LECCIÓN 2. VHDL por comportamiento

Deco 3 → 8 con habilitación V8

74HC138

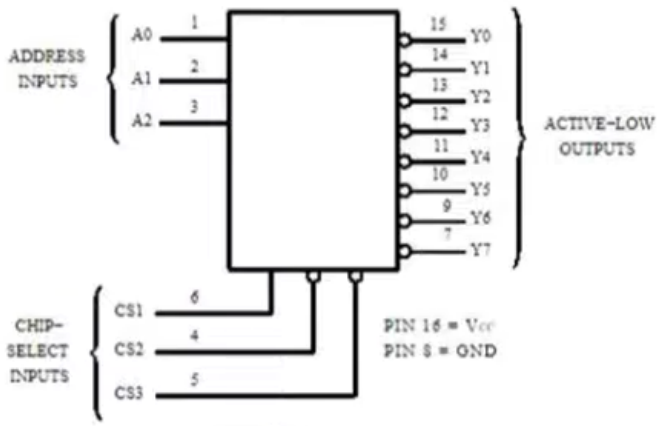


Figure 2. Logic Diagram

```
entity Deco3a8 is;
    Port
        (
        a_i : in  std_logic_vector(2 downto 0);
        en_i : in  std_logic;
        Y_i : out std_logic_vector(7 downto 0)
        );
end entity Deco3a8;
```

FUNCTION TABLE

INPUTS						OUTPUTS							
$\overline{E_1}$	$\overline{E_2}$	E_3	A_0	A_1	A_2	$\overline{Y_0}$	$\overline{Y_1}$	$\overline{Y_2}$	$\overline{Y_3}$	$\overline{Y_4}$	$\overline{Y_5}$	$\overline{Y_6}$	$\overline{Y_7}$
H	X	X	X	X	X	H	H	H	H	H	H	H	H
X	H	X	X	X	X	H	H	H	H	H	H	H	H
X	X	L	X	X	X	H	H	H	H	H	H	H	H
L	L	H	L	L	L	L	H	H	H	H	H	H	H
L	L	H	H	L	L	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	L	H	H	H	H	H
L	L	H	H	H	L	H	H	H	L	H	H	H	H
L	L	H	L	L	H	H	H	H	H	L	H	H	H
L	L	H	H	L	H	H	H	H	H	H	L	H	H
L	L	H	L	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	L

Notes

1. H = HIGH voltage level
   L = LOW voltage level
   X = don't care

```
arch... Tabla of Deco3a8 is
    signa auxY : std_logic_vector(7 downto 0);
begin
    With a_i select
        auxY <= "00000001" when "000",
                :
                "00000000" when others;
    Y_o <= auxY when ena_i = '1' else
           "00000000";
end architecture Tabla
```

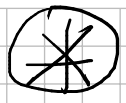
el "00000000" se puede reemplazar:

(others => '0')

Con esto no es necesario escribir todos los bits, es más general.

Testbench para Deco3a8

```
entity Deco3a8 is
end entity Deco3a8;
architecture Test of Deco3a8_tb is
    Component Deco3a8 is
     Port(a_i : in std_logic_vector(2 downto 0);
     ena_i : in std_logic;
      Y_o : out std_logic_vector(7 downto 0));
     end component Deco3a8;
    Signal a_t : std_logic_vector(2 downto 0) := "000";
    Signal ena_t : std_logic := '1';
    Signal X_t : std_logic_vector (7 downto 0);
begin
dut : Deco3a8 port map (a_i => a_t,
                        ena_i => ena_t,
                        Y_o => Y_t);
Prueba:
    Process begin
        report "Verificando el deco 3a8"
        Severity note;
        -- aqui van las Pruebas (*)
        report "Verificacion exitosa!"
        severity note;
        wait;
    End Process Prueba
End architecture
```



```
ena_t <= '0';
a_t <= "100";          } estímulo
wait for 1ns;
assert Y_t = "0000 0000"
        report "Activa salida con ena = 0"
        severity failure;
```

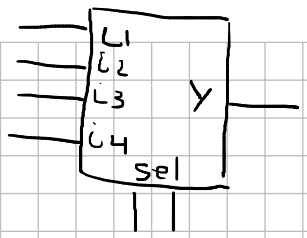
Repito para otros casos

Otra forma

```
ena_t <= '0';
for i in 0 to 7 loop
  a_t <= std_logic_vector(to_unsigned(i,3));
  wait for 1ns;
  assert to_integer(unsigned(Y_t)) = 2**i
     report "No activa la salida" & integer'image(i)
                & " con ena = 1"
     severity failure;
end loop;
```

A veces puede ser mejor intercalar con algunos casos y luego todos los demás con ciclos for.

Mux 4 canales V10




```
entity Mux4 is
    port
    (
        i_i : in std_logic_vector (3 downto 0);
      sel_i : in std_logic_vector(1 downto 0);
        y_o : out  std_logic
    );
end entity Mux4

architecture Tabla of Mux4 is
begin
        with  sel_i  select
      y_o <= i_i(0) when "00",
             i_i(1) when "01",
             i_i(2) when "10",
             i_i(3) when others;        ???
end architecture Tabla;
```

Test bench

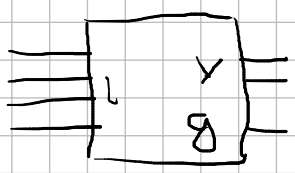
```
library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

Entity Mux4_tb is
end entity Mux4_tb;

architecture Test of Mux4_tb is

    Component Mux4 is
        Port( D_i   : in std_logic_vector(3 downto 0);
              sel_i : in "          (1 downto 0);
              Y_o   : out std_logic );
    end component Mux4;

    Signal i_t : std_logic_vector(3 downto 0) := "0000";
    Signal sel_t : std_logic_vector(1 downto 0) := "00";
    Signal Y_t : std_logic;

begin
dut : MUX4 Port map (i_i => i_t,
                     sel_i => sel_t,
                     Y_o => Y_t);

Pruebas:

    Process begin
        report "verificando el mux..";
        severity note;
```

```
i_t <= "1010";
sel_t <= "10";
wait for 1ns;
assert Y_t='0'
        report "falla para sel=2 e i=1010"
        severity failure;
i_t <= "1111"
for i in 0 to 3 loop
  sel_t <= std_logic_vector (to_unsgnd(i,2));
  wait for 1ns;
  assert Y_t=i_t(i)
    report "falla sel=" & integer'image(i) &
                & " e i=1111"
    Severity failure;
end loop;
  .
  .
  .
report "Verificacion exitosa"
Severity note;
wait;
end process Prueb
end architecture Test;
```

Probando las 64 posibilidades, ahora d_t no es fijo sino que cambie con un for

```
for j in 0 to 15 loop
    d_t <= std_logic_vector(to_unsigned(j,4));
    for i in 0 to 3 loop
        sel_t <= std_logic_vector(to_unsgnd(i,2));
        wait for 1 ns;
        assert x_t = d_t(i);
            report "falla sel=" & integer'image(i) &
                    & " e i=1111"
            Severity failure;
    end loop;
end loop;
```

Descripción codificador de prioridad VII

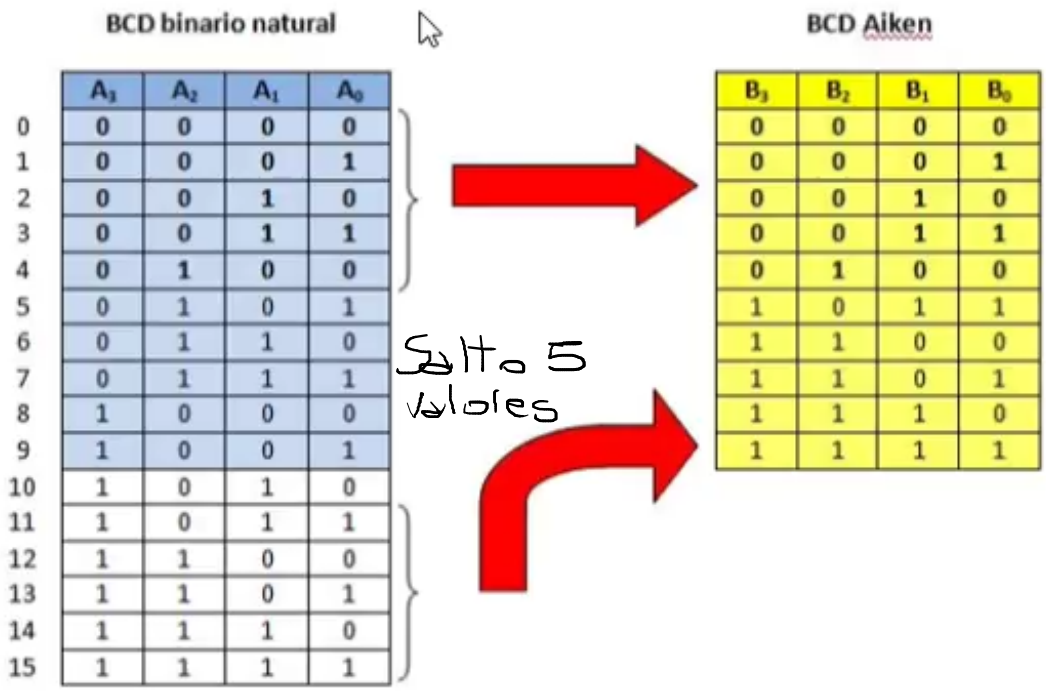



```
entity Cod4a2 is
    Port
    (
        i_i : in std_logic_vector(3 downto 0);
        Y_o : out    "    (1 downto 0);
        g_o : out std_logic
    );
-- La entrada 3 es la de mayor prioridad
-- G=1 Hay al menos una entrada activa

architecture WhenElse of Cod4a2 is
begin
        Y_o <= "11" when i_i(3) = '1' else
               "10" when i_i(2) = '1' else
               "01" when i_i(1) = '1' else
               "00";
  -- esta ultima no tiene condicion, siempre será
     "00" la salida.
  -- Analiza hasta que se cumpla.
     g_o <= i_i(3) or i_i(2) or i_i(1) or i_i(0);
  -- Señal de grupo indica ninguna activa
end architecture WhenElse;
```

la prueba sería:

```
for i in 0 to 15 loop
  i_t <= std_logic_vector(to_unsigned(i,4));
  wait for 1ns;
     if i_t(3) = '1' then
          assert y_t = "1_1"
           report ...    & integer'image(i) & ...
           severity failure
          assert s_t '1'
             report ...
             severity ...
     elsif i_t(2) = '1' then
                .
                .
                .

     else
                .
                .
                .
     end if;
end loop
```

Conversor BCD Natural → BCD Aiken V12



	A_3	A_2	A_1	A_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

B_3	B_2	B_1	B_0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

```
entity Nat2Aik is
port
(
    n_i in std_logic_vector(3 downto 0);
    a_o out std_logic_vetor(3 downto 0)
);
end entity Nat2Aik;
```

Con STD_logic no se pueden realizar operaciones

Queremos comparar con 5 y sumar 6

a_0 <= n_i when n_i < 5 else n_i + 6;

No se puede realizar, se necesita cambiar el tipo de datos

Para esto usamos el package numeric_std de IEEE.

use ieee.numeric_std.all;

a_0 <= n_i when unsigned(n_i) < 5 ...
... else std_logic_vector(unsigned(n_i) + 6)

Se castea unsigned(n_i) + 6 → un vector

```
arch
    comp
        Port (n_i : in std_... ( 3dto0) := "0000";
    end

    signal n_t :
    signal

type tabla is array(0 to 9) of std_l_v (3dto0);

constant TABLA_AIKEN:tabla := ("0000",
                               "0001",
                                  :
                               "1111" );

begin

dut: Nat2Aik Port map (n_i => n_t,
                       a_o => a_t);

                    ← ATRIBUTO

    For i in tabla'range loop
        n_t <= std_logic_vector (to_unsgnd(i,4));
        wait for 1 ns;
        assert a_t = TABLA_AIKEN(i);          ← valor estímulo
          report "falla para =" &
                 &integer'image (to_integer (unsigned) (n_t))
          Severity failure;
```

El estímulo no es un string, entonces se castea

Atributos (*Attribute*) para arreglos (*array*)

'high	el elemento mayor
'low	el elemento menor
'right	q' está más a la derecha
'left	q' está más a la izquierda
'range	el rango (0 to 4)
'reverse_range	(4 downto 0)
'length	cantidad elementos
'ascending	Si están ordenados da True