UNIVERSIDAD NEBRIJA

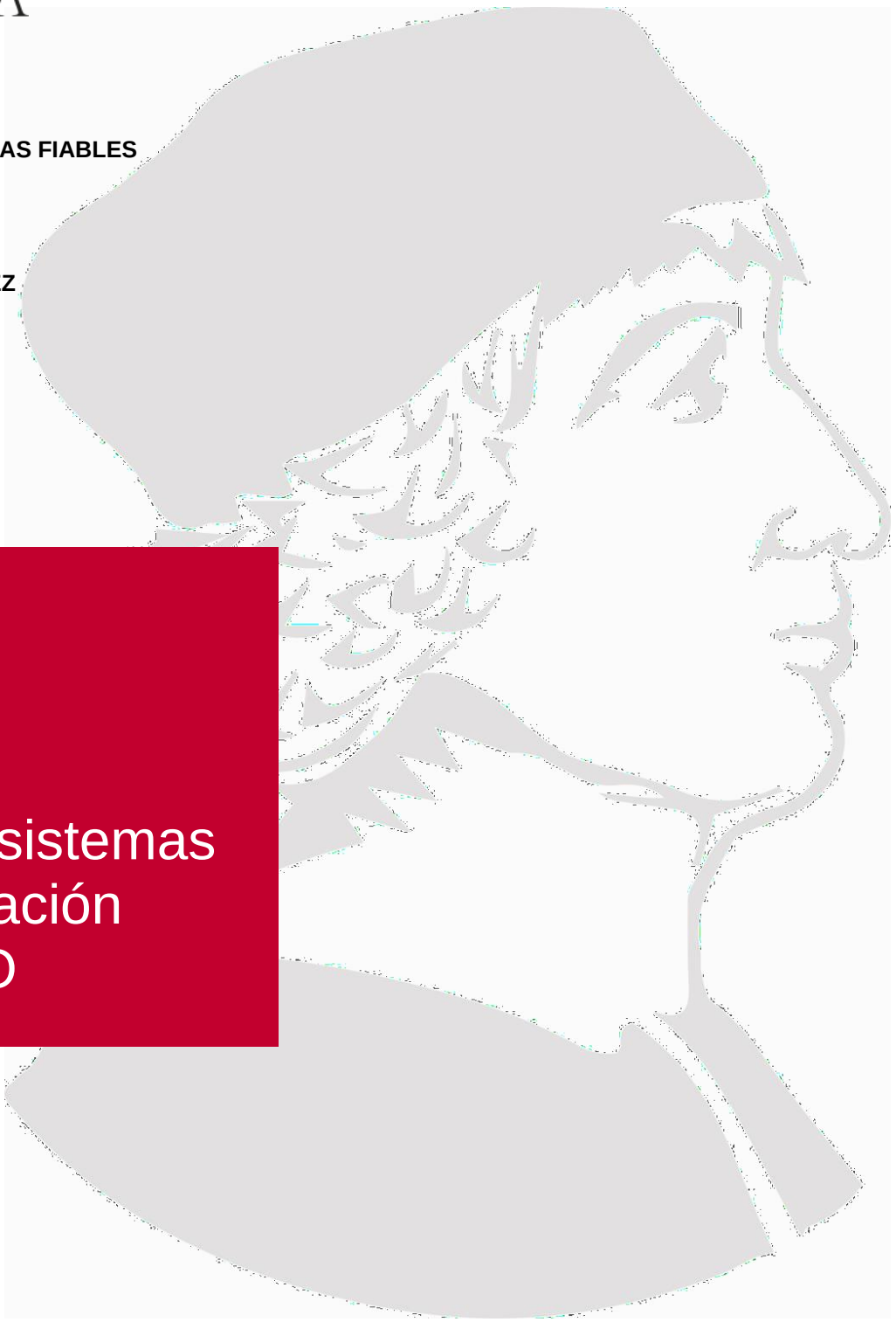
**DISEÑO AUTOMÁTICO DE SISTEMAS FIABLES**

**ALUMNOS:**

**IGNACIO MURUBE CREGO**
**JUAN MANUEL VICENTE MARTINEZ**
**JESUS NAVAS MARTIN**

**PROFESOR:**

**IGNACIO AZNÁREZ RAMOS**



# Práctica 3:
## Verificación de sistemas mediante simulación con archivos I/O

# Tabla de contenidos

# Introducción

El objetivo de esta práctica manejamos la verificación de un sistema a través la lectura y escritura de ficheros, mediante la función asserts, la cual se enfocará en comprobar los resultados.

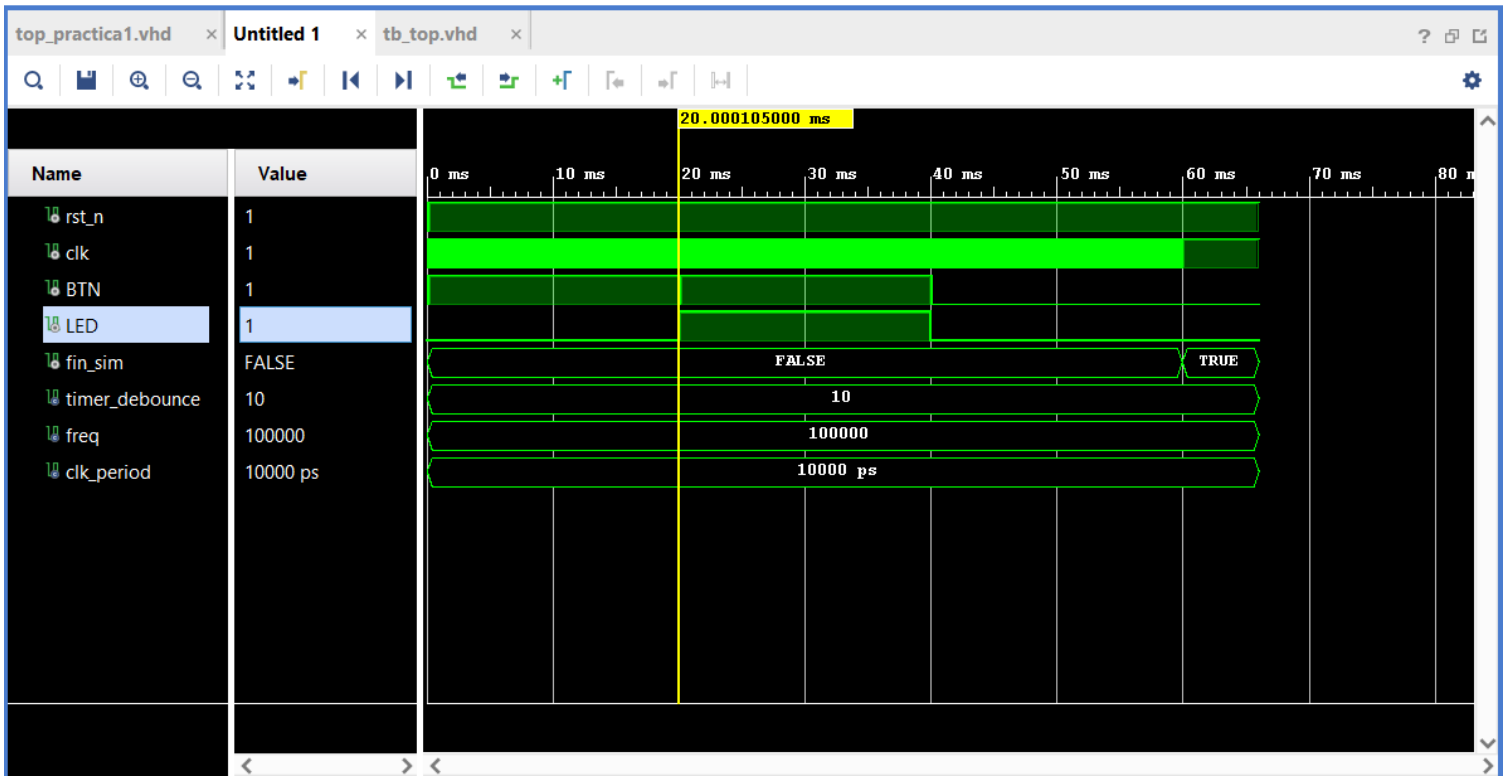
Los dos ficheros que usaremos input.txt (leerán los valores de entrada de BTNC y rst_n, así como el valor esperado de la salida LED y el tiempo a esperar hasta la próxima instrucción) y el output (en el cual se escribirán los casos probados, incluyendo el tiempo de delay, las entradas y salidas o los errores si los hubiera).

# Cambios

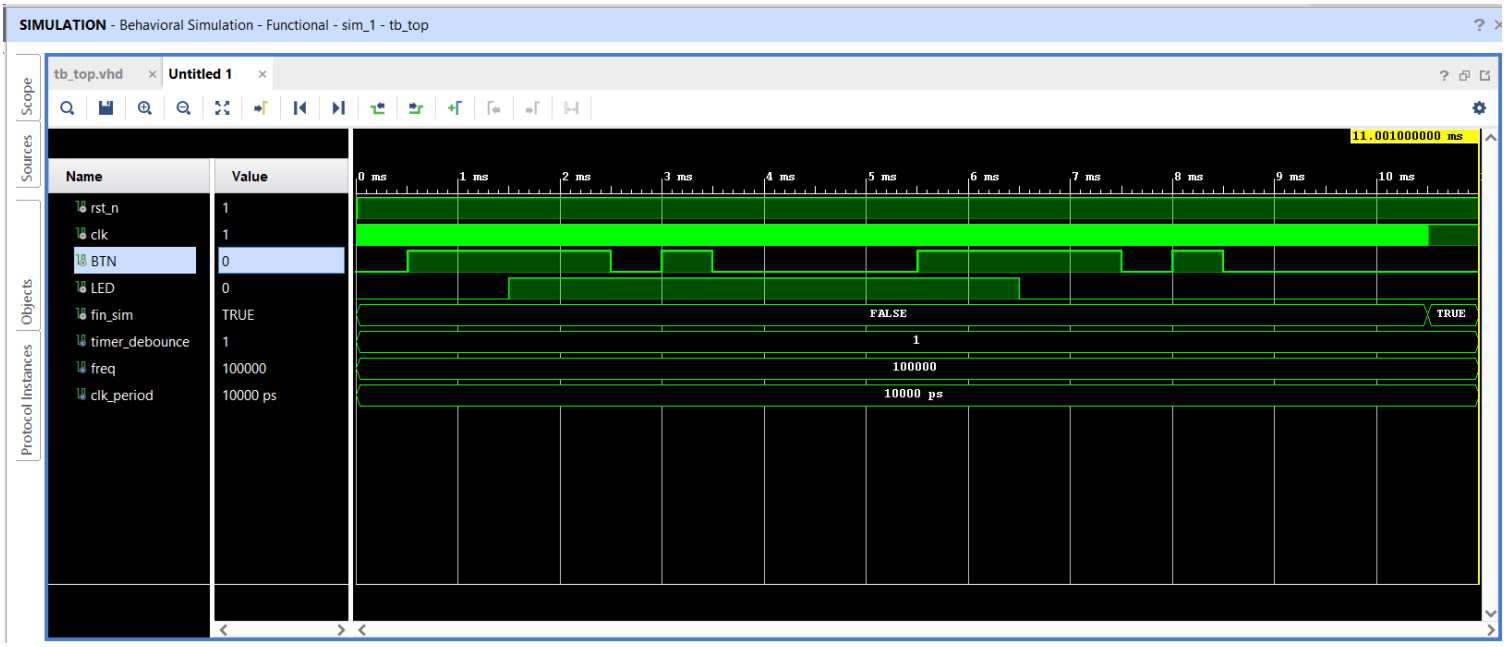
Los cambios establecidos en esta práctica:

## Corrección practica 1

Como nos fue indicado, el funcionamiento del bloque top_practica1 era incorrecto debido a que el LED de salida no estaba respondiendo correctamente con un funcionamiento 'toggle' a las pulsaciones validadas de la señal 'BTN' de entrada al sistema. Este error resultaba en que el LED no permanecía encendido cuando se dejaba de pulsar el botón, como se puede observar en la siguiente imagen:




Cómo fue indicado en los comentarios de la corrección realizada por el profesor, se presentaba un error de concepto en cuanto al funcionamiento de la FSM presente en el bloque interno 'debouncer'. Por lo tanto, tras modificar los estímulos de prueba presentes en tb_top y conectar la señal interna 'timer_debounce' al mapa de conexiones genérico del UUT se pudo solucionar satisfactoriamente el error dando como resultado el funcionamiento tipo interruptor esperado en el LED, como se observa en la siguiente imagen:




En esta simulación se ha establecido el tiempo de validación de pulsación en 1[ms], adicionalmente se han separado los casos estudiados para su fácil visualización y se ha añadido el cuarto caso de testeo de falsa pulsación de encendido al final.

Casos de prueba:

1. **“BTN on with noise”:** Se pulsa el botón y se incluye un ruido intermedio durante el transcurso del tiempo de validación de la pulsación para finalmente mantenerlo pulsado para encender el LED de salida.
2. **“False BTN off”:** Se realiza una breve pulsación del botón y se suelta antes del tiempo de validación para modelar el caso de una falsa pulsación de apagado del LED.
3. **“BTN off with noise”:** Se pulsa el botón y se incluye un ruido intermedio durante el transcurso del tiempo de validación de la pulsación para finalmente mantenerlo pulsado para apagar el LED de salida.
4. **“False BTN on”:** Se realiza una breve pulsación del botón y se suelta antes del tiempo de validación para modelar el caso de una falsa pulsación de encendido del LED.

## tb_top

Añadiremos el paquete textio de la librería std para así poder manejar los ficheros y leer y escribir en ellos.

```
1  library IEEE;
2  library std; --Libreria que usaremos en la practica 3
3  use std.textio.all;
4  use IEEE.STD_LOGIC_1164.ALL;
5  use IEEE.NUMERIC_STD.ALL;
6
```

Continuando justo como en la práctica 1, ahora declaramos la entidad, las constantes y señales necesarias, junto con el mapa del componente top_practica1 que hicimos en la práctica 1, y la creación de la señal de reloj:

```
20    constant timer_debounce : integer := 10; --ms
21    constant freq : integer := 100_000; --KHZ
22    constant clk_period : time := (1 ms/ freq);
23
24    -- Inputs
25    signal  rst_n       :   std_logic := '0';
26    signal  clk         :   std_logic := '0';
27    signal  BTN     :   std_logic := '0';
28    -- Output
29    signal  LED   :   std_logic;
30    -- Señal fin de simulacion
31    signal  fin_sim : boolean := false;
32
33  begin
34    UUT: top_practica1
35      port map (
36        rst_n     => rst_n,
37        clk100Mhz => clk,
38        BTNC       => BTN,
39        LED       => LED
40      );
41
42    --Proceso de generacion del reloj
43    clock: process
44    begin
45        clk <= '0';
46        wait for clk_period/2;
47        clk <= '1';
48        wait for clk_period/2;
49        if fin_sim = true then
50          wait;
51       end if;
52    end process;
```

Empezaremos un nuevo proceso en el que declararemos las variables necesarias para manejar los ficheros de input y output, además de las variables para leer y escribir los:

```
process is
--variables para manejar los ficheros
file file_INPUT : text;
file file_OUTPUT : text;
variable v_status_input: file_open_status;
variable v_status_output: file_open_status;

variable v_ILINE: line; --Para almacenar cada linea del fichero
variable v_OLINE: line;
--Variables para los valores de entrada
variable v_RST: integer; --Valor del boton que obtendremos del fichero
variable v_BTNC: integer;
variable v_TIME: time;

--Variables para los valores de salida
variable v_expected: std_logic;
variable v_LED: integer;

begin
```

Empezaremos a manejar la apertura y escritura de los ficheros correspondientes. Para ello, se utiliza la función file_open (para abrir los ficheros) y assert (para verificar si se abrieron correctamente con la señal de estado. Si no es así, enviara un reporte, y con failure se fuerza la simulación a parar). Tras esto, con los procesos write y writeline escribiremos en el output

```
--Abrimos los ficheros con la instruccion file_open (punto 2.2.2)
file_open(v_status_input, file_INPUT, "../input.txt", read_mode);
file_open(v_status_output, file_OUTPUT, "../output.txt", write_mode);

--2.2.3 Uso de assert para verificacion de seniales

--Comprobamos que se abren correctamente, en caso contrario paramos la simulacion
assert v_status_input = open_ok
  report "El fichero input.txt no se ha abierto correctamente"
  severity failure; --Con severity failure forzamos la simulacion a parar

assert v_status_output = open_ok
  report "El fichero output.txt no se ha abierto correctamente"
  severity failure; --Con severity failure forzamos la simulacion a parar
```

Dentro de un bucle while mientras no se llegue al final del fichero, se vayan leyendo todas las líneas del input gracias a la función readline, para luego transformar de los valores leídos a unos valores deseados.

```
--2.2.6
write(v_oline, string'("Simulation of top_practica1.vhd"));
writeline(file_OUTPUT, v_OLINE);

 while (not endfile(file_INPUT)) loop
     readline(file_INPUT, v_ILINE); --lee toda la linea
     read(v_ILINE, v_TIME);         --lee hasta un espacio en blanco
     read(v_ILINE, v_RST);
     read(v_ILINE, v_BTNC);
     read(v_ILINE, v_LED);

     --2.2.5
     BTN <= to_unsigned(v_BTNC,1)(0);
     rst_n <= to_unsigned(v_RST,1)(0);
     v_expected:= to_unsigned(v_LED,1)(0);

     wait for v_TIME;

     --2.2.6 Escribimos el reporte
     write(v_OLINE, "Time: " & time'image(v_TIME) & "  rst_n: " & integer'image(v_RST) & "  BTNC: " & integer'image(v_BTNC));
     writeline(file_OUTPUT, v_OLINE);
     assert v_expected /= LED
       report "ERROR"
       severity note;

       if(v_expected = LED) then
           write(v_OLINE, "LED: " & integer'image(v_LED));
           writeline(file_OUTPUT, v_OLINE);
       else
           write(v_OLINE, "ERROR: " & "  Expected LED to be: " & integer'image(v_LED) &
           "  actual value: " & std_logic'image(LED));
           writeline(file_OUTPUT, v_OLINE);
         end if;
 end loop;
```

Al concluir el proceso, generamos un informe en el resultado con los datos recolectados. Dentro de este mismo proceso, realizamos una comprobación con un "assert" para determinar si el valor esperado del LED es diferente del valor actual. Si coinciden, registramos el valor del LED, si no es así, emitimos un mensaje de error.
Para terminar, escribimos un mensaje para indicar la finalización de la simulación.

```
write(v_oline, string'("END SIMULATION"));
writeline(file_OUTPUT, v_OLINE);
 fin_sim <= true;
wait;
```

# Solución

Tras ejecutar el proyecto con los cambios establecidos mostraremos los resultados obtenidos desde el fichero input.txt al output.txt

En el input:

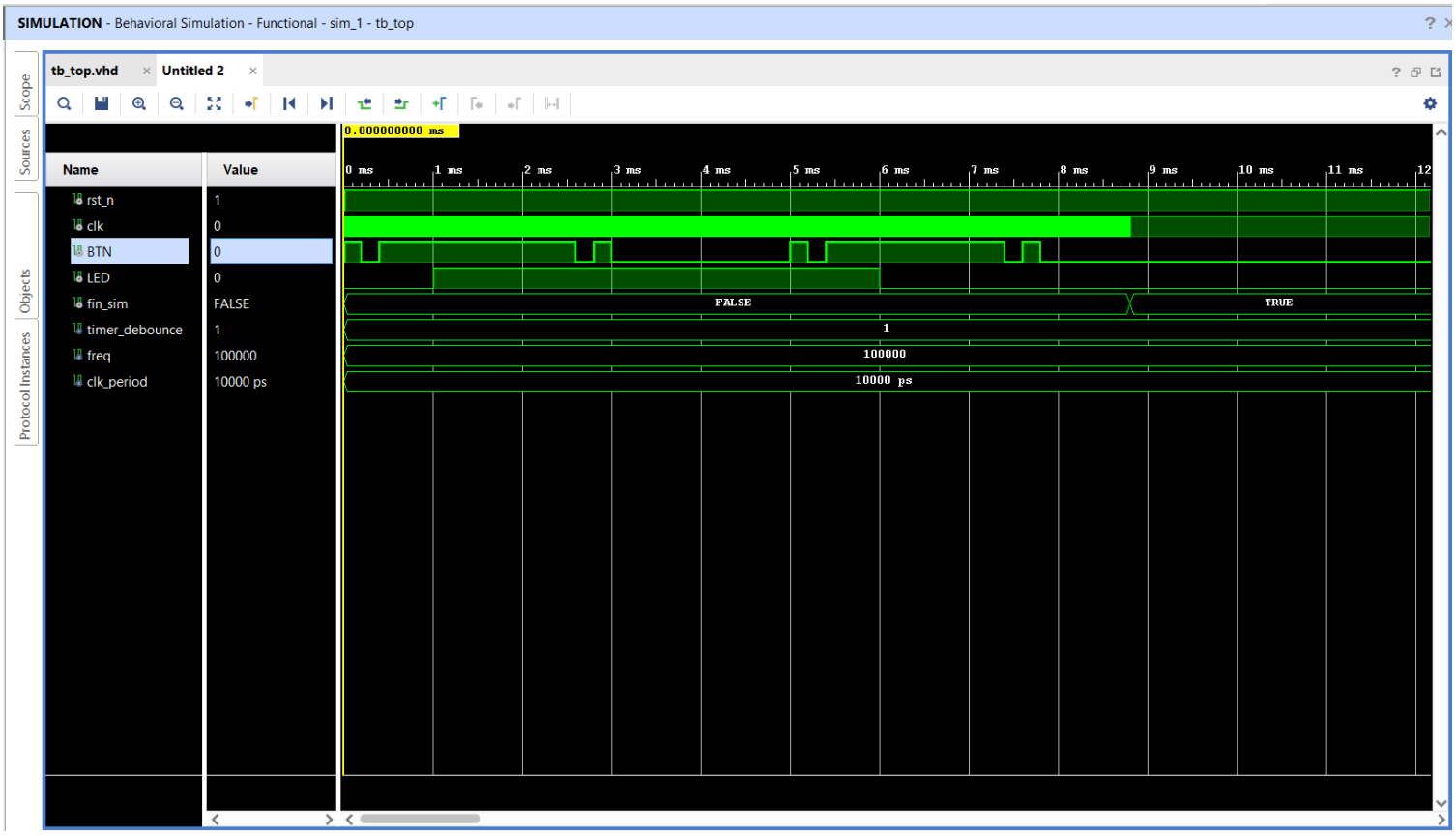
```
200ns           1 0 0   # Reset inactive
200ns           0 0 0   # Reset active
200ns           1 0 0   # Reset inactive
200000ns        1 1 0   # Btn on with noise
200000ns        1 0 0   # Btn on with noise
200000ns        1 1 0   # Btn on with noise
1000000ns       1 1 1   # Btn on with noise
200000ns        1 0 1   # False boton off
200000ns        1 1 1   # False boton off
1000000ns       1 0 1   # False boton off
200000ns        1 1 1   # Boton off with noise
200000ns        1 0 1   # Boton off with noise
1000000ns       1 1 0   # Boton off with noise
200000ns        1 0 0   # False boton on
200000ns        1 1 0   # False boton on
1000000ns       1 0 0   # False boton on
```

En el output:

```
Simulation of top_practica1.vhd
Time: 200000 ps  rst_n: 1  BTNC: 0
LED: 0
Time: 200000 ps  rst_n: 0  BTNC: 0
LED: 0
Time: 200000 ps  rst_n: 1  BTNC: 0
LED: 0
Time: 200000000 ps  rst_n: 1  BTNC: 1
LED: 0
Time: 200000000 ps  rst_n: 1  BTNC: 0
LED: 0
Time: 200000000 ps  rst_n: 1  BTNC: 1
LED: 0
Time: 1000000000 ps  rst_n: 1  BTNC: 1
LED: 1
Time: 200000000 ps  rst_n: 1  BTNC: 0
LED: 1
Time: 200000000 ps  rst_n: 1  BTNC: 1
LED: 1
Time: 1000000000 ps  rst_n: 1  BTNC: 0
LED: 1
Time: 200000000 ps  rst_n: 1  BTNC: 1
LED: 1
Time: 200000000 ps  rst_n: 1  BTNC: 0
LED: 1
Time: 1000000000 ps  rst_n: 1  BTNC: 1
LED: 0
Time: 200000000 ps  rst_n: 1  BTNC: 0
LED: 0
Time: 200000000 ps  rst_n: 1  BTNC: 1
LED: 0
Time: 1000000000 ps  rst_n: 1  BTNC: 0
LED: 0
END SIMULATION
```

Podemos observar que en el output no hay ningún error por lo que hemos obtenido lo que se esperaba obtener.

Resultado de la simulación:




Resultados con errores:

A continuación, haremos que nos den errores (solo para ver si funciona lo de escribir los errores), para eso haremos que creyésemos que el led no se enciende nunca.

Input.txt:

```
200ns           1 0 0   # Reset inactive
200ns           0 0 0   # Reset active
200ns           1 0 0   # Reset inactive
200000ns        1 1 0   # Btn on with noise
200000ns        1 0 0   # Btn on with noise
200000ns        1 1 0   # Btn on with noise
1000000ns       1 1 0   # Btn on with noise
200000ns        1 0 0   # False boton off
200000ns        1 1 0   # False boton off
1000000ns       1 0 0   # False boton off
200000ns        1 1 0   # Boton off with noise
200000ns        1 0 0   # Boton off with noise
1000000ns       1 1 0   # Boton off with noise
200000ns        1 0 0   # False boton on
200000ns        1 1 0   # False boton on
1000000ns       1 0 0   # False boton on
```

Output.txt:

```
Simulation of top_practica1.vhd
Time: 200000 ps  rst_n: 1  BTNC: 0
LED: 0
Time: 200000 ps  rst_n: 0  BTNC: 0
LED: 0
Time: 200000 ps  rst_n: 1  BTNC: 0
LED: 0
Time: 200000000 ps  rst_n: 1  BTNC: 1
LED: 0
Time: 200000000 ps  rst_n: 1  BTNC: 0
LED: 0
Time: 200000000 ps  rst_n: 1  BTNC: 1
LED: 0
Time: 1000000000 ps  rst_n: 1  BTNC: 1
ERROR:   Expected LED to be: 0  actual value: '1'
Time: 200000000 ps  rst_n: 1  BTNC: 0
ERROR:   Expected LED to be: 0  actual value: '1'
Time: 200000000 ps  rst_n: 1  BTNC: 1
ERROR:   Expected LED to be: 0  actual value: '1'
Time: 1000000000 ps  rst_n: 1  BTNC: 0
ERROR:   Expected LED to be: 0  actual value: '1'
Time: 200000000 ps  rst_n: 1  BTNC: 1
ERROR:   Expected LED to be: 0  actual value: '1'
Time: 200000000 ps  rst_n: 1  BTNC: 0
ERROR:   Expected LED to be: 0  actual value: '1'
Time: 1000000000 ps  rst_n: 1  BTNC: 1
LED: 0
Time: 200000000 ps  rst_n: 1  BTNC: 0
LED: 0
Time: 200000000 ps  rst_n: 1  BTNC: 1
LED: 0
Time: 1000000000 ps  rst_n: 1  BTNC: 0
LED: 0
END SIMULATION
```

Como podemos observar como hemos hecho que el valor esperado siempre sea 0 cuando el LED se enciende da un error porque no es el valor que esperábamos.

# Conclusión

En conclusión, hemos aprendido a manejar ficheros de texto para el testeo de sistemas introduciendo estímulos y generando informes en estos a partir de los resultados una simulación.