INGENIERÍA DE COMPUTADORES III

Solución al Ejercicio de Autocomprobación 1

PREGUNTA 1 (3 puntos)

Escriba en VHDL la **architecture** que describe el comportamiento de un contador síncrono ascendente módulo 4 en términos de una máquina de Moore. La salida del contador toma cíclicamente los valores "00", "01", "10", "11". Si la cuenta está habilitada, se obtiene un nuevo valor de la salida en cada flanco de subida de la señal de reloj. La entrada `c` habilita la cuenta cuando está a '1' y la deshabilita cuando está a '0'. La entrada `reset` es una señal asíncrona activa a nivel alto que pone la cuenta a "00". La **entity** del contador se muestra a continuación.

```
entity contador_Mod_4 is port(
   state           : out std_logic_vector(1 downto 0);
   clock, reset, c : in std_logic);
end entity contador_Mod_4;
```

Solución a la Pregunta 1

El Código VHDL 1.2 es un posible diseño del contador módulo 4. El Código VHDL 1.1 es un **package** en el cual se ha definido la codificación de los estados.

```
-------------------------------------------------------
library IEEE;     use IEEE.std_logic_1164.all;
package STATE_CONSTANTS is
   constant S0: std_logic_vector(1 downto 0) := "00";     -- Estados
   constant S1: std_logic_vector(1 downto 0) := "01";
   constant S2: std_logic_vector(1 downto 0) := "10";
   constant S3: std_logic_vector(1 downto 0) := "11";
end package;
-------------------------------------------------------
```

Código VHDL 1.1: Constantes del diseño del contador ascendente módulo 4.

```
-------------------------------------------------------
-- Contador módulo 4 implementado como máquina de Moore
library IEEE;
use IEEE.std_logic_1164.all;
use work.STATE_CONSTANTS.all;

entity contador_Mod_4 is
   port( state           : out std_logic_vector(1 downto 0);
         clock, reset, c : in std_logic);
end entity contador_Mod_4;

architecture contador_Mod_4 of contador_Mod_4 is
   signal internal_state : std_logic_vector(1 downto 0);
begin
   state <= internal_state;

   proximo_estado: process (clock, reset)     -- Cálculo del próximo estado
   begin
      if (reset = '1') then
           internal_state <= S0;
      elsif (rising_edge(clock)) then
           case internal_state is
                when S0 =>
                     if c = '1' then
                          internal_state <= S1;
                     else
                          internal_state <= S0;
                     end if;
                when S1 =>
                     if c = '1' then
                          internal_state <= S2;
                     else
                          internal_state <= S1;
                     end if;
                when S2 =>
                     if c = '1' then
                          internal_state <= S3;
                     else
                          internal_state <= S2;
                    end if;
                 when S3 =>
                      if c = '1' then
                          internal_state <= S0;
                       else
                          internal_state <= S3;
                     end if;
                 when others=>
                      internal_state <= S0;
                end case;
           end if;
end process proximo_estado;

end architecture contador_Mod_4;
-----------------------------------------------
```

Código VHDL 1.2: Diseño de un contador ascendente módulo 4.

PREGUNTA 2 (3 puntos)

Programe en VHDL el banco de pruebas del contador que ha diseñado al resolver la Pregunta 1. 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 indicando el número total de salidas incorrectas.

Solución a la Pregunta 2

El Código VHDL 1.3 y 1.4 es un posible banco de pruebas para el circuito. En el diagrama de transición de estados del circuito, que se muestra en la Figura 1.1, se ha señalado el orden en el cual el programa de test recorre los arcos.

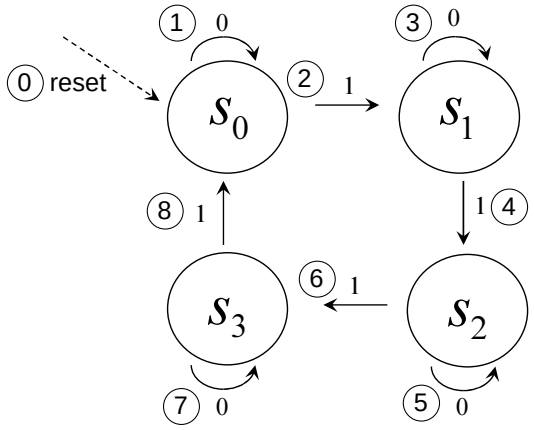

Figura 1.1: Diagrama de transición de estados de un contador ascendente módulo 4. En el diagrama se ha señalado el orden en el cual el programa de test recorre los arcos.

```
----------------------------------------
-- Banco de pruebas del contador ascendente sincrono modulo 4
library IEEE;
use IEEE.std_logic_1164.all;
use work.STATE_CONSTANTS.all;

entity bp_contador_Mod_4 is
end entity bp_contador_Mod_4;

architecture bp_contador_Mod_4 of bp_contador_Mod_4 is
   constant PERIODO    : time       := 100 ns;          -- Reloj
   signal   q          : std_logic_vector(1 downto 0);  -- Salidas UUT
   signal   clk        : std_logic  := '0';             -- Entradas UUT
   signal   reset, c   : std_logic;


   component contador_Mod_4 is
      port (   state           : out std_logic_vector(1 downto 0);
               clock, reset, c : in std_logic );
   end component contador_Mod_4;

    -- Procedimiento para comprobar las salidas
    procedure comprueba_salidas
       ( esperado_q             :          std_logic_vector(1 downto 0);
         actual_q               :          std_logic_vector(1 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(1))    &
                   std_logic'image(esperado_q(0))    &
                   "), estado actual ("              &
                   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 contador_Mod_4 port map     (q, clk, reset, c);

   reset <=    '0','1' after (PERIODO/4),
               '0' after (PERIODO+PERIODO/4);

   clk <= not clk after (PERIODO/2);
-----------------------------------------------
```

Código VHDL 1.3: Parte inicial del banco de pruebas del contador ascendente módulo 4.

```
-------------------------------------
-- Continuación del banco de pruebas del contador ascendente síncrono modulo 4

   gen_vec_test : process is
      variable error_count : integer := 0; -- Núm. errores
   begin
      report "Comienza la simulación";

      -- Vectores de test y comprobación del resultado
      C <= '0';    wait for PERIODO;    -- 1
      comprueba_salidas(S0, q, error_count);
      C <= '1';    wait for PERIODO;    -- 2
      comprueba_salidas(S1, q, error_count);
      C <= '0';    wait for PERIODO;    -- 3
      comprueba_salidas(S1,q, error_count);
      C <= '1';    wait for PERIODO;    -- 4
      comprueba_salidas(S2, q, error_count);
      C <= '0';    wait for PERIODO;    -- 5
      comprueba_salidas(S2 ,q, error_count);
      C <= '1';    wait for PERIODO;    -- 6
      comprueba_salidas(S3, q, error_count);
      C <= '0';    wait for PERIODO;    -- 7
      comprueba_salidas(S3, q, error_count);
      C <= '1';    wait for PERIODO;    -- 8
      comprueba_salidas(S0, q, error_count);

      -- 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_contador_Mod_4;
-------------------------------------
```

Código VHDL 1.4: Parte final del banco de pruebas del contador ascendente módulo 4.

PREGUNTA 3 (2 puntos)

Escriba en VHDL la **architecture** que describa el comportamiento de un circuito combinacional que realiza operaciones de desplazamiento sobre una entrada de 4 bits. La entrada de selección de operación del circuito es `op`. Las señales de entrada y salida se llaman, respectivamente, `entrada` y `salida`. La tabla de operaciones y la **entity** del circuito son:

op	Operación
0 0	Desplaza 1 bit a la izquierda rellenando con '0'
0 1	Desplaza 1 bit a la derecha rellenando con '1'
1 0	Rota 1 bit a la izquierda
1 1	Rota 1 bit a la derecha

```
entity desplazador is
    port( salida  : out std_logic_vector(3 downto 0);
          op      : in std_logic_vector(1 downto 0);
          entrada : in std_logic_vector(3 downto 0));
end entity desplazador;
```

Solución a la Pregunta 3

El Código VHDL 1.5 es un posible diseño del circuito.

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

entity desplazador is
  port(  salida  : out std_logic_vector(3 downto 0);
         op      : in std_logic_vector(1 downto 0);
         entrada : in std_logic_vector(3 downto 0));
end desplazador;

architecture desplazador of desplazador is
begin

    salida <=   entrada(2 downto 0) & '0'            when (op = "00")
                else '1' & entrada(3 downto 1)       when (op = "01")
                else entrada(2 downto 0) & entrada(3) when (op = "10")
                else entrada(0) & entrada(3 downto 1);

end architecture desplazador;
----------------------------------
```

Código VHDL 1.5: Diseño del circuito desplazador.

PREGUNTA 4 (2 puntos)

Escriba en VHDL la **architecture** del flip-flop D cuyo comportamiento se describe a continuación, empleando para ello una sentencia **if**. Además de la entrada de reloj (`Clock`), el flip-flop tiene otras cuatro señales de entrada: `D`, `Enable`, `Set` y `Clear`. El flip-flop tiene una única señal de salida, cuyo valor coincide en todo momento con el valor del estado del circuito (`Q`).

```
entity FFD is
    port( Q                                : out std_logic;
          Clock, Enable, Set, Clear, D : in std_logic);
end entity FFD;
```

Cuando la entrada asíncrona `Set` es puesta al valor '1', el estado del flip-flop cambia inmediatamente al valor '1'. Por el contrario, cuando la entrada asíncrona `Clear` es puesta al valor '1', el estado del flip-flop cambia inmediatamente al valor '0'. La entrada `Set` ha de tener prioridad sobre la entrada `Clear`. Si la señal `Enable` vale '1', entonces en el flanco de subida de la señal de reloj se asigna el valor de la entrada `D` al estado del flip-flop (`Q`). Por el contrario, si la señal `Enable` vale '0', la carga del flip-flop desde la entrada `D` está deshabilitada.

Solución a la Pregunta 4

```
---------------------------------------
library IEEE;  use IEEE.std_logic_1164.all;
entity FFD is
     port(  Q                              : out std_logic;
            Clock, Enable, Set, Clear, D : in std_logic);
end entity FFD;

architecture FFD of FFD is
begin
    process(Clock, Set, Clear)
    begin
      if Set = '1' then
         Q <= '1';
      elsif Clear = '1' then
         Q<='0';
      elsif (rising_edge(Clock)) then
         if (Enable ='1') then
            Q <= D;
         end if;
      end if;
    end process;
end architecture FFD;
---------------------------------------
```

Código VHDL 1.6: Diseño de un flip-flop D con señales enable, set y clear.