INGENIERÍA DE COMPUTADORES III

Solución al Ejercicio de Autocomprobación 6

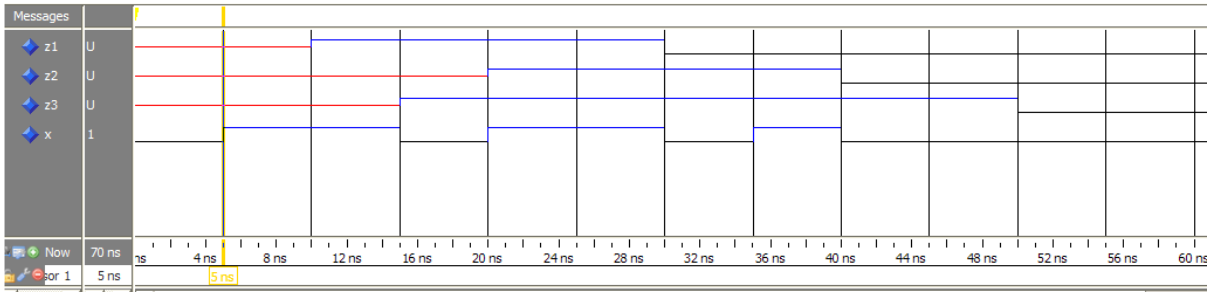
PREGUNTA 1 (2 puntos)

Tomando como base el siguiente código VHDL, dibuje el cronograma de evolución de las señales `x`, `z1`,`z2` y `z3` entre los instantes 0 y 60 ns.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity crono2 is
end entity crono2;
architecture crono2 of crono2 is
  signal z1: std_logic;
  signal z2: std_logic;
  signal z3: std_logic;
  signal x:  std_logic;
begin
  process is
    variable var1: std_logic;
  begin
    for i in 1 to 4 loop
      var1 := x;
      z1 <= var1;
      z2 <= z1;
      wait for 10 ns;
    end loop;
  end process;
  z3 <=  x after 10 ns;
  x  <= '0',             '1' after  5 ns,
        '0' after 15 ns, '1' after 20 ns,
        '0' after 30 ns, '1' after 35 ns,  '0' after 40 ns;
end architecture crono2;
```

Solución a la Pregunta 1

A continuación se muestra el cronograma de evolución de las señales `x`, `z1`, `z2` y `z3` 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.

La sentencia **wait for** 10 ns provoca que al final de cada iteración del bucle el bloque **process** se suspenda 10 ns.

Las asignaciones a variables se producen instantáneamente, sin ningún retardo. Por tanto, las dos asignaciones `var1:=x` y `z1<=var1` son equivalentes a `z1<=x`. La señal `z1` toma así en los instantes 0 ns+δ, 10 ns+δ, 20 ns+δ, 30 ns+δ, 40 ns+δ y 50 ns+δ el mismo valor que tiene la señal `x` en los instantes 0 ns, 10 ns, 20 ns, 30 ns, 40 ns y 50 ns respectivamente.

La señal `z2` cambia de valor en los instantes 0 ns+δ, 10 ns+δ, 20 ns+δ, 30 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 0 ns, 10 ns, 20 ns, 30 ns, ... Por ello, la señal `z2` toma en el instante 10 ns+δ el valor que tenía la señal `z1` en el instante 10 ns, que coincide con el valor que tenía la señal `z1` en el instante 0 ns. En el instante 20 ns+δ toma el valor que tenía la señal `z1` en el instante 20 ns, que coincide con el valor de la señal `z1` en el instante 10 ns +δ. En consecuencia, la señal `z2` es igual a la señal `z1` pero retardada 10 ns.

La señal `z3` tiene un retardo inercial de 10 ns respecto la señal `x`. Por ello, tiene la misma variación que la señal `x` eliminando los pulsos cuya duración es inferior a 10 ns y retardándola 10 ns.

PREGUNTA 2 (3.5 puntos)

Escriba en VHDL la **architecture** que describe:

2.a) (0.5 puntos) El comportamiento de un multiplexor con un entrada de selección (`sel`), dos entradas de 8 bits (`x1`, `x2`) y una salida de 8 bits (`y`). Emplee para ello un bloque **process** con una sentencia **case**. La **entity** del circuito es la siguiente:

```
entity mux is port(
   y      :   out std_logic_vector(7 downto 0);
   x1, x2 :   in  std_logic_vector(7 downto 0);
   sel    :   in std_logic);
end entity mux;
```

2.b) (1 punto) El comportamiento de una unidad aritmética con dos entradas `A` y `B`. El tamaño de los dos operandos de entrada, `A` y `B`, es de 8 bits. La operación que realiza es especificada por la señal de entrada `sel`. Emplee en la programación del circuito una sentencia concurrente condicional (**when - else**). A continuación, se muestra la **entity** y la tabla de operaciones correspondiente al circuito.

```
entity uaritm is port(
   y      :   out std_logic_vector(7 downto 0);
   A, B   :   in  std_logic_vector(7 downto 0);
   sel    :   in  std_logic_vector(1 downto 0));
end entity uaritm;
```

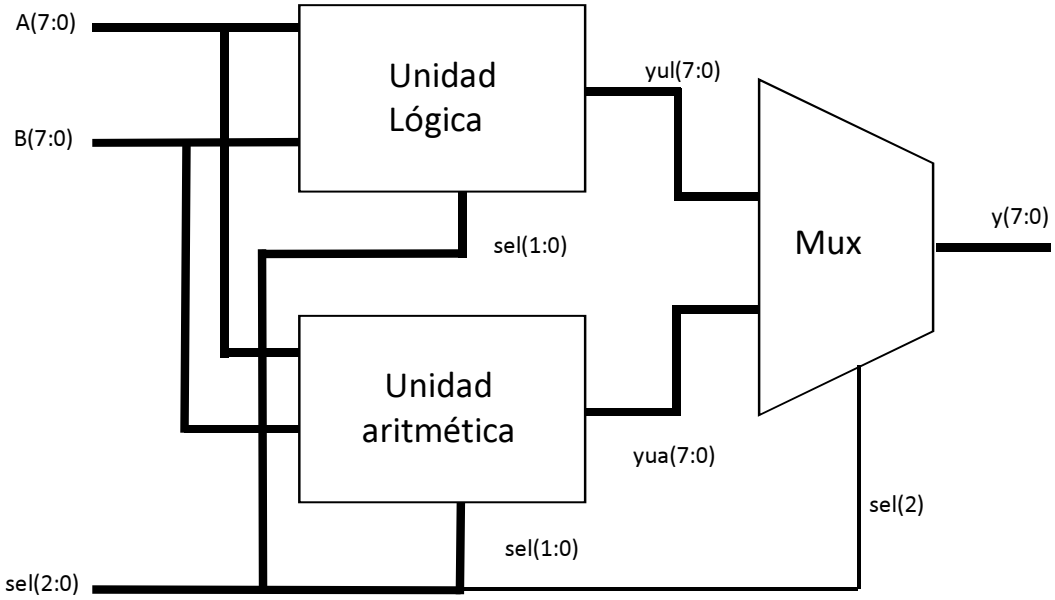
`sel`	Operación
0 0	A
0 1	$A + B$
1 0	A - B
1 1	-A

2.c) (1 punto) El comportamiento de una unidad lógica con dos entradas `A` y `B`. El tamaño de los dos operandos de entrada, `A` y `B`, es de 8 bits. La operación que realiza es especificada por la señal de entrada `sel`. Emplee en la programación del circuito un bloque **process** con una sentencia **if**. A continuación, se muestra la **entity** y la tabla de operaciones correspondiente al circuito.

```
entity ulogic is port(
   y      :   out std_logic_vector(7 downto 0);
   A, B   :   in  std_logic_vector(7 downto 0);
   sel    :   in  std_logic_vector(1 downto 0));
end entity ulogic;
```

`sel`	Operación
0 0	A or B
0 1	A nand B
1 0	A nor B
1 1	A xor B

2.d) (1 punto) La estructura de una unidad aritmético lógica (ALU). La **architecture** debe describir la estructura del circuito combinacional mostrado en la figura, instanciando y conectando adecuadamente los circuitos cuyo diseño ha realizado al contestar los tres apartados anteriores. Los dos bits menos significativos de la señal `sel` seleccionan la operación a realizar por la unidad lógica y por la unidad aritmética. El bit más significativo de la señal `sel` selecciona si la salida de la ALU es la salida de la unidad aritmética o la salida de la unidad lógica.



La **entity** del circuito es:

```
entity alu is port(
   y      :   out std_logic_vector(7 downto 0);
   A, B   :   in  std_logic_vector(7 downto 0);
   sel    :   in  std_logic_vector(2 downto 0));
end entity alu;
```

Solución a la Pregunta 2

El código VHDL de la **architecture** del multiplexor, la unidad aritmética, la unidad lógica y la unidad aritmético lógica se muestran en Código VHDL 1.1–1.4.

```
-------------------------------------------------
-- Multiplexor 2 a 1
library IEEE;
use IEEE.std_logic_1164.all;

entity mux is port(
       y : out std_logic_vector(7 downto 0);
       x1, x2 : in std_logic_vector(7 downto 0);
       sel : in std_logic);
end entity mux;

architecture mux of mux is
begin
   process(x1, x2, sel) is
      begin
      case sel is
            when ('0') =>
               y <= x1;
            when others =>
               y <= x2;
         end case;
   end process;
end architecture mux;
-------------------------------------------------
```

Código VHDL 1.1: Diseño del multiplexor 2 a 1.

```
-------------------------------------------------
-- Unidad aritmética
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity uaritm is port(
      y       :    out std_logic_vector(7 downto 0);
      A, B    :    in  std_logic_vector(7 downto 0);
      sel     :    in std_logic_vector(1 downto 0));
end entity uaritm;

architecture uaritm of uaritm is
begin
   y <=    A when (sel="00")
      else std_logic_vector (signed(A)+signed(B)) when (sel="01")
      else std_logic_vector (signed(A)-signed(B)) when (sel="10")
      else std_logic_vector (-signed(A)          ) ;
end architecture uaritm;
-------------------------------------------------
```

Código VHDL 1.2: Diseño de la unidad aritmética.

```
-------------------------------------------------
-- Unidad lógica
library IEEE;
use IEEE.std_logic_1164.all;

entity ulogic is port(
       y       :     out std_logic_vector(7 downto 0);
       A, B    :     in   std_logic_vector(7 downto 0);
       sel     :     in std_logic_vector(1 downto 0));
end entity ulogic;

architecture ulogic of ulogic is
begin
   process(A, B, sel) is
      begin
         if (sel = "00") then
             y <= A or B;
         elsif (sel = "01") then
             y <= A nand B;
         elsif (sel = "10") then
             y <= A nor B;
         elsif (sel = "11") then
             y <= A xor B;
         end if;
   end process;
end architecture ulogic;
-------------------------------------------------
```

Código VHDL 1.3: Diseño de la unidad lógica.

```
---------------------------------
-- Unidad aritmetico logica
library IEEE;
use IEEE.std_logic_1164.all;

entity alu is port(
      y       :    out std_logic_vector(7 downto 0);
      A, B    :    in   std_logic_vector(7 downto 0);
      sel     :    in   std_logic_vector(2 downto 0));
end entity alu;

architecture alu of alu is
   signal yul, yua: std_logic_vector(7 downto 0);

   component mux is
    port(y : out std_logic_vector(7 downto 0);
         x1, x2 : in std_logic_vector(7 downto 0);
         sel : in std_logic);
   end component mux;

   component uaritm is
     port(    y       :    out std_logic_vector(7 downto 0);
              A, B    :    in   std_logic_vector(7 downto 0);
              sel     :    in std_logic_vector(1 downto 0));
   end component uaritm;

   component ulogic is
      port(   y       :    out std_logic_vector(7 downto 0);
              A, B    :    in   std_logic_vector(7 downto 0);
              sel     :    in std_logic_vector(1 downto 0));
    end component ulogic;

begin
   -- Instanciación y conexión de los componentes
   ulogic_1 : component ulogic port map
            (y => yul, A => A, B => B, sel => sel(1 downto 0) );
   uaritm_1 : component uaritm port map
            (y => yua, A => A, B => B, sel => sel(1 downto 0) );
   mux_1 : component mux port map
            (y => y, x1 => yua, x2 => yul, sel => sel(2) );
end architecture alu;
---------------------------------
```

Código VHDL 1.4: Diseño de la unidad aritmético lógica.

PREGUNTA 3 (2.5 puntos)

Diseñe usando VHDL un registro de desplazamiento de 4 bits conversor serie a paralelo. El circuito tiene las entradas siguientes: señal de reloj (`Clock`), señal de control de desplazamiento hacia la derecha (`Shift`), señal de reset asíncrono activo a nivel alto (`Reset`) y señal de entrada serie de datos (`Serial_in`). El circuito tiene una señal de 4 bits de salida paralelo de datos (`Q`). La **entity** del circuito es:

```
entity registro is port(
   Q : out std_logic_vector(3 downto 0);
   Clock, Shift, Serial_in, Reset : in std_logic);
end entity registro
```

Mientras `Reset` vale '0' y `Shift` vale '1', en cada flanco de subida de la señal de reloj se desplaza el contenido del registro un bit a la derecha, cargándose en el bit más significativo del registro el valor de `Serial_in`.

El diseño del registro en VHDL debe realizarse describiendo el comportamiento del circuito, empleando para ello un único bloque **process**.

Solución a la Pregunta 3

El código VHDL que describe el comportamiento del circuito se muestra en Código VHDL 1.5.

```
-------------------------------------------------
-- Registro serie-a-paralelo
library IEEE;
use IEEE.std_logic_1164.all;

entity registro is port(
   Q : out std_logic_vector(3 downto 0);
   Clock, Shift, Serial_in, Reset : in std_logic);
end entity registro;

architecture registro of registro is
     signal content: std_logic_vector(3 downto 0);
begin
     process(Clock, Reset)
     begin
        if (Reset = '1') then
            content <= "0000";
          elsif(rising_edge(Clock)) then
                if (Shift = '1') then
                   content <= Serial_in & content(3 downto 1);-- desplazamiento de-
recha
                                          --y rellena a la izqda con bits
                                          --entrada Serial_in
                 end if;
          end if;
     end process;

     Q <= content;

end architecture registro;
-------------------------------------------------
```

Código VHDL 1.5: Registro de desplazamiento de 4 bits conversor serie a paralelo.

PREGUNTA 4 (2 puntos)

Programe en VHDL un banco de pruebas para el registro que ha diseñado al contestar a la Pregunta 3. El programa de test debe primero resetear el registro y a continuación cargar en el registro, a través de la entrada serie y por este orden, los cuatro bits siguientes: '0', '1', '1', '1'. Si los valores de la señal de salida de la UUT no coinciden con lo esperado, el programa de test debe mostrar el correspondiente mensaje.

Solución a la Pregunta 4

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

```
-------------------------------------
-- Banco de pruebas del registro
--conversor serie a paralelo
library IEEE;
use IEEE.std_logic_1164.all;

entity bp_registro is
end entity bp_registro;

architecture bp_registro of bp_registro is
   constant PERIODO    : time         := 100 ns; -- Reloj
   signal    Q          : std_logic_vector(3 downto 0);-- Salidas UUT
   signal    Clock      : std_logic    := '0';      -- Entradas UUT
   signal    Shift, Serial_in, Reset   : std_logic;

   component registro is
       port (   Q  : out std_logic_vector(3 downto 0);
                Clock, Shift, Serial_in, Reset : in std_logic);
   end component registro;

    -- 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;
begin
   -- Instanciar y conectar UUT
   uut : component registro port map
                    (Q,   Clock, Shift, Serial_in, Reset);
   reset <=    '0','1' after (PERIODO/4),
               '0' after (PERIODO+PERIODO/4);

   Clock <= not Clock after (PERIODO/2);
   gen_vec_test : process is
      variable error_count : integer := 0; -- Núm. errores
```

Código VHDL 1.6: Diseño del banco de pruebas del registro de desplazamiento de 4 bits conversor serie a paralelo.

```
   begin
     report "Comienza la simulación";
      -- Vectores de test y comprobación del resultado
       Shift <= '0';Serial_in <= '0';
       wait for PERIODO;    -- 1
       comprueba_salidas("0000",q,error_count);
       Shift <= '1';Serial_in <= '0';
       wait for PERIODO;    -- 2
       comprueba_salidas("0000",q,error_count);
       Shift <= '1';Serial_in <= '1';
       wait for PERIODO;    -- 3
       comprueba_salidas("1000",q,error_count);
       Shift <= '1';Serial_in <= '1';
       wait for PERIODO;    -- 4
       comprueba_salidas("1100",q,error_count);
       Shift <= '1';Serial_in <= '1';
       wait for PERIODO;    -- 5
       comprueba_salidas("1110",q,error_count);
       Shift <= '0';
     -- 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_registro;
---------------------------------------
```

Código VHDL 1.7: Continuación del diseño del banco de pruebas del registro de desplazamiento de 4 bits conversor serie a paralelo.