INGENIERÍA DE COMPUTADORES 3

Solución al examen de Junio 2018, Primera Semana

PREGUNTA 1 (2 puntos)

Tomando como base el siguiente código VHDL, dibuje el cronograma de evolución de las señales `x1`, `x2`, `x3`, `x4`, `x5` y `x6` entre los instantes 0 y 100 ns.

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

entity crono is
end entity crono;

architecture crono of crono is
  signal x1, x2, x3, x4, x5, x6 : std_logic;
begin
  x1 <= '0', '1' after 10 ns,
        '0' after 15 ns,'1' after 25 ns,
        '0' after 30 ns;
  x2 <= transport x1 after 8 ns;
  Proc1: process(x1,x2)
  begin
     x3 <= x1 xor x2;
  end process;
   x4 <= x1 after 8 ns;
   x5 <= x1 xor x2;
   x6 <= x5;
end architecture crono;
```

Solución a la Pregunta 1

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

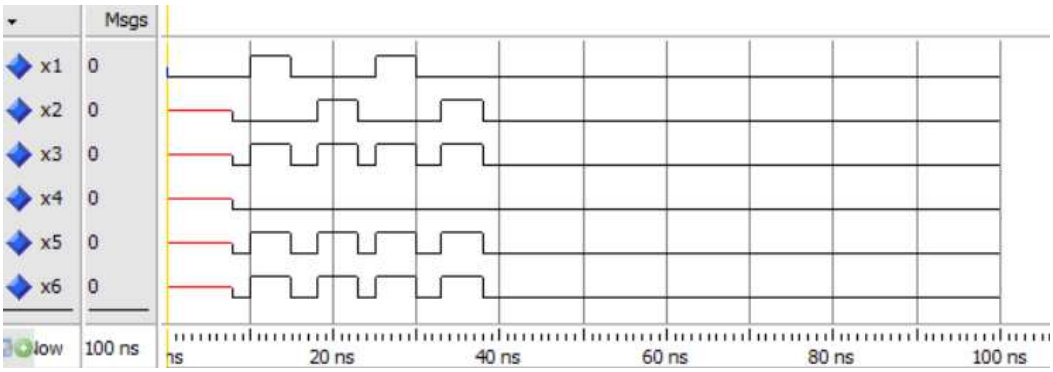


Figura 1.1: Cronograma de evolución de las señales.

PREGUNTA 2 (3 puntos)

Diseñe usando VHDL el circuito mostrado en la siguiente figura, que está compuesto por una unidad aritmético lógica (ALU) y un registro de desplazamiento. La ALU realiza operaciones sobre dos operandos de 8 bits, denominados `A` y `B`, y un operando de un bit denominado `cin`. La salida de la ALU es la entrada al registro de desplazamiento. El registro de desplazamiento opera en el flanco de subida de la señal de reloj `CLK`. La señal `S`, de 5 bits, determina la operación realizada por el circuito tal como se indica en las dos tablas de operaciones mostradas a continuación.

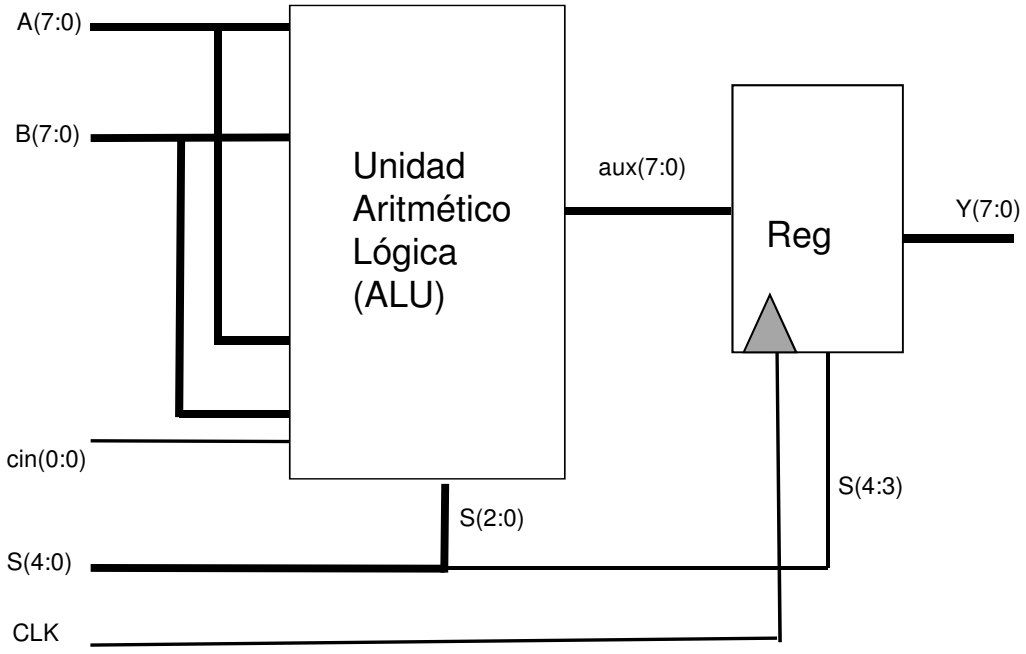


Tabla 1: Operaciones del registro.

S(4)S(3)	**Operación**
00	Carga de la señal `aux` desplazada un bit a la derecha, introduciendo un '0' en el bit más significativo
01	Carga de la señal `aux` desplazada un bit a la izquierda, introduciendo un '0' en el bit menos significativo
10	Carga de la señal `aux` rotada un bit a la derecha
11	Carga de la señal `aux` sin modificarla

Tabla 2: Operaciones de la ALU.

S(2)S(1)S(0)	**Operación**
000	`A + B`
001	`A + B + cin`
010	`A + 1`
011	`B - 1`
100	`not B`
101	`A or B`
110	`A and B`
111	`A xor B`

Escriba en VHDL la **architecture** que describe el comportamiento del circuito empleando un bloque **process** que describa el comportamiento de la ALU y otro bloque **process** que describa el comportamiento del registro. Asimismo, en el diseño únicamente pueden emplearse los dos siguientes paquetes de la librería IEEE:

```
IEEE.std_logic_1164
IEEE.numeric_std
```

El circuito ha de tener la **entity** siguiente:

```
entity ALUReg is
   port( Y    : out std_logic_vector ( 7 downto 0);
         A, B : in std_logic_vector (7 downto 0);
         cin  : in std_logic_vector (0 downto 0);
         CLK : in std_logic;
         S  : in std_logic_vector ( 4 downto 0));
end entity ALUReg;
```

Solución a la Pregunta 2

El código VHDL del circuito combinacional descrito en la Pregunta 2 se muestra en Código 1.1–1.2.

```
----------------------------------------------
-- ALU con registro
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
architecture ALUReg of ALUReg is
signal aux : std_logic_vector(7 downto 0);
begin
--ALU
 alu: process (s, A, B, cin)
 begin
    case s (2 downto 0) is
       when "000" =>
          aux <= std_logic_vector(signed(A) + signed(B));
       when "001" =>
          aux <= std_logic_vector (signed(A) + signed(B) + signed("0" & cin));
       when "010" =>
          aux <= std_logic_vector (signed(A) + 1);
       when "011" =>
          aux <= std_logic_vector (signed(B) - 1);
      when "100" =>
          aux <= not B;
      when "101" =>
          aux <= A or B;
      when "110" =>
          aux <= A and B;
      when others =>
          aux <= A xor B;
    end case;
  end process alu;
```

Código VHDL 1.1: Architecture de la ALU y el registro de desplazamiento (1/2).

```
  --Registro
    reg:process(clk)
    begin
      if rising_edge(clk) then
        case s (4 downto 3) is
          when "00" =>
            Y <= '0' & aux(7 downto 1);
          when "01" =>
            Y <= aux(6 downto 0) & '0';
          when "10" =>
            Y <= aux(0) & aux(7 downto 1);
          when others =>
            Y <= aux;
        end case;
      end if;
    end process reg;
end architecture ALUReg;
-----------------------------------------------
```

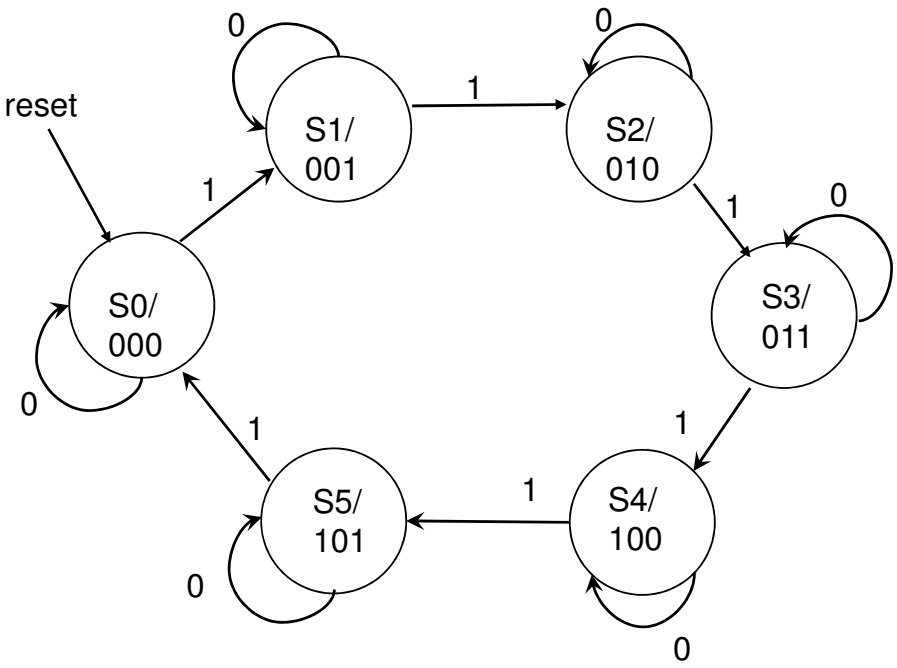
Código VHDL 1.2: Architecture de la ALU y el registro de desplazamiento (2/2).

PREGUNTA 3 (3 puntos)

Realice el diseño usando VHDL de un contador síncrono ascendente módulo 6. Es decir, la salida del circuito (señal `y`) toma cíclicamente los valores “000”, “001”, “010”, “011”, “100” y “101”. Si la cuenta está habilitada, se obtiene un nuevo valor de la salida en cada flanco de subida de la señal de reloj. El contador debe tener una entrada `c` activa a nivel alto que permita habilitar y deshabilitar la cuenta, la señal de reloj de entrada `clock` y una entrada `reset` síncrona activa a nivel alto, que pone la cuenta a ’0’. Describa el comportamiento del circuito en términos de una máquina de Moore. Dibuje el diagrama de estados que describe el comportamiento del circuito.

Solución a la Pregunta 3

El diagrama de estados correspondiente al circuito contador se muestra en la siguiente figura.



El código VHDL correspondiente al contador se muestra en Código VHDL 1.3–1.4.

```
-----------------------------------------------------------
---Contador modulo 6 implementado como máquina de Moore
-----------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
entity contador_Mod_6 is port(
  y: out std_logic_vector(2 downto 0);
  clock, reset, c: in std_logic);
end entity contador_Mod_6;

architecture contador_Mod_6 of contador_Mod_6 is
        signal internal_state: std_logic_vector(2 downto 0);
begin
  y <= internal_state;
```

Código VHDL 1.3: Contador módulo 6 (1/2).

```
--Cálculo del próximo estado
proximo_estado: process (clock)
begin
     if (rising_edge(clock)) then
       if reset= '1' then
          internal_state <= "000";
       else
       case internal_state is
              when "000" =>
                   if c = '1' then
                        internal_state <= "001";
                   end if;
              when "001" =>
                   if c = '1' then
                        internal_state <= "010";
                   end if;
              when "010" =>
                   if c = '1' then
                         internal_state <= "011";
                   end if;
               when "011" =>
                    if c = '1' then
                         internal_state <= "100";
                    end if;
               when "100" =>
                    if c = '1' then
                         internal_state <= "101";
                    end if;
               when "101" =>
                    if c = '1' then
                          internal_state <= "000";
                     end if;
               when others=>
                    internal_state <= "000";
              end case;
              end if;
          end if;
end process proximo_estado;

end architecture contador_Mod_6;
-----------------------------------------
```

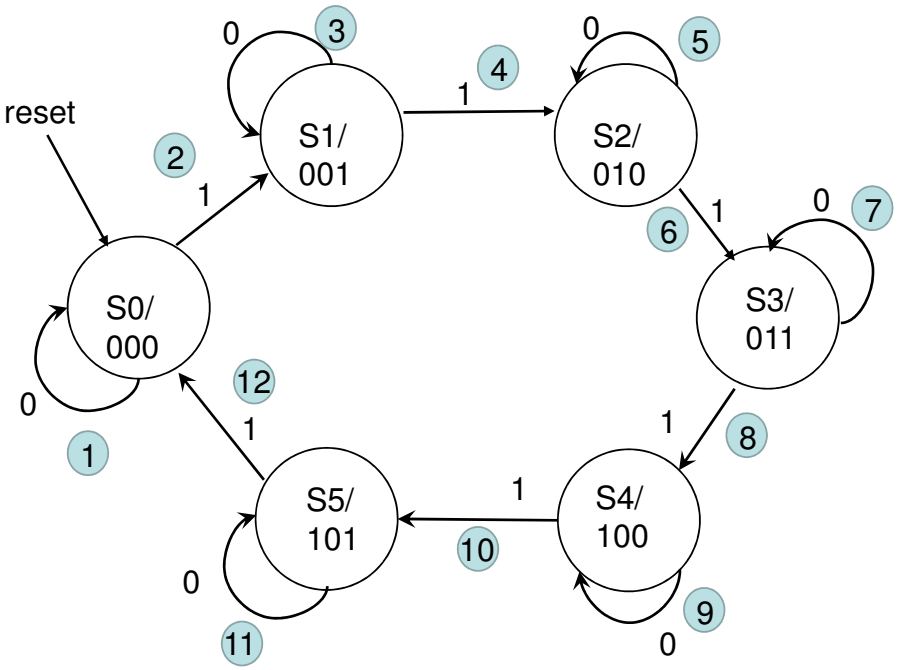
Código VHDL 1.4: Contador módulo 6 (2/2).

PREGUNTA 4 (2 puntos)

Programe el banco de pruebas del circuito secuencial que ha diseñado en la Pregunta 3. Explique detalladamente cómo el programa de test comprueba de manera sistemática el funcionamiento del circuito. 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 4

En la siguiente figura se muestra el orden en que el banco de pruebas va testeando cada una de las transiciones entre los estados.



El código VHDL del banco de pruebas se muestra en Código VHDL 1.5–1.6.

```
  ----------------------------------
-- Banco de pruebas del contador
--ascendente sincrono modulo 6
library IEEE;
use IEEE.std_logic_1164.all;

entity bp_contador_Mod_6 is
end entity bp_contador_Mod_6;

architecture bp_contador_Mod_6 of bp_contador_Mod_6 is
   constant PERIODO    : time         := 100 ns; -- Reloj
   signal     q           : std_logic_vector(2 downto 0);-- Salidas UUT
   signal     clk         : std_logic   := '0';      -- Entradas UUT
   signal     reset, c   : std_logic;

   component contador_Mod_6 is
       port ( y              : out std_logic_vector(2 downto 0);
              clock, reset, c   : in std_logic );
   end component contador_Mod_6;
    -- Procedimiento para comprobar las salidas
    procedure comprueba_salidas
        ( esperado_q            :         std_logic_vector(2 downto 0);
          actual_q               :         std_logic_vector(2 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(2))  &
                       std_logic'image(esperado_q(1))    &
                       std_logic'image(esperado_q(0))    &
                       "), estado actual ("            &
                       std_logic'image(actual_q(2)) &
                       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;
```

Código VHDL 1.5: Diseño del banco de pruebas del contador módulo 6 (1/2).

```
begin
  -- Instanciar y conectar UUT
  uut : component contador_Mod_6 port map
                    (q,  clk, reset, c);
  reset <=   '0','1' after (PERIODO/4),
             '0' after (PERIODO+PERIODO/4);
  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
      C <= '0';   wait for PERIODO;    -- 1
      comprueba_salidas("000",q,error_count);
      C <= '1';   wait for PERIODO;    -- 2
      comprueba_salidas("001",q,error_count);
      C <= '0';   wait for PERIODO;    -- 3
      comprueba_salidas("001",q,error_count);
      C <= '1';   wait for PERIODO;    -- 4
      comprueba_salidas("010",q,error_count);
      C <= '0';   wait for PERIODO;    -- 5
      comprueba_salidas("010" ,q,error_count);
      C <= '1';   wait for PERIODO;    -- 6
      comprueba_salidas("011",q,error_count);
      C <= '0';   wait for PERIODO;    -- 7
      comprueba_salidas("011",q,error_count);
      C <= '1';   wait for PERIODO;    -- 8
      comprueba_salidas("100",q,error_count);
      C <= '0';   wait for PERIODO;    -- 9
      comprueba_salidas("100" ,q,error_count);
      C <= '1';   wait for PERIODO;    -- 10
      comprueba_salidas("101",q,error_count);
      C <= '0';   wait for PERIODO;    --11
      comprueba_salidas("101",q,error_count);
      C <= '1';   wait for PERIODO;    -- 12
      comprueba_salidas("000",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_6;
-------------------------------------
```

Código VHDL 1.6: Continuación del banco de pruebas del contador módulo 6 (2/2).