Capítulo 3

Circuitos Lógicos Combinatorios

Circuitos Lógicos Combinatorios

Diseño (Programación) de una Estructura Básica Combinatoria

Declaración *Entidad*

Declaración *Arquitectura*

Sintaxis:

architecture *nombre_arquitectura* of *nombre_entidad* is

{*Declarativas de Bloque*} –Se analizarán posteriormente

begin

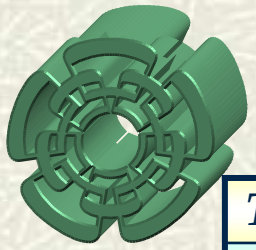
{*Enunciados Concurrentes*}

end [*nombre_arquitectura*]

Enunciado Concurrente.

Unidad de Cómputo/Cálculo que realiza lo siguiente:

- Lectura de Señales.
- Realiza cálculos basados en los valores de las Señales.
- Asigna los valores calculados a Señales específicas.



Enunciados Concurrentes

Tipos de Enunciados Concurrentes.	
Asignación de Señal	Permite asignar un valor calculado a una señal o puerto.
Proceso (**process**)	Permite definir un algoritmo secuencial que lee valores de Señales y calcula nuevos valores que son asignados a otras Señales.
Bloque (**block**)	Grupo de enunciados concurrentes.
Llamada a un Componente predefinido	
Llamada a un Procedimiento (**procedure**)	Llama a un algoritmo que calcula y asigna valores a Señales

Asignación de Señales

Tipos:

- Asignaciones Condicionales de Señales – La construcción **when-else**
- Asignaciones de Señales mediante **Ecuaciones Booleanas**
- Asignaciones de Señales por Selección – La construcción **with-select-when**

Operadores Lógicos

Operadores Lógicos
and, or, xor, nand, nor, xnor, not
Tipos de Operandos permisibles: *bit, boolean y arreglos unidimensionales* (del tipo bit o boolean)
Operandos deben tener la misma longitud, excepto para el operador **not**, el cual se aplica por lo general a un solo operando.
Si una expresión incluye varios de estos operadores (p.ej. AND, NOT, XNOR) es necesario utilizar paréntesis para evaluarla correctamente.

Ejemplos:

Ecuación Booleana	Expresión VHDL
$q = a + (x \cdot y)$	q = a or (x and y)
$y = a + (\overline{b} \cdot \overline{c}) + d$	y = a or (not b and not c) or d

Asignación de Señales

Asignaciones Condicionales de Señales - La construcción **when-else**

a	b	c	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

L	Ejemplo Nº 1 - Uso de la construcción **when-else**
1	**library** ieee;
2	**use** ieee.std_logic_1164.**all**;
3	**entity** tabla **is**
4	**port** (a,b,c: **in** std_logic;
5	f: **out** std_logic);
6	**end** tabla;
7	**architecture** arq_tabla **of** tabla **is**
8	**begin**
9	f <= ‘1’ **when** (a = ‘0’ **and** b=‘0’ **and** c=‘0’) **else**
10	‘1’ **when** (a = ‘0’ **and** b=‘1’ **and** c=‘1’) **else**
11	‘1’ **when** (a = ‘1’ **and** b=‘1’ **and** c=‘0’) **else**
12	‘1’ **when** (a = ‘1’ **and** b=‘1’ **and** c=‘1’) **else**
13	‘0’;
14	**end** arq_tabla;

La construcción **when-else** permite definir paso a paso el comportamiento de un sistema. Para esto, se declaran los valores que se deben asignar a una señal (o grupo) en función de las diferentes condiciones de entrada posibles. El orden en el que se declaren las condiciones de entrada, no es importante.

Asignación de Señales

A	B	C	D	F
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

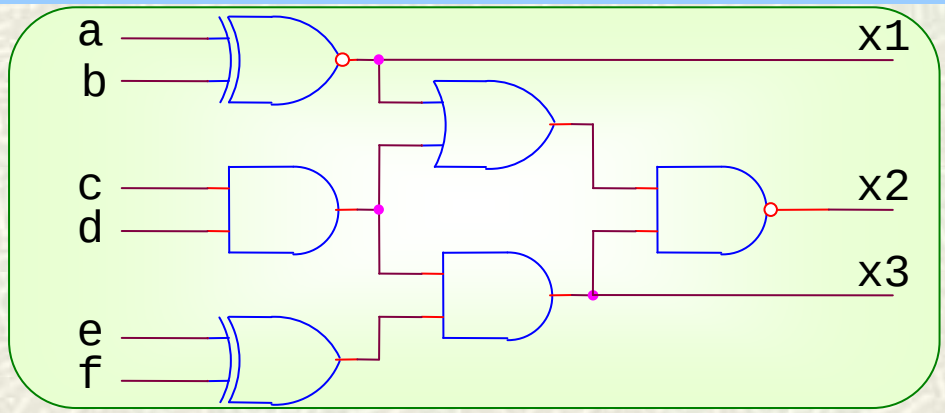
Asignaciones Condicionales de Señales - La construcción **when-else**

L	Ejemplo Nº 2 - Uso de la construcción **when-else**
1	**library** ieee;
2	**use** ieee.std_logic_1164.**all**;
3	**entity** funcion **is**
4	**port** (A,B,C,D: **in** std_logic;
5	F: **out** std_logic);
6	**end** funcion;
7	**architecture** a_func **of** funcion **is**
8	**begin**
9	F <= ‘1’ **when** (A = ‘0’ **and** B=‘0’ **and** C=‘0’ **and** D=‘0’) **else**
10	‘1’ **when** (A = ‘0’ **and** B=‘1’ **and** C=‘0’ **and** D=‘1’) **else**
11	‘1’ **when** (A = ‘0’ **and** B=‘1’ **and** C=‘1’ **and** D=‘0’) **else**
12	‘1’ **when** (A = ‘1’ **and** B=‘1’ **and** C=‘1’ **and** D=‘1’) **else**
13	‘0’;
14	**end** a_func;

Asignación de Señales

Asignaciones de Señales mediante

Ecuaciones Booleanas



L	Ejemplo Nº 3 – Asignaciones de Señales – Uso de Ecs. Booleanas
1	**library** ieee;
2	**use** ieee.std_logic_1164.**all**;
3	**entity** logica **is**
4	**port** (a,b,c, d, e, f: **in** std_logic;
5	x1, x2, x3: **out** std_logic);
6	**end** logica;
7	**architecture** booleana **of** logica **is**
8	**begin**
9	x1 <= a **xnor** b;
10	x2 <= ((c **and** d) **or** (a **xnor** b)) **nand** ((e **xor** f) **and** (c **and** d));
11	x3 <= (e **xor** f) **and** (c **and** d);
12	**end** booleana;

En este tipo de asignaciones, cada función de salida es descrita mediante su ecuación booleana correspondiente, lo cual implica el uso de operadores lógicos.

Asignación de Señales

Asignaciones de Señales mediante Ecuaciones Booleanas

A	B	C	X	Y	Z
0	0	0	1	0	1
0	0	1	1	1	0
0	1	0	0	0	1
0	1	1	1	0	1
1	0	0	0	0	0
1	0	1	0	1	0
1	1	0	0	1	0
1	1	1	1	0	0

L	Ejemplo Nº 4 – Asignaciones de Señales – Uso de Ecs. Booleanas
1	**library** ieee;
2	**use** ieee.std_logic_1164.**all**;
3	**entity** logica **is**
4	**port** (A,B,C: **in** std_logic;
5	X,Y,Z: **out** std_logic);
6	**end** logica;
7	**architecture** a_log **of** logica **is**
8	**begin**
9	X <= (not A and not B and not C) or (not A and not B and C)
10	or (not A and B and C) or (A and B and C);
11	Y <= (not A and not B and C) or (A and not B and C)
12	or (A and B and not C);
13	Z <= (not A and not B and not C) or (not A and B and not C)
14	or (not A and B and C);
15	**end** a_log;

$$X = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}BC + ABC$$

$$Y = \bar{A}\bar{B}C + A\bar{B}C + AB\bar{C}$$

$$Z = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC$$

Asignación de Señales

Asignaciones de Señales por Selección – La construcción **with-select-when**

a(1)	a(0)	C
0	0	1
0	1	0
1	0	1
1	1	1

L	Ejemplo Nº 5 – Uso de la construcción **with-select-when**
1	**library** ieee;
2	**use** ieee.std_logic_1164.**all**;
3	**entity** circuito **is**
4	**port** (a: **in** std_logic_vector (1 **downto** 0);
5	C: **out** std_logic);
6	**end** circuito;
7	**architecture** arq_cir **of** circuito **is**
8	**begin**
9	**with** a **select**
10	C <= ‘1’ **when** “00”,
11	‘0’ **when** “01”,
12	‘1’ **when** “10”,
13	‘1’ **when others**;
14	**end** arq_cir;

Únicamente, se utiliza la *coma* (**,**), el *punto y coma* (**;**) se utiliza cuando se finaliza la construcción ***with-select***

- La estructura **with-select-when** se utiliza para asignar un valor (de varios posibles) a una señal o grupo de señales con base a los diferentes valores de otra señal o grupo de señales previamente seleccionada(o).
- Por lo general, un grupo de señales forman un vector, como en el ejemplo descrito *a(1)* y *a(0)* forman el vector *a*.

Asignación de Señales

Asignaciones de Señales por Selección – La construcción **with-select-when**

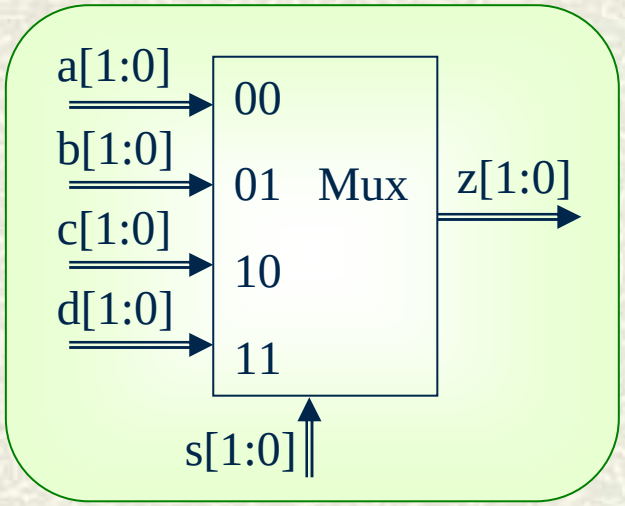
X3	X2	X1	X0	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

L | Ejemplo Nº 6 – Uso de la construcción **with-select-when**
Circuito Combinatorio que detecte Números Primos de 4-Bits

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  entity seleccion is
4          port (X: in std_logic_vector (3 downto 0);
5                  F: out std_logic);
6  end seleccion;
7  architecture a_selec of seleccion is
8  begin
9          with X select
10                 F <= '1' when "0001",
11                         '1' when "0010",
12                         '1' when "0011",
13                         '1' when "0101",
14                         '1' when "0111",
15                         '1' when "1011",
16                          '1' when "1101",
17                         '0' when others;
18 end a_selec;
```

Asignación de Señales

Multiplexores: Mux 4 a 1 (2-Bits)

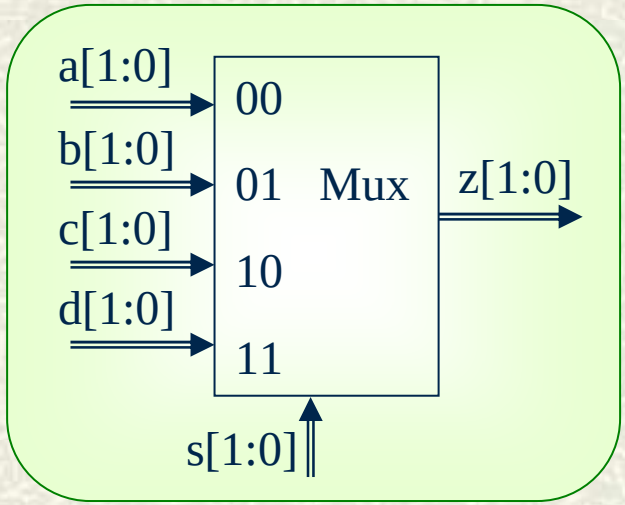


Ejemplo Nº 7 – Multiplexor 4 a 1 / Uso de **with-select-when**

```
library ieee;
use ieee.std_logic_1164.all;
entity mux is
        port (a, b, c, d: in std_logic_vector (1 downto 0);
                s: in std_logic_vector (1 downto 0);
                z: out std_logic_vector (1 downto 0));
end mux;
architecture arqmux of mux is
begin
with s select
        z <= a when “00”,
                b when “01”,
                c when “10”,
                d when others;
end arqmux;
```

Asignación de Señales

Multiplexores:Mux 4 a 1 (2-Bits)

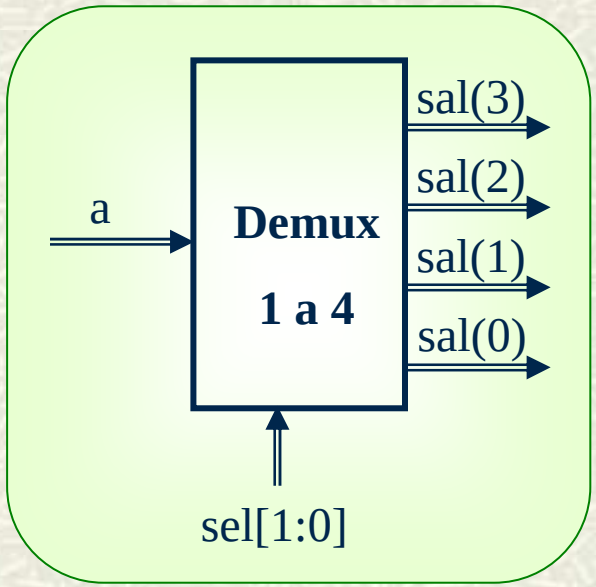


Ejemplo Nº 8 – Multiplexor 4 a 1 / Uso de Ecs. Booleanas

```
library ieee;
use ieee.std_logic_1164.all;
entity mux_eb is
        port (a, b, c, d: in std_logic_vector (1 downto 0);
                s: in std_logic_vector (1 downto 0);
                z: out std_logic_vector (1 downto 0));
end mux_eb;
architecture arqmux_eb of mux_eb is
begin
        z(1) <= (a(1) and not s(1) and not s(0)) or
                (b(1) and not s(1) and s(0)) or
                (c(1) and s(1) and not s(0)) or
                (d(1) and s(1) and s(0));
        z(0) <= (a(0) and not s(1) and not s(0)) or
                (b(0) and not s(1) and s(0)) or
                (c(0) and s(1) and not s(0)) or
                (d(0) and s(1) and s(0));
end arqmux_eb;
```

Asignación de Señales

Demultiplexor 1 a 4



Ejemplo Nº 9 – Demultiplexor 1 a 4 / Uso de **with-select-when**

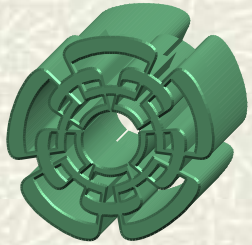
```
library ieee;
use ieee.std_logic_1164.all;
entity demux is
        port (a: in std_logic;
                sel: in std_logic_vector (1 downto 0);
                sal: out std_logic_vector (3 downto 0));
end demux;
architecture arqdemux_eb of demux is
begin
      with sel select
          sal <= "000"&a     when "00",
                 "00"&a'0'  when "01",
                 '0'&a"00"  when "10",
                 a&"000"    when "11",
                 "0000"     when others;
end arqdemux;
```

Procesos (process)

Tipos de Enunciados Concurrentes.	
Asignación de Señal	Permite asignar un valor calculado a una señal o puerto.
Proceso (**process**)	Permite definir un algoritmo secuencial que lee valores de Señales y calcula nuevos valores que son asignados a otras Señales.
Bloque (**block**)	Grupo de enunciados concurrentes.
Llamada a un Componente predefinido	
Llamada a un Procedimiento (**procedure**)	Llama a un algoritmo que calcula y asigna valores a Señales

Proceso (process)

- Cada proceso es conformado por un conjunto de enunciados secuenciales.
- Enunciados Secuenciales ➔ Son interpretados por la herramienta de síntesis en forma secuencial, es decir, uno por uno; por lo que el orden en el cual son declarados tiene un efecto significativo en la lógica que se intenta describir o sintetizar.



Procesos (process)

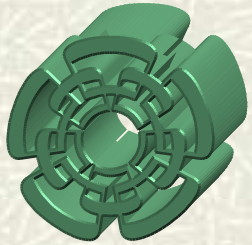
Proceso (process)

Enunciados Secuenciales

- Enunciados de Asignación de Variables
- Enunciados de Asignación de Señales
- Enunciados ***if***
- Enunciados ***case***
- Enunciados ***loop***
- Enunciados ***next***
- Enunciados ***exit***
- Enunciados de Subprogramas
- Enunciados ***return***
- Enunciados ***wait***
- Enunciados ***null***

Nota importante:

Una ***señal*** que se vea involucrada dentro de un proceso no recibe inmediatamente el valor asignado, sólo hasta el final del mismo. Una ***variable*** que sea utilizada dentro de un proceso sí recibe el valor de forma inmediata.



Formato del enunciado IF-THEN-ELSE

Enunciados if:

- La construcción **if-then-else**

if *la_condición_es_cierta* **then**

{ejecuta grupo-1 de enunciados secuenciales};

else

{ejecuta grupo-2 de enunciados secuenciales};

end if;

Enunciados if:

- La construcción **if-then-elsif-then-else**

if *la_condición-1_se_cumple* **then**

{ejecuta grupo-1 de enunciados secuenciales};

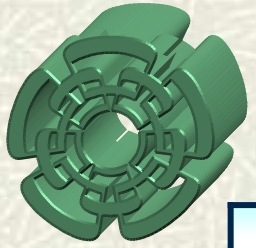
elsif *la_condición-2_se_cumple* **then**

{ejecuta grupo-2 de enunciados secuenciales};

else

{ejecuta grupo-3 de enunciados secuenciales};

end if;



Operadores Relacionales

Operadores Relacionales

Características.

- Uso: Para fines de comparación de datos.
- Operadores incluidos en los paquetes: std_numeric y std_logic_arith
- Los operadores de Igualdad y Desigualdad (= , /=) utilizan todos los tipos de datos.
- Los operadores (<, <=, >, >=) son definidos para los tipos escalar y arreglos unidimensionales de tipos de datos enumerados o enteros.

Operador	Significado
=	Igual
/=	Diferente
<	Menor
<=	Menor o Igual
>	Mayor
>=	Mayor o Igual

Enunciado IF-THEN-ELSE

La construcción: **if-then-else**

La construcción if-then-else sirve para seleccionar una operación con base al análisis (evaluación lógica ➔ Cierto o Falso) de una condición.

L	Ejemplo Nº 10 - La construcción **if-then-else** *Comparador de dos palabras con long. de 2-bits*
1	**library** ieee;
2	**use** ieee.std_logic_1164.**all**;
3	**entity** comp **is**
4	**port** (a,b: **in** std_logic_vector (1 downto 0);
5	c: **out** std_logic);
6	**end** comp;
7	**architecture** funcional **of** comp **is**
8	**begin**
9	compara: **process** (a,b)
10	**begin**
11	**if** a = b **then**
12	c <= ‘1’;
13	**else**
14	c <=‘0’;
15	**end if**;
16	**end process** compara;
17	**end** funcional;

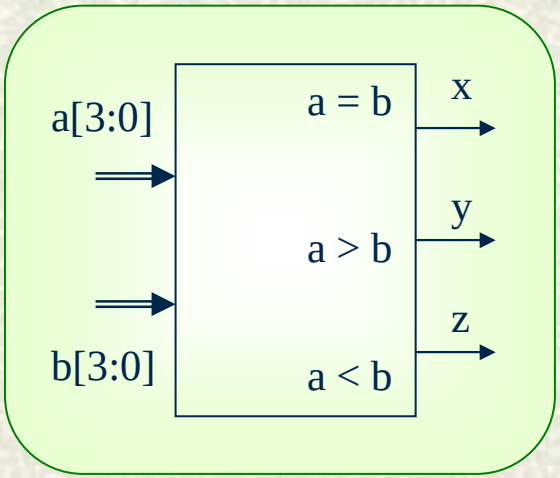
Lista-Sensitiva

Señales (incluyendo puertos) leídas por el proceso.

Enunciado IF-THEN-ELSIF-THEN-ELSE

La construcción:

if-then-elsif-then-else



¿Qué valores tienen las otras salidas en este instante?

La construcción **if-then-elsif-then-else** se utiliza cuando se requiere analizar más de una condición de entrada.

L	Ejemplo Nº 11 - La construcción **if-then-elsif-then-else** *Comparador de Magnitud 2-Words de 4-bits*
1	**library** ieee;
2	**use** ieee.std_logic_1164.**all**;
3	**entity** comp4 **is**
4	**port** (a,b: **in** std_logic_vector (3 downto 0);
5	x,y,z: **out** std_logic);
6	**end** comp4;
7	**architecture** arq_comp4 **of** comp4 **is**
8	**begin**
9	**process** (a,b)
10	**begin**
11	**if** (a = b) **then**
12	x <= ‘1’;
13	**elsif** (a > b) **then**
14	y <=‘1’;
15	**else**
16	z <=‘1’;
17	**end if**;
18	**end process**;
19	**end** arq_comp4;

¿Qué circuito es inferido?

Enunciado IF-THEN-ELSIF-THEN-ELSE

A1	A0	B1	B0	Z1	Z0
0	0	0	0	1	1
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	1	1
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	0
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	1	1

Ejemplo Nº 12 - La construcción **if-then-elsif-then-else**

Comparador de Magnitud 2-Words de 2-Bits / Salida Codificada

```
library ieee;
use ieee.std_logic_1164.all;
entity comp is
        port (A,B: in std_logic_vector (1 downto 0);
                Z: out std_logic_vector (1 downto 0));
end comp;
architecture a_comp of comp is
begin
process (A,B)
begin
        if (A = B) then
                Z <= "11";
        elsif (A < B) then
                Z <= "01";
        else
                Z <= "10";
        end if;
end process;
end a_comp;
```

Operación deseada:

Si: A = B entonces Z = 11

Si: A < B entonces Z = 01

Si: A > B entonces Z = 10

Ecs. Booleanas:

$$Z1 = A0A1 + \overline{B0}\,\overline{B1} + A1\overline{B0} + A1\overline{B1} + A0\overline{B1}$$

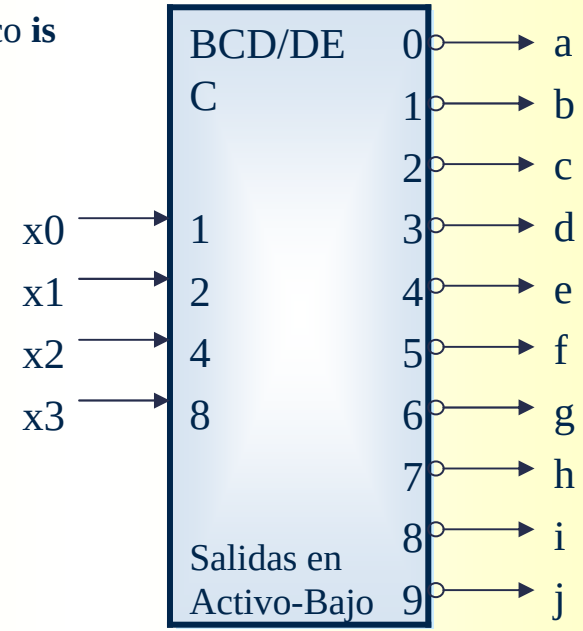
$$Z0 = \overline{A0}\,\overline{A1} + B0B1 + \overline{A0}B1 + \overline{A1}B1 + \overline{A1}B0$$

Enunciado IF-THEN-ELSIF-THEN-ELSE

Decodificadores: BCD a Decimal

Ejemplo Nº 13 – Decodificador de BCD a Decimal

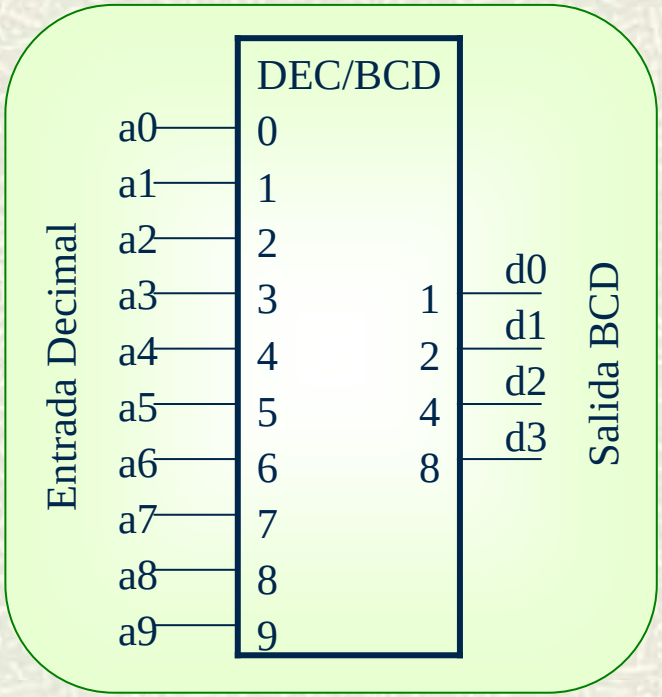
```
library ieee;
use ieee.std_logic_1164.all;
entity deco is
        port (x: in std_logic_vector (3 downto 0);
                a, b, c, d, e, f, g, h, i, j: out std_logic);
end deco;
architecture arqdeco of deco is
begin
process (x) begin
        a <= '1';
        b <= '1';
        c <= '1';
        d <= '1';
        e <= '1';
        f <= '1';
        g <= '1';
        h <= '1';
        i <= '1';
        j <= '1';
```




```
if  x = "0000" then
        a <= '0';
elsif  x = "0001" then
        b <= '0';
elsif  x = "0010" then
        c <= '0';
elsif  x = "0011" then
        d <= '0';
elsif  x = "0100" then
        e <= '0';
elsif  x = "0101" then
        f <= '0';
elsif  x = "0110" then
        g <= '0';
elsif  x = "0111" then
        h <= '0';
elsif  x = "1000" then
        i <= '0';
else j <= '0';
end if;
end process;
end arqdeco;
```

Enunciado IF-THEN-ELSIF-THEN-ELSE

Codificadores: Decimal a BCD



Ejemplo Nº 14 – Codificador Decimal a BCD

```
library ieee;
use ieee.std_logic_1164.all;
entity codif is
        port (a: in std_logic_vector (9 downto 0);
                d: out std_logic_vector (3 downto 0));
end codif;
architecture arqcodif of codif is
begin
process (a)
begin
        if      a = "0000000001" then d <= "0000";
        elsif a = "0000000010" then d <= "0001";
        elsif a = "0000000100" then d <= "0010";
        elsif a = "0000001000" then d <= "0011";
        elsif a = "0000010000" then d <= "0100";
        elsif a = "0000100000" then d <= "0101";
        elsif a = "0001000000" then d <= "0110";
        elsif a = "0010000000" then d <= "0111";
        elsif a = "0100000000" then d <= "1000";
        elsif a= "1000000000"  then d <= "1001";
        else d <= "1111";
        end if;
end process;
end arqcodif;
```

Formato del enunciado CASE

Enunciados CASE:

• La construcción **case** - **when** ejecuta una o varias instrucciones secuenciales que dependen del valor de una sola expresión.

SINTAXIS

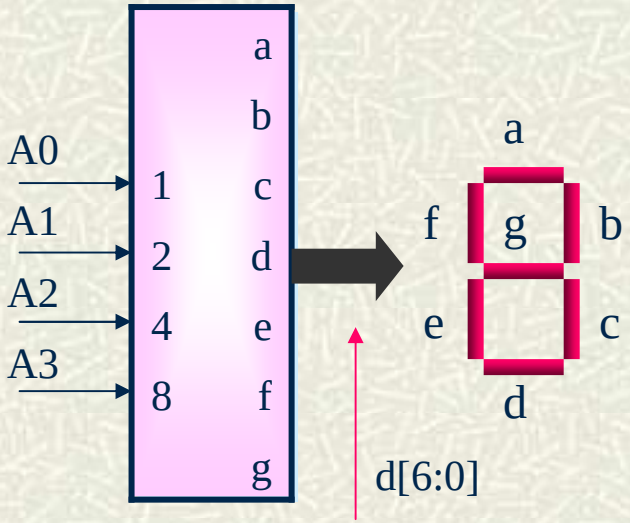
case *expresion* **is**
when *caso* **=>** *enunciados secuenciales;*
{**when** *caso* **=>** *enunciados secuenciales;* }
[**when others =>** *enunciados secuenciales;*]
end case;

EJEMPLO:

```
case puntuacion of
        when 9 to 10          => acta <="Sobresaliente";
        when 8 downto 7       => acta <="Notable";
        when 5  |  6          => acta <="Aprobado";
        when 0                => acta <="No presentado";
        when others => acta <="Suspenso";
end case;
```

Enunciado CASE

Decodificadores: BCD a 7-Segmentos



Salidas en Activo-Bajo

Código BCD (A)				Segmentos del Display (d)						
A3	A2	A1	A0	d6	d5	d4	d3	d2	d1	d0
				a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	1	1	1	1
0	0	1	0	0	0	1	0	0	1	0
0	0	1	1	0	0	0	0	1	1	0
0	1	0	0	1	0	0	1	1	0	0
0	1	0	1	0	1	0	0	1	0	0
0	1	1	0	0	1	0	0	0	0	0
0	1	1	1	0	0	0	1	1	1	0
1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	1	0	0

Enunciado CASE

Decodificadores: BCD a 7-Segmentos

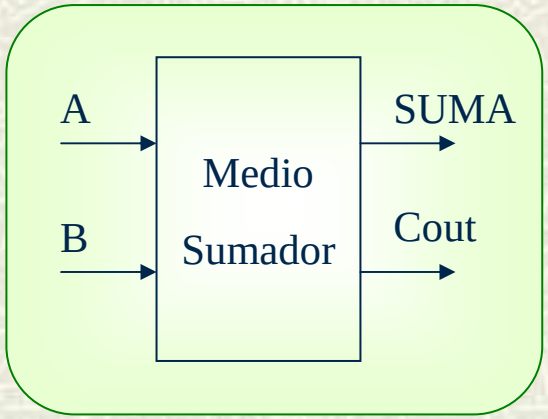
Ejemplo Nº 15 – Decodificador
BCD a 7-Segmnetos
(Uso de construcción **case-when**)

```
library ieee;
use ieee.std_logic_1164.all;
entity decobcd_7s is
        port (A: in std_logic_vector (3 downto 0);
                d: out std_logic_vector (6 downto 0));
end decobcd_7s;
architecture arqdeco of decobcd_7s is
begin
process (A) begin
        case A is
        when "0000" => d <= "0000001";
        when "0001" => d <= "1001111";
        when "0010" => d <= "0010010";
        when "0011" => d <= "0000110";
        when "0100" => d <= "1001100";
        when "0101" => d <= "0100100";
        when "0110" => d <= "0100000";
        when "0111" => d <= "0001110";
        when "1000" => d <= "0000000";
        when "1001" => d <= "0000100";
        when others => d <= "1111111";
        end case;
end process;
end arqdeco;
```

Construcción **case-when**: En esta construcción se evalua la expresión especificada (**case**) y el valor que se obtenga se compara con los asociados a las diferentes opciones descritas. Aquella opción (**when**) que coincida con dicho valor, le serán ejecutados sus enunciados secuenciales adyancentes.

Otras Estructuras Básicas

Sumadores: Medio Sumador



A	B	Suma	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

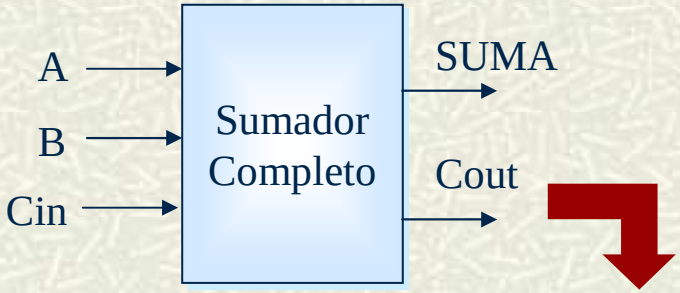
Ejemplo Nº 16 – Medio Sumador

```
library ieee;
use ieee.std_logic_1164.all;
entity m_sum is
        port (A,B: in std_logic;
                SUMA, Cout: out std_logic);
end m_sum;
architecture am_sum of m_sum is
begin
        SUMA <= A xor B;
        Cout <= A and B;
end am_sum;
```

$$Suma = A \oplus B$$
$$Cout = AB$$

Otras Estructuras Básicas

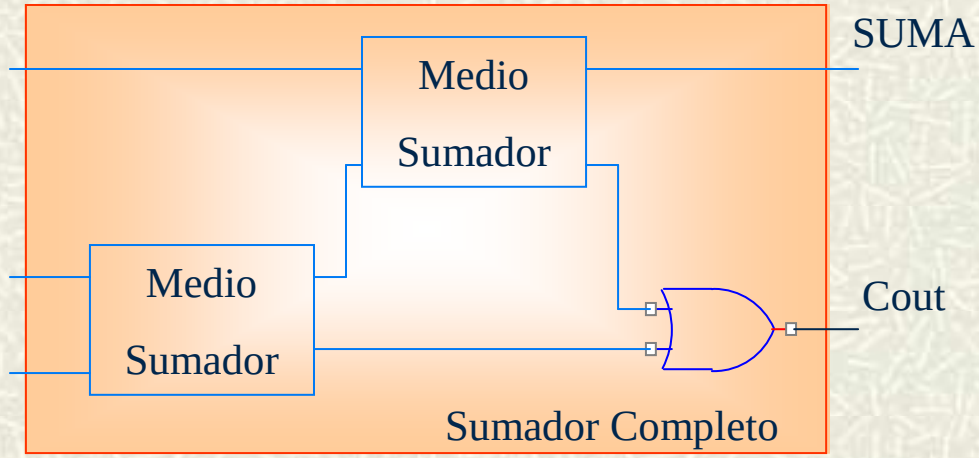
Sumadores: Sumador Completo

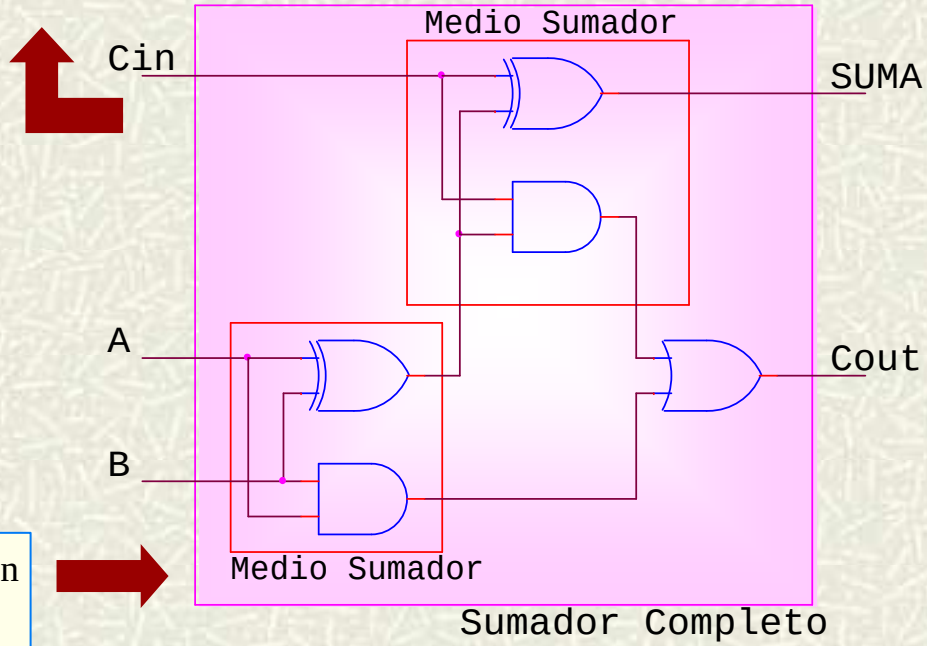


A	B	Cin	Suma	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$\text{Suma} = \overline{A}\,\overline{B}\,\overline{Cin} + \overline{A}\,B\overline{Cin} + A\overline{B}\,\overline{Cin} + ABCin = A \oplus B \oplus Cin$$

$$\text{Cout} = AB + BCin + ACin = AB + (A \oplus B)Cin$$





Otras Estructuras Básicas

Sumadores: Sumador Completo

Ejemplo Nº 17 – Sumador Completo

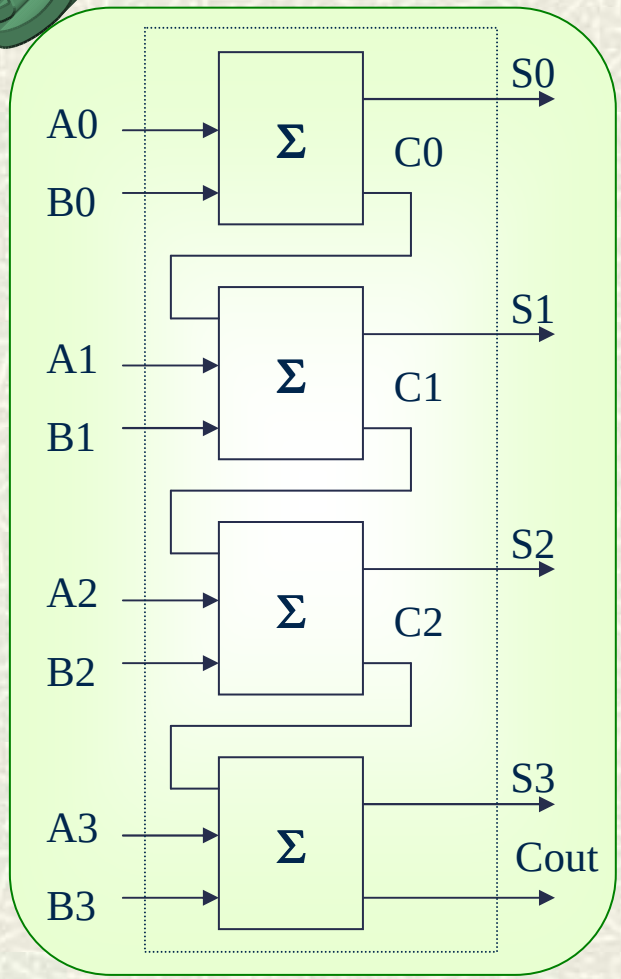
```
library ieee;
use ieee.std_logic_1164.all;
entity sum is
        port (A, B, Cin: in std_logic;
                Suma, Cout: out std_logic);
end sum;
architecture a_sum of sum is
begin
        Suma <= A xor B xor Cin;
        Cout <= (A and B) or ((A xor B) and Cin);
end a_sum;
```

$$Suma = \overline{A}B\overline{Cin} + \overline{A}B\overline{Cin} + A\overline{B}\overline{Cin} + ABCin = A \oplus B \oplus Cin$$

$$Cout = AB + BCin + ACin = AB + (A \oplus B)Cin$$

Otras Estructuras Básicas

Sumadores: Sumador Paralelo 4-Bits



Declaraciones de Señales (**signal**): Especifican señales que permiten conectar los diferentes tipos de enunciados concurrentes (*asignación de señales, bloques, procesos y llamadas a componentes o procedimientos*) de que consta una arquitectura.

Ejemplo Nº 18 – Sumador Paralelo 4-Bits

```
library ieee;
use ieee.std_logic_1164.all;
entity suma is
        port (A, B: in std_logic_vector (3 downto 0);
                S: out std_logic_vector (3 downto 0);
                Cout: out std_logic);
end suma;
architecture arqsuma of suma is
signal C: std_logic_vector (2 downto 0);
begin
        S(0) <= A(0) xor B(0);
        C(0) <= A(0) and B(0);
        S(1) <= (A(1) xor B(1)) xor C(0);
        C(1) <= (A(1) and B(1)) or (C(0) and (A(1) xor B(1)));
        S(2) <= (A(2) xor B(2)) xor C(1);
        C(2) <= (A(2) and B(2)) or (C(1) and (A(2) xor B(2)));
        S(3) <= (A(3) xor B(3)) xor C(2);
        Cout <= (A(3) and B(3)) or (C(2) and (A(3) xor B(3)));
end arqsuma;
```

Otras Estructuras Básicas

Sumadores: Sumador Paralelo 4-Bits

Operadores Aritméticos	
Operador	Descripción
+	Suma
-	Resta
/	División
*	Multiplicación
**	Potencia

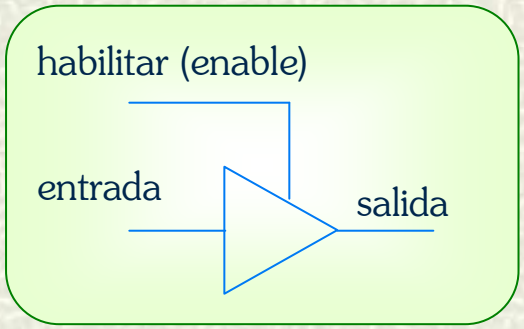
Ejemplo Nº 19 – Sumador Paralelo 4-Bits con Cout (Uso Operador Aritmético '+')

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity sum4b_arit is
        port (A, B: in std_logic_vector (3 downto 0);
                S: out std_logic_vector (3 downto 0)
                Cout: out std_logic );
end sum4b_arit;
architecture arqsum of sum4b_arit is
signal sum: std_logic_vector (4 downto 0);
begin
        sum <= '0'&A + '0'&B;
        S <= sum (3 downto 0);
        Cout <= sum(4);
end arqsum;
```

Otras Estructuras Básicas

Buffer-Salida de 3-Estados



Tipos Lógicos Estándares	
‘U’	Valor No-Inicializado
‘X’	Valor Fuerte Desconocido
‘0’	0 Fuerte
‘1’	1 Fuerte
‘Z’	Alta Impedancia
‘W’	Valor Débil Desconocido
‘L’	0 Débil
‘H’	1 Débil
‘-’	No Importa (Don’t Care)

Ejemplo Nº 20 – Buffer Salida de 3-Estados

```
library ieee;
use ieee.std_logic_1164.all;
entity tri_est is
        port (enable, entrada: in std_logic;
                salida: out std_logic);
end tri_est;
architecture arq_buffer of tri_est is
begin
process (enable, entrada)
begin
        if (enable = ‘0’) then
                salida <= ‘Z’;
        else
                salida <= entrada;
        end if;
end process;
end arq_buffer;
```

El tipo de dato ***bit*** no soporta el valor ‘Z’, por lo que se debe utilizar el tipo ***std_logic***, que si lo soporta.

Resumen de Circuitos combinatorios

Multiplexores

```
with sel select
  dout <= a when “00”,
          b when “01”,
          c when “10”,
          d when “11”,
          (others => ‘x’) when others;
```

```
process(sel, a, b, c, d)
begin
  case sel is
    when “00”   => dout <= a;
    when “01”   => dout <= b;
    when “10”   => dout <= c;
    when “11”   => dout <= d;
    when others => dout <= (others => ‘X’);
  end case;
end process;
```

```
process(sel, a, b, c, d)
begin
  if (sel = “00”) then
    dout <= a;
  elsif (sel = “01”) then
    dout <= b;
  elsif (sel = “10”) then
    dout <= c;
  elsif (sel = “11”) then
    dout <= d;
  else
    dout <= (others => ‘X’);
  end if;
end process;
```

```
dout <= a when sel = “00” else
        b when sel = “01” else
        c when sel = “10” else
        d when sel = “11” else
        (others => ‘x’);
```