

## Práctica 3

### Diseño Automático de Sistemas Fiabiles

José María Fernández Gómez

Álvaro Terrón

Gonzalo Vázquez



UNIVERSIDAD  
NEBRIJA

José María Fernández Gómez

Álvaro Terrón

Gonzalo Vázquez

# Objetivos De la Práctica

Se pretende trabajar la lectura y escritura de archivos de texto, en este caso como archivos de verificación y testeo automático de sistemas diseñados en FPGAS. Trabajamos con la librería textio, perteneciente al paquete std, más adelante se explicarán las apis y prototipos empleados. Por otro lado se ha trabajado con la expresión de verificación ASSERT.

“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).”

## EXPRESIÓN ASSERT

, Esta expresión se emplea para hacer la verificación de una expresión o resultado de una operación, es un condicional que reporta, normalmente se empleará en los testbench para comprobar si ciertas condiciones se cumplen en el flujo de la verificación y testeo de una unidad bajo test (UUT).

# Funcionamiento

## 1. Imports requeridos:

```
3  -----  
4  library IEEE;  
5  use IEEE.STD_LOGIC_1164.ALL;  
6  use IEEE.NUMERIC_STD.ALL;  
7  library std;  
8  use STD.TEXTIO.ALL;
```

A parte de los imports usuales, tenemos que importar la librería textio para poder leer y escribir archivos de texto.

## 2. Señales y constantes

```
constant timer_debounce : integer := 10; --ms
constant freq : integer := 100_000; --KHZ
constant clk_period : time := (1 ms/ freq);

file file_input : text; -- Manejar ficheros
file file_output : text;

-- Inputs
signal rst_n      : std_logic := '0';
signal clk        : std_logic := '0';
signal BTN        : std_logic := '0';
-- Output
signal LED        : std_logic;
--Senhal fin de simulacion
signal fin_sim : boolean := false;
```

Tenemos las señales que ya vimos en la práctica 1, pero aquí

encontramos dos nuevos prototipos de señales, siendo estas las de los manejadores de archivos de texto, necesarias para poder trabajar con dicho formato.

## 3. Variables dentro del process

```
process is
    -- Variables para manejar los ficheros
    variable v_estado_input : file_open_status;
    variable v_estado_output : file_open_status;

    variable v_iLINE : line;
    variable v_oLine : line;

    -- Variables para los valores de entrada

    variable v_time : time;
    variable v_rst : integer;
    variable v_btn : integer;

    -- Variables para los valores de salida

    variable v_xptc_led : integer;

begin
```

Aquí tenemos variables necesarias para el manejador de archivos, tanto variables de estado, como lo son v\_estado\_input como v\_estado\_output, cada una siendo correspondiente al archivo de entrada de datos como el de salida correspondientemente. Justo debajo tenemos variables de tipo línea necesarias para ir iterando y escribiendo cada línea dentro del archivo, que previamente estamos tratando con el manejador de archivo correspondiente, tanto entrada como salida tienen que tener su propia variable de línea.

Las variables del segundo grupo son las variables donde tenemos guardado los datos de lectura de los que sacaremos los datos para simular nuestro sistema.

#### 4. Verificación apertura y cierre de archivos

```
begin

file_open(v_estado_input, file_input, "../../../input.txt", read_mode);
file_open(v_estado_input, file_output, "../../../output.txt", write_mode);

-- Verificamos que se haya abierto
assert v_estado_input = open_ok
    report "Could not open file input.txt"
    severity failure;

assert v_estado_output = open_ok
    report "Could not open file output.txt"
    severity failure;
```

Como suele ser normal en cualquier tipo de programa o diseño, ya sea hardware o software, cuando se trata con recursos externos al propio flujo del programa hay que hacer comprobaciones de apertura, lectura y cierre, ya que si esto no se hace, cabe la posibilidad de que se de corrupción en los datos y fallos de tipo IO.

#### 5. Flujo normal del TestBench:

```
wait until rising_edge(clk);
wait until rising_edge(clk);
rst_n <= '1';
wait until rising_edge(clk);
rst_n <= '0';
wait until rising_edge(clk);
rst_n <= '1';

-- Fin de secuencia de reset
wait until rising_edge(clk);

) while not endfile(file_input) loop
    readline(file_input, v_iline);

    read(v_iline, v_time);
    report "The value of v_time is " & time'image(v_time);

    read(v_iline, v_rst);
    report "The value of v_rst is " & integer'image(v_rst);

    read(v_iline, v_btn);
    report "The value of v_btn is " & integer'image(v_btn);
```

En esta primera parte del código del testbench se ve como se itera a través de las líneas del archivo de texto, que previamente hemos abierto con el manejador de archivos, hasta que llegamos al final de este leyendo cada una de sus líneas y dejando un report a forma de traza de cada uno de los valores leídos por este manejador, valores que luego serán implementados en el testbench a manera de estímulos.

## 6. Contenido del IPUT.TXT

Archivo	Duración	Formato	Ver	Ayuda
10ns	1	0 0		#After reset sequence start test stimulus
13ns	1	1 1		#LED SE ENCIENDE
235ns	1	0 0		#LED NO SE ENCIENDE
166ns	1	1 1		#LED SE ENCIENDE
20ms	1	0 0		#FALSO BOTON OFF
164ns	1	1 1		#LED SE ENCIENDE
20ms	1	0 0		#BOTON OFF WITH NOISE
150ns	1	1 1		#LED ENCENDIDO
162ns	1	0 0		#FIN SIMULACION

Aquí encontramos el contenido de nuestros estímulos, marcando en la primera columna el tiempo que debe durar cada pulso del tren de datos que le estamos suministrando y la duración de cada uno de estos trenes de datos.

## 7. Guardado de datos y traza

```

--
rst_n <= to_unsigned(v_rst, 1)(0);
BTN <= to_unsigned(v_btn, 1)(0);

wait for v_time;

write(file_output, "Time value is:" &time'image(v_time));
write(file_output, "nRst value is:" &integer'image(v_rst));
write(file_output, "Led value is:" &integer'image(v_xptc_led));
writeline(file_output, v_oLINE);

end loop;

write(file_output, "End simulation");

-- Fin simulacion
fin_sim <= true;
wait;

end process;
end testBench;

```

Por último, al finalizar los tiempos de simulación establecidos, escribimos a través del manejador de archivos de escritura el resultado de cada iteración y los valores de las señales que estaban bajo supervisión, esto para automatizar el proceso de verificación y dejarlo en el mismo formato que el fichero de entrada de prueba.

## 8. ASSERT SEVERITY

```
-- Verificamos que se haya abierto
assert v_estado_input = open_ok
    report "Could not open file input.txt"
    severity failure;

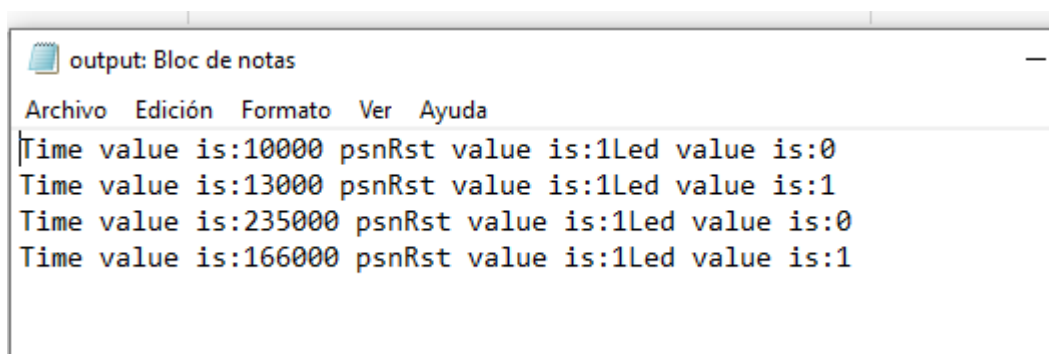
assert v_estado_output = open_ok
    report "Could not open file output.txt"
    severity failure;
```

El otro objetivo de la práctica es trabajar sobre las expresiones `assert`, y `severity`, estas son empleadas a modo de verificación y traza en un testbench. La primera es el equivalente al operador `==` en programación software, y se encarga de comprobar expresamente una condición, en este caso comprobar que el código de estado del manejador de archivos es exitoso, y en el caso contrario lanzar un `severity`, similar a un "throw new error", en el que se le puede incluir la criticidad de este error, siendo `note` el más bajo y `failure` el más alto.

Los posibles valores que puede tomar este `report` en orden de criticidad son:

- Note:** Siendo el más leve y prácticamente no se considera un error si no simplemente un mensaje informativo o de traza
- Warning:** Se da cuando el flujo del programa no ha sido perfecto y se ha encontrado un valor no esperado, pero el flujo de la simulación puede continuar
- Error:** El flujo del programa no puede continuar ya que se ha encontrado un valor no esperado crítico para la secuencia siguiente
- Failure:** El error es tan crítico que la simulación debe ser detenida en ese mismo instante

## 9. EJEMPLOS DE EJECUCIÓN Y CONCLUSIONES



```
output: Bloc de notas
Archivo Edición Formato Ver Ayuda
Time value is:10000 psnRst value is:1Led value is:0
Time value is:13000 psnRst value is:1Led value is:1
Time value is:235000 psnRst value is:1Led value is:0
Time value is:166000 psnRst value is:1Led value is:1
```

En la salida se nos crea un fichero `txt` que se llama `output` que como se puede ver en el extracto generado por nuestro testbench el sistema se comporta según lo hemos descrito en el fichero de `inputs`, por lo que el sistema funciona correctamente bajo los tests sometidos