DESDE 2013
https://repository.uaeh.edu.mx/revistas/index.php/icbi/issue/archive
Pädi Boletín Científico de Ciencias Básicas e Ingenierías del ICBI
ISSN: 2007-6363

Publicación Semestral Pädi Vol. 12 No. Especial 4 (2024) 1-6

Verificación funcional de sistemas digitales descritos en HDL mediante ambientes UVM basado en agentes
Functional Verification of Digital Systems Described in HDL Using UVM-Based Environments with Agents

Miguel A. Aleman-Arce[a,*], Salvador Mendoza-Acevedo[b], Luz N. Oliva-Moreno[c]

[a] *Laboratorio Microse del Centro de Investigación en Computación, Instituto Politécnico Nacional, 07700, Ciudad de México, México.*
[b] *Centro de Nanociencias y Micro y Nanotecnologías, Instituto Politécnico Nacional, 07738, Ciudad de México, México.*
[c] *Unidad Interdisciplinaria de Ingeniería campus Hidalgo, Instituto Politécnico Nacional, 42184, Pachuca, Hidalgo, México.*

Resumen

Actualmente, la creciente demanda de circuitos integrados (chips) está generando una necesidad cada vez mayor de profesionales especializados en el diseño de estos componentes. Las industrias, como la automotriz y la informática móvil, no solo están requiriendo un mayor número de chips, sino que también enfrentan una expansión en las aplicaciones que requieren estos circuitos. Esto incluye desde microprocesadores avanzados hasta sistemas embebidos complejos, que requieren el uso de Lenguajes de Descripción de Hardware (HDL) a Nivel de Transferencia de Registros (RTL) y expertos en verificación. La verificación funcional se vuelve crucial para asegurar que los sistemas diseñados cumplan con los requisitos especificados en las propuestas de los clientes. Para llevar a cabo esta verificación, se emplea la Metodología Universal de Verificación (UVM), una metodología estandarizada que facilita la creación de entornos de verificación modulares, mantenibles, escalables y reutilizables. Este trabajo propone la aplicación de UVM en la creación de entornos de verificación basados en agentes, lo que garantiza una alta modularidad, capacidad de expansión y reutilización, mejorando la efectividad de la verificación y permitiendo una rápida detección de errores en el diseño.

Palabras Clave: Verificación funcional, metodología universal de verificación, sistemas digitales, diseño digital.

Abstract

Currently, the increasing demand for integrated circuits (chips) is creating a growing need for professionals specialized in their design. Industries such as automotive and mobile computing are not only requiring a greater number of chips but are also facing an expansion in the applications that need these circuits. This includes advanced microprocessors and complex embedded systems, which require the use of Hardware Description Languages (HDL) at the Register Transfer Level (RTL) and verification experts. Functional verification becomes crucial to ensure that the designed systems meet the requirements specified in client proposals. To perform this verification, the Universal Verification Methodology (UVM) is employed—a standardized methodology that facilitates the creation of modular, maintainable, scalable, and reusable verification environments. This work proposes applying UVM to create agent-based verification environments, ensuring high modularity, expandability, and reusability, thereby improving verification effectiveness and enabling rapid detection of design errors.

Keywords: Functional verification, universal verification methodology, digital system, digital system design.

1. Introducción

El aumento constante en la demanda de Circuitos Integrados (CI), tales como microprocesadores, sistemas embebidos y sistemas en chip (SoC), ha intensificado la necesidad de especialistas en diseño de CI e ingenieros de verificación de sistemas digitales (FUMEC, 2024). Con el avance de las tecnologías, los requisitos de estos sistemas digitales han incrementado en complejidad, lo que demanda nuevas metodologías de diseño y verificación. En los equipos de trabajo en estas áreas, se sugiere que un 30% de los ingenieros se dediquen al diseño del sistema digital y un 70% a la verificación (Bergeron, 2006). Esto subraya la importancia de garantizar que el sistema cumpla con las funciones

*Autor para la correspondencia: maleman@ipn.mx
Correo electrónico: maleman@ipn.mx (Miguel Angel Aleman-Arce), smendozaa@ipn.mx (Salvador Mendoza-Acevedo), loliva@ipn.mx (Luz Noe Oliva-Moreno).

Historial del manuscrito: recibido el 26/06/2024, última versión-revisada recibida el 30/07/2024, aceptado el 30/07/2024, publicado el 30/11/2024. **DOI:** https://doi.org/10.29057/icbi.v12iEspecial4.13286

especificadas en la propuesta del proyecto y que satisfaga los requisitos tanto al implementarse en una Matriz de Compuertas Programables en Campo (FPGA, por sus siglas en inglés) como en un CI de propósito específico.

Los métodos tradicionales de verificación han demostrado ser insuficientes para abordar la complejidad de los nuevos sistemas digitales. La verificación funcional asegura que se cumplan los requisitos del diseño, convirtiéndose en una etapa crucial en el proceso de implementación de un sistema digital.

Actualmente, los métodos de verificación se han estandarizado en metodologías avanzadas que permiten la creación de plataformas de verificación robustas. Estas plataformas no solo verifican de manera exhaustiva el funcionamiento del sistema, sino que también promueven la reutilización, mantenibilidad y escalabilidad del entorno de verificación (Bhuvaneswary, 2022).

Una de estas metodologías es UVM (Universal Verification Methodology), implementada en SystemVerilog. UVM aprovecha la programación orientada a objetos y proporciona una amplia gama de bibliotecas para generar entornos de verificación. Aunque UVM incluye varios módulos o componentes, en este trabajo nos centraremos en uno de los más esenciales: el agente. Los agentes en UVM encapsulan objetos similares y su configuración permite almacenar la información necesaria y gestionar múltiples instancias del mismo agente a través de una base de datos.

En los trabajos de verificación revisados (Franchesconi, 2015; Vaca, 2017), se propone la creación de entornos de verificación utilizando UVM con un solo agente activo. Este agente se encarga de enviar secuencias al Dispositivo Bajo Prueba (DUT), a través del *Driver* y recibir las respuestas del DUT mediante el *Monitor*, gestionando todo el proceso desde una única instancia. Por otro lado, en los proyectos de verificación presentados por Cooper (2023) y Rodríguez (2018), se sugiere la utilización de dos agentes distintos: un agente pasivo que maneja las secuencias enviadas desde el *Driver* a través de la interfaz, y otro agente pasivo que recibe exclusivamente las señales de respuesta del DUT, también a través de la interfaz, para el *Monitor*.

Este trabajo propone el uso de UVM como metodología de verificación en el diseño de CI, enfocándose en la descripción y aplicación de agentes para mejorar la eficiencia del proceso de verificación.

2. UVM basado en agentes.

La implementación de una metodología de verificación como UVM tiene como objetivo establecer un flujo de trabajo uniforme en el equipo de verificación y crear entornos de verificación reutilizables. Al generar un entorno de verificación, se puede aumentar la productividad al diseñar componentes con una visión de reutilización tanto a nivel de bloque como de sistema. UVM facilita la modularidad, lo que oculta la complejidad, permite la reutilización y simplifica el proceso de verificación (Cooper, 2023).

UVM es compatible con la mayoría de los simuladores y ha sido establecido por Accellera, una iniciativa dedicada a la creación de estándares para el diseño, modelado y verificación de sistemas digitales. Además, las bibliotecas de UVM son de código abierto, lo que permite su acceso y modificación en cualquier etapa del proyecto si fuera necesario.

2.1. Implementación de UVM basado en agentes.

El primer ambiente de verificación que se presenta es de un circuito combinacional simple, un multiplexor de 4 entradas de 4 bits de ancho de palabra cuyo diagrama se observa en la Figura 1.

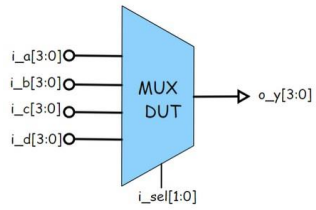

Figura 1: Diagrama del dispositivo bajo prueba (DUT).

A partir de este paso, el multiplexor se conocerá como el Dispositivo Bajo Prueba (DUT). Una vez definido el requerimiento de diseñar un multiplexor, se plantea un plan de verificación. Es necesario diseñar un plan de verificación donde se especifique no solo qué se va a probar, sino también cómo se va a probar. Para este ejemplo, se propone probar: 1) todas las posibles opciones para la señal ***i_sel***, y 2) todos los posibles valores que puedan portar las entradas al multiplexor y, por lo tanto, ser transmitidas a la salida del multiplexor.

Se propone un ambiente de verificación UVM basado en agentes, como se muestra en la Figura 2. En el ambiente propuesto, se considera que cada uno de los bloques pueda ser reutilizado en futuros proyectos, de tal manera que se pueda generar una estructura o modelo base para el grupo de verificación.

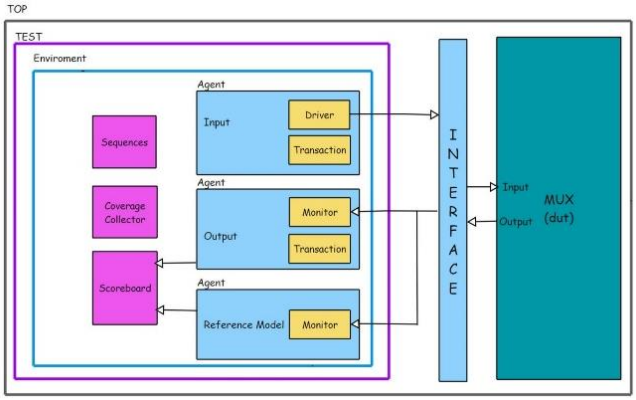

Figura 2: Diagrama a bloques del ambiente de verificación.

Se generarán tres agentes: el agente de las entradas, el agente de las salidas y el agente del modelo de referencia. Cada agente encapsulará una funcionalidad importante en el proceso de verificación. Los agentes son clases creadas extendiendo una clase disponible en las bibliotecas de UVM llamada ***uvm_agent***, en la cual podemos encapsular los objetos que se utilizarán para enviar una secuencia de estímulos al DUT a través de la interfaz. Los componentes típicos de un agente son: Diver, Monitor y secuenciador. El secuenciador controlará el flujo de la secuencia que contendrá la información con la que se generará la señal que estimulará el DUT, de esta manera se enviará la transacción al Driver para que este generé la señal que se enviará al DUT a través de la interfaz.

En el caso analizado, el agente de entradas se encargará de manejar la secuencia y comunicarla al DUT a través de la interfaz. Los resultados generados por el DUT se enviarán al *Monitor* del agente de salidas a través de la interfaz. De esta manera el agente de las entradas se encargará solo de manejar las señales que estimularan el DUT y el agente de salidas se encargara de recibir la respuesta a través del *Monitor*.

Para evitar utilizar pruebas dirigidas y darle más fortaleza a la prueba, las señales con la información con la que se generarán las señales para estimular el DUT se generarán con la función *randomize* de *SystemVerilog* (Spear, 2020). En la Figura 3 se puede observar el flujo que sigue la información, desde que se genera la transacción hasta que llega al tablero de resultados.

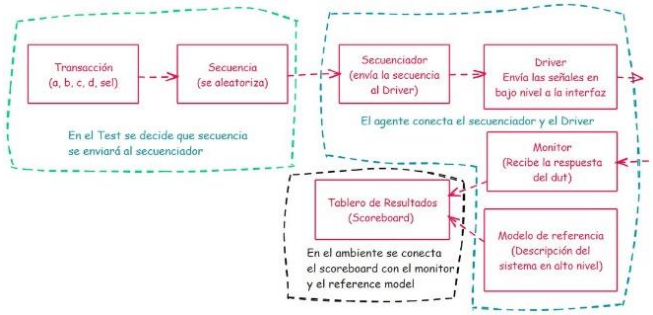

Figura 3: Flujo de la transacción generada.

Como cualquier *testbench* plano, podemos observar las señales generadas utilizando la herramienta de visualización de formas de onda, Figura 4.

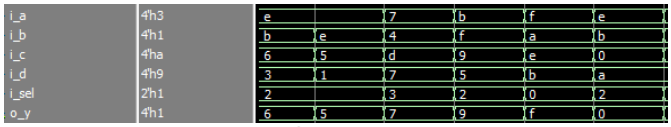

Figura 4: Visualización de señales respuesta del dut.

La primera verificación consiste en revisar de manera visual la forma de las señales generadas y comprobar que las salidas obtienen el resultado esperado, sin embargo, uno de los propósitos de implementar una metodología de verificación es generar información que nos dé un reporte completo de la cobertura obtenida de la prueba de funcionalidad del diseño.

3. Cobertura Funcional

Durante la verificación del sistema digital, uno de los objetivos es medir objetivamente la cobertura funcional de la prueba y del diseño (Metha, 2020). La mayoría de los autores manejan la cobertura funcional como un lenguaje independiente dentro de la metodología de verificación funcional que se debe conocer para obtener resultados satisfactorios. En el diagrama a bloques de la Figura 5 se puede observar como ahora, además estimular y el dispositivo bajo prueba y verificar la respuesta obtenida, obtenemos información de la funcionalidad del sistema.

El análisis de la cobertura funcional busca determinar si se ha cubierto completamente el diseño. En otras palabras, se pretende evaluar si el *testbench* ha sido lo suficientemente exhaustivo como para garantizar que todas las pruebas sean adecuadas.

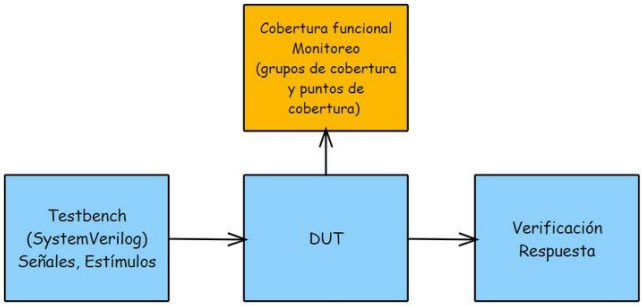

Figura 5: Implementación de la cobertura funcional.

En el caso del multiplexor, el plan de verificación incluye la estimulación para asegurar que todas las entradas puedan ser seleccionadas para observar la salida correspondiente y que todas las combinaciones posibles de datos de entrada sean probadas.

Se proponen dos grupos de cobertura: uno para verificar que se generen todas las combinaciones posibles de ceros y unos, y otro para asegurar que todos los valores de salida sean evaluados. La Figura 6 ilustra la estructura general de la definición de un grupo de cobertura.

```
covergroup inputs;  // grupo de covertura  etiqueta

  a_in: coverpoint mx_vif.A {
    bins zeros  = {4'b0000};          // ceros
    bins others = {['h0001:4'b1110]}; // valores intermedios
    bins ones   = {4'b1111};          // unos
```

Figura 6: Definición de un grupo de cobertura.

La cobertura funcional ofrece información sobre cuántas veces se ha ejecutado una prueba y si esta ha cumplido de manera exhaustiva con todas las funciones requeridas. En particular, evalúa si se han probado todas las entradas, si cada entrada ha sido combinada con todas las posibles variaciones de datos, y si se han generado todas las posibles combinaciones de salida. La información obtenida puede visualizarse de distintas formas: gráficamente, a través de barras que muestran porcentajes, como se ilustra en la Figura 7, o en un reporte de texto plano, como se detalla en la Figura 8.

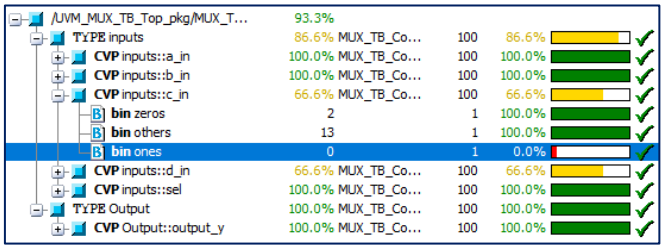

Figura 7. Reporte de cobertura funcional gráfico.

Siempre surge la pregunta: ¿es suficiente alcanzar una cobertura del 100%? En algunos casos, la respuesta dependerá de la adecuada implementación del plan de verificación propuesto. En otros, es recomendable complementar la evaluación con información adicional generada durante la simulación, como la cobertura de código. La cobertura de código proporciona información sobre la exhaustividad con la que se ha estimulado la estructura del código, mientras que la cobertura funcional indica si el diseño cumple con su propósito en su totalidad.

La verificación utilizando UVM también permite realizar una cobertura estructural mediante la cobertura de código. En

la Figura 10 se presenta el reporte en texto plano de las coberturas obtenidas durante la simulación en el entorno UVM. Aunque el reporte muestra que la cobertura funcional ha alcanzado el 100%, la cobertura de código sigue siendo insuficiente. En este caso, es fundamental que el equipo de verificación informe al equipo de diseño para que evalúe la necesidad de alcanzar una cobertura de código del 100%.+

```
Coverage Report Summary Data by file

================================================================================
=== File: ../RTL/MUX.v
================================================================================
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Stmts                            6         5         1      83.3
    Branches                         5         4         1      80.0
    FEC Condition Terms              0         0         0     100.0
    FEC Expression Terms             0         0         0     100.0
    Toggle Bins                     44        44         0     100.0

TOTAL COVERGROUP COVERAGE: 93.3%  COVERGROUP TYPES: 2

TOTAL ASSERTION COVERAGE: 100.0%  ASSERTIONS: 1

Total Coverage By File (code coverage only, filtered view): 87.7%
```

Figura 8: Reporte de coberturas en texto plano.

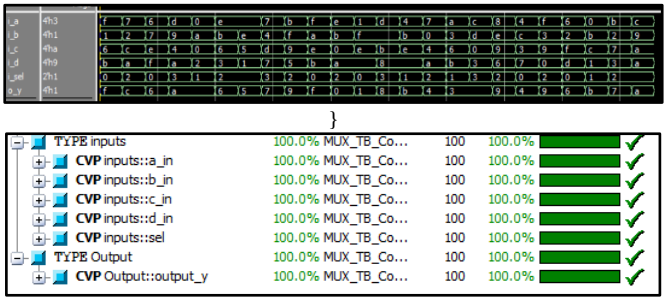

Figura 9: Obtención del 100% de cobertura funcional.

```
Coverage Report Summary Data by file

================================================================================
=== File: ../RTL/MUX.v
================================================================================
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Stmts                            6         5         1      83.3
    Branches                         5         4         1      80.0
    FEC Condition Terms              0         0         0     100.0
    FEC Expression Terms             0         0         0     100.0
    Toggle Bins                     44        44         0     100.0

TOTAL COVERGROUP COVERAGE: 100.0%  COVERGROUP TYPES: 2

TOTAL ASSERTION COVERAGE: 100.0%  ASSERTIONS: 1

Total Coverage By File (code coverage only, filtered view): 87.7%
```

Figura 10: Reporte de coberturas.

4. Modelo de Referencia

Otro módulo que se puede incorporar al ambiente de verificación UVM es el modelo de referencia. El modelo de referencia se incorpora como una agente donde se describe el DUT preferentemente en un modelo de abstracción más alto. Al ser un *testbench* y no la descripción de un sistema digital que tiene que cumplir con la sintaxis necesaria para poder ser sintetizados, el modelo de referencia se puede describir usando todas las herramientas disponibles en SystemVerilog para poder ser descrito. En la Figura 11 se puede observar a) modelo del multiplexor utilizando una estructura *case*, b) modelo del multiplexor usando una estructura *if*.

```
case(i_sel)
     2'b00 : o_y = i_a;
     2'b01 : o_y = i_b;
     2'b10 : o_y = i_c;
     2'b11 : o_y = i_d;
     default : o_y = 4'b0000;
   endcase
```

a)

```
if (sel_sim == 2'b00 )
 mux_output_tran.y = a_sim;
else if (sel_sim == 2'b01 )
 mux_output_tran.y = b_sim;
else if (sel_sim == 2'b10 )
 mux_output_tran.y = c_sim;
else if (sel_sim == 2'b11 )
 mux_output_tran.y = d_sim;
else
 mux_output_tran.y = 4'b0000;
```

b)

Figura 11: Representación del multiplexor con dos modelos diferentes

El modelo de referencia tomará los valores generados en la secuencia manejada por las componentes del agente de entrada a través de la base de datos, las procesara en el *Monitor* y generara resultados que se enviaran tablero de resultados o *Scoreboard*. El *Scorboard* recibirá los resultados obtenidos por el modelo de referencia y los resultados del DUT a través del agente de salida. Si los resultados son diferentes, el *Scorboard* podrá generar un mensaje de error que nos indicará que el resultado de alguna manera no es correcto y se tendrá que analizar.

5. Reutilización del código.

Para probar la reutilización del código, se creó un entorno UVM basado en agentes para la Máquina de Estados Finitos (FSM) que se muestra en la Figura 11. En esta FSM, AB representan los posibles estados, *x* es la entrada, *y* es la salida, y *t* y *t+1* representan el estado actual y el siguiente estado, respectivamente.

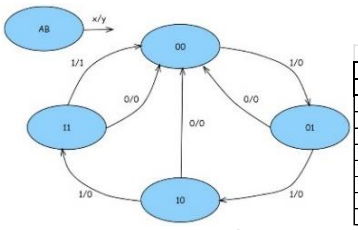

Tabla de estados

t			t+1		
A	B	x	A	B	y
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	1	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	0	0
1	1	0	1	1	0
1	1	1	1	1	1

Figura 12: Máquina de estados FSM a verificar.

El modelo a bloques del ambiente de simulación se muestra en la Figura 13.

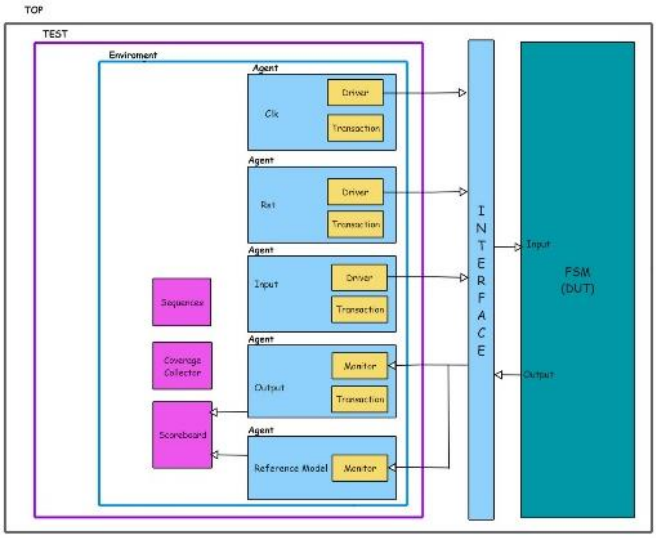

Figura 13: Diagrama a bloques UVM del FSM.

En el caso de la FSM se proponen 2 agentes más para encapsular las secuencias que se pueden generar de la entrada correspondiente al reloj (Clk) y la entrada correspondiente al *reset* (Rst); de esta manera se pueden incorporar transacciones como la frecuencia del reloj, duración del *reset*, ciclo de trabajo del *reset* o cualquier característica que se quisiera controlar y monitorear. De la misma manera se incorpora un agente para manejar la señal de entrada *x* a la FSM y un agente de salida para monitorear los resultados generados de la estimulación del DUT.

Para la FSM el plan de verificación es asegurarse que se genere todos los estados de entrada posibles de manera seudo aleatoria, que se genere un resultado de *y =1* y generar todas las transiciones posibles entre estados. En la Figura 14 se pueden visualizar las señales generadas para la estimulación del DUT y en la Figura 15 se puede observar el resultado del colector de cobertura. El visualizador de las señales continúa siendo una herramienta importante en la verificación del sistema. En las formas de onda se puede analizar con más detalle la forma en la que está estimulando el DUT y las señales que se están generando debido a estas. En el caso del colector de coberturas se puede observar que la descripción de los grupos de cobertura y los puntos de cobertura se están cumpliendo al 100%, sin embargo, también se puede observar que la cobertura de código solo alcanza una cobertura del 98%, este resultado se tendrá que analizar con el grupo de diseño.

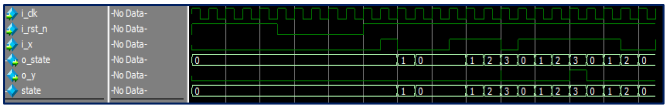

Figura 14: Visualización de señales de la prueba.

```
Coverage Report Summary Data by file

================================================================================
=== File: ../RTL/MyFSM.v
================================================================================
    Enabled Coverage            Active      Hits    Misses % Covered
    ----------------            ------      ----    ------ ---------
    Stmts                            7         7         0     100.0
    Branches                        13        12         1      92.3
    FEC Condition Terms              0         0         0     100.0
    FEC Expression Terms             3         3         0     100.0
    Toggle Bins                     16        16         0     100.0

TOTAL COVERGROUP COVERAGE: 100.0%  COVERGROUP TYPES: 3

TOTAL ASSERTION COVERAGE: 100.0%  ASSERTIONS: 6

Total Coverage By File (code coverage only, filtered view): 98.0%
```

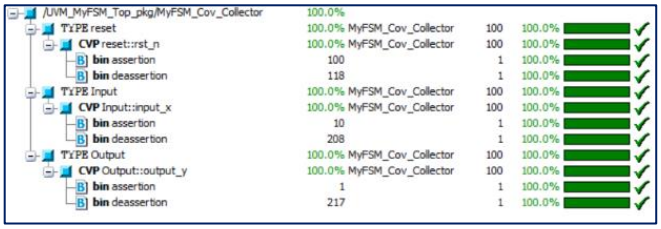

Figura 15: Resultado de cobertura.

Para la verificación de la FSM también se agrega un Modelo de Referencia el cual toma la información generada por la secuencia y calcula el estado y la salida que debe tomar la FSM. Finalmente es comparada en la Tabla de Resultados (*Scoreboard*) con los resultados obtenidos del DUT generados de la estimulación.

6. Resultados

Con la implementación de un ambiente de verificación UVM basado en agentes se pueden integrar diversas herramientas, con las cuales se puede verificar de manera exhaustiva un diseño digital descrito en VHDL, verilog y systemverilog.

La separación en módulos de los diferentes componentes que forman el ambiente de verificación nos permite reusabilidad acortando los tiempos en los procesos de verificación. En el ambiente de verificación, cada componente es la extensión de una clase definida en las bibliotecas de UVM.

Usando modularidad en los ambientes de verificación, permite encapsular bloques con características específicas cuya funcionalidad se requiera revisar. Trabajando la modularidad con agentes en ambientes UVM se encapsulan objetos similares cuya configuración permite almacenar información y manejarla a través de la base de datos configurada para el ambiente.

Se propuso un modelo de UVM basado en agentes para poder manejar de una manera más simple generando agentes para cada aspecto de diseño que se quiera monitorear. Se proponen agentes independientes dependiendo de la funcionalidad de la señal, por ejemplo, una agente para las entradas, un agente para las salidas, un agente para la señal de reloj y una agente para la señal de *reset* en el caso de la FSM, de esta manera se pueden controlar de manera independiente la secuencias que se enviaran al DUT para estimularlo, entradas, reloj, *reset*, y monitorear la señal de salida que el DUT generara como respuesta a la excitación. De esta manera lograremos mayor reusabilidad y mayo control y monitoreo en el ambiente de verificación.

Los ambientes de verificación como UVM, permiten a los ingenieros del grupo de verificación generar plataformas de verificación reusables, a los cuales pueden dar mantenimiento y se convierten en ambientes con posibilidad de crecimiento.

7. Conclusiones

El proceso de verificación en el diseño de sistemas digitales es un componente crucial que requiere un enfoque metódico y riguroso. La adopción de una metodología estandarizada, como la Metodología Universal de Verificación (UVM), confiere solidez y validez a las propuestas de diseño, garantizando que estos sistemas cumplan con sus objetivos previstos y funcionen adecuadamente.

Un plan de verificación bien estructurado permite comprobar que el diseño cumple con los requerimientos solicitados por el usuario, a través de un exhaustivo reporte de verificación. Este reporte valida la funcionalidad del sistema y asegura que todas las especificaciones se han cumplido.

Dada la creciente complejidad de los sistemas digitales, especialmente en el contexto de sistemas en chip (SoC), la verificación se ha convertido en una tarea de gran relevancia. De hecho, el esfuerzo destinado a la verificación a menudo supera el dedicado al propio diseño del sistema. La metodología UVM, que emplea agentes de verificación, ofrece una solución robusta y reutilizable para estos desafíos. Los agentes en UVM no solo facilitan una verificación más detallada, sino que también permiten la creación de

herramientas de verificación que son fácilmente replicables y mantenibles. Cada agente encapsula características específicas que requieren monitoreo, proporcionando una visibilidad más clara de los resultados obtenidos.

En este trabajo se ha demostrado la utilidad del entorno UVM descrito mediante agentes, aplicando esta metodología a la creación de un entorno de verificación para sistemas más complejos, como una máquina de estados finitos (FSM). La incorporación de múltiples agentes ha permitido una verificación más exhaustiva de las características críticas de la FSM, mejorando la eficacia del proceso de verificación en sistemas digitales complejos.

Agradecimientos

Agradecimientos al proyecto SIP20240036 aprobado por la Secretaría de Investigación y Posgrado del IPN.

Referencias

Bergeron, J., (2006). Writing testbenches using systemverilog. Springer. USA.

Bhuvaneswary N., Denny J., (2022). Exploration on reusability of universal verification methodology. 2d International Conference on Advanced Computing and Innovative Technology in Engineering (ICACITE).

Coopper, V., (2023). Getting Started with UVMT. Verilab Publishing, TX USA.

FUMEC, USAID, (2024). Mapa de Ruta: Oportunidades para el Nearshofing de Semiconductores en México. México.

Gayathri M., Rini, S., (2026). A SV-UVM framework for Verification of SGMII IP Core with reusable AXI to WC Bridge UVC.

Hosny, S., (2022). A Unified UVM Methodology For MPSoC Hardware/Software Functional Verification. 11th International Conference on Modern Circuits and Systems Technologies (MOCAST).

Mehta, A., (2020). Systemverilog assertions and functional coverage, Springer. USA.

Rodriguez, S. (2018). Entorno UVM para la verificación funcional de un IP multi-interfaz orientado a la compresión de imágenes. [Tesis de Maestría, Universidad de las Palmas de Gran Canaria]. Repositorio.

Spear, C., Tumbush G., (2012). Systemverilog for verification. 3ed Ed. Springer. USA.

Vaca, P., (2017). Diseño de un banco de test con metodología UVM (Universal Verification Methodology). [Tesis de Ingeniería, Universidad Politécnica de Madrid]. Repositorio.

Apéndice A. Primer Apéndice

Los ambientes de verificación descritos en el presente trabajo se pueden consultar en el siguiente repositorio:

Código UVM-MUX
Código UVM-FSM