Tecnología de Computadores

Práctica 3

UNIVERSIDAD NEBRIJA

1. INTRODUCCIÓN

En la tercera sesión de laboratorio se aprenderá a crear testbenches en Vivado para simular circuitos y se implementarán sistemas complejos utilizando diseño modular.

Una vez terminada la práctica el alumno será capaz de:

- Simular códigos VHDL en Vivado para verificar su funcionamiento.
- Crear diseños modulares reutilizando componentes ya hechos.

2. SIMULACIÓN DE CÓDIGOS VHDL. TESTBENCHES

En esta primera parte de la práctica se explicará cómo simular códigos VHDL en Vivado para verificar el correcto funcionamiento de un circuito. Para ello, primero vamos a crear un proyecto y a añadir el código VHDL "*simpleFunction.vhd*" proporcionado por el profesor. Como se puede observar, este código implementa una función lógica sencilla que tiene 3 entradas (A, B y C) y una salida (Y).

El primer objetivo propuesto para el alumno consiste en crear un *testbench* para simular el circuito dado con todos los posibles valores de entrada. Para ello, pulsamos en *Add Sources > Add or create simulation sources* y creamos un fichero con el nombre que queramos (normalmente se suele elegir "*tb_nombre_del_código_a_simular*" como nomenclatura). El fichero se encontrará dentro de la carpeta *Simulation Sources > sim_1*.

Una vez creado el archivo de simulación, lo abrimos para definir nuestro *testbench* como una entidad vacía que englobará al componente o componentes a simular, para ello escribimos lo siguiente al principio del fichero:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Empty Testbench Entity
entity tb_simpleFunction is
end tb_simpleFunction;
```

Una vez hecho esto, debemos definir e instanciar nuestro circuito a simular (*simpleFunction*) dentro de la arquitectura del *testbench*. Para ello escribimos las siguientes líneas:

```
architecture Behavioral of tb_simpleFunction is
   -- Define Component
   component simpleFunction
      port( A, B, C : in  std_logic;
            Y       : out std_logic
      );
   end component;

   -- Declare Signals
   signal A, B, C, Y : std_logic;
begin
```

```
  -- Instantiate Device Under Test (DUT)
  DUT : simpleFunction port map(
    A => A,
    B => B,
    C => C,
    Y => Y
  );

end Behavioral;
```

Por último, añadimos un process justo después de la instancia del DUT que se encargará de introducir los estímulos en nuestro circuito a simular:

```
-- Stimuli Process
  process begin
    A <= '0'; B <= '0'; C <= '0'; wait for 10 ns;
    A <= '0'; B <= '0'; C <= '1'; wait for 10 ns;
    A <= '0'; B <= '1'; C <= '0'; wait for 10 ns;
    A <= '0'; B <= '1'; C <= '1'; wait for 10 ns;
    A <= '1'; B <= '0'; C <= '0'; wait for 10 ns;
    A <= '1'; B <= '0'; C <= '1'; wait for 10 ns;
    A <= '1'; B <= '1'; C <= '0'; wait for 10 ns;
    A <= '1'; B <= '1'; C <= '1'; wait for 10 ns;
    wait; -- wait forever
  end process;
```

Una vez completado el código del testbench, lo siguiente que debemos hacer es lanzar la simulación. Deberemos pulsar en *Run Simulation > Run Behavioral Simulation* y, si todo ha ido bien, se nos abrirá una ventana negra con la simulación del circuito.

Compruebe que la simulación es correcta y aprenda el funcionamiento de los distintos botones de la ventana de simulación antes de pasar a la siguiente parte de la práctica y pregunte al profesor si tiene alguna duda al respecto.

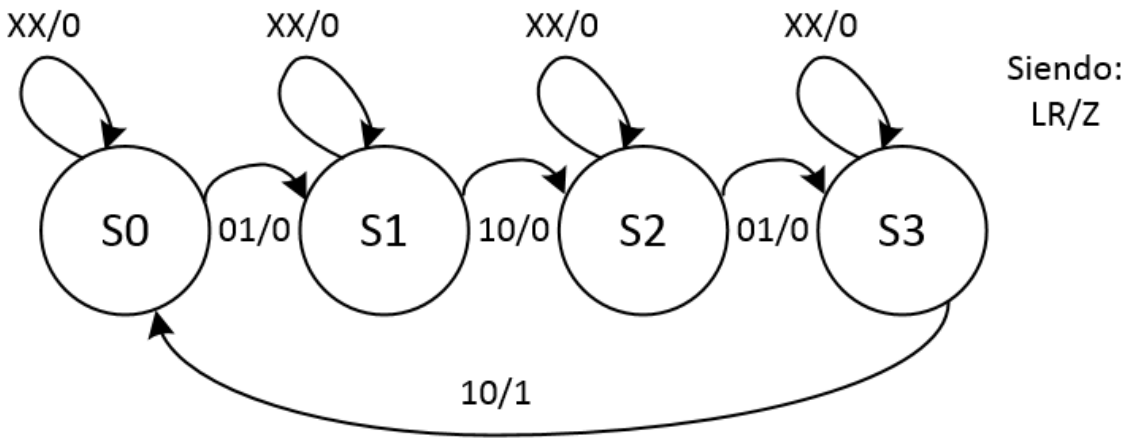
Tarea para el alumno:

1) Cree un proyecto nuevo en el que simulará el código "*flipflopD.vhd*". Para ello cree un *testbench* como hizo para la función lógica anterior pero teniendo en cuenta las entradas de su nuevo circuito a la hora de instanciarlo.
2) Modifique el *process* anterior de modo que se cambie únicamente la señal de entrada D del flip-flop y tenga el siguiente comportamiento:
    a. Comienza valiendo '0' durante 10ns.
    b. Pasa a valer '1' y permanece así durante 40ns.
    c. Vuelve a valer '0' durante 30ns.
    d. Pasa a valer '1' y permanece así indefinidamente.
3) Añada otro process para la señal de reset de modo que valga '0' inicialmente y que, tras 5ns, pase a valer '1' durante 10ns.
4) Por último, cree un tercer *process* que simule el funcionamiento de una señal de reloj con 10ns de periodo.
5) Lance la simulación y enseñe el resultado al profesor.

3. DISEÑO MODULAR

En esta segunda parte de la práctica se pretende que el alumno sea capaz de crear diseños complejos a partir de códigos más sencillos para aprender el concepto de diseño modular o multimódulo. Para ello, se va a realizar el siguiente ejercicio.

Dado el siguiente diagrama de estados de una máquina que detecta la secuencia de entrada "R-L-R-L" (right-left-right-left) siendo R el botón BTNR (pin M17) y L el botón BTNL (pin P17):



- Obtenga las ecuaciones de excitación y de salida suponiendo que se utilizan biestables D.
- Con las anteriores ecuaciones, y reutilizando el fichero "*flipflopD.vhd*" de la parte anterior de la práctica, cree un diseño modular de la máquina de estados.
- Implemente la máquina de estados en la FPGA y compruebe su funcionamiento. ¿Por qué no funciona?
- Cree un testbench y simule el comportamiento de la máquina de estados. ¿Sabe ahora por qué no funciona?
- Cree los códigos VHDL necesarios para que el sistema funcione.
- Implemente nuevamente en la FPGA y verifique su correcto funcionamiento.