Escuela de Ingeniería Electrónica
Licenciatura en Ingeniería Electrónica
EL-5811 Verificación Funcional de Circuitos Integrados
II Semestre 2023

TEC | Tecnológico de Costa Rica

Proyecto 1

Verificación funcional de un controlador de bus parametrizable

Alexander Castro Lara, Mario Montero Marenco

1. Introducción

En el campo de verificación funcional de circuitos integrados convergen diversas metodologías para llevar a cabo comprobaciones. Un estándar muy común y básico es la verificación aleatoria en capas, la cual, como muchas otras cosas, posee sus puntos fuertes y debilidades.

En el presente proyecto se planteará un ambiente de verificación basado en esta modalidad, el cual se usará para 'testear' un Dispositivo Bajo Pueba (DUT) que consiste en un BUS serial implementado en SystemVerilog. El bus en cuestión permite gestionar múltiples terminales de manera simultánea.

El ambiente que se desarrollará tendrá 3 niveles de abstracción (test, agente y driver), lo que permitirá comprender fácilmente la base de su funcionamiento. Finalmente, para cuantificar la validez de la verificación, se documentarán los resultados, mediante un archivo de salida '.csv', el cual contiene información importante como: el paquete que se envía, el que se recibe, latencia, ancho de banda, tipo de transacción, entre otros datos.

2. Metodología

En esta sección se incluyen los métodos y pasos que permiten realizar un ambiente de verificación funcional con aleatorización controlada en capas básico para comprobar un BUS serial, además de la decripción de todos los componentes que lo integran.

Primeramente, se sabe que los entornos de verificación pueden variar sus componentes y características de un diseño a otro. Por lo tanto, se debe determinar la estructura y la forma en que se

llevará a cabo la comunicación entre bloques. Esto permite estar bien orientado durante toda la implementación.

El esquema propuesto para este ambiente se muestra en la Fig. 1. Como se puede observar, están bien marcados los 3 niveles de abstracción: test, escenario y comando.

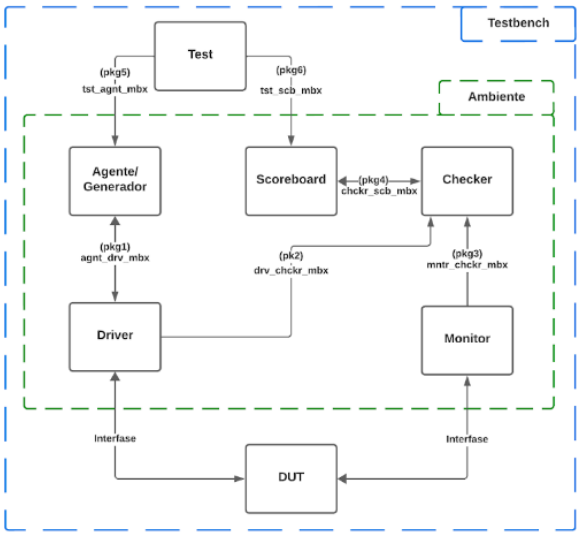

Fig. 1: Estructura del Ambiente de Verificación Funcional Aleatoria en Capas

El componente de menor nivel está constituido por el Driver y el Monitor, que son importantes porque constituyen la interface de conexión con el DUT, es decir, el driver proporciona las señales de excitación y el monitor recibe las salidas.

El Driver está constituido por 2 clases: 'driver_hijo' y 'driver_padre'. La clase 'driver_hijo' simula la FIFO de entrada de una única terminal, declara la interfaz de conexión con el BUS y prepara la señales de 'pop', 'pndng' y 'id' (terminal de destino), ver Fig. 2(a). También se crea un puntero a mailbox para la comunicación y se instancia un manejador de tipo 'trans_bus', que simula la interfaz de las transacciones, ver Fig. 2(b). La acción de funcionamiento 'normal' o 'activado' se describe en una tarea al final de la clase que está en constante obtención y transmisión de datos.

```
class driver_hijo #(parameter ...);

    bit [ancho-1:0] fifo_emul [$:profundidad];
    bit pop;     // señal de pop que viene desde
    bit pndng;   // señal de pending hacia el DU
    int id; // identificador de la FIFO de entr

    virtual bus_if #(...) vif; // interfaz vi
```

(a)

```
trans_bus_mbx drv_padre_mbx;
trans_bus #(...) transaction;
```

(b)

Fig. 2: Partes del 'driver_hijo'

La clase 'driver_padre' pretende instanciar y ser el controlador de todas las terminales o 'drivers hijos', lo que permite emular conexiones físicas reales e insertar las transacciones al BUS. Las otras

dos funciones que tiene son: simular la interfaz de conexión del bus y de las transacciones (Fig. 3(a)), y declarar los 'mailboxes' que utilizará el Driver para comunicarse con el ambiente; para este caso se establecen las comunicaciones agente-driver y driver-checker (Fig. 3(b)). Como en el caso de la clase 'driver_hijo', el comportamiento en tiempo de simulación se establece mediante un bloque 'task', que revisa el mailbox agente-driver, asigna la transacción a la FIFO de la terminal requerida, proporciona los retardos entre transacciones y captura el tiempo de envío.

```
virtual bus_if #(...) vif; //
trans_bus #(...)transaction;
```

(a)

```
//mailboxes
trans_bus_mbx drv_padre_mbx [terminales];
trans_bus_mbx agnt_drv_mbx;
trans_bus_mbx drv_chckr_mbx;
```

(b)

Fig. 3: Partes del 'driver_padre'

Para completar este nivel, tenemos el Monitor, que se construye de manera similar al Driver. Es una clase que emula la FIFO de salida del BUS, instancia un puntero para el mailbox y establece la interfaz virtual de conexión y de las transacciones. En la Figura 4 se puede observar su estructura. En la implementación, se utiliza un bloque 'task' para describir el comportamiento del Monitor en tiempo de simulación. Básicamente, en este bloque se capturan constantemente datos como los paquetes provenientes del BUS, el tiempo en que se extrae la transacción y el ID del terminal responsable.

```
class monitor #(...);
    bit [ancho-1:0] fifo_emul [$:profundidad];
    virtual bus_if #(...) vif;

    trans_bus_mbx mnt_chckr_mbx;
    trans_bus #(...)transaction;
```

Fig. 4: Estructura del monitor

La segunda capa de abstracción es la de escenario, compuesta por el Agente, el Scoreboard y el Checker. Esta capa genera el entorno que permite realizar pruebas específicas, aleatorias, generales y de esquina. Además, registra todos los eventos en el Scoreboard y el Checker se asegura de que los resultados sean correctos.

El Agente sigue una estructura similar a los componentes de la capa anterior. La diferencia radica en que aquí se define el vector de prueba y se introduce en el mailbox 'agente-driver' para su posterior ejecución. En su bloque 'task', implementa una especie de 'case' para realizar acciones específicas para cada tipo de prueba. La comunicación se lleva a cabo mediante los mailboxes 'test-agente' y 'agente-driver'. Ver Fig. 5

Por su parte, el Scoreboard es un conjunto de arreglos de tipo 'trans_sb' (clase definida en 'interface_transaction'). Captura transacciones provenientes o ejecutadas desde el mailbox 'driver-scoreboard' y registra sus resultados. Los mailboxes del Scoreboard son 'driver-scoreboard' y

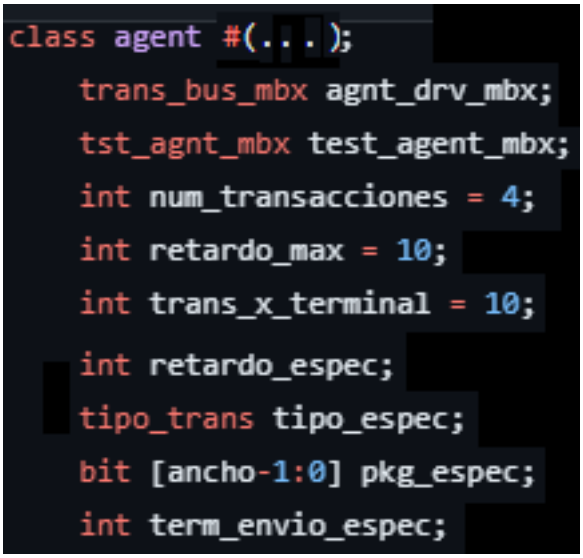

Fig. 5: Estructura del agente

'test-scoreboard' (Fig. 6). La sección más destacada del Scoreboard se encuentra en el bloque 'task'. Aquí, se realiza un seguimiento de las transacciones completadas, se calcula estadísticas como el retardo promedio, se generan informes de transacciones completadas y se mantiene un registro de las transacciones incompletas. Además, se gestiona el ancho de banda y se genera un informe relacionado con el rendimiento del sistema.

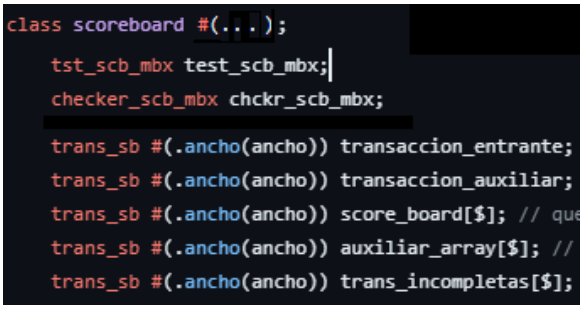

Fig. 6: Estructura del scoreboard

El Checker comienza obteniendo la transacción del Driver, registra también la del Monitor y las compara con las transacciones almacenadas en las colas correspondientes para verificar su validez. Calcula la latencia de las transacciones completadas y genera los informes que permiten hacer un análisis posterior (Fig. 7(a)). La comunicación del Checker se realiza a través de tres mailboxes: 'driver-checker', 'monitor-checker' y 'checker-scoreboard' (Fig. 7(b)).

En la capa de mayor nivel, se encuentra la capa de test, implementada en 'Test.sv'. El objetivo principal de este componente es inicializar los mailboxes 'test-agente' y 'test-scoreboard', instanciar el ambiente y configurar los parámetros clave para las pruebas (Fig. 8(a)). A continuación, se ejecuta una serie de pruebas secuenciales que incluyen transacciones aleatorias, transacciones específicas, secuencias de transacciones aleatorias, transmisiones de broadcast y otras pruebas específicas (como se muestra en la Fig. 8(b)).

La creación del ambiente implica la instanciación y la interconexión de los manejadores de las

```
if (from_mntr.paquete == auxiliar.paquete) begin
    to_sb.paquete_enviado = auxiliar.paquete;
    to_sb.paquete_recibido = from_mntr.paquete;
    to_sb.tiempo_envio = auxiliar.tiempo_envio;
    to_sb.tiempo_recibido = from_mntr.tiempo_recibido;
    to_sb.term_tx = auxiliar.terminal_envio;
    to_sb.term_rx = from_mntr.terminal_recibido;
    to_sb.completado = 1;
    to_sb.calc_latencia();
    `ifdef DEBUG
    to_sb.print("Checker: Transacción Completada");
    `endif
    chckr_scb_mbx.put(to_sb);
    queue[from_mntr.terminal_recibido].delete(j);
    break;
```

(a) Instancias iniciales

```
trans_bus_mbx drv_chckr_mbx;
trans_bus_mbx mnt_chckr_mbx;
checker_scb_mbx chckr_scb_mbx;
```

(b) Mailboxes

Fig. 7: Checker

```
class test #(,,,);

    tst_agnt_mbx test_agent_mbx;
    tst_scb_mbx test_scb_mbx;

    parameter num_transacciones = 100;
    parameter retardo_max = 10;
    parameter trans_x_terminal = 5;
    instruccion_agente instr_agent;
    reporte_scb instr_scb;
    ambiente #(,,,) ambiente_inst;
```

(a) Inicialización

```
ambiente_inst.agent_inst.term_envio_espec = 3;
test_agent_mbx.put(instr_agent);
$display("[%g]  Test: Enviada la segunda instruccion al agente

instr_agent = sec_trans_aleatorias;
test_agent_mbx.put(instr_agent);
$display("[%g]  Test: Enviada la tercera instruccion al agente

instr_agent = brdcst; // primer broadcast generado a "la fuerz
test_agent_mbx.put(instr_agent);
$display("[%g]  Test: Enviada la cuarta instruccion al agente:

instr_agent = trans_aleat_x_terminal;
ambiente_inst.agent_inst.trans_x_terminal = trans_x_terminal;
test_agent_mbx.put(instr_agent);
$display("[%g]  Test: Enviada la quinta instruccion al agente:
```

(b) Ejemplo de pruebas

Fig. 8: Test

clases de todos los componentes y sus respectivos mailboxes (Fig. 9(a)). En su bloque de ejecución principal, todos los 'task.run' se ejecutan para simular su funcionamiento en tiempo de simulación, lo que da la impresión de que todo funciona en paralelo (Fig. 9(b)).

El 'interface_transaction' se utiliza para declarar las clases 'trans_bus' y 'trans_sb'. Estas clases emulan la transmisión de información en el ambiente y hacia el scoreboard. También registran todos los datos relevantes (Fig. 10(a)). Otra función es crear los mailboxes que se utilizan en todo el ambiente y proporcionar una interfaz para conectar las señales con el DUT (Fig. 10(b)).

```
class ambiente #(   );
    agent #(    ) agent_inst;
    driver_padre #(    )
    driver_hijo #(    ) driver_h_inst [terminales];
    monitor #(    ) monitor_inst [terminales];
    scoreboard #(    ) scoreboard_inst;
    checker #(    ) checker_inst;

    trans_bus_mbx agnt_drv_mbx;
    trans_bus_mbx drv_chckr_mbx;
    trans_bus_mbx drv_padre_mbx [terminales];
    trans_bus_mbx mnt_chckr_mbx;
    checker_scb_mbx chckr_scb_mbx;
    tst_agnt_mbx test_agent_mbx;
    tst_scb_mbx test_scb_mbx;
```

(a) Instanciación

```
fork // se corren los componentes en paralelo
    agent_inst.run();
    driver_inst.run();
    checker_inst.run();
    scoreboard_inst.run();
    for (int j = 0; j < terminales; j++) begin
        fork
            automatic int n = j;
            driver_h_inst[n].run();
            monitor_inst[n].run();
        join_none
    end
join_none
```

(b) Componentes ejecutados en paralelo

Fig. 9: Ambiente

```
class trans_bus #(   );
    rand bit [ancho-1:0] paquete;
    rand int retardo;
    rand tipo_trans tipo;
    ...

class trans_sb #(parameter ancho = 16);
    bit [ancho-1:0] paquete_enviado;
    bit [ancho-1:0] paquete_recibido;
    int tiempo_envio;
    int tiempo_recibido;
    ...
```

(a) 'trans_bus' y 'trans_sb'

```
typedef mailbox #(      ) trans_bus_mbx;
typedef mailbox #(      ) checker_scb_mbx;
typedef mailbox #(      ) tst_agnt_mbx;
typedef mailbox #(      ) tst_scb_mbx;

interface bus_if #(
    parameter bits = 1,
    parameter drvrs = 4,

    )(
    input bit clk
    );

    logic   reset;
    logic   pndng[bits-1:0][drvrs-1:0];
```

(b) Mailboxes e interfaz

Fig. 10: Ambiente

El testplan también permite la programación de pruebas específicas y aleatorias que abarcan los casos generales. Estas pruebas están diseñadas para evaluar el funcionamiento general o "normal"del DUT. La aleatorización en las pruebas garantiza la exploración de múltiples escenarios, que incluyen diferentes números de transacciones por terminal, profundidades de las FIFOs de entrada, tiempos de envío de mensajes y otros parámetros.

Luego, se generan los 'set up' para la implementación de casos de esquina. Estos casos representan situaciones extremas que ponen a prueba los límites del sistema, incluyendo escenarios con mensajes de error y cargas de trabajo intensas.

Como se mencionó anteriormente, existen secciones destinadas a la documentación durante las pruebas, las cuales crean un archivo de salida en formato '.csv'.

Finalmente, se procede a la ejecución en el entorno de simulación 'Synopsys VCS'. Cada unidad del ambiente, como el agente, el scoreboard, el checker, el driver y el monitor, se ejecuta de manera independiente para evaluar su comportamiento individual y se ejecuta en conjunto para analizar su interacción con los otros componentes.

3. Indicaciones para correr la prueba

Para correr la prueba se dispone de dos formas: la primera es con un shell script llamado "comando.sh" y la segunda es manual por medio de comandos en la terminal. La diferencia entre ellas resalta en que la prueba con el script permite simular con una cantidad de terminales/drivers y profundidad de FIFO's aleatoria, en cambio, en la prueba manual se elige la cantidad de dispositivos que se quiere simular.

3.1. Utilizando el shell script

Para correr el script simplemente se debe utilizar el siguiente comando en el directorio en el que se encuentren los archivos fuente de la prueba:

```
#!/bin/bash
source comando.sh
```

Al realizarlo se le pedirá insertar un número de semilla. Esta semilla permite aleatorizar la prueba llamada "module_params.sv", la cual se encarga de generar un *package* ("parameters_pkg.sv") con distintos valores para la profundidad de las FIFO's y la cantidad de terminales del sistema, el cual se incluye en el Testbench del DUT para la posterior simulación. Cabe recalcar que la prueba del DUT **siempre** se corre con la misma semilla, la que ofrece por defecto el simulador.

3.2. Manual

Si por el contrario se desea realizar una prueba con una cantidad específica de dispositivos y una profundidad específica de las FIFO's se deben escribir los siguientes comandos:

```
source /mnt/vol_NFS_rh003/estudiantes/archivos_config/synopsys_tools.sh;
```

```
vcs -Mupdate Testbench.sv -o salida -full64 -sverilog  -kdb -lca
-debug_acc+all -debug_region+cell+encrypt -l log_test +lint=TFIPC-L
-cm line+tgl+cond+fsm+branch+assert;

./salida -cm line+tgl+cond+fsm+branch+assert;
```

Debe tenerse en cuenta que antes de correr dichos comandos debe cambiar los parámetros de cantidad de terminales y profundidad de las FIFO's desde el archivo "parameters_pkg.sv"para poder ver los resultados correctos en el Testbench. El identificador de broadcast y el ancho de la palabra se deben cambiar desde el archivo "Testbench.sv".

4. Resultados de las pruebas

Para la prueba se utilizó el script insertando la semilla 1, la cual genera 10 terminales con FIFO's de profundidad 31, con un identificador de broadcast de 0xff y un ancho de palabra de 32 bits. Las instrucciones generadas en el test se detallan a continuación:

4.1. Transacción aleatoria

Aquí se prueba el envío de una sola transacción con paquete (ID y payload), retardo y terminal de envío aleatorio. La Fig. 11 muestra la transacción creada por el agente, en tanto que en la Fig. 12 se observa que se recibió dicha transacción en el scoreboard desde el checker.

Fig. 11: Creación de la transacción aleatoria

Fig. 12: Recepción de la transacción aleatoria

4.2. Transacción hacia el mismo terminal

Para este escenario se prueba el envío de una transacción con un ID inválido, es decir, el terminal transmite un paquete cuyo contenido tiene como destino el mismo terminal. Este es un caso de esquina que si bien no tiene sentido, podría suceder dado el diseño del dispositivo. En la Fig. 13 se observa que se generó una transacción específica desde el Test, cuyo ID (0x03) coincide con el terminal de envío (0x3).

Fig. 13: Creación de la transacción específica

La Fig. 14 muestra que la transacción fue recibida desde en el scoreboard. Sin embargo, el estado de la transacción se marca con un cero, dado que nunca fue recibida por la FIFO de salida del terminal 0x3.

```
Scoreboard: Enviada Recibida Tiempo_Env Tiempo_Rcbd Term_Env Term_Rcbd Latencia Estado
[200300] SB_Report: dato_tx=0x03aaaaaa, dato_rx=0x00000000, t_tx=85, t_rx=0, term_tx=3, term_rx=0, ltncy=0, state=0
```

Fig. 14: Recepción de la transacción específica

4.3. Transacción hacia un terminal que no existe

Este caso de esquina se prueba con una transacción específica generada con los datos del Test. La transacción creada por el agente se muestra en la Fig. 15, mientras que en la Fig. 16 se puede ver que el DUT no envió la transacción por ninguno de los terminales disponibles.

```
[50]  Agente Instrucción: trans_especifica
[50] Agente: transacción creada Retardo=6 Tipo=envio paquete=0xfeaaaaaa Terminal_env=1 Tiempo_env=0 Terminal rcb=0 Tiempo_rcb=0
```

Fig. 15: Creación de la transacción específica hacia un terminal que no existe

```
[200300] SB_Report: dato_tx=0xfeaaaaaa, dato_rx=0x00000000, t_tx=9695, t_rx=0, term_tx=1, term_rx=0, ltncy=0, state=0
```

Fig. 16: Resultado de la transacción específica

4.4. Sección de transacciones aleatorias

En este caso general se generan una serie de 100 transacciones aleatorias en total. Las mismas pueden ser enviadas desde cualquier terminal y con cualquier tipo de retardo, el paquete puede incluir identificador de broadcast así como envíos hacia el mismo terminal. La Fig. 17 muestra una parte de las transacciones generadas y en la Fig. 18 se observan algunas de las transacciones completadas.

```
[15]  Agente Instrucción: sec_trans_aleatorias
[15] Agente: transacción creada Retardo=3 Tipo=envio paquete=0x0042ec9d Terminal_env=8 Tiempo_env=0 Terminal rcb=0 Tiempo_rcb=0
[15] Agente: transacción creada Retardo=5 Tipo=envio paquete=0x045c7ed3 Terminal_env=4 Tiempo_env=0 Terminal rcb=0 Tiempo_rcb=0
[15] Agente: transacción creada Retardo=6 Tipo=envio paquete=0x04aac058 Terminal_env=3 Tiempo_env=0 Terminal rcb=0 Tiempo_rcb=0
[15] Agente: transacción creada Retardo=8 Tipo=envio paquete=0x02d1250d Terminal_env=6 Tiempo_env=0 Terminal rcb=0 Tiempo_rcb=0
[15] Agente: transacción creada Retardo=3 Tipo=envio paquete=0x008066b3 Terminal_env=4 Tiempo_env=0 Terminal rcb=0 Tiempo_rcb=0
[15] Agente: transacción creada Retardo=6 Tipo=envio paquete=0x03ad8d7b Terminal_env=4 Tiempo_env=0 Terminal rcb=0 Tiempo_rcb=0
```

Fig. 17: Creación de una serie de 100 transacciones aleatorias

```
Scoreboard: Enviada Recibida Tiempo_Env Tiempo_Rcbd Term_Env Term_Rcbd Latencia Estado
[200100] SB_Report: dato_tx=0x0979b0a3, dato_rx=0x0979b0a3, t_tx=715, t_rx=1185, term_tx=5, term_rx=9, ltncy=470, state=1
[200100] SB_Report: dato_tx=0x02d1250d, dato_rx=0x02d1250d, t_tx=345, t_rx=1545, term_tx=6, term_rx=2, ltncy=1200, state=1
[200100] SB_Report: dato_tx=0x0677442f, dato_rx=0x0677442f, t_tx=35, t_rx=1905, term_tx=7, term_rx=6, ltncy=1870, state=1
[200100] SB_Report: dato_tx=0x0042ec9d, dato_rx=0x0042ec9d, t_tx=125, t_rx=2265, term_tx=8, term_rx=0, ltncy=2140, state=1
[200100] SB_Report: dato_tx=0x07c3cb70, dato_rx=0x07c3cb70, t_tx=545, t_rx=2625, term_tx=9, term_rx=7, ltncy=2080, state=1
```

Fig. 18: Algunas transacciones aleatorias completadas

4.5. Broadcast

El broadcast se evalúa de varias maneras en la prueba, la que se muestra en la Fig. 19 corresponde a un broadcast generado por medio de una transacción específica. En la Fig. 20 se muestra su correspondiente resultado, el cual es correcto, ya que todos los demás terminales reciben el paquete en el mismo instante de tiempo.

Fig. 19: Broadcast generado de manera específica

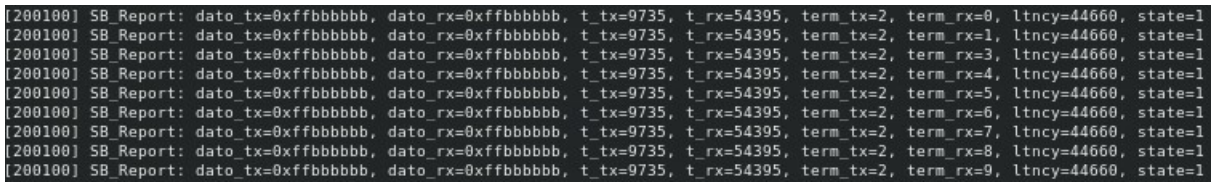

Fig. 20: Resultado del broadcast

4.6. Sección de transferencias aleatorias por terminal

Posteriormente, se ejecuta una serie de 5 transacciones aleatorias por terminal, lo que en este caso serían 50 transacciones aleatorias en total. La diferencia de esta con la sección de 100 transacciones aleatorias es que este escenario permite ver que todos los terminales son capaces de enviar y recibir datos a través de sus FIFO's de entrada y salida. Además de que en este escenario no se ejecutan envíos hacia el mismo terminal que creó la transacción.

4.7. Llenado de FIFO's de entrada

Básicamente, en este escenario se genera una cantidad de transacciones aleatorias por terminal igual a la profundidad de las FIFO's de entrada (para este caso: 10 terminales x 31 profundidad = 310 transacciones), lo que genera el mayor flujo de datos en el sistema. Dicho escenario permite calcular el retardo promedio general y por terminal del sistema, así como el cálculo del ancho de banda promedio, máximo y mínimo de la transmisión.

En la Fig. 21 se observa una parte del resultado de esta prueba, donde se puede además comprobar que el DUT va de terminal en terminal para enviar los paquetes.

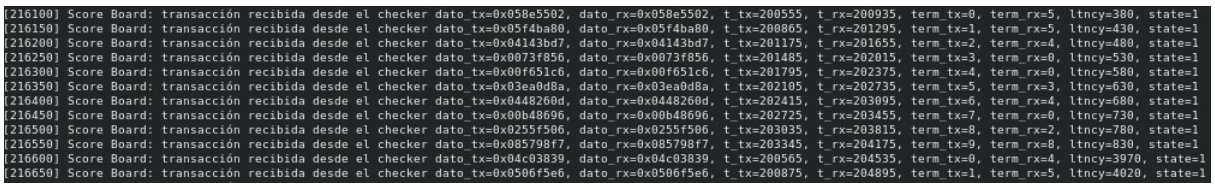

Fig. 21: Parte del resultado de llenar las FIFO's de entrada de los terminales del sistema

A partir de los resultados anteriores se pide al scoreboard un reporte de los retardos y anchos de banda. El scoreboard siempre imprime los resultados tanto en consola como en un archivo .csv para su posterior manipulación. En las Fig. 22 y 23 se muestran dichos reportes para el caso analizado.

```
Scoreboard: Retardo promedio
[700600] Scoreboard: el retardo promedio es: 54455.00 ns
[700600] Scoreboard: el retardo promedio en la terminal 0 es: 54175.00 ns
[700600] Scoreboard: el retardo promedio en la terminal 1 es: 60461.00 ns
[700600] Scoreboard: el retardo promedio en la terminal 2 es: 61978.00 ns
[700600] Scoreboard: el retardo promedio en la terminal 3 es: 51135.00 ns
[700600] Scoreboard: el retardo promedio en la terminal 4 es: 52532.00 ns
[700600] Scoreboard: el retardo promedio en la terminal 5 es: 48491.00 ns
[700600] Scoreboard: el retardo promedio en la terminal 6 es: 53763.00 ns
[700600] Scoreboard: el retardo promedio en la terminal 7 es: 49050.00 ns
[700600] Scoreboard: el retardo promedio en la terminal 8 es: 57836.00 ns
[700600] Scoreboard: el retardo promedio en la terminal 9 es: 56941.00 ns
```

Fig. 22: Reporte del retardo general y por terminal del sistema

```
Scoreboard: Instrucción Recibida desde el Test
Scoreboard: Reporte de ancho de banda
Ancho de banda máximo: 84210.00 kbps
Ancho de banda mínimo: 294.00 kbps
Ancho de banda promedio: 2892.23 kbps
```

Fig. 23: Reporte del ancho de banda promedio, máximo y mínimo del sistema

5. Bug Encontrado

Otra prueba se realizó con la misma cantidad de terminales, profundidad de las FIFO's y ancho de palabra. Sin embargo, se cambió el identificador de broadcast del sistema por un 0x00. Según se tiene entendido, el cambiar este parámetro a un número igual al identificador de una terminal debería hacer que cada transacción con dicho identificador haría broadcast a los demás dispositivos. Sin embargo, esto no sucede de esta forma porque, como se puede observar en la Fig. 24, el agente crea una transacción de broadcast (ID = 0x00), pero en la Fig. 25 se observa que el único terminal que recibe dicha información es el terminal 0.

```
[20]  Agente Instrucción: brdcst
[20] Agente: transacción creada Retardo=4 Tipo=envio paquete=0x003e025e Terminal_env=4 Tiempo_env=0 Terminal rcb=0 Tiempo_rcb=0
```

Fig. 24: Envío de un paquete con un nuevo broadcast (0x00).

```
[53575] Monitor hijo #0: Push desde el DUT
[53575] Monitor: Transacción Recibida Retardo=0 Tipo=envio paquete=0x003e025e Terminal_env=0 Tiempo_env=0 Terminal rcb=0 Tiempo_rcb=53575
```

Fig. 25: Recepción de un paquete con broadcast (0x00) solo desde la terminal 0.

Para confirmar que el broadcast no cambia, aunque se cambie el parámetro desde el testbench, se realizó una transacción específica con el identificador de broadcast por defecto (0xFF). En la Fig. 26 se puede ver una muestra de lo anterior. Mientras que en la Fig. 27 se observa que los demás terminales reciben este paquete, por lo que el checker imprime un mensaje de error y detiene la prueba.

```
[55]  Agente Instrucción: trans_especifica
[55] Agente: transacción creada Retardo=3 Tipo=envio paquete=0xff00000a Terminal_env=2 Tiempo_env=0 Terminal rcb=0 Tiempo_rcb=0
```

Fig. 26: Envío de un paquete con broadcast por defecto (0xFF).

```
[54395] Monitor hijo #0: Push desde el DUT
[54395] Monitor: Transacción Recibida Retardo=0 Tipo=envio paquete=0xff00000a Terminal_env=0 Tiempo_env=0 Terminal rcb=0 Tiempo_rcb=54395
[54395] Monitor hijo #1: Push desde el DUT
[54395] Monitor: Transacción Recibida Retardo=0 Tipo=envio paquete=0xff00000a Terminal_env=0 Tiempo_env=0 Terminal rcb=1 Tiempo_rcb=54395
[54395] Monitor hijo #3: Push desde el DUT
[54395] Monitor: Transacción Recibida Retardo=0 Tipo=envio paquete=0xff00000a Terminal_env=0 Tiempo_env=0 Terminal rcb=3 Tiempo_rcb=54395
[54395] Monitor hijo #4: Push desde el DUT
[54395] Monitor: Transacción Recibida Retardo=0 Tipo=envio paquete=0xff00000a Terminal_env=0 Tiempo_env=0 Terminal rcb=4 Tiempo_rcb=54395
[54395] Monitor hijo #5: Push desde el DUT
[54395] Monitor: Transacción Recibida Retardo=0 Tipo=envio paquete=0xff00000a Terminal_env=0 Tiempo_env=0 Terminal rcb=5 Tiempo_rcb=54395
[54395] Monitor hijo #6: Push desde el DUT
[54395] Monitor: Transacción Recibida Retardo=0 Tipo=envio paquete=0xff00000a Terminal_env=0 Tiempo_env=0 Terminal rcb=6 Tiempo_rcb=54395
[54395] Monitor hijo #7: Push desde el DUT
[54395] Monitor: Transacción Recibida Retardo=0 Tipo=envio paquete=0xff00000a Terminal_env=0 Tiempo_env=0 Terminal rcb=7 Tiempo_rcb=54395
[54395] Monitor hijo #8: Push desde el DUT
[54395] Monitor: Transacción Recibida Retardo=0 Tipo=envio paquete=0xff00000a Terminal_env=0 Tiempo_env=0 Terminal rcb=8 Tiempo_rcb=54395
[54395] Monitor hijo #9: Push desde el DUT
[54395] Monitor: Transacción Recibida Retardo=0 Tipo=envio paquete=0xff00000a Terminal_env=0 Tiempo_env=0 Terminal rcb=9 Tiempo_rcb=54395
[54400] Checker: Se recibe trasacción desde el monitor Retardo=0 Tipo=envio paquete=0xff00000a Terminal_env=0 Tiempo_env=0 Terminal rcb=0 Tiempo_rcb=54395
[54400] Checker: Error el dato de la transacción no calza con el esperado Retardo=0 Tipo=envio paquete=0xff00000a Terminal_env=0 Tiempo_env=0 Terminal rcb=0 Tiempo_rcb=54395
$finish called from file "Checker.sv", line 89.
```

Fig. 27: Recepción del paquete por todos los demás terminales.

6. Gráficos

Gracias a los reportes que genera el scoreboard con cada prueba, se corrieron diferentes pruebas con distinto número de terminales y profundidad de FIFO's pero manteniendo constante el ancho de palabra en 32 bits. Por medio de los archivos .csv se generaron los gráficos de las Fig. 28, 29, 30 y 31.

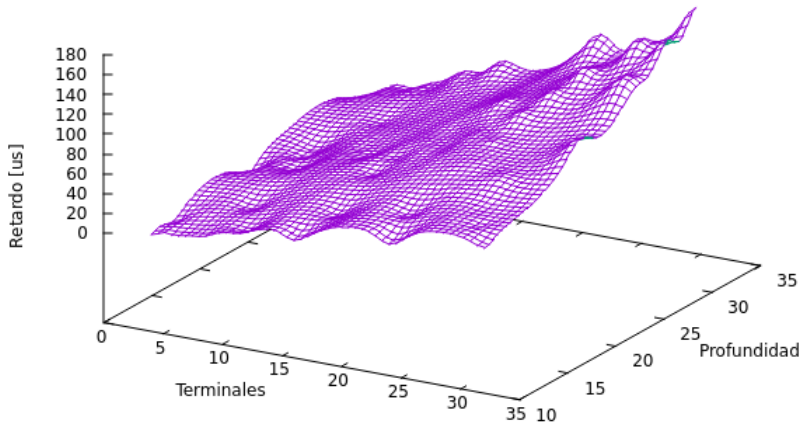

Fig. 28: Retardo promedio de las transmisiones con 32 bits

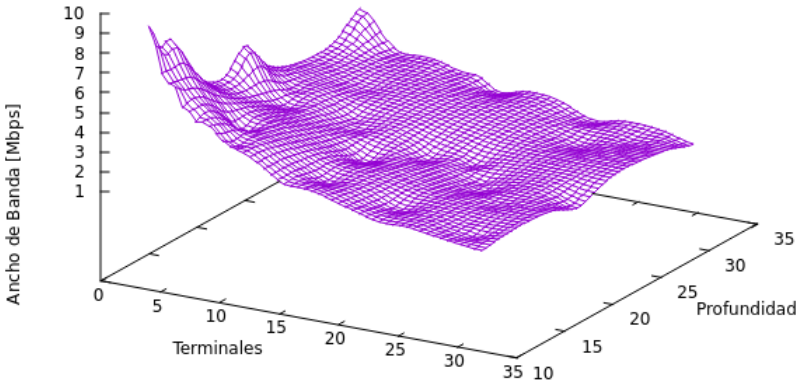

Fig. 29: Ancho de banda promedio de las transmisiones con 32 bits

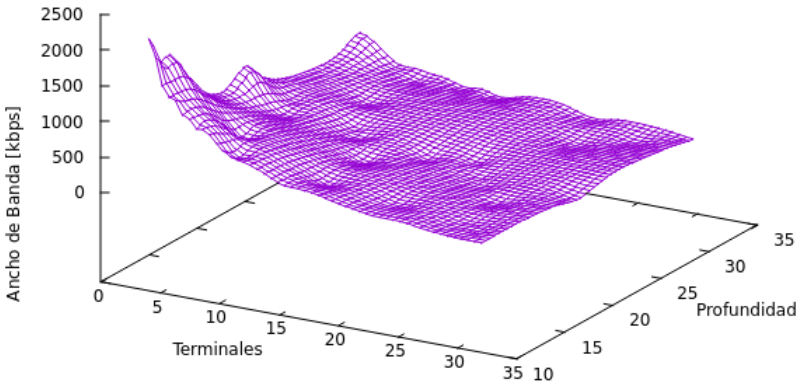

Fig. 30: Ancho de banda mínimo de las transmisiones con 32 bits

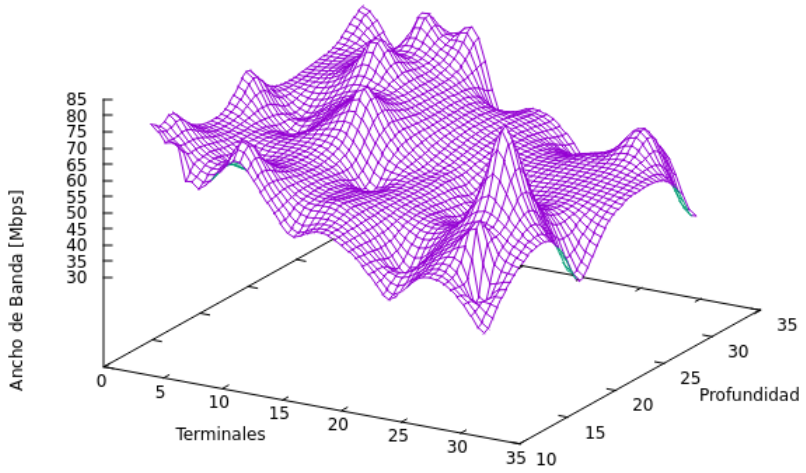

Fig. 31: Ancho de banda máximo de las transmisiones con 32 bits

7. Análisis de Resultados

A continuación, se presenta el análisis de los resultados obtenidos de la verificación del BUS.

7.1. Pruebas Exitosas

1. Transacción Aleatoria: La Figura 11 muestra la creación de una transacción aleatoria por el agente, y la Figura 12 demuestra que esta transacción se recibió correctamente en el scoreboard.

2. Transacción hacia el Mismo Terminal: Aunque es un caso de esquina, la prueba de enviar una transacción hacia el mismo terminal se ejecutó correctamente. La Figura 13 muestra la creación de la transacción, y la Figura 14 refleja que la transacción se recibió en el scoreboard, pero su estado indica que no se procesó correctamente.

3. Transacción hacia un Terminal Inexistente: Esta prueba también se realizó con éxito. La Figura 15 muestra la creación de la transacción hacia un terminal inexistente, y la Figura 16 muestra que el DUT no envió la transacción a ninguno de los terminales disponibles.

4. Sección de Transacciones Aleatorias: La ejecución de 100 transacciones aleatorias se completó sin problemas, como se muestra en la Figura 17. Las transacciones se registraron adecuadamente en el scoreboard, como se puede ver en la Figura 18.

5. Broadcast: El envío y recepción de un broadcast se ejecutó correctamente, como se ilustra en la Figura 19 y se confirma en la Figura 20.

6. Sección de Transferencias Aleatorias por Terminal: Esta prueba, que involucra 50 transacciones aleatorias en total (5 por terminal), también se completó con éxito y demostró que todos los terminales podían enviar y recibir datos a través de sus FIFOs de entrada y salida.

7. Llenado de FIFOs de Entrada: La prueba de llenado de FIFOs de entrada se realizó adecuadamente y permitió calcular el retardo promedio general y por terminal, así como el ancho de banda promedio, máximo y mínimo de la transmisión. La Figura 21 muestra parte de los resultados de esta prueba.

8. Reportes de Retardos y Anchos de Banda: Los reportes de retardo general y por terminal, así como el ancho de banda promedio, máximo y mínimo (Figuras 22 y 23), que proporcionan información valiosa sobre el rendimiento del sistema, se pudieron determinar sin problemas.

7.2. Bug Encontrado

Se identificó un problema en la prueba relacionada con el broadcast. A pesar de cambiar el parámetro de identificador de broadcast en el testbench, el DUT no envió la transacción de broadcast a todos los terminales, como se esperaba. El DUT solo envió la transacción de broadcast al terminal 0. Este comportamiento se documenta en las Figuras 24, 25, 26 y 27.

7.3. Gráficos

Se generaron gráficos a partir de los reportes del scoreboard para evaluar el rendimiento del sistema. Los gráficos incluyen el retardo promedio de las transmisiones (Figura 28), el ancho de banda promedio (Figura 29), el ancho de banda mínimo (Figura 30) y el ancho de banda máximo (Figura 31). Estos gráficos proporcionan una representación visual de los datos recopilados durante las pruebas y permiten evaluar el rendimiento del sistema en función de sus parámetros.

Por ejemplo, el retardo promedio es proporcional a la cantidad de terminales y a la profundidad de las FIFO's. Por el contrario, el BW mínimo y promedio decrece conforme aumentan estos parámetros.

7.4. Valores Destacados

1. El retardo promedio menor del sistema es de 49050 ns, con un máximo de 61978 ns.

2. El ancho de banda máximo alcanza los 84210 kbps, mientras que el ancho de banda mínimo es de 294 kbps. El ancho de banda promedio es de 2892 kbps.

8. Conclusiones

En general, las pruebas realizadas arrojaron resultados positivos. Las transacciones aleatorias, las transferencias entre terminales y el llenado de FIFOs de entrada se ejecutaron sin problemas, demostrando la capacidad del sistema para gestionar múltiples operaciones de manera eficiente.

Además, los reportes de retardo y ancho de banda proporcionan información valiosa sobre el rendimiento del sistema.

Sin embargo, se identificó un problema en la prueba relacionada con el broadcast. Este inconveniente necesita una revisión en el diseño para solucionarse.

Los gráficos generados a partir de los reportes destacan algunos aspectos importantes del rendimiento del sistema. Por ejemplo, el retardo promedio y el ancho de banda están directamente relacionados con la cantidad de terminales y la profundidad de las FIFOs. Observamos un retardo promedio mínimo de 49050 ns y un ancho de banda promedio de 2892 kbps, lo que sugiere un rendimiento veloz en términos de tiempo y ancho de banda.