INGENIERÍA DE COMPUTADORES III

Solución al Ejercicio de Autocomprobación 5

PREGUNTA 1 (2 puntos)

Tomando como base el siguiente código VHDL, dibuje el cronograma de evolución de las señales `x`, `z1` y `z2` entre los instantes 0 y 60 ns.

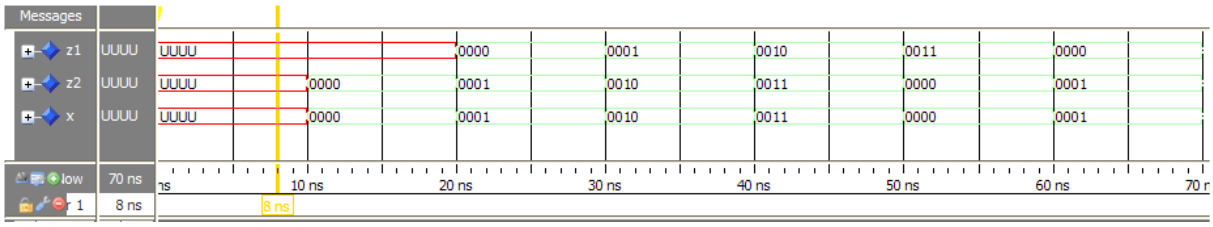
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity crono1 is
end entity crono1;

architecture crono1 of crono1 is
   signal z1: std_logic_vector (3 downto 0);
   signal z2: std_logic_vector (3 downto 0);
   signal x:  std_logic_vector (3 downto 0);
begin
process is
   begin
   for i in 0 to 3 loop
      wait for 10 ns;
      x<=std_logic_vector(TO_UNSIGNED(i,4));
      z1<=x;
   end loop;
end process;
z2<=x;
end architecture crono1;
```

Solución a la Pregunta 1

A continuación se muestra el cronograma de evolución de las señales `x`, `z1` y `z2` entre los instantes 0 y 60 ns.



El bloque **process** se ejecuta indefinidamente, ya que no existe ninguna sentencia **wait** que pare su ejecución. Por tanto, el bucle que existe en el interior de este bloque se ejecuta continuamente tomando *i* los valores 0, 1, 2 o 3.

Al inicio de cada iteración del bucle se suspende 10 ns el bloque **process** debido a la sentencia **wait for** 10 ns. Después, se ejecutan las sentencias de asignación a las señales `x` y `z1`. Estas sentencias de asignación se evalúan en los instantes 10 ns, 20 ns, 30 ns, 40 ns... pero la asignación a la señal tiene lugar en los instantes 10 ns+δ, 20 ns+δ, 30 ns+δ, 40 ns+δ, ...

Por tanto, la señal `x` cambia de valor en los instantes 10 ns+δ, 20 ns+δ, 30 ns+δ, 40 ns+δ, ... La señal `z1` cambia también de valor en los instantes 10 ns+δ, 20 ns+δ, 30 ns+δ, 40 ns+δ, ... Y el valor que se le asigna es el resultado de evaluar la expresión a la derecha de dicha asignación en los instantes 10 ns, 20 ns, 30 ns, 40 ns, ... Por ello la señal `z1` toma en el instante 10 ns+δ el valor que tenía la señal `x` en el instante 10 ns, que coincide con el valor que tenía la señal `x` en el instante 0 ns. En el instante 20 ns+δ toma el valor que tenía la señal `x` en el instante 20 ns, que coincide con el valor de la señal `x` en el instante 10 ns +δ. En consecuencia, la señal `z1` es igual a la señal `x` pero retardada 10 ns.

La asignación de valor a la señal `z2` se produce concurrentemente con la ejecución del bloque **process**. Se evalúa la expresión de asignación a la señal `z2` cada vez que una de las señales que existe en dicha expresión cambia de valor. En este caso, cada vez que la señal `x` cambia de valor se evalúa la expresión que existe a la derecha de la asignación a la señal `z2` y se asigna el valor evaluado trancurrido un tiempo δ. En consecuencia, las señales `z2` y `x` toman los mismos valores pero con un desfase de tiempo δ.

PREGUNTA 2 (3 puntos)

Escriba en VHDL, de las cuatro formas que se detallan a continuación, la **architecture** que describe el comportamiento de un circuito combinacional decodificador 3 a 8 con entrada enable. La **entity** del circuito es:

```
entity decodificador8 is
   port (  x          : out std_logic_vector(7 downto 0);
           s          : in  std_logic_vector(2 downto 0);
           enable     : in std_logic );
end entity decodificador8;
```

2.a) (0.75 puntos) Empleando una sentencia concurrente condicional (**when - else**).

2.b) (0.75 puntos) Empleando una asignación concurrente de selección (**with - select**).

2.c) (0.75 puntos) Empleando un bloque **process** con una sentencia **if**.

2.d) (0.75 puntos) Empleando un bloque **process** con una sentencia **case**.

Solución a la Pregunta 2

El diseño del decodificador 3 a 8 con entrada enable realizado empleando una sentencia concurrente condicional, concurrente de selección, un bloque **process** con una sentencia **if** y un bloque **process** con una sentencia **case** se muestra respectivamente en Código VHDL 1.1– 1.4.

```
-------------------------------------------------
-- Decodificador binario 3 a 8
-- diseñado empleando una sentencia de
-- asignacion concurrente condicional
library IEEE;
use IEEE.std_logic_1164.all;


architecture arch_decodificador8cond of decodificador8 is
begin
    x <= "00000001" when (s="000" and enable = '1') else
         "00000010" when (s="001" and enable = '1') else
         "00000100" when (s="010" and enable = '1') else
         "00001000" when (s="011" and enable = '1') else
         "00010000" when (s="100" and enable = '1') else
         "00100000" when (s="101" and enable = '1') else
         "01000000" when (s="110" and enable = '1') else
         "10000000" when (s="111" and enable = '1') else
         "00000000";
end architecture arch_decodificador8cond;
-------------------------------------------------
```

Código VHDL 1.1: Diseño del decodificador 3 a 8 empleando una sentencia de asignación concurrente condicional.

```
-------------------------------------------------
-- Decodificador binario 3 a 8
-- empleando una sentencia de
-- asignacion concurrente de seleccion
library IEEE;
use IEEE.std_logic_1164.all;


architecture arch_decodificador8sel of decodificador8 is
   signal temp : std_logic_vector(3 downto 0);
begin
    temp <= enable&s;
    with temp select
        x <= "00000001" when ("1000"),
             "00000010" when ("1001"),
             "00000100" when ("1010"),
             "00001000" when ("1011"),
             "00010000" when ("1100"),
             "00100000" when ("1101"),
             "01000000" when ("1110"),
             "10000000" when ("1111"),
             "00000000" when others;
end architecture arch_decodificador8sel;
-------------------------------------------------
```

Código VHDL 1.2: Diseño del decodificador 3 a 8 empleando una sentencia de asignación concurrente de selección.

```
-----------------------------------------------------
-- Decodificador binario 3 a 8
-- empleando un bloque process
-- con una sentencia if
library IEEE;
use IEEE.std_logic_1164.all;


architecture arch_decodificador8if of decodificador8 is
begin
   process(enable, s) is
      begin
         if (enable = '0') then
             x <= "00000000";
         elsif (s= "000") then
             x <= "00000001";
         elsif (s  = "001") then
             x <= "00000010";
         elsif (s  = "010") then
             x <= "00000100";
         elsif (s="011") then
             x <= "00001000";
         elsif (s="100") then
             x <= "00010000";
         elsif (s="101") then
            x<="00100000";
         elsif (s="110") then
            x<="01000000";
         elsif (s="111") then
            x<="10000000";
         end if;
   end process;
end architecture arch_decodificador8if;
-----------------------------------------------------
```

Código VHDL 1.3: Diseño del decodificador 3 a 8 empleando un bloque process y una sentencia if.

```
-------------------------------------------------
-- Decodificador binario 3 a 8
-- empleando un bloque process
-- con una sentencia if
library IEEE;
use IEEE.std_logic_1164.all;

architecture arch_decodificador8if of decodificador8 is
begin
   process(enable, s) is
      variable temp: std_logic_vector(3 downto 0);
      begin
      temp := enable&s;
      case temp is
           when ("1000") =>
              x <= "00000001";
           when ("1001") =>
              x<="00000010";
           when ("1010") =>
              x<="00000100";
           when ("1011") =>
              x<="00001000";
           when ("1100") =>
              x<="00010000";
           when ("1101") =>
              x<="00100000";
           when ("1110") =>
              x<="01000000";
           when ("1111") =>
              x<="10000000";
           when others =>
              x<="00000000";
      end case;
   end process;
end architecture arch_decodificador8if;
-------------------------------------------------
```

Código VHDL 1.4: Diseño del decodificador 3 a 8 empleando un bloque process y una sentencia case.

PREGUNTA 3 (3 puntos)

Realice el diseño usando VHDL de un contador síncrono BCD. Es decir, la salida del circuito toma cíclicamente los valores “0000”, “0001”, “0010”, “0011”, “0100”, “0101”, “0110”, “0111”, “1000” y “1001”. Se obtiene un nuevo valor de la salida en cada flanco de subida de la señal de reloj. El contador debe tener una entrada `reset` asíncrona activa a nivel alto, que pone la cuenta a cero. Describa el comportamiento del circuito en términos de una máquina de Moore.

La **entity** del circuito es:

```
entity contador is port(
   count      :   out std_logic_vector(3 downto 0);
   clk, reset :   in  std_logic);
end entity contador;
```

Solución a la Pregunta 3

El circuito diseñado tiene 10 estados, cada uno de ellos se corresponde con un valor de la señal de salida `count` distinto.

El código VHDL que describe el comportamiento del circuito en términos de una máquina de Moore se muestra en Código VHDL 1.5.

```
-----------------------------------------------------
---Contador BCD implementado como máquina de Moore
-----------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
   entity contador is port(
       count      :    out std_logic_vector(3 downto 0);
       clk, reset :    in   std_logic);
   end entity contador;

architecture contador of contador is
       signal internal_state: std_logic_vector(3 downto 0);
begin
 count <= internal_state;

--Cálculo del próximo estado
 proximo_estado: process (clk, reset)
 begin
      if (reset = '1') then
           internal_state <= "0000";
      elsif (rising_edge(clk)) then
           case internal_state is
                when "0000" =>
                   internal_state <= "0001";
                when "0001" =>
                   internal_state <= "0010";
                when "0010" =>
                   internal_state <= "0011";
                when "0011" =>
                   internal_state <= "0100";
                when "0100" =>
                   internal_state <= "0101";
                when "0101" =>
                   internal_state <= "0110";
                when "0110" =>
                   internal_state <= "0111";
                when "0111" =>
                   internal_state <= "1000";
                when "1000" =>
                    internal_state <= "1001";
                when "1001"=>
                    internal_state <= "0000";
               when others =>
                    internal_state <= "0000";
               end case;
      end if;
end process proximo_estado;

end architecture contador;
-----------------------------------------------
```

Código VHDL 1.5: Diseño del contador BCD.

PREGUNTA 4 (2 puntos)

Programe en VHDL el banco de pruebas del circuito secuencial que ha diseñado al contestar a la Pregunta 3. Explique detalladamente cómo el programa de test comprueba de manera sistemática el funcionamiento del circuito. El banco de pruebas debe comprobar que los valores obtenidos de la UUT coinciden con los esperados, mostrando el correspondiente mensaje en caso de que no coincidan. Al final del test, debe mostrarse un mensaje indicando el número total de errores.

Solución a la Pregunta 4

El banco de pruebas comprueba que cuando la señal `reset` se activa el circuito está en el estado inicial, que se corresponde con un valor de la señal de salida `count` "0000". También comprueba que cada vez que se produce un flanco de subida de la señal de reloj la señal de salida `count` cambia tomando consecutivamente los valores "0000", "0001", "0010", "0011","0100", "0101", "0110", "0111", "1000", "1001" y "0000".

El banco de pruebas del contador BCD diseñado en la Pregunta 3 se muestra en Código VHDL 1.6 y en Código VHDL 1.7.

```
-------------------------------------
-- Banco de pruebas del contador
--BCD
library IEEE;
use IEEE.std_logic_1164.all;

entity bp_contador is
end entity bp_contador;

architecture bp_contador of bp_contador is
   constant PERIODO    : time         := 100 ns; -- Reloj
   signal    q          : std_logic_vector(3 downto 0);-- Salidas UUT
   signal    clk        : std_logic   := '0';      -- Entradas UUT
   signal    reset   : std_logic;

   component contador is
       port ( count              : out std_logic_vector(3 downto 0);
                clk, reset    : in std_logic );
   end component contador;

    -- Procedimiento para comprobar las salidas
    procedure comprueba_salidas
        ( esperado_q             :          std_logic_vector(3 downto 0);
          actual_q                :          std_logic_vector(3 downto 0);
          error_count             : inout integer) is
    begin
          -- Comprueba q
          if ( esperado_q /= actual_q ) then
              report "ERROR: Estado esperado ("      &
                       std_logic'image(esperado_q(3))    &
                       std_logic'image(esperado_q(2))    &
                       std_logic'image(esperado_q(1))    &
                       std_logic'image(esperado_q(0))    &
                       "), estado actual ("             &
                       std_logic'image(actual_q(3))      &
                       std_logic'image(actual_q(2))      &
                       std_logic'image(actual_q(1))      &
                       std_logic'image(actual_q(0))      &
                       "), instante: "                   &
                       time'image(now);
              error_count := error_count + 1;
          end if;
     end procedure comprueba_salidas;
```

Código VHDL 1.6: Diseño del banco de pruebas del contador BCD.

```
begin
   -- Instanciar y conectar UUT
   uut : component contador port map
                      (q,  clk, reset);

   reset <=    '0','1' after (PERIODO/4),
               '0' after (PERIODO+PERIODO/4);
   clk <= not clk after (PERIODO/2);
   gen_vec_test : process is
      variable error_count : integer := 0; -- Núm. errores
   begin
      report "Comienza la simulación";
       -- Vectores de test y comprobación del resultado
         wait for PERIODO;     -- 1
         comprueba_salidas("0000",q,error_count);
         wait for PERIODO;     -- 2
         comprueba_salidas("0001",q,error_count);
         wait for PERIODO;     -- 3
         comprueba_salidas("0010",q,error_count);
         wait for PERIODO;     -- 4
         comprueba_salidas("0011",q,error_count);
         wait for PERIODO;     -- 5
         comprueba_salidas("0100" ,q,error_count);
         wait for PERIODO;     -- 6
         comprueba_salidas("0101",q,error_count);
         wait for PERIODO;     -- 7
         comprueba_salidas("0110",q,error_count);
         wait for PERIODO;     -- 8
         comprueba_salidas("0111",q,error_count);
         wait for PERIODO;     -- 9
         comprueba_salidas("1000",q,error_count);
         wait for PERIODO;     -- 10
         comprueba_salidas("1001",q,error_count);
         wait for PERIODO;     -- 11
         comprueba_salidas("0000",q,error_count);
      -- Informe final
      if (error_count = 0) then
           report "Simulación finalizada sin errores";
      else
           report   "ERROR: Hay "                &
                     integer'image(error_count ) &
                     " errores.";
      end if;
      wait;        -- Final del bloque process
   end process gen_vec_test;
end architecture bp_contador;
-------------------------------------
```

Código VHDL 1.7: Continuación del diseño del banco de pruebas del contador BCD.