INGENIERÍA DE COMPUTADORES III

Solución al examen de Junio 2012, Primera Semana

PREGUNTA 1 (2 puntos)

Tomando como base el siguiente código VHDL, dibuje el cronograma de evolución de las señales `x1`, `x2`, `x3`, `s`, `y` entre los instantes 0 y 50 ns.

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

entity crono is
end entity crono;

architecture crono of crono is
  signal x1, x3, s, y : std_logic;
  signal x2  : std_logic := '0';
begin
  x1 <= '1', '0' after 10 ns,
        '1' after 20 ns,
        '0' after 30 ns,
        '1' after 40 ns;
  x2 <= '0', '1' after 25 ns;
  x3 <= '0', '1' after 10 ns,
        '0' after 30 ns;
  Proc1: process (x1, x2, x3)
  begin
    s <= x1;
    y <= s or x3;
    s <= X2;
  end process;
end architecture crono;
```

Solución a la Pregunta 1

En la Figura 1.1 se muestra el cronograma de evolución de las señales.

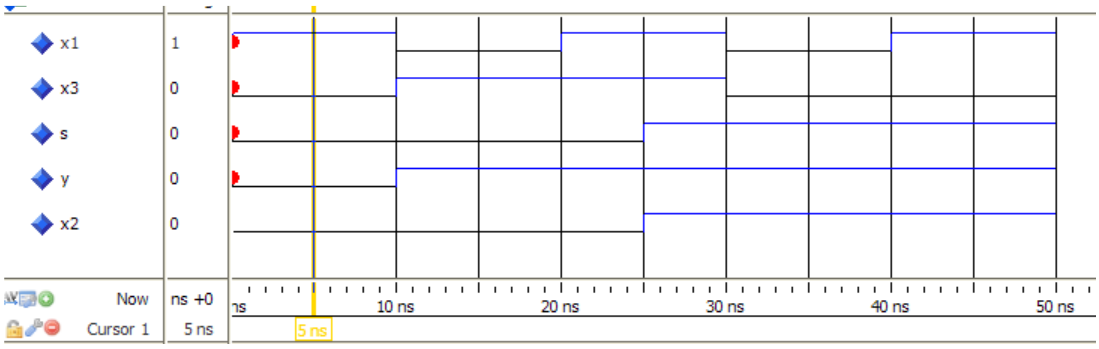


Figura 1.1: Cronograma.

La señal `x2` vale '0' en el instante inicial, mientras que el resto de señales valen 'U'. En dicho instante, se ejecuta el bloque **process**. En consecuencia, se planifica para el instante δ que la señal `s` tome el valor que tiene la señal `x2` en el $t = 0$ ('0'). Obsérvese que la primera sentencia de asignación a la señal `s` no tiene efecto.

En el instante δ la señal `x1` toma el valor '1', la señal `x3` toma el valor '0' y la señal `s` toma el valor '0'. Los cambios en el valor de las señales `x1` y `x3` provocan que se ejecute nuevamente el bloque **process**. Se planifica para el instante $2 \cdot \delta$ que la señal `y` tome el valor resultante de realizar la or lógica de los valores de las señales `s` ('0') y `x3` ('0') en el instante δ.

En el instante $2 \cdot \delta$ la señal `y` toma el valor '0' y el resto de señales no cambian de valor.

En el instante 10 ns la señal `x1` toma el valor '0' y la señal `x3` toma el valor '1'. Ello provoca una nueva ejecución del bloque **process**, planificándose una nueva asignación ala señal `y` para el instante 10 ns $+\delta$. Así, en $t = 10$ ns $+\delta$ se le asigna a `y` el valor '1'.

En el instante 20 ns la señal `x1` toma el valor '1'. Este cambio hace que se ejecute el **process** pero no se planifican nuevas asignaciones a las señales.

En el instante 25 ns la señal `x2` toma el valor '1'. Ello provoca una nueva ejecución del bloque **process**, planificándose la asignación de un nuevo valor a la señal `s` para el instante 25 ns $+\delta$. Así, en $t = 25$ ns $+\delta$ se le asigna a `s` el valor '1'.

En el instante 30 ns la señal `x1` toma el valor '0' y la señal `x3` toma el valor '0'. Ello provoca una nueva ejecución del bloque **process** pero no se planifican nuevas asignaciones las señales `s` e `y`.

En el instante 40 ns la señal `x1` toma el valor '1'. Ello provoca una nueva ejecución del bloque **process**, pero no se planifican nuevas asignaciones las señales `s` e `y`.

PREGUNTA 2 (3 puntos)

Se pretende diseñar un circuito comparador cuya salida (`F`) vale '1' si el valor del número de cuatro bits de entrada (`x`) es menor que el número decimal 9. Los cuatro bits de entrada se interpretan como un número binario sin signo. La **entity** del circuito se muestra a continuación.

```
entity comparaXmenor9 is port
        ( F : out std_logic;
          x : in  std_logic_vector(3 downto 0) );
end entity comparaXmenor9;
```

Escriba en VHDL la **architecture** del circuito comparador de las dos formas siguientes:

2.a) (1 punto) Empleando únicamente asignaciones concurrentes y operadores lógicos.

2.b) (2 puntos) Describiendo su estructura. Dibuje el diagrama del circuito al nivel de puertas lógicas. Escriba en VHDL la **entity** y **architecture** de todos los tipos de puertas lógicas que contiene el diseño del comparador. Finalmente, escriba en VHDL la **architecture** del comparador usando las puertas lógicas que ha definido previamente.

Solución a la Pregunta 2

La **architecture** describiendo el comportamiento del circuito empleando únicamente asignaciones concurrentes y operadores lógicos se muestra en Código VHDL 1.1.

En la Figura 1.2 se muestra el diagrama del circuito comparador al nivel de puertas lógicas. Para su definición, se emplean puertas AND de 2 entradas (and2), OR de 2 entradas (or2) e inversor (not1). Las **entity** y las **architecture** de estas puertas lógicas se muestran en Código VHDL 1.2.

Para componer el circuito comparador usando las puertas lógicas anteriores, es necesario instanciar los componentes necesarios y conectarlos. Para ello, es preciso escribir la **entity** de cada uno de los componentes, instanciar los componentes y describir la conexión entre ellos. El Código VHDL 1.3 describe la estructura del circuito comparador.

```
------------------------------------
-- Comparador F = X<9:
--       F = x3' + x3x2'x1'x0'

library IEEE; use IEEE.std_logic_1164.all;

entity comparaXmenor9 is port
   ( F  : out std_logic;
       x  : in    std_logic_vector(3 downto 0) );
end entity comparaXmenor9;

architecture comparaXmenor9 of comparaXmenor9 is
begin
     F  <= not x(3)
           or (x(3) and not x(2) and not x(1) and not x(0));
end architecture comparaXmenor9;
------------------------------------
```

Código VHDL 1.1: Descripción del comportamiento del comparador mediante asignaciones concurrentes y operadores lógicos.

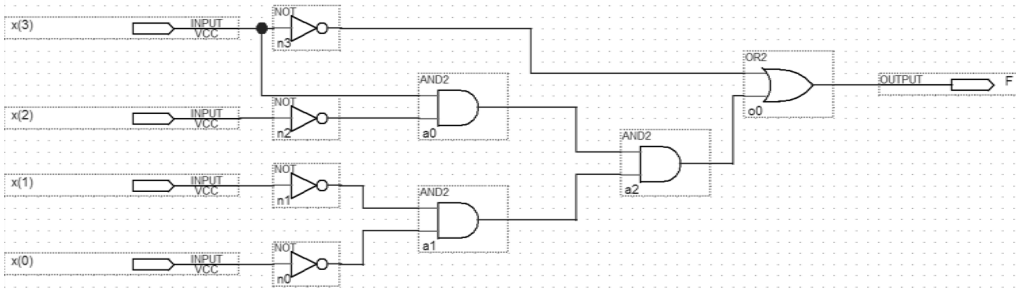


Figura 1.2: Diagrama del circuito comparador.

```
-------------------------------------
-- Inversor de 1 entrada.
library IEEE; use IEEE.std_logic_1164.all;

entity not1 is port
   ( y0   : out std_logic;
     x0   : in    std_logic );
end entity not1;

architecture not1 of not1 is
begin
   y0 <= not x0;
end architecture not1;
-------------------------------------

-------------------------------------
-- Puerta AND de 2 entradas.
library IEEE; use IEEE.std_logic_1164.all;

entity and2 is port
   ( y0       : out std_logic;
     x0, x1   : in    std_logic );
end entity and2;

architecture and2 of and2 is
begin
   y0 <= x0 and x1;
end architecture and2;
-------------------------------------

-------------------------------------
-- OR de 2 entradas: and2
library IEEE; use IEEE.std_logic_1164.all;

entity or2 is port
   ( y0       : out std_logic;
     x0, x1   : in    std_logic );
end entity or2;

architecture or2 of or2 is
begin
   y0 <= x0 or x1;
end architecture or2;
-------------------------------------
```

Código VHDL 1.2: Diseño de las puertas lógicas not1, and2 y or2.

```
-------------------------------------
-- Descripción estructural
--      F = x3' + x3x2'x1'x0'
library IEEE; use IEEE.std_logic_1164.all;

architecture comparaXmenor9Estruc of comparaXmenor9 is
   signal x3n, x2n, x1n, x0n, x3x2n, x1nx0n, x3x2nx1nx0n: std_logic;
   -- Declaración de las clases de los componentes
   component and2 is port
      ( y0          : out std_logic;
        x0, x1      : in  std_logic );
   end component and2;
   component not1 is port
      ( y0          : out std_logic;
        x0          : in  std_logic );
   end component not1;
   component or2 is port
      ( y0          : out std_logic;
        x0, x1      : in  std_logic );
   end component or2;
begin
   -- Instanciación y conexión de los componentes
   n3 : component not1 port map (x3n, x(3));
   n2 : component not1 port map (x2n, x(2));
   n1 : component not1 port map (x1n, x(1));
   n0 : component not1 port map (x0n, x(0));
   a0 : component and2 port map (x3x2n, x(3), x2n);
   a1 : component and2 port map (x1nx0n, x1n, x0n);
   a2 : component and2 port map (x3x2nx1nx0n, x3x2n, x1nx0n);
   o0 : component or2 port map (F, x3n, x3x2nx1nx0n);
end architecture comparaXmenor9Estruc;
---------------------------------
```

Código VHDL 1.3: Descripción de la estructura del circuito comparador.

PREGUNTA 3 (2 puntos)

Programe en VHDL el banco de pruebas del comparador que ha diseñado al resolver la Pregunta 2. El banco de pruebas debe comprobar el funcionamiento del circuito de forma exhaustiva. El banco de pruebas debe generar un conjunto de vectores de test, comprobar si la salida de la UUT es correcta, mostrar un mensaje cada vez que la salida de la UUT no sea correcta y mostrar un mensaje al finalizar el test en el que se indique el número total de salidas incorrectas.

Solución a la Pregunta 3

El código VHDL del banco de pruebas del circuito comparador se muestra en Código VHDL 1.4.

```
-------------------------------------
-- Banco de pruebas comparador X<9
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity bp_comparaXmenor9 is
end entity bp_comparaXmenor9;

architecture bp_comparaXmenor9 of bp_comparaXmenor9 is
   signal F : std_logic;   -- Conectar salidas UUT
   signal x : std_logic_vector(3 downto 0);   -- Conectar entradas UUT

   component comparaXmenor9 is port
      ( F : out std_logic;
        x : in   std_logic_vector(3 downto 0));
   end component comparaXmenor9;

begin
   -- Instanciar y conectar UUT
   uut : component comparaXmenor9 port map
         ( F => F, x => x );
   gen_vec_test : process
      variable test_in      : unsigned (3 downto 0):= B"0000_"; -- Vector de test
      variable num_errores  : integer := 0;              -- Numero de errores
   begin
      for count in 0 to 15 loop
         x(3)  <= test_in(3);
         x(2)  <= test_in(2);
         x(1)  <= test_in(1);
         x(0)  <= test_in(0);
         wait for 10 ns;

-- Comprueba resultado
         if (F='0' and test_in<9) or (F='1' and test_in>=9) then
         report("Error. Valor "&integer'image(to_integer(test_in)));
         num_errores := num_errores + 1;
         end if;
         test_in := test_in + 1;
      end loop;
      report "Test completo. Hay "       &
               integer'image(num_errores) &
               " errores.";
      wait; --Final simulación
   end process gen_vec_test;
end architecture bp_comparaXmenor9;
-------------------------------------
```

Código VHDL 1.4: Diseño del detector de secuencia.

PREGUNTA 4 (3 puntos)

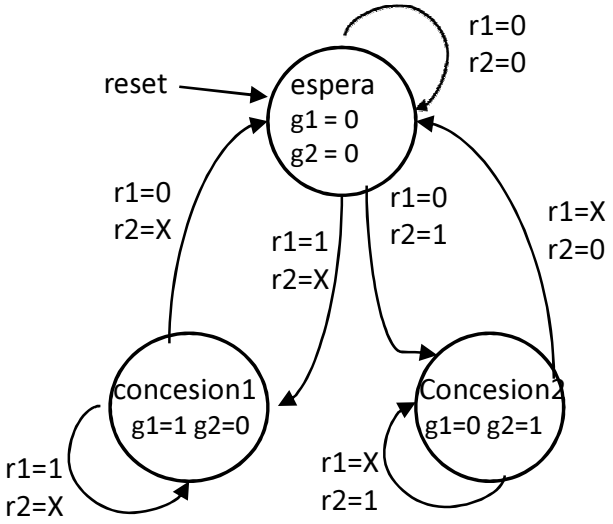
El circuito denominado *árbitro* permite controlar el acceso de varios dispositivos, denominados *clientes*, a un determinado recurso compartido. Este recurso compartido puede ser una memoria, un bus o cualquier componente de uso común. El objetivo del circuito árbitro es permitir en un momento dado el acceso de un único cliente al recurso. Para controlar el acceso de los clientes al recurso por medio del árbitro se usa un sistema de peticiones (*request*) y concesiones (*grant*).

Se quiere diseñar un árbitro que tenga dos clientes, que llamamos cliente1 y cliente2. La comunicación entre el cliente1 y el árbitro se realiza mediante las señales de petición (`r1`) y de concesión (`g1`). Análogamente, la comunicación entre el cliente2 y el árbitro se realiza mediante las señales `r2` y `g2`.

El cliente activa su señal de petición cuando quiere acceder al recurso compartido y el árbitro da acceso al cliente activando su señal de concesión correspondiente. Tras completar la tarea, el cliente libera el recurso y desactiva su señal de petición. Cuando ambos clientes realizan una petición simultánea, cliente1 tiene siempre la prioridad.

El comportamiento del árbitro puede describirse mediante una máquina de estados finitos con tres estados: espera, concesion1 y concesion2. El estado espera indica que el recurso está disponible y el árbitro está esperando peticiones. Los estados concesion1 y concesion2 indican que el recurso se ha dado al cliente1 y al cliente2, respectivamente. El árbitro ha de tener también una señal de reset asíncrona activa a nivel alto.

El diagrama de estados de esta máquina se muestra a continuación. Se ha empleado el símbolo X para indicar que el valor de la señal puede ser 0 ó 1.



Escriba en VHDL la **architecture** que describe el comportamiento de este árbitro de dos clientes como una máquina de estados de Moore síncrona, donde las transiciones tienen lugar en el flanco de subida de la señal de reloj.

La **entity** de este circuito árbitro se muestra a continuación.

```
entity arbitro is port
        ( g1, g2 : out std_logic;
          r1, r2, clk, reset : in  std_logic );
end entity arbitro;
```

Solución a la Pregunta 4

El código VHDL correspondiente al árbitro se muestra en Código VHDL 1.5.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity arbitro is port
( g1,g2 : out std_logic;
  r1,r2,clk,reset : in  std_logic );
end entity arbitro;

architecture comportamiento OF arbitro IS
type tipo_de_estado is ( espera,concesion1,concesion2 );
signal estado :tipo_de_estado;
begin
process( reset,clk )
begin
 if reset='1' then
    estado <= espera;
 elsif ( rising_edge(clk)) then
      case estado is
          when espera =>
            if r1 = '1' then estado<=concesion1;
            elsif r2 = '1' then estado<=concesion2;
            else estado<=espera;
            end if;
          when concesion1 =>
            if r1 = '1' then estado<=concesion1;
            else estado<=espera;
            end if;
          when concesion2 =>
            if r2 = '1' then estado<=concesion2;
            else estado<=espera;
            end if;
       end case;
 end if;
end process;
g1 <= '1' when estado=concesion1 else '0';
g2 <= '1' when estado=concesion2 else '0';
end comportamiento;
```

Código VHDL 1.5: Diseño del circuito árbitro.