Estructura de los Computadores

Práctica 2 – Circuito VHDL

Alejandro Bernabeu Calatayud

abc4.onil4@gmail.com

Grupo 40 I2ADE

48788949-S

08/03/18

Índice

Archivos

Implementación:

Comparador_4Bits: Un comparador básico de 4 bits.

Comparador_8Bits: Un comparador de 8 bits que aúna en su interior 2 comparadores de 4 bits.

Contador_4Bits: Un Contador que puede ser tanto ascendente como descendente y con la posibilidad de cargar un dato. También se puede restablecer a 0.

Test:

Test_Comparador_4Bits: El test para comprobar el correcto funcionamiento del Comparador_4Bits.

Test_Comparador_8Bits: El test para comprobar el correcto funcionamiento del Comparador_8Bits.

Test_Contador_4Bits: El test para comprobar el correcto funcionamiento del Contador_4Bits.

2.1 Circuito Comparador

Para implementar el comparador de 8 bits he usado dos comparadores de 4 bits cada uno.

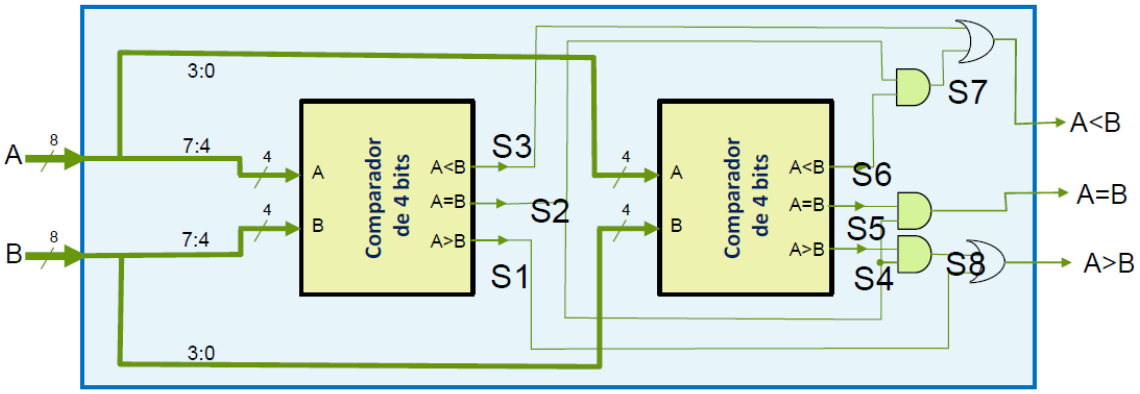
El Comparador de 4 Bits:

```
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  -- Uncomment the following library declaration if using
24  -- arithmetic functions with Signed or Unsigned values
25  --use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  --library UNISIM;
30  --use UNISIM.VComponents.all;
31
32  entity Comparador_4Bits is
33      Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
34             B : in  STD_LOGIC_VECTOR (3 downto 0);
35             Mayor : out  STD_LOGIC;
36             Menor : out  STD_LOGIC;
37             Igual : out  STD_LOGIC);
38  end Comparador_4Bits;
39
40  architecture Behavioral of Comparador_4Bits is
41
42  begin
43
44  Mayor <= '1' when (A>B) else '0';
45  Menor <= '1' when (A<B) else '0';
46  Igual <= '1' when (A=B) else '0';
47
48  end Behavioral;
49
```

Esta compuesto por dos entradas y tres salidas. Se activará solamente una salida, y esta nos indicará si el número A es mayor, menor o igual a B.

El primer comparador analiza los 4 primero dígitos, puesto que son decisivos. El segundo comparador solo actúa eficazmente si el resultado del primero comparador es Igual.

El comparador de 8 Bits:




```
Mayor2 <= S8 OR S1;
Menor2 <= S7 OR S3;
```

Al pasar por el primer comparador se activa la salida S1 o S3, a continuación tenemos una puerta OR, es decir que siempre pasará, independientemente del valor de S7 o S8.

A continuación, podemos apreciar que, si el resultado de los 4 primeros bits es Igual, el resultado sí dependerá de segundo comparador, que realiza la comparación entre los últimos 4 bits.

```
S7 <= S6 AND S2;
Igual2 <= S5 AND  S2;
S8 <= S4 AND S2;
```

Y por eso colocamos puertas AND, porque ahora sí depende de los dos comparadores y para poder decidir si es Mayor, Menor o Igual deben estar activas las dos entradas del AND.

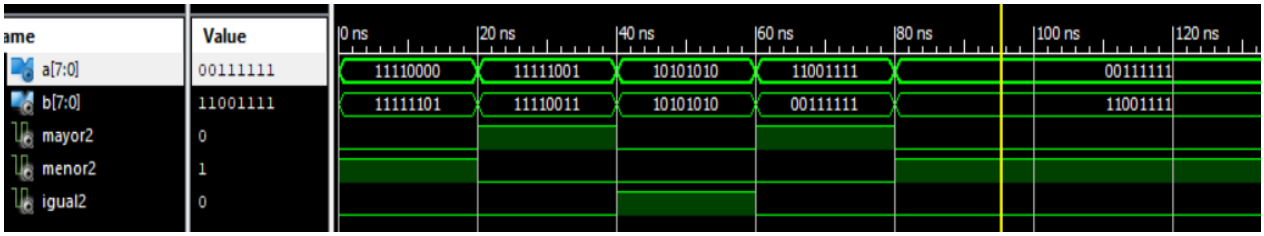
A continuación, podemos apreciar la estructura entera del comparador de 8 Bits:

```
32  entity Comparador_8Bits is
33      Port ( A : in  STD_LOGIC_VECTOR (7 downto 0);
34             B : in  STD_LOGIC_VECTOR (7 downto 0);
35             Mayor2 : out  STD_LOGIC;
36             Menor2 : out  STD_LOGIC;
37             Igual2 : out  STD_LOGIC);
38  end Comparador_8Bits;
39
40  architecture Behavioral of Comparador_8Bits is
41
42  component Comparador_4Bits is
43      Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
44             B : in  STD_LOGIC_VECTOR (3 downto 0);
45             Mayor : out  STD_LOGIC;
46             Menor : out  STD_LOGIC;
47             Igual : out  STD_LOGIC);
48  end component;
49
50  SIGNAL S1,S2,S3,S4,S5,S6,S7,S8 : STD_LOGIC;
51
52  begin
53
54  Comp1: Comparador_4Bits PORT MAP(A=> A(7 downto 4), B=> B(7 downto 4), Mayor => S1, Igual =>S2, Menor =>S3);
55  Comp2: Comparador_4Bits PORT MAP(A=> A(3 downto 0), B=> B(3 downto 0), Mayor => S4, Igual =>S5, Menor =>S6);
56
57  S7 <= S6 AND S2;
58  Igual2 <= S5 AND  S2;
59  S8 <= S4 AND S2;
60
61  Mayor2 <= S8 OR S1;
62  Menor2 <= S7 OR S3;
63
64  end Behavioral;
```

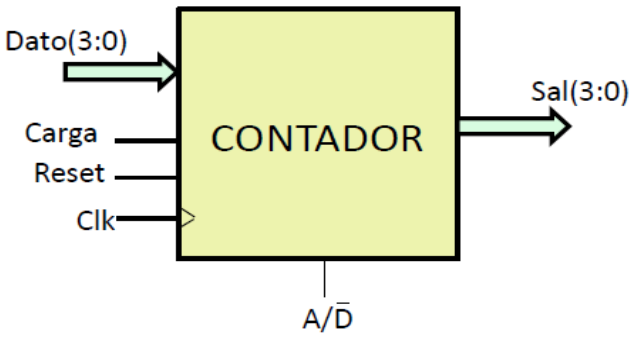
Aquí podemos apreciar como indicamos al primer comparador que coja los 4 primeros bits, y al segundo los 4 últimos de cada número.

También se aprecian las 8 señales que hemos creado ya que a la salida del comparador de 8 Bits no le podemos asignar más de un valor.

En el test podemos apreciar que funciona correctamente para valores con todas las posibilidades.



2.2 Circuito Contador



El circuito Contador de 4 bits tiene 5 entradas y 1 salida solamente.

La Salida y la entrada Dato son un bus de 4 bits mientras que el resto son simples.

También hemos creado una señal llamada señal temporal (cnt_tmp) a la que aplicaremos todos los cambios y finalmente asignaremos a la salida. Su uso es como el de una variable en programación.

```
entity Contador_4Bits is
    Port ( Dato : in  STD_LOGIC_VECTOR (3 downto 0);
           Carga : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           Clk : in  STD_LOGIC;
           AD : in  STD_LOGIC;
           Sal : out  STD_LOGIC_VECTOR (3 downto 0));
end Contador_4Bits;

architecture Behavioral of Contador_4Bits is

   signal cnt_tmp: STD_LOGIC_VECTOR(3 DOWNTO 0) := "0000";

begin
```

El funcionamiento del Contador:

- Lo primero de todo es la entrada Reset, si ésta esta activada, lo demás no importa. Colocaremos el valor a “0000”.
- La entrada de la Carga será síncrona por flanco de subida del reloj, es decir, se cargará el dato a cada ciclo del reloj si la Carga está activada (que esté activada depende de nosotros).

- En el caso de que NO esté activada, el contador realizará su funcionamiento habitual, Sumar o Restar 1 a cada flanco de reloj dependiendo de la entrada AD.
- Para el correcto funcionamiento hemos añadido que si llega al máximo o mínimo vuelva a empezar.

```
46  begin
47
48    proceso_contador: process (Carga, Reset, Clk, Dato) begin
49          if Reset = '1' then
50              cnt_tmp <= "0000";
51          elsif rising_edge(Clk) then
52              if Carga = '1' then
53              cnt_tmp <= Dato;
54              else
55                  if AD='1' then
56                      if cnt_tmp = "1111" then
57                      cnt_tmp <= "0000";
58                      else
59                      cnt_tmp <= cnt_tmp + 1;
60                      end if;
61                  else
62                      if cnt_tmp = "0000" then
63                      cnt_tmp <= "1111";
64                      else
65                      cnt_tmp <= cnt_tmp-1;
66                      end if;
67                  end if;
68              end if;
69          end if;
70      end process;
71
72      Sal <= cnt_tmp;
73
74
75  end Behavioral;
76
```

Para ir acabando asignamos valor al periodo del reloj y a las entradas que dependen de nosotros.

```
-- Clock period definitions
constant Clk_period : time := 20 ns;
```

```
AD <= '1', '0' AFTER 200 ns;
Reset <= '0', '1' AFTER 100 ns, '0' AFTER 120 ns;
Dato <= "0111";
Carga <= '0', '1' AFTER 140 ns, '0' AFTER 180 ns;
```

Y finalmente comprobamos en el Test: