Escuela de Ingeniería Electrónica
Licenciatura en Ingeniería Electrónica
Verificación funcional de circuitos
II Semestre 2024

TEC | Tecnológico de Costa Rica

**Proyecto 1**

# Testplan: Verificación del Bus Driver

David Medina Mayorga, Pedro Medinila Robles

## 1. Descripción del DUT

Para el diseño del ambiente de verificación, se toma en cuenta el funcionamiento del dispositivo, que en este caso es un manejador de bus de datos que permite la transmisión de información entre diferentes periféricos. Estos periféricos serán emulados mediante FIFOs, y el objetivo específico es verificar el correcto funcionamiento del bus. Las FIFOs serán descritas a nivel de software y los datos transmitidos serán definidos como específicos o aleatorios, dependiendo del tipo de prueba que se ejecute.

El ambiente de verificación se organiza en torno a dos módulos principales: el controlador del bus de datos, también conocido como Bus Driver, y los dispositivos FIFO. El Bus Driver tiene la responsabilidad de administrar el acceso de los diferentes periféricos al bus de datos, asegurando que las comunicaciones se realicen de manera ordenada y sin conflictos. Por otro lado, los dispositivos FIFO funcionan como una interfaz intermedia que almacena temporalmente los datos transmitidos entre el DUT y el ambiente de verificación, simulando el comportamiento de periféricos reales conectados al bus.

### 1.1. Señales

- **pndng:** si la FIFO está vacía, la señal `pndng` es 0.
- **push:** señal de control que indica que se debe realizar una operación de `push` (es decir, ingresar datos a la FIFO).
- **pop:** señal de control que indica que se debe realizar una operación de `pop` (es decir, extraer datos de la FIFO).

- **Dpush:** datos a ser almacenados en la FIFO.
- **Dpop:** datos a ser extraídos de la FIFO.
- **Broadcast:** representa una señal con un valor en los bits de ID por defecto de 8 bits (todo 1s: `11111111`). Esto significa que el módulo tiene una funcionalidad de transmisión de mensajes a múltiples destinatarios simultáneamente (*broadcast*).

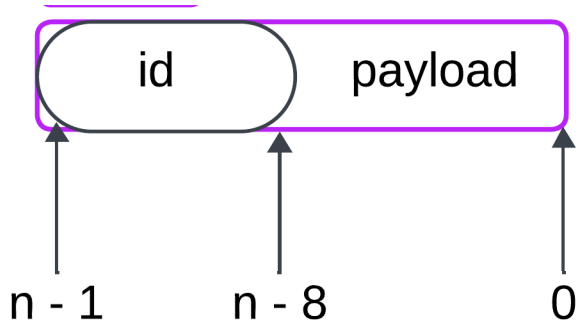


Figura 1: Formato del paquete de información

# 2. Escenarios

## 2.1. Casos Generales

Como casos generales o de uso común se tienen las pruebas con cada transacción:

- **Enviar datos:** cada periférico debe poder enviar un paquete a otro periférico a través del bus de datos.
- **Recepción de datos enviados:** si se envía un paquete a cierto periférico, este debe recibirlo dado su ID, y el *payload* debe coincidir con el enviado.
- **Funcionamiento del reset:** el dispositivo debe ser capaz de reiniciarse correctamente cuando se active la señal de reset, vaciando todas las FIFOs y anulando las transacciones pendientes.
- **broadcast:** si un periférico realiza un *Broadcast* con el identificador 8'd1, todos los periféricos deben recibir el *payload* enviado.

## 2.2. Casos de Esquina

- **Overflow:** ocurre cuando se intenta realizar un `push` en una FIFO que ya está llena.
- **Underflow:** ocurre cuando se intenta realizar una lectura (`pop`) en una FIFO que está vacía.

- **Varios envían a la vez:** este caso prueba el comportamiento del bus cuando varios periféricos intentan enviar datos al mismo tiempo.
- **Todos envían a la vez:** este caso prueba el comportamiento del bus cuando cada periférico envía datos simultáneamente.
- **Se envía un paquete con un ID ilegal:** con un número determinado de periféricos, se utiliza un ID que no existe en el sistema.
- **Todos hacen broadcast simultáneamente:** similar al caso anterior, pero en este caso todos los periféricos intentan hacer *broadcast* al mismo tiempo.
- **Varios hacen broadcast:** prueba el comportamiento del bus cuando varios periféricos realizan *broadcast* simultáneamente.
- **Se envía a sí mismo:** consiste en enviar un dato desde un periférico para volver a recibirlo en el mismo periférico.
- **Uno solo envía una ráfaga de paquetes:** prueba el comportamiento del dispositivo cuando un periférico envía datos de forma continua y exhaustiva.
- **Todos le envían al mismo simultáneamente:** evalúa la capacidad de un periférico para recibir datos simultáneamente de todos los demás periféricos.
- **Reset cuando no se ha terminado una transacción:** el comportamiento esperado es que el dispositivo se vacíe, anulando las transacciones en curso y reiniciando el sistema.

# 3. Aleatorización

Además de las pruebas descritas, se aleatorizarán distintas variables del test para evaluar el comportamiento del sistema bajo diferentes condiciones:

- **Número de transacciones:** se probarán distintas cantidades de transacciones para verificar el rendimiento y comportamiento del dispositivo.
- **Largo de paquetes:** se aleatorizará el largo de los paquetes entre 16, 32 y 64 bits.
- **Tiempo de espera entre eventos:** para las distintas transacciones se aleatorizarán los tiempos de retardo entre eventos.
- **Cantidad de periféricos:** dado que el dispositivo es capaz de manejar una cantidad variable de periféricos, se probará con diferentes números de periféricos para identificar posibles cambios en el comportamiento.
- **Dimensión de profundidad en la FIFO:** los periféricos pueden tener FIFOs de diferentes tamaños, lo que se probará para asegurar que el comportamiento sea consistente sin importar la profundidad de la FIFO.

- **Tipos de transacciones:** las transacciones pueden variar en orden y cantidad, cada una realizando diferentes acciones, lo que será aleatorizado.
- **Payload:** los datos enviados a los periféricos se aleatorizarán para probar diferentes contenidos.
- **Identificadores válidos:** cada periférico posee un ID, los cuales se asociarán a los periféricos presentes durante la prueba.
- **Identificadores no válidos:** como prueba adicional, se utilizarán identificadores asociados a periféricos que no existen para verificar el comportamiento del sistema ante situaciones no válidas.

# 4. Diseño del Ambiente

El ambiente consiste en varios transacciones, conformado por un generador de *test*, un agente, un *driver*/*monitor*, un *checker* y un *scoreboard*. Además, el *driver* incluye una interfaz para comunicarse con el DUT.

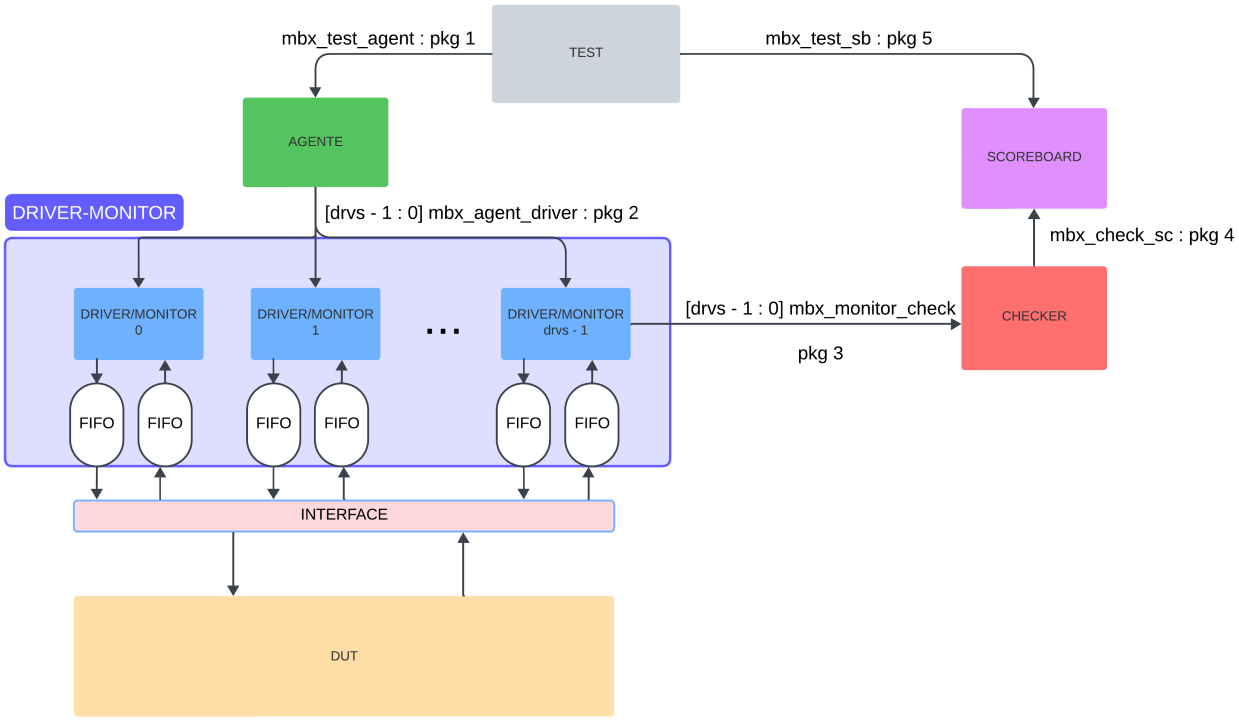


Figura 2: Diagrama del ambiente de verificación con el DUT

## 4.1. Generador de Tests

El generador es el bloque encargado de crear los estímulos o entradas que se enviarán al DUT. Estos estímulos pueden ser aleatorios o predefinidos, dependiendo del tipo de prueba. Este componente

genera transacciones, que son estructuras de datos que representan las entradas para el DUT.

## 4.2. Agente

El agente es un transactor que recibe las instrucciones de alto nivel del test y las traduce a un nivel de abstracción menor para enviarlas a los *drivers*.

## 4.3. Driver-Monitor

El *driver* convierte las transacciones generadas por el generador de *tests* en señales específicas que el DUT puede entender. Estas señales son enviadas al DUT a través de una interfaz, interactuando directamente con la lógica de entrada del DUT. A su vez, el *monitor* observa las señales provenientes del DUT sin modificarlas. Su función es recopilar datos para su análisis posterior. Los datos recogidos pueden ser utilizados para verificar el correcto funcionamiento del DUT.

## 4.4. Checker

El *checker* compara las salidas del DUT con los valores esperados, para asegurar que el DUT se comporta de acuerdo con las especificaciones.

## 4.5. Scoreboard

El *scoreboard* es una base de datos que contiene los resultados y reportes de las distintas pruebas, permitiendo hacer un seguimiento del desempeño del DUT durante el proceso de verificación.

## 4.6. Interfaz del Driver con el DUT

La interfaz es el medio a través del cual el *driver* interactúa con el DUT. A través de esta interfaz, el *driver* envía los estímulos convertidos al DUT, y el DUT devuelve las respuestas correspondientes. La interfaz maneja las señales de bajo nivel necesarias para la comunicación adecuada entre ambos. En la siguiente imagen se detallan las señales de salida y entrada del DUT y su conexión a las FIFOs emuladas en el *Driver-Monitor*, esto se puede observar en la Figura 3.

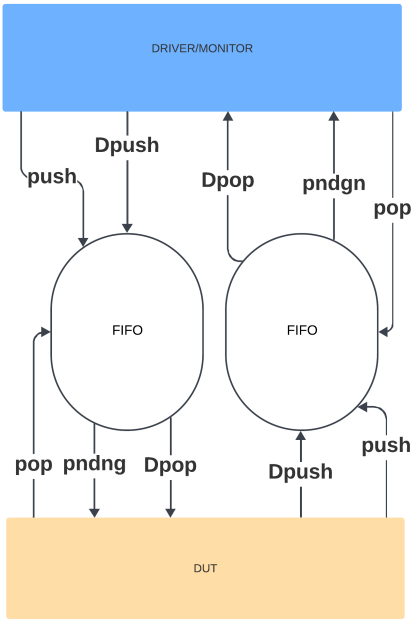


Figura 3: Interfase entre Driver y DUT

## 4.7. Interfaces de Comunicación entre Bloques y Paquetes

Relacionado al diagrama de bloques del ambiente, los paquetes de transacción son los siguientes:

### 4.7.1. mbx_test_agent : pkg1

Este *mailbox* comunica al test con el agente. El tipo de *mailbox* es paramétrico de datos `instrucciones_agente`, el cual es una estructura que contiene las siguientes características:

- **Tipos de secuencia:**
  - `send_random_payload_legal_id`
  - `send_random_payload_ilegal_id`
  - `send_w_mid_reset`
  - `consecutive_send`
  - *`broadcast`*`_random`
  - `some_sending_random`
  - `some_`*`broadcast`*
  - `all_for_one`
  - `all_sending_random`
  - `all_`*`broadcast`*
  - `auto_send_random`

#### 4.7.2. mbx_agent_driver : pkg2

Este es un arreglo de *mailboxes* de ancho `drvs` que comunica al agente con el *driver* de cada periférico según corresponda. Estos *mailboxes* contienen datos `instrucciones_driver`.

- **Tipos de secuencia:**
  - `send`
  - `reset`

#### 4.7.3. mbx_monitor_check : pkg3

Este es un arreglo de *mailboxes* de ancho `drvs` que comunica a cada monitor de cada periférico con el *checker*. Estos *mailboxes* contienen datos `instrucciones_monitor`.

- **Tipo de secuencia:**
  - `receive`
- **Datos:**
  - `max_delay`
  - `delay`
  - `package`
  - `send_time`
  - `receive_time`
  - `dest`
  - `receiver_monitor`
  - `id`
  - `payload`

#### 4.7.4. mbx_check_sb : pkg4

Este *mailbox* comunica al *checker* con el *scoreboard* a un nivel de abstracción más alto. El tipo de dato es `res_check`.

- **Tipos de secuencia:**
  - `send_time`
  - `receive_time`
  - `latency`
  - `data`

- `id`
- `result`

- **Tipos de resultado:**
  - `overflow`
  - `underflow`
  - `reset`
  - `completed`
  - `error`

#### 4.7.5. mbx_test_sb : pkg5

Este *mailbox* comunica al *test* con el *scoreboard* para hacer las consultas de reportes. El tipo de dato es `consulta_sb`.

- **Tipo de reporte:**
  - Paquetes recibidos por terminal.
  - Paquetes enviados por terminal.
  - Retardo promedio por terminal.
  - Retardo promedio total.
  - Transacciones totales.
  - Reporte de *broadcast*.
  - Detalle de tipos de resultados.
  - Reporte completo.

# 5. Plan de Pruebas

## 5.1. Recursos generales

Para llevar a cabo el plan de pruebas, es necesario cumplir con los siguientes requisitos:

- Archivos RTL que contengan el DUT.
- FIFOs emuladas.
- Ambiente de verificación.
- Acceso al servidor con las herramientas de Synopsys, específicamente VCS y Verdi.
- Acceso al repositorio de git del proyecto.

| Escenario | Objetivo | Detalle |
|---|---|---|
| DUT con drivers entre 8 y 16 unidades, con **profundidades de FIFO de 32** y tamaños de palabra de 8 a 32 bits, todo aleatorizado, con entre 10 y 100 transacciones, cada paquete tendrá un ID fuera de los límites establecidos. Cada transacción tendrá un retardo de entre 1 y 10 ciclos y un driver que realizará la transacción. | Verificar el overflow al estar una FIFO llena | El ambiente debe aleatorizar el tamaño del DUT y de la palabra, con un payload aleatorio y un ID fuera de los límites. La cantidad de transacciones y retardos debe aleatorizarse, y cada transacción debe asignarse a un driver. Encardado de la prueba: David Medina |
| DUT con drivers entre 8 y 16 unidades, con profundidades de FIFO infinitas y tamaños de palabra de 8 a 32 bits, todo aleatorizado, con entre 10 y 100 transacciones, cada paquete tendrá un ID fuera de los límites establecidos. Cada transacción tendrá un retardo de entre 1 y 10 ciclos y un driver que realizará la transacción. | Verificar que cuando el DUT recibe un paquete con un ID fuera de los límites de los drivers, **lo descarte** correctamente. | El ambiente debe aleatorizar el tamaño del DUT y de la palabra, con un payload aleatorio y un ID fuera de los límites. La cantidad de transacciones y retardos debe aleatorizarse, y cada transacción debe asignarse a un driver. Encardado de la prueba: Encargado de la prueba: Pedro Medinila |
| DUT con drivers entre 8 y 16 unidades, con profundidades de FIFO de 1 a 32 y tamaños de palabra de 8 a 32 bits aleatorizados, con entre 100 y 200 transacciones consecutivas enviadas desde un mismo driver, con un payload aleatorio y tiempos de transacción aleatorios de entre 1 y 10 ciclos. | Verificar que un mismo driver pueda enviar múltiples **transacciones consecutivas** sin perder datos, y que todos los datos lleguen a su destino correcto sin interferencias o pérdidas. | El ambiente debe aleatorizar la cantidad de drivers, la profundidad de las FIFOs y el tamaño de los paquetes. Debe establecer un número aleatorio de transacciones consecutivas desde un solo driver, donde cada transacción tenga su propio retardo y FIFO de origen. Encardado de la prueba: David Medina |
| DUT con drivers entre 8 y 16 unidades, con profundidades de FIFO infinitas y tamaños de palabra de 8 a 32 bits, todo aleatorizado, con entre 10 y 100 transacciones, cada palabra con un payload aleatorio y retardos de 1 a 10 ciclos. Tras el retardo, todos los drivers envían datos **a sí mismos**, manteniendo un ID constante. | Verificar que cuando un driver se envía un dato a sí mismo, lo reciba correctamente bajo diferentes retardos. | El ambiente debe aleatorizar el tamaño de palabra y la cantidad de drivers. Las transacciones serán aleatorias, cada una con su retardo, y un vector de drivers realizará la transacción con un payload aleatorio. Encardado de la prueba: Encargado de la prueba: Pedro Medinila |

| Escenario | Objetivo | Detalle |
|---|---|---|
| DUT con drivers entre 8 y 16 unidades, con profundidades de FIFO de 1 a 32 y tamaños de palabra de 8 a 32 bits, todos aleatorizados, y con entre 100 y 200 transacciones, cada palabra con un payload y un ID aleatorios dentro de los límites de los drivers, con tiempos de transacción aleatorios de entre 1 y 10 ciclos. Cada transacción tendrá dos bits adicionales indicando push y pop. | Verificar que si el DUT realiza un **reset** mientras se toma o envía un dato a través de una FIFO, el paquete no llegue a su destino. | El ambiente debe poder aleatorizar la cantidad de drivers, profundidades de FIFO y tamaños de palabra al inicio de la prueba, además de establecer transacciones aleatorias, cada una con un retardo y una FIFO de origen. El *broadcast* puede generarse de manera aleatoria, y cada transacción debe tener dos bits indicando si el reset se realiza en push o pop. Encardado de la prueba: David Medina |
| DUT con drivers entre 8 y 16 unidades, con profundidades de FIFO infinitas y tamaños de palabra de 8 a 32 bits, todo aleatorizado, con entre 100 y 200 transacciones, cada palabra con un payload aleatorio y un ID de destino aleatorio. Cada transacción tendrá un retardo de entre 1 a 10 ciclos, tras el cual todos los drivers envían datos **al mismo tiempo** a un solo driver. | Verificar que un driver pueda recibir paquetes desde todos los demás drivers simultáneamente, y probar este escenario con cada driver. | El ambiente debe aleatorizar la cantidad de FIFOs, los tamaños de palabras, la cantidad de transacciones y el contenido del paquete, con IDs dentro de los límites de las FIFOs. El ambiente también debe aleatorizar el retardo en el que varias FIFOs envían datos al mismo tiempo y determinar cuáles envían los datos y cuál los recibe. Encardado de la prueba: Encargado de la prueba: Pedro Medinila |
| DUT con drivers entre 8 y 16 unidades, profundidades de FIFO de 1 a 32 y tamaños de palabra de 8, 16 y 32 bits (todos aleatorizados) y entre 100 y 500 transacciones, donde cada palabra y su payload son generados aleatoriamente según el tamaño, con IDs aleatorios que respetan el límite de los drivers, con **posibilidad** de *broadcast*, y tiempos de transacción entre 1 y 10 ciclos, todo aleatorio. | Verificar que los dispositivos conectados al DUT puedan intercambiar datos aleatoriamente desde cualquier driver, y que funcionen bajo diferentes condiciones: retardos, profundidades, tamaños de palabra, cantidad de FIFOs y número de transacciones. Además, asegurar que el *broadcast* desde cualquier driver sea recibido por todos los demás correctamente. | El ambiente debe poder aleatorizar la cantidad de drivers al inicio de cada prueba, así como la profundidad de las FIFOs y el tamaño de los paquetes. Además, debe generar un número aleatorio de transacciones, cada una con su propio retardo y FIFO de origen. El *broadcast* debe generarse aleatoriamente en algún punto. Encardado de la prueba: David Medina |

| Escenario | Objetivo | Detalle |
|---|---|---|
| DUT con drivers entre 8 y 16 unidades, con profundidades de FIFO de 1 a 32 y tamaños de palabra de 8 a 32 bits, todos aleatorizados, y con entre 100 y 200 transacciones, cada una con payload aleatorio y un ID de *broadcast* **constante**. Los tiempos de transacción varían de 1 a 10 ciclos y el orden de las transacciones es aleatorio. | Verificar que, cuando cualquier driver realiza un *broadcast*, todos los demás dispositivos reciban los datos, excepto el driver emisor. | El ambiente debe aleatorizar la cantidad de drivers, la profundidad de las FIFOs y el tamaño de los paquetes al inicio de la prueba. Además, debe generar transacciones aleatorias, cada una con un retardo asociado y una FIFO de origen. En este caso, el *broadcast* será constante y su ID fijo durante la prueba para verificar su correcto funcionamiento. Encardado de la prueba: Encargado de la prueba: Pedro Medinila |
| DUT con drivers entre 8 y 16 unidades, con profundidades de FIFO infinitas y tamaños de palabra de 8 a 32 bits, todo aleatorizado, con entre 10 y 100 transacciones, cada palabra con un payload aleatorio y retardos de 1 a 10 ciclos. Tras el retardo, todos los drivers realizan **broadcast simultáneamente**, manteniendo un ID constante definido en la compilación. Un vector de drivers seleccionará cuáles harán *broadcast*. | Verificar que cuando varios drivers realizan *broadcast* simultáneamente, todos los demás dispositivos reciban los datos correctamente, excepto los emisores. | El ambiente debe aleatorizar el tamaño de palabra, la cantidad de FIFOs, el número de transacciones y los retardos. También debe generar un vector que indique qué drivers realizan el *broadcast*, sin que necesariamente sea al mismo tiempo. Encardado de la prueba: David Medina |

# 6. Cronograma de actividades