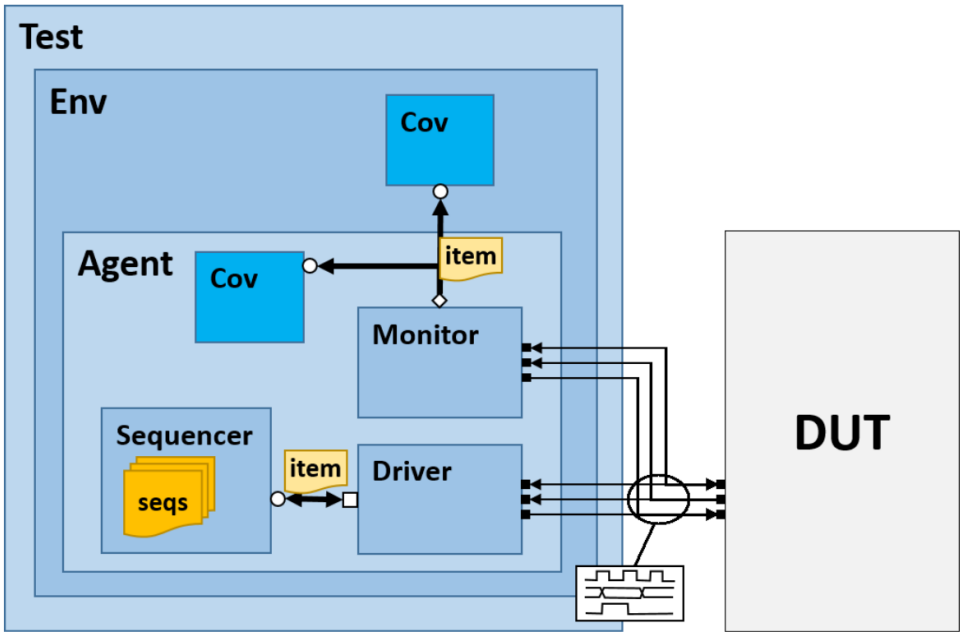


Figura 3.2: Estructura básica de un ambiente de verificación en UVM [16].

- **Driver:** recibe instrucciones y las interpreta para generar estímulos en el DUT.
- **Monitor:** el monitor recibe las salidas del DUT, las analiza y las envía a bloque encargado de verificarlas.
- **Agent:** el agente es el bloque que se encarga de generar estímulos y monitorear las señales, este bloque está compuesto por el driver y el monitor.
- **Sequencer:** el secuenciador es el bloque que se encarga de enviar las instrucciones al driver.

- **Scoreboard:** su propósito general es comparar las salidas del DUT con las salidas esperadas generadas mediante un modelo generador de predicciones (golden reference).
- **Env:** en este bloque se incluyen todos los componentes necesarios para generar estímulos y validar el funcionamiento del DUT.
- **Test:** es una instancia específica de un ambiente de verificación que configura, ejecuta y evalúa el DUT bajo condiciones específicas.

3.1.1. Interfaces

En SystemVerilog, las clases no pueden acceder directamente a señales a menos que estén declaradas dentro del ámbito de un módulo o interfaz que defina esas señales, lo cual puede limitar la flexibilidad del ambiente. No obstante, es posible que las clases se conecten a señales dentro de una interfaz utilizando un manejador de interfaz virtual. De este modo, una clase puede leer o modificar los valores de las señales en la interfaz, así como invocar tareas o funciones definidas en ellas [16]. Dichas interfaces son usadas principalmente en la implementación del driver que se encarga de generar los estímulos y comunicarlos con el DUT y el monitor que se encarga de escuchar las salidas del DUT.

3.1.2. Sequencer

Sequence item

Una transacción de alto nivel de abstracción que permite transmitir las instrucciones necesarias al driver para que éste las interprete y genere los estímulos en el DUT, los cuáles son dependientes de la prueba o test que se están ejecutando en el momento, el sequence item es una clase, por lo tanto, puede encapsular funciones que permiten hacer cálculos de datos necesarios para los estímulos e incluso predicciones [16].

Secuencia

Una secuencia en UVM es una entidad basada en clases que representa uno o más estímulos, conocidos como ítems de secuencia no sequence items, que son ejecutados por un driver. Las secuencias pueden organizarse en jerarquías que permiten la abstracción progresiva de estímulos y también pueden usar aleatorización controlada para generar variaciones de estímulos más complejas que las que se obtendrían con una sola transacción. Estas secuencias pueden modelar tanto una serie temporal de estímulos como estímulos paralelos que se aplican a múltiples interfaces de forma simultánea o independiente. Además, se pueden diseñar de forma modular para que sea más sencillo adaptarlos a cualquier caso de uso del DUT. UVM facilita la creación y gestión automática de secuencias, que en su forma más básica operan como llamadas a funciones. En este proceso, la secuencia

coloca un sequence item en el secuenciador a través de una función, mientras que el driver recoge este dato mediante otra función. De esta manera, el sistema logra mantenerse sincronizado, asegurando una interacción fluida y coordinada entre los estímulos generados y la respuesta del driver [16].

3.1.3. Test

Una prueba o un test es una clase que hace una instancia del ambiente y se encarga de establecer las restricciones necesarias para poder verificar características o escenarios específicos del DUT.

Gracias a la modularidad que ofrece el ambiente de UVM, es posible desarrollar pruebas individuales para cada una de las funcionalidades que el DUT proporciona. Esto permite desarrollar un test para cada uno de los casos generales y de esquina que el sistema pueda presentar. De esta manera, se puede evaluar exhaustivamente cada uno de estos casos, asegurando que no presenten problemas en el futuro. La modularidad de UVM permite dividir el proceso de verificación en componentes independientes y reutilizables, lo que facilita el desarrollo de pruebas específicas para diferentes aspectos del diseño. Cada prueba puede centrarse en una funcionalidad particular del DUT o en un escenario específico de uso, lo que proporciona una cobertura de verificación completa y detallada. Al desarrollar pruebas individuales para cada caso de esquina, se pueden identificar y solucionar posibles problemas o errores en el diseño de manera anticipada. Esto ayuda a mejorar la calidad y la fiabilidad del sistema, ya que se asegura que todas las funcionalidades del DUT funcionen correctamente bajo una variedad de condiciones de funcionamiento aleatorizadas [12].

3.1.4. Ambiente

Un ambiente de verificación en UVM tiene la responsabilidad de encapsular todos los bloques o componentes necesarios para llevar a cabo la verificación del DUT. Esto incluye no solo la inicialización de los componentes, sino también la conexión de los puertos de comunicación TLM (Transaction-Level Modeling), que permiten la interacción entre dichos componentes de manera eficiente y abstracta.

El entorno generalmente incluye varios elementos fundamentales como agentes, secuenciadores, drivers, monitores, y el scoreboard. Cada uno de estos módulos desempeña un papel clave en el proceso de verificación. Los secuenciadores generan las secuencias de transacciones, los drivers las traducen en actividad de señales hacia el DUT, los monitores observan la actividad en el DUT y crean modelos abstractos de las transacciones observadas, y el scoreboard compara los resultados reales con los esperados.

La conexión de los puertos TLM es esencial para que la comunicación fluya entre estos componentes. A través de estos puertos, las transacciones pueden moverse de un componente a otro sin necesidad de tener una conexión física directa [16].

3.1.5. Scoreboard

La función del Scoreboard es verificar si el DUT está operando correctamente. Aunque suele ser la parte más compleja de desarrollar en un testbench, su implementación se puede dividir en dos pasos principales: primero, definir claramente cuál es la funcionalidad correcta del DUT. Luego, el scoreboard compara los resultados obtenidos del DUT con los resultados esperados. La arquitectura más eficiente para un scoreboard implica separar las tareas de predicción y evaluación, lo que maximiza la flexibilidad al permitir la sustitución de los modelos de predicción y evaluación [16].

El scoreboard genera las predicciones utilizando una referencia de oro, conocida como golden reference. Esta referencia de oro es una versión del DUT implementada en software que replica el comportamiento del dispositivo, pero sin consumir tiempo de simulación. Gracias a esto, se pueden comparar los resultados reales obtenidos del DUT con los resultados generados por el predictor, permitiendo una validación eficiente del correcto funcionamiento del dispositivo [17].

3.1.6. Agente

Driver

El driver en UVM tiene la tarea de gestionar la comunicación de transacciones con la secuencia a través de puertos TLM y de convertir los sequence items en actividad de señales a nivel de pines al comunicarse con el DUT mediante una interfaz virtual. Un puerto TLM se encarga de transferir tipos de datos entre bloques o clases, por ejemplo, la transmisión de un paquete desde un monitor a un scoreboard.

En su funcionamiento típico, el driver recibe un sequence item y usa esa información para controlar las señales hacia el DUT. En algunas aplicaciones, también puede recibir respuestas del DUT a nivel de pines y convertirlas de nuevo en sequence items para completar la transacción con la secuencia. Adicionalmente, un driver puede responder en modo subordinado, reaccionando a la actividad de señales en la interfaz y comunicándose con una secuencia que le envía una transacción de respuesta para completar el ciclo de protocolo [16].

Monitor

Al igual que el Driver, el Monitor es parte de un agente. Ambos, Monitor y Driver, transforman la actividad de las señales en una representación abstracta, pero la diferencia principal es que el Monitor siempre opera en modo pasivo, es decir, no controla las señales en la interfaz. El Monitor utiliza una interfaz virtual para comunicarse con las señales del DUT y detectar patrones de protocolo en ellas. Cuando detecta un patrón, genera un modelo abstracto de transacción y lo envía a los componentes que están interesados en esa información, en este caso el scoreboard, que posteriormente se encargará de realizar el análisis de dicha transacción para verificar que el comportamiento es el esperado [16].

3.1.7. Aleatorización controlada

En el ámbito de la verificación, se implementan una serie de prácticas establecidas para aumentar de manera significativa la cobertura de prueba del RTL. Una de estas prácticas fundamentales es la aleatorización, la cuál se utiliza principalmente para abarcar una amplia gama de posibles estímulos que el RTL podría recibir durante su funcionamiento. La aleatorización permite generar estímulos de manera aleatoria, lo que brinda la oportunidad de explorar escenarios de prueba diversos y no predecibles. Esto es especialmente útil para exponer el RTL a condiciones de funcionamiento que podrían no ser evidentes en casos de prueba estáticos o predefinidos. Además, es posible establecer límites para la aleatorización, definiendo rangos o restricciones específicas para los valores de entrada. Esto asegura que los estímulos generados estén dentro de parámetros válidos y relevantes para el diseño. Otra ventaja de la aleatorización es su capacidad para ejecutar una cantidad prácticamente infinita de semillas durante la prueba. Esto amplía significativamente la variedad de escenarios de prueba explorados, aumentando así la probabilidad de detectar errores y comportamientos inesperados en el RTL [12].

3.1.8. Cobertura

La cobertura es un componente esencial en el proceso de verificación, ya que ofrece una visión clara de los resultados de las pruebas y permite evaluar si se han explorado todas las posibilidades de funcionamiento del sistema. El objetivo principal es alcanzar un nivel de cobertura cercano al 100 %, lo que implica cubrir casi todas las condiciones y escenarios posibles. Para lograr este objetivo, es crucial utilizar cantidades significativas de semillas en cada prueba. Una semilla es una configuración aleatoria de las características del DUT y del ambiente, cada semilla es única y siempre configurará el ambiente al estado que corresponde a esa semilla. Estas semillas se emplean para generar una variedad de estímulos que pondrán a prueba el DUT en diferentes situaciones. La idea es que, al ejecutar estas pruebas con un amplio conjunto de semillas, podamos abordar una amplia gama de casos de prueba y situaciones de funcionamiento. La ejecución de semillas por cada una de las pruebas se llama regresión. El deseo es completar una cantidad considerable de pruebas sin encontrar errores durante la simulación. Esto dará la confianza de que el diseño ha sido verificado en su mayoría y que es altamente probable que funcione correctamente en la práctica [12].

La cobertura abarca covergroups, coverpoints y cobertura cruzada, todos elementos importanes para la cobertura en UVM. Los covergroups establecen los modelos de cobertura, mientras que los coverpoints definen objetivos de cobertura específicos dentro de cada covergroup. Por otro lado, la cobertura cruzada (cross coverage) permite medir simultáneamente los valores observados en dos o más coverpoints. Esta técnica es crucial para analizar interacciones y combinaciones de variables, ya que proporciona información valiosa sobre eventos específicos, como errores en transacciones, al registrar todas las combinaciones posibles de valores [18, 19].

En el presente proyecto se analizan tres tipos de cobertura: la cobertura toggle, la cobertura FSM y la cobertura funcional. La cobertura toggle es una métrica de código utilizada para contabilizar cuántas veces cada bit de un registro o señal ha cambiado de valor (de cero a uno y de uno a cero). La cobertura FSM (Finite State Machine) mide la frecuencia con la que se visitan los estados en una máquina de estados finitos, así como el número de transiciones entre estados adyacentes y las secuencias específicas de dichas transiciones. Finalmente, la cobertura funcional evalúa qué tan bien se han ejercido las funcionalidades especificadas de un diseño durante las pruebas, asegurando el cumplimiento de los requisitos funcionales [18].

3.1.9. Regresiones

El ambiente de verificación ofrece la posibilidad de llevar a cabo simulaciones que comprenden una cantidad significativa de pruebas y semillas. La duración de estas simulaciones puede variar en función del tamaño y la complejidad del proyecto, extendiéndose a lo largo del tiempo. Sin embargo, usando buenas prácticas de codificación, es posible disminuir el tiempo de simulación. Al usar software para la programación del ambiente es posible generar un script capaz de ejecutar una lista de pruebas con su cantidad respectiva de semillas. Esta práctica ofrece una forma más sencilla de ejecutar las simulaciones, ya que proporciona una visión general clara de todas las pruebas que serán ejecutadas y de la cantidad de semillas asociadas a cada una. Además, esta estructura brinda la flexibilidad necesaria para realizar ajustes según las necesidades del proyecto. Se pueden agregar nuevas pruebas, modificar la cantidad de semillas o ajustar otros parámetros según lo requiera el desarrollo del proyecto [12].

3.2. Dispositivo bajo verificación

AXI es una especificación de interfaz que define la conexión entre dos bloques, solamente existen dos tipos de interfaces, maestro y subordinado, dichas interfaces contienen las mismas señales, lo que permite que la integración en diferentes circuitos integrados sea relativamente simple. La conexión directa proporciona un ancho de banda máximo sin lógica extra y con un solo protocolo en el sistema es mucho más sencillo validar el integrado. Las figuras 3.3 y 3.4 ilustran lo mencionado anteriormente [10].

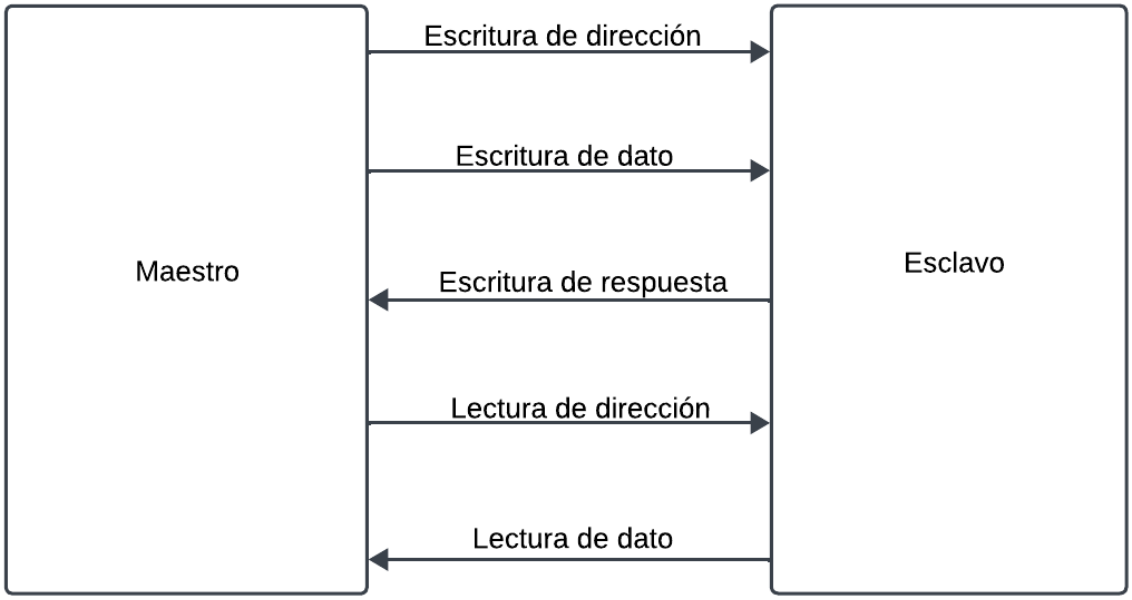


Figura 3.3: Interface AXI.

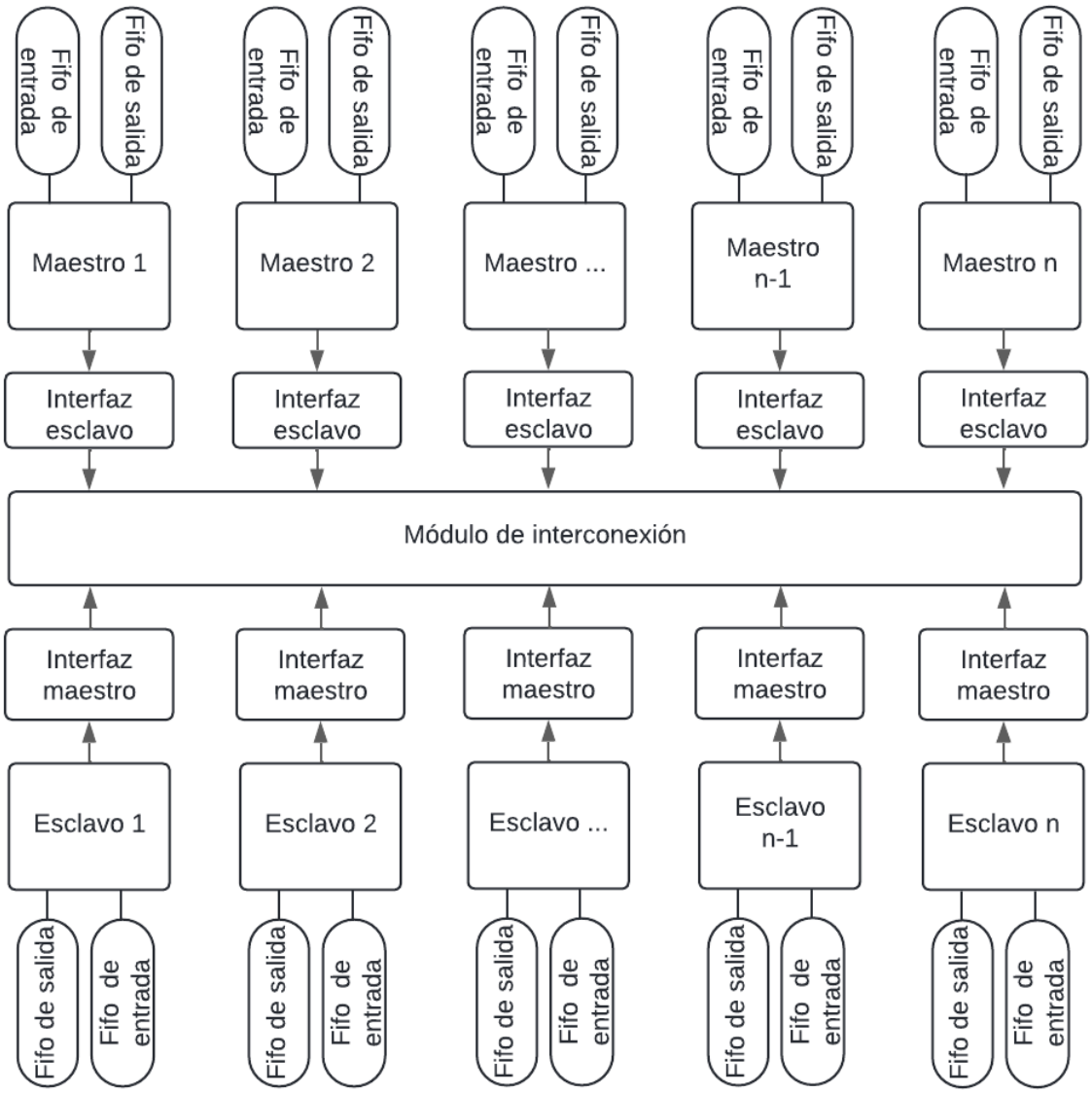


Figura 3.4: Interconexión de maestros y subordinados.

Las características básicas del módulo de interconexión AXI ilustrado en la figura 3.4 se encuentran listadas a continuación [10]:

- Capacidad de escritura y lectura a una dirección de memoria desde cualquier maestro a cualquier subordinado existente en el módulo.
- Capacidad de tener a todos los maestro y subordinados ejecutando transacciones de forma simultánea.
- Capacidad de realizar de 1 a 256 escrituras o lecturas en una sola transacción.
- Cada escritura o lectura puede tener un tamaño de 1 a 128 bytes.

- Capacidad de configurar de que forma se van a almacenar los datos en memoria en las transacciones de escritura.
- Capacidad de definir un rango de prioridad para las transacciones que se están ejecutando, dicho rango es de 1 hasta 16.
- Las transacciones pueden colocarse de forma consecutiva sin necesidad de esperar a que la transacción anterior haya finalizado.
- Uso de protocolos de handshake para asegurar seguridad en cada una de las transacciones.
- Uso de ID's y direcciones de memoria para reconocer la fuente y el destino de las transacciones respectivamente.
- Capacidad de indicar cuáles bytes de la transacción son bloqueados y cuáles son enviados.

Como se muestra en la figura 3.3, la conexión entre el maestro y el subordinado generalmente se realiza a través de tres canales: escritura, lectura y respuesta. Cada canal cuenta con dos señales, VALID y READY, que indican cuándo se van a enviar los datos. Los datos sólo pueden ser enviados cuando ambas señales están activas. Para iniciar una transacción de escritura o lectura, primero se debe proporcionar la dirección donde se leerán o escribirán los datos. Una vez que la dirección ha sido establecida, se procede con la transacción. Al finalizar la transacción, el canal de respuesta proporciona un dato que indica el estado de la operación, señalando si fue exitosa o si ocurrió algún error [10]. En las figuras 3.5, 3.6, 3.7 y 3.8 se ilustra el funcionamiento básico del protocolo, la ejecución de escrituras y lecturas únicas o en secuencia [10].

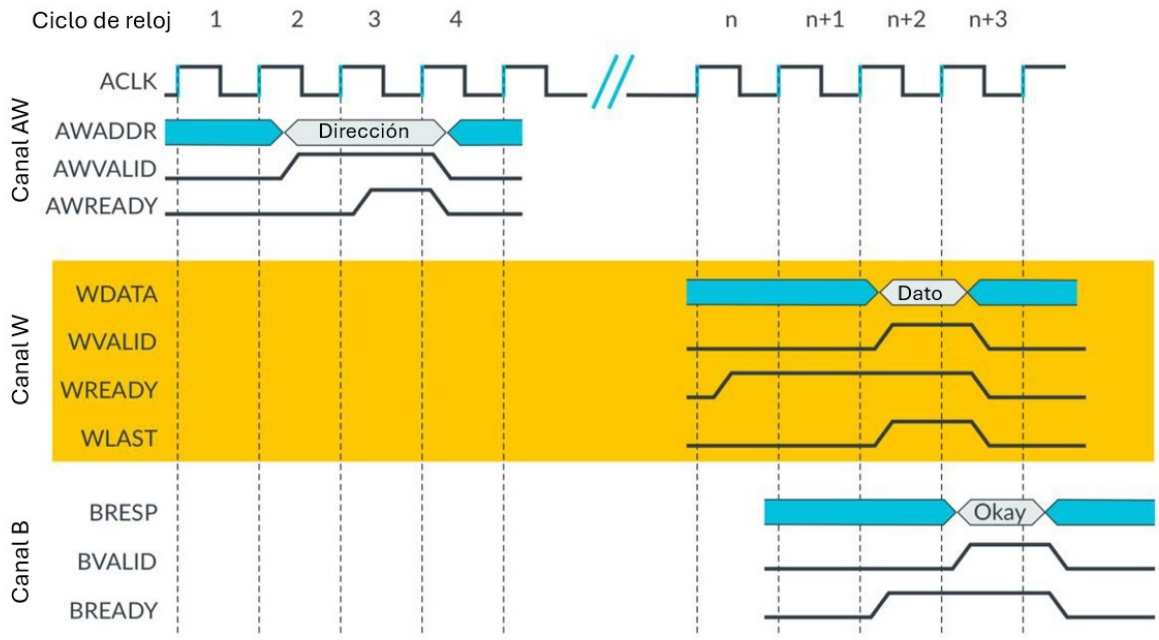


Figura 3.5: Transacción de escritura de un solo dato.

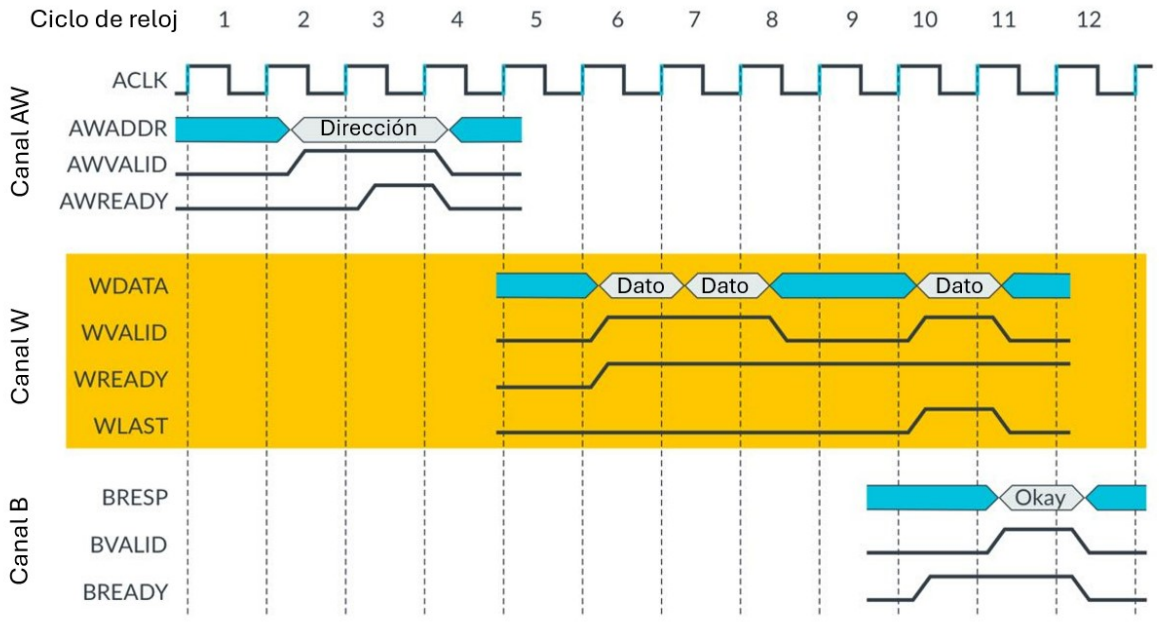


Figura 3.6: Transacción de escritura de una secuencia de datos.

Según las figuras 3.6 y 3.5:

- **AW channel**: canal encargado de la dirección
- **W channel**: canal encargado de los datos de escritura.
- **B channel**: canal encargado de la señal de respuesta.

- **ACLK**: Reloj.
- **AWADDR**: Dirección de escritura.
- **AWVALID**: Señal que indica que la dirección está lista para ser enviada por el maestro.
- **AWREADY**: Señal que indica que la dirección está lista para ser recibida por el subordinado.
- **WDATA**: dato o datos que serán escritos.
- **WVALID**: señal que indica que los datos ya están listos para ser enviados por el maestro.
- **WREADY**: señal que indica que los datos ya están listos para ser recibidos por el subordinado.
- **WLAST**: señal que indica cuando ya se escribió el último dato.
- **BRESP**: dato de respuesta al finalizar la transacción.
- **BVALID**: señal que indica que la señal está lista para ser enviada por el subordinado.
- **BREADY**: señal que indica que el maestro está listo para recibir la señal de respuesta.

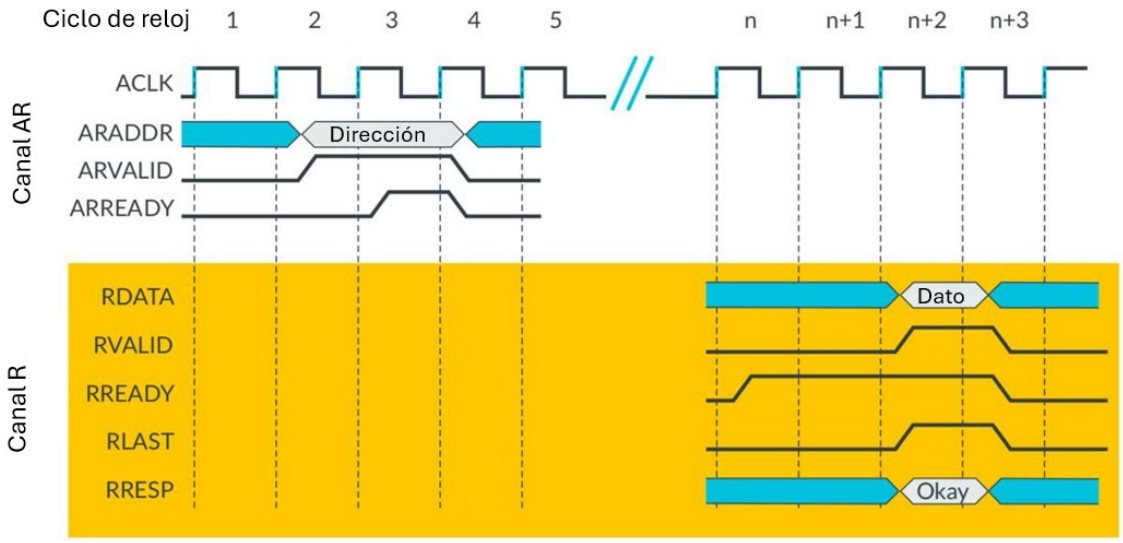


Figura 3.7: Transacción de lectura de un solo dato.

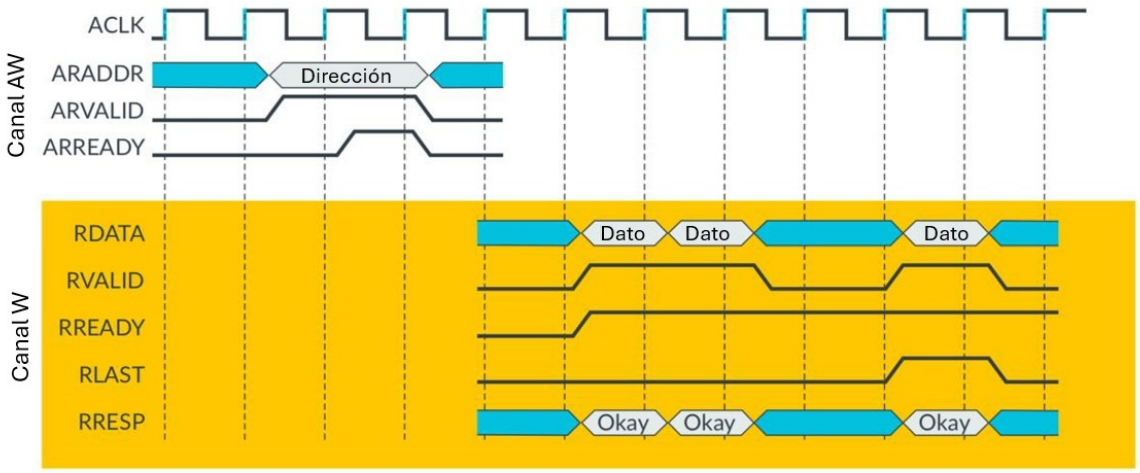


Figura 3.8: Transacción de lectura de una secuencia de datos.

Según las figuras 3.7 y 3.8:

- **AR channel**: canal encargado de la dirección de lectura.
- **R channel**: canal encargado de los datos de lectura.
- **ACLK**: Reloj.
- **ARADDR**: dirección de escritura.
- **ARVALID**: señal que indica que la dirección de lectura está lista para ser enviada por el maestro.
- **ARREADY**: señal que indica que el subordinado está listo para recibir la dirección de lectura.
- **RDATA**: dato o datos de lectura.
- **RVALID**: señal que indica que el subordinado está listo para enviar los datos de lectura.
- **RREADY**: señal que indica que el maestro está listo para recibir los datos de lectura.
- **RLAST**: señal que indica cuando ya se recibió el último dato de lectura.
- **BRESP**: dato de respuesta al finalizar la transacción.

Capítulo 4

Diseño del ambiente

4.1. Ambiente

En la sección 3.1.4, se explicó el funcionamiento básico de un ambiente UVM y los componentes que lo componen, como el (o los) agente(s) y el scoreboard. A nivel de diseño, este ambiente debe cumplir con las siguientes funciones esenciales:

- Enviar paquetes con la información necesaria a los drivers, permitiendo que estos generen los estímulos adecuados para el DUT.
- Transmitir los datos recopilados por los monitores al scoreboard, facilitando así el proceso de verificación y comparación de resultados.
- Enviar paquetes desde los monitores hacia el colector de cobertura para evaluar la completitud de la verificación.

En esencia, el ambiente tiene como función principal establecer las conexiones adecuadas entre los componentes instanciados internamente para permitir una comunicación fluida y un eficiente flujo de información.

Las conexiones de estos componentes se muestran en la figura A.1. Esta ilustración del ambiente, presentada en alto nivel, facilita la comprensión de cómo se conectan los componentes y los tipos de paquetes de información que manejan. En la figura, las conexiones entre los agentes, el scoreboard y el colector de cobertura representan puertos TLM o puertos de análisis, encargados de transmitir paquetes del tipo *xaction* al scoreboard. La descripción de las variables internas de estos paquetes se detalla en la sección A.3.2, también ubicada en el apéndice A.

De manera más detallada, el DUT es un módulo de interconexión AXI que incluye 4 maestros y 4 subordinados. Por ello, se definió un agente para cada maestro y subordinado, cada uno con su propio driver y monitor. Dado que el funcionamiento de los maestros

y subordinados es diferente, sus respectivos drivers y monitores también deben comportarse de manera distinta. Por lo tanto, fue necesario desarrollar un driver y un monitor específicos para cada tipo de dispositivo, el funcionamiento detallado de los mismos se encuentra en las secciones 4.2 y 4.3. Finalmente, cada monitor tiene un puerto de análisis conectado al scoreboard y también cada puerto de análisis de los monitores del maestro específicamente está conectado al colector de cobertura, de esta forma, cada monitor es capaz de enviar la transacción que se está ejecutando en ese momento y se puede registrar la cobertura.

Otras consideraciones importantes del ambiente se pueden consultar en la sección A.3.3.

4.2. Agente del maestro

El agente del maestro encapsula tanto al driver como al monitor del maestro. Estos dos componentes se comunican a través de la interfaz tipo DUT, tal como se describe en la sección A.3.1 del apéndice A. El agente simula un periférico conectado al dispositivo bajo prueba (DUT) que desea realizar transacciones. Por lo tanto, genera estímulos mediante el driver y evalúa las respuestas a través del monitor. En la figura A.3, se muestran sus bloques internos y los puertos que utiliza para comunicarse con el DUT, específicamente el maestro. El puerto de análisis envía el paquete *xaction* al scoreboard y al colector de cobertura, mientras que la conexión del driver con el sequencer permite la transmisión de paquetes necesarios desde la secuencia al maestro para generar los estímulos. Además, los bloques Sim FF read y Sim FF cmd son FIFOs que almacenan los datos leídos para las transacciones de lectura y que envían estímulos al DUT, respectivamente.

En las figuras 4.2 y 4.1 se ilustra el flujo que sigue el driver para enviar los estímulos correspondientes al DUT. Se observa cómo el driver espera un *item*, evalúa el tipo de transacción y luego ajusta las señales de la interfaz según corresponda. Además, se aprecia que el driver permanece a la espera de la señal *pop* en la interfaz para enviar el dato solicitado en el momento en que dicha señal se activa.

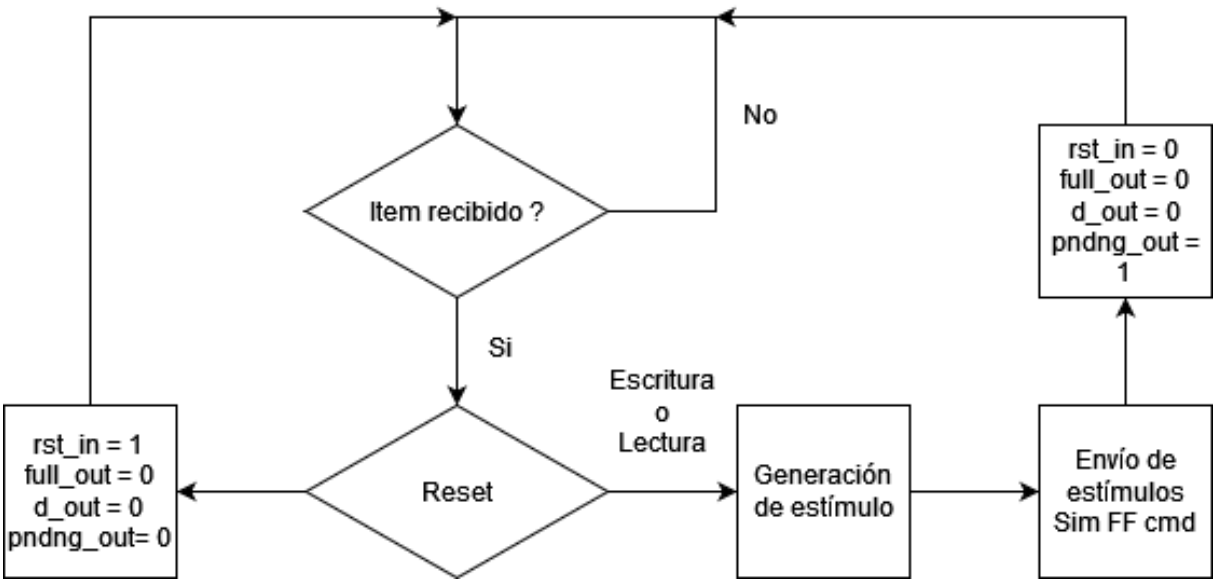


Figura 4.1: Diagrama de flujo de la interpretación de la secuencia en el driver del maestro.

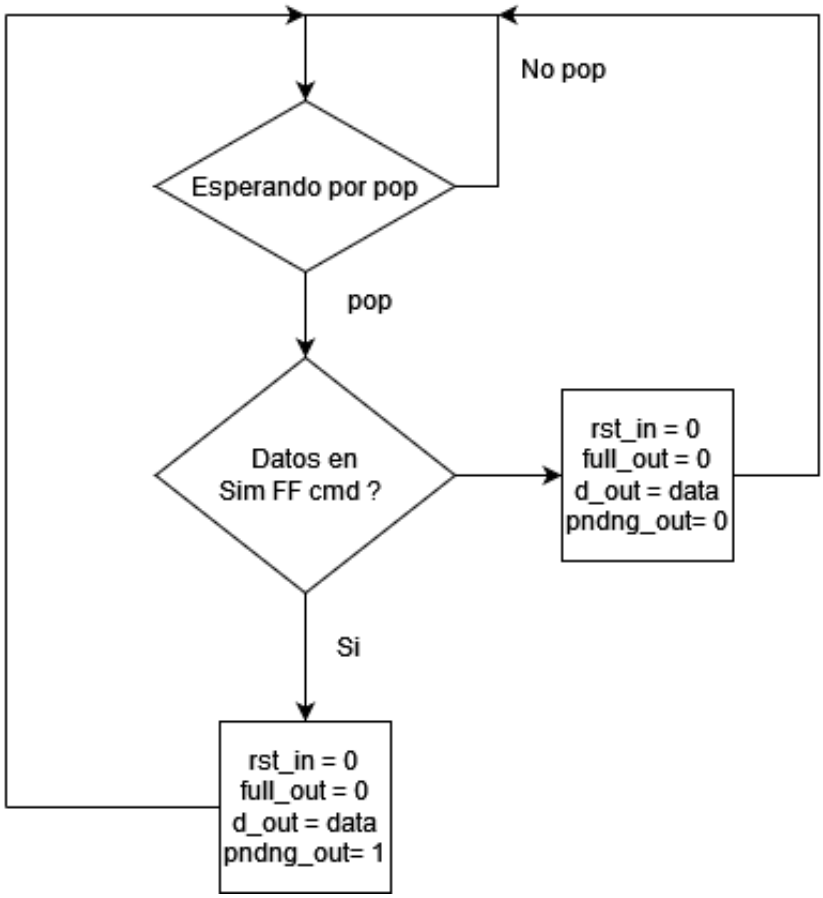


Figura 4.2: Diagrama de flujo del pop del driver en el maestro.

La figura 4.3 muestra cómo el ambiente interpreta las señales del DUT para luego verificarlas. Se observa cómo se utilizan las dos interfaces, la del DUT y la de AXI, descritas

en la sección A.3.1 para ambos casos de transacción (escritura o lectura), para comparar los datos capturados en ambas interfaces y verificar que son equivalentes. Finalmente, el paquete AXI se envía al *scoreboard* y al recolector de cobertura para asegurar que el paquete llegó a su destino y realizar un muestreo de las señales para mejorar la cobertura. Es importante destacar que el monitor se mantiene verificando constantemente el cumplimiento del protocolo AXI.

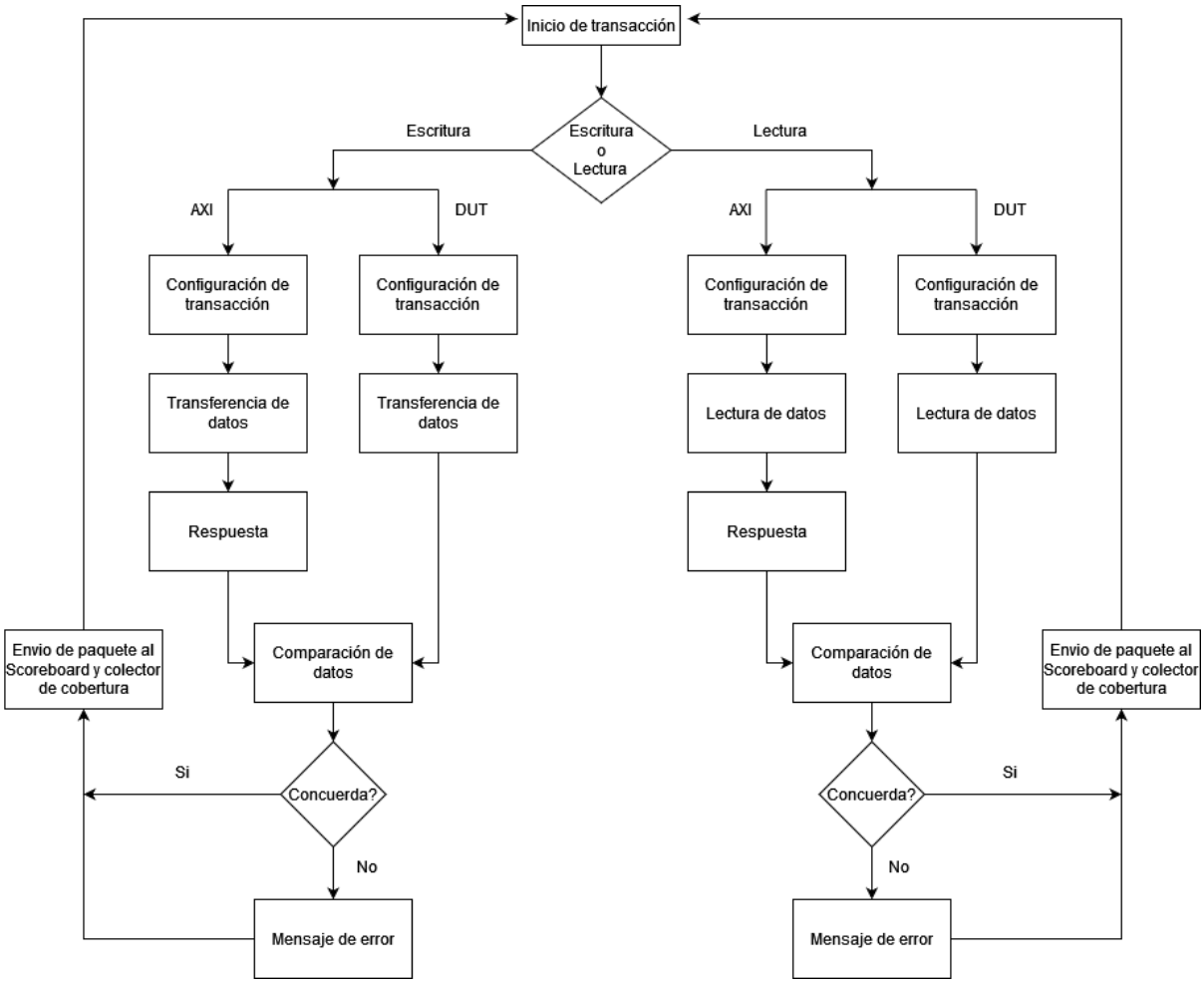


Figura 4.3: Diagrama de flujo del monitor del maestro.

4.3. Agente del subordinado

Al igual que el agente del maestro, el agente del subordinado está compuesto por un driver y un monitor, y representa un periférico destino al cual se desea escribir o leer información. En este caso, el driver no recibe paquetes desde un sequencer; en su lugar, recibe un dato desde el monitor cuando se trata de una transacción de lectura. La función del driver es enviar datos, por lo que busca en una memoria interna que posee el agente en la dirección de lectura indicada por el monitor y transmite esa información al DUT. En caso de realizar una transacción de escritura, el agente solo escribe en la memoria interna

y no se comunica directamente con el DUT.

La ilustración de alto nivel se puede ver en la figura A.4 en el apéndice A, donde se aprecia un puerto de análisis que se dirige al scoreboard, transmitiendo datos del tipo *xaction*, así como un bloque que también transmite, por un puerto de análisis, el dato de lectura que se desea transferir al subordinado. Además, al igual que el agente del maestro, se pueden observar bloques llamados Sim FF cmd_w_or_r y Sim FF data. Estos bloques representan FIFOs encargadas de recibir datos de escritura junto con su dirección, o solo la dirección para los datos de lectura, mientras que el bloque Sim FF data simplemente transmite los datos de lectura.

La figura 4.4 ilustra el flujo seguido por el driver del subordinado para comunicarse con el DUT. Este componente se utiliza principalmente para enviar los datos de lectura solicitados por el DUT. Para simplificar el diseño del ambiente y considerando el amplio rango de direcciones de memoria, el valor enviado al DUT será la misma dirección de memoria del dato solicitado. En la figura se observa cómo, tras recibir la dirección de lectura, el driver modifica las señales de la interfaz y coloca la señal *pending* en 1. Esto permite que el DUT detecte un dato pendiente en la FIFO simulada dentro del agente, activando así la señal *pop* y permitiendo que se envíe el dato de lectura al DUT.

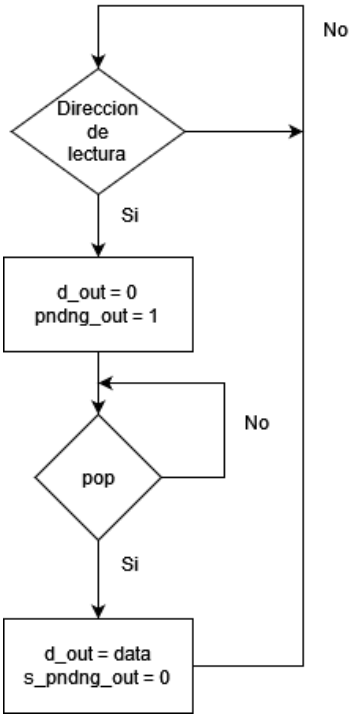


Figura 4.4: Diagrama de flujo del driver del subordinado.

En la figura 4.5 se observa que el flujo del monitor del subordinado es similar al flujo del monitor del maestro, aunque más simplificado y sin necesidad de enviar el paquete de transacción al recolector de cobertura. En resumen, el monitor detecta la inicialización de una transacción, recopila los datos relevantes durante el flujo en las interfaces AXI y DUT, y luego compara dichos datos para verificar su equivalencia. Si los datos no coinciden, se

genera una señal de error y finalmente el paquete es enviado al *scoreboard* para verificar que se haya enviado desde algún maestro.

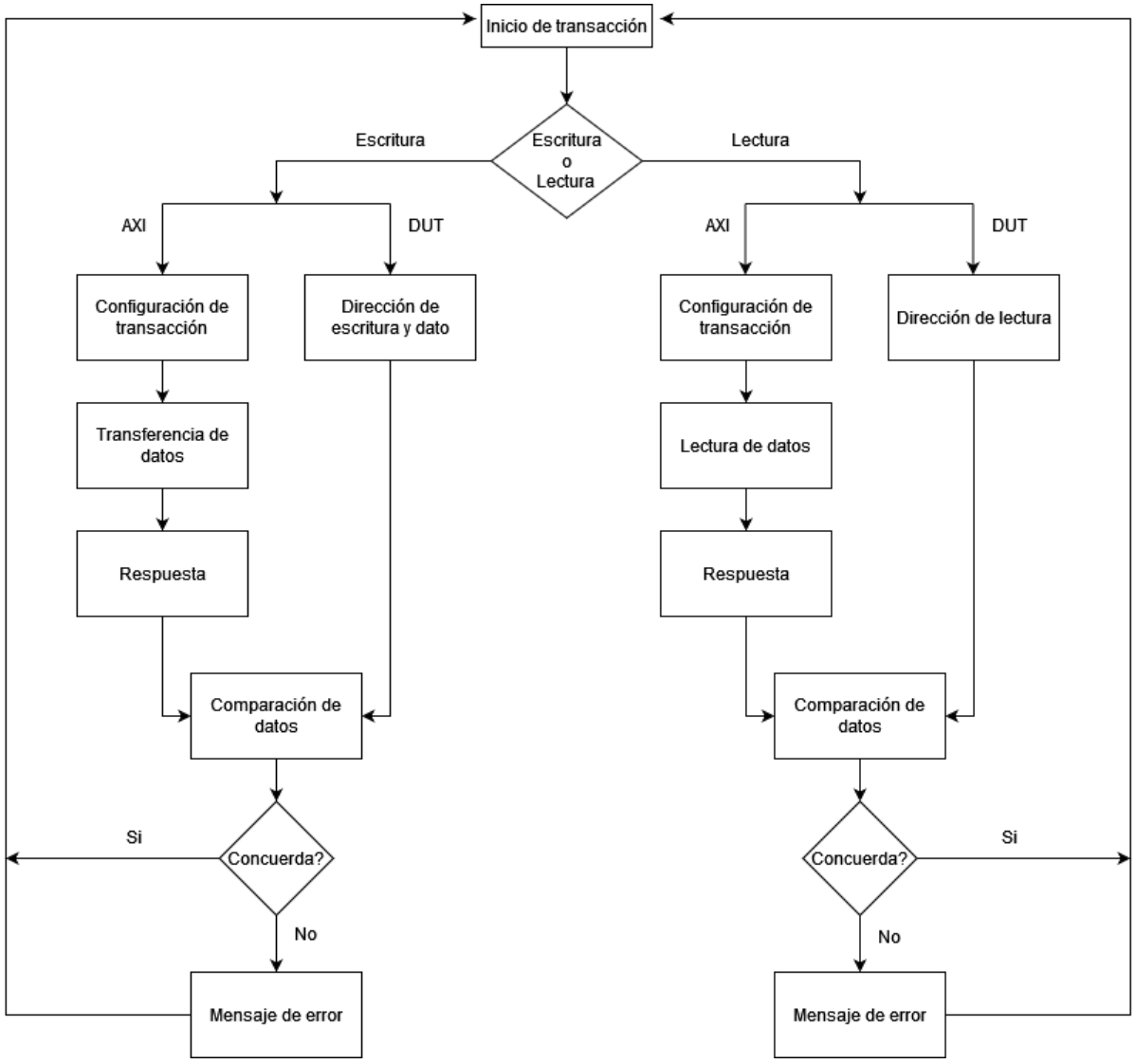


Figura 4.5: Diagrama de flujo del monitor del subordinado.

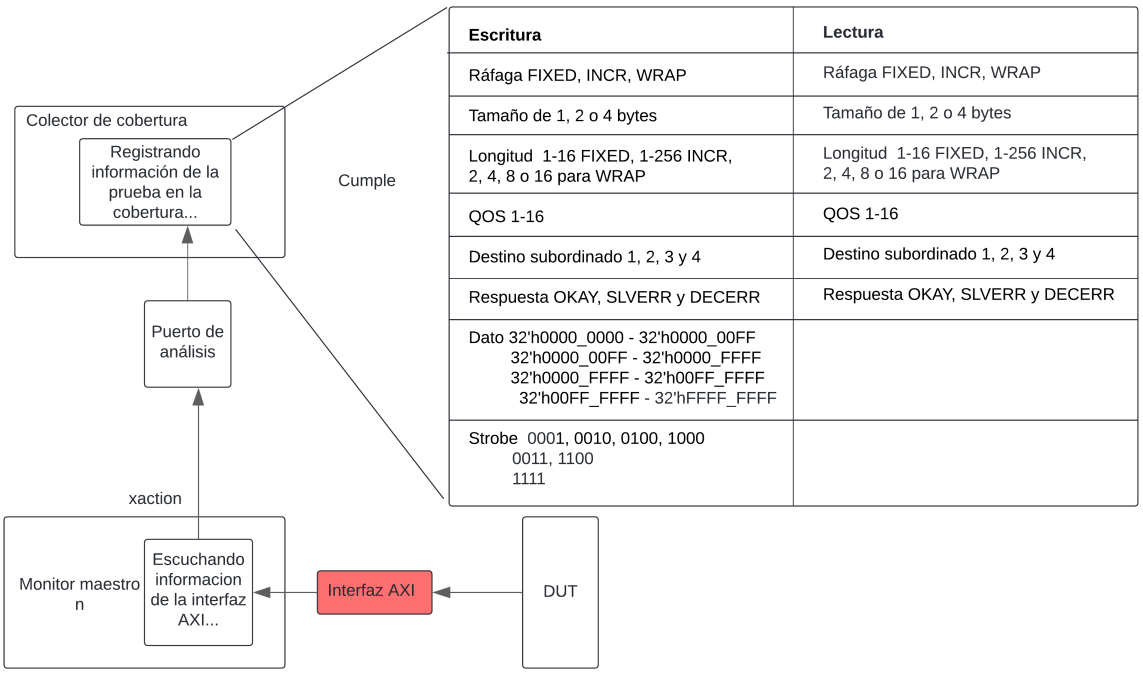
4.4. Colector de cobertura

El recolector de cobertura es un componente encargado de recibir paquetes de tipo *xaction* que contienen información sobre el tipo de transacción que ejecuta cada monitor. Como se explicó anteriormente, cada monitor envía un *xaction* al recolector de cobertura a través de un puerto de análisis cada vez que se realiza una transacción. Internamente, el recolector de cobertura muestrea esta información y la almacena en una base de datos, lo que permite, al finalizar una regresión o una prueba individual, mostrar el porcentaje de casos cubiertos.

En la imagen A.6 del A, se observa cómo los bloques rojos representan los maestros

conectados al recolector de cobertura, mientras que el bloque rojo interno indica el proceso de muestreo

En la siguiente figura se ilustra el proceso de muestreo, desde que el monitor detecta la transacción del DUT y la encapsula en un paquete, hasta que la envía al recolector de cobertura, que luego muestrea la información de dicho paquete. Además, se pueden observar los *covergroups* que corresponden al tipo de transacción (escritura o lectura) y los *coverpoints* que corresponden a los elementos listados en la tabla:



Escritura	Lectura
Ráfaga FIXED, INCR, WRAP	Ráfaga FIXED, INCR, WRAP
Tamaño de 1, 2 o 4 bytes	Tamaño de 1, 2 o 4 bytes
Longitud 1-16 FIXED, 1-256 INCR, 2, 4, 8 o 16 para WRAP	Longitud 1-16 FIXED, 1-256 INCR, 2, 4, 8 o 16 para WRAP
QOS 1-16	QOS 1-16
Destino subordinado 1, 2, 3 y 4	Destino subordinado 1, 2, 3 y 4
Respuesta OKAY, SLVERR y DECERR	Respuesta OKAY, SLVERR y DECERR
Dato 32'h0000_0000 - 32'h0000_00FF 32'h0000_00FF - 32'h0000_FFFF 32'h0000_FFFF - 32'h00FF_FFFF 32'h00FF_FFFF - 32'hFFFF_FFFF	
Strobe 0001, 0010, 0100, 1000 0011, 1100 1111	

Figura 4.6: Covergroups y coverpoints del maestro.

Además de los *covergroups* también se hizo *crosscoverage*, detallado en la tabla 4.1:

Tabla 4.1: Cobertura cruzada

Escritura	Lectura
FIXED hacia el subordinado 0	FIXED hacia el subordinado 0
FIXED hacia el subordinado 1	FIXED hacia el subordinado 1
FIXED hacia el subordinado 2	FIXED hacia el subordinado 2
FIXED hacia el subordinado 3	FIXED hacia el subordinado 3
INCR hacia el subordinado 0	INCR hacia el subordinado 0
INCR hacia el subordinado 1	INCR hacia el subordinado 1
INCR hacia el subordinado 2	INCR hacia el subordinado 2
INCR hacia el subordinado 3	INCR hacia el subordinado 3
WRAP hacia el subordinado 0	WRAP hacia el subordinado 0
WRAP hacia el subordinado 1	WRAP hacia el subordinado 1
WRAP hacia el subordinado 2	WRAP hacia el subordinado 2
WRAP hacia el subordinado 3	WRAP hacia el subordinado 3

En la tabla 4.1 se puede apreciar la cobertura cruzada que se está realizando por cada uno de los maestros con el fin de ver que dichos casos se estén cubriendo dentro de la simulación.

4.5. Scoreboard

El *scoreboard* se conecta con los puertos de análisis de los monitores del maestro y del subordinado, permitiendo así verificar que los paquetes se transfieren correctamente de un maestro a un subordinado. Es importante señalar que la verificación del cumplimiento del protocolo se realiza en los monitores, antes de enviar los paquetes al *scoreboard*. En la figura 4.7 se muestra el algoritmo utilizado para verificar que cada transacción enviada por un maestro se encuentre en alguno de los subordinados. Internamente, el *scoreboard* dispone de un espacio específico para almacenar las transacciones de cada maestro y cada subordinado, por lo que es necesario iterar dentro de estos espacios.

En la figura se observa cómo, inicialmente, se selecciona un maestro y una transacción, la cual se compara con todas las transacciones presentes en cada subordinado para verificar su existencia. La comparación se realiza en aspectos específicos, tales como los datos escritos o leídos, el tipo de transacción, el tipo de *burst*, el valor de *len*, entre otros. Si al menos uno de estos aspectos no coincide, las transacciones no son equivalentes y se pasa a comparar con la siguiente transacción. Si ya se han verificado todas y no hay coincidencia, se concluye que la transacción no llegó a su destino. Si la transacción se encuentra en alguno de los subordinados, el ambiente confirma que fue enviada correctamente. Este proceso se repite para cada transacción de todos los maestros.

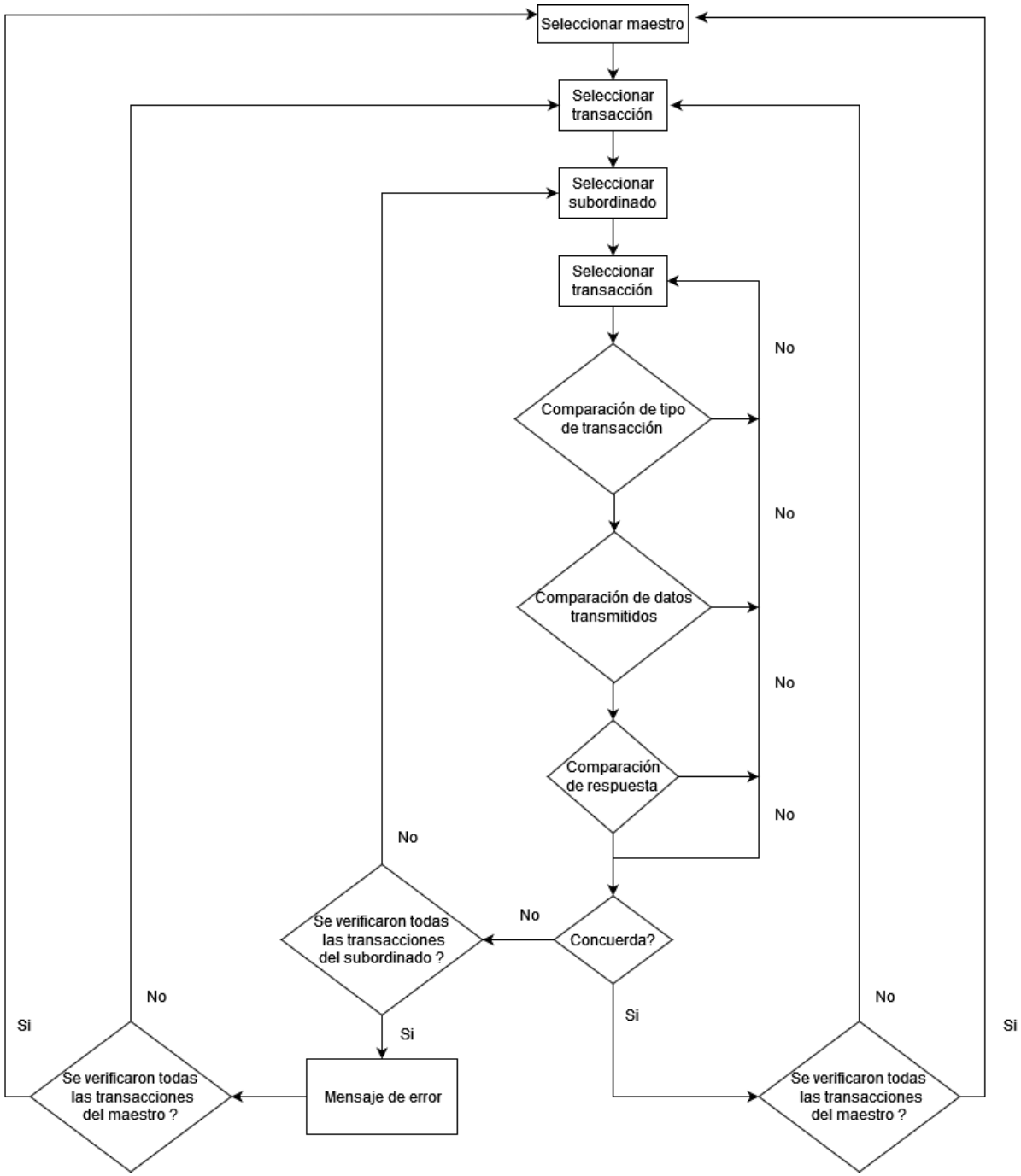


Figura 4.7: Diagrama de flujo del scoreboard.

Capítulo 5

Definición de las pruebas

Inicialmente, se definió la lista de características del DUT con el fin de desarrollar las pruebas necesarias para verificar su correcto funcionamiento. En la siguiente tabla se especifican dichas características y las pruebas diseñadas para validar cada escenario. Información más detallada sobre las pruebas y los escenarios se presenta en la sección A.1.2.

Características	**Prueba**
Transacciones de escritura o lectura con longitud (len) unitaria	▪ write_unique_test ▪ read_unique_test ▪ write_read_unique_test
Transacciones de escritura o lectura con tipo de burst FIXED	▪ write_burst_fixed_test ▪ read_burst_fixed_test ▪ write_read_burst_fixed_test
Transacciones de escritura o lectura con tipo de burst INCR	▪ write_burst_incr_test ▪ read_burst_incr_test ▪ write_read_burst_incr_test

Continúa en la siguiente página

Características	**Prueba**
Transacciones de escritura o lectura con tipo de burst WRAP	▪ write_burst_wrap_test ▪ read_burst_wrap_test ▪ write_read_burst_wrap_test
Transacciones de escritura con strobe y size de 1 a 4 bytes	▪ write_strobe_test
Transacciones de lectura con size de 1 a 4 bytes	▪ read_size_test
Transacciones de escritura o lectura con quality of service (qos)	▪ write_qos_test ▪ read_qos_test ▪ write_read_qos_test
Transacciones de escritura o lectura desde todos los maestros al mismo tiempo	▪ write_read_test
Correcto arbitraje cuando todos los maestros quieren transmitir a un solo subordinado	▪ all_for_one_test
Detección de errores en la configuración o accesos de memoria fuera de rango en las transacciones de escritura y lectura	▪ decerr_gen_test ▪ slverr_gen_test ▪ slverr_decerr_gen_test
Correcto reinicio después de levantar la señal de reset	▪ reset_test

Tabla 5.1: Características del DUT y pruebas que las verifican

En general, el criterio de cumplimiento de las pruebas es que estas confirmen el correcto funcionamiento de la característica para la cual fueron desarrolladas. Además, durante la simulación, se debe verificar el cumplimiento del protocolo AXI, asegurando que las transacciones lleguen correctamente a sus destinos y que los datos de las transacciones coincidan con las predicciones generadas. Más información sobre los criterios de cumplimiento por cada prueba se encuentra en la sección A.1.2.

Capítulo 6

Sistema de regresión

Para este proyecto, se desarrolló un sistema de regresiones o mejor conocido en la industria como continues integrations que permite realizar un seguimiento detallado de los resultados de las simulaciones. Este sistema centraliza la información en un directorio dedicado, recopilando los resultados de todas las pruebas, los datos de cobertura y los errores específicos encontrados, así como el comando utilizado para ejecutar cada simulación. La implementación se realizó en Python, empleando comandos de VCS para gestionar las simulaciones UVM y la cobertura. En la figura 6.1 se ilustra el directorio resultante tras ejecutar una regresión, donde se observa que se generó una carpeta para cada prueba ejecutada, además de una carpeta para la cobertura y otra que resume el resultado de la regresión, denominada *regression_ results*.

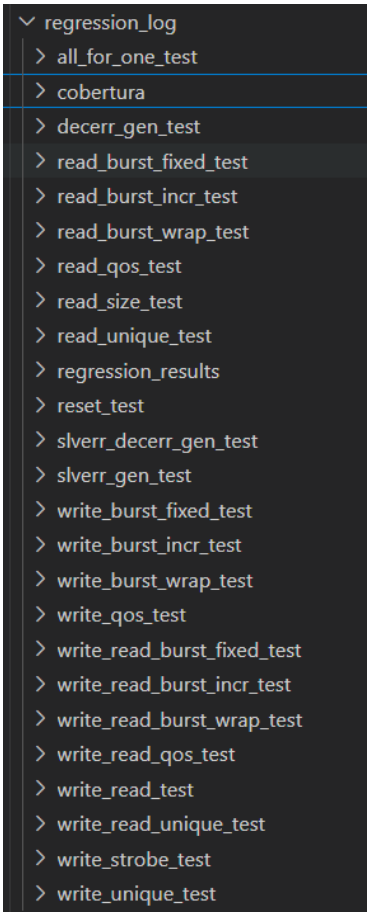


Figura 6.1: Directorio de resultado de regresiones.

Luego, en la figura 6.2, tomando como ejemplo la prueba *all_for_one_test*, se observa que dentro de esa carpeta se encuentran múltiples subcarpetas numeradas. Cada número corresponde a una de las semillas utilizadas en la ejecución de esa prueba.

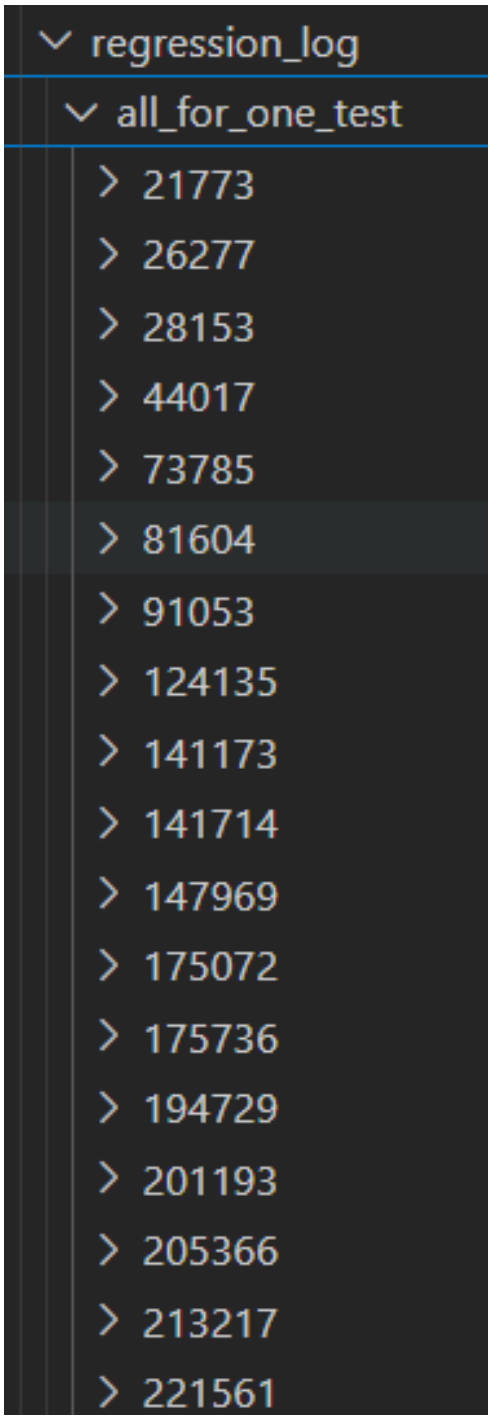


Figura 6.2: Carpetas para las semillas de la prueba.

Dentro de cada una de las carpetas correspondientes a las semillas, se encuentran dos archivos: *log.txt* y *postmortem.txt* como se aprecia en la figura 6.3. Estos archivos almacenan, respectivamente, los mensajes generados durante la simulación y el resultado de la misma (éxito o fallo), en la figura 6.4 se puede apreciar el contenido del archivo *log.txt*, en la figura 6.5 se aprecia el contenido del archivo *postmortem.txt* para el caso de una prueba que paso con éxito y la figura 6.6 para el caso de una prueba que fallo.

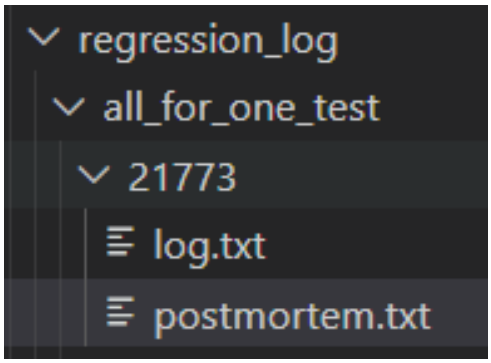


Figura 6.3: Archivo postmortem y log.

```
[m_mntr[0]] [1428425000] La transaccion numero 6 tuvo 0 errores
[m_mntr[0]] [1428425000] La transaccion numero 7 tuvo 0 errores
[m_mntr[0]] [1428425000] La transaccion numero 8 tuvo 0 errores
[m_mntr[0]] [1428425000] La transaccion numero 9 tuvo 0 errores
[m_mntr[0]] [1428425000] La transaccion numero 9 tuvo 0 errores
[m_mntr[0]] [1428425000] La transaccion numero 10 tuvo 0 errores
[m_mntr[0]] [1428425000] La transaccion numero 11 tuvo 0 errores
[m_mntr[0]] [1428425000] La transaccion numero 11 tuvo 0 errores
[m_mntr[0]] [1428425000] La transaccion numero 12 tuvo 0 errores
[m_mntr[0]] [1428425000] La transaccion numero 13 tuvo 0 errores
[m_mntr[0]] [1428425000] La transaccion numero 14 tuvo 0 errores
[m_mntr[0]] [1428425000] La transaccion numero 15 tuvo 0 errores
[m_mntr[0]] [1428425000] La transaccion numero 15 tuvo 0 errores
[m_mntr[0]] [1428425000] La transaccion numero 16 tuvo 0 errores
[m_mntr[0]] [1428425000] La transaccion numero 17 tuvo 0 errores
[m_mntr[0]] [1428425000] La transaccion numero 17 tuvo 0 errores
[m_mntr[0]] [1428425000] La transaccion numero 18 tuvo 0 errores
```

Figura 6.4: Contenido del archivo *log.txt*.

```
1  Test Name: all_for_one_test
2  Seed: 21773
3  Result: PASSED
4
```

Figura 6.5: Contenido del archivo *postmortem.txt* para el caso de una prueba que fue exitosa.

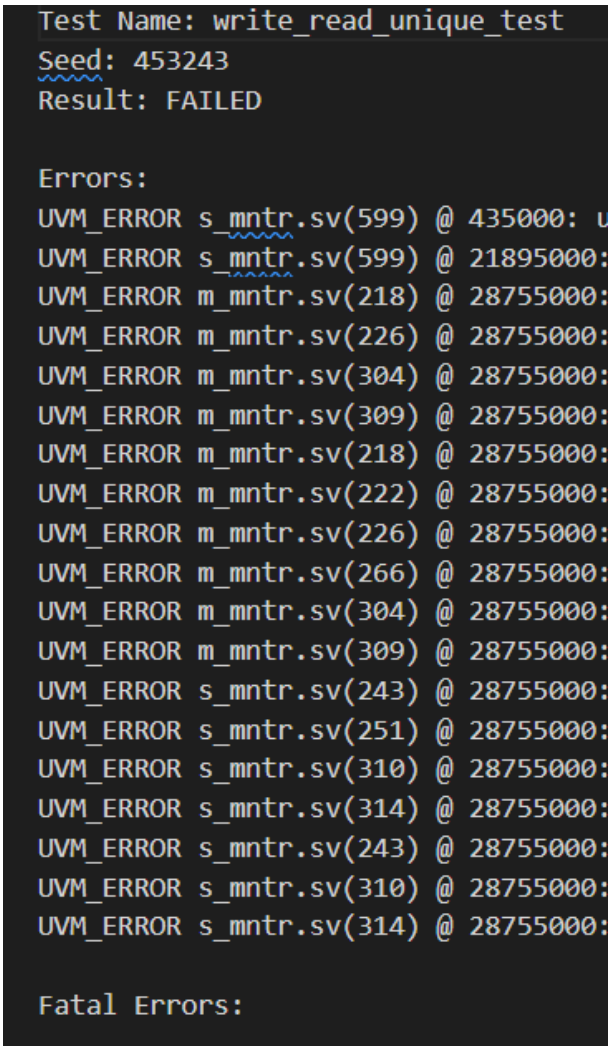


Figura 6.6: Contenido del archivo *postmortem.txt* para el caso de una prueba que falló

Finalmente, la carpeta de cobertura contiene información relevante sobre los resultados de cobertura e incluye archivos en formato HTML, como se ilustra en la figura 6.7, los cuales se pueden abrir en el navegador para obtener información detallada sobre las partes del DUT que no están siendo ejercitadas. Esto permite tomar medidas para aumentar la cobertura. Además, en la carpeta llamada *regression_ results* se encuentra un resumen de todos y cada uno de los errores presentados en la regresión, junto con un comando para abrir la herramienta Verdi y comenzar la simulación del error, como se ilustra en la figura 6.8.

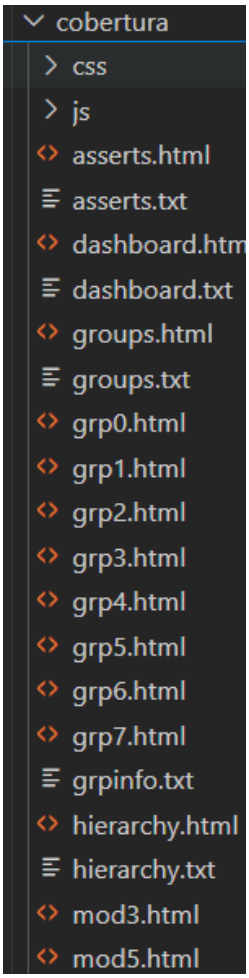


Figura 6.7: Contenido de la carpeta de cobertura.

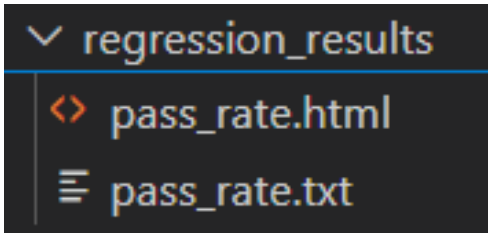


Figura 6.8: Contenido de la carpeta *regression_ results*

Capítulo 7

Detección de errores y análisis de cobertura

7.1. Solución de errores encontrados

En esta sección, se presenta una recopilación de los errores identificados a través del ambiente de pruebas, junto con un análisis detallado para determinar su origen y la solución propuesta al diseñador.

7.1.1. Primer error detectado

Durante la ejecución de las pruebas *write_burst_wrap_test*, *read_burst_wrap_test* y *write_read_burst_wrap_test*, que se encargan de verificar el correcto funcionamiento de las transacciones con ráfaga de tipo WRAP, se presentó un error en el cálculo de dichas transacciones. En particular, el subordinado no lograba calcular correctamente la dirección de escritura o lectura. Afortunadamente, el ambiente de verificación fue capaz de detectar este error. A continuación, se muestra la figura con los resultados de una de las simulaciones:

```
La direccion de escritura esperada en axi 30401e4 y la direccion de escritura en dut 24 no son equivalentes
La direccion de escritura esperada en axi 30401e8 y la direccion de escritura en dut 28 no son equivalentes
La direccion de escritura esperada en axi 30401ec y la direccion de escritura en dut 2c no son equivalentes
La direccion de escritura esperada en axi 30401f0 y la direccion de escritura en dut 30 no son equivalentes
La direccion de escritura esperada en axi 30401f4 y la direccion de escritura en dut 34 no son equivalentes
La direccion de escritura esperada en axi 30401f8 y la direccion de escritura en dut 38 no son equivalentes
La direccion de escritura esperada en axi 30401fc y la direccion de escritura en dut 3c no son equivalentes
```

Figura 7.1: Error de cálculo de dirección para transacciones con ráfaga WRAP.

En la imagen anterior, se puede observar que en el monitor del subordinado 3 se está realizando incorrectamente el cálculo de la dirección.

Se concluyó con certeza que el error se originó en el módulo del subordinado, porque este es el encargado de realizar dichos cálculos. Al revisar este módulo, se determinó que la

lógica para realizar dicho cálculo no estaba implementada correctamente. Por lo tanto, se sugirió al diseñador que replanteara esta lógica, utilizara el ambiente de simulación y verificara que el test no presentara errores después de realizar los cambios necesarios para solucionar el problema.

7.1.2. Segundo error detectado

El ambiente de pruebas es capaz de detectar si el DUT no logra finalizar todas las transacciones. Dado que los módulos maestros y subordinados están basados en máquinas de estados, existe la posibilidad de que alguno de estos módulos quede atrapado en un estado específico, lo cual podría provocar que todo el dispositivo se estanque y no pueda completar las transacciones. En la siguiente figura se muestra el mensaje de error correspondiente:

```
@ 6215000: uvm_test_top [uvm_test_top] El RTL no fue capaz de finalizar todas las transacciones
```

Figura 7.2: Error de estancamiento del DUT.

En este análisis se observa un problema de sincronización en el que varios maestros se quedan esperando la confirmación de sus transacciones debido a la falta de respuesta del subordinado. Tres de los maestros quedan detenidos en espera de una señal de confirmación para avanzar: en dos casos, se encuentran esperando la señal *ARREADY* del subordinado, la cual indica que la transacción de lectura puede continuar; en el tercer caso, el maestro espera la señal *AWREADY* para la transacción de escritura. Sin embargo, estas señales nunca se reciben, lo cual indica que el subordinado ha quedado atrapado en un estado y no es capaz de generar dichas confirmaciones.

Adicionalmente, se presenta una situación en la que dos maestros intentan ejecutar simultáneamente transacciones diferentes: uno realiza una transacción de lectura y otro una de escritura. En este escenario, el subordinado inicialmente decide procesar la transacción de escritura; sin embargo, el análisis combinacional del *Quality of Service* (QoS) en el DUT determina que debería priorizar la transacción de lectura. A pesar de esta prioridad, el subordinado ya se encuentra en un estado correspondiente al flujo de la transacción de escritura, del cual no puede salir, lo que impide que responda adecuadamente al maestro que espera la transacción de lectura.

Finalmente, al recibir datos de escritura, el subordinado entra en el estado *AWVALID*, en el cual espera el *handshake* de validación por parte del maestro. Sin embargo, debido a un cambio de transacción causado por un *glitch*, el maestro cambia de una transacción de escritura a una de lectura, generando un *handshake* de validación que corresponde a esta última. Mientras tanto, el subordinado sigue esperando un *handshake* de escritura, lo que bloquea la confirmación. En este caso, la transacción de escritura no se pierde, sino que se pausa para dar prioridad a la de lectura.

Como solución a este problema por parte del diseñador se propuso agregar en el subor-

dinado la capacidad de alternar entre los estados de lectura y escritura cuando ocurran estas situaciones, de modo que el DUT no se quede atascado en un solo estado.

7.1.3. Tercer error detectado

La dirección de lectura en las transacciones independientemente de su tipo de ráfaga, no se estaba calculando correctamente, resultados de la simulación se pueden apreciar en la figura 7.3:

```
Las direcciones de lectura en axi 1c74e34 y dut 19b107c no son equivalentes
Las direcciones de lectura en axi 1c74e34 y dut 19b107c no son equivalentes
Las direcciones de lectura en axi 1c74e34 y dut 19b107c no son equivalentes
Las direcciones de lectura en axi 1c74e34 y dut 19b107c no son equivalentes
Las direcciones de lectura en axi 1c74e34 y dut 19b107c no son equivalentes
Las direcciones de lectura en axi 1c74e34 y dut 19b107c no son equivalentes
Las direcciones de lectura en axi 1c74e34 y dut 19b107c no son equivalentes
Las direcciones de lectura en axi 1c74e34 y dut 19b107c no son equivalentes
Las direcciones de lectura en axi 1c74e34 y dut 19b107c no son equivalentes
Las direcciones de lectura en axi 1c74e34 y dut 19b107c no son equivalentes
Las direcciones de lectura en axi 1c74e34 y dut 19b107c no son equivalentes
Las direcciones de lectura en axi 1c74e34 y dut 19b107c no son equivalentes
Las direcciones de lectura en axi 1c74e34 y dut 19b107c no son equivalentes
Las direcciones de lectura en axi 1c74e34 y dut 19b107c no son equivalentes
Las direcciones de lectura en axi 1c74e34 y dut 19b107c no son equivalentes
Las direcciones de lectura en axi 168c42c y dut 1243dd8 no son equivalentes
Las direcciones de lectura en axi 168c42c y dut 1243dd8 no son equivalentes
Las direcciones de lectura en axi 168c42c y dut 1243dd8 no son equivalentes
Las direcciones de lectura en axi 168c42c y dut 1243dd8 no son equivalentes
Las direcciones de lectura en axi 168c42c y dut 1243dd8 no son equivalentes
```

Figura 7.3: Mensaje de error de cálculo incorrecto de direcciones.

En el análisis del problema, se observa que, durante el proceso de selección de la transacción de mayor prioridad, ocurre un *glitch* que lleva al subordinado a elegir erróneamente una transacción de escritura. Posteriormente, el subordinado ejecuta esta transacción equivocada, lo cual genera un desajuste (*mismatch*) entre la transacción percibida en el monitor y la transacción que realmente fue enviada.

Además, se observa que el subordinado realiza una transición de estados basada en la transacción incorrecta, lo cual contribuye al desajuste en la ejecución.

Para resolver este problema, se propone al diseñador reajustar los estados del subordinado, de manera que este sea capaz de identificar el momento adecuado para seleccionar la transacción correcta, evitando así la selección incorrecta de transacciones.

7.2. Cuarto error detectado

Cada vez que se ejecutaba una transacción con ráfaga INCR y un burst de 256, la cantidad de transferencias superaba el límite establecido de 256, lo que generaba un error. Esto

ocurre porque, en el protocolo AXI, el máximo de transferencias permitidas por INCR es de 256. En la figura 7.4, se puede apreciar el mensaje de error correspondiente.

```
s_mntr[3] [s_mntr[3]] [32325000] Se levanto el wlast sin que se haya transmitido la cantidad correcta de datos
m_mntr[1] [m_mntr[1]] [32325000] Se levanto el wlast sin que se haya transmitido la cantidad correcta de datos
s_mntr[3] [s_mntr[3]] [66895000] Se levanto el wlast sin que se haya transmitido la cantidad correcta de datos
m_mntr[3] [m_mntr[3]] [66895000] Se levanto el wlast sin que se haya transmitido la cantidad correcta de datos
```

Figura 7.4: Mensaje de error de error.

En este análisis se identifica que el DUT debía ejecutar una transacción compuesta por 256 transferencias (representadas como `FF` en hexadecimal). No obstante, el DUT experimentó un error y ejecutó un número mayor de transferencias, enviando datos no válidos. La causa del problema fue un desbordamiento en la variable del maestro que cuenta las transferencias en ejecución. Esta variable, destinada a almacenar la cantidad máxima de transferencias permitidas para la transacción, no contaba con suficientes bits para representar adecuadamente el valor, lo que provocó el desbordamiento.

La solución propuesta consiste en añadir un bit adicional a dicha variable, permitiendo así que almacene correctamente el límite de transferencias y evitando el desbordamiento.

7.3. Análisis de cobertura

Para este proyecto se analizarán las coberturas de tipo TOGGLE, FSM y funcional, descritas en la sección 3.1.8. Una vez ejecutada la regresión, se obtuvieron los resultados que se muestran en la figura 7.5.

Total Coverage Summary

SCORE	LINE	COND	TOGGLE	FSM	BRANCH	ASSERT	GROUP
81.68	85.42	70.02	85.87	64.44	84.92		99.38

Figura 7.5: Resultados de la primera regresión.

Considerando únicamente las coberturas GROUP (funcional), TOGGLE y FSM, se observa que se alcanzó más del 95 % en la cobertura funcional, un 85 % en TOGGLE y menos del 65 % en FSM. Por lo tanto, se puede concluir que la única cobertura que cumple con el objetivo del proyecto es la funcional.

Se realizó un análisis detallado de la cobertura FSM para determinar la causa del porcentaje inferior al 65 %, y se identificó que tanto en el maestro como en el subordinado había transiciones de estado que no se estaban alcanzando. Las transiciones de estados ejecutadas por los maestros y los subordinados se aprecian en las figuras 7.6 y 7.7 respectivamente.

transitions	Line No.	Covered
ARVALID_R->IDLE	153	Not Covered
ARVALID_R->RREADY_R	248	Covered
AWVALID_W->IDLE	153	Not Covered
AWVALID_W->WAIT_DATA	205	Covered
AW_OR_AR->ARVALID_R	199	Covered
AW_OR_AR->AWVALID_W	197	Covered
AW_OR_AR->IDLE	153	Not Covered
BREADY_W->BRESP_W	239	Covered
BREADY_W->IDLE	153	Not Covered
BRESP_W->IDLE	153	Covered
ERROR->IDLE	153	Not Covered
IDLE->AW_OR_AR	191	Covered
RLAST_R->IDLE	153	Covered
RREADY_R->IDLE	153	Not Covered
RREADY_R->RLAST_R	254	Covered
WAIT_DATA->IDLE	153	Not Covered
WAIT_DATA->WVALID_W	215	Covered
WLAST_W->BREADY_W	233	Covered
WLAST_W->IDLE	153	Not Covered
WLAST_W->WAIT_DATA	235	Covered
WREADY_W->IDLE	153	Not Covered
WREADY_W->WLAST_W	225	Covered
WVALID_W->IDLE	153	Not Covered
WVALID_W->WREADY_W	221	Covered

Figura 7.6: Transiciones cubiertas y no cubiertas del maestro.

transitions	Line No.	Covered
ARVALID_R->AWVALID_W	313	Covered
ARVALID_R->IDLE	134	Not Covered
ARVALID_R->RREADY_R	308	Covered
AWVALID_W->ARVALID_R	289	Covered
AWVALID_W->IDLE	134	Not Covered
AWVALID_W->WREADY_W	284	Covered
AW_OR_AR->ARVALID_R	278	Covered
AW_OR_AR->AWVALID_W	276	Covered
AW_OR_AR->IDLE	134	Covered
IDLE->AW_OR_AR	272	Covered
RLAST_R->IDLE	134	Covered
RLAST_R->RREADY_R	338	Covered
RREADY_R->IDLE	134	Not Covered
RREADY_R->WAIT_DATA	330	Covered
RVALID_R->IDLE	134	Not Covered
RVALID_R->RLAST_R	326	Covered
WAIT_DATA->IDLE	134	Not Covered
WAIT_DATA->RVALID_R	319	Covered
WLAST_W->IDLE	134	Covered
WREADY_W->IDLE	134	Not Covered
WREADY_W->WLAST_W	295	Covered

Figura 7.7: Transiciones cubiertas y no cubiertas del subordinado.

Para aumentar el porcentaje de cobertura FSM, se diseñó la prueba *reset_ test*, que permite al DUT ejecutar las transiciones de estado que no fueron cubiertas durante la regresión completa. Esta prueba, centrada en el reset, se diseñó debido a que, como se observa en las figuras 7.6 y 7.7, las transiciones no cubiertas corresponden a aquellas hacia el estado IDLE. Esto indica que el DUT no experimenta un estímulo que le permita retornar al estado inicial en medio de la ejecución de una transacción, y ese estímulo es precisamente el reset.

Una vez diseñado el test de reset y ejecutada nuevamente la regresión, se obtuvieron los resultados de la cobertura FSM que se muestran en la figura 7.8. Como se puede observar, gracias a esta prueba, la cobertura FSM alcanzó más del 95 %.

Total Coverage Summary

SCORE	LINE	COND	TOGGLE	FSM	BRANCH	ASSERT	GROUP
87.90	86.63	71.29	85.87	97.73	86.18		99.69

Figura 7.8: Resultados de la nueva regresión.

A continuación, los resultados de la cobertura toggle se muestran en las figuras 7.9 y 7.10. Estos resultados indican que, principalmente, las señales encargadas de manejar las direcciones de memoria y los ID no están cumpliendo con los objetivos de cobertura. Esto se debe a que, en el caso de los ID, se están utilizando solo 4 maestros y 4 subordinados; por lo tanto, para asignar un ID a un maestro se requieren únicamente 2 bits. Sin embargo, internamente esta variable es de 4 bits, lo que implica que dos de sus bits nunca serán utilizados.

En el caso de las direcciones, el DUT no cuenta con la capacidad de detectar direcciones que se encuentren completamente fuera de los rangos de memoria. Por lo tanto, este escenario no fue considerado al desarrollar las pruebas. Como resultado, de los 32 bits definidos para las direcciones de memoria, solo se están utilizando 26 bits, lo que implica que habrá 6 bits que nunca serán utilizados. Cabe destacar que la variable *full* jamás fue usada.

Name	Toggle	Toggle 1->0	Toggle 0->1	Direction
dout[31:26]	No	No	No	INPUT
dout[72:69]	No	No	No	INPUT
full	No	No	No	INPUT
AWADDR[31:26]	No	No	No	OUTPUT
AWID[3:0]	No	No	No	OUTPUT
BID[3:0]	No	No	No	INPUT
ARADDR[31:26]	No	No	No	OUTPUT
ARID[3:0]	No	No	No	OUTPUT
RID[3:0]	No	No	No	INPUT

Figura 7.9: Señales que no fueron cubiertas en toggle del maestro.

full	No	No	No	INPUT
din[63:57]	No	No	No	OUTPUT
din[95:89]	No	No	No	OUTPUT
AWADDR[31:24]	No	No	No	INPUT
AWID[3:2]	No	No	No	INPUT
BID[3:2]	No	No	No	OUTPUT
ARADDR[31:24]	No	No	No	INPUT
ARID[3:2]	No	No	No	INPUT
RID[3:2]	No	No	No	OUTPUT

Figura 7.10: Señales que no fueron cubiertas en toggle del subordinado.

Las variables denominadas *dout* y *din* también almacenan direcciones de memoria de 32 bits, de las cuales solo se utilizan 26 bits. Por lo tanto, estas variables tampoco cumplen con los requisitos de cobertura TOGGLE. Estas variables se utilizan para comunicar al DUT con los periféricos simulados, que corresponden a los monitores del maestro y el subordinado.

En conclusión, el incremento de cobertura para las variables de ID no es posible, ya que solo existen 4 maestros y, por lo tanto, los bits adicionales no pueden ser activados. Respecto a las direcciones fuera de rango, no se implementará una prueba debido a que esta funcionalidad no está soportada en el DUT, por lo que no resulta relevante probarla. Para optimizar la cobertura, dichas señales se excluirán del análisis, permitiendo evaluar el porcentaje de cobertura *toggle* sin considerar estas señales. En la figura 7.11 se observa el porcentaje de cobertura, reflejando una visión más precisa del desempeño general.

Total Coverage Summary

SCORE	LINE	COND	TOGGLE	FSM	BRANCH	ASSERT	GROUP
89.50	86.63	71.29	95.46	97.73	86.18		99.69

Figura 7.11: Porcentajes de cobertura luego de hacer exclusiones para TOGGLE.

Capítulo 8

Conclusiones

La cobertura funcional y la cobertura de la máquina de estados finita (FSM) alcanzaron el objetivo del proyecto, superando el 95 %. Inicialmente, la cobertura *toggle* no lograba alcanzar dicho porcentaje, debido a ciertas decisiones de diseño del dispositivo. No obstante, se logró incrementar la cobertura *toggle* a más del 95 % tras realizar algunas exclusiones específicas. Como siguiente paso en el proyecto, se recomienda presentar estos resultados de cobertura al equipo de diseño para que evalúen posibles ajustes adicionales en el DUT que permitan cumplir completamente con los objetivos de cobertura sin depender de exclusiones.

El ambiente logró detectar 4 errores críticos en el funcionamiento del dispositivo, los cuales provocaban un comportamiento totalmente incorrecto, como se detalla en la sección 7.1. Esto permitió llevar a cabo un análisis de las simulaciones para identificar la causa principal de los problemas y proponer soluciones. Si se hubieran realizado simulaciones manuales, el tiempo requerido para detectar y solucionar los errores habría sido considerablemente mayor. Es importante destacar que la frecuencia de éxito de las pruebas aumentó del 30 % a más del 90 % en la ejecución de las regresiones, tras implementar los cambios necesarios para corregir los errores detectados. El 10 % restante corresponde a errores que el ambiente aún puede identificar; sin embargo, estos no representan una amenaza para el correcto funcionamiento del DUT, ya que no se pierden datos y el DUT ejecuta todas las transacciones, siendo simplemente glitches ocasionados por el procesamiento de la información interna en el DUT. No obstante, es preferible que el funcionamiento del dispositivo sea lo más limpio posible. Por lo tanto, como próximos pasos de mejora para el proyecto, se recomienda dialogar con el equipo de diseño, presentar los errores actuales detectados en las regresiones y proponer soluciones.

El sistema de regresiones es una herramienta muy útil para hacer seguimiento de los resultados de todas las pruebas, facilitando la detección de errores, el establecimiento de un checklist y la eliminación sistemática de los errores que aparecen en la regresión. Un paso siguiente para mejorar este sistema es desarrollar una página web que almacene semanalmente los resultados de las regresiones y ejecute las regresiones de forma periódica. Así, los ingenieros verificadores podrán consultar la página web para revisar los errores

existentes, simularlos, visualizar la curva de *pass rate* (cuyo objetivo es aumentar conforme se resuelvan los errores) y verificar que, una vez realizados los cambios, dichos errores no aparezcan en las siguientes regresiones. De esta forma, se facilita un seguimiento del progreso del proceso de verificación y se asegura que los errores registrados no se pasen por alto, ya que estarán guardados en una base de datos.

Bibliografía

[1] T. Whitney y G. Neville-Neil, «SoC: Software, Hardware, Nightmare, Bliss,» *Queue*, vol. 1, págs. 24-28, 2003. dirección: https://api.semanticscholar.org/CorpusID:21826064.

[2] S. Li, K. T. Lim, P. Faraboschi, J. Chang, P. Ranganathan y N. P. Jouppi, «System-level integrated server architectures for scale-out datacenters,» *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, págs. 260-271, 2011. dirección: https://api.semanticscholar.org/CorpusID:11763671.

[3] D. D. Buss, «Technology in the Internet age,» *2002 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No.02CH37315)*, vol. 1, 18-21 vol.1, 2002. dirección: https://api.semanticscholar.org/CorpusID:24869743.

[4] M. Mehendale, «SoC - The Road Ahead,» en *VLSI design (Print)*, 2006. dirección: https://api.semanticscholar.org/CorpusID:206842601.

[5] K. M. y D. R., «INTEGRATED CIRCUITS AND THEIR APPLICATIONS IN ELECTRONICS,» *American Journal of Applied Science and Technology*, 2024. dirección: https://api.semanticscholar.org/CorpusID:269569711.

[6] S. I. Shah, «Interconnects for next generation SoC designs,» *Proceedings of the 2nd International Workshop on Advanced Interconnect Solutions and Technologies for Emerging Computing Systems*, 2017. dirección: https://api.semanticscholar.org/CorpusID:21559544.

[7] M. Stojcÿev, «An Overview of SoC Buses,» 2007. dirección: https://api.semanticscholar.org/CorpusID:59888146.

[8] M. Mitic y M. K. Stojcev, «An Overview of On-Chip Buses,» *Facta universitatis. Series electronics and energetics*, vol. 19, págs. 405-428, 2006. dirección: https://api.semanticscholar.org/CorpusID:4833479.

[9] A. Lines, «Nexus: an asynchronous crossbar interconnect for synchronous system-on-chip designs,» *11th Symposium on High Performance Interconnects, 2003. Proceedings.*, págs. 2-9, 2003. dirección: https://api.semanticscholar.org/CorpusID:1799204.

[10] ARM, *Learn the architecture - An introduction to AMBA AXI*, Company Documentation, 2020-2022.

[11] B. Wile, J. C. Goss y W. Roesner, *Comprehensive Functional Verification: The Complete Industry Cycle*, 1st. Morgan Kaufmann, 2005.

[12] R. Salemi, *The UVM Primer: An Introduction to the Universal Verification Methodology*. Editorial (si está disponible), 2014.

[13] S. Qamar, W. H. Butt, M. W. Anwar, F. Azam y M. Q. Khan, «A Comprehensive Investigation of Universal Verification Methodology (UVM) Standard for Design Verification,» *Proceedings of the 2020 9th International Conference on Software and Computer Applications*, 2020. dirección: https://api.semanticscholar.org/CorpusID:219132594.

[14] A. S. Initiative, *Universal Verification Methodology (UVM) 1.1 User's Guide*. Accellera Systems Initiative, 2011, https://accellera.org/downloads/standards/uvm.

[15] Vijayan, V. Raja y A. Kumar, «Development of Basic Template Environment for Functional Verification of VLSI Design Using UVM,» 2013. dirección: https://api.semanticscholar.org/CorpusID:111086756.

[16] S. D. I. Software, *Universal Verification Methodology (UVM) Cookbook*, Siemens, 2020. dirección: https://verificationacademy.com/cookbook/uvm-universal-verification-methodology/.

[17] N. Campos, H. A. Monteiro, A. V. D. Brito, A. M. N. Lima, E. U. K. Melcher y M. R. A. Morais, «A framework for design and validation of face detection systems,» *2017 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON)*, págs. 1-7, 2017. dirección: https://api.semanticscholar.org/CorpusID:13336132.

[18] A. Fiergolski, «Simulation environment based on the Universal Verification Methodology,» *Journal of Instrumentation*, vol. 12, págs. C01001-C01001, 2017. dirección: https://api.semanticscholar.org/CorpusID:126137237.

[19] C. Spear y G. Tumbas, *SystemVerilog for Verification: A Guide to Learning the Testbench Language Features*, Third. New York, NY: Springer, 2021.

Apéndice A

Plan de pruebas

A.1. Estudio del dispositivo bajo pruebas

A.1.1. Uso esperado

El DUT (Device under test) debe ser capaz de establecer comunicación entre n maestros y n subordinados, es decir, que todos los maestros sean capaces de comunicarse con cualquier subordinado y viceversa, además, cada maestro y subordinado tendrá una interfaz para conectar dos FIFO's, una FIFO se encarga de la transmisión de datos y la otra se encargará de la recepción de los mismos, en el caso del maestro, la primera FIFO enviará intrucciones de escritura/lectura y posteriormente los datos que serán escritos si la instrucción es de escritura, en el caso del subordinado la primera FIFO se encargará de recibir los datos y la segunda FIFO los transmitirá en el caso de solicitudes de lectura por parte de algún maestro. Se espera que el DUT funcione en un ambiente de constante comunicación mediante lecturas y escrituras desde varios maestros a varios subordinados de forma paralela y aleatoria, respetando las transacciones con mayor prioridad, permitiendo un mejor uso de ancho de banda.

A.1.2. Lista de características y pruebas

El módulo de interconexión basado en el protocolo AXI debe ser capaz de cumplir con todas las funcionalidades descritas en cada prueba (test), la descripción detallada de estas pruebas se encuentra en la sección A.2. A continuación, se enumeran las características, restricciones y criterio de cumplimiento para cada prueba.

write_unique_test

Características:

- Escrituras únicas.
- Direcciones por subordinado con rangos de 16MB.
- Escrituras alineadas en 4 bytes.
- Identificador único para cada maestro (ID).
- Quality of service (qos) para cada transacción aleatorio de 1 a 16.
- Respuesta de escrituras de OKAY.

Restricciones(constraints):

- Transacción tipo WRITE.
- Ráfaga (burst) aleatorio (FIXED, INCR, WRAP).
- Tamaño (size) aleatorio de 1 a 4 bytes.
- Longitud (len) de 1.
- Estrobo (strobe) dependiente del size.
- Quality of service (qos) aleatorio.
- Dirección de escritura aleatoria dentro del rango de 16MB.
- Paquete aleatorio.
- ID correspondiente al maestro que realiza la transacción.

Criterio de cumplimiento:

- Cumplimiento del protocolo AXI en la escritura de datos con las restricciones establecidas anteriormente.
- Sólo se puede transmitir un dato por transacción.
- Las direcciones de escritura deben estar dentro del rango de memoria del subordinado que recibió la transacción.
- Las direcciones calculadas por el subordinado de las transacciones tienen que ser equivalentes a las direcciones calculadas por el ambiente.
- Los datos escritos tienen que ser equivalentes a los calculados por el subordinado.
- El ID de la transacción recibida por el subordinado debe ser equivalente al ID del maestro que la realizó.
- Los datos recibidos tienen que ser equivalentes a los que fueron enviados.
- Los datos transmitidos a través de los adaptadores(sockets) deben ser equivalentes.

write_burst_fixed_test

Características:

- Escrituras en ráfaga (burst) tipo FIXED.
- Escritura de longitud (len) de 1 a 16 para ráfagas (burst) tipo FIXED.
- Direcciones por subordinado con rangos de 16MB.
- Escrituras alineadas en 4 bytes.
- Identificador único para cada maestro (ID).
- Quality of service (qos) para cada transacción aleatorio de 1 a 16.
- Respuestas de escritura OKAY.

Restricciones(constraints):

- Transacción de escritura.
- Ráfaga (burst) tipo FIXED.
- Tamaño (size) aleatorio.
- Longitud (len) aleatorio de 1 a 16.
- Estrobo (strobe) dependiente del size.
- Quality of service (qos) aleatorio.
- Dirección de escritura aleatoria dentro del rango de 16MB.
- Paquete aleatorio.
- ID correspondiente al maestro que realiza la transacción.

Criterio de cumplimiento:

- Cumplimiento del protocolo AXI en la escritura de datos con las restricciones establecidas anteriormente.
- Transmisión de varios datos por transacción en el rango de 1 a 16.
- La dirección de memoria debe ser estática, es decir, la misma para todas las escrituras.
- Cantidad de datos transmitidos equivalentes a la longitud (len) de la transacción.

- Las direcciones de escritura deben estar dentro del rango de memoria del subordinado que recibió la transacción.
- Las direcciones calculadas por el subordinado de las transacciones tienen que ser equivalentes a las direcciones calculadas por el ambiente.
- Los datos escritos tienen que ser equivalentes a los calculados por el subordinado.
- El ID de la transacción recibida por el subordinado debe ser equivalente al ID del maestro que la realizó.
- Los datos recibidos tienen que ser equivalentes a los que fueron enviados.
- Los datos transmitidos a través de los adaptadores(sockets) deben ser equivalentes.

write_burst_incr_test

Características:

- Escrituras en ráfaga (burst) tipo INCR.
- Escritura de longitud (len) de 1 a 256 para ráfagas (burst) tipo incremental INCR.
- Direcciones por subordinado con rangos de 16MB.
- Escrituras alineadas en 4 bytes.
- Identificador único para cada maestro (ID).
- Quality of service (qos) para cada transacción aleatorio de 1 a 16.
- Respuestas de escritura OKAY.

Restricciones(constraints):

- Transacción de escritura.
- Ráfaga (burst) tipo INCR.
- Tamaño (size) aleatorio de 1 a 4 bytes.
- Longitud (len) aleatorio de 1 a 256.
- Estrobo (strobe) dependiente del size.
- Quality of service (qos) aleatorio.
- Dirección de escritura aleatoria dentro del rango de 16MB.
- Paquete aleatorio.

- ID correspondiente al maestro que realiza la transacción.

Criterio de cumplimiento:

- Cumplimiento del protocolo AXI en la escritura de datos con las restricciones establecidas anteriormente.
- Transmisión de varios datos por transacción en el rango de 1 a 256 transferencias.
- Las direcciones de memoria calculadas deben ser equivalentes a las direcciones de memoria esperada de forma incremental.
- Cantidad de datos transmitidos equivalentes a la longitud (len) de la transacción.
- Las direcciones de escritura deben estar dentro del rango de memoria del subordinado que recibió la transacción.
- Las direcciones calculadas por el subordinado de las transacciones tienen que ser equivalentes a las direcciones calculadas por el ambiente.
- Los datos escritos tienen que ser equivalentes a los calculados por el subordinado.
- El ID de la transacción recibida por el subordinado debe ser equivalente al ID del maestro que la realizó.
- Los datos recibidos tienen que ser equivalentes a los que fueron enviados.
- Los datos transmitidos a través de los adaptadores(sockets) deben ser equivalentes.

write_burst_wrap_test

Características:

- Escrituras en ráfaga (burst) tipo WRAP.
- Escritura de longitud (len) de 2, 4, 8, o 16 para ráfagas (burst) tipo WRAP.
- Direcciones por subordinado con rangos de 16MB.
- Escrituras alineadas en 4 bytes.
- Identificador único para cada maestro (ID).
- Quality of service (qos) para cada transacción aleatorio de 1 a 16.
- Respuestas de escritura OKAY.

Restricciones(constraints):

- Transacción de escritura.
- Ráfaga (burst) tipo WRAP.
- Tamaño (size) aleatorio de 1 a 4 bytes.
- Longitud (len) aleatoria 2, 4, 8, o 16.
- Estrobo (strobe) dependiente del size.
- Quality of service (qos) aleatorio.
- Dirección de escritura aleatoria dentro del rango de 16MB.
- Paquete aleatorio.
- ID correspondiente al maestro que realiza la transacción.

Criterio de cumplimiento:

- Cumplimiento del protocolo AXI en la escritura de datos con las restricciones establecidas anteriormente.
- Transmisión de 2, 4, 8, o 16 datos por transacción.
- Las direcciones de memoria calculadas deben ser equivalentes a las direcciones de memoria esperadas en ráfaga tipo WRAP.
- Cuando las direcciones de memoria alcanzan el límite del rango, el siguiente cálculo debe ser la dirección inicial de la transacción, los resultados serán comparados con los resultados esperados.
- Los datos escritos tienen que ser equivalentes a los calculados por el subordinado.
- Cantidad de datos transmitidos equivalentes a la longitud (len) de la transacción.
- Las direcciones de escritura deben estar dentro del rango de memoria del subordinado que recibió la transacción.
- El ID de la transacción recibida por el subordinado debe ser equivalente al ID del maestro que la realizó.
- Los datos recibidos tienen que ser equivalentes a los que fueron enviados.
- Los datos transmitidos a través de los adaptadores(sockets) deben ser equivalentes.

write_strobe_test

Características:

- Escrituras en ráfaga de cualquier tipo (FIXED, INCR, WRAP) con bytes deshabilitados, es decir, el strobe indica cuáles bytes se deshabilitan.
- Direcciones por subordinado con rangos de 16MB.
- Escrituras alineadas en 4 bytes.
- Identificador único para cada maestro (ID).
- Quality of service (qos) para cada transacción aleatorio de 1 a 16.
- Respuestas de escritura OKAY.

Restricciones(constraints):

- Transacción de escritura.
- Ráfaga (burst) aleatoria.
- Tamaño (size) aleatorio de 1 a 4 bytes.
- Longitud (len) aleatoria de 1 a 256 para INCR, de 1 a 16 para FIXED y 2, 4, 8 o 16 para WRAP.
- Estrobo (strobe) dependiente del size.
- Quality of service (qos) aleatorio.
- Dirección de escritura aleatoria dentro del rango de 16MB.
- Paquete aleatorio.
- ID correspondiente al maestro que realiza la transacción.

Criterio de cumplimiento:

- Cumplimiento del protocolo AXI en la escritura de datos con las restricciones establecidas anteriormente.
- El dato con el strobe aplicado debe ser equivalente al dato esperado.
- Cantidad de datos transmitidos equivalentes a la longitud (len) de la transacción.
- Las direcciones de escritura deben estar dentro del rango de memoria del subordinado que recibió la transacción.

- Las direcciones calculadas por el subordinado de las transacciones tienen que ser equivalentes a las direcciones calculadas por el ambiente.
- Los datos escritos tienen que ser equivalentes a los calculados por el subordinado.
- El ID de la transacción recibida por el subordinado debe ser equivalente al ID del maestro que la realizó.
- Los datos recibidos tienen que ser equivalentes a los que fueron enviados.
- Los datos transmitidos a través de los adaptadores(sockets) deben ser equivalentes.

write_qos_test

Características:

- Quality of service (qos) para cada transacción aleatorio de 1 a 16.
- Direcciones por subordinado con rangos de 16MB.
- Escrituras alineadas en 4 bytes.
- Identificador único para cada maestro (ID).
- Respuestas de escritura OKAY.

Restricciones(constraints):

- Transacción de escritura.
- Ráfaga (burst) aleatoria.
- Tamaño (size) aleatorio de 1 a 4 bytes.
- Longitud (len) aleatoria de 1 a 256 para INCR, de 1 a 16 para FIXED y 2, 4, 8 o 16 para WRAP.
- Estrobo (strobe) dependiente del size.
- Quality of service (qos) aleatorio, donde solo uno de los maestros tendrá el mayor qos, los otros iguales.
- Dirección de escritura desde todos los maestros a un sólo subordinado dentro del rango de 16MB en memoria, esto se repite para cada subordinado.
- Paquete aleatorio.
- ID correspondiente al maestro que realiza la transacción.

Criterio de cumplimiento:

- Cumplimiento del protocolo AXI en la escritura de datos con las restricciones establecidas anteriormente.
- Ejecución por prioridad de las transacciones, la transacción tomada primero siempre debe ser la de mayor quality of service.
- Cantidad de datos transmitidos equivalentes a la longitud (len) de la transacción.
- Las direcciones de escritura deben estar dentro del rango de memoria del subordinado que recibió la transacción.
- Las direcciones calculadas por el subordinado de las transacciones tienen que ser equivalentes a las direcciones calculadas por el ambiente.
- Los datos escritos tienen que ser equivalentes a los calculados por el subordinado.
- El ID de la transacción recibida por el subordinado debe ser equivalente al ID del maestro que la realizó.
- Los datos recibidos tienen que ser equivalentes a los que fueron enviados.
- Los datos transmitidos a través de los adaptadores(sockets) deben ser equivalentes.

read_unique_test

Características:

- Lecturas únicas.
- Direcciones por subordinado con rangos de 16MB.
- Lecturas alineadas en 4 bytes.
- Identificador único para cada maestro (ID).
- Quality of service (qos) para cada transacción aleatorio de 1 a 16.
- Respuesta de lecturas de OKAY.

Restricciones(constraints):

- Transacción de lectura.
- Ráfaga (burst) aleatoria (FIXED, INCR, WRAP).
- Tamaño (size) aleatorio de 1 a 4 bytes.

- Longitud (len) de 1 transferencia.
- Quality of service (qos) aleatorio.
- Dirección de lectura aleatoria dentro del rango de 16MB.
- ID correspondiente al maestro que realiza la transacción.

Criterio de cumplimiento:

- Cumplimiento del protocolo AXI en la lectura de datos con las restricciones establecidas anteriormente.
- Sólo se puede leer un dato por transacción.
- Las direcciones de lectura deben estar dentro del rango de memoria del subordinado que recibió la transacción.
- El ID de la transacción recibida por el subordinado debe ser equivalente al ID del maestro que la realizó y el ID de los datos leídos debe ser equivalente al ID del maestro que está recibiendo los datos.
- Las direcciones calculadas por el subordinado de las transacciones tienen que ser equivalentes a las direcciones calculadas por el ambiente.
- Los datos recibidos en el maestro tienen que ser equivalentes a los que fueron enviados desde el subordinado.
- Los datos transmitidos a través de los adaptadores(sockets) deben ser equivalentes.

read_burst_fixed_test

Características:

- Lecturas en ráfaga (burst) tipo FIXED.
- Lectura de longitud (len) de 1 a 16 para ráfagas (burst) tipo FIXED.
- Direcciones por subordinado con rangos de 16MB.
- Lecturas alineadas en 4 bytes.
- Identificador único para cada maestro (ID).
- Quality of service (qos) para cada transacción aleatorio de 1 a 16.
- Respuestas de lectura OKAY.

Restricciones(constraints):

- Transacción de lectura.
- Ráfaga (burst) tipo FIXED.
- Tamaño (size) aleatorio de 1 a 4 bytes.
- Longitud (len) aleatoria de 1 a 16.
- Quality of service (qos) aleatorio.
- Dirección de lectura aleatoria dentro del rango de 16MB.
- ID correspondiente al maestro que realiza la transacción.

Criterio de cumplimiento:

- Cumplimiento del protocolo AXI en la escritura de datos con las restricciones establecidas anteriormente.
- Lectura de varios datos por transacción en el rango de 1 a 16
- La dirección de memoria debe ser estática.
- Las direcciones calculadas por el ambiente todas deben ser estáticas.
- Cantidad de datos leídos equivalentes a la longitud (len) de la transacción.
- Las direcciones de lectura deben estar dentro del rango de memoria del subordinado que recibió la transacción.
- Las direcciones calculadas por el subordinado de las transacciones tienen que ser equivalentes a las direcciones calculadas por el ambiente.
- El ID de la transacción recibida por el subordinado debe ser equivalente al ID del maestro que la realizó y el ID de los datos leídos debe ser equivalente al ID del maestro que está recibiendo los datos.
- Los datos recibidos en el maestro tienen que ser equivalentes a los que fueron enviados desde el subordinado.
- Los datos transmitidos a través de los adaptadores(sockets) deben ser equivalentes.

read_burst_incr_test

Características:

- Lecturas en ráfaga (burst) tipo INCR.
- Lectura de longitud (len) de 1 a 256 para ráfagas (burst) tipo incremental INCR.

- Direcciones por subordinado con rangos de 16MB.
- Lecturas alineadas en 4 bytes.
- Identificador único para cada maestro (ID).
- Quality of service (qos) para cada transacción aleatorio de 1 a 16.
- Respuestas de lectura OKAY.

Restricciones(constraints):

- Transacción de lectura.
- Ráfaga (burst) tipo INCR.
- Tamaño (size) aleatorio de 1 a 4 bytes.
- Longitud (len) aleatoria de 1 a 256.
- Quality of service (qos) aleatorio.
- Dirección de lectura aleatoria dentro del rango de 16MB.
- ID correspondiente al maestro que realiza la transacción.

Criterio de cumplimiento:

- Cumplimiento del protocolo AXI en la escritura de datos con las restricciones establecidas anteriormente.
- Lectura de varios datos por transacción en el rango de 1 a 256 transferencias en ráfaga (burst) tipo INCR.
- Las direcciones de memoria calculadas deben ser equivalentes a las direcciones de memoria esperada de forma incremental.
- Cantidad de datos leídos equivalentes a la longitud (len) de la transacción.
- Las direcciones de lectura deben estar dentro del rango de memoria del subordinado que recibió la transacción.
- Las direcciones calculadas por el subordinado de las transacciones tienen que ser equivalentes a las direcciones calculadas por el ambiente.
- El ID de la transacción recibida por el subordinado debe ser equivalente al ID del maestro que la realizó y el ID de los datos leídos debe ser equivalente al ID del maestro que está recibiendo los datos.
- Los datos recibidos en el maestro tienen que ser equivalentes a los que fueron enviados desde el subordinado.
- Los datos transmitidos a través de los adaptadores(sockets) deben ser equivalentes.

read_burst_wrap_test

Características:

- Lecturas en ráfaga (burst) tipo WRAP.
- Lectura de longitud (len) de 2, 4, 8, o 16 para ráfagas (burst) tipo WRAP.
- Direcciones por subordinado con rangos de 16MB.
- Lecturas alineadas en 4 bytes.
- Identificador único para cada maestro (ID).
- Quality of service (qos) para cada transacción aleatorio de 1 a 16.
- Respuestas de escritura OKAY.

Restricciones(constraints):

- Transacción de lectura.
- Ráfaga (burst) tipo WRAP.
- Tamaño (size) aleatorio de 1 a 4 bytes.
- Longitud (len) aleatoria 2, 4, 8, o 16.
- Quality of service (qos) aleatorio.
- Dirección de lectura aleatoria dentro del rango de 16MB.
- ID correspondiente al maestro que realiza la transacción.

Criterio de cumplimiento:

- Cumplimiento del protocolo AXI en la escritura de datos con las restricciones establecidas anteriormente.
- Lectura de 2, 4, 8, o 16 datos por transacción.
- Las direcciones de memoria calculadas deben ser equivalentes a las direcciones de memoria esperadas en ráfaga tipo WRAP.
- Cuando las direcciones de memoria alcanzan el límite del rango, el siguiente cálculo debe ser la dirección inicial de la transacción, los resultados serán comparados con los resultados esperados.
- Cantidad de datos leídos equivalentes a la longitud (len) de la transacción.

- Las direcciones de lectura deben estar dentro del rango de memoria del subordinado que recibió la transacción.
- Las direcciones calculadas por el subordinado de las transacciones tienen que ser equivalentes a las direcciones calculadas por el ambiente.
- El ID de la transacción recibida por el subordinado debe ser equivalente al ID del maestro que la realizó y el ID de los datos leídos debe ser equivalente al ID del maestro que está recibiendo los datos.
- Los datos recibidos en el maestro tienen que ser equivalentes a los que fueron enviados desde el subordinado.
- Los datos transmitidos a través de los adaptadores(sockets) deben ser equivalentes.

read_size_test

Características:

- Size para transacciones únicas o en ráfaga de cualquier tipo (FIXED, INCR, WRAP).
- Direcciones por subordinado con rangos de 16MB.
- Lecturas alineadas en 4 bytes.
- Identificador único para cada maestro (ID).
- Quality of service (qos) para cada transacción aleatorio de 1 a 16.
- Respuestas de lectura OKAY.

Restricciones(constraints):

- Transacción de lectura.
- Ráfaga (burst) aleatoria.
- Tamaño (size) aleatorio de 1, 2 o 4 bytes.
- Longitud (len) aleatoria de 1 a 256 para ráfaga INCR de 1 a 16 para FIXED y 2, 4, 8, o 16 para WRAP.
- Quality of service (qos) aleatorio.
- Dirección de lectura aleatoria dentro del rango de 16MB.
- Paquete aleatorio.
- ID correspondiente al maestro que realiza la transacción.

Criterio de cumplimiento:

- Cumplimiento del protocolo AXI en la escritura de datos con las restricciones establecidas anteriormente.
- Cálculo correcto de las direcciones de memoria tomando en cuenta el size, el resultado debe ser equivalente al resultado esperado.
- Cantidad de datos leídos equivalentes a la longitud (len) de la transacción.
- Las direcciones de lectura deben estar dentro del rango de memoria del subordinado que recibió la transacción.
- Las direcciones calculadas por el subordinado de las transacciones tienen que ser equivalentes a las direcciones calculadas por el ambiente.
- El ID de la transacción recibida por el subordinado debe ser equivalente al ID del maestro que la realizó y el ID de los datos leídos debe ser equivalente al ID del maestro que está recibiendo los datos.
- Los datos recibidos en el maestro tienen que ser equivalentes a los que fueron enviados desde el subordinado.
- Los datos transmitidos a través de los adaptadores(sockets) deben ser equivalentes.

read_qos_test

Características:

- Quality of service (qos) para cada transacción aleatorio de 1 a 16.
- Direcciones por subordinado con rangos de 16MB.
- Lecturas alineadas en 4 bytes.
- Identificador único para cada maestro (ID).
- Respuestas de lectura OKAY.

Restricciones(constraints):

- Transacción de lectura.
- Ráfaga (burst) aleatoria (FIXED, INCR, WRAP).
- Tamaño (size) aleatorio de 1 a 4 bytes.
- Longitud (len) aleatoria de 1 a 256 para ráfaga INCR y de 1 a 16 para FIXED y de 2, 4, 8 o 16 para WRAP.

- Quality of service (qos) aleatorio.
- Dirección de lectura desde todos los maestros a un sólo subordinado dentro del rango de 16MB en memoria, cada subordinado se escoge de forma aleatoria.
- ID correspondiente al maestro que realiza la transacción.

Criterio de cumplimiento:

- Cumplimiento del protocolo AXI en la escritura de datos con las restricciones establecidas anteriormente.
- Ejecución por prioridad de las transacciones, siempre se debe tomar primero la transacción con mayor qos, de lo contrario el ambiente detectará un error.
- Cantidad de datos leídos equivalentes a la longitud (len) de la transacción.
- Las direcciones de lectura deben estar dentro del rango de memoria del subordinado que recibió la transacción.
- Las direcciones calculadas por el subordinado de las transacciones tienen que ser equivalentes a las direcciones calculadas por el ambiente.
- El ID de la transacción recibida por el subordinado debe ser equivalente al ID del maestro que la realizó y el ID de los datos leídos debe ser equivalente al ID del maestro que está recibiendo los datos.
- Los datos recibidos en el maestro tienen que ser equivalentes a los que fueron enviados desde el subordinado.
- Los datos transmitidos a través de los adaptadores(sockets) deben ser equivalentes.

write_read_test

Características:

- Ejecución de transacciones de escritura o lectura desde varios maestros al mismo tiempo.
- Direcciones por subordinado con rangos de 16MB.
- Transacciones alineadas en 4 bytes.
- Identificador único para cada maestro (ID).
- Respuestas de OKAY.

Restricciones(constraints):

- Transacción aleatoria.
- Ráfaga (burst) aleatoria.
- Tamaño (size) aleatorio.
- Longitud (len) aleatoria de 1 a 256 para ráfaga INCR y de 1 a 16 para FIXED y de 2, 4, 8 o 16 para WRAP.
- Estrobo (strobe) dependiente del size.
- Quality of service (qos) aleatorio.
- Paquete aleatorio.
- Dirección de transacción de escritura o lectura dentro del rango de 16MB en memoria.
- ID correspondiente al maestro que realiza la transacción.

Criterio de cumplimiento:

- Cumplimiento del protocolo AXI en la escritura y lectura de datos con las restricciones establecidas anteriormente.
- Todas las transacciones de escritura o lectura deben ser ejecutadas de forma correcta.
- Cantidad de datos leídos o escritos equivalentes a la longitud (len) de la transacción.
- Las direcciones de la transacción deben estar dentro del rango de memoria del subordinado que recibió la transacción.
- Las direcciones de memoria calculadas por el subordinado deben ser iguales a las calculadas dentro del ambiente.
- Los datos escritos deben ser equivalentes a los calculados por el ambiente.
- El ID de la transacción recibida por el subordinado debe ser equivalente al ID del maestro que la realizó y todos los datos deben llegar de forma completa desde el maestro al subordinado para escritura y viceversa para lectura.
- Los datos transmitidos a través de los adaptadores(sockets) deben ser equivalentes.
- El ambiente no debe detectar ningún error dentro de la ejecución de la prueba.

write_read_unique_test

Características:

- Escrituras y lecturas únicas.
- Direcciones por subordinado con rangos de 16MB.
- Transacciones alineadas en 4 bytes.
- Identificador único para cada maestro (ID).
- Quality of service (qos) para cada transacción aleatorio de 1 a 16.
- Respuesta de escrituras de OKAY.

Restricciones(constraints):

- Transacción aleatoria tipo WRITE o READ.
- Ráfaga (burst) aleatorio (FIXED, INCR, WRAP).
- Tamaño (size) aleatorio de 1 a 4 bytes.
- Longitud (len) de 1.
- Estrobo (strobe) dependiente del size y del tipo de transacción.
- Quality of service (qos) aleatorio.
- Dirección de escritura aleatoria dentro del rango de 16MB.
- Paquete aleatorio.
- ID correspondiente al maestro que realiza la transacción.

Criterio de cumplimiento:

- Cumplimiento del protocolo AXI en la escritura y lectura de datos con las restricciones establecidas anteriormente.
- Sólo se puede transmitir un dato por transacción.
- Las direcciones de escritura deben estar dentro del rango de memoria del subordinado que recibió la transacción.
- Las direcciones calculadas por el subordinado tienen que ser equivalentes a las calculadas por el ambiente.

- Los datos escritos tienen que ser equivalentes a los datos calculados por el subordinado.
- El ID de la transacción recibida por el subordinado debe ser equivalente al ID del maestro que la realizó.
- Los datos recibidos tienen que ser equivalentes a los que fueron enviados.
- Los datos transmitidos a través de los adaptadores(sockets) deben ser equivalentes.

write_read_burst_fixed_test

Características:

- Escrituras y lecturas en ráfaga (burst) tipo FIXED.
- Escritura de longitud (len) de 1 a 16 para ráfagas (burst) tipo FIXED.
- Direcciones por subordinado con rangos de 16MB.
- Transacciones alineadas en 4 bytes.
- Identificador único para cada maestro (ID).
- Quality of service (qos) para cada transacción aleatorio de 1 a 16.
- Respuestas de escritura OKAY.

Restricciones(constraints):

- Transacción aleatoria tipo WRITE o READ.
- Ráfaga (burst) tipo FIXED.
- Tamaño (size) aleatorio.
- Longitud (len) aleatorio de 1 a 16.
- Estrobo (strobe) dependiente del size y del tipo de transacción.
- Quality of service (qos) aleatorio.
- Dirección de escritura aleatoria dentro del rango de 16MB.
- Paquete aleatorio.
- ID correspondiente al maestro que realiza la transacción.

Criterio de cumplimiento:

- Cumplimiento del protocolo AXI en la escritura y lectura de datos con las restricciones establecidas anteriormente.
- Transmisión de varios datos por transacción en el rango de 1 a 16
- La dirección de memoria debe ser estática, es decir, la misma para todas las escrituras y lecturas.
- Las direcciones calculadas por el subordinado de las transacciones tienen que ser equivalentes a las direcciones calculadas por el ambiente.
- Los datos escritos tienen que ser equivalentes a los calculados por el subordinado.
- Cantidad de datos transmitidos equivalentes a la longitud (len) de la transacción.
- Las direcciones de escritura y lectura deben estar dentro del rango de memoria del subordinado que recibió la transacción.
- Las direcciones calculadas por el subordinado de las transacciones tienen que ser equivalentes a las direcciones calculadas por el ambiente.
- Los datos escritos tienen que ser equivalentes a los calculados por el subordinado.
- El ID de la transacción recibida por el subordinado debe ser equivalente al ID del maestro que la realizó.
- Los datos recibidos tienen que ser equivalentes a los que fueron enviados.
- Los datos transmitidos a través de los adaptadores(sockets) deben ser equivalentes.

write_read_burst_incr_test

Características:

- Escrituras o lecturas en ráfaga (burst) tipo INCR.
- Escritura o lectura de longitud (len) de 1 a 256 para ráfagas (burst) tipo incremental INCR.
- Direcciones por subordinado con rangos de 16MB.
- Transacciones alineadas en 4 bytes.
- Identificador único para cada maestro (ID).
- Quality of service (qos) para cada transacción aleatorio de 1 a 16.
- Respuestas de escritura OKAY.

Restricciones(constraints):

- Transacción aleatoria tipo WRITE o READ.
- Ráfaga (burst) tipo INCR.
- Tamaño (size) aleatorio de 1 a 4 bytes.
- Longitud (len) aleatorio de 1 a 256.
- Estrobo (strobe) dependiente del size y del tipo de transacción.
- Quality of service (qos) aleatorio.
- Dirección de escritura o lectura aleatoria dentro del rango de 16MB.
- Paquete aleatorio.
- ID correspondiente al maestro que realiza la transacción.

Criterio de cumplimiento:

- Cumplimiento del protocolo AXI en la escritura y lectura de datos con las restricciones establecidas anteriormente.
- Transmisión de varios datos por transacción en el rango de 1 a 256 transferencias.
- Las direcciones de memoria calculadas deben ser equivalentes a las direcciones de memoria esperada de forma incremental.
- Cantidad de datos transmitidos equivalentes a la longitud (len) de la transacción.
- Las direcciones de escritura o lectura deben estar dentro del rango de memoria del subordinado que recibió la transacción.
- Las direcciones calculadas por el subordinado de las transacciones tienen que ser equivalentes a las direcciones calculadas por el ambiente.
- Los datos escritos tienen que ser equivalentes a los calculados por el subordinado.
- El ID de la transacción recibida por el subordinado debe ser equivalente al ID del maestro que la realizó.
- Los datos recibidos tienen que ser equivalentes a los que fueron enviados.
- Los datos transmitidos a través de los adaptadores(sockets) deben ser equivalentes.

write_read_burst_wrap_test

Características:

- Escrituras o lecturas en ráfaga (burst) tipo WRAP.
- Escritura o lectura de longitud (len) de 2, 4, 8, o 16 para ráfagas (burst) tipo WRAP.
- Direcciones por subordinado con rangos de 16MB.
- Transacciones alineadas en 4 bytes.
- Identificador único para cada maestro (ID).
- Quality of service (qos) para cada transacción aleatorio de 1 a 16.
- Respuestas de escritura OKAY.

Restricciones(constraints):

- Transacción aleatoria tipo WRITE o READ.
- Ráfaga (burst) tipo WRAP.
- Tamaño (size) aleatorio de 1 a 4 bytes.
- Longitud (len) aleatoria 2, 4, 8, o 16.
- Estrobo (strobe) dependiente del size y del tipo de transacción.
- Quality of service (qos) aleatorio.
- Dirección de escritura aleatoria dentro del rango de 16MB.
- Paquete aleatorio.
- ID correspondiente al maestro que realiza la transacción.

Criterio de cumplimiento:

- Cumplimiento del protocolo AXI en la escritura y lectura de datos con las restricciones establecidas anteriormente.
- Transmisión de 2, 4, 8, o 16 datos por transacción.
- Las direcciones de memoria calculadas deben ser equivalentes a las direcciones de memoria esperadas en ráfaga tipo WRAP.
- Cuando las direcciones de memoria alcanzan el límite del rango, el siguiente cálculo debe ser la dirección inicial de la transacción, los resultados serán comparados con los resultados esperados.

- Cantidad de datos transmitidos equivalentes a la longitud (len) de la transacción.
- Las direcciones de escritura o lectura deben estar dentro del rango de memoria del subordinado que recibió la transacción.
- Las direcciones calculadas por el subordinado de las transacciones tienen que ser equivalentes a las direcciones calculadas por el ambiente.
- Los datos escritos tienen que ser equivalentes a los calculados por el subordinado.
- El ID de la transacción recibida por el subordinado debe ser equivalente al ID del maestro que la realizó.
- Los datos recibidos tienen que ser equivalentes a los que fueron enviados.
- Los datos transmitidos a través de los adaptadores(sockets) deben ser equivalentes.

write_read_qos_test

Características:

- Quality of service (qos) para cada transacción aleatorio de 1 a 16.
- Direcciones por subordinado con rangos de 16MB.
- Transacciones alineadas en 4 bytes.
- Identificador único para cada maestro (ID).
- Respuestas de escritura OKAY.

Restricciones(constraints):

- Transacción aleatoria tipo WRITE o READ.
- Ráfaga (burst) aleatoria.
- Tamaño (size) aleatorio de 1 a 4 bytes.
- Longitud (len) aleatoria de 1 a 256 para INCR, de 1 a 16 para FIXED y 2, 4, 8 o 16 para WRAP.
- Estrobo (strobe) dependiente del size y de la transacción.
- Quality of service (qos) aleatorio, donde solo uno de los maestros tendrá el mayor qos, los otros iguales.
- Dirección de escritura desde todos los maestros a un sólo subordinado dentro del rango de 16MB en memoria, esto se repite para cada subordinado.

- Paquete aleatorio.
- ID correspondiente al maestro que realiza la transacción.

Criterio de cumplimiento:

- Cumplimiento del protocolo AXI en la escritura o lectura de datos con las restricciones establecidas anteriormente.
- Ejecución por prioridad de las transacciones, la transacción tomada primero siempre debe ser la de mayor quality of service.
- Cantidad de datos transmitidos equivalentes a la longitud (len) de la transacción.
- Las direcciones de escritura o lectura deben estar dentro del rango de memoria del subordinado que recibió la transacción.
- Las direcciones calculadas por el subordinado de las transacciones tienen que ser equivalentes a las direcciones calculadas por el ambiente.
- Los datos escritos tienen que ser equivalentes a los calculados por el subordinado.
- El ID de la transacción recibida por el subordinado debe ser equivalente al ID del maestro que la realizó.
- Los datos recibidos tienen que ser equivalentes a los que fueron enviados.
- Los datos transmitidos a través de los adaptadores(sockets) deben ser equivalentes.

all_for_one_test

Características:

- Ejecución de transacciones desde todos los maestros en un solo subordinado.
- Direcciones por subordinado con rangos de 16MB.
- Transacciones alineadas en 4 bytes.
- Identificador único para cada maestro (ID).
- Respuestas de transacciones OKAY.

Restricciones(constraints):

- Transacción aleatoria.
- Ráfaga (burst) aleatoria.

- Tamaño (size) aleatorio.
- Longitud (len) aleatoria de 1 a 256 para ráfaga INCR y de 1 a 16 para FIXED y de 2, 4, 8 o 16 para WRAP.
- Estrobo (strobe) dependiente del size.
- Quality of service (qos) aleatorio.
- Dirección de transacción de escritura o lectura dentro del rango un solo subordinado de 16MB, esto se repite para cada subordinado.
- ID correspondiente al maestro que realiza la transacción.

Criterio de cumplimiento:

- Cumplimiento del protocolo AXI en la escritura o lectura de datos con las restricciones establecidas anteriormente.
- Todas las transacciones de escritura o lectura de los maestros deben ser ejecutadas exitosamente desde un sólo subordinado.
- Cantidad de datos leídos o escritos equivalentes a la longitud (len) de la transacción.
- Las direcciones de la transacción deben estar dentro del rango de memoria del subordinado que recibió la transacción.
- Las direcciones calculadas por el subordinado de las transacciones tienen que ser equivalentes a las direcciones calculadas por el ambiente.
- Los datos escritos tienen que ser equivalentes a los calculados por el subordinado.
- El ID de la transacción recibida por el subordinado debe ser equivalente al ID del maestro que la realizó y todos los datos deben llegar de forma completa desde el maestro al subordinado para escritura y viceversa para lectura.
- Los datos transmitidos a través de los adaptadores(sockets) deben ser equivalentes.

decerr_gen_test

Características:

- Detección de intento de transacciones que sobrepasan los límites de los rangos de memoria.
- Direcciones por subordinado con rangos de 16MB.
- Transacciones alineadas en 4 bytes.

- Identificador único para cada maestro (ID).
- Respuestas de transacción DECERR (decode error).

Restricciones(constraints):

- Transacción aleatoria.
- Ráfaga (burst) aleatoria entre INCR o WRAP.
- Tamaño (size) aleatorio.
- Longitud (len) aleatoria.
- Estrobo (strobe) dependiente del size.
- Quality of service (qos) aleatorio.
- Dirección de transacción de escritura o lectura fuera o saliendo del rango de memoria de 16MB.
- ID correspondiente al maestro que realiza la transacción.

Criterio de cumplimiento:

- Cumplimiento del protocolo AXI en la escritura y lectura de datos con las restricciones establecidas anteriormente.
- Todas las transacciones de escritura o lectura deben ser ejecutadas pero indicando un error de DECERR.
- El ID de la transacción recibida por el subordinado debe ser equivalente al ID del maestro que la realizó y deben tener el mismo tipo de error.
- El error resultante debe ser el mismo que fue calculado por el ambiente.
- Los datos transmitidos a través de los adaptadores(sockets) deben ser equivalentes.

slverr_gen_test

Características:

- Detección de intento de transacciones con configuración no válida, por ejemplo, un len que no corresponde a una ráfaga tipo WRAP o FIXED.
- Direcciones por subordinado con rangos de 16MB.
- Transacciones alineadas en 4 bytes.

- Identificador único para cada maestro (ID).
- Respuestas de transacción SLVERR (subordinate error).

Restricciones(constraints):

- Transacción aleatoria.
- Ráfaga (burst) WRAP o FIXED.
- Tamaño (size) de 1 a 128 bytes para generar un error.
- Longitud mayor a 16 para la ráfaga tipo FIXED y mayor o sin tomar en cuenta los valores 2, 4, 8 o 16 para ráfaga tipo WRAP.
- Estrobo (strobe) dependiente del size.
- Quality of service (qos) aleatorio.
- Dirección de transacción de escritura o lectura aleatorio en el rango de memoria de 16MB.
- ID correspondiente al maestro que realiza la transacción.

Criterio de cumplimiento:

- Cumplimiento del protocolo AXI en la escritura o lectura de datos con las restricciones establecidas anteriormente.
- Todas las transacciones de escritura o lectura deben ser ejecutadas pero indicando un error de slverr.
- El ID de la transacción recibida por el subordinado debe ser equivalente al ID del maestro que la realizó y deben tener el mismo tipo de error.
- El error resultante debe ser igual al que fue calculado por el ambiente.
- Los datos transmitidos a través de los adaptadores(sockets) deben ser equivalentes.

slverr_decerr_gen_test

Características:

- Detección de intento de transacciones con configuración no válida o con una dirección no válida, por ejemplo, un len que no corresponde a una ráfaga tipo WRAP o FIXED o una dirección que es posible que se transfiera a otro rango.
- Direcciones por subordinado fuera de rango.

- Transacciones alineadas en 4 bytes.
- Identificador único para cada maestro (ID).
- Respuestas de transacción slverr (subordinate error).

Restricciones(constraints):

- Transacción aleatoria.
- Ráfaga (burst) FIXED, INCR o WRAP.
- Tamaño (size) de 1 a 128 bytes para generar un error.
- Longitud mayor a 16 para la ráfaga tipo FIXED y mayor o sin tomar en cuenta los valores 2, 4, 8 o 16 para ráfaga tipo WRAP.
- Estrobo (strobe) dependiente del size.
- Quality of service (qos) aleatorio.
- Dirección de transacción de escritura o lectura aleatorio fuera del rango establecido.
- ID correspondiente al maestro que realiza la transacción.

Criterio de cumplimiento:

- Cumplimiento del protocolo AXI en la escritura o lectura de datos con las restricciones establecidas anteriormente.
- Todas las transacciones de escritura o lectura deben ser ejecutadas pero indicando un error de slverr.
- El ID de la transacción recibida por el subordinado debe ser equivalente al ID del maestro que la realizó y deben tener el mismo tipo de error.
- El error resultante debe ser el mismo calculado por el ambiente, DECERR o SLVERR.
- Los datos transmitidos a través de los adaptadores(sockets) deben ser equivalentes.

reset_test

Características:

- Reset del dut en estados que están a punto de iniciar una transacción.
- Direcciones por subordinado con rangos de 16MB.

- Transacciones alineadas en 4 bytes.
- Identificador único para cada maestro (ID).

Restricciones(constraints):

- Transacción aleatoria.
- Ráfaga (burst) FIXED, INCR o WRAP.
- Tamaño (size) de 1 a 4 bytes.
- Longitud (len) de 1 a 256 para INCR, de 1 a 16 para FIXED y 2, 4, 8 o 16 para WRAP.
- Estrobo (strobe) dependiente del size.
- Quality of service (qos) aleatorio.
- Dirección de transacción de escritura o lectura aleatorio en el rango de memoria de 16MB.
- ID correspondiente al maestro que realiza la transacción.

Criterio de cumplimiento:

- El dispositivo bajo prueba tiene que ser capaz de devolverse a un estado inicial después del reset.
- El dispositivo tiene que ser capaz de reiniciar su funcionamiento sin ningún tipo de problema.
- Ningún componente del dut puede continuar con las transacción en caso de reset, en todo caso, el ambiente detectará si esto sucede.

A.2. Plan de pruebas

A.2.1. Recursos del ambiente

- El ambiente debe ser capaz de enviar estímulos y recibir las respuestas del dut para su análisis.
- El ambiente debe tener un historial de transacciones realizadas.
- El ambiente al finalizar una transacción debe verificar que dicha transacción se encuentre en el historial.

- El ambiente debe ser capaz de aleatorizar los valores necesarios para cada prueba, dichos valores se pueden apreciar en la sección A.1.2.
- Se debe verificar de manera continua que la comunicación cumpla con los lineamientos del protocolo de comunicación AXI.
- El ambiente debe predecir las direcciones de memoria que son calculadas por el subordinado.
- El ambiente debe predecir los datos resultantes una vez aplicado el size y el strobe.
- El ambiente debe ser capaz de predecir el resultado de la transacción, ya sea, una transacción exitosa o algún error (SLVEER, DECERR).
- El ambiente debe predecir el orden en el que se ejecutarán las transacciones.
- El ambiente debe revisar que los datos capturados por los monitores concuerden con la señal de longitud(len) de la transacción.
- El ambiente debe ser capaz de identificar cuando una transacción no llegó a su destino correcto o si no fue enviada por el maestro correcto.
- El ambiente debe verificar de manera continua que los datos transmitidos lleguen de forma correcta sin ningún tipo de perdida.
- El ambiente debe ser capaz de identificar cualquier tipo de error dentro de la simulación.
- El ambiente debe simular la memoria de cada subordinado para poder escribir los datos y leerlos.
- El ambiente debe ser capaz de finalizar la simulación en dado caso que el dispositivo se quede atrapado en algún estado.

A.2.2. Escenarios generales

Transacciones únicas

Escenario

Transacciones de escritura o lectura de un solo valor, es decir, la longitud de la transacción es de 1. Todos los maestros ejecutarán transacciones de escritura o lectura sobre los subordinados direccionados de forma aleatoria.

Objetivo

Verificar que el dispositivo es capaz de realizar lecturas o escrituras de un solo valor

en memoria desde todos los maestros a cualquier rango de memoria.

Cantidad de semillas: 100

Pruebas:

- write_unique_test
- read_unique_test
- write_read_unique_test

Transacciones en ráfaga (burst) tipo FIXED

Escenario

Transacciones de escritura o lectura en ráfaga tipo FIXED. Todos los maestros ejecutarán escrituras o lecturas de ráfaga tipo FIXED en los subordinados de forma aleatoria, donde se hará de 1 a 16 escrituras o lecturas en los subordinados.

Objetivo

Verificar que el dispositivo es capaz de ejecutar lecturas y escrituras en ráfaga tipo FIXED desde todos los maestros con una longitud de 1 a 16 transferencias por transacción a cualquier rango de memoria.

Cantidad de semillas: 100

Pruebas:

- write_burst_fixed_test
- read_burst_fixed_test
- write_read_burst_fixed_test

Transacciones en ráfaga (burst) tipo INCR

Escenario

Transacciones de escritura o lectura en ráfaga tipo INCR. Todos los maestros ejecutarán escrituras o lecturas de ráfaga tipo INCR en los subordinados de forma aleatoria, donde se hará de 1 a 256 escrituras o lecturas en los subordinados.

Objetivo

Verificar que el dispositivo es capaz de ejecutar lecturas y escrituras en ráfaga tipo INCR desde todos los maestros con una longitud de 1 a 256 transferencias por transacción a cualquier rango de memoria.

Cantidad de semillas: 100

Pruebas:

- write_burst_incr_test
- read_burst_incr_test
- write_read_burst_incr_test

Transacciones en ráfaga (burst) tipo WRAP

Escenario

Transacciones de escritura o lectura en ráfaga tipo WRAP. Todos los maestros ejecutarán escrituras o lecturas de ráfaga tipo WRAP en los subordinados de forma aleatoria, donde se hará de 2, 4, 8 o 16 escrituras o lecturas en los subordinados.

Objetivo

Verificar que el dispositivo es capaz de ejecutar lecturas y escrituras en ráfaga tipo WRAP desde todos los maestros con una longitud de 2, 4, 8 o 16 transferencias por transacción a cualquier rango de memoria.

Cantidad de semillas: 100

Pruebas:

- write_burst_wrap_test
- read_burst_wrap_test
- write_read_burst_wrap_test

Transacciones de escritura con estrobos(strobe)

Escenario

Transacciones de escritura de cualquier tipo de ráfaga con los bytes deshabilitados dependientes del size. Todos los maestros ejecutarán transacciones de escritura con una longitud aleatoria para cada tipo de ráfaga, con datos de escritura con un strobe aleatorio.

Objetivo

Verificar que el dispositivo es capaz de aplicar el estrobo de forma correcta a los datos escritos en memoria desde cualquier maestro hacia cualquier subordinado.

Cantidad de semillas: 100

Pruebas:

- write_strobe_test

Transacciones de escritura o lectura con size de 1, 2 o 4

Escenario

Transacciones de escritura o lectura con el tamaño (size) del dato escrito o leído aleatorio de 1, 2 o 4 bytes. Todos los maestros ejecutarán transacciones de escritura o lectura con tipo de ráfaga aleatorio y con longitud de transacción aleatoria sobre todos los subordinados.

Objetivo

Verificar que el dispositivo esta haciendo escrituras o lecturas desde cualquier maestro a cualquier subordinado de forma correcta tomando en cuenta el valor del size y el cálculo de las direcciones de memoria.

Cantidad de semillas: 100

Pruebas:

- write_strobe_test
- read_size_test

Transacciones con diferentes qos

Escenario

Transacciones de escritura o lectura con diferente valor de Quality of service (qos) en cualquier tipo de ráfaga. Todos los maestros ejecutarán transacciones de escritura o lectura con un qos aleatorio sobre todos los subordinados.

Objetivo

Verificar que el dispositivo siempre ejecute primero las transacciones con mayor qos sin importar la configuración de las transacción.

Cantidad de semillas: 100

Pruebas:

- write_qos_test
- read_qos_test
- write_read_qos_test

Transacciones de escritura y lectura

Escenario

Transacciones de escritura o lectura en un sola prueba con tipo de ráfaga aleatoria y longitud aleatoria. Los maestros ejecutarán transacciones de escritura y lectura en un solo test de forma aleatoria sobre todos los subordinados.

Objetivo

Verificar que el dispositivo puede manejar transacciones de escritura y lectura en su funcionamiento normal con cualquier tipo de ráfaga y longitud de transacción.

Cantidad de semillas: 100

Pruebas:

- write_read_test

A.2.3. Escenarios de esquina

Transacciones de escritura o lectura a un solo subordinado

Escenario

Transacciones de escritura o lectura dirigidas a un solo subordinado por todos los maestros al mismo tiempo. Todos los maestros ejecutarán transacciones de escritura o lectura en el rango de memoria de un solo subordinado. Esto será ejecutado para cada subordinado.

Objetivo

Verificar que un subordinado puede manejar las transacciones de todos los maestros sin importar su configuración.

Cantidad de semillas: 100

Pruebas:

- all_for_one_test

Error de decodificación (decerr)

Escenario

Transacciones de escritura o lectura fuera del rango de memoria. Los maestros ejecutarán transacciones fuera de los rangos permitidos por el dispositivo. Todas las transacciones deben resultar en error.

Objetivo

Verificar que el dispositivo es capaz de detectar intentos de acceso a rangos de memoria inexistentes o no permitidos para terminar la transacción con un decode error (decerr).

Cantidad de semillas: 100

Pruebas:

- decerr_gen_test

Error de subordinado (slverr)

Escenario

Transacciones de escritura o lectura con configuración inválida, por ejemplo, tamaño no permitido para las transacciones, longitud de transacción no permitido para el tipo de ráfaga (burst). Los maestros ejecutarán transacciones con una configuración inválida para que el dispositivo pueda reconocer dichos errores. Todas las transacciones deben dar como resultado un error de subordinate error (slverr).

Objetivo

Verificar que el dispositivo es capaz de detectar intentos de ejecución de transacciones con una configuración inválida para terminar la transacción con un subordinate error (slverr).

Cantidad de semillas: 100

Pruebas:

- slverr_gen_test

Error de subordinado (slverr) y error de decodificación (decerr)

Escenario

Transacciones de escritura o lectura con configuración inválida o dirección inválida. Todas las transacciones deben dar como resultado un error de subordinate error (slverr) o error de decodificación (decerr).

Objetivo

Verificar que el dispositivo es capaz de detectar intentos de ejecución de transacciones con una configuración inválida o con una dirección inválida.

Cantidad de semillas: 100

Pruebas:

- slverr_decerr_gen_test

Reset en medio de una transacción

Escenario

Funcionamiento normal del dispositivo pero con un reset en media ejecución.

Objetivo

Verificar que el dispositivo es capaz de reiniciar su funcionamiento correctamente sin generar errores.

Cantidad de semillas: 100

Pruebas:

- reset_test

A.3. Diseño del ambiente

A.3.1. Interfaces

Interfaz para el DUT

La interfaz posee las siguientes señales:

- **clk_in**: señal de reloj.
- **rst_in**: señal de reset.
- **m_full_out [n]**: señal de full de la FIFO del maestro, indica cuando la FIFO de comandos esta llena.
- **m_d_out [n]**: señal de dato de salida del maestro, obtiene la información almacenada en la FIFO de comandos.
- **m_pndng_out [n]**: señal de pendiente de salida del maestro, indica cuando hay datos pendientes por tomar de la FIFO de comandos.
- **m_pop_in [n]**: señal de pop del maestro, cuanto se levanta toma los datos de la FIFO del maestro.
- **m_d_in [n]**: señal de dato de entrada del maestro, transmite datos a la FIFO de datos.

- **m_push_in [n]**: señal de push del maestro, cuando se levanta permite que la FIFO absorba el dato.
- **s_full_out [n]**: señal de full de la FIFO de datos del subordinado, indica cuando la FIFO de datos esta llena.
- **s_d_out [n]**: señal de datos de salida del subordinado, obtiene la información almacenada en la FIFO de datos.
- **s_pndng_out [n]**: señal de pendiente de salida del subordinado, indica cuando hay datos pendientes por tomar de la FIFO de datos.
- **s_pop_in [n]**: señal de pop del subordinado, cuanto se levanta toma los datos de la FIFO del subordinado.
- **s_d_in [n]**: señal de dato de entrada del subordinado, transmite datos a la FIFO de comandos.
- **s_push_in [n]**: señal de push del subordinado, cuando se levanta permite que la FIFO absorba el dato.

Interfaz para AXI

La interfaz posee las siguientes señales:

- **aw_valid [n]**: VALID del canal de dirección de escritura.
- **aw_ready [n]**: READY del canal de dirección de escritura.
- **aw_addr [n]**: dirección de escritura.
- **aw_size [n]**: tamaño en bytes de las transferencias.
- **aw_burst [n]**: tipo de burst.
  - **FIXED**: la dirección de escritura se mantiene fija.
  - **INCR**: la dirección de escritura aumenta conforme se hace cada transferencia.
  - **WRAP**: la dirección de escritura aumenta pero al alcanzar un límite superior inicia desde la dirección más baja de ese rango.
- **aw_id [n]**: identificador único del maestro enviado en el canal de dirección de escritura.
- **aw_len [n]**: cantidad de transferencias de escritura que se realizan.
- **aw_qos [n]**: prioridad de la transacción de escritura.
- **w_valid [n]**: VALID del canal de escritura.

- **w_ready [n]**: READY del canal de escritura.
- **w_last [n]**: señal que indica que la ultima transferencia fue realizada.
- **w_data [n]**: dato de escritura.
- **w_strb [n]**: señal que indica cuáles bytes ignorar de la transferencia.
- **w_id [n]**: identificador único del maestro enviado en el canal de escritura.
- **b_valid [n]**: VALID del canal de respuesta del maestro.
- **b_ready [n]**: READY del canal de respuesta del maestro.
- **b_resp [n]**: resultado de la transacción de escritura.
- **b_id [n]**: identificador único del maestro enviado en el canal de respuesta de escritura.
- **ar_valid [n]**: VALID del canal de dirección de lectura.
- **ar_ready [n]**: READY del canal de dirección de lectura.
- **ar_addr [n]**: dirección de lectura.
- **ar_size [n]**: tamaño en bytes de las transferencias de lectura.
- **ar_burst [n]**: tipo de burst.
  - **FIXED**: la dirección de lectura se mantiene fija.
  - **INCR**: la dirección de lectura aumenta conforme se hace cada transferencia.
  - **WRAP**: la dirección de lectura aumenta pero al alcanzar un límite superior inicia desde la dirección más baja de ese rango.
- **ar_id [n]**: identificador único del maestro que está realizando la lectura.
- **ar_len [n]**: cantidad de transferencias de lectura que se realizan.
- **ar_qos [n]**: prioridad de la transacción de lectura.
- **r_valid [n]**: VALID del canal de lectura.
- **r_ready [n]**: READY del canal de lectura.
- **r_last [n]**: último dato de lectura enviado.
- **r_data [n]**: datos de lectura.
- **r_resp [n]**: resultado de la transacción de lectura.
- **r_id [n]**: identificador único del maestro en el canal de lectura.

A.3.2. Definición de tipos de datos y paquetes

xaction

Este paquete gestiona el envío de datos al driver, permitiéndole interpretar la configuración de las transacciones y generar los estímulos correspondientes. Además, se utiliza para almacenar los resultados de las respuestas del DUT, que luego son enviados al scoreboard para su verificación, los contenidos de dicho paquete se listan a continuación:

- **xaction_type**: indica si la transacción se trata de una de lectura o escritura.
- **w_addr**: indica la dirección inicial de la transacción.
- **payload**: paquete que será transmitido en caso de escritura.
- **strobe**: bytes del paquete que serán ignorados.
- **burst**: tipo de burst de la transacción.
  - **FIXED**: no aumenta de dirección.
  - **INCR**: aumenta de dirección en un rango de memoria con cada dato transmitido.
  - **WRAP**: aumenta la dirección, pero si una dirección límite es alcanzada entonces se devuelve a la primera dirección de ese rango.
- **id**: identificador único de cada maestro.
- **qos**: nivel de prioridad de cada transacción.
- **range**: este valor indica en el rango de memoria que se encuentra la transacción que será ejecutada.
- **addrs_list**: lista de direcciones en las que se realizará la lectura o escritura.
- **error**: este bit predice si la transacción finalizará en error.
- **range**: este bit representa al subordinado que se dirige la transacción.
- **start**: variable que almacena el tiempo de inicio de la transacción.
- **finish**: variable que almacena el tiempo de finalización de la transacción.
- **monitor_payload**: queue que almacena los datos de escritura transmitidos.
- **monitor_strb**: queue que almacena los strobes transmitidos.
- **bresponse**: variable que almacena la respuesta de la transacción de escritura al finalizar.

- **bid**: id que es parte de la respuesta de la escritura.
- **monitor_read_data**: este queue almacena los datos que se leen desde el subordinado.
- **monitor_read_resp**: este queue almacena las respuestas de los datos leídos.
- **monitor_read_id**: id que es parte de la respuesta de lectura.
- **monitor_xaction_addr**: este queue almacena las direcciones de escritura o lectura, dependiendo de cual sea el tipo de transacción.
- **predict_addr**: este queue almacena las direcciones de memoria esperadas que se calcularon tomando en cuenta la configuración de la transacción.
- **predict_data**: este queue almacena los datos de escritura esperados luego de aplicar un strobe.

A.3.3. Definición de bloques y estructuras

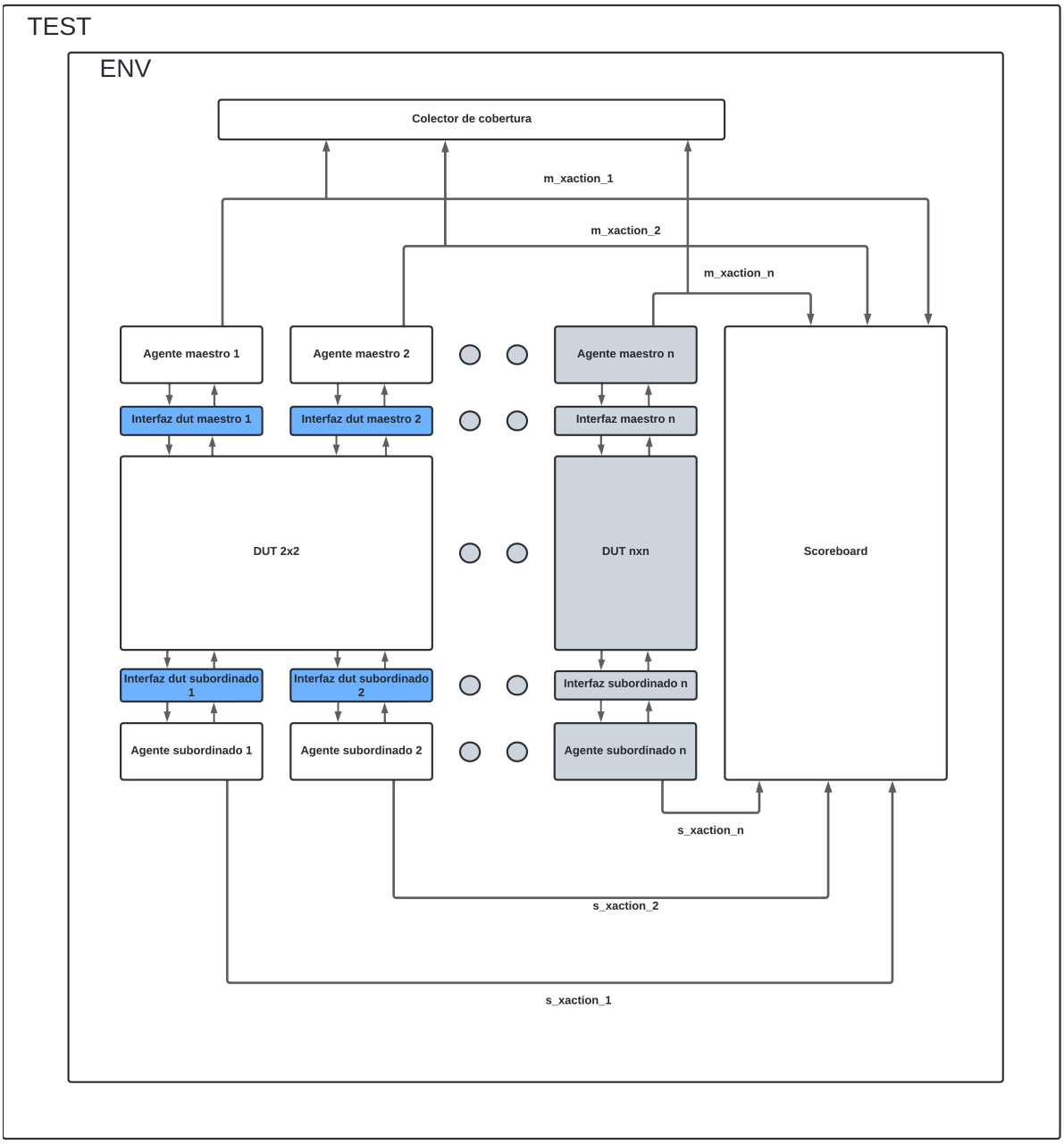


Figura A.1: Estructura del ambiente de verificación.

Consideraciones del ambiente según la figura A.1:

- Todos los componentes ilustrados en la figura son parte del ambiente de verificación, el DUT no está incluido en ese dominio.
- El DUT indica que es 2x2 simplemente con propósitos de ejemplificación, el DUT real es de 4x4.

- Los bloques en color gris indican que si el DUT es de un tamaño mayor, estos mismos componentes pueden extenderse para adaptarse a los requerimientos del DUT.
- Las señales de m_xaction y s_xaction descritas en las sección A.3.2 son enviadas para verificar la existencia de la transacción en el subordinado respectivo.
- El bloque de interfaz se encarga de comunicar el ambiente con el DUT descrito en la figura A.2, además de permitir al monitor del maestro y del subordinado escuchar las señales para generar los paquetes que serán enviados al scoreboard para ser verificados.

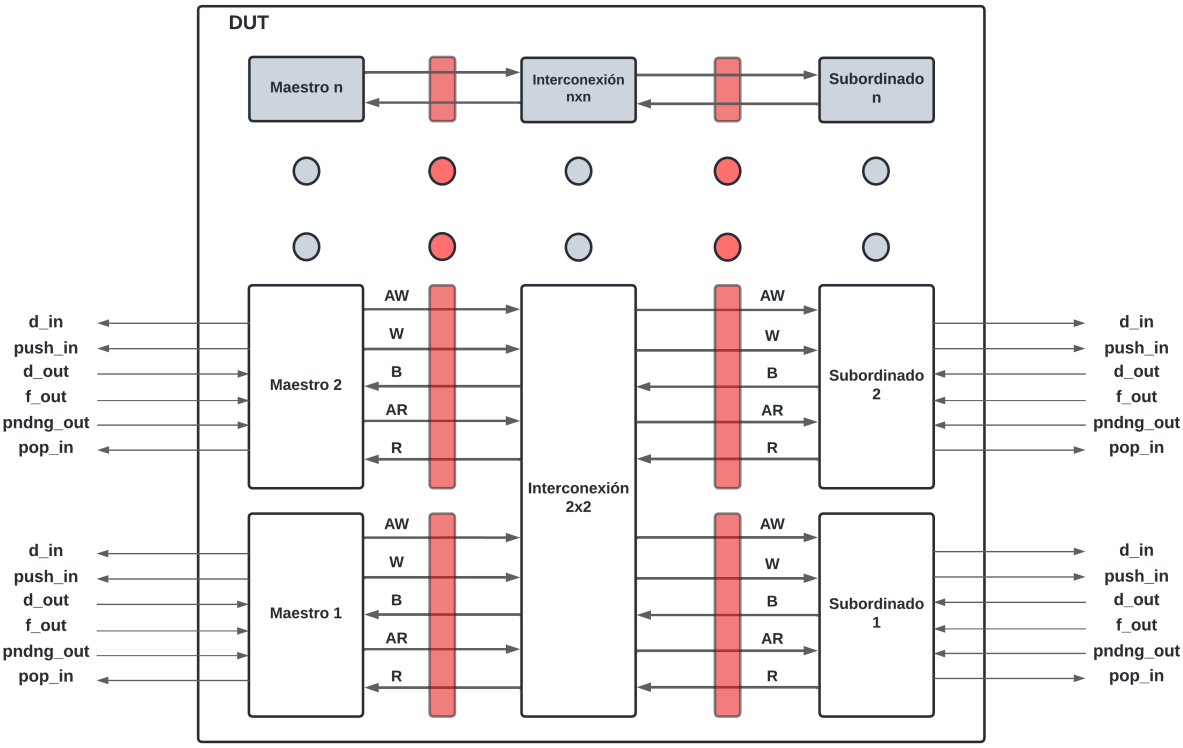


Figura A.2: Dispositivo bajo prueba (DUT).

Consideraciones del dispositivo bajo prueba (DUT) según la figura A.2:

- El DUT es un módulo de interconexión 2x2 a forma de ejemplificación, sin embargo, los bloques grises y los círculos indican que este bloque puede extenderse dependiendo de la cantidad de conexiones de maestros y subordinados.
- Los bloques rojos ilustran la interfaz descrita en la sección A.3.1 que será usada por el monitor del maestro y el monitor del subordinado para tomar las transacciones generadas y enviarlas al scoreboard, además de la verificación del cumplimiento del protocolo AXI.

- El tipo de verificación es de caja gris, es decir, se están accediendo a señales internas del DUT y se espera un comportamiento específico en los puertos de salida del mismo.

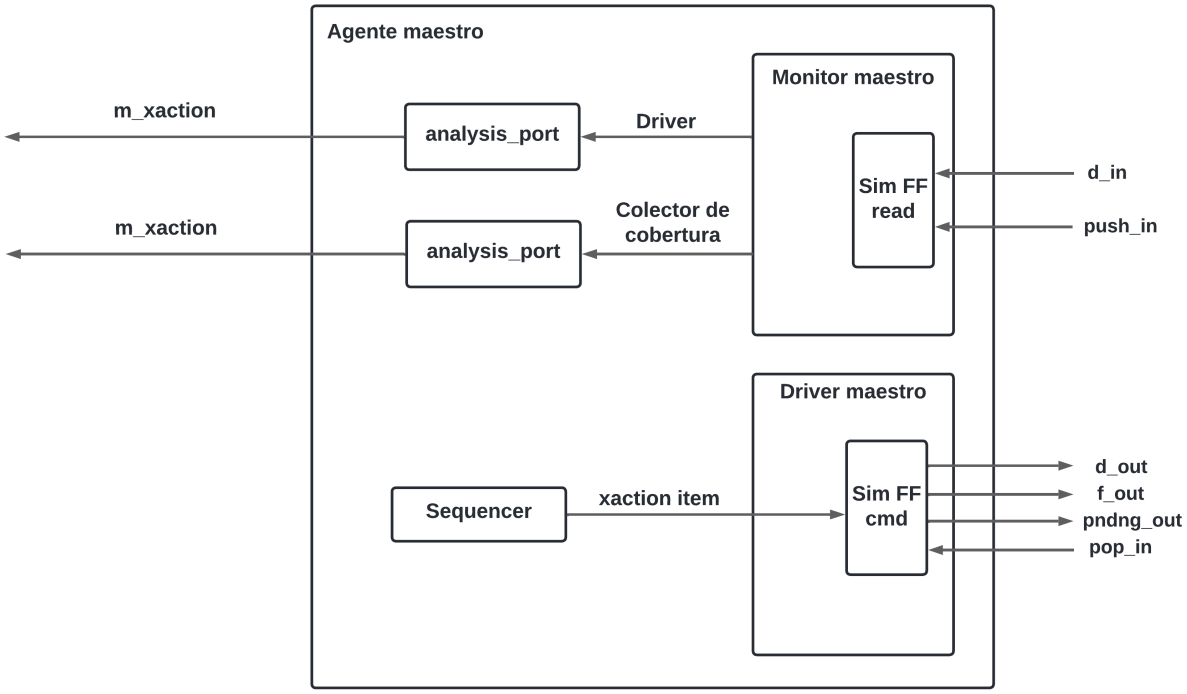


Figura A.3: Agente del maestro.

Consideraciones de la figura A.3:

- El driver va a generar los estímulos mediante el paquete xaction descrito en la sección A.3.2 recibido desde el sequencer.
- El monitor envía por un puerto de análisis el paquete xaction con las señales escuchadas desde la interfaz para su verificación.
- El monitor verifica que la información de entrada del maestro concuerda con la información de la salida del mismo.
- El bloque sequencer se encarga de enviar cada uno de los paquetes xaction al driver para que este los interprete y los convierta en estímulos.
- El driver posee una FIFO de comandos que se encarga de enviar las transacciones al DUT.
- El monitor posee una FIFO de datos leídos que se encarga de almacenar los datos leídos.

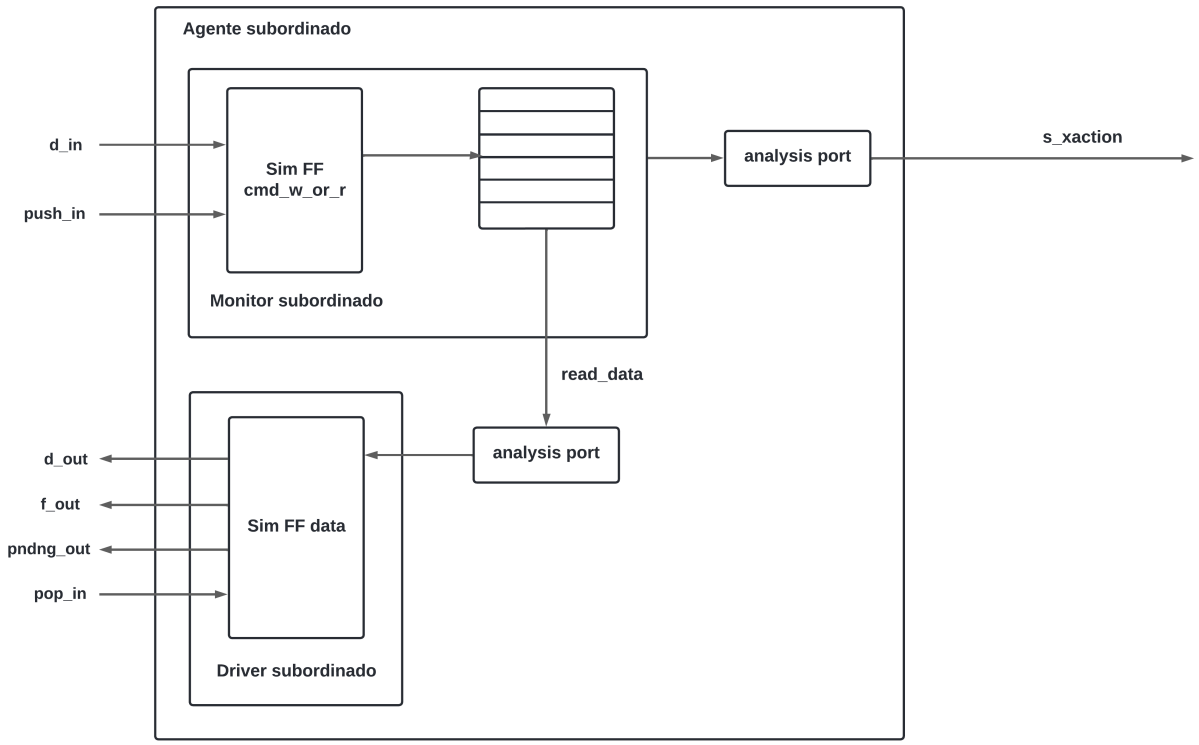


Figura A.4: Agente del subordinado.

Consideraciones de la figura A.4:

- El agente del subordinado posee una memoria interna que se encarga de simular una memoria real para poder responder a las transacciones de lectura y escritura.
- Los datos de escritura recibidos por la FIFO de comandos se almacenan en la memoria interna del monitor.
- La FIFO de datos en el driver se encarga de enviar los datos de lectura al maestro en el caso que se esté realizando una transacción de lectura.

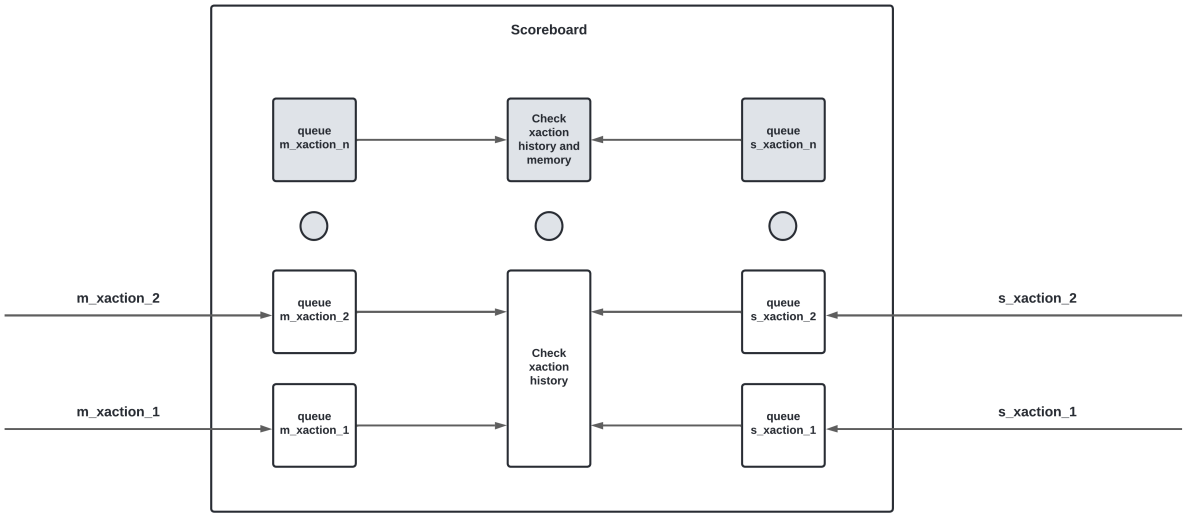


Figura A.5: Scoreboard.

Consideraciones de la figura A.5:

- Los bloques en color gris ilustran que estos pueden ser extendidos a la cantidad de conexiones de maestros y subordinados.
- El bloque en color blanco al igual que todos los anteriores esta diseñado para un módulo de interconexión de 2x2, sin embargo, esto es meramente con propósitos de ejemplo, el DUT puede tener más conexiones.
- Los queue de m_xaction y s_xaction almacenan los items xactions enviados desde los maestros y los subordinados para su verificación, dicho paquete tiene funciones internas para generar las predicciones, dichas predicciones se verifican desde el monitor.
- El bloque de verificación de historial se encarga de verificar que efectivamente la transacción ejecutada por el maestro llegó al subordinado correspondiente.

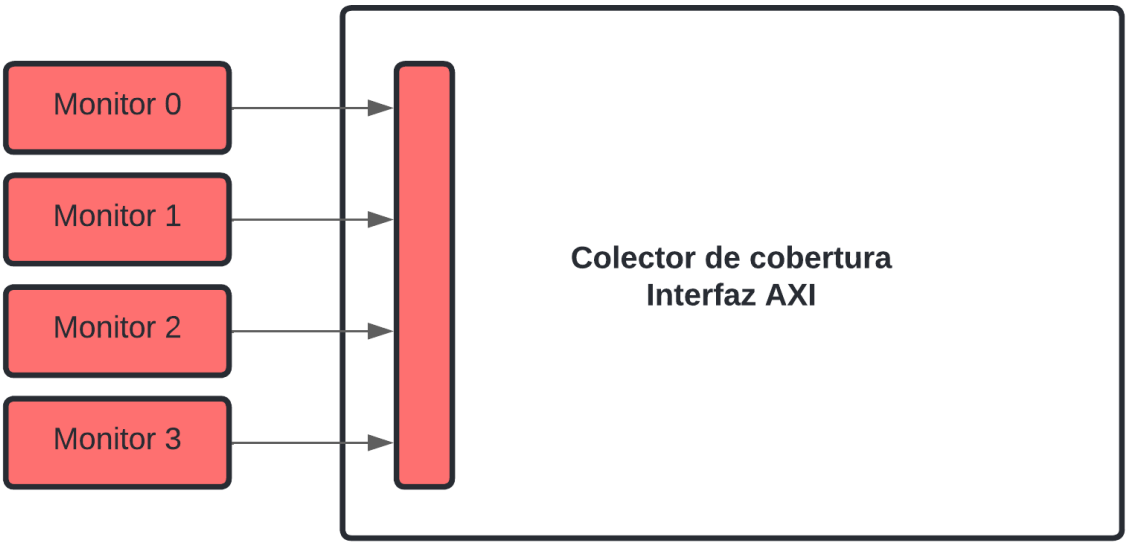


Figura A.6: Colector de cobertura.

Consideraciones de la figura A.5:

- Los bloques rojos externos representan a los maestros enviando la información de las transacciones que se están ejecutando.
- El bloque en color rojo interno representa el procesamiento de la información que le llega al colector de cobertura desde los maestros.