VHDL 4: simulación con testbenches

Hipólito Guzmán Miranda
Departamento de Ingeniería Electrónica
Universidad de Sevilla
hguzman@us.es

Acknowledgement to Fernando Muñoz,
Universidad de Sevilla

Contexto docente

BT01: Introducción a VHDL y su aplicación en FPGAs

- Tema 1: Estructura de un fichero VHDL
- Tema 2: Describiendo la funcionalidad
- Tema 3: Diseño de circuitos síncronos
- Tema 4: Simulación con testbenches

Conocimientos previos requeridos:

- Electrónica digital básica (puertas lógicas y biestables)
- Temas 1, 2 y 3: Estructura de un fichero VHDL, Describiendo la funcionalidad, Diseño de circuitos síncronos

Objetivos de aprendizaje

- Aprender qué es un testbench en VHDL
- Aprender a diseñar testbenches sencillos
- Conocer algunas funcionalidades del lenguaje y sus librerías que, a pesar de no ser sintetizables, son útiles para simular

Contenido

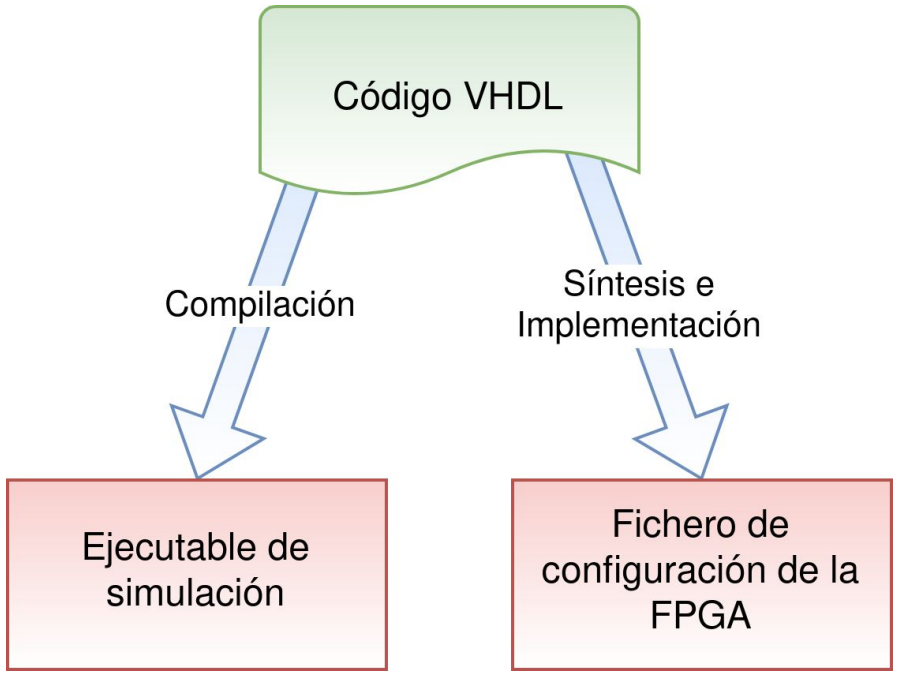
- Introducción
- Diseño de un testbench
- Ejemplo
- Sentencia wait
- Acceso a ficheros

¿Cómo saber que mi diseño funciona?

- Probarlo en la FPGA es divertido pero no deja de ser una ‘caja negra’
  - No tenemos visibilidad de los estados internos
  - Si algo no funciona como esperamos no tenemos mucha capacidad de depuración
- Siempre es recomendable simular bien antes de probar en la FPGA
  - A la larga se ahorra tiempo

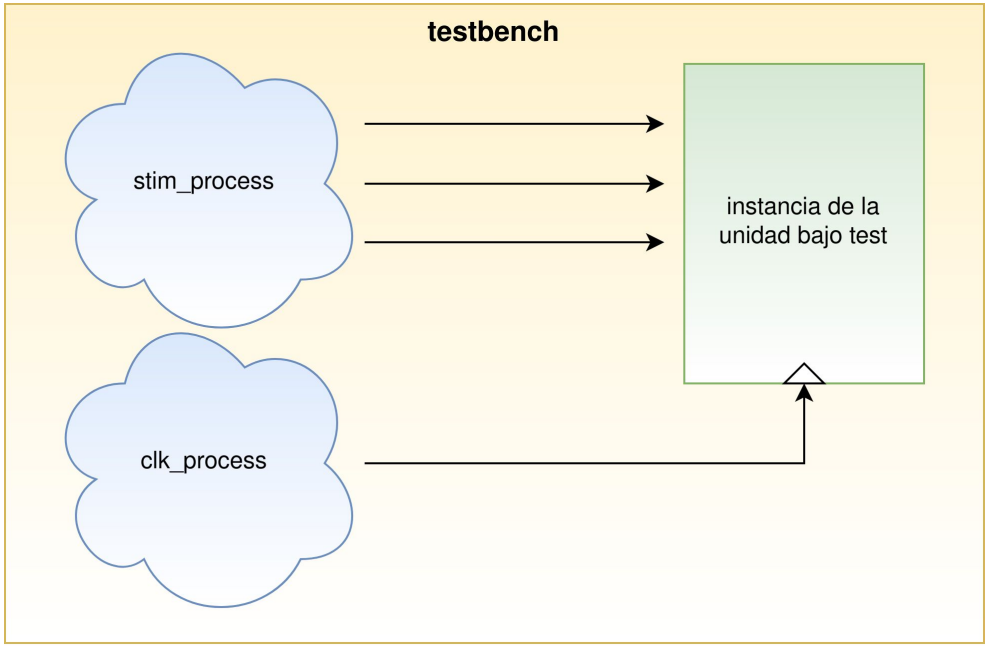
¿Simulación?

- El VHDL no sólo se sintetiza, también se compila
  - Se sintetiza cuando implementamos para configurar la FPGA
  - Se compila cuando queremos simularlo para comprobar su funcionamiento



¿Simulación?

- Para simular un diseño es necesario generar estímulos para todas sus entradas
  - Si no, todas estáran a `'U'` (u otros valores sin inicializar, dependiendo del tipo de dato)
- La buena noticia es que esto se hace también usando el lenguaje VHDL, describiendo el *entorno* en el que se encuentra el circuito
- Esto se conoce como crear un testbench (o banco de pruebas)



Partimos de un diseño ya realizado que queramos simular

Puede ser un diseño pequeño o grande, que incluya submódulos o no, es indiferente


```
entity contador is
  generic (N: integer := 10);
  port (
    clk    : in  std_ulogic;
    rst    : in  std_ulogic;
    ena    : in  std_ulogic;
    cuenta : out unsigned(N-1 downto 0);
    sat    : out std_ulogic
);
end contador;
```

Creamos un `entity` sin `ports`

- Normalmente se le llama `tb_nombreentidad` o `nombreentidad_tb`
  - Donde *`nombreentidad`* es el nombre de la entidad que queremos simular
- Esto sirve para organizar el código
  - Y ayuda a algunas herramientas que auto-descubren testbenches como VUnit

```
library ieee;
use ieee.std_logic_1164.all;

entity tb_contador is
end tb_contador;

architecture testbench of tb_contador is
  ...
begin
  ...
end testbench;
```

Diseño de un testbench

Declaramos señales

- Necesitamos un signal por cada uno de los puertos del diseño
- Para las entradas podemos usar valores iniciales

```
architecture testbench of tb_contador is

  -- Entradas
  signal rst : std_ulogic := '0';
  signal clk : std_ulogic := '0';
  signal ena : std_ulogic := '0';

  -- Salidas
  signal sat : std_ulogic;
  signal cuenta : unsigned(9 downto 0);

begin
  ...
```

Instanciamos la entidad bajo test

- Podemos instanciarla como componente o como entidad
  - Si vamos a instanciarla como componente, hay que declarar el `component` antes del begin de la arquitectura
- UUT = “Unit Under Test”
  - También se suele utilizar “MUT” (Module Under Test), “DUT” (Device Under Test), o simplemente `<nombreentidad>_inst`)

```
uut: entity work.contador
 generic map (N => 10)
 port map (
   rst    => rst,
   clk    => clk,
   ena    => ena,
   sat    => sat,
   cuenta => cuenta
 );
```

Asignamos estímulos a las entradas

- En testbenches sencillos, se utilizan dos procesos:
  - `clk_process`: para generar un reloj independiente (free-running clock)
  - `stim_process`: para el resto de los estímulos
- Estos procesos no tienen lista de sensibilidad
  - En su lugar, controlan los tiempos utilizando sentencias `wait`
- Al no tener lista de sensibilidad, siempre que se esté fuera de ellos se vuelve a entrar
  - Las sentencias `wait` evitan que se produzcan bucles infinitos

Sentencia wait

- Como su nombre indica, realiza una espera
- Las asignaciones a señales se hacen efectivas cuando se llega a un `wait`
- Generalmente no es sintetizable
- No es compatible con tener lista de sensibilidad

Sentencia	Efecto	Ejemplos

`wait for <tiempo>;`	Espera el tiempo indicado	`wait for 10 ns;`
		`wait for clk_period;`
		`wait for 10*clk_period;`

`wait on <lista sensibilidad>;`	Espera a que cambie alguna de las señales de la lista	`wait on salida1, salida2;` `wait on request;`
`wait until <condición>;`	Espera a que cambie alguna de las señales de la condición **y** se cumpla la condición	`wait until rst = '0';` `wait until rising_edge(clk);`
`wait;`	Espera indefinidamente (nunca sale del wait)	`wait;`

Procesos de estímulo y reloj

```
clk_process:
 begin
   clk <= '0';
   wait for clk_period;
   clk <= '1';
   wait for clk_period;
 end process;
```

(antes del begin de la arquitectura:

```
constant clk_process : time := 10 ns;
```

, o si no, cambiamos `clk_period` por `10 ns` –o el tiempo que queramos– en el process)

```
stim_process:
 begin
   rst <= '1';
   wait for 15 ns;
   rst <= '0';
   wait for 20 ns;
   ena <= '1';
   wait for 100 ns;
   ena <= '0';
   wait;  -- wait forever
 end process;
```

Ejemplo: contador

- Hagamos un testbench para el contador que hicimos en el tema anterior
  - Con `ena` y `updown`
- Nuestro testbench será una entidad sin ports que contendrá:
  - La declaración de tantas señales como ports tenga el contador
  - La instancia del contador, conectado a dichas señales
  - Un proceso para generar el reloj (`clk_process`)
  - Un proceso para generar el resto de estímulos (`stim_process`)

Otras consideraciones

- Un testbench, al igual que cualquier otra entidad VHDL, puede incluir más de una instancia de entidad
- A veces se incluyen modelos de simulación del hardware externo, u otras entidades que nos ayudan a la verificación
- Un testbench es un programa!

Acceso a ficheros

- Las librerías básicas contienen funciones simples para acceso a ficheros
  - Es posible leer estímulos de ficheros
  - Es posible guardar resultados en ficheros
- Paquete `textio` de la librería `std`
  - Para tipos de datos del estándar

```
library std;
use std.textio.all;
```

- Paquete `std_logic_textio` de la librería `ieee`
  - Para tipos de datos tipo `std_[u]logic[_vector]`

```
library ieee;
use ieee.std_logic_textio.all;
```

- El acceso a ficheros no es sintetizable

Acceso a ficheros

FILE y LINE

- VHDL sólo permite leer y escribir líneas completas en los ficheros
- Un **FILE** es un objeto especial (no es ni `signal` ni `variable`)

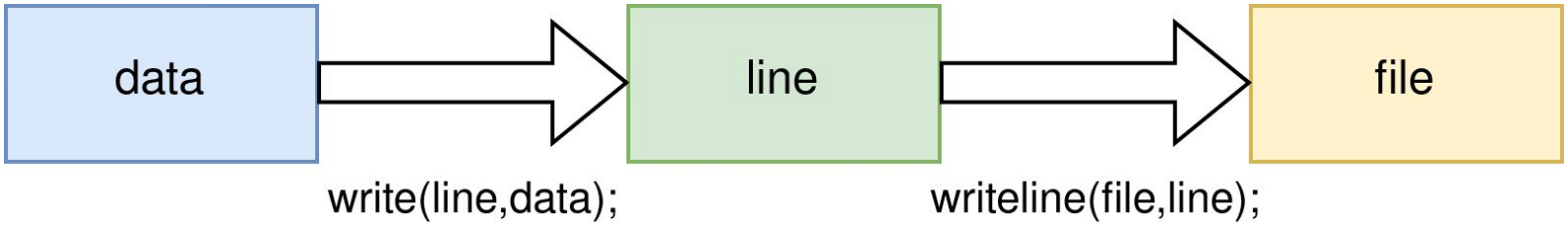
```
FILE fin  : TEXT open READ_MODE  is “i.txt";
FILE fout : TEXT open WRITE_MODE is “o.txt";
```

- Existe un tipo de dato, **LINE**, que representa una línea

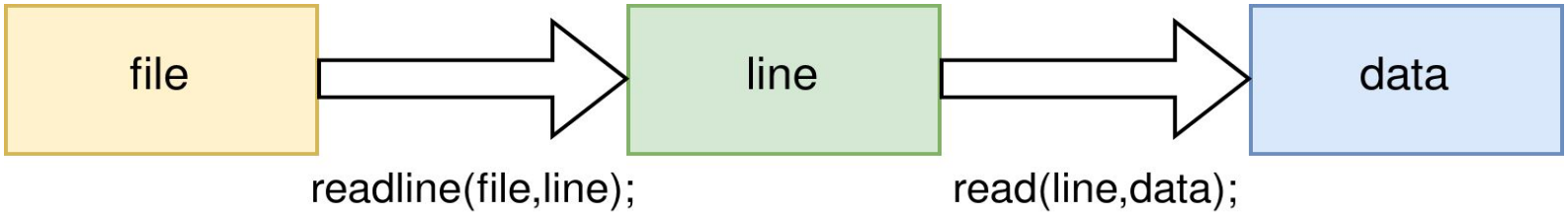
```
variable myline: LINE;
```

Lecturas y escrituras

- Para escribir, primero escribimos en la línea y cuando la tenemos preparada la volcamos al fichero



- Para leer, leemos la línea completa y de la línea sacamos los campos de uno en uno



Ejemplo

```
process (load)
  variable lineain, lineaout: line;
  variable campo1, campo2: integer;
begin
  if (load = '1') then
    readline(fin, lineain);
    read(lineain, campo1);
    read(lineain, campo2);
    entrada1 <= campo1;
    entrada2 <= campo2;
  end if;
  campo1 := conv_integer(senal);
  write(lineaout, campo1);
  writeline(fout, lineaout);
end process;
```

También existen otras funciones como `hwrite`, `owrite` y `bwrite` para escribir en hexadecimal, octal y binario, respectivamente

Conclusiones

- En simulación, podemos hacer muchas cosas que no son VHDL sintetizable
- Un testbench VHDL es un programa
  - Se compila y se ejecuta
  - Simula el funcionamiento concurrente de nuestro testbench + circuito
  - El simulador, puede permitir la ejecución paso a paso
  - Se puede comunicar de una manera primitiva con otros programas a través del uso de ficheros

Resultados de aprendizaje

- Distinguir cuándo un código VHDL se sintetiza y cuándo se compila
- Ser capaz de realizar un testbench sencillo en VHDL para un `entity` del que conocemos su interfaz (secciones `generic` y `port`)
- Entender el funcionamiento de la sentencia `wait`
- Conocer la posibilidad de realizar accesos a ficheros en testbenches