Escuela Politécnica Superior,
Grado en Informática

Diseño Automático de Sistemas

Prof: Ignacio Aznárez Ramos

Práctica 3:

Verificación de sistemas mediante simulación con archivos I/O

UNIVERSIDAD
NEBRIJA

UNIVERSIDAD NEBRIJA

1. Introducción

1.1 Presentación

Esta práctica consiste en la verificación del sistema realizado en la práctica 1 de una forma automática. Para esto se va a trabajar con lectura y escritura en archivos de texto, así como con la instrucción assert para verificar los resultados.

1.2 Funcionamiento

Para hacer esto se va a definir un fichero testbench que haga uso del paquete *std.textio.all* de la librería *std.* Mediante instrucciones de este paquete se va a poder leer y escribir en ficheros de texto.

En un fichero de inputs.txt se van a definir los valores de entrada de BTNC y rst_n, así como el valor esperado de la salida LED y un marcador de tiempo a esperar hasta la siguiente instrucción. De forma que se leerán dichas entradas, se comprobará que el valor del fichero LED es el esperado y se esperará el tiempo necesario hasta la siguiente instrucción.

1.3 Material provisto

Se ofrece el fichero de valores de entrada a completar. En este solo se ha indicado una línea inicial.

1.4 Evaluación y entrega

Estas prácticas van a ser realizadas en grupos de 2 personas. Estos se mantendrán durante el transcurso de las cinco prácticas. La entrega se debe realizar **tras dos semanas de haber realizado el laboratorio**.

En esta entrega se deben incluir:

- Los ficheros .vhd modificados o creados de acuerdo a lo pedido. Por lo tanto el testbench a realizar.

- Un breve informe en el que se describa brevemente la solución implementada. Incluir descripción de los casos de uso que se han tenido que tener en cuenta. En la práctica 1 estos no estaban incluidos en la mayoría de los grupos.

- Si la práctica 1 **no funcionaba**, como se notificó en la entrega de la misma, se espera un test exhaustivo del funcionamiento y la subsanación de los errores, quedando reflejado en la memoria los errores encontrados (es decir test fallando y mostrando los errores), cómo se han arreglado y subir la práctica sin errores.

Este hito se corresponde con un **30%** del valor total de la nota de prácticas.

2. Testbench a implementar

2.1 Introducción

El archivo testbench a implementar sigue en grandes rasgos la estructura del resto de testbench realizados hasta ahora. Solo habiendo cambios en el proceso interno. Por lo tanto se puede construir sobre el testbench ofrecido en la práctica 1.

Se va a utilizar un fichero de texto para generar las entradas y salidas esperadas en el sistema de la práctica 1 (top_practica1.vhd). Específicamente se van a leer las entradas (rst_n y BTNC), el valor esperado del LED, y el tiempo a esperar hasta la próxima instrucción.

El formato del fichero inputs.txt está compuesto por columnas separadas por un espacio: la primera columna contiene el delay (formato *time* de VHDL), la segunda contiene el valor de rst_n (std_logic), la tercera contiene el valor de BTNC (std_logic), y la cuarta columna contiene el valor esperado del LED (std_logic). Finalmente una quinta columna contiene comentarios precedidos por el símbolo almohadilla (#).

Se debe escribir en un fichero output.txt los casos probados, incluyendo el tiempo de delay, las entradas y salidas, comentarios o errores si los hubiera. Esta información y su formato está libre a interpretación pero se incluye a continuación una posible impresión en caso de error en el primer input y no haber más entradas a continuación.

```
Simulation of top_practica1.vhd
Time : 1000000000 ps; rst_n : 0; BTNC: 0;
ERROR: Expected LED to be 0 actual value '1'
Finished simulation
```

2.2 Implementación

Para realizar la implementación del testbench va a ser necesario agregar el paquete *std.textio.all* de la librería *std.* Estos será necesario incluirlos al comienzo del código.

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

Tras la inclusión de los paquetes se define la entidad y la componente el comienzo de la arquitectura como en casos anteriores. A continuación se definen las señales internas necesarias. Junto a las necesarias para conectar con la unidad a testear (UUT), y a las constantes que puedan hacer falta para definir el reloj. Se añaden dos señales internas más, los filehanders que el sistema asigna a los archivos de texto que vamos a usar.

```
-- File handler signals
file file_INPUT : text;
file file_OUTPUT : text;
```

Dentro del funcionamiento de la arquitectura, tras una instrucción begin, se define la Unit Under Test (UUT) y el funcionamiento cíclico que usa el reloj. Finalmente, se define el proceso en el que se realizará la lectura, comprobación y escritura de los datos.

2.2.1 Definición de variables.

Además de las señales y las constantes, existe un tercer tipo de objeto: las variables. Pueden ser declaradas antes del begin de la architecture y/o antes del begin del process, en su declaración se les puede asignar un valor por defecto. Se les asigna un valor mediante el operando :=. Las variables NO representan conexiones o estados de memoria.

Estas variables van a ser fundamentales para este proceso, pues vais a necesitar usarlas como buffer para las líneas o variables leídas. Digamos que se van a querer leer dos valores, un time y un integer, de cada línea de un archivo. Para esto se necesitan las siguientes variables:

```
process is
 variable v_ILINE : line;
 variable estado : file_open_status;
 variable v_RST : integer;
 variable v_TIME : time;
```

La variable estado que se ha añadido es de un tipo especial y va a ser usada para comprobar que un archivo se abrió correctamente. **Tened en cuenta que para vuestra implementación van a hacer falta más variables.**

2.2.2 Apertura de ficheros

Para abrir ficheros se va a usar la instrucción file_open. Esta instrucción va a tomar cuatro valores de entrada:

- estado, donde se va a indicar si este se abrió correctamente
- file_INPUT, el handler que marcará este fichero abierto
- “input.txt”, la dirección del archivo a abrir. **Esta dirección es relativa desde el lugar donde se encuentre el archivo de configuración xsim** (Generalmente en un proyecto de vivado: proyecto.sim/sim_1/behav/xsim).
- read_mode, el modo con el que abrir el archivo a elegir entre (read_mode, write_mode, append_mode).

```
file_open(estado, file_INPUT, "../input.txt",  read_mode);
assert estado=open_ok
```

2.2.3 Uso de assert para verificación de señales

Al abrir un archivo hemos indicado que se va a comprobar si se abrió correctamente con la señal estado. Para esto se usa la instrucción assert de la siguiente manera.

```
assert estado=open_ok
  report "No se pudo abrir el archivo con los datos"
  severity note;
```

En este caso, se ha comprobado que la variable estado sea del tipo open_ok. Esta instrucción manda el reporte indicado con un nivel de severidad específico en la consola Tcl. **Tenéis que hacer uso de esta señal para comprobar que la señal de salida esperada es correcta.** Hay cuatro valores válidos para la severidad (note, warning, error, failure). El último de estos fuerza la simulación a parar.

2.2.4 Lectura de líneas y variables

Con el archivo abierto se puede proceder a la lectura. Para eso **se debe usar un bucle while** el cual compruebe que no se ha llegado al final del fichero. Esta comprobación se realiza de forma sencilla mediante la instrucción *endfile(file_INPUT)***.**

```
while not endfile(file_INPUT) loop
    readline(file_INPUT, v_ILINE);
    read(v_ILINE, v_TIME);
```

Dentro de dicho bucle se procede a leer las líneas mediante la instrucción *readline(file_INPUT, v_ILINE)***.** Y a leer las variables de una línea mediante la instrucción *read(v_ILINE, v_TIME).* En este enlace podéis encontrar las interfaces de estos procedimientos (https://www.hdlworks.com/hdl_corner/vhdl_ref/VHDLContents/TEXTIOPackage.htm) y en este enlace encontrais una pequeña guia sobre lectura y escritura de textos (https://vhdlguide.com/2017/09/22/textio/).

2.2.5 Casting del valor leido

Como podéis ver en estos enlaces, no existe un procedimiento para la lectura de un valor std_logic. Se puede solucionar este problema leyendo el valor con una variable integer, y realizando un casting de el valor leído a un valor deseado. Específicamente, para castear de integer a std_logic se puede realizar:

```
rst_n <= to_unsigned(v_RST, 1)(0);
```

2.2.6 Escritura del report

Finalmente, con las comprobaciones realizadas, se procede a escribir el reporte. Esta vez haciendo uso de las procedures *write* y *writelines.* A la hora de escribir ciertos tipos de variable, puede que haga falta hacer uso del método ‘image. Por ejemplo, al escribir un integer es necesario o se escribiría en binario.

```
write(v_OLINE, integer'image(v_RST));
writeline(file_OUTPUT, v_OLINE);
```

Para la escritura del encabezado y del final del fichero utilizar la siguiente sentencia:

```
--Escribimos encabezado del output.txt
write(v_OLINE, string'("Simulation of top_practica1.vhd"));
```

2.1 Objetivos

Con estas herramientas a vuestra disposición el objetivo es el siguiente.

- Definir un archivo de entradas input.txt en el que se considere todos los posibles casos de cambio de estado del sistema descrito en la práctica 1.
- Definir en el testbench un proceso que lea el archivo de entradas línea a línea. Asigne los valores marcados a las entradas del sistema y compruebe si el valor esperado es correcto. (Tened en cuenta que el LED no cambia inmediatamente por dos cosas: el delay y el propio funcionamiento de debouncer).
- Tanto en caso correcto como incorrecto debe imprimir la simulación realizada en un fichero de salida output.txt.
- El sistema debe también hacer uso de instrucciones assert para verificar de forma automática el valor esperado, realizando un reporte en caso contrario.. (No usar severity failure o la simulación no continua al haber fallo).