INGENIERÍA DE COMPUTADORES III

Solución al Ejercicio de Autocomprobación 10

PREGUNTA 1 (1.5 puntos)

Dibuje el diagrama conceptual correspondiente a:

1.a) (0.75 puntos) Las sentencias **if** anidadas siguientes:

```
if boolean_expr_1 then
   if boolean_expr_2 then
      a <= valor_expr_a_1;
   else
      a <= valor_expr_a_2;
   end if;
else
   if boolean_expr_3 then
      a <= valor_expr_a_3;
   else
      a <= valor_expr_a_4;
   end if;
end if;
```

1.b) (0.75 puntos) La sentencia **case** siguiente, suponiendo que `expr_case` tiene cinco posibles valores: `c0`, `c1`, `c2`, `c3` y `c4`.

```
case expr_case is
   when c0 =>
      a <= valor_expr_a_0;
      b <= valor_expr_b_0;
   when c1 =>
      a <= valor_expr_a_1;
      b <= valor_expr_b_1;
   when others =>
      a <= valor_expr_a_n;
      b <= valor_expr_b_n;
end case;
```

Solución a la Pregunta 1

En las Figuras 1.1 y 1.2 se muestran los diagramas conceptuales.

```
if boolean_expr_1 then
    if boolean_expr_2 then
        a <= valor_expr_a_1;
    else
        a <= valor_expr_a_2;
    end if;
else
    if boolean_expr_3 then
        a <= valor_expr_a_3;
    else
        a <= valor_expr_a_4;
    end if;
end if;
```

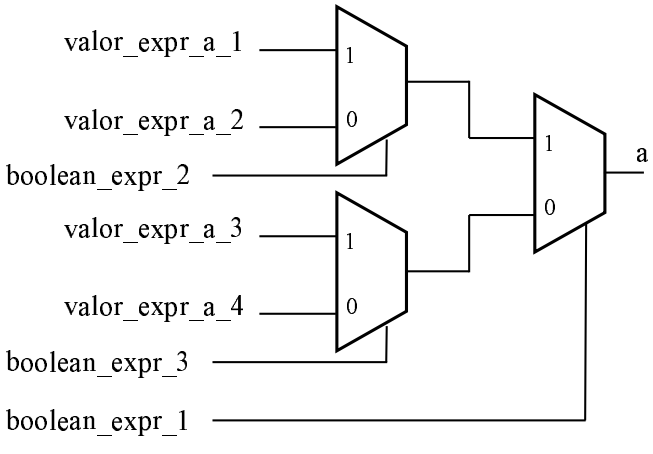

Figura 1.1: Solución a la Pregunta 1.a: sentencia **if** con su diagrama conceptual.

```
case expr_case is
    when c0 =>
      a <= valor_expr_a_0;
      b <= valor_expr_b_0;
    when c1 =>
      a <= valor_expr_a_1;
      b <= valor_expr_b_1;
    when others =>
      a <= valor_expr_a_n;
      b <= valor_expr_b_n;
 end case;
```

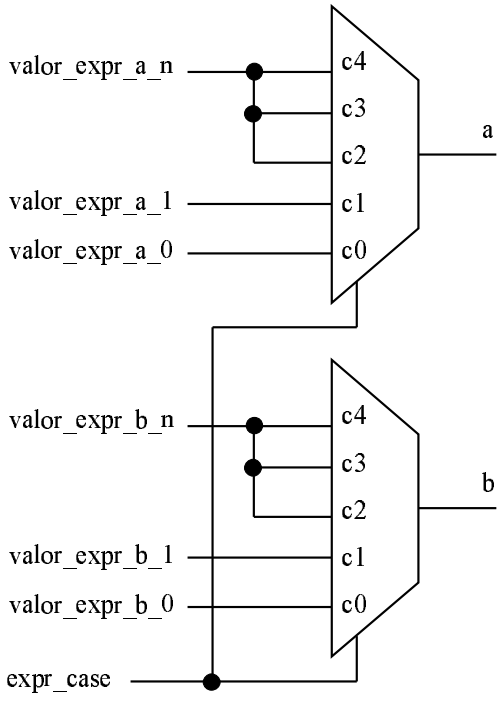

Figura 1.2: Solución a la Pregunta 1.b: sentencia **case** con su diagrama conceptual.

PREGUNTA 2 (3.5 puntos)

A continuación, se muestra la **entity** del circuito *restador completo de 1 bit*.

```
entity rest_completo is
   port ( res, acarreo_out : out std_logic;
          a, b, acarreo_in : in  std_logic );
end entity rest_completo;
```

2.a) (1 punto) Escriba en VHDL una **architecture** que describa el *comportamiento* del circuito.

2.b) (1 punto) Dibuje el diagrama del circuito restador completo de 1 bit al nivel de puertas lógicas. A continuación, escriba en VHDL la **entity** y la **architecture** de cada una de las puertas lógicas que componen el circuito que acaba de dibujar.

2.c) (1.5 puntos) Escriba en VHDL una **architecture** que describa la *estructura* del circuito restador completo de 1 bit que ha dibujado, instanciando y conectando las puertas lógicas que ha diseñado anteriormente.

Solución a la Pregunta 2

La **architecture** describiendo el comportamiento del circuito se muestra en Código VHDL 1.1.

En la Figura 1.3 se muestra un circuito restador completo de 1 bit. Para su definición, se emplean puertas OR-exclusiva de 2 entradas (xor2), AND de dos entradas (and2), OR de tres entradas (or3) e inversor (not1). Las **entity** y las **architecture** de estas puertas lógicas se muestran en Código VHDL 1.2.

Para componer el circuito restador 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 restador completo.

```
----------------------------------------
-- Restador completo de 1 bit
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

architecture rest_completo_comport of rest_completo is
   signal result :unsigned (1 downto 0 );
begin
   result      <= ('0' & a) - ('0' & b) - ('0' & acarreo_in);
   acarreo_out <= result(1);
   res         <= result(0);
end architecture rest_completo_comport;
----------------------------------------
```

Código VHDL 1.1: Descripción del comportamiento de un restador completo de un bit.

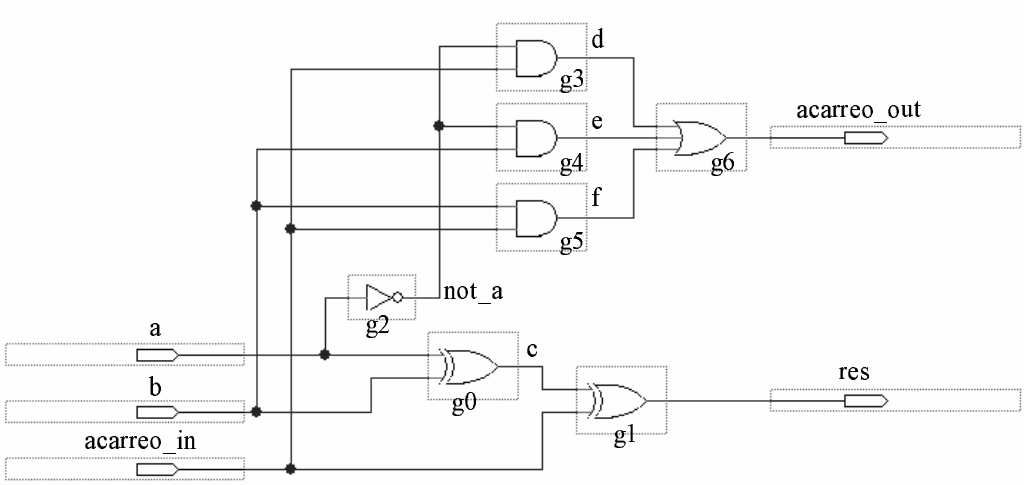

Figura 1.3: Diagrama de un circuito restador completo de 1 bit.

```
---------------------------------------
-- Puerta OR exclusiva de 2 entradas.
library IEEE; use IEEE.std_logic_1164.all;

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

architecture xor2 of xor2 is
begin
   y0 <= x0 xor x1;
end architecture xor2;
---------------------------------------

---------------------------------------
-- 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;
---------------------------------------

---------------------------------------
-- Puerta OR de 3 entradas.
library IEEE; use IEEE.std_logic_1164.all;

entity or3 is port
   ( y0            : out std_logic;
     x0, x1, x2 : in    std_logic );
end entity or3;

architecture or3 of or3 is
begin
   y0 <= x0 or x1 or x2;
end architecture or3;
---------------------------------------
```

Código VHDL 1.2: Diseño de las puertas lógicas XOR2, NOT1, AND2 y OR3.

```
---------------------------------
-- Restador completo de 1 bit
library IEEE;
use IEEE.std_logic_1164.all;

architecture rest_completo_estruc of rest_completo is

   signal not_a, c, d, e, f : std_logic;

   -- Declaración de las clases de los componentes
   component xor2 is port
      ( y0          : out std_logic;
        x0, x1      : in  std_logic );
   end component xor2;

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

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

   component or3 is port
      ( y0          : out std_logic;
        x0, x1, x2  : in  std_logic );
   end component or3;

begin

   -- Instanciación y conexión de los componentes
   g0 : component xor2 port map (c, a, b);
   g1 : component xor2 port map (res, c, acarreo_in);
   -- g2 : component not1 port map (not_a, a);
   g2 : component not1 port map (y0 => not_a, x0 => a);
   g3 : component and2 port map (d, not_a, acarreo_in);
   g4 : component and2 port map (e, not_a, b);
   g5 : component and2 port map (f, b, acarreo_in);
   g6 : component or3  port map (acarreo_out, d, e, f);

end architecture rest_completo_estruc;
---------------------------------
```

Código VHDL 1.3: Descripción de la estructura de un restador completo de un bit.

PREGUNTA 3 (3 puntos)

Se propone diseñar un circuito secuencial síncrono capaz de detectar cuándo recibe la secuencia "0100" por su entrada `X`. La **entity** del circuito se muestra a continuación.

```
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;
```

El circuito tiene una señal de reloj (`clk`), una 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 los últimos 4 bits recibidos por la entrada `X` se corresponden con la secuencia "0100". La máquina no vuelve al estado inicial tras haber reconocido la secuencia, sino que 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.

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. Está en los estados S1, S2, S3 o S4 cuando se han detectado respectivamente 1, 2, 3 ó 4 primeros bits de la secuencia. En la Figura 1.4, 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.

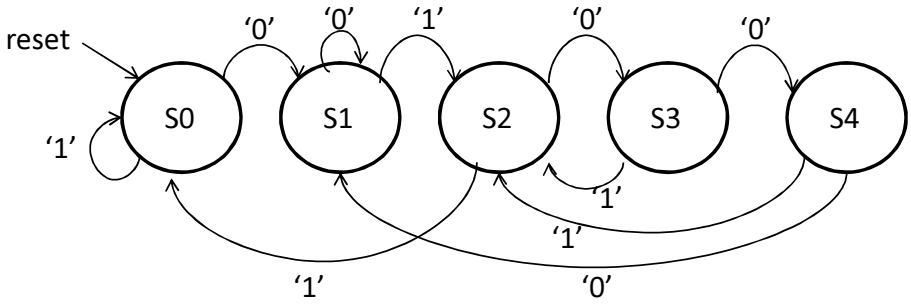

Figura 1.4: Diagrama de estados del circuito.

```
-------------------------------------------------------
---Detector de la secuencia 0100
-------------------------------------------------------
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 <= "000";
                      else
                           internal_state <= "001";
                      end if;
                 when "001" =>    --Estado actual: S1
                      if X = '1' then
                           internal_state <= "010";
                      else
                           internal_state <= "001";
                      end if;
                 when "010" =>    --Estado actual: S2
                      if X = '1' then
                            internal_state <= "000";
                      else
                            internal_state <= "011";
                      end if;
                  when "011" =>    -- Estado actual: S3
                       if X = '1' then
                            internal_state <= "010";
                       else
                            internal_state <= "100";
                       end if;
                  when "100" =>    -- Estado actual: S4
                       if X = '1' then
                            internal_state <= "010";
                       else
                            internal_state <= "001";
                       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, los siete bits siguientes (en el orden indicado): '0', '1', '0', '0', '1', '0', '0'. 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
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
       X <= '0';   wait for PERIODO;    -- 1
       comprueba_salidas("001",state,'0',Y,error_count);
       X <= '1';   wait for PERIODO;    -- 2
       comprueba_salidas("010",state,'0',Y,error_count);
       X <= '0';   wait for PERIODO;    -- 3
       comprueba_salidas("011",state,'0',Y,error_count);
       X <= '0';   wait for PERIODO;    -- 4
       comprueba_salidas("100",state,'1',Y,error_count);
       X <= '1';   wait for PERIODO;    -- 5
       comprueba_salidas("010" ,state,'0',Y,error_count);
       X <= '0';   wait for PERIODO;    -- 6
       comprueba_salidas("011",state,'0',Y,error_count);
       X <= '0';   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.