

# EL253 - Sistemas Digitales

Semestre 2022-1

Profesor Kalun José Lau Gan

Sesión de Teoría Semana 3

1

## Agenda

- Modelo de sumador completo en VHDL
- Modelo de decodificador en VHDL
- Revisión teórica del multiplexor
- Lógica basada en multiplexores
- Modelo de multiplexor en VHDL
- Modelo de sumador binario en VHDL
- Modelo de comparador de magnitud VHDL
- Ejercicios

2

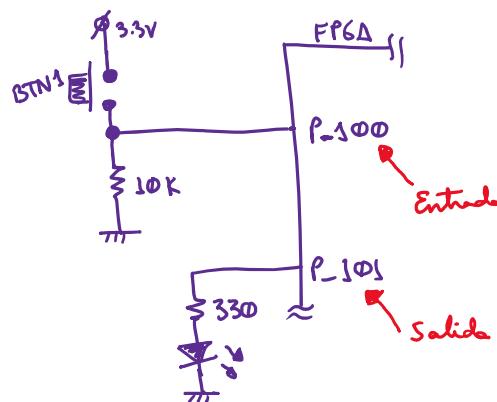
## Preguntas previas

- ¿Hay evaluación de laboratorio mañana?
  - Segundo sílabo si, está programado el LB1, lleguen puntuales, el que llegue tarde trabajará individualmente.
  - Los grupos son asignados de manera aleatoria con los que están presentes, para cada evaluación de laboratorio.
  - Recuerden: Alumno que llegue después de la asignación de grupos trabajará solo.
- ¿Cuánto durará el LB1?
  - La evaluación es durante el horario asignado, sin extensiones.
- ¿Qué hace que la tarjeta de FPGA pueda malograrse?
  - No tocarlo mucho (puedes dañarlo debido al ESD)
  - Incorrecta fuente de alimentación
  - Incorrecta implementación (ej. Colocar circuitería de entrada como si fuera de salida)
  - Ingresar señales de 5V en las E/S
  - Doblar la tarjeta, aplicarle calor o vibración excesivos

3

## Preguntas previas

- ¿Conexiones de E/S?
  - Para entradas = alta impedancia
  - Para salidas = baja impedancia



4

## Preguntas previas

- ¿Debo de tener soldados los pines de conexión en la tarjeta de FPGA?
  - Si, deben de estar correctamente soldadas para asegurarnos que las conexiones en la implementación no tengan falsos contactos.
- La configuración del FPGA se borra al desconectarle la energía en la tarjeta. ¿A qué se debe?
  - La configuración de la FPGA se aloja en celdas RAM

5

## Modelo de sumador en VHDL

- El código mostrado es de un sumador medio de 4 bits



```

1 library IEEE;
2 use IEEE.std_logic_1164.all;
3 use IEEE.std_logic_arith.all;
4 use IEEE.std_logic_unsigned.all;
5
6 entity sumadormon is
7 port( A, B: in std_logic_vector(3 downto 0);
8       SUM: out std_logic_vector(3 downto 0);
9       CO: out std_logic);
10 end sumadormon;
11
12 architecture flujaso of sumadormon is
13 signal INA, INB, OUTC: std_logic_vector(4 downto 0);
14 begin
15     INA(3 downto 0) <= A;
16     INB(3 downto 0) <= B;
17     OUTC <= INA + INB;
18     SUM <= OUTC(3 downto 0);
19     CO <= OUTC(4);
20 end flujaso;
  
```

6

Casos:

En decimal:

$$\begin{array}{r} 5+ \\ \hline 08 \\ \hline \overset{\text{C}}{\cancel{0}} \overset{\text{R}}{\cancel{0}} \end{array}$$

En binario:

$$\begin{array}{r} 9+ \\ \hline 18 \\ \hline \overset{\text{C}}{\cancel{1}} \overset{\text{R}}{\cancel{0}} \end{array}$$

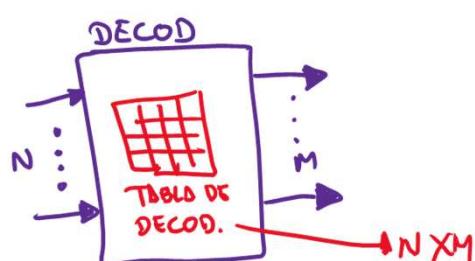
En hexadecimal.

$$\begin{array}{r} 10+ \\ \hline 11 \\ \hline \overset{\text{C}}{\cancel{1}} \overset{\text{R}}{\cancel{0}} \end{array}$$

$$\begin{array}{r} F+ \\ \hline F \\ \hline \overset{\text{C}}{\cancel{1}} \overset{\text{R}}{\cancel{E}} \end{array}$$

7

Modelo de decodificador en VHDL

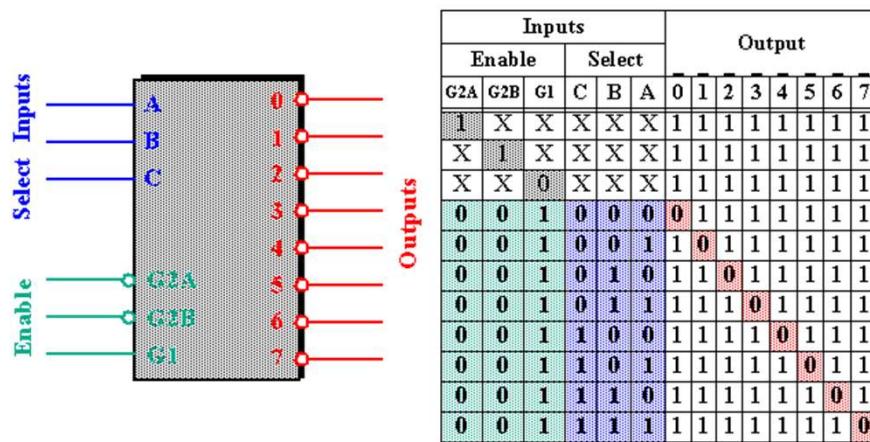


- Para cada combinación de entrada se obtiene una salida en base a la tabla de decodificación.
- El codificador (encoder) hace la función inversa del decodificador

8

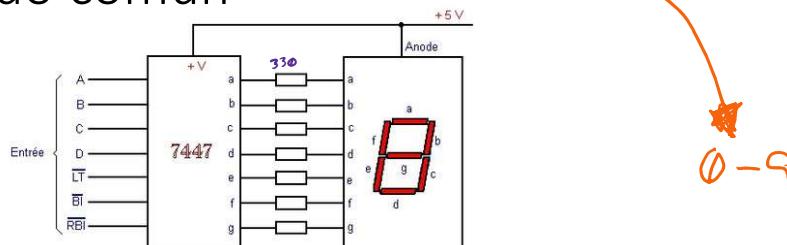
## Decodificador binario 74LS138

- Decodificador 3x8 (tres entradas, ocho salidas)
- Se le conoce también como decodificador de memoria.

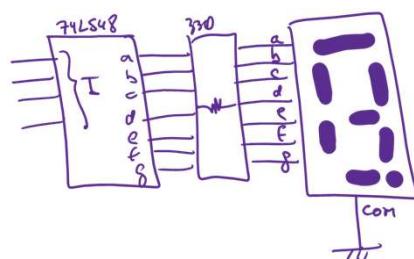


9

74LS47: Decodificador **BCD** – 7 segmentos del tipo ánodo común



- 74LS48 es para usarse con displays de cátodo común



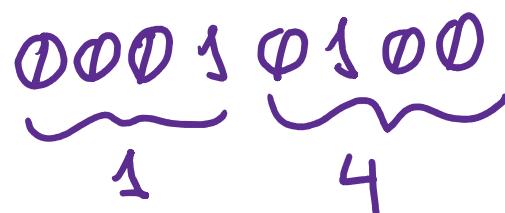
10

Tabla de decodificación para display de siete segmentos:

Decimal Digit	Input lines				Output lines							Display pattern
	A	B	C	D	a	b	c	d	e	f	g	
0	0	0	0	0	1	1	1	1	1	1	0	0
1	0	0	0	1	0	1	1	0	0	0	0	1
2	0	0	1	0	1	1	0	1	1	0	1	2
3	0	0	1	1	1	1	1	1	0	0	1	3
4	0	1	0	0	0	1	1	0	0	1	1	4
5	0	1	0	1	1	0	1	1	0	1	1	5
6	0	1	1	0	1	0	1	1	1	1	1	6
7	0	1	1	1	1	1	1	0	0	0	0	7
8	1	0	0	0	1	1	1	1	1	1	1	8
9	1	0	0	1	1	1	1	1	0	1	1	9

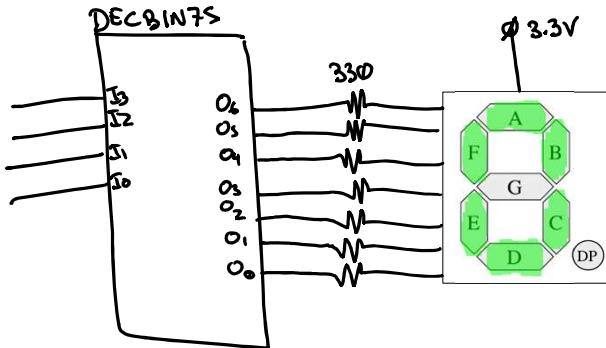
11

Caso: Representar el número 14 (decimal) en formato BCD:


  
binario:  
1110

12

## Decodificador Binario a siete segmentos (ánodo común) en sentencia process (case—when)



¿Cómo modiflico el código descriptivo para que funcione en displays de cátodo común?

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY decoder_test IS
PORT(
    I : IN std_logic_vector (3 DOWNTO 0);
    O : OUT std_logic_vector (6 DOWNTO 0)
);

END decoder_test ;
ARCHITECTURE Behavioral OF decoder_test IS
BEGIN
    decode:PROCESS(I)
    BEGIN
        CASE I IS
            WHEN "0000" => O <= "1000000"; -- 0 3f
            WHEN "0001" => O <= "1111001"; -- 1 06
            WHEN "0010" => O <= "0100100"; -- 2 5b
            WHEN "0011" => O <= "0110000"; -- 3 4f
            WHEN "0100" => O <= "0011001"; -- 4 66
            WHEN "0101" => O <= "0010010"; -- 5 6d
            WHEN "0110" => O <= "0000010"; -- 6 7d
            WHEN "0111" => O <= "1111000"; -- 7 07
            WHEN "1000" => O <= "0000000"; -- 8 7f
            WHEN "1001" => O <= "0010000"; -- 9 67
            WHEN "1010" => O <= "0001000"; -- A 77
            WHEN "1011" => O <= "0000111"; -- B 7c
            WHEN "1100" => O <= "1000110"; -- C 39
            WHEN "1101" => O <= "0100001"; -- D 5e
            WHEN "1110" => O <= "0000110"; -- E 79
            WHEN "1111" => O <= "0001110"; -- F 71
            WHEN others => O <= "1111111"; -- 00
        END CASE;
    END PROCESS decode;
END Behavioral;

```

13

## Decodificador Binario a siete segmentos (cátodo común) en sentencia selectiva (with-select)

- En este modelo las salidas están individualizadas (Sa, Sb, Sc, Sd, Se, Sf, Sg) y la entrada E es un vector de 4 bits

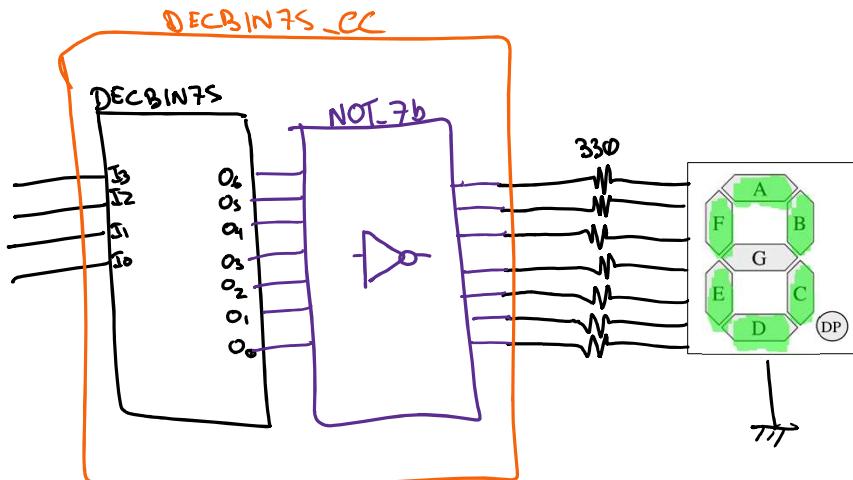
```

1 library IEEE;
2 use IEEE.std_logic_1164.all;
3 use IEEE.std_logic_arith.all;
4 use IEEE.std_logic_unsigned.all;
5
6 entity decbin7scc is
7 port( E: in std_logic_vector(3 downto 0);
8 Sa, Sb, Sc, Sd, Se, Sf, Sg: out std_logic);
9 end decbin7scc;
10
11 architecture combinacional of decbin7scc is
12 signal interno: std_logic_vector(6 downto 0);
13 begin
14 with E select
15     interno <= "0111111" when "0000",
16             "0000110" when "0001",
17             "1011011" when "0010",
18             "1001111" when "0011",
19             "1100110" when "0100",
20             "1101101" when "0101",
21             "1111101" when "0110",
22             "0000111" when "0111",
23             "1111111" when "1000",
24             "1100111" when "1001",
25             "1110111" when "1010",
26             "1111100" when "1011",
27             "0111001" when "1100",
28             "1011110" when "1101",
29             "1111001" when "1110",
30             "1110001" when "1111",
31             "ZZZZZZZ" when others;
32 Sa <= interno(0);
33 Sb <= interno(1);
34 Sc <= interno(2);
35 Sd <= interno(3);
36 Se <= interno(4);
37 Sf <= interno(5);
38 Sg <= interno(6);
39 end combinacional;

```

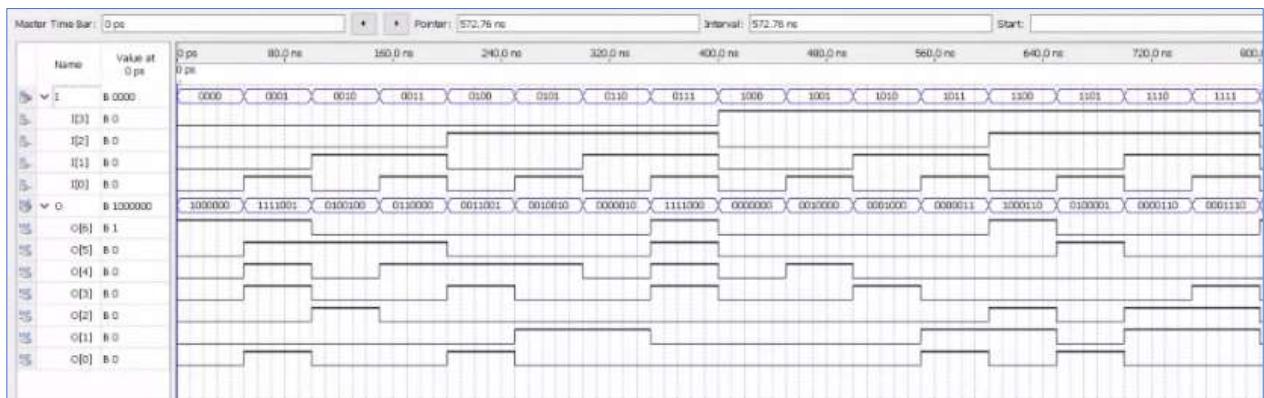
14

Caso: Si ya he realizado el modelo para ánodo común. ¿Cómo lo modiflico para que funcione en displays de cátodo común?



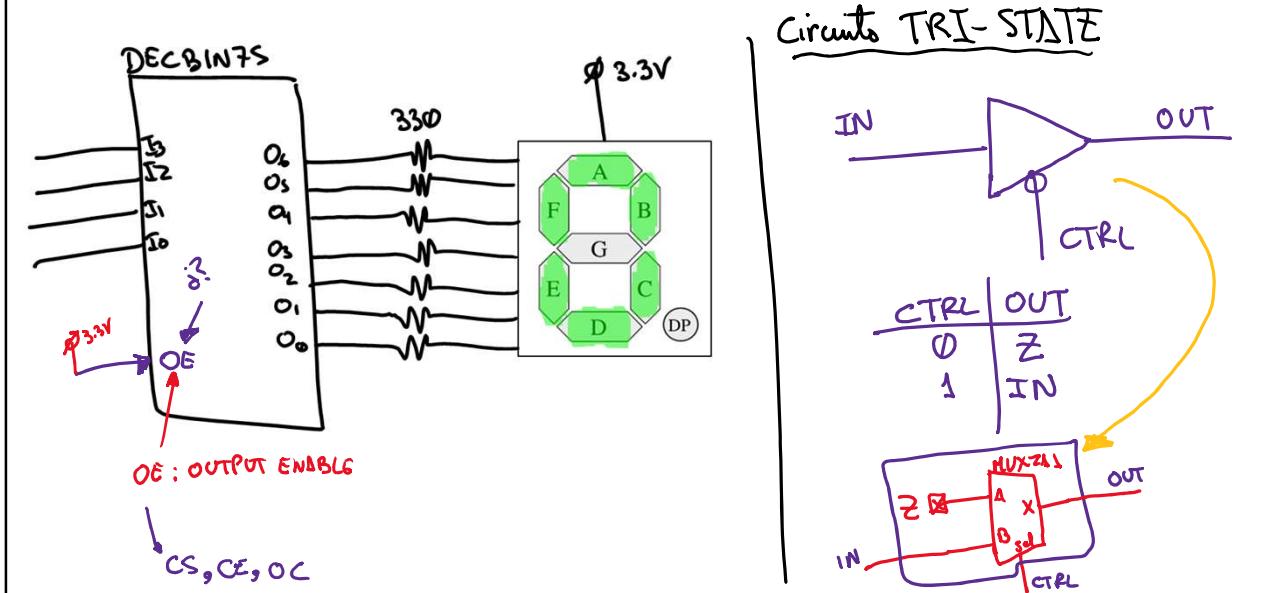
15

Simulación en formas de onda:



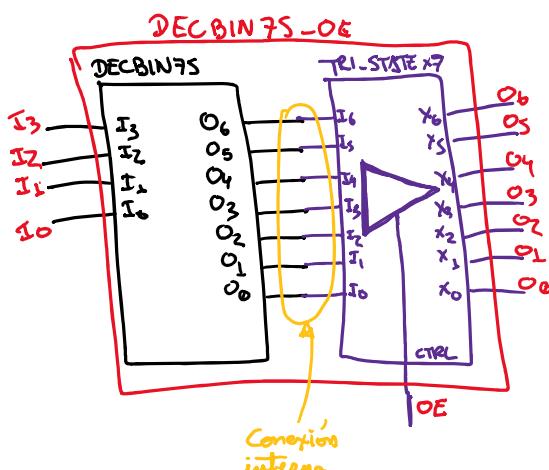
16

## Señales de control ó habilitación de circuitos



17

## Implementando DECBIN7S con OE



```

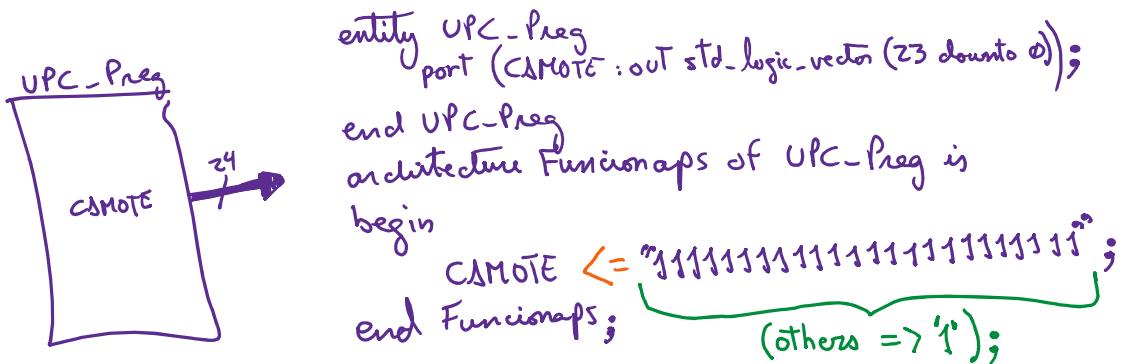
1 LIBRARY IEEE;
2 USE IEEE.std_logic_1164.all;
3 USE IEEE.std_logic_arith.all;
4 USE IEEE.std_logic_unsigned.all;
5
6 ENTITY decbin7s IS
7 PORT (
8   I: IN std_logic_vector (3 downto 0);
9   OE: IN std_logic;
10  O: OUT std_logic_vector (6 downto 0)
11 );
12 END decbin7s;
13
14 ARCHITECTURE Behavioral OF decbin7s IS
15 signal interno: std_logic_vector(6 downto 0);
16 BEGIN
17   with OE select
18     O <= interno when '1', (others => 'Z') when others;
19   decode: PROCESS(I)
20 BEGIN
21   CASE I IS
22     WHEN "0000" => interno <= "1000000";
23     WHEN "0001" => interno <= "1111001";
24     WHEN "0010" => interno <= "0100100";
25     WHEN "0011" => interno <= "0110000";
26     WHEN "0100" => interno <= "0011001";
27     WHEN "0101" => interno <= "0010010";
28     WHEN "0110" => interno <= "0000010";
29     WHEN "0111" => interno <= "1111000";
30     WHEN "1000" => interno <= "0000000";
31     WHEN "1001" => interno <= "0010000";
32     WHEN "1010" => interno <= "0001000";
33     WHEN "1011" => interno <= "0000011";
34     WHEN "1100" => interno <= "1000110";
35     WHEN "1101" => interno <= "0100001";
36     WHEN "1110" => interno <= "0000110";
37     WHEN "1111" => interno <= "0001110";
38   END CASE;
39   END PROCESS decode;
40 END Behavioral;
41
42

```

18

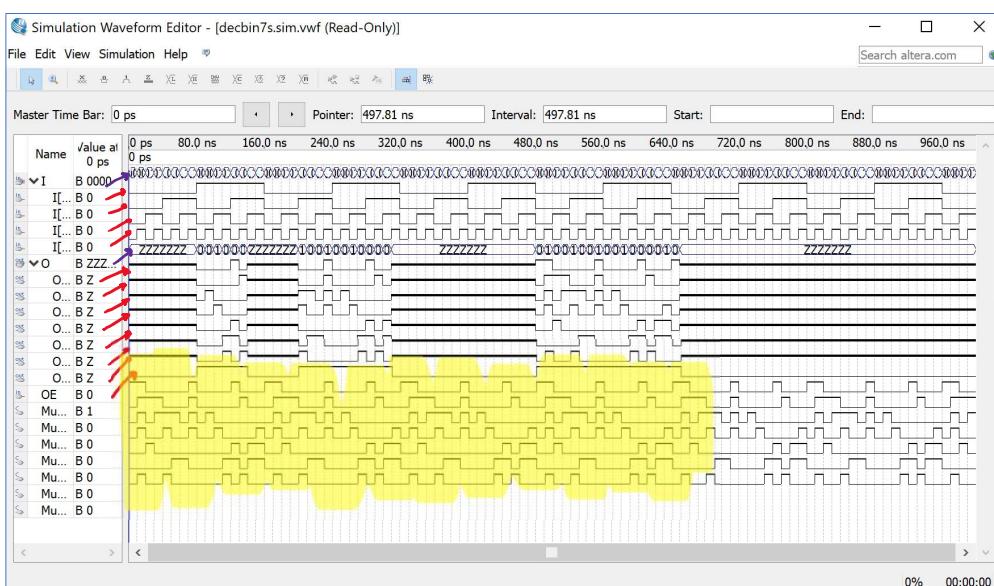
## Caso:

- Tengo una señal de salida de nombre CAMOTE, de 24 bits y deseo que todos sus bits estén en uno lógico.



19

## Simulación en formas de onda de decodificador con habilitador



20

## El multiplexor

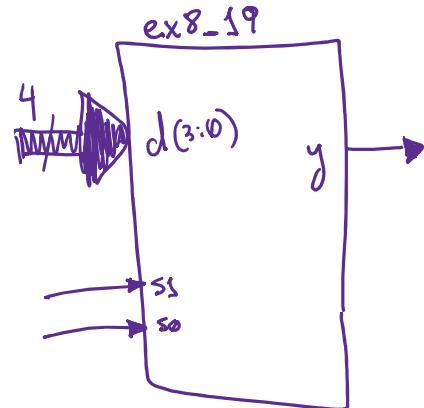
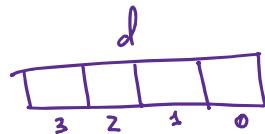
Analicemos el siguiente código en VHDL:

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL; -- 4-line multiplexer
-- using the
-- selected signal assignment
ENTITY ex8_19 IS
  PORT(d : IN  std_logic_vector (3 DOWNTO 0);
        s : IN  std_logic_vector (1 DOWNTO 0);
        y : OUT std_logic);
END ex8_19;

ARCHITECTURE arc OF ex8_19 IS
BEGIN
  WITH s SELECT
    y<=d(3) WHEN "11"
    d(2) WHEN "10",
    d(1) WHEN "01",
    d(0) WHEN "00",
    '0' WHEN OTHERS;
END arc;

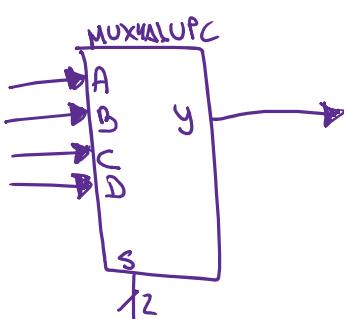
```



debido a IEEE1164

21

Modificando el anterior MUX para que las entradas de datos sean señales independientes



```

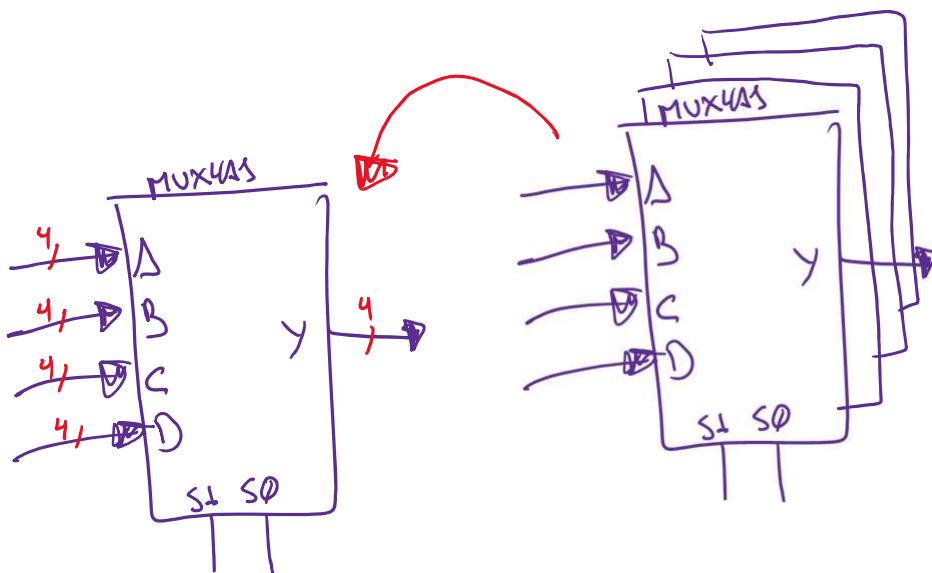
entity MUX4A1UPC is
  port( A, B, C, D:  in std_logic;
        S:         in std_logic_vector(1 downto 0);
        Y:         out std_logic);
end MUX4A1UPC;

architecture Flujazo of MUX4A1UPC is
begin
  with S select
    Y <= A when "00",
    B when "01",
    C when "10",
    D when "11",
    'Z' when others;
end Flujazo;

```

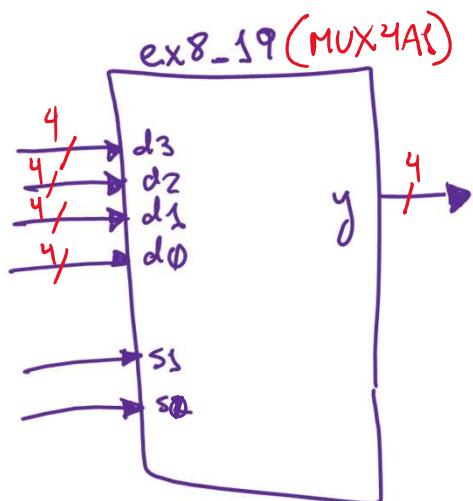
22

## MUX4A1 de 4 bits de ancho



23

Modificando el anterior circuito para que sea de ancho de 4 bits en los datos



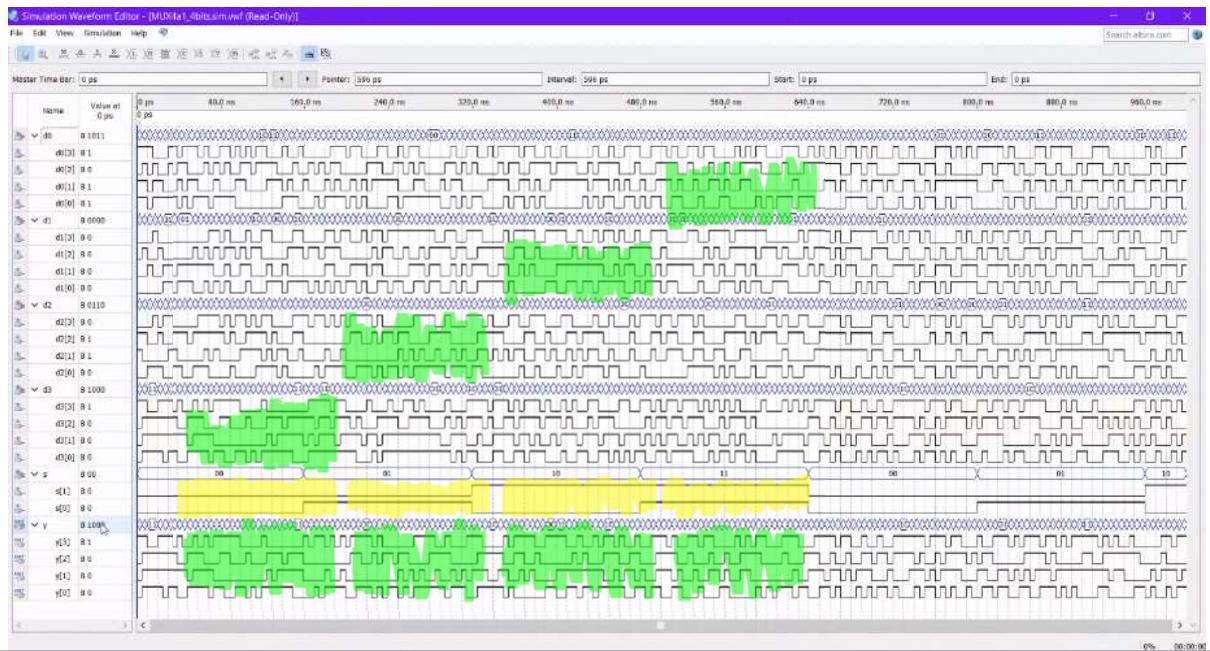
```

entity ex8_19 is
port (d3, d2, d1, d0: in std_logic_vector(3 downto 0);
      s: in std_logic_vector (1 downto 0);
      y: out std_logic_vector(3 downto 0));
end ex8_19;
architecture mas_grande of ex8_19 is
begin
  with s select
    y <= d3 when "00",
    d2 when "01",
    d1 when "10",
    d0 when "11",
    (others => 'Z') when others;
end mas_grande;

```

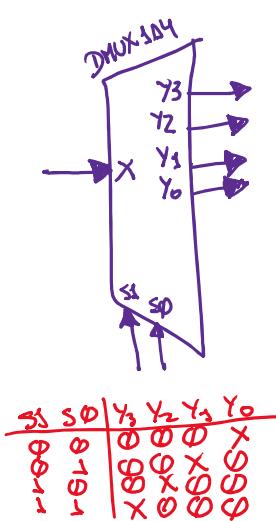
24

## Simulación



25

## Demultiplexor: función inversa del MUX



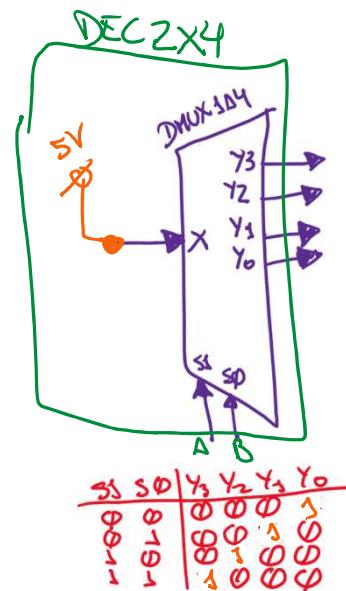
```

entity dmux1a4
port(x: in std_logic;
      s: in std_logic_vector(1 downto 0);
      y: out std_logic_vector(3 downto 0));
end dmux1a4;

architecture flujazo2 of dmux1a4 is
begin
  with s select
    y(3) <= x when "11", '0' when others;
  with s select
    y(2) <= x when "10", '0' when others;
  with s select
    y(1) <= x when "01", '0' when others;
  with s select
    y(0) <= x when "00", '0' when others;
end flujazo2;
  
```

26

Ejercicio: Desarrollar un decodificador binario 2x4 empleando demultiplexor 1 a 4



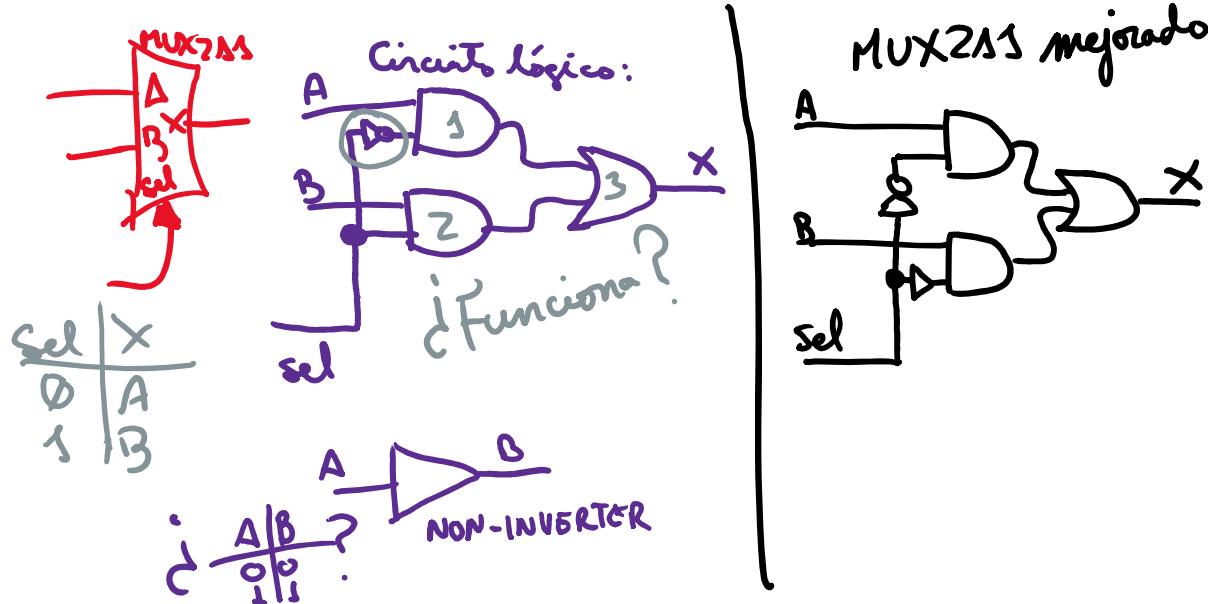
27

### Ejercicio:

- Desarrollar un demultiplexor 1 a 8 donde la entrada de datos tenga un ancho de 8 bits al igual que las salidas de distribución, contemplar una entrada OE para controlar la habilitación de las salidas ('1' habilitadas las salidas, '0' las salidas se mantendrán en 'Z')
- Desarrollar un decodificador binario-palabra con conexión a un display de siete segmentos del tipo cátodo común. El decodificador tendrá una tabla con las letras de la palabra “INGENIERIA” de manera ordenada. De tal manera que cuando se ingrese una cuenta de manera secuencial a la entrada de datos del decodificador, se mostrará la referida palabra a razón de una letra a la vez.

28

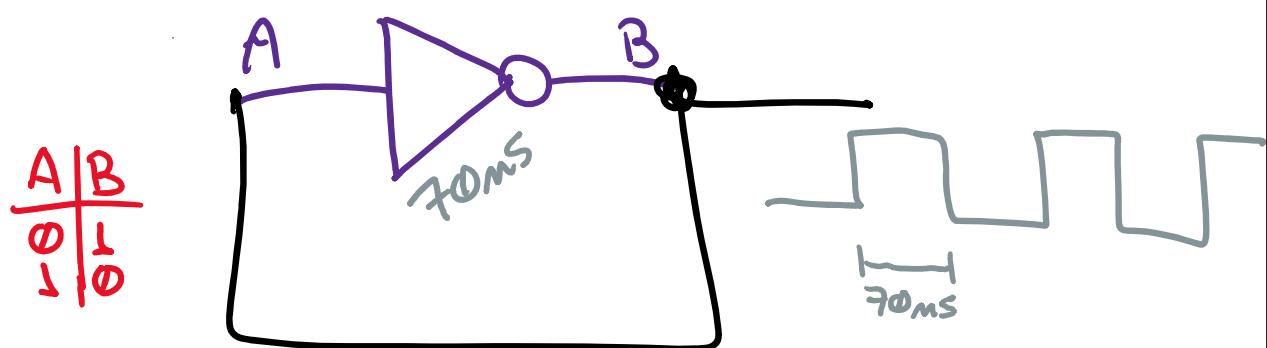
## El multiplexor (MSI)



29

Un paréntesis:

Tiempo real  $\leftarrow$  mínimo retardos



- Revisar retardos: inercial y de transporte
- Velocidad de la señal eléctrica en cobre.

30

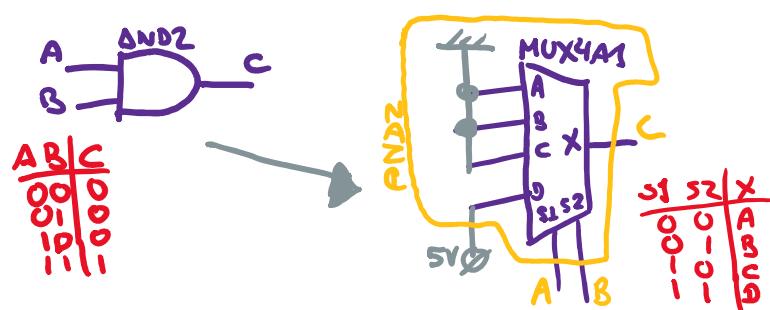
## Aplicaciones desarrolladas con multiplexores:

- Comunicaciones celulares (TDM,FDM,CDM)
- Seleccionadores de equipajes en los aeropuertos
- Filtrado de información
- Clasificación de tamaño de frutas

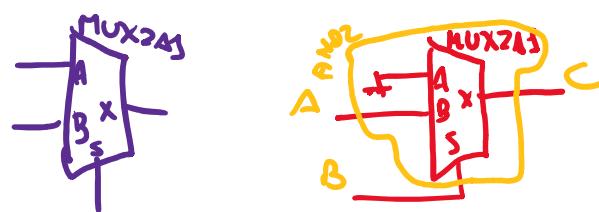


31

## Lógica basada en multiplexores



¿Se podrá construir una AND2 con MUX2A1?

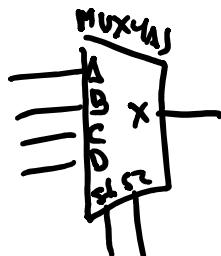


32

### Ejercicio:

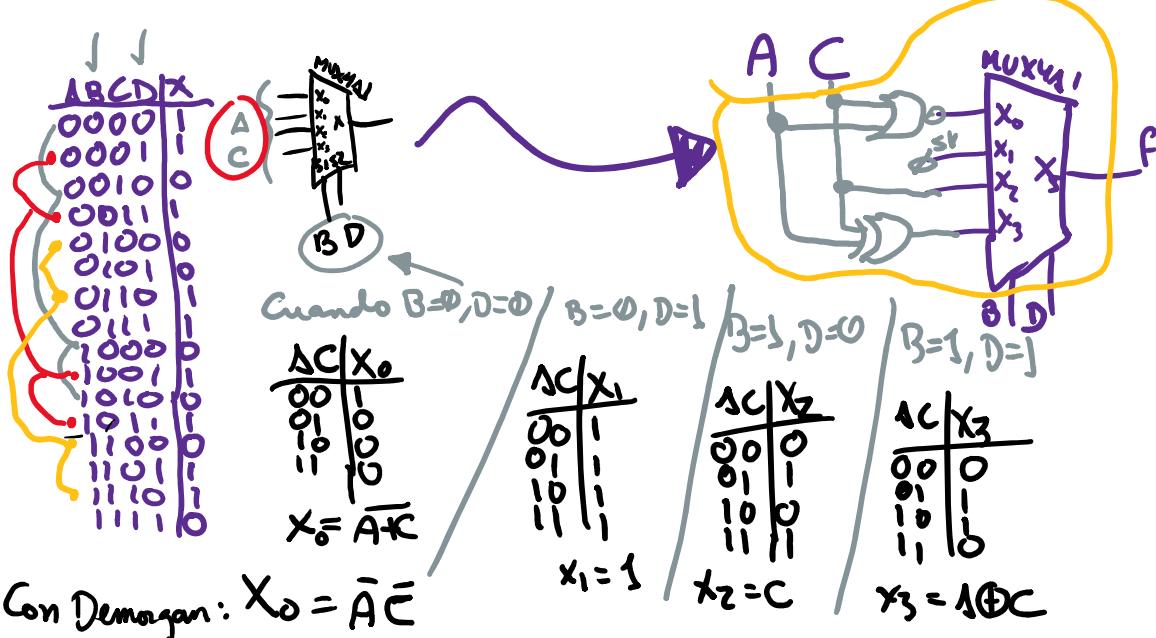
- Implementar la siguiente función empleando lógica basada en solo multiplexor de 4A1:

$$f = \sum_{m=4} (0, 1, 3, 6, 7, 9, 11, 13, 14)$$



33

### Método para solucionar lo anterior:



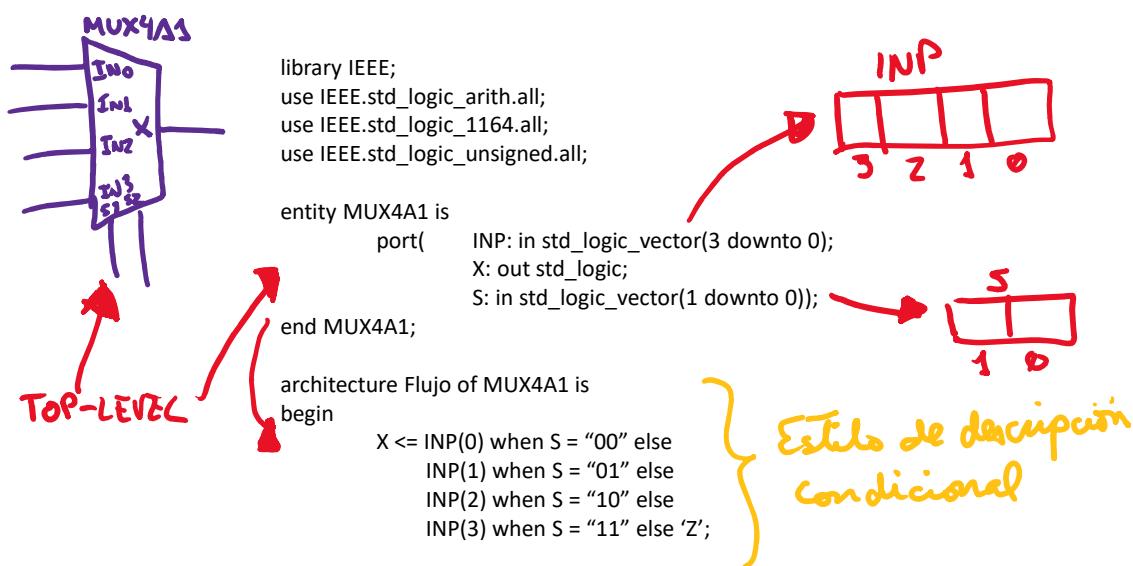
34

## ¿Por qué hemos visto lo anterior?

- Porque los dispositivos FPGA emplean lógica de multiplexor para implementar las funciones lógicas!

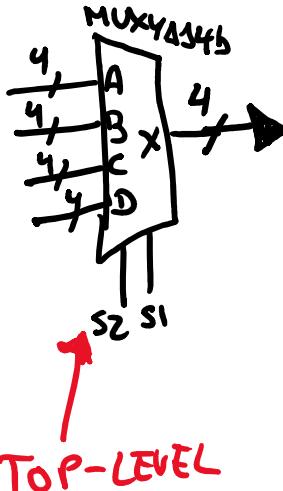
35

## ¿Cuál es el modelo de un MUX4A1 en VHDL?



36

¿Cuál es el modelo de un MUX4A1 de 4 bits en VHDL?



```

library IEEE;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity MUX4A14b is
port( A, B, C, D: in std_logic_vector(3 downto 0);
      X: out std_logic_vector(3 downto 0);
      S: in std_logic_vector(1 downto 0));
end MUX4A14b;

architecture Flujo of MUX4A14b is
begin
--      X <= A when S = "00" else
--      B when S = "01" else
--      C when S = "10" else
--      D when S = "11" else 'Z';
      with S select
          X <= A when "00", B when "01", C when "10",
          D when "11", (others => 'Z') when others;
end Flujo;

```

D, B, C, D  
3 2 1 0

condicional  
selección.

37

## El comparador de magnitud

- Compara dos números de igual magnitud y determina si son mayor, igual ó menor.



```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity comparador_4b is
port( EN_A, EN_B:           in std_logic_vector(3 downto 0);
      mayor, igual, menor: out std_logic);
end comparador_4b;

architecture flujoso of comparador_4b is
begin
process(EN_A, EN_B)
begin
    if EN_A > EN_B then
        mayor <= '1';
        igual <= '0';
        menor <= '0';
    elsif EN_A = EN_B then
        mayor <= '0';
        igual <= '1';
        menor <= '0';
    else
        mayor <= '0';
        igual <= '0';
        menor <= '1';
    end if;
end process;
end flujoso;

```

38

Fin de la sesión de teoría