INGENIERÍA DE COMPUTADORES III

Solución al Ejercicio de Autocomprobación 7

PREGUNTA 1 (2 puntos)

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

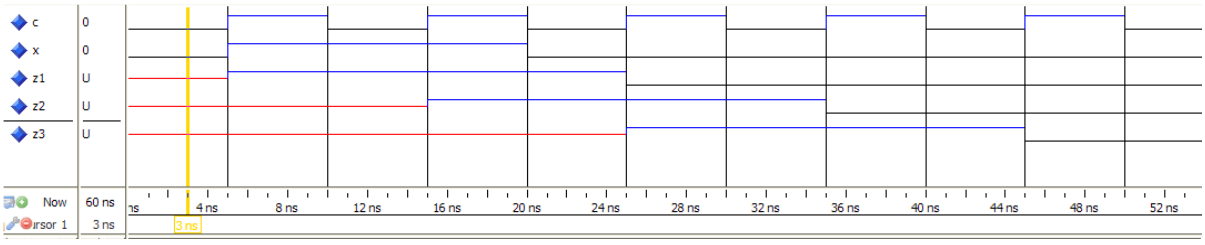
```
library IEEE;
use IEEE.std_logic_1164.all;

entity crono is
end entity crono;

architecture crono of crono is
   constant PER : time :=10 ns;
   signal c: std_logic:='0';
   signal x: std_logic;
   signal z1, z2, z3: std_logic;
begin
   process(c)
   begin
     if (rising_edge(c)) then
       z1<=x;
       z2<=z1;
       z3<=z2;
     end if;
   end process;
   c<=not c after (PER/2);
   x<= '0', '1' after 5 ns, '0' after 20 ns;
end architecture crono;
```

Solución a la Pregunta 1

A continuación se muestra el cronograma de evolución de las señales `c`, `x`, `z1`, `z2` y `z3` entre los instantes 0 y 60 ns.



El bloque **process** se ejecuta cuando la señal `c` cambia su valor y el contenido de la sentencia **if** de este bloque se ejecuta únicamente el flanco de subida de la señal `c`, que se produce cada 10 ns. Es decir, en los instante 5 ns, 15 ns, 25 ns, 35 ns, 45 ns, ...

Por tanto, las asignaciones incluidas en la sentencia **if** se evalúan cada 10 ns empezando en el instante 5 ns (5 ns, 15 ns, 25 ns, 35 ns, 45 ns, ...). Pero las asignaciones a señal se producen tras un retardo δ, teniendo lugar en los instantes 5 ns+δ, 15 ns+δ, 25 ns+δ, 35 ns+δ, 45 ns+δ, ...

En consecuencia, la asignación a la señal `z1` se efectúa en los instantes 5 ns+δ, 15 ns+δ, 25 ns+δ, 35 ns+δ, 45 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 5 ns, 15 ns, 25 ns, 35 ns, 45 ns, ... Por ello la señal `z1` toma en el instante 5 ns+δ el valor que tenía la señal `x` en el instante 5 ns. En el instante 15 ns+δ toma el valor que tenía la señal `x` en el instante 15 ns, que coincide con el valor de la señal `x` en el instante 5 ns +δ. Finalmente, el último cambio de valor de la señal `z1` se produce en el instante 25 ns+δ, tomando el valor que tenía la señal `x` en el instante 25 ns.

Siguiendo un razonamiento análogo se concluye que las señales `z2` y `z1` toman los mismos valores pero con un desfase de tiempo de 10 ns. Análogamente, las señales `z3` y `z2` toman los mismos valores pero con un desfase de tiempo de 10 ns.

PREGUNTA 2 (3 puntos)

Escriba en VHDL, de las tres formas que se detallan a continuación, la **architecture** que describe el comportamiento de un circuito combinacional conversor de código binario de 3 bits a código Gray. La **entity** del circuito y su tabla de verdad son las siguientes:

```
entity codGray is
   port (  y          : out std_logic_vector(2 downto 0);
           x          : in  std_logic_vector(2 downto 0));
end entity codGray;
```

x	y
000	000
001	001
010	011
011	010
100	110
101	111
110	101
111	100

2.a) (1 punto) Empleando un bloque **process** con una sentencia **case**.

2.b) (1 punto) Empleando una asignación concurrente de selección (**with - select**).

2.c) (1 punto) Realizando una descripción de su estructura empleando para ello dos puertas lógicas XOR de dos entradas. Tenga en cuenta que el bit en la posición i de una palabra del código Gray es 1 si los bits en las posiciones i e $i+1$ de la correspondiente palabra binaria son diferentes.

La **entity** de la puerta lógica XOR se muestra a continuación.

```
entity xor2 is
  port ( y0     : out std_logic;
         x0, x1 : in  std_logic );
end entity xor2;
```

Solución a la Pregunta 2

El diseño del conversor realizado empleando un bloque **process** con una sentencia **case**, una asignación concurrente de selección (**with - select**) y siguiendo una descripción de su estructura empleando puertas lógicas XOR de dos entradas se muestra respectivamente en Código VHDL 1.1– 1.3.

```
-------------------------------------------------
-- Codificador binario a código Gray
-- empleando un bloque process con una sentencia case
library IEEE;
use IEEE.std_logic_1164.all;


architecture arch_codGrayCase of codGray is
begin
   process(x) is
      begin
      case x is
            when ("000") =>      y <= "000";
            when ("001") =>      y<="001";
            when ("010") =>      y<="011";
            when ("011") =>      y<="010";
            when ("100") =>      y<="110";
            when ("101") =>      y<="111";
            when ("110") =>      y<="101";
            when ("111") =>      y<="100";
            when others =>       y<="000";
      end case;
   end process;

end architecture arch_codGrayCase;
-------------------------------------------------
```

Código VHDL 1.1: Diseño del conversor empleando un bloque **process** con una sentencia **case**.

```
-------------------------------------------------
-- Codificador binario a código Gray
-- asignacion concurrente de seleccion
library IEEE;
use IEEE.std_logic_1164.all;


architecture arch_codGraySelect of codGray is
begin
        with x   select
        y <= "000" when ("000"),
             "001" when ("001"),
             "011" when ("010"),
             "010" when ("011"),
             "110" when ("100"),
             "111" when ("101"),
             "101" when ("110"),
             "100" when ("111"),
             "000" when others;
end architecture arch_codGraySelect;
-------------------------------------------------
```

Código VHDL 1.2: Diseño del conversor empleando una asignación concurrente de selección (**with - select**).

```
-----------------------------------
-- Conversor binario a Gray
-- descripcion estructural usando
-- 2 puertas XOR de 2 entradas
library IEEE;
use IEEE.std_logic_1164.all;

architecture arch_codGrayEst of codGray is
   component xor2 is
     port( y0       : out std_logic;
           x0, x1 : in   std_logic);
   end component xor2;
begin
   y(2) <= x(2);
   -- Instanciación y conexión de los componentes
   xor2_1 : component xor2 port map
            (y0 => y(1), x0 => x(2), x1 => x(1) );
   xor2_2 : component xor2 port map
            (y0 => y(0), x0 => x(1), x1 => x(0) );
end architecture arch_codGrayEst;
-----------------------------------
```

Código VHDL 1.3: Diseño del conversor siguiendo una descripción de su estructura.

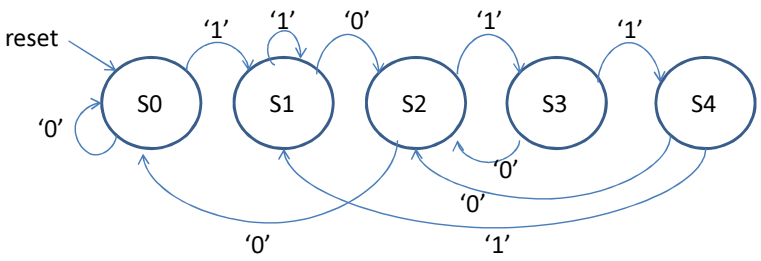
PREGUNTA 3 (3 puntos)

Diseñe un circuito secuencial síncrono capaz de detectar cuando le llega la secuencia "1011" por su entrada. La **entity** del circuito se muestra a continuación. El circuito tiene una señal de reloj (`clk`), un entrada serie de un bit (`X`), una señal de reset asíncrona activa en '1' (`reset`), una señal que indica el estado en que se encuentra el circuito (`state`) y una señal de salida de un bit (`Y`). La señal `Y` se pone a '1' si por la entrada `X` los últimos 4 bits que han llegado se corresponden con la secuencia "1011". La máquina no vuelve al estado inicial tras haber reconocido la sencuencia y detecta secuencias solapadas. La señal `reset` pone el circuito en su estado inicial. Todos los cambios tienen lugar en el flanco de subida de la señal de reloj. Escriba en VHDL la **architecture** que describe el comportamiento del circuito en términos de una máquina de Moore. Dibuje el diagrama de estados correspondiente al circuito que ha diseñado.

```
entity detector is
   port( Y     : out std_logic;
         state : out std_logic_vector(2 downto 0);
         X     : in std_logic;
         reset : in std_logic;
         clk   : in std_logic);
end entity detector;
```

Solución a la Pregunta 3

El circuito diseñado tiene 5 estados (S0, S1, S2, S3 y S4). El circuito se encuentra en el estado S0 cuando no se ha detectado ningún bit de la secuencia. Y está en los estados S1, S2, S3 o S4 cuando se han detectado respectivamente 1, 2, 3 o 4 primeros bits de la secuencia. A continuación se muestra el diagrama de estados de dicho circuito.



El código VHDL que describe el comportamiento del circuito en términos de una máquina de Moore se muestra en Código VHDL 1.4– 1.5.

```
-----------------------------------------------------
---Detector de la secuencia 1011
-----------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;

entity detector is
    port( Y       : out std_logic;
          state   : out std_logic_vector(2 downto 0);
          X       : in std_logic;
          reset   : in std_logic;
          clk     : in std_logic);
end entity detector;

architecture detector of detector is
       signal internal_state: std_logic_vector(2 downto 0);
begin
 state <= internal_state;

  --Cálculo salida
 salida: process (internal_state) is
 begin
    case internal_state is
         when "100" => Y <= '1';
         when others => Y <= '0';
      end case;
 end process salida;
```

Código VHDL 1.4: Diseño del detector de secuencia.

```
--Cálculo del próximo estado
 proximo_estado: process (clk, reset)
 begin
      if (reset = '1') then --reset asíncrono
           internal_state <= "000";
      elsif (rising_edge(clk)) then
           case internal_state is
                when "000" =>     -- Estado actual: S0
                     if X = '1' then
                          internal_state <= "001";
                     else
                          internal_state <= "000";
                    end if;
                when "001" =>    --Estado actual: S1
                     if X = '1' then
                          internal_state <= "001";
                     else
                          internal_state <= "010";
                     end if;
                when "010" =>    --Estado actual: S2
                     if X = '1' then
                           internal_state <= "011";
                     else
                           internal_state <= "000";
                    end if;
                 when "011" =>    -- Estado actual: S3
                      if X = '1' then
                           internal_state <= "100";
                      else
                           internal_state <= "010";
                     end if;
                 when "100" =>    -- Estado actual: S4
                      if X = '1' then
                           internal_state <= "001";
                      else
                           internal_state <= "010";
                     end if;
                 when others=>    -- Por completitud
                      internal_state <= "000";
                end case;
           end if;
end process proximo_estado;

end architecture detector;
-----------------------------------------
```

Código VHDL 1.5: Continuación del diseño del detector de secuencia.

PREGUNTA 4 (2 puntos)

Programe en VHDL un banco de pruebas para el circuito secuencial que ha diseñado al contestar a la Pregunta 3. El programa de test debe primero resetear el circuito y a continuación cargar en el circuito, a través de la entrada (X) y por este orden, los siete bits siguientes: '1', '0', '1', '1', '0', '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 detector de secuencia diseñado en la Pregunta 3 se muestra en Código VHDL 1.6 y en Código VHDL 1.7. Este banco de pruebas comprueba que al resetear el circuito éste pasa al estado "0000" cuya salida (Y) vale '0'. Después, para cada carga del bit de entrada (X) comprueba que el circuito pasa al estado que le corresponde y que la salida (Y) tiene el valor correcto. Finalmente, el programa muestra un mensaje con el número total de errores detectados.

```
-------------------------------------------------------
-- Banco de pruebas del detector de secuencias
library IEEE;
use IEEE.std_logic_1164.all;

entity bp_detector is
end entity bp_detector;

architecture bp_detector of bp_detector is
   constant PERIODO    : time         := 100 ns; -- Reloj
   signal   state      : std_logic_vector(2 downto 0);-- Salidas UUT
   signal   Y          : std_logic;
   signal   clk        : std_logic   := '0';      -- Entradas UUT
   signal   reset, X   : std_logic;


   component detector is
    port( Y       : out std_logic;
          state   : out std_logic_vector(2 downto 0);
          X       : in std_logic;
          reset   : in std_logic;
          clk     : in std_logic);
   end component detector;

    -- Procedimiento para comprobar las salidas
    procedure comprueba_salidas
        ( esperado_state  :        std_logic_vector(2 downto 0);
          actual_state    :         std_logic_vector(2 downto 0);
          esperado_Y      :         std_logic;
          actual_Y        :         std_logic;
          error_count     :         inout integer) is
    begin
         -- Comprueba state
         if ( esperado_state /= actual_state ) then
             report "ERROR: Estado esperado ("      &
                    std_logic'image(esperado_state(2))     &
                    std_logic'image(esperado_state(1))     &
                    std_logic'image(esperado_state(0))     &
                    "), estado actual ("           &
                    std_logic'image(actual_state(2))       &
                    std_logic'image(actual_state(1))       &
                    std_logic'image(actual_state(0))       &
                    "), instante: "                &
                    time'image(now);
             error_count := error_count + 1;
         end if;          -- Comprueba Y
         if ( esperado_Y /= actual_Y ) then
             report "ERROR: Salida Y esperada ("      &
                    std_logic'image(esperado_Y)     &
                    "), salida actual ("            &
                    std_logic'image(actual_Y)       &
                    "), 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 detector de secuencias.

```
begin
   -- Instanciar y conectar UUT
   uut : component detector port map
                    (Y, state, X, reset, clk);

   reset <=   '0','1' after (PERIODO/4),
              '0' after (PERIODO/2);
   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 3*PERIODO/4;
         comprueba_salidas("000",state,'0',Y,error_count);
         X <= '1';    wait for PERIODO;    -- 1
         comprueba_salidas("001",state,'0',Y,error_count);
         X <= '0';    wait for PERIODO;    -- 2
         comprueba_salidas("010",state,'0',Y,error_count);
         X <= '1';    wait for PERIODO;    -- 3
         comprueba_salidas("011",state,'0',Y,error_count);
         X <= '1';    wait for PERIODO;    -- 4
         comprueba_salidas("100",state,'1',Y,error_count);
         X <= '0';    wait for PERIODO;    -- 5
         comprueba_salidas("010" ,state,'0',Y,error_count);
         X <= '1';    wait for PERIODO;    -- 6
         comprueba_salidas("011",state,'0',Y,error_count);
         X <= '1';    wait for PERIODO;    -- 7
         comprueba_salidas("100",state,'1',Y,error_count);
      -- Informe final
          report   "Hay "                &
                   integer'image(error_count ) &
                   " errores.";
      wait;        -- Final del bloque process
   end process gen_vec_test;
end architecture bp_detector;
-------------------------------------
```

Código VHDL 1.7: Continuación del diseño del banco de pruebas del detector de secuencias.