INGENIERÍA DE COMPUTADORES III

Solución al Ejercicio de Autocomprobación 2

PREGUNTA 1 (3 puntos)

Escriba en VHDL la **architecture** que describe el comportamiento de un divisor de frecuencias por 3, con señal de reset asíncrona activa a nivel bajo. El código VHDL de la **entity** del divisor de frecuencias se muestra a continuación.

```
entity divisor_frecuencia_3 is port(
    clk3       :  out std_logic;
    clk, resetn:  in  std_logic );
end entity divisor_frecuencia_3;
```

La salida `clk3` es una señal periódica con la misma forma de onda que la señal `clk`, pero con un tercio de su frecuencia. Es decir, un periodo de la señal `clk3` se corresponde con tres periodos de la señal `clk`. Los cambios síncronos en la señal `clk3` se producen en el flanco de la señal `clk`. La entrada `resetn` pone asíncronamente la señal `clk3` a 0, manteniéndose dicho valor mientras el valor de `resetn` valga 0.

Puede tomar las decisiones de diseño que estime convenientes, siempre y cuando las argumente y no estén en contradicción con las especificaciones anteriores.

Solución a la Pregunta 1

El Código VHDL 1.1, que se muestra a continuación, es un posible diseño del divisor de frecuencias por 3.

```
---------------------------------
--Divisor de frecuencias por 3
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity divisor_frecuencia_3 is
port (
    clk3          : out std_logic;
    clk, resetn : in std_logic );
end divisor_frecuencia_3;

architecture divisor_frecuencia_3 of divisor_frecuencia_3 is
signal rise,fall: std_logic;
begin
process (clk,resetn) --Calculo señal fall
variable r_cnt : integer range 0 to 2;--Variable contadora
begin
if resetn = '0' then
   r_cnt := 0;
   rise <= '0';
elsif rising_edge(clk) then
   if r_cnt = 2 then
      r_cnt := 0;
      rise <= not rise;
   else
      r_cnt := r_cnt + 1;
   end if;
end if;
end process;

process (clk, resetn) --Calculo señal rise
variable f_cnt : integer range 0 to 2;--Variable contadora
begin
if resetn = '0' then
   f_cnt := 2;
   fall <= '1';
elsif falling_edge(clk) then
   if f_cnt = 2 then
        f_cnt := 0;
        fall <= not fall;
   else
        f_cnt := f_cnt + 1;
   end if;
end if;
end process;
clk3 <= fall xnor rise;
end divisor_frecuencia_3;
 ---------------------------------
```

Código VHDL 1.1: Diseño del divisor de frecuencias por 3.

En el diseño anterior se han empleado dos bloques **process**, sensibles a la señal de reloj y a la señal `resetn`.

En el primero de estos bloques se genera una señal llamada `rise` y en el segundo una señal denominada `fall`. Estas dos señales tienen un sexto de la frecuencia de la señal `clk` y un cierto desfase entre ambas, de modo que, en el semiperiodo en que la señal `rise` tiene valor '1', la señal fall tiene valor '0' la mitad del tiempo y después pasa a tener valor '1'.

La señal `clk3` se genera haciendo la operación lógica xnor de las señales `rise` y `fall`.

En la Figura 1.1 se muestra el cronograma de evolución las señales `clk`, `resetn`, `rise`, `fall` y `clk3`.

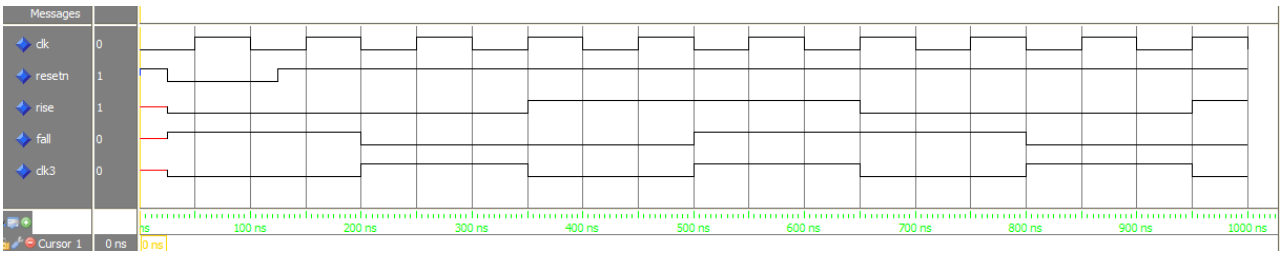

Figura 1.1: Cronograma de las señales clk, resetn, rise, fall y clk3.

PREGUNTA 2 (2 puntos)

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

2.a) (0.25 puntos) El comportamiento de una puerta NOT.

2.b) (0.25 puntos) El comportamiento de una puerta XOR de 2 entradas.

2.c) (1.5 puntos) La estructura de un circuito combinacional detector de paridad de números de n bits, con $n >= 2$. La salida del circuito es 1 si la entrada tiene un número par de unos. En cualquier otro caso, la salida del circuito es 0. La **architecture** debe describir la estructura del circuito combinacional, instanciando y conectando adecuadamente las puertas lógicas NOT y XOR necesarias. Emplee las sentencias **generic**, **generate** y las puertas lógicas cuyo diseño ha realizado al contestar los dos apartados anteriores.

Solución a la Pregunta 2

El Código VHDL 1.2 es una posible forma de diseñar la puerta NOT de una entrada y la puerta XOR de dos entradas.

```
-------------------------------------
-- NOT de 1 entrada
library IEEE; use IEEE.std_logic_1164.all;

entity not1 is
   port ( y        : out std_logic;
          x        : in    std_logic );
end entity not1;

architecture not1 of not1 is
begin
   y <= not x;
end architecture not1;
-------------------------------------
-- XOR 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;
-------------------------------------
```

Código VHDL 1.2: Diseño de las puertas NOT y XOR de 2 entradas.

El Código VHDL 1.3 es un posible diseño del detector de paridad de números de n bits. Con el fin de ilustrar el tipo de circuito diseñado, en la Figura 1.2 se muestra un diagrama esquemático del circuito detector para $n = 3$.

```
-----------------------------------------------
-- Detector de paridad usando sentencia GENERATE condicional
library IEEE;
use IEEE.std_logic_1164.all;

entity paridad is
     generic ( n : integer := 3);
     port ( parity       : out    std_logic;
            parity_IN    : in     std_logic_vector(n-1 downto 0));
end entity paridad;

architecture paridad of paridad is
   signal temp: std_logic_vector(n-2 downto 0);
   signal tempn : std_logic;
   component xor2 is
      port ( y0       : out std_logic;
             x0, x1 : in    std_logic );
   end component xor2;

   component not1 is
      port ( y       : out std_logic;
             x       : in    std_logic );
   end component not1;
begin
    gen_array: for I in 0 to n-2 generate
        inicial: if I = 0 generate
          xor_0: xor2
          port map(temp(I), parity_IN(0), parity_IN(I+1));
        end generate inicial;

        intermedio: if I /= 0 generate
          resto_xor: xor2
          port map(temp(I), temp(I-1), parity_IN(I+1));
        end generate intermedio;
   end generate gen_array;
   not1_0: not1
   port map(parity, temp(n-2));
end architecture paridad;
-----------------------------------------------------
```

Código VHDL 1.3: Diseño del detector de paridad.

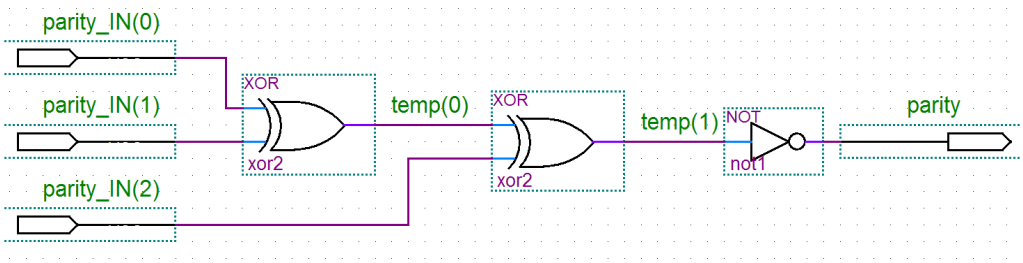

Figura 1.2: Circuito detector de paridad de números de tres bits.

PREGUNTA 3 (3 puntos)

Programe en VHDL el banco de pruebas del circuito combinacional que ha diseñado al contestar a la Pregunta 2c. Suponga que el número de bits que tiene como entrada el circuito es 3 (es decir, $n = 3$). Explique detalladamente cómo el programa de test comprueba exhaustivamente el valor de la UUT para todos los posibles valores de la entrada. 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 3

El Código VHDL 1.4, mostrado en la página siguiente, es un posible banco de pruebas del circuito detector de paridad de tres bits.

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

entity bp_detector_paridad is
end entity bp_detector_paridad;

architecture bp_detector_paridad of bp_detector_paridad is
   signal parity      : std_logic;   -- Conectar salidas UUT
   signal parity_IN : std_logic_vector( 2  downto 0);   -- Conectar entradas UUT

   component paridad is
       generic (n: integer := 3);
       port (   parity      : out    std_logic;
             parity_IN   : in     std_logic_vector(n-1  downto 0));
   end component paridad;

begin
   -- Instanciar y conectar UUT
   uut  : component paridad
         port map( parity, parity_IN );
   gen_vec_test  : process
      variable test_in      : unsigned  (2  downto 0); -- Vector de test
      variable esperado_parity     : std_logic;
      variable error_count          : integer     := 0;
   begin
      test_in := B"000";
      report "Comienza la simulación";
      for count  in 0  to 7  loop
         parity_IN <= std_logic_vector(test_in);
         if (count=0) then     esperado_parity    := '1'; --parity_IN = 000
         elsif (count=1) then   esperado_parity := '0'; --parity_IN = 001
         elsif (count=2) then   esperado_parity := '0'; --parity_IN = 010
         elsif (count=3) then   esperado_parity := '1'; --parity_IN = 011
         elsif (count=4) then   esperado_parity := '0'; --parity_IN = 100
         elsif (count=5) then   esperado_parity := '1'; --parity_IN = 101
         elsif (count=6) then   esperado_parity := '1'; --parity_IN = 110
         else   esperado_parity := '0';   --parity_IN = 111
         end if;

         wait for 10 ns;
         test_in := test_in + 1;
         if ( esperado_parity /= parity ) then
              report "ERROR en la salida valida. Valor esperado: "  &
                      std_logic'image(esperado_parity)                &
                      ", valor actual: "                                 &
                      std_logic'image(parity)                         &
                      " en el instante: "                                 &
                      time'image(now);
              error_count := error_count + 1;
         end if;

      end loop;
      report   "ERROR: Hay "                &
                integer'image(error_count) &
                " errores.";
      wait;
   end process gen_vec_test;
end architecture bp_detector_paridad;
-------------------------------------
```

Código VHDL 1.4: Banco de pruebas del detector de paridad.

PREGUNTA 4 (2 puntos)

A continuación, se muestra el diseño de cuatro circuitos. La **entity** de todos ellos es la siguiente.

```
entity ffd is port(
    q          :  out std_logic;
    d, clk, rst:  in  std_logic );
end entity ffd;
```

La **architecture** de cada uno de los circuitos se muestra a continuación (véase Solución 1, 2, 3 y 4).

```
---- Solucion 1-------------
architecture arch1 of ffd is
begin
  process (clk, rst)
  begin
    if (rst = '1') then
        q <= '0';
    elsif (rising_edge(clk)) then
        q <= d;
    end if;
  end process;
end arch1;
```

```
---- Solucion 2-------------
architecture arch2 of ffd is
begin
  process (clk)
  begin
    if (rst = '1') then
        q <= '0';
    elsif (rising_edge(clk)) then
        q <= d;
    end if;
   end process;
end arch2;
```

```
---- Solucion 3-------------
architecture arch3 of ffd is
begin
  process (clk)
  begin
    if  (rst = '1') then
        q <= '0';
    elsif (clk = '1') then
        q <= d;
    end if;
  end process;
end arch3;
```

```
---- Solucion 4-------------
architecture arch4 of ffd is
begin
  process (clk, rst, d)
  begin
    if  (rst = '1') then
        q <= '0';
    elsif (clk = '1') then
        q <= d;
    end if;
  end process;
end arch4;
```

Para cada circuito, explique detalladamente si tiene la funcionalidad de un flip-flop D con señal de reset asíncrona.

Solución a la Pregunta 4

De los cuatro circuitos mostrados en el enunciado, el único que tiene la funcionalidad de un flip-flop D con señal reset asíncrona es el circuito Solución 1.

El circuito Solución 2 se corresponde con un flip-flop D con señal reset síncrona. Para que dicho circuito se comportase como un flip-flop D con señal reset asíncrona, la lista sensible de su bloque **process** tendría que incluir la señal `rst`.

Los diseños Solución 3 y Solución 4 no describen el comportamiento de un flip-flop D, ya que el estado de estos dos circuitos no cambia ni en el flanco de subida, ni en el flanco de bajada de la señal de reloj. En particular, el circuito Solución 4 describe la funcionalidad de un latch D con señal reset asíncrona.