Escuela de Ingeniería Electrónica
Licenciatura en Ingeniería Electrónica
Verificación funcional de circuitos integrados
II Semestre 2023

TEC | Tecnológico de Costa Rica

Proyecto 1

Plan de pruebas para la verificación de un bus de datos parametrizable

Jeremy Cordoba Wright
Jesús Rojas Vargas

Índice

1. Planteamiento 3

2. Diagrama del ambiente de verificación 3

3. Paquetes e interfaces de comunicación 4

4. Escenarios 6

Resumen

El proyecto tiene como objetivo principal diseñar un ambiente de verificación efectivo para un bus de datos implementado en SystemVerilog. Este bus de datos se utiliza para la transferencia de información entre diferentes periféricos. La verificación es una parte crítica del proceso de diseño de hardware, ya que garantiza que el bus de datos funcione de manera confiable y cumpla con las especificaciones definidas.

1. Planteamiento

Para el diseño del ambiente de verificación se toma en cuenta el funcionamiento del dispositivo que en este caso es un bus de datos que transmite información a diferentes periféricos interpretándolos como fifos. Como el objetivo especifico es verificar el bus, las fifos serán descritas por software y sus datos serán todos definidos como específicos o aleatorios dependiendo del caso o escenario que se este verificando, además, todos los escenarios serán verificados en todas las simulaciones del sistema que se realicen. También, cada unas de las simulaciones tendrán un reporte final para poder apreciar de manera textual, todos los datos relevantes como el tiempo de simulación, retardos, datos transmitidos, escenarios verificados y errores encontrados.

En el documento se presentara un diagrama que describe el ambiente de verificación a desarrollar, con el objetivo de desarrollar un sistema en software entendible, ordenado y reutilizable para otros buses de datos o actualizaciones del mismo en estudio. Cada modulo se comunicara mediante mailboxes con el fin de poder intercambiar información entre unidades de prueba y solicitar datos de forma mas controlada, además, cada dato transferible entre bloque tendrá su propio formato definido, con el fin de poder enviar u obtener información que describa el inicio de una simulación (casos, aliteración, retardos, datos) y el fin de la misma (duración, datos recibidos, errores, ancho de banda).

El bus de datos se encargara de transmitir información donde pueden ocurrir una serie de casos todos distintos entre si, por lo tanto, se van definir dentro del ambiente de verificación en desarrollo con el fin de que este encargue de someter al dispositivo bajo prueba a estos tipos de escenarios, cabe destacar, que existen escenarios con una probabilidad muy baja de ocurrencia, por lo que, se va a hacer un estudio exhaustivo de todos estos casos de esquina para poder definirlos dentro del ambiente y cubrir mas porcentaje de verificación, finalmente, el ambiente contara con aleatorizacion en la mayoría de sus características lo que resultara en un ambiente mas autónomo.

2. Diagrama del ambiente de verificación

En la Fig. 1 se presenta el diagrama en capas correspondiente al testbench para el bus parametrizable, en este se muestran todos los módulos así como las conexiones entre estos para llevar a cabo la verificación.

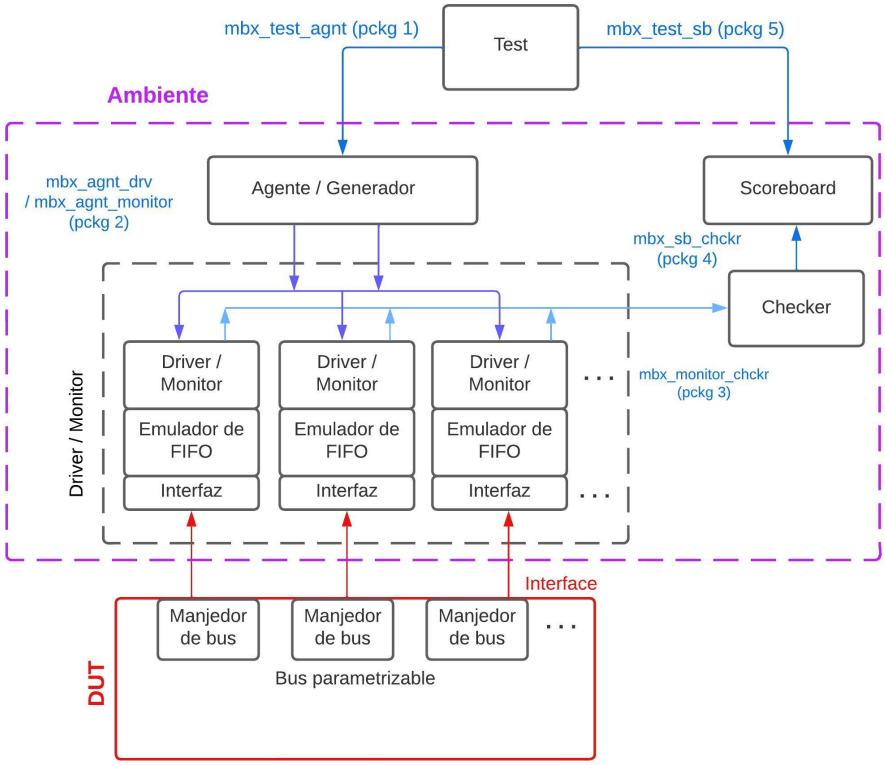

Figura 1: Diagrama para el sistema de pruebas

3. Paquetes e interfaces de comunicación

Como se observa en la 1 cada unidad tiene su conexión correspondiente, las cuales se indican como *mbx _ unidad _ unidad (pckg #)*. A continuación se presenta el formato de cada uno de los paquetes de comunicación entre todas las unidades de prueba.

1. **mbx_test_agnt (pckg 1):** Se trata de un mailbox que comunica al test con el agente, el tipo de mailbox es de *instrucciones_agente* el cual es un tipo de estructura que contiene las siguientes características son:
   - Tipo de secuencia:
     - Envío aleatorio.
     - Broadcast aleatorio.
     - Reset half sent.
     - All for one.
     - All broadcast.
     - One for one.
     - Unknow id.

2. **mbx_ agnt_drv (pckg 2):** Se trata de un arreglo de mailboxes que comunica al agente con el driver y con el monitor para cada dispositivo, ambos son de tipo *trans_bus* , sus características son:

    - Tipo:
        - Envio.
        - Broadcast.
        - Reset.
    - Max retardo.
    - Retardo.
    - Paquete.
    - Tiempo envio.
    - Tiempo recibido.
    - Driver.
    - Destino.
    - Dato recibido.
    - Monitor receptor.
    - ID.
    - ID unknow.
    - Payload.

3. **mbx_monitor_chckr (pckg 3):** Arreglo de mailboxes de tipo *trans_bus* que comunica al monitor de cada dispositivo con el checker, sus características son:

4. **mbx_chckr_sb (pckg 4):** Mailbox que comunica al checker con el scoreboard, sus características son:

    - Tiempo de envío.
    - Tiempo de recibido.
    - Latencia.
    - Dato.
    - Tipo de resultado:
        - Overflow.
        - Underflow.
        - Reset.
        - Completado.

5. **mbx_test_sb (pckg 5):** Mailbox que comunica al test con el scoreboard, sus características son:

    - Tipo de reporte:
        - Paquetes recibidos por terminal.

- Paquetes enviados por terminal.
- Retardo promedio por terminal.
- Ancho de banda promedio máximo.
- Ancho de banda promedio mínimo.

4. Escenarios

En esta sección se definirán los diferentes casos generales y de esquina que pueden presentarse dentro del dispositivo bajo prueba.

Casos comunes		
Escenarios	Objetivos	Recursos
En un DUT con una cantidad de drivers de 8 a 16 aleatorizado para cada prueba, con las FIFOS de profundidad de 1 a 32 y con un tamaño de palabra de 8 a 32 bits aleatorizado, con una cantidad de transacciones aleatorias de entre 100 a 200 transacciones, una palabra aleatoria con un payload totalmente aleatorio dependiendo del tamaño de la palabra, con un ID aleatorio pero dentro de los límites de la cantidad de drivers y con posibilidad de broadcast , con tiempos entre cada transacción aleatorio de entre 1 ciclo a 10 ciclos y con el orden de transacciones de envio de las FIFOs aleatorios dependiendo de la cantidad de transacciones.	Verificar que todos los dispositivos conectados al DUT puedan enviar y recibir datos de forma aleatoria desde cualquier driver y que también sean capaces de funcionar con diferentes retardos, profundidades, tamaños de palabra, cantidad de FIFOs, número de transacciones y diferente orden, también se debe verificar que si algún driver hace broadcast todos los demás drivers lo reciban correctamente.	El ambiente debe ser capaz de aleatorizar la cantidad de drivers en el inicio de la prueba, la profundidad de las FIFOs y el tamaño de su paquete debe establecer una cantidad aleatoria de transacciones donde cada transacción tendrá asociado un retardo distinto y una FIFOs desde la cual se estará enviando el dato, el broadcast se espera que en algún momento se genere de forma aleatoria.
En un DUT con una cantidad de drivers de 8 a 16 aleatorizado para cada prueba, con las FIFOS de profundidad de 1 a 32 y con un tamaño de palabra de 8 a 32 bits aleatorizado, con una cantidad de transacciones aleatorias de entre 100 a 200 transacciones, una palabra aleatoria con un payload totalmente aleatorio dependiendo del tamaño de la palabra, con un ID constante de broadcast, tiempos entre cada transacción aleatorio de entre 1 ciclo a 10 ciclos y con el orden de transacciones de envio de las FIFOs aleatorios dependiendo de la cantidad de transacciones.	Verificar que, al hacer broadcast desde cualquier driver, con un orden y retardo aleatorios siempre todos los dispositivos estén recibiendo el dato que se está transmitiendo, excepto el mismo que lo transmitió.	El ambiente debe ser capaz de aleatorizar la cantidad de drivers en el inicio de la prueba, la profundidad de las FIFOs y el tamaño de su paquete debe establecer una cantidad aleatoria de transacciones donde cada transacción tendrá asociado un retardo distinto y una FIFO desde la cual se estará enviando el dato, sin embargo, en este caso el ambiente tomara el valor de broadcast a la entrada del DUT y lo mantendrá en el ID de forma constante con el fin de que únicamente se esté verificando la función de broadcast
Casos de esquina		
En un DUT con una cantidad de drivers de 8 a 16 aleatorizado para cada prueba, con las FIFOS de profundidad de 1 a 32 y con un tamaño de palabra de 8 a 32	Verificar que en el momento que el DUT está tomando el dato desde la FIFO de entrada o enviándolo a una FIFO de salida se haga un reset y que el	El ambiente debe ser capaz de aleatorizar la cantidad de drivers en el inicio de la prueba, la profundidad de las FIFOs y el tamaño de su paquete debe

bits aleatorizado, con una cantidad de transacciones aleatorias de entre 100 a 200 transacciones, una palabra aleatoria con un payload totalmente aleatorio dependiendo del tamaño de la palabra, con un ID aleatorio pero dentro de los límites de la cantidad de drivers y con posibilidad de broadcast , con tiempos entre cada transacción aleatorio de entre 1 ciclo a 10 ciclos y con el orden de transacciones de envio de las FIFOs aleatorios dependiendo de la cantidad de transacciones, donde cada transacción tendrá asociado un dato de dos bits que representa que represe el push y pop deL DUT.	paquete nunca llegue a su destino.	establecer una cantidad aleatoria de transacciones donde cada transacción tendrá asociado un retardo distinto y una FIFOs desde la cual se estará enviando el dato, el broadcast puede ser generado de forma aleatoria, además cada transacción tendrá asociado un valor que indica si el reset se hará en el push o pop del DUT.
En un DUT con una cantidad de drivers de 8 a 16 aleatorizado para cada prueba, con las FIFOS de profundidad de 32 y con un tamaño de palabra de 8 a 32 bits aleatorizado, con una cantidad de transacciones aleatorias de entre 100 a 200 transacciones, una palabra aleatoria con un payload aleatorio dependiendo del tamaño de la palabra y un ID de destino aleatorio, cada transacción tendrá su retardo de entre 1 a 10 ciclos de reloj donde después del retardo todos los drivers envían el dato al mismo tiempo, además se tendrá un vector de drivers que tendrán este ID donde solo se exceptúa uno del vector que será el que recibirá los datos.	Verificar que al enviar paquetes desde todos los drivers a uno solo este sea capaz de recibir todos los paquetes, cada driver será sometido a la misma prueba para comprobar el funcionamiento completo	El ambiente debe ser capaz de aleatorizar la cantidad de FIFOs y tamaños de palabras, la cantidad de transacciones, el contenido del paquete donde el ID se considera dentro de los rangos establecidos por la cantidad de FIFOs, debe aleatorizar el retardo en el cual varias FIFOs a la vez enviaran un dato a una sola, debe tener un vector que indique cuales FIFOs son las que enviaran el dato y cual lo recibirá, de esta forma se vuelve aleatorio y todas tienen posibilidad de ser probadas.
En un DUT con una cantidad de drivers de 8 a 16 aleatorizado para cada prueba, con las FIFOS de profundidad de 32 y con un tamaño de palabra de 8 a 32 bits aleatorizado, con una cantidad de transacciones	Verificar que 2 o más drivers al hacer broadcast al mismo tiempo, entonces todos reciban el dato como corresponda excepto los que hicieron el respectivo broadcast	El ambiente debe ser capaz de aleatorizar el tamaño de la palabra, la cantidad de FIFOs, el numero de transacciones y los retardos correspondientes, además, debe crear un vector con los drivers que harán el

aleatorias de entre 10 a 100 transacciones, una palabra aleatoria con un payload aleatorio dependiendo del tamaño de la palabra, cada transacción tendrá su retardo de entre 1 a 10 ciclos de reloj donde después del retardo todos los drivers hacen broadcast al mismo tiempo, su ID se mantiene constante dependiendo del que se definió en la compilación y un vector de drivers que tendrán este ID, no todos harán broadcast, eso será aleatorio.		broadcast, no necesariamente deben ser todos al mismo tiempo.
En un DUT con una cantidad de drivers de 8 a 16 aleatorizado para cada prueba, con las FIFOS de profundidad de 32 y con un tamaño de palabra de 8 a 32 bits aleatorizado, con una cantidad de transacciones aleatorias de entre 10 a 100 transacciones, una palabra aleatoria con un payload aleatorio dependiendo del tamaño de la palabra, cada transacción tendrá su retardo de entre 1 a 10 ciclos de reloj donde después del retardo todos los drivers se envían el dato a ellos mismos, el ID se mantiene constante al driver que corresponde.	Verificar que en el momento que un driver se mande un dato a si mismo este lo reciba, con diferentes retardos.	El ambiente debe ser capaz de aleatorizar el tamaño de la palabra y la cantidad de drivers, la cantidad de transacciones debe ser aleatoria donde cada transacción tiene su tiempo de retardo, su vector de drivers que harán la transacción, un ID constante y un payload aleatorio.
En un DUT con una cantidad de drivers de 8 a 16 aleatorizado para cada prueba, con las FIFOS de profundidad de 32 y con un tamaño de palabra de 8 a 32 bits aleatorizado, con una cantidad de transacciones aleatorias de entre 10 a 100 transacciones, una palabra aleatoria con un payload aleatorio dependiendo del tamaño de la palabra, cada paquete tendrá un ID fuera de los límites establecidos por la cantidad de drivers, cada transacción tendrá su retardo	Verificar que en el momento que el DUT reciba un paquete con un ID que no corresponde a ninguno de los drivers este simplemente lo deseche.	El ambiente debe ser capaz de aleatorizar el tamaño de DUT y el tamaño de la palabra, cada palabra tendrá un payload aleatorio y un ID fuera de los limites establecidos, debe ser capaz de aleatorizar la cantidad de transacciones donde cada transacción tendrá asociado un retardo y un driver el cual lo realizará.

de entre 1 a 10 ciclos de reloj y el driver que va a realizar la transacción.		