Ambiente de pruebas para verificación y cobertura funcional con metodología UVM.

S. Ortega Cisneros[(1)], J.J. Raygoza Panduro[(2)], F. Sandoval Ibarra[(1)], A. Pedroza de la Cruz[(1)], M. Carrazco Díaz[(1)].

***Resumen*—Actualmente, con el incremento en el tamaño y complejidad de los sistemas digitales, es necesario invertir gran cantidad de recursos y tiempo para asumir con certeza que el comportamiento del diseño esacorde al deseado, por lo que se vuelve de suma importancia seguir una metodología de verificación optimizada para poder asegurar la mayor confiabilidad del diseño en el menor tiempo posible. En base a esta problemática se han implementado a lo largo del tiempo diferentes metodologías para conseguir el óptimo funcionamiento del diseño, sin embargo una gran desventaja que se ha observado es que las camas de prueba por lo general son incompatibles con nuevos diseños. Una de las técnicas más recientes es la metodología de verificación universal (UVM) que proporciona una biblioteca de clases, macros y funciones, las cuales facilitan el diseño del ambiente de verificación. Este artículo propone un procedimiento sistemático, estructurado y funcional para la verificación de diseños de sistemas digitales complejos. Para mostrar detalladamente la estructura de verificación propuesta se realiza la verificación funcional de un transmisor – receptor universal asíncrono (UART). Se muestran los resultados de los porcentajes de cobertura funcional y de código.**

Palabras clave—**UVM, Metodología de Verificación, Cobertura Funcional, SystemVerilog, Diseño Digital, Simulación.**

I. Introducción

EL incesante aumento en la complejidad, la cantidad de funciones y el número de transistores en los circuitos integrados, provoca un reto cada vez más grande para la verificación de estos. Un mayor número de circuitos en un chip significa una mayor probabilidad de errores. Cada vez los sistemas digitales contienen una mayor cantidad de elementos lógicos por lo que se torna muy complejo probar todos y cada uno de los estados lógicos posibles y en algunos casos es prácticamente imposible, ya que los tiempos requeridos para realizar ese tipo de simulaciones y verificar el correcto funcionamiento del circuito se disparan exponencialmente, por esta razón se ha optado por aplicar la verificación funcional. La verificación funcional se basa en la cobertura de los puntos más significativos o de mayor interés en el diseño de un sistema digital, con los cuales se puede garantizar la correcta funcionalidad de todo el sistema. Este incremento en la complejidad de los sistemas digitales y el deseo de lograr un menor tiempo de acceso al mercado, ha aumentado la importancia del proceso de verificación y ha provocado una acelerada evolución de las herramientas destinadas a este fin. Los errores no detectados y no reparados en un circuito integrado, antes de que este sea distribuido o empleado, pueden provocar toda clase de problemas, desde un píxel incorrecto en una pantalla de un videojuego, hasta la caída en el funcionamiento de un servidor o el sistema de emergencia de un establecimiento.

La verificación es el proceso inverso al diseño, toma una implementación y confirma si ésta cumple con las especificaciones. Por tanto, en cada paso del diseño debe existir la verificación correspondiente para asegurar la integridad del diseño descrito a diferentes niveles de abstracción durante el proceso de construcción.

La figura 1 muestra el diagrama de flujo para la fabricación de un circuito integrado.

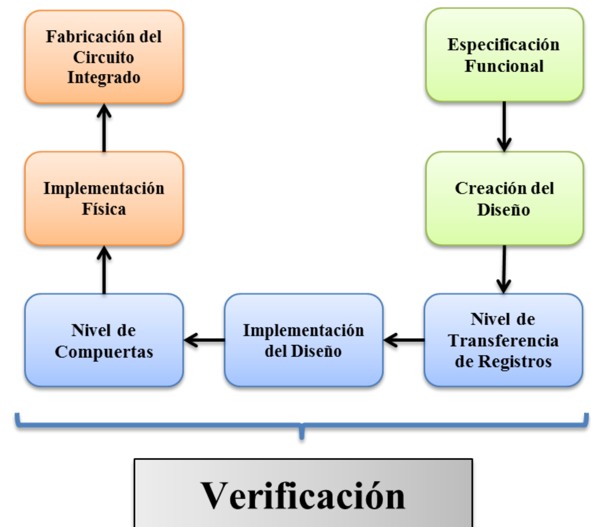

Fig. 1. Etapas del proceso de fabricación de un circuito integrado

El ambiente de verificación es el conjunto de software y herramientas que permiten al verificador encontrar los errores en el diseño. El código de software suele ser específico para cada diseño, mientras que las herramientas son más genéricas y por tanto utilizadas en varios proyectos de verificación.

Existen muchos tipos de ambientes relacionados con las diferentes metodologías de verificación, entre ellos: determinísticos, aleatorios, formales y generadores de casos de prueba. Cada uno de estos tiene distintas formas de crear los estímulos y revisar la respuesta del dispositivo bajo verificación (DUV).

II. Área detrabajo autogenerada para el ambiente de verificación

Cuando se realiza un diseño digital es importante buscar la manera más efectiva y rápida de realizar la

(1) Centro de Investigación y de Estudios Avanzados del I.P.N., Unidad Guadalajara. sortega@gdl.cinvestav.mx
(2) Departamento de Electrónica, CUCEI, Universidad de Guadalajara juan.raygoza@cucei.udg.mx

verificación del circuito, sobre todo cuando se trata de un diseño extremadamente extenso.

En este artículo se propone una estructura fundamental que agiliza el proceso de verificación. Esta consiste en el uso de un archivo de configuración *bash*, *makefiles* y una serie de directorios que componen el ambiente de verificación.La estructura es capaz de contener varios proyectos diferentes sin tener que preparar una nueva estructura para cada uno. También de esta manera se facilita la verificación por bloques de un mismo proyecto si se requiere.

El archivo de configuración *bash* contiene las variables de entorno necesarias para cada proyecto, además especifica la herramienta que se usara para simulación, cobertura y verificación. Los *makefiles* son utilizados para crear la estructura automáticamente, compilar el diseño, correr simulaciones con la cama de prueba y reportar la cobertura funcional del proyecto.

En la figura 2 se muestra la estructura que generan los *makefiles* para realizar la verificación, en la que se observa un archivo principal en el directorio raíz que a su vez llama a otros *makefiles* dentro del directorio "bin" que contienen los comandos de compilación y simulación adecuados por cada proyecto. También en el directorio "bin" se encuentra el archivo de configuración *bash*.

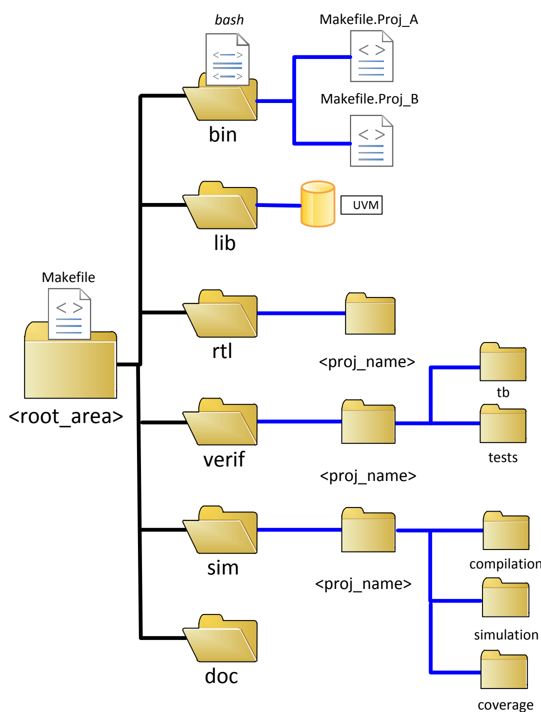

Fig. 2. Estructura de directorios del ambiente de verificación

En el directorio "rtl" se encuentra un subdirectorio por proyecto por cada diseño a verificar y una lista de archivos a compilar.

La cama de pruebas se encuentra en el directorio "verif", donde también hay una lista de compilación por proyecto.

Las bibliotecas utilizadas para la compilación o verificación se encuentran en el directorio "lib", donde se tiene que ubicar la biblioteca de la metodología UVM.

Algo que no debe faltar en un ambiente de verificación es la documentación tanto del diseño como de la cama de prueba, esta se localiza en el directorio "doc". Es esencial documentar el como hacer uso de la cama de prueba para poder llevar un control y facilitar la reutilización de bloques específicos de la cama de prueba en algún otro proyecto que lo requiera.

Por último en el directorio "sim" se encuentran las compilaciones, las simulaciones, la cobertura y los reportes en un subdirectorio por cada proyecto.

El uso de esta estructura facilita la elaboración del ambiente de pruebas y reduce el tiempo que toma la verificación de un diseño, debido a la organización y al control de la documentación.

III. VERIFICACIÓN FUNCIONAL CON SYSTEMVERILOG

Hoy en día, con la disponibilidad de sintetizadores confiables y la aplicación de técnicas de diseño síncronas, el menor nivel de detalle que los diseñadores deben considerar es el nivel de transferencia de registros RTL (*Register Transfer Level*). Como su nombre lo indica, los elementos principales en un diseño representado a éste nivel son: registros, interconexiones entre registros, y el cálculo necesario para modificar sus valores. Dado que cada registro recibe nuevos valores sólo con los pulsos de reloj, toda la lógica combinatoria necesaria para calcular el valor del registro, puede ser abstraída como un conjunto de expresiones en álgebra Booleana.

La verificación funcional de un diseño significa comparar: la intención del diseñador contra el comportamiento observado, para determinar su equivalencia. Se considera un diseño verificado satisfactoriamente, cuando cumple con todas las especificaciones de acuerdo a los requerimientos de diseño. Para afirmar que un diseño funciona, se debe comparar contra un modelo de referencia conocido; que represente el propósito del diseñador. Cada banco de prueba tiene un modelo de referencia y un medio para comparar la función del diseño de referencia.

La verificación funcional del diseño se refiere a la validación de la descripción inicial, para asegurar que el diseño realiza las tareas como lo estipula la arquitectura. La verificación predominantemente se realiza a nivel de lógica de transferencia de registros (RTL), donde la simulación funcional es la principal técnica para validarla [1].

El lenguaje de descripción de hardware System Verilog permite el uso de un lenguaje unificado para la descripción abstracta y detallada de un diseño, la especificación de las aserciones, la cobertura y el banco de pruebas de verificación basadas en metodologías manuales o automáticas. System Verilog ofrece métodos que permiten a los diseñadores seguir utilizando lenguajes actuales de diseño cuando sea necesario para aprovecharlos diseños existentes y los de propiedad intelectual [2].

IV. METODOLOGÍA UVM

Un ambiente de verificación UVM está compuesto por componentes reusables de verificación. Cada componente sigue una arquitectura y consiste de un

conjunto de elementos para simulación, comprobación, y recolección de información de cobertura [3].

Un agente es una parte primordial del ambiente de verificación, este se compone de un *Sequencer*, un *Driver* y un *Monitor*, además la configuración puede ser dinámica según requiera para cada prueba, permitiendo activar o desactivar componentes. Los elementos mínimos necesarios son:

- Sequencer
- Driver
- Monitor

La figura 3 muestra el diagrama de conexión de un agente.

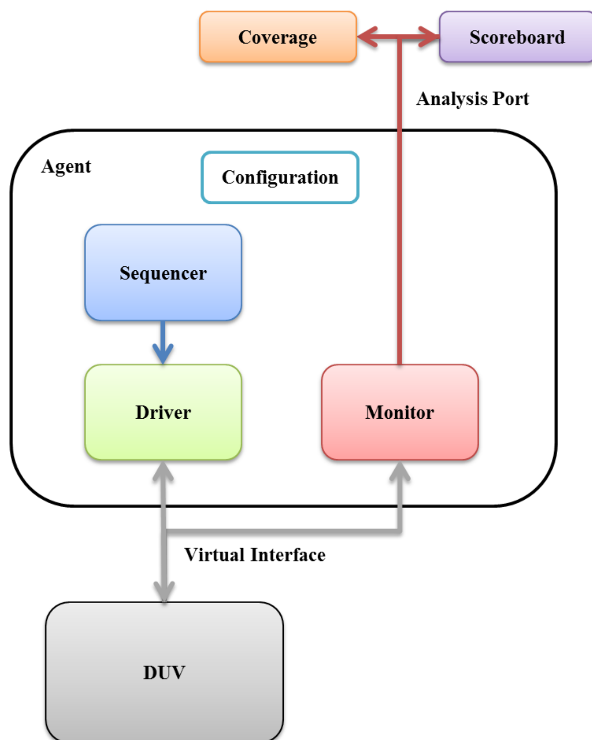

Fig. 3. Diagrama a bloques de un agente

Los componentes manejan paquetes de datos. Los paquetes son objetos de transacción, usados para estimular el dispositivo bajo verificación (DUV).

El *Sequencer* se encarga de controlar el envío de paquetes de datos hacia el driver, genera estímulos de datos según el tipo de paquete que maneje y su funcionamiento depende de la secuencia que el agente le configure.

El *Driver* recibe los paquetes de datos provenientes del *Sequencer* para inyectarlos hacia el DUV, ya que tiene conexión directa con la interfaz virtual conoce el protocolo de comunicación con el DUV.

El *Monitor* es el responsable de extraer información y el estado de la interfaz virtual a base de eventos, también puede recolectar y enviar información para análisis de cobertura y comprobación por medio de los puertos de análisis hacia otros componentes como el *Coverage* y el *Scoreboard*.

El agente interconecta los componentes mencionados y los configura con el comportamiento deseado, también puede tener puertos de análisis que envían información capturada de la interfaz,hacia el *Coverage*y al *Scoreboard.*

El colector de cobertura (Coverage) se encarga de contar y organizar los eventos que recibe mientras se está ejecutando una prueba, así como de generar las bases de datos de donde se analiza la cobertura funcional del dispositivo bajo prueba. Como resultado del análisis de cobertura se puede saber que partes han sido ejecutadas en el DUV.

Para concluir que una prueba ha sido satisfactoria o no, se requiere un *Scoreboard*, también se da a la tarea de cotejar el comportamiento deseado del DUV mientras una prueba está en ejecución e informa si ha ocurrido algún error o comportamiento inesperado.

El ambiente es el componente principal de la cama de prueba, contiene uno o más agentes y componentes adicionales como monitores, colectores de cobertura y un *Scoreboard*. En el ambiente se configura la funcionalidad de los agentes, también se puede habilitar o deshabilitarcomponentes para cada prueba.

V. Cobertura Funcional

La cobertura funcional es un punto importante dentro de la verificación funcional, ya que indica si las pruebas realizadas estimulan completamente las partes que se consideran importantes o críticas del diseño [4].

La cobertura funcional indica el porcentaje de utilización de las capacidades de algún bloque en específico o de todo el circuito, con los reportes generados se conoce si algún bloque del circuito precisa de más pruebas específicas para verificar toda su funcionalidad [5].

Para realizar una cobertura funcional es necesario primero definir y documentar la funcionalidad adecuada del circuito, después en base a ésta, crear una lista con los puntos de cobertura que se desean estimular en el circuito, estos puntos pueden ser por ejemplo:bloques del circuito que realizan escrituras, lecturas, o ciertas direcciones de memoria.

En el diagrama de la figura 4 se observa que cada bloque interno del circuito RTL a verificar, tiene un grupo de cobertura definido y cada grupo de cobertura tiene diferentes puntos que verifican que algún tipo de transacción o estímulo suceda en los bloques; por ejemplo una lectura.

Al final, se genera un reporte con los datos obtenidos en cada prueba realizada. El reporte muestra cuáles de los grupos de cobertura se cumplieron al cien por ciento y cuales no. Si la cobertura funcional no se cumple es necesario agregar nuevas pruebas para estimular los bloques completamente.

Para validar el circuito no es suficiente con que se cumplan los puntos de cobertura, también es necesario que el *Scoreboard* de UVM reporte que la prueba arroja los resultados adecuados y no encuentre ningún error.

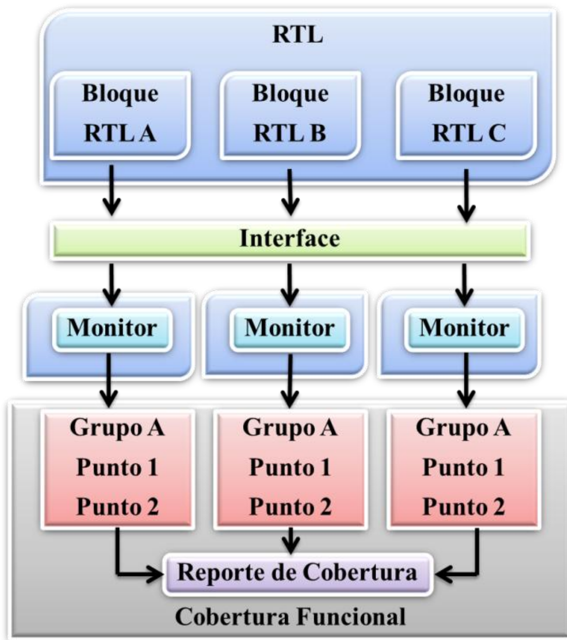

Fig. 4. Diagrama a bloques de la cobertura funcional

VI. MODELO DE AMBIENTE DE VERIFICACIÓN REUTILIZABLE

Este artículo aporta un modelo reutilizable de un ambiente de verificación que sirve como base para cualquier circuito digital que se pretenda implementar además de un método lógico y una estructura de directorios escalable. La figura 5 muestra el ambiente de verificación estandarizado.

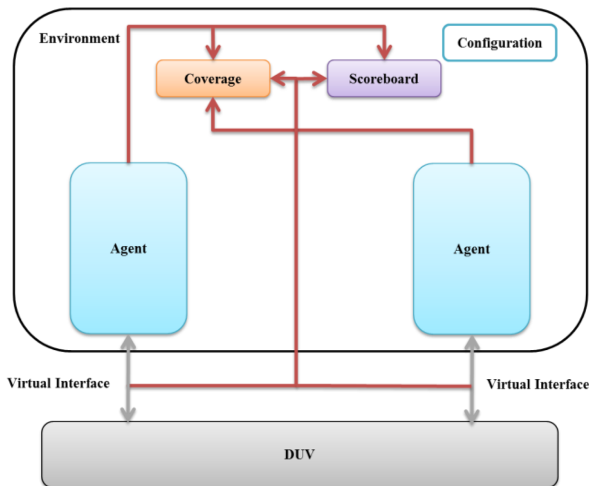

Fig. 5. Ambiente modelo de verificación.

La primera etapa que se debe modificar para un nuevo diseño es la interfaz virtual, donde se deben agregar las conexiones para cada puerto que se conecta con el dispositivo bajo verificación. En el caso que las interfaces sean iguales se puede hacer de manera paramétrica.

Posteriormente en los Drivers se agrega la lógica temporal para manejar la interfaz virtual a la que están conectados. El Monitor conecta las señales que se pretenden evaluar, para optimización del tiempo de prueba el Monitor no debe de capturar datos cada ciclo de reloj sino cuando suceda un evento esperado. El *Sequencer* está conectado al Driver por medio de un puerto de nivel de transacción, el cual no conoce el reloj del sistema solo asigna la secuencia que debe seguir el driver.

El Agente conecta el Driver, el Monitor y el *Sequencer*. El ambiente se encarga de crear los agentes que sean requeridos, en el caso que las interfaces sean iguales se sugiere que se asignen parámetros para evitar código redundante. En adición se configura el *Coverage* según los criterios que sean requeridos para garantizar la verificación funcional del diseño.

En el *Scoreboard* se configuran todos los criterios que deben de cumplir los protocolos de comunicación con la interfaz. El *Coverage* y el *Scoreboard* se conectan en el ambiente por medio de los puertos de análisis de los Monitores, para recibir las transacciones capturadas por los Monitores de los agentes.

Por último el ambiente debe ser configurado por la prueba a realizar, ya que cada prueba tiene la capacidad de asignar secuencias a cada uno de los *Sequencers*, habilitar las características de cada agente, y los componentes que van a funcionar.

Es altamente recomendable que las pruebas sean lo mas aleatorias posibles ya que los datos que manejan los circuitos por lo general no son muy predecibles, solo en casos especiales se deben hacer pruebas dirigidas.

VII. PRUEBA DE VERIFICACIÓN

Utilizando el modelo del ambiente de verificación propuesto, la estructura de directorios y los *makefiles,* se realizo la verificación de un circuito UART con una cama de pruebas estándar de UVM. El ambiente de verificación incluye ambiente, agentes, *Drivers*, *Sequencers*, *Monitores*, *Coverage* y un *Scoreboard* que indica si los datos suministrados en el emisor son capturados correctamente por el receptor[6]. En la figura 6 se muestra la cama de prueba utilizada para la verificación funcional, la cual reutilizo el código base del modelo de verificación propuesto.

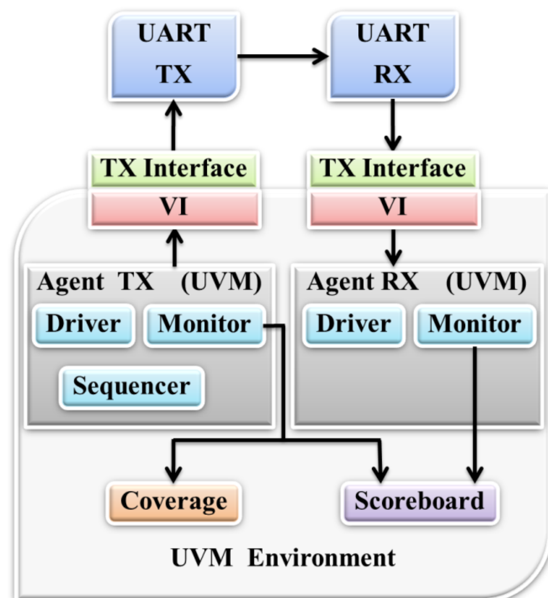

Fig. 6. Ambiente de verificación

Además se establece un punto de cobertura funcional, el cual se da a la tarea de revisar que el circuito transfiera todas las letras minúsculas, mayúsculas y cinco diferentes símbolos del código ASCII. De igual manera se realizaron tres pruebas individuales; una estimula el circuito con letras minúsculas, la segunda prueba con letras mayúsculas y una tercera transfiere los cinco símbolos.

En la figura 7 se muestra un diagrama a bloques del flujo que se debe seguir para configurar del ambiente de verificacióndel UART.

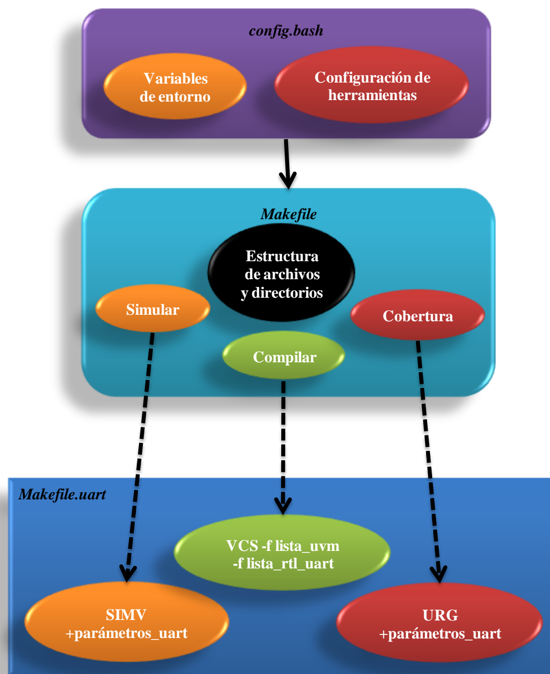

Fig. 7. Configuración del ambiente de verificación

Primero con el archivo *config.bash* se deben configurar las variables de entorno y las herramientas de simulación a utilizar. La herramienta de simulación utilizada es VCS de Synopsys® [7].

Después el *makefile* principal se encarga de crear el sistema de directorios del proyecto UART. En seguida se debe de mover la biblioteca de UVM [8] en el directorio "lib", así como el RTL y el ambiente de verificación en "rtl/uart" y "verif/uart" respectivamente.

Por último la compilación, simulación y cobertura del ambiente de verificación se ejecuta por medio del *makefile* principal que a su vez llama al *makefile.uart,* el cual contiene los comandos y argumentos necesarios únicamente para el proyecto UART. Si se requiere configurar otro proyecto de verificación en la misma área de trabajo que la del UART, solo es necesario agregar un *makefile.proyecto* con la configuración adecuada. En este caso para elegir el proyecto utilizar se debe de indicar en el *config.bash*.

El proceso de la figura 7 se puede resumir en los siguientes comandos.

- source *bin/config.bash* uart
- make directorios
- make compilación
- make simulación
- make cobertura

Este procedimiento agiliza el proceso debido a que nos ahorra el tener que escribir el comando completo cada vez que se quiera compilar o simular. También se puede crear dentro del *makefile* una instrucción que realice todos los paso de una vez sin tener que teclear todas las instrucciones, por ejemplo "make all".

Al final de la compilación y simulación del proyecto UART se puede observar los resultados generados de la simulación por el *scoreboard* y la cobertura en el directorio "sim". La herramienta de Synopsys que obtiene la cobertura funcional del proyecto es URG. Esta herramienta analizalos directorios donde se guardaron las bases de datos de cobertura de cada una de las pruebas y genera un reporte de la cobertura funcional y de la cobertura de código.

VIII. RESULTADOS

La estructura de trabajo se configura y crea en un máximo de 5 minutos gracias a la ayuda de los archivos *bash* y *makefiles*. Tomando como referencia el modelo del ambiente de verificación propuesto con las clases y métodos base, se invirtieron 8 horas en la implementación de las interfaces, pruebas, la configuración de agentes y el ambiente.

La figura 8 muestra la transferencia serial entre el transmisor (*tx_data*) y el receptor (*rx_data*), con la señal *ready* se indica el inicio de la transmisión y la señal *rdrf* se indica cuando ha terminado la transferencia. Los caracteres han sido generados de manera aleatoria.

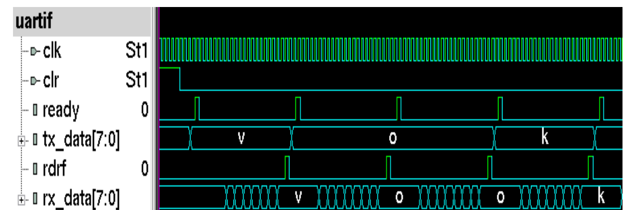

Fig. 8. Simulación obtenida de la prueba test_minus.

En la cobertura de código no es obligatorio obtenerel 100%, ya que muchos estados llegan a ser inaccesibles, algunas líneas no conmutan y no siempre es necesario entrar a todas las bifurcaciones debido a las restricciones del diseño, sin embargo la cobertura funcional debe de cumplirse ya que son los puntos de interés requeridos para poder asegurar que la funcionalidad es correcta.

Como resultado de las pruebas, para una simulación de 10000 paquetes aleatorios, que se transfieren entre receptor y emisor, se obtuvo la cobertura funcional y de código que se muestra en la figura 9 donde el U0 es el módulo transmisor, el U2 es el receptor y el duv es el módulo principal que contiene a ambos.

La cobertura de líneas de código RTL (*Lines*), las conmutaciones en los puertos (*Toogle*), los estados y transiciones cubiertos en las máquinas de estados (*FSM*), las bifurcaciones visitadas (*Branch*) y el promedio de los anteriores (*Score*), se muestran en la figura 9.

SCORE	LINE	TOGGLE	FSM	BRANCH	
75.31	90.67	68.92	66.67	75.00	duv

SCORE	LINE	TOGGLE	FSM	BRANCH	
77.68	94.74	67.50	66.67	81.82	U0
70.54	86.49	59.78	66.67	69.23	U2

Fig. 9. Cobertura de líneas, conmutación, máquinas de estado y bifurcaciones por módulo.

Para la cobertura por grupo, han sido declarados 57 contenedores de los cuales 26 son para letras mayúsculas, otros 26 para las minúsculas y 5 más para símbolos.

La figura 10 muestra la gráfica de distribución de contenedores definidos para la cobertura funcional.

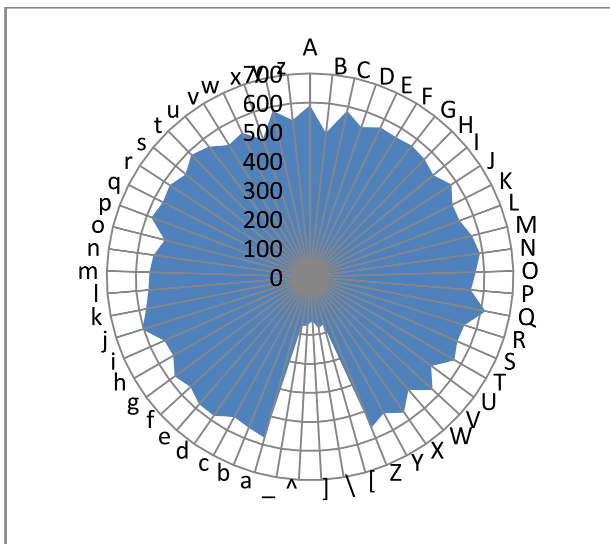

Fig. 10. Gráfica de distribución de caracteres por grupo de cobertura.

El *Scoreboard* analizó todos los paquetes transferidos por el UART y no encontró ningún error en los datos, por lo tanto pasó las tres pruebas de UVM con las que se verificó el circuito. La verificación del circuito resultó exitosa debido a la organización de la estructura de archivos y a la metodología de verificación UVM, que facilitó y agilizó el proceso.

IX. Conclusión

Se comprueba que al seguir el modelo propuesto se realiza la verificación de manera sistemática organizada y no se tiene que rediseñar el ambiente de verificación para agregar pruebas.

Sin utilizar el modelo propuesto, el tiempo de implementación de la cama de prueba puede tomar varios días y no permitir el reusó e incremento de pruebas. La estructura de directorios y los *makefiles* facilitan y reducen el tiempo que nos toma realizar la verificación funcional.

Los resultados demuestran que se cubren todos los puntos que se consideran como aspectos funcionales del diseño, además el ambiente de verificación es altamente reusable y sirve de base para adecuarlo a cualquier otro diseño a verificar.

X. Bibliografía

[1] Young-Jin Oh; Gi-Yong Song; "Simple hardware verification platform using SystemVerilog," *TENCON 2011 - 2011 IEEE Region 10 Conference*, vol., no., pp.1414-1417, 21-24 Nov. 2011
doi: 10.1109/TENCON.2011.6129042
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6129042&isnumber=6128995

[2] IEEE Computer Society, "IEEE Standard for System Verilog-Unified Hardware Design, Specification, and Verification Language," *IEEE STD 1800-2009*, vol., no., pp.C1-1285, 2009
doi: 10.1109/IEEESTD.2009.5354441
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5354441&isnumber=5354440

[3] Young-Nam Yun; Jae-Beom Kim; Nam-Do Kim; Byeong Min; , "Beyond UVM for practical SoC verification," *SoC Design Conference (ISOCC), 2011 International* , vol., no., pp.158-162, 17-18 Nov. 2011
doi: 10.1109/ISOCC.2011.6138671
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6138671&isnumber=6138611

[4] Aihong Yao; Jian Wu; Zhijun Zhang; , *"Functional Coverage Driven Verification for TAU-MVBC,"*Internet Computing for Science and Engineering *(ICICSE), 2010 Fifth International Conference on* , vol., no., pp.89-92, 1-2 Nov. 2010
doi: 10.1109/ICICSE.2010.13
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6076547&isnumber=6076519

[5] Mishra, P.; Dutt, N.; *"Functional coverage driven test generation for validation of pipelined processors,"Design, Automation and Test in Europe, 2005. Proceedings*, vol., no., pp. 678- 683 Vol. 2, 7-11 March 2005
doi: 10.1109/DATE.2005.162
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1395653&isnumber=30361

[6] Sudhish, N.; Raghavendra, B.R.; Yagain, H.; , *"An Efficient Method for Using Transaction Level Assertions in a Class Based Verification Environment,"*Electronic System Design (ISED), 2011 International Symposium on, vol., no., pp.72-76, 19-21 Dec. 2011
doi: 10.1109/ISED.2011.32
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6117329&isnumber=6117297

[7] Synopsys, Inc. *"High-performance RTL verification"*, ©2011 Synopsys, Inc.
URL: http://www.synopsys.com/Tools/Verification/FunctionalVerification/Pages/default.aspx

[8] Accellera Organization,"Standard Universal Verification Methodology Class Reference Manual, Release 1.1" *Copyright© 2011 Accellera*
URL: http://www.accellera.org/downloads/standards/uvm/UVM_1.1_Class_Reference_Final_06062011.pdf