Verificación Funcional de Circuitos Integrados
Tecnológico de Costa Rica
Proyecto I - Documentación

Hernández Zárate Amanda Pérez Ramírez Gabriel

1 Funcionamiento

A continuación se trata de resumir el funcionamiento general que tiene el DUT.

- El DUT tiene como objetivo general el recibir paquetes de datos sin un orden definido, y dar como salida estos mismos paquetes pero ordenados. La distribución que se va a seguir para el proceso de ordenamiento está definida en el registro de control. Los bits de "SIZE" corresponden a la cantidad de paquetes que hay por columna, siendo `SIZE = 1`, cuatro columnas de un solo paquete, `SIZE = 2`, dos columnas con dos paquetes y `SIZE = 4`, una única columna con los 4 paquetes recibidos. Mencionar que solo estos tres datos son válidos, en caso de ingresar algún otro, resultará en un error por parte del DUT. Ahora, los bits de "OFFSET" definen dicho parámetro para los datos que se envían. Para un `SIZE = 1` son válidos valores de `OFFSET = {0, 1, 2, 3}`, para un `SIZE = 2` son válidos valores de `OFFSET = {0, 2}` y con un `SIZE = 4` es válido el valor de `OFFSET = {0}`.

Ahora se van a mostrar algunas funcionalidades concretas que tiene el dispositivo en cuestión.

- Se pueden parametrizar los módulos, siendo `ALGN_DATA_WIDTH` el ancho de los datos y `FIFO_DEPTH` la profundidad de las dos FIFOs.
- Con la señal `md_rx_valid`, el usuario indica que quiere iniciar una transacción para enviar paquetes a ordenar. Este bit debe mantenerse en alto hasta que la señal `md_rx_ready` se ponga también en alto.
- Cuando se quiere enviar un dato desde la RX FIFO para que sea ordenado, si este es válido la señal `md_rx_ready` se pone en alto, indicando que se va a proceder con la transacción. De mantenerse en bajo, se detiene el tráfico de entrada. Para que una señal se considere válida, debe cumplir con `((ALGN_DATA_WIDTH / 8) + offset) % size == 0`.
- Con el registro de control se pueden definir ciertos criterios como el tamaño, en bytes, de los datos a alinear, `SIZE`. El offset, en bytes, de los datos a alinear, `OFFSET`. Y también un bit, `CLR`, para limpiar el contador de solicitudes inválidas `CNT_DROP`.
- Con el registro de estatus se pueden observar ciertos valores de interés. En primera instancia, el contador de solicitudes inválidas, `CNT_DROP`. Una vez este llegue a su valor máximo, va a mantenerse en este, hasta que se reinicie por medio de escribir un 1 en `CTRL.CRL`. Cada vez que este contador alcanza su valor máximo, se dispara una interrupción. En segunda instancia, el estado de capacidad en la que se encuentra la RX FIFO, `RX_LVL`. En tercera instancia, el estado de capacidad en la que se encuentra la TX FIFO, `TX_LVL`.
- Se tiene también el registro habilitador de solicitudes de interrupción. En este se pueden habilitar las solicitudes de interrupción que se muestran en el registro de solicitudes de interrupción. Este registro está compuesto por: `IRQEN.RX_FIFO_EMPTY`, `IRQEN.RX_FIFO_FULL`, `IRQEN.TX_FIFO_EMPTY`, `IRQEN.TX_FIFO_FULL` y `IRQEN.MAX_DROP`.
- Además, está el registro de solicitudes de interrupción. Estos son: `IRQ.RX_FIFO_EMPTY`, `IRQ.RX_FIFO_FULL`, `IRQ.TX_FIFO_EMPTY`, `IRQ.TX_FIFO_FULL` y `IRQ.MAX_DROP`. Mencionar que estos funcionan con una lógica "sticky", la cual indica que una vez el bit se ponga en alto, la única forma de bajarlo es manualmente. Esto por medio de escribir en el respectivo campo, por ejemplo `IRQ.TX_FIFO_EMPTY`, un uno.
- El "Controller" es el responsable de realizar el alineamiento en sí, usando como parámetros `CTRL.SIZE` y `CTRL.OFFSET`. Cada vez que el RX FIFO tiene datos disponibles, este módulo los toma y los envía al TX FIFO.

- En el acceso de registro, dado requisitos de sincronización, se pueden dar estados de espera antes de brindar los datos consultados. Sin embargo, una transferencia en el APB no puede tener más de 5 estados de espera, de lo contrario se considera como ilegal.
- Detección y contador de datos inválidos y errores.
- Acceso al registro de archivo, para leer información sobre las FIFO, escribir parámetros de control e interrupciones.
- Hay interfases para leer y escribir datos.

1.1 Features

A continuación se muestra una lista de capacidades que tiene el dispositivo bajo prueba.

- Leer y escribir datos de entrada con diferentes size y offset.
- Errores en la entrada indicados (se dispara una flag) con error.
- Leer y escribir registros.
- Errores en la lectura y escritura de registros.
- Interrupciones, flagged, enmascarado.
- Contador de errores, el cual se puede resetear.
- Contador del estado de parámetros de las FIFO, los cuales se pueden resetear.
- Alineaciones de salida.

2 TestPlan

A continuación se muestra el testplan planteado, este dividido en los casos de uso general y los casos de esquina.

- Casos de uso común:
  - Aleatorizar atributos de RX.
    - `md_rx_valid`. Este corresponde a un bit que indica cuándo se quiere iniciar una transferencia de un dato para que sea alineado.
    - `md_rx_data`. Corresponde al dato que se va a enviar para ordenar. Tiene 32 bits.
    - `md_rx_offset`. Indica el offset, en bytes, en el bus del `md_rx_data`, a partir de dónde los datos válidos empiezan.
    - `md_rx_size`. Representa el tamaño, en bytes, de los datos válidos en el bus del `md_rx_data`.
    - Acceder a información de los distintos registros a traves del "APB". Por ejemplo, CONTRL, STATUS, IRQEN Y IRQ.
  - Aleatorizar atributos de TX.
    - `md_tx_ready`. Bit que le indica al transmisor cuando ya se quiere iniciar a recibir datos.
- Casos de esquina:
  - Realizar transferencias ilegales. Por ejemplo, poner un `SIZE` y o `OFFSET` que no sean válidos, como un $SIZE = 4 \wedge OFFSET = 2$. Ver que esto va a resultar en un aumento del contador de transacciones inválidas, `CNT_DROP`.

En seguida se muestra otro planteamiento del Test Plan, el cual fue elaborado de manera conjunta con el profesor.

Caso de prueba general: Se van a aleatorizar los siguientes datos / funcionalidades.

- **Datos de entrada:** size, entre $[1:4]$, offset, entre $[0:3]$, dato, las combinaciones que pueda generar con 32 bits. Con respecto al size y el offset, hay 7 de las 16 posibles combinaciones que son válidas, por lo que hay que generar constraints que permitan definir si se quiere trabajar con más o menos combinaciones válidas.

- **Configurar el registro de control:** Se va a especificar el size, valor entre [1 : 4] y el del offset, valor entre [0 : 3]. Se deben implementar constraints que permitan definir qué tipo de transacciones se quieren probar, solo válidas, solo inválidas o una combinación de ambas. Además de esto, estos cambios en la configuración se deben de realizar cada [2 : 32] datos de entrada.
- **Registro de interrupciones:** Cada cuánto voy a resetear uno de los 5 bits, valores entre [0 : 4], y cada cuánto voy a configurar este registro, cada [1 : 32] transacciones.
- **Reseteo del flag de las interrupciones:** Cuando se da una interrupción, se decide si resetearlo o no. Si sí se resetea, cuánto ciclos duro en resetearlo, esto entre [1 : 5]. Además, se realiza un resetero aleatorio cada [3 : 32] ciclos. Cuando se da este reseteo, se aleatoriza cuál de los 5 bits se resetea, [0 : 4].

3 Estructura del ambiente

3.1 Diagrama del ambiente

En la Fig. 1 se observan los distintos transactores y las conexiones entre estos, para formar el ambiente de pruebas.

3.2 Paquetes de transacción

A continuación se muestran los tipos de paquetes que se van a utilizar en el ambiente de pruebas. Además, se muestran los atributos y qué transactores van a conectar.

- **trans_apb_in:** Paquete que se usa para realizar la comunicación de entrada con la interfaz APB. Este sale del agente, pasa por el driver y se conecta con la interfaz del APB.
  1. psel: APB select, permite selección del registros.
  2. penable: APB enable, señal para habilitar el bus del APB.
  3. pwrite: señal que indica una operación de escritura.
  4. paddr: Dirección de 16 bits que selecciona el registro específico a acceder. Los bits [1:0] se ignoran, forzando alineamiento a palabras de 4 bytes.
  5. pwdata: Datos de 32 bits que se escribirán en el registro seleccionado.
- **trans_apb_out:** Paquete que se usa para realizar la comunicación de salida con la interfaz APB. Este sale de la interfaz del APB, pasa por el monitor y llega al scoreboard.
  1. pready: Señal que indica que se completo la transferencia.
  2. prdata: Datos de lectura de 32 bits que se reciben de los registros.
  3. pslverr: Señal que indica si ha ocurrido un error en la transferencia
  4. psel: APB select, permite selección del registros.
  5. penable: APB enable, señal para habilitar el bus del APB.
  6. pwrite: señal que indica una operación de escritura.
  7. paddr: Dirección de 16 bits que selecciona el registro específico a acceder. Los bits [1:0] se ignoran, forzando alineamiento a palabras de 4 bytes.
  8. pwdata: Datos de 32 bits que se escribirán en el registro seleccionado.
- **trans_rx_in:** Paquete que se usa para realizar la comunicación de entrada con la interfaz MD del FIFO RX. Este sale del sequencer, pasa por el driver y se conecta con la interfaz del MD del FIFO RX.
  1. md_rx_valid: Es un solo bit, el cual se usa para indicarle al DUT que se quiere ingresar un dato al FIFO RX. Este se debe mantener en alto hasta que el md_rx_ready esté en alto.
  2. md_rx_data: Dato, de 32 bits, que se va a ingresar a la FIFO RX. Este debe mantenerse constante hasta que md_rx_ready se ponga en alto.
  3. md_rx_offset: Valor entre [0 : 3] que indica el offset, en bytes, del dato de entrada a la FIFO RX.
  4. md_rx_size: Valor entre [1 : 4] que indica el size, en bytes, del dato de entrada a la FIFO RX.
- **trans_rx_out:** Paquete que se usa para realizar la comunicación de salida con la interfaz MD del FIFO RX. Este sale de la interfaz del MD del FIFO RX, pasa por el monitor y llega al scoreboard.

1. md_rx_ready: Señal de un bit que el DUT usa para indicar que los datos que se quieren ingresar son válidos y se pueden recibir.
2. md_rx_err: Señal de un bit que el DUT levanta cuando se incumplen las posibles combinaciones de offset con size.

- **trans_tx_in:** Paquete que se usa para realizar la comunicación de entrada con la interfaz MD del FIFO TX. Este sale del sequencer, pasa por el driver y se conecta con la interfaz del MD del FIFO TX.

  1. md_tx_ready: Señal para indicar que el receptor esta listo para aceptar datos.
  2. md_tx_err: Señal que indica si hubo una condición de error en el receptor.
  3. md_tx_valid: Señal de un bit que el DUT usa para indicar que sí va a poder recibir en la FIFO el dato ordenado.

- **trans_tx_out:** Paquete que se usa para realizar la comunicación de salida con la interfaz MD del FIFO TX. Este sale de la interfaz del MD del FIFO TX, pasa por el monitor y llega al scoreboard.

  1. md_tx_valid: Señal de un bit que el DUT usa para indicar que sí va a poder recibir en la FIFO el dato ordenado.
  2. md_tx_data: Señal de 32 bits que contiene el dato de entrada del FIFO RX, pero ya alineado por el DUT.
  3. md_tx_offset: Valor entre $[0:3]$ que indica el offset, en bytes, del dato que sale del alineador.
  4. md_tx_size: Valor entre $[1:4]$ que indica el size, en bytes, del dato que sale del alineador.

3.3 Interfaces

- Se implementan tres interfaces, las cuales permiten comunicarse con los controladores de RX y de TX, además de con el archivo de registro. En estas se implementan "Modports", los cuales permiten definir a qué señales pueden acceder los transactores (el agente, el driver y el monitor). Además de esto, se implementan aserciones, las cuales permiten verificar que se cumplan algunas condiciones específicas.

3.4 Módulo APB

El módulo APB implementa la interfaz y componentes necesarios para comunicacion con el registro del sistema mediante el protocolo APB.

- Agente: Su función principal es generar transacciones de estimulo para el bus APB según las instrucciones recibidas del test. Tiene nueve modos que incluye: configuración inicial del dispositivo, envío de configuraciones válidas e inválidas, gestión de interrupciones, lectura de registros de estado y de interrupciones, y acceso a direcciones ilegales.
- Driver: Implementa el protocolo APB manejando las fases de SETUP y ACCESS. Se encarga de la temporalización correcta de las señales.
- Monitor: Transactor que observa el bus APB y captura transacciones completas cuando se detecta.

3.5 Módulo MD_Tx

- Agente: Su función principal es generar condiciones de control para la transmisión según las instrucciones recibidas del test. Tiene tres modos que incluyen: siempre listo para recibir datos sin restricciones, simulación de backpressure para modelar un sistema ocupado, e inyección controlada de errores en las transferencias.
- Driver: Implementa el protocolo de la interfaz de transmisión, controlando las señales de ready y error. Se encarga de aplicar las políticas de backpressure y errores según la configuración del agente.
- Monitor: Transactor que observa la interfaz de transmisión y captura transacciones completas cuando se detecta que tanto `md_tx_valid` como `md_tx_ready` están activos. Recopila las señales de datos y control para verificación posterior en el checker.

3.6 Módulo MD_Rx

- Agente: Transactor que genera los objetos a enviar. Este aleatoriza los objetos según instrucciones recibidas del *test*. Puede generar objetos con totalmente aleatorios, únicamente válidos o inválidos. Envía dichos objetos a través de un *mailbox* al driver y al checker.
- Driver: Transactor que comunica el ambiente con el DUT. Una vez que recibe un objeto de parte del agente, pone la señal de válido en alto, para que cuando el DUT ponga la señal de listo en alto, se pueda completar la transacción. Una vez se detecte esta en alto, se baja la señal de válido.
- Monitor: Transactor que permite recibir los datos procesados por el DUT. Una vez detecta que las señales de válido y listo están en alto, toma los datos de la interfaz y los asigna a un objeto, para luego enviarlo al checker.

3.7 Checker

Recibe paquetes del agente del APB y del RX, del monitor del RX y del TX, además de enviar los resultados al *Scoreboard*. En este se define una clase que sirve para resumir los paquetes recibidos de los distintos transactores. Además, realiza una inspección de distintos parámetros en su módulo central, los cuales se mencionan a continuación:

- Pasa de manera continua monitoreando los paquetes recibidos del APB. Si este posee la dirección de memoria correspondiente al registro de control, este va a tomar los valores del size y el offset, para así tener claro qué resultados debe esperar en la salida del TX.
- Verifica que las combinaciones de size y offset recibidas de los distintos transactores sean legales, de lo contrario, documenta dicho error.
- Tiene un modelo que simula el alineamiento del DUT, el cual, al recibir los datos del RX y con la configuración recibida del APB, permite estimar cuál va a ser el resultado que va a enviar el TX. Permite comparar las transacciones de TX con sus correspondientes de RX.
- Permite indicar el tipo de transacción que se está procesando, además de mostrar si esta es válida o inválida. También muestra el momento en el que se procesan los resultados. Además, muestra el dato recibido de TX (junto con su size y offset), el dato estimado con el golden reference y el o los datos requeridos de RX para generar el TX.

3.8 Scoreboard

Este transactor sirve para recolectar los datos recibidos del checker y almacenarlos de manera que puedan ser útiles. En seguida se mencionan algunas de las funciones específicas que posee:

- Crear y rellenar con los resultados del checker un archivo de tipo .csv.
- Al finalizar la simulación, genera una impresión con algunas métricas resultantes del ambiente de pruebas.
- Posee algunas métricas, como la cantidad de pruebas que se realizan, cuántas de estas se pasaron o no y el número de transacciones inválidas detectadas.

3.9 Test

La clase Test es la encargada de orquestar la ejecución de las pruebas en el ambiente de verificación. Coordina los diferentes agentes para realizar una serie de pruebas predefinidas que validan el funcionamiento del DUT. **Función principal:** Configurar y ejecutar una secuencia de pruebas que incluyen la configuración del dispositivo a través del bus APB, la generación de transacciones de entrada (MD_RX) y la verificación de las transmisiones de salida (MD_TX), pueden generarse diferentes combinaciones para estresar el sistema y cubrir diferentes caso ya sean de esquina o generales.

- **Prueba 1:** Configuración inicial del Aligner a través de APB, seguida de transacciones válidas en MD_RX y verificación de salida en MD_TX.
- **Prueba 2:** Configuración APB con múltiples configuraciones válidas, transacciones válidas en MD_RX y verificación de salida.
- **Prueba 3:** Configuración APB con configuraciones válidas, transacciones inválidas en MD_RX y verificación de manejo de errores.

- **Prueba 4:** Configuración APB con configuraciones válidas, transacciones válidas en MD_RX y condiciones de backpressure en MD_TX.
- **Prueba 5:** Configuración APB con configuraciones válidas, transacciones de llenado aleatorio en MD_RX y verificación de salida.
- **Prueba 6:** Configuración APB con escrituras en el registro de interrupciones (IRQ) y verificación del manejo de interrupciones.

Utiliza mailboxes para comunicarse con los agentes y un environment que contiene todos los componentes del ambiente de verificación. Sincroniza las pruebas mediante esperas de ciclos de reloj y finaliza con la generación de reportes del scoreboard.

4 Comandos de compilación

Para correr el proyecto en el servidor, luego de cargar las herramientas de Synopsys, se debe de usar el código mostrado en 1 para realizar la compilación. Mientras que para correr el archivo ejecutable, se usa lo que se observa en 2.

Listing 1: Alias para compilar el testbench en VCS

```
vcs -Mupdate tb.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
```

Listing 2: Alias para compilar el testbench en VCS

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

5 Resultados obtenidos

Algunos de los resultados obtenidos se describen brevemente a continuación.

- Ejemplo 1: Este resultado corresponde a una configuración válida del DUT donde el Aligner opera con parámetros de control SIZE=2 y OFFSET=2. El checker verificó que la palabra transmitida por la interfaz TX coincidiera con la referencia dorada (*golden expected*) reconstruida a partir de los bytes recibidos por la interfaz `RX`.

  En este caso, los bytes provinieron de dos transferencias consecutivas:

  1. La primera (`RX#27`) aportó el byte `0x8C`.
  2. La segunda (`RX#28`) aportó el byte `0xE3`.

  El scoreboard reporta que el valor transmitido (`0xE38C0000`) coincide exactamente con el esperado, por lo que la verificación de alineamiento de datos resultó PASSED. Esto demuestra que el DUT realizó correctamente la combinación y alineación de bytes provenientes de distintas transferencias RX conforme a la configuración actual.

Listing 3: Resultado del scoreboard para el test #31

```
============================================================
SCOREBOARD - Resultado #31
============================================================
Timestamp: 1715000
Test: Data Alignment Verification
Resultado: PASSED
Secuencias: RX=27 TX=31
------------------------------------------------------------
Config: SIZE=2, OFFSET=2
RX: data=0x12588c27, offset=1, size=1
TX: data=0xe38c0000, offset=2, size=2
Contribuyentes RX: (RX#27: data=0x12588c27, off=1, size=1, bytes=[1:0x8c]) (RX#28: data=0x7c9b42e3,
    ↪ off=0, size=4, bytes=[0:0xe3])
Golden esperado: 0xe38c0000
============================================================
T=1715000 [Scoreboard] Test #31 PASSED: Data Alignment Verification
```

- Ejemplo 2: En este caso se presenta otra verificación de alineamiento de datos con una configuración válida del registro de control (SIZE=1, OFFSET=2). El checker detectó que el dato transmitido por la interfaz TX 0x003C0000) coincide con el valor reconstruido por el modelo dorado (*golden expected*), a partir de los bytes observados en la interfaz RX .

  Durante esta transacción, el único byte relevante provino de la transferencia RX#85 , cuyo valor fue 0x3C . El DUT alineó correctamente ese byte en la posición indicada por el offset configurado, completando la palabra de salida de 32 bits con ceros en el resto de los campos.

  El resultado PASSED confirma que el mecanismo de alineación implementado por el DUT conserva la correspondencia entre los datos recibidos y transmitidos aun con SIZE=1, validando el funcionamiento del control de tamaño y desplazamiento en condiciones normales.

Listing 4: Scoreboard - Resultado #103

```
Resultado del scoreboard para el test #103
---
Timestamp: 3995000
Test: Data Alignment Verification
Resultado: / PASSED
Secuencias: RX=85 TX=103
---
Config: SIZE=1, OFFSET=2
RX: data=0x69003c9f, offset=0, size=2
TX: data=0x003c0000, offset=2, size=1
Contribuyentes RX: (RX#85: data=0x69003c9f, off=0, size=2, bytes=[1:0x3c])
Golden esperado: 0x003c0000
---
T=3995000 [Scoreboard] / Test #103 PASSED: Data Alignment Verification
```

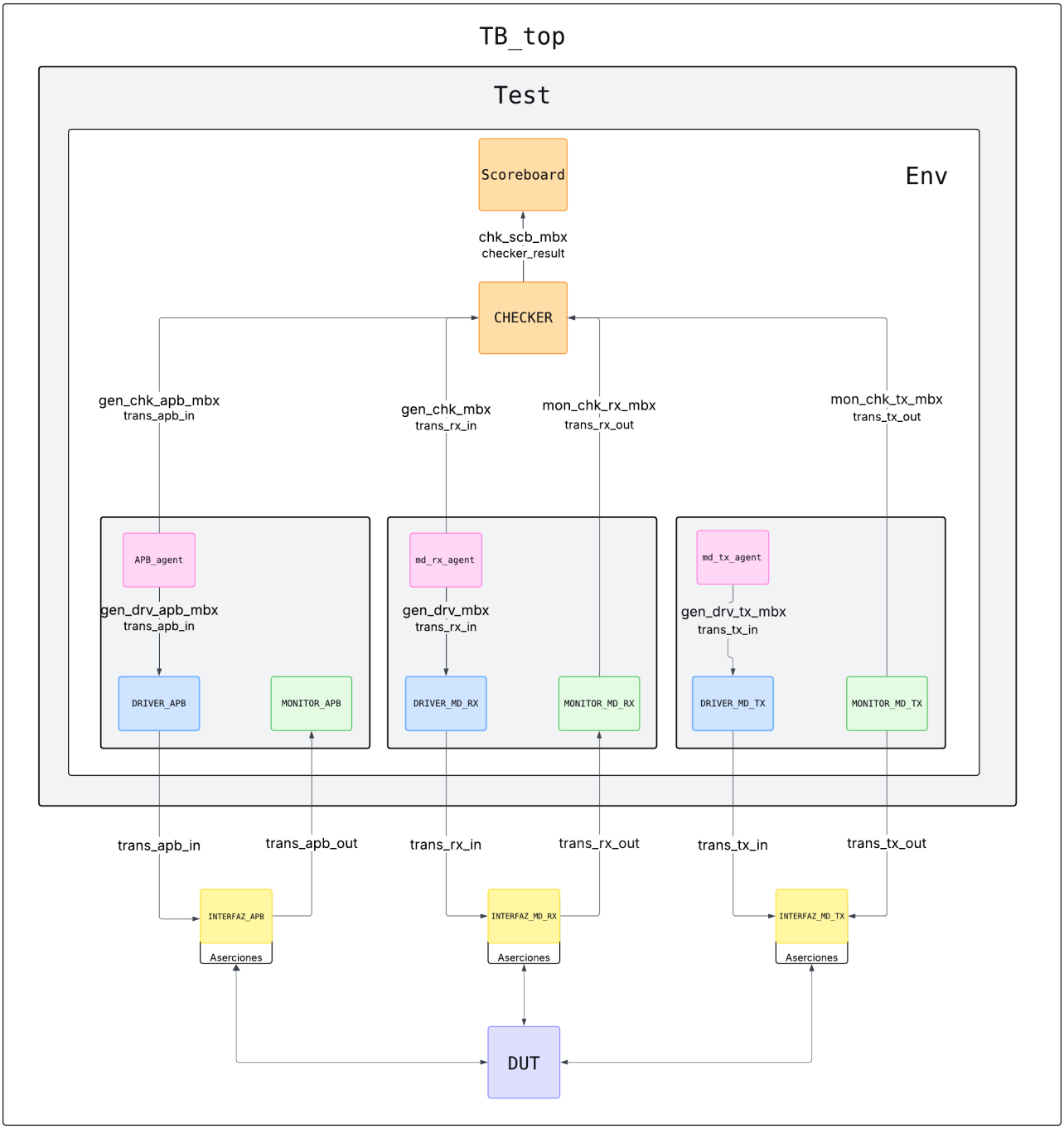

Fig. 1: Diagrama de módulos (transactores), conexiones entre estos y paquetes que se usan para comunicarse entre estos.