

CIRCUITOS VHDL

INTEGRANTES:
JIMENEZ TECALERO IVAN
XICOHTENCATL PEDRAZA RICARDO
FRANCISCO CASTILLO ABRAHAM



DESCRIPCION TEORICA

Un multiplexor 4x1 es un circuito combinacional que tiene cuatro entradas de datos (D0, D1, D2 y D3), dos entradas de selección (S0 y S1) y una salida (Y). La función principal del multiplexor es seleccionar uno de los cuatro datos de entrada y enviarlo a la salida, según los valores de las entradas de selección.

La entrada de selección S0 y S1 se utiliza para determinar cuál de las cuatro entradas de datos se va a transmitir a la salida. Estas entradas de selección actúan como una dirección binaria que selecciona una de las cuatro líneas de entrada de datos.

La tabla de verdad de un multiplexor 4x1 se muestra a continuación:

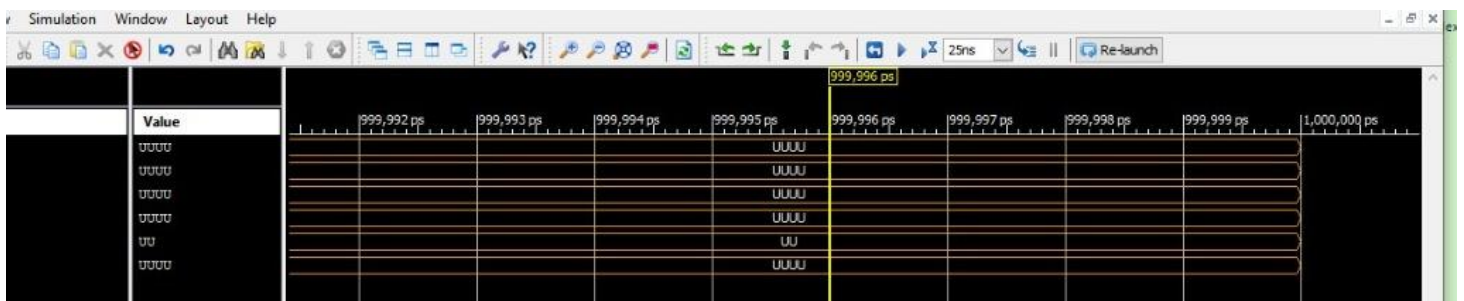
S1	S0	D3	D2	D1	D0	Y
0	0	D0	D1	D2	D3	D0
0	1	D0	D1	D2	D3	D1
1	0	D0	D1	D2	D3	D2
1	1	D0	D1	D2	D3	D3

La salida Y del multiplexor se conecta a la línea de datos correspondiente a la combinación de las entradas de selección. Por ejemplo, si S1 = 0 y S0 = 1, la salida Y será igual a D1. Si S1 = 1 y S0 = 1, la salida Y será igual a D3.

❖ MULTIPLEXOR4X1

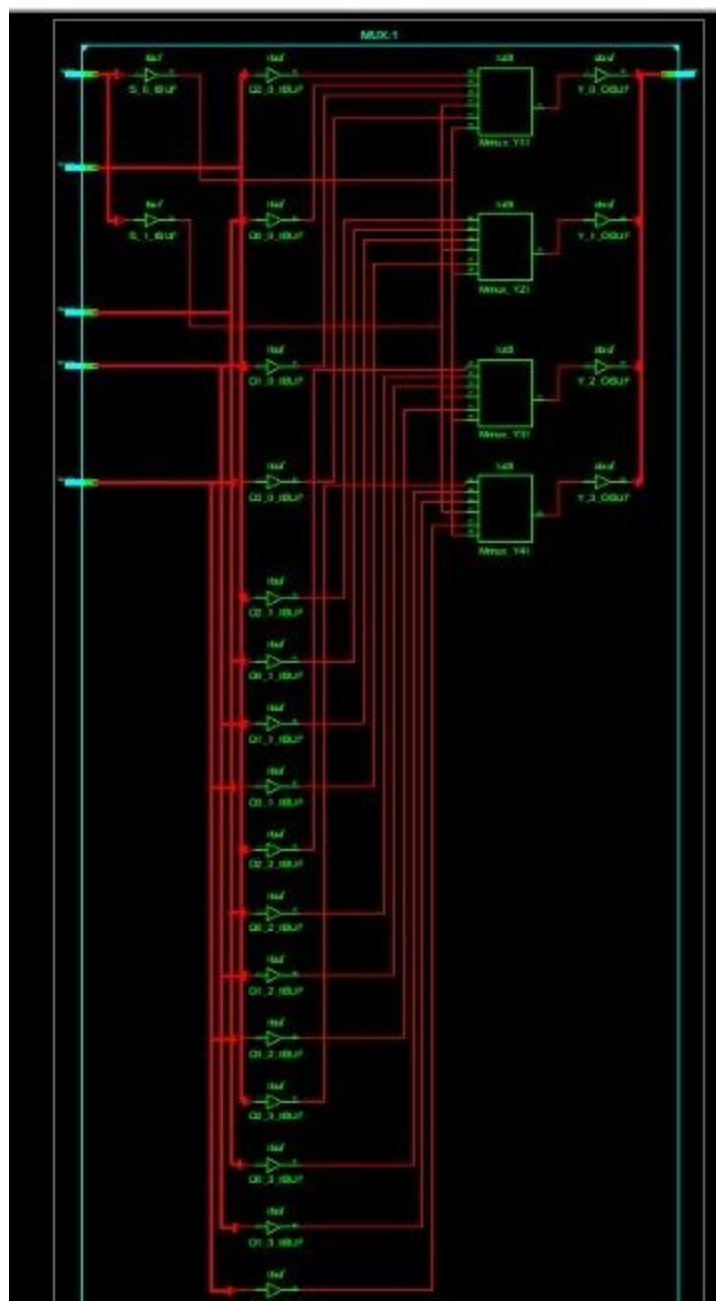
- SOLUCION PDF

```
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 -- Uncomment the following library declaration if using
26 -- arithmetic functions with Signed or Unsigned values
27 --use IEEE.NUMERIC_STD.ALL;
28
29 -- Uncomment the following library declaration if instantiating
30 -- any Xilinx primitives in this code.
31 --library UNISIM;
32 --use UNISIM.VComponents.all;
33
34 entity MUX is
35     port (
36         D0, D1, D2, D3 : in std_logic_vector(3 downto 0);
37         S : in std_logic_vector(1 downto 0);
38         Y : out std_logic_vector(3 downto 0));
39 end MUX;
40
41 architecture Behavioral of MUX is
42
43 begin
44
45 process (D0, D1, D2, D3, S)
46     begin
47         case S is
48             when "00" =>
49                 Y <= D0;
50             when "01" =>
51                 Y <= D1;
52             when "10" =>
53                 Y <= D2;
54             when others =>
55                 Y <= D3;
56         end case;
57     end process;
58
59 end Behavioral;
```



- SOLUCION VIDEO

```
3 library IEEE;
4 use IEEE.std_logic_unsigned.all;
5
6
7 -- Uncomment the following library declaration if using
8 -- arithmetic functions with Signed or Unsigned values
9 --use IEEE.NUMERIC_STD.ALL;
10
11 -- Uncomment the following library declaration if instantiating
12 -- any Xilinx primitives in this code.
13 --library UNISIM;
14 --use UNISIM.VComponents.all;
15
16 entity Multiplexor4x1 is
17     port (
18         a : in STD_LOGIC;
19         b : in STD_LOGIC;
20         i0 : in STD_LOGIC;
21         i1 : in STD_LOGIC;
22         i2 : in STD_LOGIC;
23         i3 : in STD_LOGIC;
24         y : out STD_LOGIC);
25 end Multiplexor4x1;
26
27 architecture Behavioral of Multiplexor4x1 is
28 begin
29
30     y <= (not a and not b and i0) or
31         (not a and b and i1) or
32         (a and not b and i2) or
33         (a and b and i3);
34 end architecture Behavioral;
35
36 end Behavioral;
```



DESCRIPCION TEORICA

Un sumador de 4 bits es un circuito combinacional que se utiliza para sumar dos números binarios de 4 bits. Toma dos entradas de 4 bits (A y B) y produce una salida de 4 bits (Suma) y un acarreo de salida (Carry_out). La suma se calcula bit a bit, teniendo en cuenta cualquier acarreo de entrada (Carry_in).

La operación de suma en un sumador de 4 bits se realiza de la siguiente manera:

El bit menos significativo de A (A0) se suma con el bit menos significativo de B (B0) y el acarreo de entrada (Carry_in) para producir el bit menos significativo de la suma (Suma0) y el acarreo de salida (Carry_out0).

El bit siguiente de A (A1) se suma con el bit siguiente de B (B1) y el acarreo de salida de la etapa anterior (Carry_out0) para producir el segundo bit de la suma (Suma1) y el acarreo de salida (Carry_out1).

El proceso se repite para los bits restantes, sumando A2 con B2 y Carry_out1 para obtener Suma2 y Carry_out2, y sumando A3 con B3 y Carry_out2 para obtener Suma3 y Carry_out3.

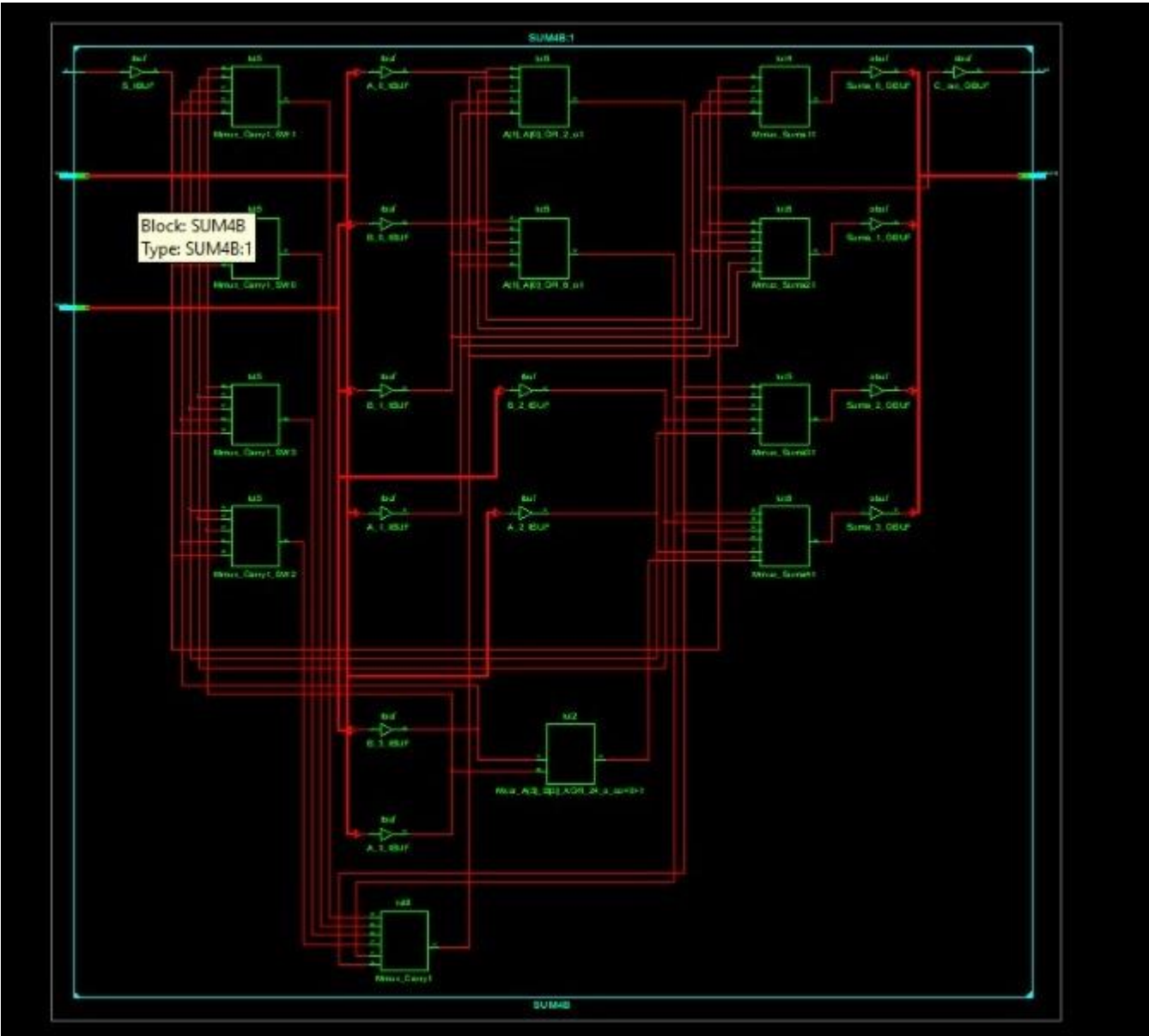
Finalmente, la salida de la suma (Suma) se compone de los bits Suma3, Suma2, Suma1 y Suma0, y el acarreo de salida (Carry_out) es Carry_out3.

❖ SUMADOR 4 BITS

```
20
21 library IEEE;
22 use IEEE.STD_LOGIC_1164.ALL;
23
24
25
26 -- Uncomment the following library declaration if using
27 -- arithmetic functions with Signed or Unsigned values
28 --use IEEE.NUMERIC_STD.ALL;
29
30 -- Uncomment the following library declaration if instantiating
31 -- any Xilinx primitives in this code.
32 --library UNISIM;
33 --use UNISIM.VComponents.all;
34
35 entity SUMA4BITS is
36     port (
37         A : in STD_LOGIC_VECTOR(3 downto 0);
38         B : in STD_LOGIC_VECTOR(3 downto 0);
39         Cin : in STD_LOGIC;
40         Sum : out STD_LOGIC_VECTOR(3 downto 0);
41         Cout : out STD_LOGIC);
42 end SUMA4BITS;
43
44 architecture Behavioral of SUMA4BITS is
45
46 begin
47
48 process (A, B, Cin)
49     variable temp : STD_LOGIC_VECTOR(3 downto 0);
50 begin
51     temp := ('0' & A) + ('0' & B) + Cin;
52     Sum <= temp(3 downto 0);
53     Cout <= temp(4);
54
55 end process;
56
57 end Behavioral;
58
```

• SOLUCION DEL PDF

```
0 library IEEE;
1 use IEEE.STD_LOGIC_1164.ALL;
2
3 -- Uncomment the following library declaration if using
4 -- arithmetic functions with Signed or Unsigned values
5 --use IEEE.NUMERIC_STD.ALL;
6
7 -- Uncomment the following library declaration if instantiating
8 -- any Xilinx primitives in this code.
9 --library UNISIM;
10 --use UNISIM.VComponents.all;
11
12 entity SUM4B is
13     port (
14         A, B : in std_logic_vector(3 downto 0);
15         S : in std_logic;
16         C_out : out std_logic;
17         Suma : out std_logic_vector(3 downto 0));
18 end SUM4B;
19
20 architecture Behavioral of SUM4B is
21
22 begin
23
24 process (A, B, S)
25     variable Carry : std_logic := '0';
26     begin
27         if (S = '0') then -- Suma
28             for i in 0 to 3 loop
29                 Suma(i) <= A(i) xor B(i) xor Carry;
30                 Carry := (A(i) and B(i)) or (Carry and (A(i) xor B(i)));
31             end loop;
32         else -- Resta
33             for i in 0 to 3 loop
34                 Suma(i) <= A(i) xor B(i) xor not Carry;
35                 Carry := (A(i) and not B(i)) or (Carry and (A(i) xor B(i)));
36             end loop;
37         end if;
38
39         C_out <= Carry;
40     end process;
41
42 end Behavioral;
```



DESCRIPCION TEORICA

Un codificador de 7 segmentos es un circuito combinacional utilizado para convertir un número binario de 4 bits en una señal de salida que pueda ser utilizada para mostrar el número en un display de 7 segmentos. Cada segmento del display se representa con una letra (a, b, c, d, e, f, g) y corresponde a un bit de entrada.

Un codificador de 7 segmentos típico utiliza una tabla de verdad para determinar qué combinación de segmentos debe encenderse para representar cada número decimal (0-9) en el display de 7 segmentos.

La tabla de verdad para un codificador de 7 segmentos se muestra a continuación:

ENTRADA(BINARIO)	SALIDA(SEGMENTOS)
0000	a' b' c' d' e' f' g'
0001	b' c'
0010	a' b c d g'
0011	a' b' c d g'
0100	a b' c' d' g'
0101	a b' c d g'
0110	a b c d e g'
0111	a' b'c'
1000	a b c d e f g
1001	a b c d f g'

El codificador de 7 segmentos toma los 4 bits de entrada y produce una combinación de señales de salida correspondientes a los segmentos del display (a-g). Cada bit de entrada está asociado a un segmento específico, y según los valores de los bits de entrada, se encienden los segmentos correspondientes para formar el número deseado en el display.

❖ CODIFICADOR 7 SEGMENTOS

- SOLUCION DEL PDF

```
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 CODI7SEG is
33     port (
34         D, C, B, A : in std_logic;
35         a, b, c, d, e, f, g : out std_logic);
36 end CODI7SEG;
37
38 architecture Behavioral of CODI7SEG is
39
40     begin
41
42     process (D, C, B, A)
43         begin
44             case (D, C, B, A) is
45                 when "0000" =>
46                     a <= '1';
47                     b <= '1';
48                     c <= '1';
49                     d <= '1';
50                     e <= '1';
51                     f <= '1';
52                     g <= '0';
53                 when "0001" =>
54                     a <= '0';
55                     b <= '1';
56                     c <= '1';
57                     d <= '0';
58                     e <= '0';
59                     f <= '0';
60                     g <= '0';
61                 when "0010" =>
62                     a <= '1';
63                     b <= '1';
64                     c <= '0';
65                     d <= '1';
66                     e <= '1';
67                     f <= '0';
68                     g <= '1';
69                 -- Resto de los casos para las combinaciones del 3 al
```

```

70      -- ...
71      when others =>
72          a <= '0';
73          b <= '0';
74          c <= '0';
75          d <= '0';
76          e <= '0';
77          f <= '0';
78          g <= '1';
79      end case;
80  end process;
81
82 end Behavioral;
83
84

```

• SOLUCION VIDEO

```

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 -- Uncomment the following library declaration if using
5 -- arithmetic functions with Signed or Unsigned values
6 --use IEEE.NUMERIC_STD.ALL;
7
8 -- Uncomment the following library declaration if instantiating
9 -- any Xilinx primitives in this code.
10 --library UNISIM;
11 --use UNISIM.VComponents.all;
12
13 entity CO7SEG is
14     port (
15         Input : in std_logic_vector(3 downto 0);
16         Output : out std_logic_vector(6 downto 0));
17 end CO7SEG;
18
19 architecture Behavioral of CO7SEG is
20
21 begin
22
23     process (Input)
24     begin
25         case Input is
26             when "0000" =>
27                 Output <= "0000001"; -- 0
28             when "0001" =>
29                 Output <= "1001111"; -- 1
30             when "0010" =>
31                 Output <= "0010010"; -- 2
32             when "0011" =>
33                 Output <= "0000110"; -- 3
34             when "0100" =>
35                 Output <= "1001100"; -- 4
36             when "0101" =>
37                 Output <= "0100100"; -- 5
38             when "0110" =>
39                 Output <= "0100000"; -- 6
40             when "0111" =>
41
42                 Output <= "0001111"; -- 7
43             when "1000" =>
44                 Output <= "0000000"; -- 8
45             when "1001" =>
46                 Output <= "0000100"; -- 9
47             when "1010" =>
48                 Output <= "0001000"; -- A
49             when "1011" =>
50                 Output <= "1100000"; -- B
51             when "1100" =>
52                 Output <= "0110001"; -- C
53             when "1101" =>
54                 Output <= "1000010"; -- D
55             when "1110" =>
56                 Output <= "0110000"; -- E
57             when "1111" =>
58                 Output <= "0111000"; -- F
59             when others =>
60                 Output <= "-----"; -- Valor inválido
61         end case;
62     end process;
63
64 end Behavioral;
65
66

```

