Modelación Funcional de Bus (Bus Functional Model)

Nota Técnica 09 Cristian Sisterna

Introducción

BFM es una sigla muy usada en el ámbito de diseño de HDL-FPGA, sin embargo no todos saben que es ni a que se refiere, otros diseñadores se dedican a escribir BFMs por lo que saben muy bien lo que es. Bueno, este blog es para los que pertenecen al grupo que dice “Qué es eso?” cuando escucha ‘BFM’.

Básicamente como su nombre lo indica, BFM es un *modelo* de simulación simplificado que describe correctamente la *funcionalidad* de I/O *bus* de un sistema normalmente complejo, sin inmiscuirse con el comportamiento interno de ese componente.

Este modelo se usa dentro del test bench como estimulo (stimulus) del sistema que se quiere testear (DUT), permitiendo en forma directa una comunicación entre el DUT y el sistema externo con el que se quiere interactuar, evitando de esta manera tener que escribir un muy complejo test bench. La siguiente figura muestra el sistema de simulación cuando se usa un BFM.

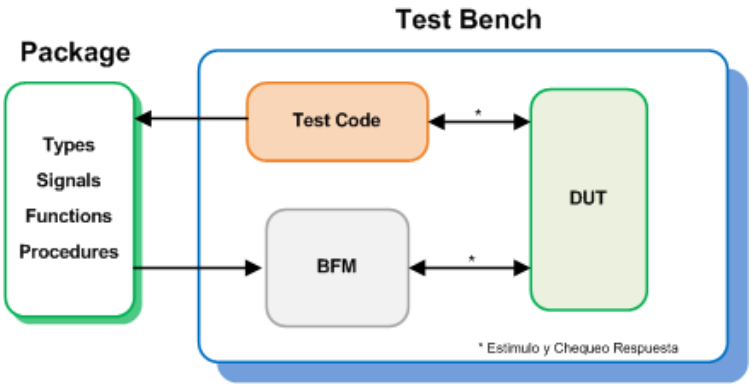

Figura 1 – Esquema de simulación usando BFM

Explico ahora los distintos componentes de la Figura 1:

Test Code: es el código del test bench en sí, escrito en VHDL. En este parte del test bench se escribe el código VHDL para estimular el DUT, principalmente haciendo uso de los procedimientos que modelan algún ciclo de I/O del componente BFM instanciado en el test bench.

BFM: es el código BFM del sistema que interactúa con el DUT. Este código es provisto por el fabricante del sistema/memoria o escrito por alguno de nosotros. Dentro del test bench se crea una declaración de este componente y su respectiva instanciación. Actúa como generación de estimulo para el DUT al proveer un preciso estimulo del protocolo de comunicación, ciclo por ciclo en funcionalidad y tiempo. Al mismo tiempo el BFM acepta comandos para seleccionar la funcionalidad a ser modelada. Estos comandos son normalmente asignados a alguna señal en el test bench, declarada en el paquete, y ejecutados por el procedimiento respectivo del BFM. Este bloque básicamente se codifica como una simple maquina de estados, cuyos estados por ejemplo pueden ser *read, write, idle*. Y de acuerdo a lo que el test code 'solicita' la máquina de estados pasa al respectivo estado y activa las I/O necesarias del sistema modelado (BFM).

DUT: es el device under test que queremos depurar.

Package: es la unidad de diseño que contiene todas las declaraciones y descripciones de las señales, funciones, procedimientos relacionadas al test bench y al BFM. Actúa como medio de comunicación entre las señales del test bench y los procedimientos/funciones del BFM.

El símbolo * lo uso para detallar que el código de test y el BFM 'pueden' (no es siempre el caso) proveer no solo el estimulo al DUT, sino también chequear la respuesta del DUT a cierto estimulo. Esta funcionalidad es opcional, sin embargo es *altamente aconsejada* para automatizar el test y proveer un rápido resultado del comportamiento del DUT bajo determinado estimulo.

Quién escribe un BFM?

Normalmente para componentes complejos, por ejemplo memorias DDR, Flash, dispositivos PCI, PCIe, etc; el fabricante ofrece un modelo BFM para que el diseñador pueda verificar que el protocolo que ha diseñado interactúa correctamente con el componente con el que quiere comunicarse.

Ahora, bien el BFM, es en sí un modelo escrito en VHDL por un ingeniero de la empresa fabricante, por lo que el modela será tan bueno como el ingeniero que lo escriba.

Debo aclarar que hay algunas empresas que escriben sus propios BFMs a fin de facilitar la simulación de componentes, normalmente varios, que tienen protocolos de comunicación complejos. En algunas de estas empresas tuve la oportunidad de escribir y usar algunos BFModels, tarea que al principio era linda, pero luego se transformó en tediosa… ☺

Porque usar BFM?

El proceso de prueba (simulación) a veces es una tarea compleja que involucra muchas horas de escritura del código (test bench) y otras más de correr la simulación. Normalmente más de la mitad del tiempo de un proyecto involucra la simulación del mismo, y más se incrementa este tiempo cuando la complejidad del proyecto crece. Por ello es necesario usar métodos de prueba (test) más eficientes y que sigan asegurando un diseño libre de errores.

El uso de BFM habilita el test de complejos protocolos y funciones con un gran nivel de abstracción de la funcionalidad interna del componente simulado, menos instrucciones a simular, preciso timing de la interface, todo lo que resulta en un test más robusto, completo y sobre todo más rápido de ejecutar. Esta última es una característica que se destaca cuando se comparan los tiempos de simulación de sistemas complejos, IP cores, etc., que usualmente toman un largo tiempo en simular.

Características de un BFM

- Provee un medio para modelar la *interface* del componente que se está modelando, SIN importar la estructura interna del componente o si el componente en si tiene un microprocesador, una o varias FSMs, etc.
- Provee información de todos los tiempos del protocolo de comunicación de I/Os.
- Provee chequeo funcional para verificar el correcto funcionamiento del protocolo (handshaking).
- Normalmente son escritas usando las instrucciones típicamente usadas en Test Bench. Paquetes, funciones, procedimientos, registros y tipos son comúnmente encontradas en BFMs.
- El alto nivel de abstracción que VHDL ofrece permite modelar complejos sistemas, en funcionalidad y tiempo, abstrayendo al diseñador de la complejidad interna del sistema modelado.

Ejemplo Flujo de Uso de BFM

La siguiente figura detalla el procedimiento que se debería seguir cuando se usa un BFM.

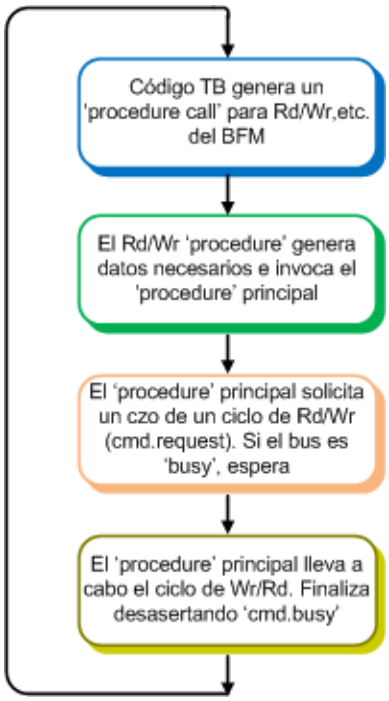

Figura 2 - Pasos a seguir para usar los procedimientos del BFM

Paso 1: Dependiendo de la funcionalidad del BFM que se desea invocar, en el código del test bench se genera un llamado al procedimiento respectivo asignando los valores necesarios. Por ejemplo para comenzar un ciclo de escritura (Wr), habrá que asignar a la senal respectiva un 'valor' que entienda el procedimiento que se desea invocar.

Paso 2: El procedimiento invocado seguramente necesita generar algún dato o señal antes de invocar al procedimiento principal.

Paso 3: El procedimiento principal primero solicita la disponibilidad del bus de I/O, normalmente por medio de alguna señal de pedido (request) de bus. En caso que el bus no este disponible, por ejemplo porque todavía se está ejecutando algún otro procedimiento, deberá esperar hasta que el bus esté disponible.

Paso 4: Las I/O del BFM son controladas por el procedimiento principal con el comportamiento deseado (Wr/Rd, etc.) y con el respectivo timing. Una vez finalizado el ciclo, el procedimiento avisa por medio de la misma senal de ocupado (busy) desacertándola. Permitiendo así el recomienzo del ciclo.

Escritura de BFMs

Uno de los principales usos de BFMs es por ejemplo para representar ciclos de Wr/Rd del bus del componente que se desea modelar. Estos ciclos normalmente se describen mediante el uso de funciones y procedimientos definidos en un paquete para permitir la accesibilidad no solo del BFM pero también del test bench y del código VHDL en sí.

Los ciclos de Wr/Rd son fácilmente descriptos mediante el uso de procedimientos (*procedures*) que modelan perfectamente el comportamiento de los ciclos de Wr/Rd ya sea sobre memoria o sobre I/O, no solo funcionalmente sino también con los tiempos respectivos. La BFM debe ser capaz de leer un comando, por ejemplo, y devolver la respuesta a ese comando, de ese modo se prueba tanto la escritura como la lectura de mi componente sobre el componente modelado con BFM. Al usar procedimientos y ser un modelo de un sistema *toda* la potencia del lenguaje VHDL puede ser usada al escribir el BFM.

La entidad/arquitectura del BFM debe ser instanciada junto con nuestro componente (DUT) dentro del test bench. Mínima configuración del BFM a veces es necesaria al comienzo del TB, hasta que pueda comenzar el handshaking entre mi componente y el componente BFM.

Ejemplo del código escrito para una BFM.

Declaraciones en el paquete:

```
-- type and signal declarations --
type cycle_op is (idle, nop, read, write);

type instruction_op is record
   bus_cycle    : cycle_op;
   addr_bus     : std_logic_vector(15 downto 0);
   data_bus     : std_logic_vector(15 downto 0);
   rd_data      : std_logic_vector(15 downto 0);
   bus_request  : std_logic;
   busy         : std_logic;
end record;

signal cmd      : instruction_op :=  (idle,
                                      (others => '0'),
                                      (others => '0'),
                                      (others => 'Z'),
                                      '0',
                                      'Z');
```

En este ejemplo el registro (record) declara las señales a ser usadas para generar el estimulo de las I/O, como asi también información a pasar al procedimiento que debe ejecutar el ciclo seleccionado. Es decir se declara:

- Tipo del ciclo (idle, nop, read, write)
- Dirección a ser utilizada en el bus de direcciones
- Dato a ser escrito si es un ciclo de escritura, o dato que se espera leer si es un ciclo de lectura
- Data leído
- Señal de pedido de bus (bus_request). Se activa para solicitar comienzo de un ciclo.
- Señal de bus ocupado (busy). Se activa durante el proceso de ejecución de algún ciclo.

Un procedimiento de un ciclo escritura puede ser escrito tal como se detalla a continuación:

```
procedure wr_cycle (
   constant p_addr_bus : in std_logic_vector(15 downto 0);
   constant p_data_bus : in std_logic_vector(15 downto 0);
   signal p_cmd_op     : inout instruction_op
                     ) is
begin
    main_cycle(write,p_addr_bus, p_data_bus,p_cmd_op);
end wr_cycle;
```

Donde el procedimiento main_cycle es el siguiente:

```
procedure main_cycle(
   constant p_cycle    : in cycle_op;
   constant p_addr_bus : in std_logic_vector(15 downto 0);
   constant p_data_bus : in std_logic_vector(15 downto 0);
   signal p_cmd_op     : inout instruction_op
                     ) is
begin
   p_cmd_op.request <= '0';
   if p_cmd_op.busy /= '0' then
      wait on p_cmd_op.busy until p_cmd_op.busy = '0';
   end if;
   p_cmd_op.request   <= '1';
   p_cmd_op.bus_cycle <= p_cycle;
   p_cmd_op.addr      <= p_addr;
   p_cmd_op.data      <= p_data;
   if p_cmd_op.busy = '0' then
      wait on p_cmd_op.busy until p_cmd_op.busy = '1';
   end if;
   p_cmd_op.request   <= '0';
end;
```

Este procedimiento lleva a cabo las tareas de 'handshaking', usando las señales *busy* y *request*, y de comunicación entre el test code y el BFM.

Ahora, cómo invoco las distintas funciones del BFM dentro del test bench? La siguiente línea de código, escrita en el test bench, muestra como se puede invocar el procedimiento del ciclo de escritura:

```
wr_cycle(x"0000",x"05ab",cmd_op);
```

Entonces lo que pasa es lo siguiente:

- El procedimiento *wr_cycle* esta declarado en un paquete (ver figura 1)
- El procedimiento *wr_cycle* le pasa los parámetros necesarios al procedimiento *main_cycle*
- El procedimiento *main_cycle* lleva a cabo el handshaking, y pasa el comando (*cmd_op*) a la máquina de estados (FSM, Finite State Machine) de la BFM
- Al ser un comando de escritura la máquina de estado pasa al estado escritura donde procederá a activar las señales correspondientes del ciclo de escritura, y habilitar las de espera que el proceso se cumpla (*ready*). Este último paso es el que realmente 've' el DUT.

La siguiente figura detalla la modificación de la primer figura al ejecutarse un ciclo de escritura, mostrando gráficamente los pasos recién enumerados.

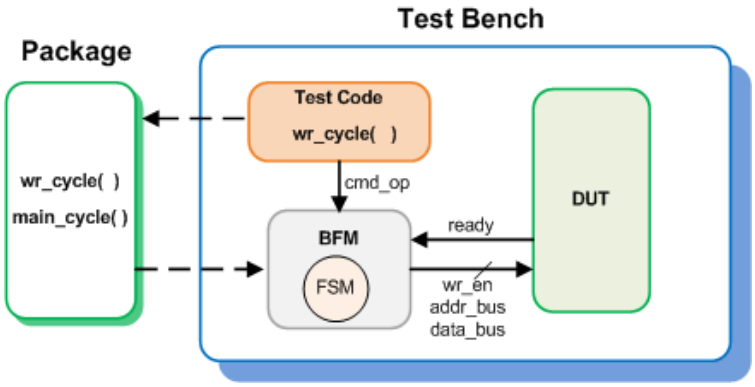

Figura 3 - Detalle de los procedimientos y código ejecutados durante un ciclo de escritura

Unos puntos a considerar de la Figura 3:

- La línea punteada entre el bloque Test Code y Package se debe a que en realidad el procedimiento *wr_cycle* está definido en el paquete pero, como bien sabemos, cuando se compila el código de Test Code, el procedimiento definido en el paquete es 'insertado' en el lugar en que éste es invocado

- Lo mismo sucede con el procedimiento *main_cycle*. El cual es también ejecutado desde Test Code.
- Tal como se puede deducir del código de los procedimientos detallados anteriormente (*wr_cycle, main_cycle*), la señal *cmd_op*, definida como un registro, contiene toda la información necesaria para que la FSM del componente BFM genere el correspondiente ciclo de escritura en las líneas de I/O de este bloque (recordar que el componente BFM está instanciado dentro del test bench).

Espero que este artículo sirva para, por lo menos, tener una idea de la pregunta: Qué es un BFM?