



Capítulo.3 - Diseño Lógico Combinacional con *VHDL*

Diseño (Programación) de una Estructura Básica Combinatoria**Biblioteca (s)****Declaración
*Entidad*****Declaración
*Arquitectura*****Sintaxis:**

```
ARCHITECTURE nombre_arquitectura OF nombre_entidad IS  
    { Declarativas de Bloque } –Se analizarán posteriormente  
BEGIN  
    { Enunciados Concurrentes }  
END [nombre_arquitectura]
```

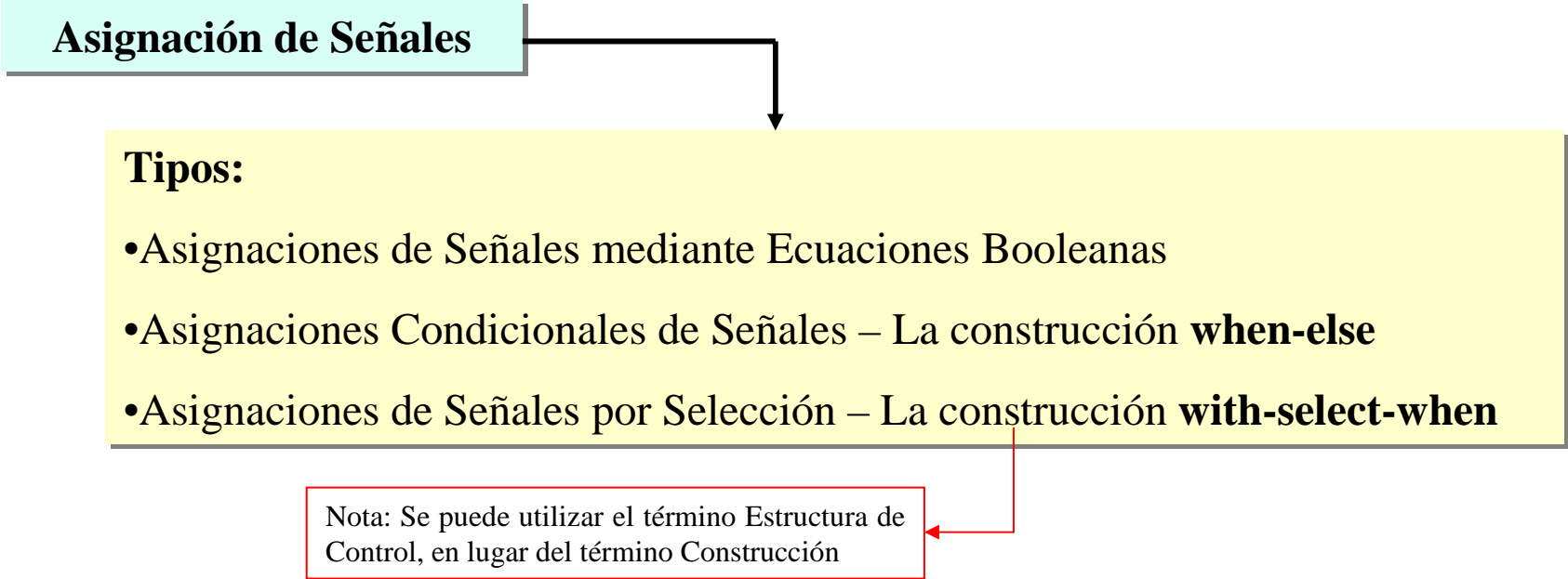
Enunciado Concurrente.

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.

<i>Tipos de Enunciados Concurrentes.</i>	
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 de Señales mediante Ecuaciones Booleanas
- Asignaciones Condicionales de Señales – La construcción **when-else**
- Asignaciones de Señales por Selección – La construcción **with-select-when**

Nota: Se puede utilizar el término Estructura de Control, en lugar del término Construcción

SEÑALES Y VARIABLES

VARIABLES:

Es similar al concepto de variable en otros lenguajes. Su valor puede ser alterado en cualquier instante y se le puede asignar un valor inicial. Las variables sólo se declaran en los procesos o subprogramas.

Utilizadas en ejecuciones en serie.

SEÑALES:

Se declaran igual que las constantes y variables. La diferencia es que pueden ser *normal*, *register* y *bus*. Si no se especifica nada en la declaración el compilador entenderá que es del tipo *normal*. Se puede decir que la señal tiene dos partes una donde se escribe y otra donde se lee. Las señales pueden ser declaradas sólo en las arquitecturas, paquetes (PACKAGE) o en bloques concurrentes (BLOCK).

Utilizadas en ejecuciones concurrentes.

SEÑALES Y VARIABLES

--Uso incorrecto de las señales

```
ARCHITECTURE ejem1 OF entidad IS  
SIGNAL a, b, c, x, y : INTEGER;  
BEGIN  
  P1: PROCESS (a,b,c)  
  BEGIN  
    c<= a; --Se ignora  
    x<=c+2;  
    c<=b; --Se mantiene  
    y<=c+2;  
  END PROCESS p1;  
END ejem1
```

--Uso correcto de las señales

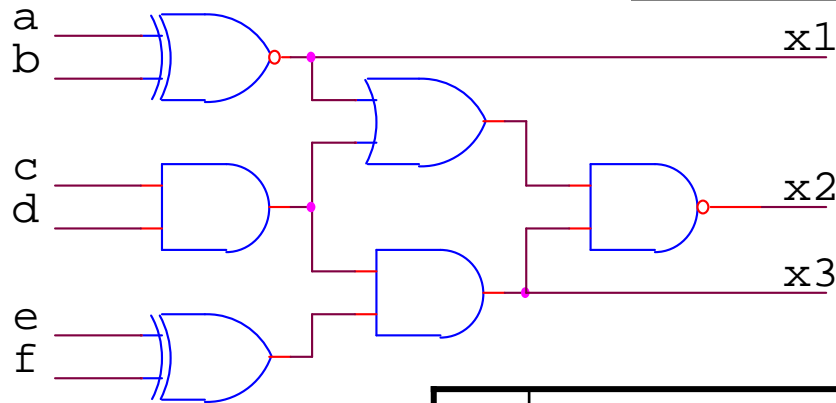
```
ARCHITECTURE ejem1 OF entidad IS  
SIGNAL a, b, x, y : INTEGER;  
BEGIN  
  P1: PROCESS (a,b)  
  VARIABLE c: INTEGER;  
  BEGIN  
    c:= a; --Inmediato  
    x<=c+2;  
    c:=b; --Inmediato  
    y<=c+2;  
  END PROCESS p1;  
END ejem1
```

Operadores Lógicos
and, or, nand, xor, xnor, not
Tipos de Operandos permisibles: <i>bit, boolean y arreglos unidimensionales</i> (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
$q = a + (x \cdot y)$	<code>q = a or (x and y)</code>
$y = a + (\bar{b} \cdot \bar{c}) + d$	<code>y = a or (not b and not c) or d</code>

Asignaciones de Señales mediante Ecuaciones Booleanas



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

L	Ejemplo N° 1 – 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;

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



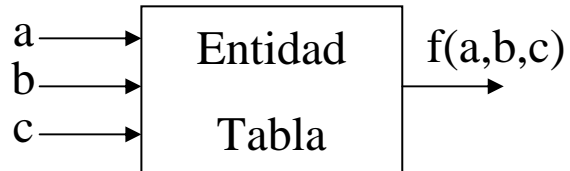
$$X = \overline{A}\overline{B}\overline{C} + \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\overline{C}$$

$$Y = \overline{A}\overline{B}C + A\overline{B}C + A\overline{B}\overline{C}$$

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



L	Ejemplo N° 2 – 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;

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 N° 3 - 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, es necesario declarar los valores que se le deben asignar a una determinada 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.

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

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

L	Ejemplo N° 4 - 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;

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 N° 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*.

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 N° 6 – Uso de la construcción with-select-when <i>Circuito Combinatorio que detecte Números Primos de 4-Bits</i>
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;

Asignaciones de Señales mediante Ecuaciones Booleanas

--Ejemplo 7. Multiplexor de 4 a 1

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

-----

ENTITY mux IS
  PORT ( a, b, c, d, s0, s1: IN  STD_LOGIC;
         y: OUT STD_LOGIC);
END mux;

-----

ARCHITECTURE ecu_booleana OF mux IS
BEGIN
  y <= ( a AND NOT s1 AND NOT s0) OR
        (b AND NOT s1 AND s0) OR
        (c AND s1 AND NOT s0) OR
        (d AND s1 AND s0);
END ecu_booleana;
```

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

--Ejemplo 8. Multiplexor de 4 a 1

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

-----

ENTITY mux IS
  PORT ( a, b, c, d: IN   STD_LOGIC;
         sel: IN   STD_LOGIC_VECTOR (1 DOWNT0 0);
         y: OUT STD_LOGIC);
END mux;

-----

ARCHITECTURE mux1 OF mux IS
BEGIN
  y <= a WHEN sel="00" ELSE
      b WHEN sel="01" ELSE
      c WHEN sel="10" ELSE
      d;
END mux1;
```

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

--Ejemplo 9. Multiplexor de 4 a 1

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

-----

ENTITY mux IS
  PORT ( a, b, c, d: IN   STD_LOGIC;
         sel: IN   STD_LOGIC_VECTOR (1 DOWNT0 0);
         y: OUT STD_LOGIC);
END mux;

-----

ARCHITECTURE mux2 OF mux IS
BEGIN
  WITH sel SELECT
    y <= a WHEN "00",      --Notar que se pone “,” en lugar de “;”
         b WHEN "01",
         c WHEN "10",
         d WHEN OTHERS;    --No es: “[d WHEN “11”]”.
END mux2;
```

Asignaciones de Señales por Selección – Con el tipo de dato INTEGER

--Ejemplo 10. Multiplexor de 4 a 1 empleando el tipo "INTEGER".

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;
```

```
-----  
ENTITY mux IS  
  PORT ( a, b, c, d: IN   STD_LOGIC;  
         sel: IN   INTEGER RANGE 0 TO 3;  
         y: OUT STD_LOGIC);  
END mux;
```

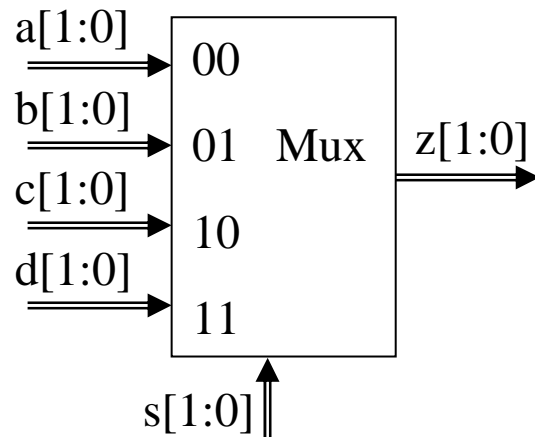
----- La construcción WHEN/ELSE

```
ARCHITECTURE mux1 OF mux IS  
BEGIN  
  y <= a WHEN sel=0 ELSE  
       b WHEN sel=1 ELSE  
       c WHEN sel=2 ELSE  
       d;  
END mux1;
```

----- La construcción WITH/SELECT/WHEN

```
ARCHITECTURE mux2 OF mux IS  
BEGIN  
  WITH sel SELECT  
  y <= a WHEN 0,      --Notar que se pone “,” en lugar de “;”  
       b WHEN 1,  
       c WHEN 2,  
       d WHEN 3;  
END mux2;
```


Multiplexores: Mux 4 a 1 (2-Bits)

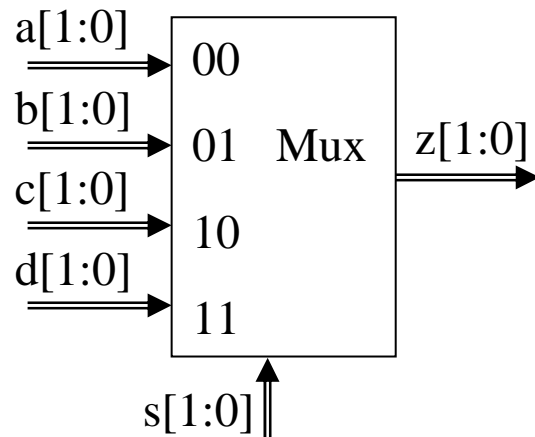


Ejemplo N° 11 – Multiplexor 4 a 1 / Uso de Ecs. Booleanas

```

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;
  
```

Multiplexores: Mux 4 a 1 (2-Bits)

Ejemplo N° 12 – 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;
```

<i>Tipos de Enunciados Concurrentes.</i>	
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 ➔ Éstos 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.

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.

Enunciados if:

- La construcción **if-then-else**

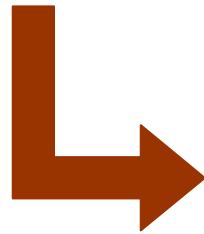
**if** *la_condición_es_cierta* **then***{ejecuta grupo-1 de enunciados secuenciales};***else***{ejecuta grupo-2 de enunciados secuenciales};***end if;****Enunciados if:**

- 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;**

Enunciados CASE:

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

**SINTAXIS**

```
case expression is
    when choices => { sequential_statement }
    when choices => { sequential_statement }
end case;
```

DESCRIPCION

expression: evalúa a un entero, o tipo enumerado.

sequential_statement: uno o mas enunciados secuenciales.

choices: opciones.

La última opción puede ser *others* (valor por omisión del resto de las opciones).

Operadores Relacionales	
<u>Características.</u>	
<ul style="list-style-type: none">•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

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.

--Ejemplo 13a. Mux 2a1 empleando if/then/else.

ENTITY *mux* IS

```
PORT ( a, b, sel: IN bit;
       sal: OUT bit );
```

END *mux*;

ARCHITECTURE *comportamental* OF *mux* IS

BEGIN

PROCESS (a, b, sel) - - Lista sensível

BEGIN

IF (sel='0') THEN

```
sal <= a;
```

ELSE

```
sal <= b;
```

END IF;**END PROCESS;**

END *comportamental*;

Lista-Sensitiva

Señales (incluyendo puertos) leídas por el proceso.

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.

--Ejemplo 13b. Mux 2a1 empleando if/then/else.

ENTITY *mux* IS

```
PORT ( a, b, sel: IN bit;
       sal: OUT bit );
```

END *mux*;

ARCHITECTURE *comportamental* OF *mux* IS

BEGIN

PROCESS (a, b, sel) - - Lista sensível

BEGIN

```
sal <= a;
```

IF (sel='1') THEN

```
sal <= b;
```

END IF;**END PROCESS;**

END comportamental;

Lista-Sensitiva

Señales (incluyendo puertos) leídas por el proceso.

La construcción: case-when

La construcción **case** es similar a la instrucción **with/select/when** por que tiene una señal de selección e incluye cláusulas **when** para diversas combinaciones de señal.

--Ejemplo 14. Mux 2a1 empleando case.

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

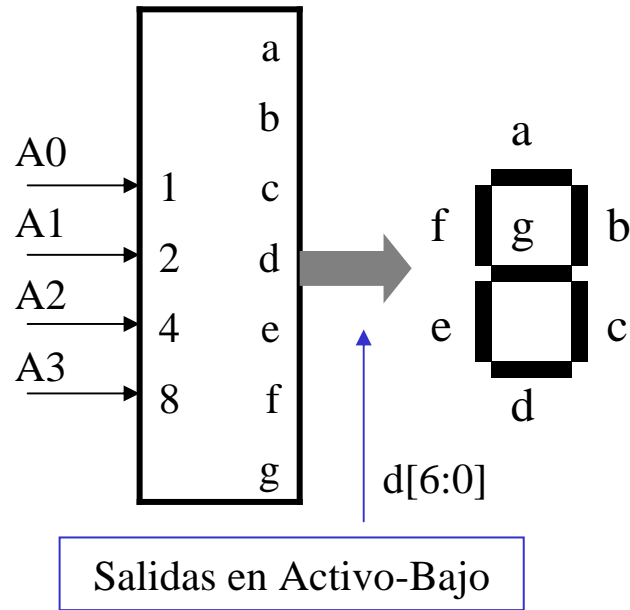
ENTITY mux IS
  PORT ( a, b, sel: IN  STD_LOGIC;
        sal: OUT STD_LOGIC );
END mux;

ARCHITECTURE comportamental OF mux IS
BEGIN
  PROCESS (a, b, sel) - - Lista sensible
  BEGIN
    CASE sel IS
      WHEN '0' =>
        sal <= a;
      WHEN OTHERS =>
        sal <= b;
    END CASE;
  END PROCESS;
END comportamental;
```

Lista-Sensitiva

Señales (incluyendo puertos) leídas por el proceso.

Ejemplo 15. Decodificador: BCD a 7-Segmentos



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

Decodificadores: BCD a 7-Segmentos

Ejemplo N° 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 evalúa 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 adyacentes.

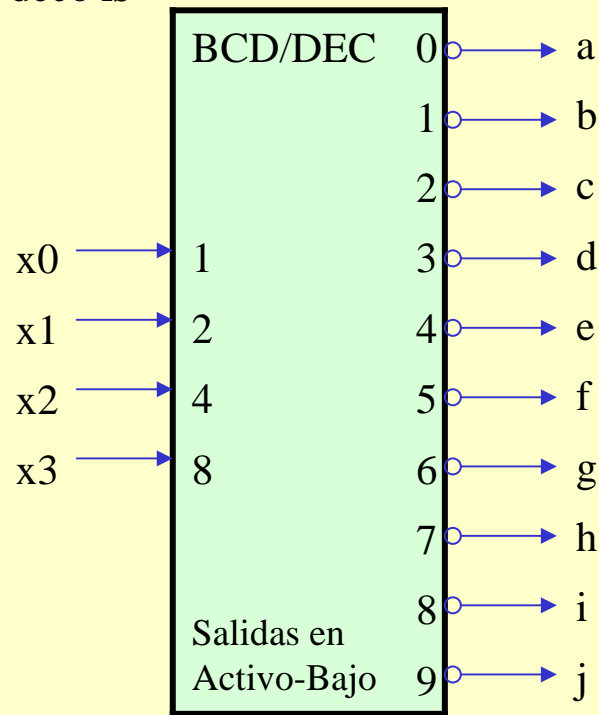
Decodificadores: BCD a Decimal

Ejemplo N° 16 – 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';
    end process;
end arqdeco;

```



```

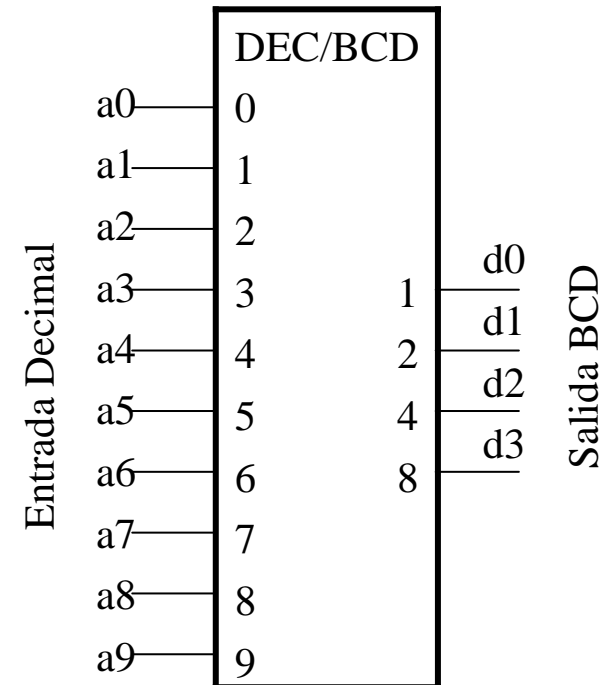
if x = "0000" then
    a <= '0';
elsif x = "0001" then
    b <= '0';
elsif x = "0010" then
    c <= '0';
elsif x = "0011" then
    d <= '0';
elsif x = "0100" then
    e <= '0';
elsif x = "0101" then
    f <= '0';
elsif x = "0110" then
    g <= '0';
elsif x = "0111" then
    h <= '0';
elsif x = "1000"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

```

Codificadores: Decimal a BCD

Ejemplo N° 17 – 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 = "0000000000" 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;
```

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.



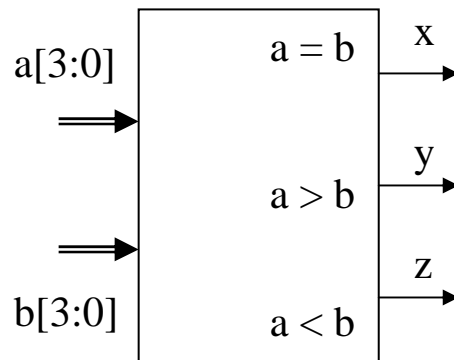
L	Ejemplo N° 18 - La construcción if-then-else <i>Comparador de dos palabras con long. de 2-bits</i>
1	library ieee;
2	use ieee.std_logic_1164.all;
3	entity comp is
4	port (a,b: in std_logic_vector (1 downto 0);
5	c: out std_logic);
6	end comp;
7	architecture funcional of comp is
8	begin
9	compara: process (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.

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

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



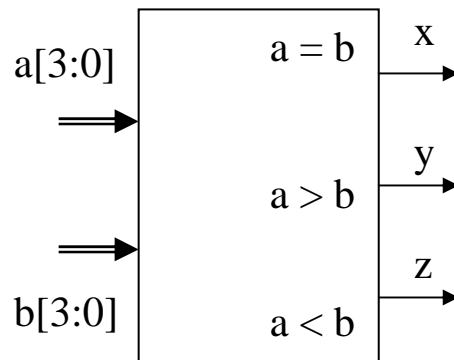
¿Qué valores tienen las otras salidas en este instante?

L	Ejemplo N° 19 - La construcción if-then-elsif-then-else <i>Comparador de Magnitud 2-Words de 4-bits</i>
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?

La construcción: **when-else**

La construcción **when-else** permite definir paso a paso el comportamiento de un sistema.



¿Qué valores tienen las otras salidas en este instante?

L	Ejemplo N° 20 - La construcción when-else <i>Comparador de Magnitud 2-palabras de 4-bits</i>
1	entity comp4 is
2	port (a,b: in bit_vector (3 downto 0);
3	x,y,z: out bit);
4	end comp4;
5	architecture arq_comp4 of comp4 is
6	begin
7	x <= '1' when a = b else
8	'0';
9	y <= '1' when a > b else
10	'0';
11	z <= '1' when a < b else
12	'0';
13	end arq_comp4;
14	
15	
16	
17	
18	
19	

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 N° 21 - 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

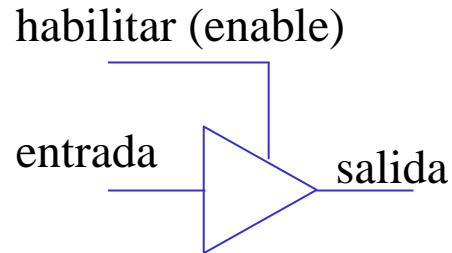
Si: A > B entonces Z = 10

Ecs. Booleanas:

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

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

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 N° 22 – Buffer Salida de 3-Estados

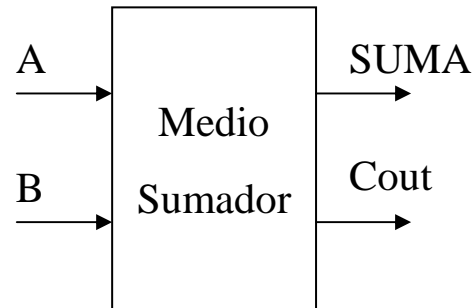
```

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.

Sumadores: Medio Sumador



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

$$Suma = A \oplus B$$

$$Cout = AB$$

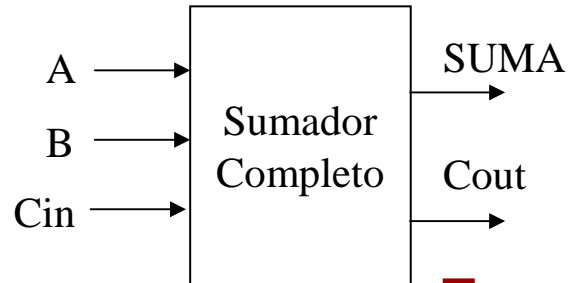
Ejemplo N° 23 – Medio Sumador

```

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;

```

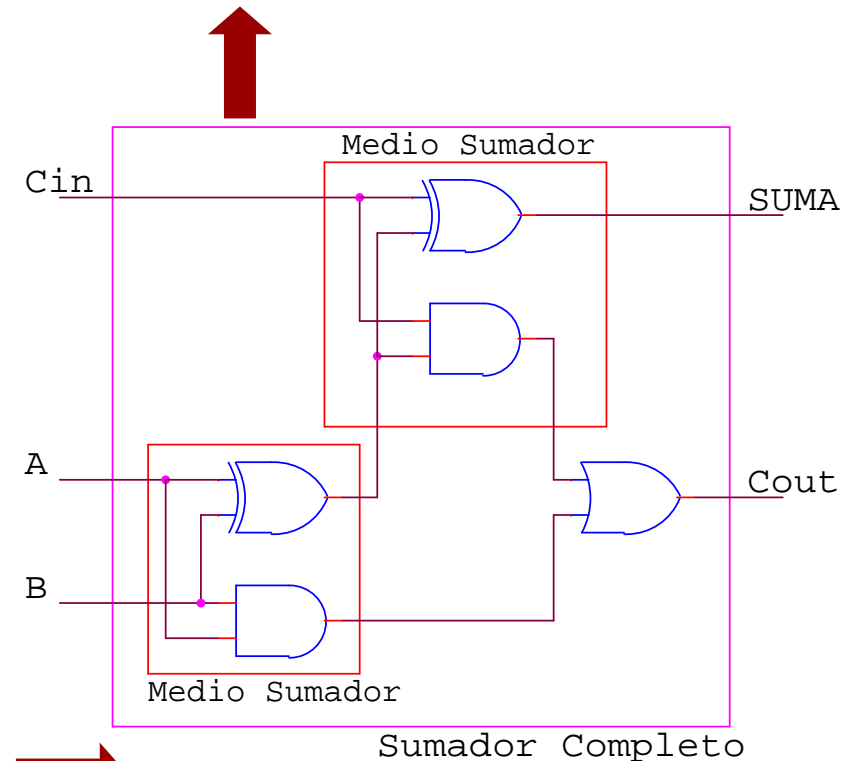
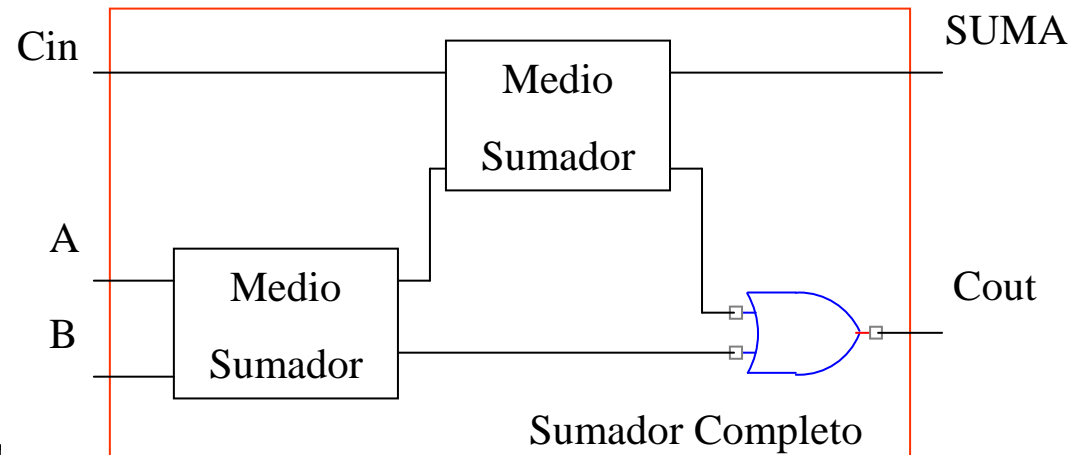
Sumadores: Sumador Completo



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

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

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



Sumadores: Sumador Completo

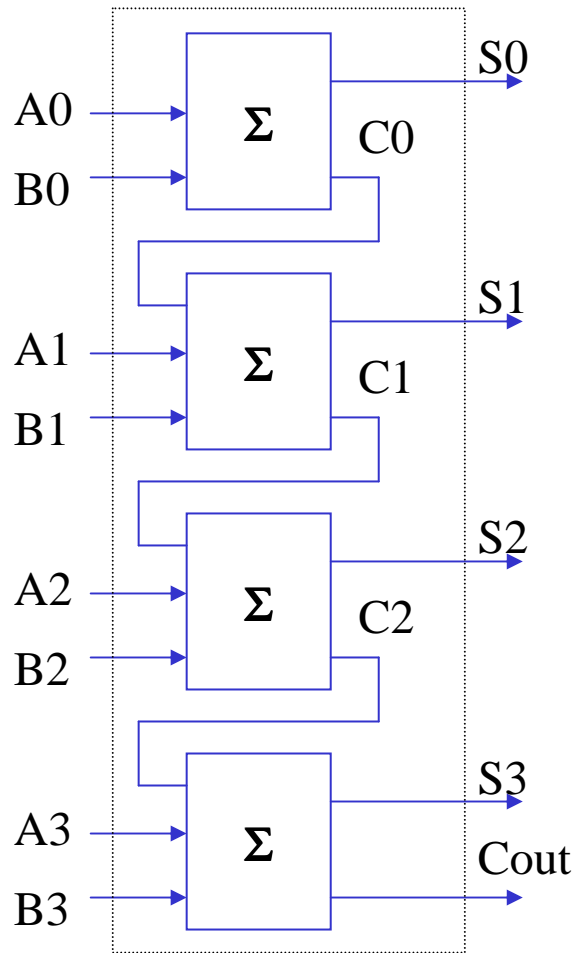
$$Suma = \overline{A}\overline{B}Cin + \overline{A}B\overline{Cin} + A\overline{B}\overline{Cin} + ABCin = A \oplus B \oplus Cin$$

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

Ejemplo N° 24 – 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;
```

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 N° 25 – 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);
    --ATTRIBUTE synthesis_off OF C: SIGNAL IS true;
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;
```

Sumadores: Sumador Paralelo 4-Bits

FOR-GENERATE: Es una instrucción para describir código concurrente jerárquico estructurado regularmente. Es obligatorio agregar una etiqueta, en este caso se llama CICLO, y se indica que se deberá hacer cuatro copias de las ecuaciones (**S(I)** y **C(I+1)**).

Label: FOR identifier IN range GENERATE
(concurrent assignments)

End GENERATE

Ejemplo N° 26 – Sumador Paralelo 4-Bits

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY suma IS
PORT(
    S : OUT STD_LOGIC_VECTOR( 3 DOWNTO 0 );
    A : IN  STD_LOGIC_VECTOR( 3 DOWNTO 0 );
    B : IN  STD_LOGIC_VECTOR( 3 DOWNTO 0 );
    C4 : OUT STD_LOGIC;
    C0 : IN  STD_LOGIC );

END suma;

ARCHITECTURE arq_suma OF suma IS
SIGNAL C : STD_LOGIC_VECTOR( 4 DOWNTO 0 );
ATTRIBUTE SYNTHESIS_OFF OF C : SIGNAL IS TRUE;
BEGIN
    C(0) <= C0;
    CICLO : FOR I IN 0 TO 3 GENERATE
        S(I) <= C(I) XOR A(I) XOR B(I);
        C(I+1) <= (B(I) AND C(I)) OR (A(I) AND C(I)) OR (A(I) AND
B(I));
    END GENERATE;
    C4 <= C(4);
END arq_suma;
```


Sumadores: Sumador Paralelo 4-Bits

FOR-LOOP: Es una instrucción parecida a la GENERATE sólo que esta instrucción es para producir una serie de instrucciones secuenciales. En el caso de GENERATE es para instrucciones concurrentes. Aquí también es para copiar cuatro veces las ecuaciones (**S(I)** y **C(I+1)**).

[label:] FOR identifier IN range LOOP
(sequential statements)
END LOOP [label]

Ejemplo N° 27 – Sumador Paralelo 4-Bits

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY suma IS PORT(
    S: OUT BIT_VECTOR( 3 DOWNT0 0 );
    A : IN  BIT_VECTOR( 3 DOWNT0 0 );
    B : IN  BIT_VECTOR( 3 DOWNT0 0 );
    C4 : OUT BIT;
    C0 : IN  BIT );
END suma;

ARCHITECTURE arq_suma OF suma IS
BEGIN
    PAU : PROCESS( A, B, C0 )
        VARIABLE C : BIT_VECTOR(4 DOWNT0 0);
        BEGIN
            C(0) := C0;
            FOR I IN 0 TO 3 LOOP
                S(I) <= C(I) XOR A(I) XOR B(I);
                C(I+1) := (B(I) AND C(I)) OR (A(I) AND C(I)) OR (A(I) AND B(I));
            END LOOP;
            C4 <= C(4);
        END PROCESS PAU;
    END arq_suma;
```

Sumadores: Sumador Paralelo 4-Bits

FOR-GENERATE: Es una instrucción para describir código concurrente jerárquico estructurado regularmente. Es obligatorio agregar una etiqueta, en este caso se llama CICLO, y se indica que se deberá hacer cuatro copias de las ecuaciones (**S(I)** y **C(I+1)**).

Label: FOR identifier IN range GENERATE
(concurrent assignments)

End GENERATE

GENERIC (parameter_name:
parameter_type := parameter_value);

Ejemplo N° 28 – Sumador Paralelo 4-Bits

```
ENTITY sumador IS
  GENERIC( N : INTEGER := 4 );
  PORT( SAL: OUT STD_LOGIC_VECTOR( N-1 DOWNT0 0 );
        A : IN  STD_LOGIC_VECTOR( N-1 DOWNT0 0 );
        B : IN  STD_LOGIC_VECTOR( N-1 DOWNT0 0 );
        C4 : OUT STD_LOGIC;
        C0 : IN  STD_LOGIC );
END sumador;

ARCHITECTURE arq_suma OF sumador IS
  SIGNAL C : STD_LOGIC_VECTOR( N DOWNT0 0 );
  SIGNAL S : STD_LOGIC_VECTOR( N-1 DOWNT0 0 );
  ATTRIBUTE SYNTHESIS_OFF OF C : SIGNAL IS TRUE;
BEGIN
  C(0) <= C0;
  CICLO : FOR I IN 0 TO N-1 GENERATE
    S(I)  <= C(I) XOR A(I) XOR B(I);
    C(I+1)<= (B(I) AND C(I)) OR (A(I) AND C(I)) OR (A(I) AND B(I));
  END GENERATE;
  C4 <= C(4);
  SAL <= S;
END arq_suma;
```

Sumadores: Sumador Paralelo 4-Bits

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

Ejemplo N° 26 – Sumador Paralelo 4-Bits sin 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);
          Suma: out std_logic_vector (3 downto 0));
end sum4b_arit;

architecture arqsum of sum4b_arit is
begin
    Suma <= A + B;
end arqsum;
```

```
#include<stdio.h>
unsigned int vbit(unsigned int num, int i,int negada) ;
int eq(unsigned int A, unsigned int B, int i);
void main()
{
    unsigned int A = 13;
    unsigned int B = 5;
    int n = 4;                /*n es el numero de bits*/
    int i,j, AmayB = 0;
    int eqp; /* es la variable que almacena el producto de los terminos eqn*/
    for ( i=0; i <= n-1; i++){ /* es la sumatoria de 0 a n-1*/
        eqp = 1;
        for(j=i+1; j<=n-1; j++) /* es la productoria de i+1 a n-1*/
            eqp*=eq(A,B,j);      /* esta funcion obtiene el termino eq en la posicion j*/
        AmayB += vbit(A,i,0) * vbit(B,i,1) * eqp; /* es A(i) *B(i) *eqn */
    }
    printf("%d", AmayB);
}
```

```
printf("%d", AmayB);
}
unsigned int vbit(unsigned int num, int i,int negada)
/*obtiene el valor de un bit en la posicion i*/
{
    unsigned int aux = 01;
    aux = aux << i;    /* pone en uno el valor del bit en la posicion i para hacer un and*/
    if ( (aux& num) > 0 ){
        if (negada == 0)
            return 1;
        else
            return 0;}
    else{
        if( negada == 0)
            return = 0;}
    return 1;
}
int eq(unsigned int A, unsigned B, int i)
{
    /*obtiene 1 si los bits A(i) y B(i) son iguales*/
    unsigned int Ai, int Bi;
    Ai = vbit(A,i,0);
    Bi = vbit(B,i,0);
    if(Ai==Bi)
        return 1;
    return 0;
}
```