Universidad de Alcalá

Departamento de Electrónica

Modelado de Sistemas Computacionales

Grado en Ingeniería de Computadores

Tema 3: Modelado para síntesis de subsistemas combinacionales en VHDL.

- **Circuitos combinacionales.**
  - Las salidas, en cada instante, dependen sólo del valor, en el mismo instante, de las entradas.
  - **No tienen memoria**.
  - **Tipos**:
    - Puertas lógicas
    - Multiplexores
    - Decodificadores
    - Codificadores
    - Comparadores
    - Sumadores/restadores
    - Multiplicadores.

Operadores Lógicos

- No tienen orden de precedencia, excepto ***not*** que tiene el mayor.
- Se asocian de izquierda a derecha si no hay paréntesis.
- Son soportados por los tipos:
  - bit
  - boolean
  - bit_vector.
  - std_logic_vectors.
  - unsigned
  - signed
- En VHDL-93 el tamaño y tipo de los vectores debe ser igual.

OPERACIÓN	OPERADOR
And	*and*
Or	*or*
Negación	*not*
Nor	*nor*
Nand	*nand*
Xor	*xor*
Xnor	*xnor*

```
signal x,y,z,t : std_logic;
signal s1,s2,s3,s4,s5,s6 : std_logic;

...
  s1 <= x and y and z;
...
  s2 <= x or y or z;
...
  s3 <= x or (y and z);
...
  s4 <= ((x xor y) or x )and y;
...
  s5 <= not x nor (y nand z);
...
  s6 <= (y or z) and x;
```

A este grupo de sentencias se llama asignación concurrente a señal

Operadores Lógicos

- Una asignación concurrente es evaluada siempre que alguna de las señales de la derecha modifiquen su valor. La señal no actualiza su valor de forma inmediata. Equivale a un proceso combinacional sensible a las señales de la derecha en la asignación.
  - Sintaxis:

```
nombre_señal <= forma_de_onda;
```

¡El orden no importa: se ejecutan en paralelo!

```
entity ejemplo is
  port(
    a, b, c, d  : in  std_logic;
    y           : out std_logic);
end entity;

architecture rtl_1 of ejemplo is
  signal s1, s2 : std_logic;
begin
  y <= s1 or s2;
  s1 <= a and c;
  s2 <= b and d and not(a);
end rtl_1;
```

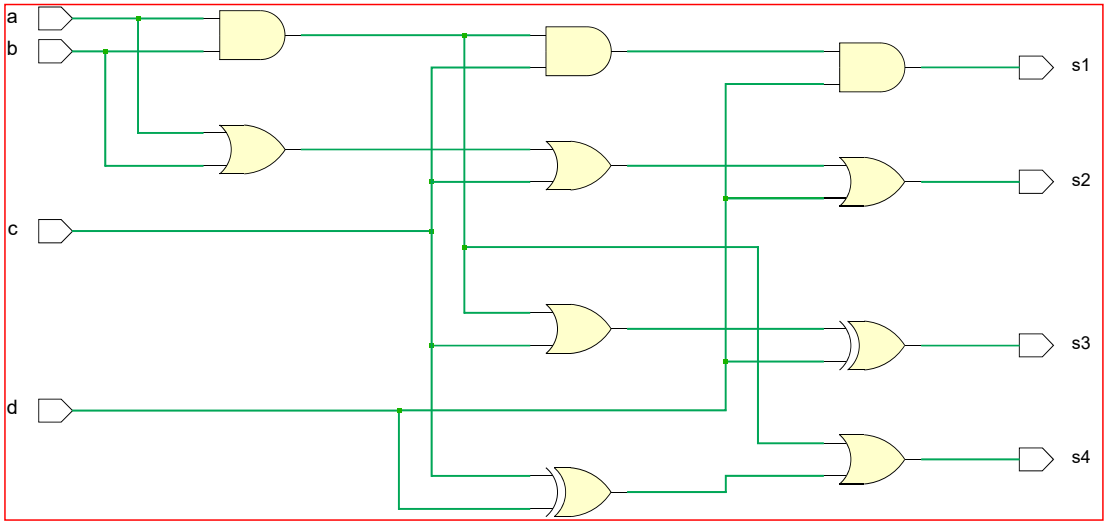
```
entity ejemplo is
  port(
    a, b, c, d  : in  std_logic;
    y           : out std_logic);
end entity;

architecture rtl_2 of ejemplo is
  signal s1, s2 : std_logic;
begin
  s1 <= a and c;
  s2 <= b and d and not(a);
  y <= s1 or s2;
end rtl_2;
```

Operadores Lógicos

```
signal a, b         : in  std_logic;
signal c, d         : in  std_logic;
architecture rtl of eje_puertas_bas is

begin
  s1 <= a and b and c and d;
  s2 <= a or b or c or d;
  s3 <= ((a and b) or c) xor d;
  s4 <= (a and b) or (c xor d);
```

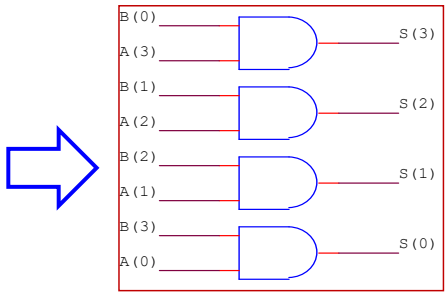
```
s3 <= a and b or c xor d;
```

¡Esta asignación es errónea!
¡Hay que utilizar paréntesis!

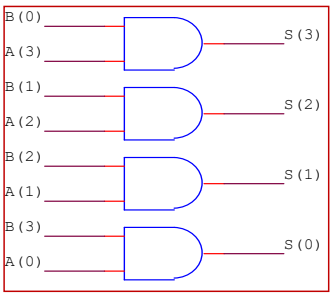


Operadores Lógicos

```
signal A, S,K: std_logic_vector(3 downto 0);
signal B: std_logic_vector(0 to 3);
signal C: std_logic;
...
 S<= A and B;
```




```
signal A, S: unsigned(3 downto 0);
signal B: unsigned(0 to 3);
...
 S<= A and B;
```

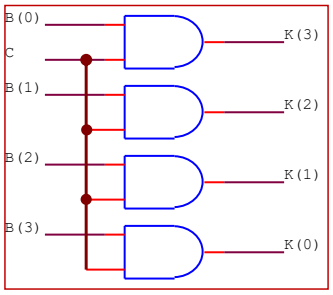



```
signal x,y,z,t : std_logic;
...
x <= y nand z nand t;
```

Error, los operadores NAND, NOR y XNOR no son asociativos

```
x <= (y nand z) nand t;
```

Operadores Lógicos




```
signal C : std_logic ;
signal B, K, AUX : std_logic_vector(3 downto 0);
. . .
```

```
K <= B when Sel = '1' else "0000" ;
```

```
AUX <= (others => C) ;
K <= B and AUX ;
```

```
K<= B and (C&C&C&C);
```

En VHDL-93 todos los argumentos tienen que ser del mismo tipo y tamaño

- En VHDL-08 se definen los operadores lógicos para aceptar vectores y elementos individuales.

```
function "and"  (l : std_ulogic_vector; r : std_ulogic) return std_ulogic_vector;
function "and"  (l : std_ulogic; r : std_ulogic_vector) return std_ulogic_vector;
```

```
K <= B and C;
```

Todas las asignaciones sería similares si los vectores son unsigned /signed.

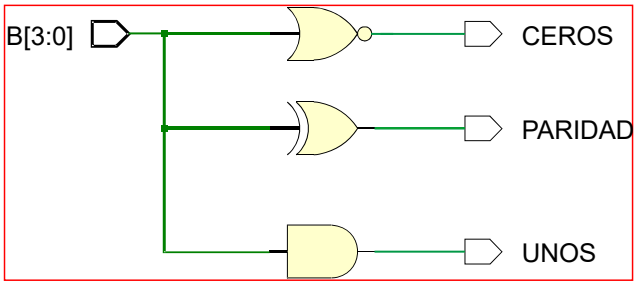
Operadores Lógicos

- VHDL-08 permite utilizar los operadores lógicos con un único operando.

```
signal B: std_logic_vector(3 downto 0);
```

```
UNOS    <= and B;
CEROS   <= nor B;
PARIDAD <= xor B;
```

Esto es válido para todo tipo de vectores



- Otros ejemplo con VHDL-08.

```
port (
    A   : in  std_logic_vector(15 downto 0);
    B   : in  std_logic_vector(15 downto 0);
    C   : in  std_logic_vector(15 downto 0);
    D   : in  std_logic_vector(15 downto 0);
    SEL : in  std_logic_vector(3 downto 0);
    S   : out std_logic_vector(15 downto 0));
. . . .

S <= (A and SEL(0)) or (B and SEL(1)) or
     (C and SEL(2)) or (D and SEL(3));
```

¿Qué circuito modela este código?

SEL	S
0001	A
0010	B
0100	C
1000	D

Resto de combinaciones, la *or* de las entradas

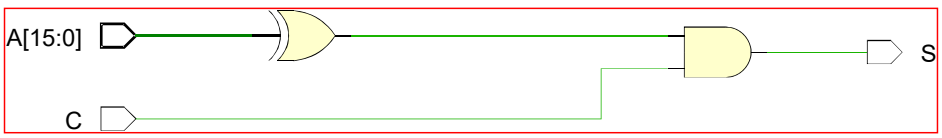
```
S <= (A and (SEL(0)and (not SEL(1)) and (not SEL(2)))and (not SEL(3))) or
     (B and (SEL(1)and (not SEL(0)) and (not SEL(2)))and (not SEL(3))) or
     (C and (SEL(2)and (not SEL(1)) and (not SEL(0)))and (not SEL(3))) or
     (D and (SEL(3)and (not SEL(1)) and (not SEL(2)))and (not SEL(0)));
```

¿Y este otro?

SEL	S
0001	A
0010	B
0100	C
1000	D

- Otros ejemplo con VHDL-08.

```
signal A: std_logic_vector(15 downto 0);
signal B,S: std_logic;
. . .
S <= (xor A) and B;
```



Modelado de la alta impedancia

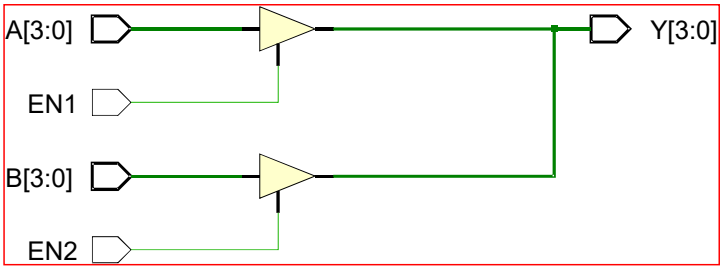
- Modelado de la alta impedancia.

```
library ieee;
use ieee.std_logic_1164.all;
entity eje_triestate is
  port(A, B     : in  std_logic_vector( 3 downto 0);
       EN1, EN2 : in  std_logic;
       y        : out std_logic_vector(3 downto 0));
end entity;
architecture RTL of eje_triestate is
begin
  Y <= A when EN1 = '1' else "ZZZZ";
  Y <= B when EN2 = '1' else "ZZZZ";
end;
```

```
Y <= A when EN1 else "ZZZZ";
Y <= B when EN2 else "ZZZZ";
```

Para VHDL-2008

(others=>'Z')

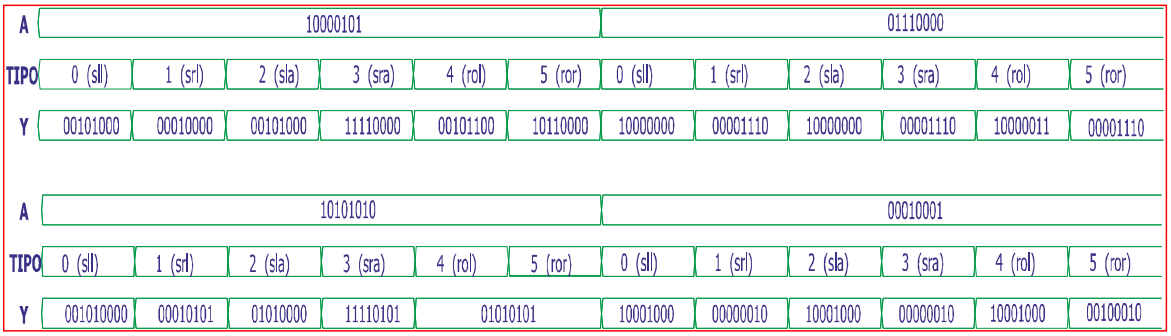


Operadores de desplazamiento

- Los desplazadores son bloques combinacionales que desplazan un dato el número de veces especificado por un parámetro. VHDL proporciona varios **operadores de desplazamiento**:
  - Desplazamiento lógico a izquierdas (**sll**-*Shift Left Logical*): descarta los bits por la izquierda y rellena con ceros por la derecha.
  - Desplazamiento lógico a derechas (**srl**-*Shift Right Logical*): descarta los bits por la derecha y rellena con ceros por la izquierda.
  - Desplazamiento aritmético a izquierdas(**sla**-*Shift Left Arithmetic*): descarta los bits por la izquierda, y rellena con ceros por la derecha.
  - Desplazamiento aritmético a derechas (**sra**-*Shift Right Arithmetic*): descarta los bits por la derecha y….
    - …para números con signo extiende el bit de signo por la izquierda.
    - …para números sin signo rellena con ceros por la izquierda.
  - Rotación a izquierdas: (**rol**-*Rotate Left*): elimina los bits por la izquierda y los reinserta por la derecha.
  - Rotación a derechas: (**ror**-*Rotate Right*): elimina los bits por la derecha y los reinserta por la izquierda.
- Los vectores implicados, origen y destino, deben tener el mismo tamaño.
- Se añadieron en VHDL-93. Estos operadores estaban definidos sólo para vectores tipo *bit* y *boolean*.
- El paquete *numeric_std* definía los operandos *rol*, *ror*, *sll* y *srl* para los tipos *unsigned/signed*.
- VHDL-08 define los operandos añade *sra* y *sla para los tipos unsigned/signed*

Operadores de desplazamiento

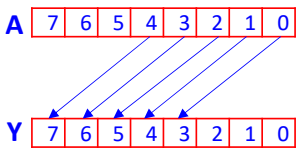
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
entity desplazador_universal is
generic(N: natural:=8; nsh: integer :=3);
port( A    : in std_logic_vector(N-1 downto 0);
      TIPO  : in std_logic_vector(2 downto 0);
      SHIFTS: in std_logic_vector(nsh-1 downto 0);
      Y1    :out std_logic_vector(N-1 downto 0));
end desplazador_universal;

architecture Behavioral of desplazador_universal is
signal veces: integer;
begin
     veces<= to_integer(unsigned(shifts));
      with tipo select y1<=
          std_logic_vector(unsigned(a) sll veces) when "000",
          std_logic_vector(unsigned(a) srl veces) when "001",
          std_logic_vector(unsigned(a) sla veces) when "010",
          std_logic_vector(signed(a)   sra veces) when "011",
          std_logic_vector(unsigned(a) rol veces) when "100",
          std_logic_vector(unsigned(a) ror veces) when "101",
          (others=>'0') when others;
end Behavioral;
```



Operadores de desplazamiento

N=8, SHIFTS=3

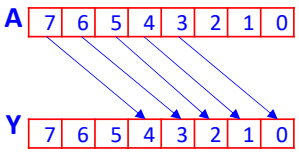
❑sll



Y(7:3)= A(4:0)

Y(N-1:J)= A(N-1-J:0)

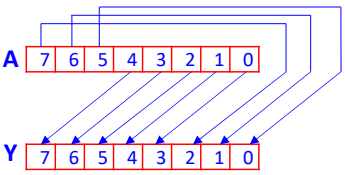
❑srl



Y(4:0)= A(7:3)

Y(N-1-J:0)= A(N-1:J)

❑rol



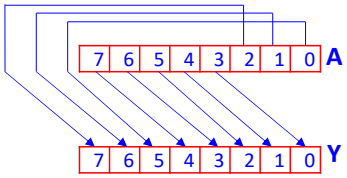
Y(7:3)= A(4:0)

Y(2:0)= A(7:5)

Y(N-1:J)= A(N-1-J:0)

Y(J-1:0)= A(N-1:J-1)

❑ror



Y(4:0)= A(7:3)

Y(7:5)= A(2:0)

Y(N-1-J:0)= A(N-1:N-J)

Y(N-1:N-J)= A(J-1:0)

```
entity desplazador_universal_2 is
 generic(N   :      natural := 8;
         nsh: integer := 3);
 port(A      : in std_logic_vector(N-1 downto 0);
      TIPO   : in std_logic_vector(2 downto 0);
      SHIFTS: in std_logic_vector(nsh-1 downto 0);
      SRL_IN: in std_logic;
      Y1     : out std_logic_vector(N-1 downto 0));
end desplazador_universal_2;
```

```
veces       <= to_integer(unsigned(shifts));
process (all)
  variable j : natural;
begin  -- process
  j := to_integer(unsigned(shifts));
  case tipo is
    when "000"                          =>  --sll
     Y1(N-1 downto j)    <= A(N-1-J downto 0);
     Y1(J-1 downto 0)    <= (others => SRL_IN);
    when "001"                          =>  --srl
     Y1(N-1-J downto 0) <= A(N-1 downto J);
     Y1(N-1 downto N-J) <= (others => SRL_IN);
    when "010"                          =>  --sla
     Y1(N-1 downto j)    <= A(N-1-J downto 0);
     Y1(J-1 downto 0)    <= (others => '0');
    when "011"                          =>  --sra
     Y1(N-1-J downto 0) <= A(N-1 downto J);
     Y1(N-1 downto N-J) <= (others => A(N-1));
    when "100"                          =>  --rol
     Y1(N-1 downto j)    <= A(N-1-J downto 0);
     Y1(J-1 downto 0)    <= A(N-1 downto N-J);
    when "101"                          =>  --ror
     Y1(N-1-J downto 0) <= A(N-1 downto J);
     Y1(N-1 downto N-J) <= A(J-1 downto 0);
    when others => y1 <= (others => '0');
  end case;
end process;
```

Operadores de desplazamiento

- Cuando el numero de posiciones a desplazar es fijo es recomendable utilizar el operador &.

```
with tipo select Y <=
    std_logic_vector(unsigned(A) sll 3) when "000",
    std_logic_vector(unsigned(A) srl 3) when "001",
    std_logic_vector(unsigned(A) sla 3) when "010",
    std_logic_vector(signed(A) sra 3)   when "011",
    std_logic_vector(unsigned(A) rol 3) when "100",
    std_logic_vector(unsigned(A) ror 3) when "101",
    (others => '0')                     when others;
```

```
when others;
  with tipo select Y <=
    A(4 downto 0)&"000"           when "000",  --sll
    "000"&A(7 downto 3)           when "001",  --srl
    A(4 downto 0)&"000"           when "010",  --sla
    A(7)&A(7)&A(7)&A(7 downto 3)  when "011",  --sra
    A(4 downto 0)&A(7 downto 5)   when "100",  --rol
    A(2 downto 0)&A(7 downto 3)   when "101",  --ror
    (others => '0')               when others;
```


Operadores aritméticos

- Se ejecutan de izquierda a derecha.
- Todos tienen la misma precedencia.

  A<= b+c*d; ⇒ A=(a+c)*d

- Es conveniente utilizar ().

Operación	Operador	Tipo de datos
Suma	+	Numérico.
Resta	-	Numérico.
Producto	*	Entero, real o físico.
División	/	Entero, real o físico.
Potencia	**	Entero, real. (Exp entero)
Módulo	mod	Entero.
Resto	rem	Entero.
Valor absoluto	abs	Numérico.

Operadores aritméticos

+ , -	Son sintetizables.
/, mod, rem	Son sintetizables si el segundo operador es una constante y potencia de 2.
*	Es sintetizables si el segundo operador es una constante y potencia de 2. En cualquier caso si el dispositivo final dispone de multiplicadores.
**	Es sintetizables si el primer operador es una constante y potencia de 2.
abs	Es sintetizables.

```
signal bus_2 : std_logic_vector(2**6 downto 0);
```

Las operaciones que proporcionan resultados constantes son sintetizable

Operadores aritméticos

- Se utilizan con los tipos ***signed*** para datos con signo y ***unsigned*** para datos sin signo (binario natural).

```
 ....
signal suma,sumando_1,sumando_2:unsigned (7 downto 0);
....
suma<= sumando_1 + sumando_2;
```

Sumador con overflow

```
signal sumando_1,sumando_2:unsigned (7 downto 0);
signal suma:unsigned (8 downto 0);
....
suma<= '0'&sumando_1 + sumando_2;
```

Sumador sin overflow para números sin signo

```
suma<= '0'&sumando_1 + '0'& sumando_2;
```

```
signal a,b: std_logic_vector (3 downto 0);
signal S  : std_logic_vector (4 downto 0);
. . .
S<= std_logic_vector( signed (a(3)&a)+ signed(b));
```

Sumador sin overflow para números con signo

```
S<= std_logic_vector( signed (a(3)&a)+ signed(b));
```

En la multiplicación, el resultado tiene tantos elementos como la suma del número de elementos de los operandos

```
signal a :unsigned (3 downto 0);
signal b :unsigned (7 downto 0);
signal s:unsigned (11 downto 0);
....
s<= a * b;
```

Operadores aritméticos

Con lo operadores aritméticos se pueden utilizar vectores y enteros (*integer*)

```
 ....
signal S,A,B:unsigned (3 downto 0);
....
S<= A + B + 8;
```

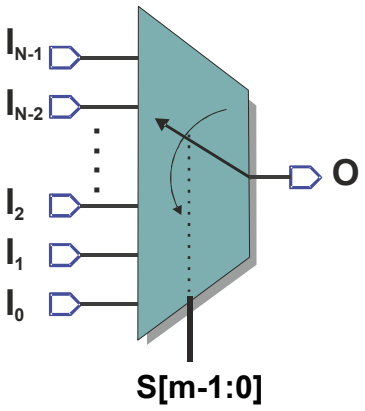
A	B	S
0011	0100	1111
0011	1000	0011
0101	0101	0010
0010	0011	1101

```
signal a,b: std_logic_vector (3 downto 0);
signal C_in: std_logic;
signal S  : std_logic_vector (4 downto 0);
. . .
S<= std_logic_vector( signed (a(3)&a)+ signed(b) + C_in);
```

Los operadores aritméticos con objetos tipo *std_logic*, solo es válido para VHDL-08

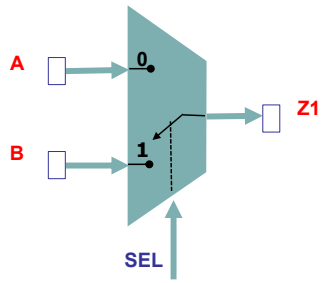
Multiplexores

- Multiplexor o selector de datos:
  - **Definición**: Circuito que permite dirigir la información de un de los N canales de entrada (I_i), determinado por los m bit de selección (S_j), a la salida (O).
  - Compuesto por:
    - **N** canales de entrada ($\mathbf{I_i}$),
    - **1** canal de salida (**O**)
    - **m** entradas de selección ($\mathbf{S_j}$)
  - Se debe cumplir que $2^m \geq N$.



- Modelado de un Multiplexor 2:1

Modelado con un proceso




```
 process (A, B, SEL)
begin
   if SEL = '0' then
      Z1 <= A;
   else
      Z1 <= B;
   end if;
end process
```

VHDL93

```
 process (all)
begin
   if SEL = '0' then
      Z1 <= A;
   else
      Z1 <= B;
   end if;
end process
```

VHDL2008

SEL	Z1
0	A
1	B

En el modelado de circuitos combinacionales con VHDL-93, en la lista de sensibilidad del proceso se deben incluir todas las señales de entrada.

Es un error muy grave que la *Lista De Sensibilidad* esté incompleta.

Para VHDL2008, en el modelado de circuitos combinacionales en la lista de sensibilidad del proceso se pone ***all***.

¿Qué modificación hay que realizar si ***A*** y ***B*** son vectores?

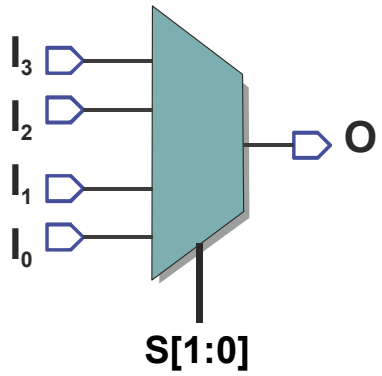
- Sentencia ***if***.
  - Permite desviar el flujo de la ejecución.
  - Admite 0 o más cláusulas **elsif**
  - Se permite el anidamiento
  - No hay restricciones en el número de anidamientos

```
[ etiqueta:  ] if condición_booleana then
                  sentencias_secuenciales
               [elsif  condición  then
                sentencias_secuenciales ]
               [ else
               sentencias_secuenciales ]
            end if  [ etiqueta ] ;
```

```
else if condición_booleana
        sentencias_secuenciales
     end if;
...
```

Es diferente a ***elsif***.

- Modelado de un Multiplexor de 4:1 .

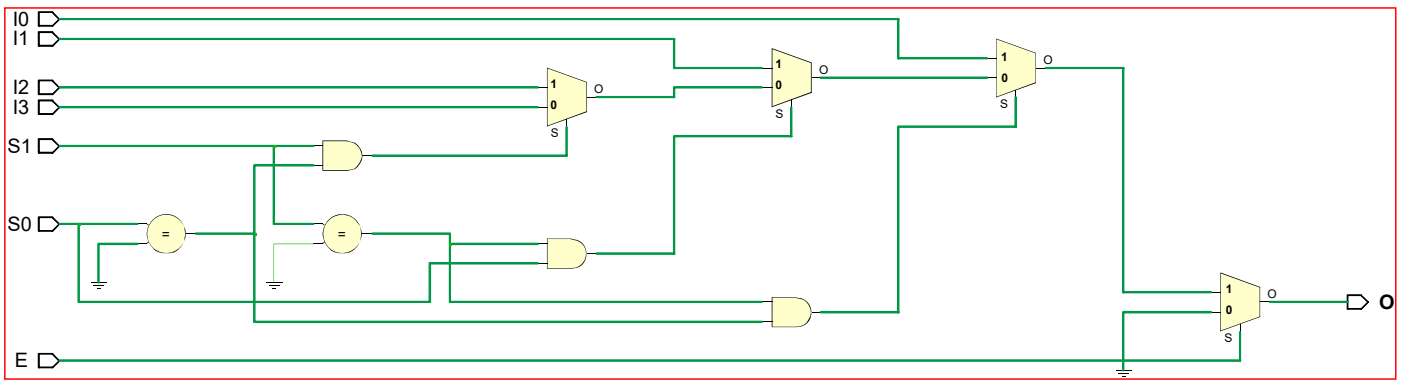


E	S_1 S_0	O
1	X X	0
0	0 0	I_0
0	0 1	I_1
0	1 0	I_2
0	1 1	I_3

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

entity mux4_1 is
  port( E, S1, S0       : in  std_logic;
        I1, I2, I3, I0  : in  std_logic;
        O               : out std_logic);
end mux4_1;

architecture RTL of mux4_1 is

begin

  process(S1, S0, E, I1, I2, I3, I0)
  begin
    if E = '0' then
      if s1 = '0' and s0 = '0' then
        O <= I0;
      elsif s1 = '0' and s0 = '1' then
        O <= I1;
      elsif s1 = '1' and s0 = '0' then
        O <= I2;
      else
        O <= I3;
      end if;
    else
      O   <= '0';
    end if;
  end process;
end RTL;
```



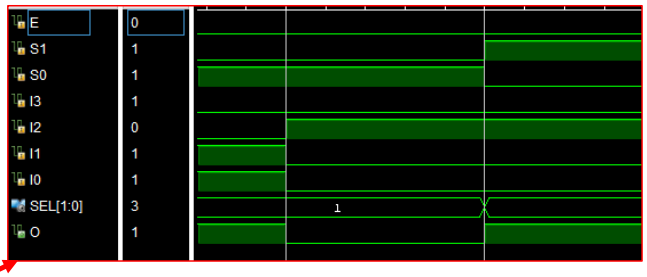
```
library ieee;
use ieee.std_logic_1164.all;

entity mux4_1 is
  port( E, S1, S0       : in  std_logic;
        I1, I2, I3, I0  : in  std_logic;
        O               : out std_logic);
end mux4_1;

architecture RTL of mux4_1 is

begin

  process(S1, S0, E, I1, I2, I3, I0)
  begin
    if E = '0' then
      if s1 = '0' and s0 = '0' then
        O <= I0;
      elsif s1 = '0' and s0 = '1' then
        O <= I1;
      elsif s1 = '1' and s0 = '0' then
        O <= I2;
      else
        O <= I3;
      end if;
    else
      O   <= '0';
    end if;
  end process;
end RTL;
```

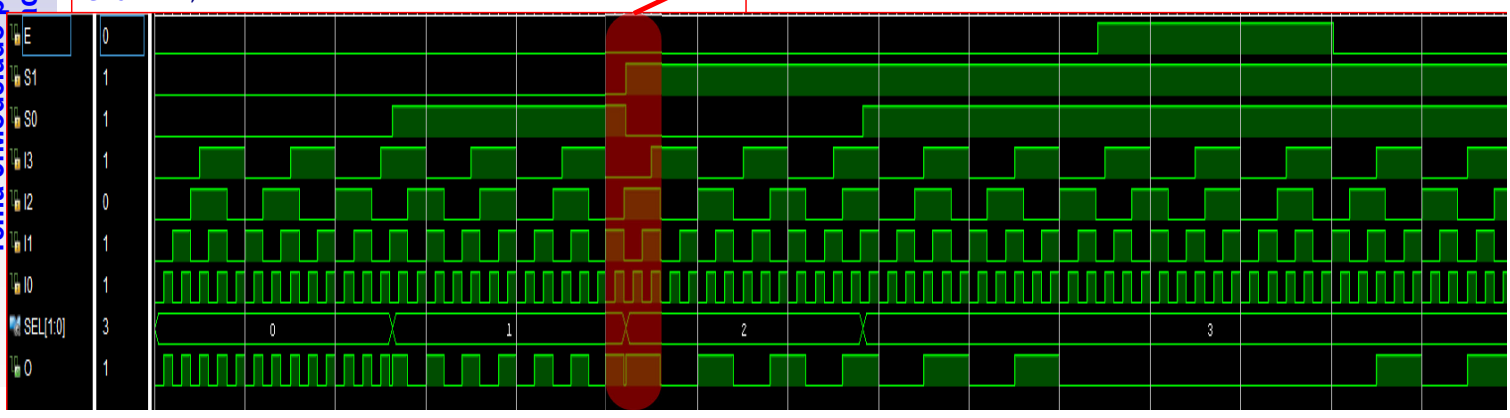


```
library ieee;
use ieee.std_logic_1164.all;
entity mux4_1 is
  port( E, S1, S0       : in  std_logic;
        I1, I2, I3, I0  : in  std_logic;
        O               : out std_logic);
end mux4_1;
architecture RTL of mux4_1 is
  signal SEL : std_logic_vector(1 downto 0);
begin

SEL <= S1&S0;

  process(SEL, E, I1, I2, I3, I0)
  begin
    if E = '0' then
      if SEL = "00" then
        O <= I0;
      elsif SEL = "01" then
        O <= I1;
      elsif SEL = "10" then
        O <= I2;
      else
        O <= I3;
      end if;
    else
      O   <= '0';
    end if;
  end process;
end RTL;
```



```
library ieee;
use ieee.std_logic_1164.all;
entity mux4_1 is
  port( E, S1, S0       : in  std_logic;
        I1, I2, I3, I0  : in  std_logic;
        O               : out std_logic);
end mux4_1;
architecture RTL of mux4_1 is
  signal SEL : std_logic_vector(1 downto 0);
begin

SEL <= S1&S0;

  process(SEL, E, I1, I2, I3, I0)
  begin
    if E = '0' then
      if SEL = "00" then
        O <= I0;
      elsif SEL = "01" then
        O <= I1;
      elsif SEL = "10" then
        O <= I2;
      else
        O <= I3;
      end if;
    else
      O   <= '0';
    end if;
  end process;
end RTL;
```



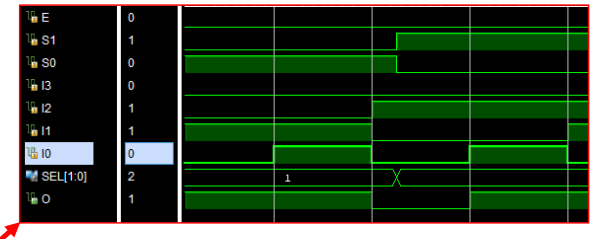


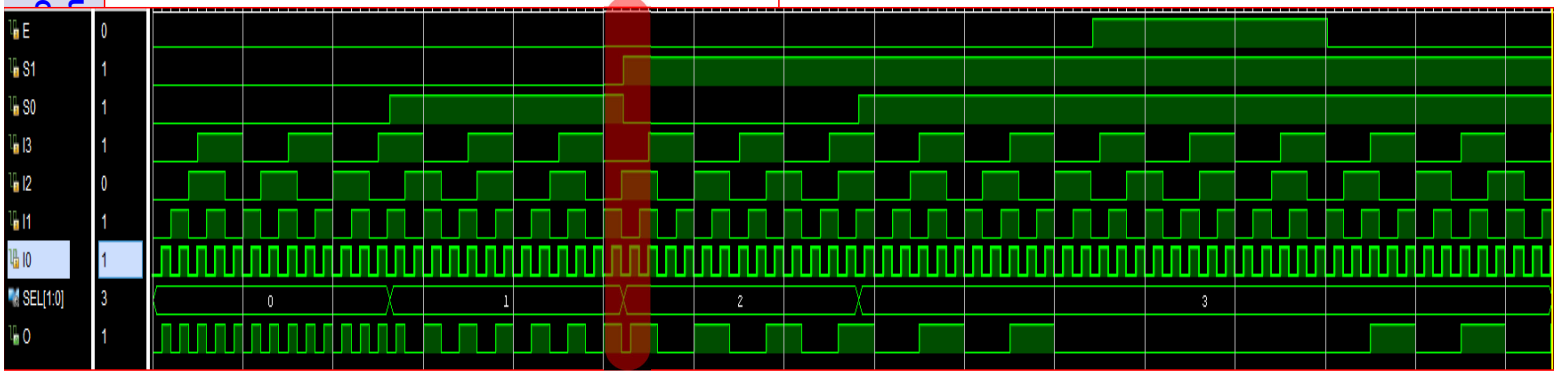
```
library ieee;
use ieee.std_logic_1164.all;
entity mux4_1 is
  port( E, S1, S0       : in  std_logic;
        I1, I2, I3, I0  : in  std_logic;
        O               : out std_logic);
end mux4_1;
architecture RTL of mux4_1 is
  signal SEL : std_logic_vector(1 downto 0);
begin

  process(S1,S0, E, I1, I2, I3, I0)
  begin

  SEL <= S1&S0;

    if E = '0' then
      if SEL = "00" then
        O <= I0;
      elsif SEL = "01" then
        O <= I1;
      elsif SEL = "10" then
        O <= I2;
      else
        O <= I3;
      end if;
    else
      O   <= '0';
    end if;
  end process;
end RTL;
```

¡El código es erróneo!

WARNING: [Synth 8-614] signal 'SEL' is read in the process but is not in the sensitivity list [C:/…]


```
library ieee;
use ieee.std_logic_1164.all;
entity mux4_1 is
  port( E, S1, S0       : in  std_logic;
        I1, I2, I3, I0  : in  std_logic;
        O               : out std_logic);
end mux4_1;
architecture RTL of mux4_1 is
  signal SEL : std_logic_vector(1 downto 0);
begin

  process(S1,S0, E, I1, I2, I3, I0)
  begin

  SEL <= S1&S0;

    if E = '0' then
      if SEL = "00" then
        O <= I0;
      elsif SEL = "01" then
        O <= I1;
      elsif SEL = "10" then
        O <= I2;
      else
        O <= I3;
      end if;
    else
      O   <= '0';
    end if;
  end process;
end RTL;
```





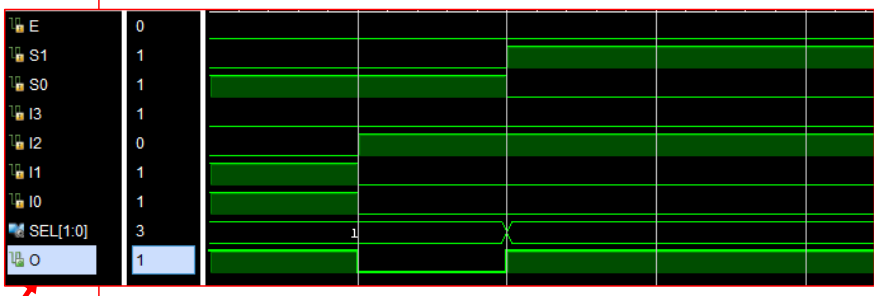
Sentencia IF

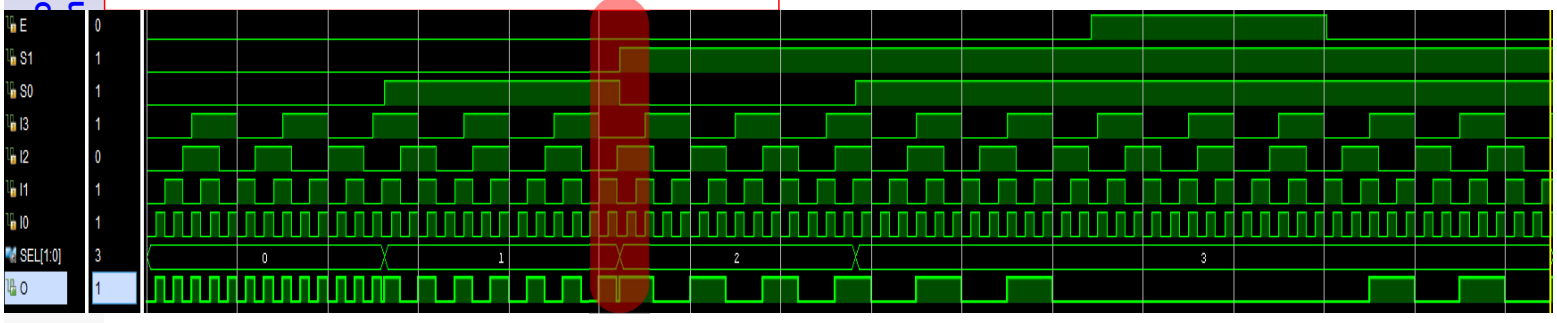
```
library ieee;
use ieee.std_logic_1164.all;
entity mux4_1 is
  port( E, S1, S0       : in  std_logic;
        I1, I2, I3, I0  : in  std_logic;
        O               : out std_logic);
end mux4_1;
architecture RTL of mux4_1 is
  signal SEL : std_logic_vector(1 downto 0);
begin

  process(S1,S0,SEL,E,I1,I2,I3,I0)
  begin

  SEL <= S1&S0;

    if E = '0' then
      if SEL = "00" then
        O <= I0;
      elsif SEL = "01" then
        O <= I1;
      elsif SEL = "10" then
        O <= I2;
      else
        O <= I3;
      end if;
    else
      O   <= '0';
    end if;
  end process;
end RTL;
```

¡Esta, es una mala solución!

Sentencia IF

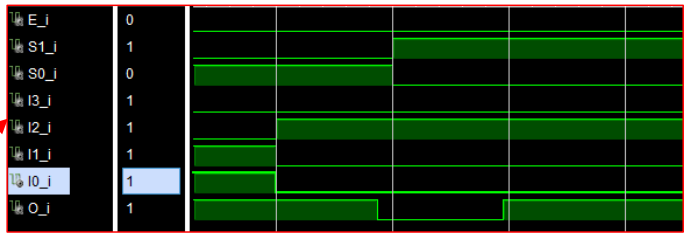
```
library ieee;
use ieee.std_logic_1164.all;
entity mux4_1 is
  port( E, S1, S0       : in  std_logic;
        I1, I2, I3, I0  : in  std_logic;
        O               : out std_logic);
end mux4_1;
architecture RTL of mux4_1 is
  signal SEL : std_logic_vector(1 downto 0);
begin

  process(S1,S0,SEL,E,I1,I2,I3,I0)
  begin

  SEL <= S1&S0;

    if E = '0' then
      if SEL = "00" then
        O <= I0;
      elsif SEL = "01" then
        O <= I1;
      elsif SEL = "10" then
        O <= I2;
      else
        O <= I3;
      end if;
    else
      O   <= '0';
    end if;
  end process;
end RTL;
```

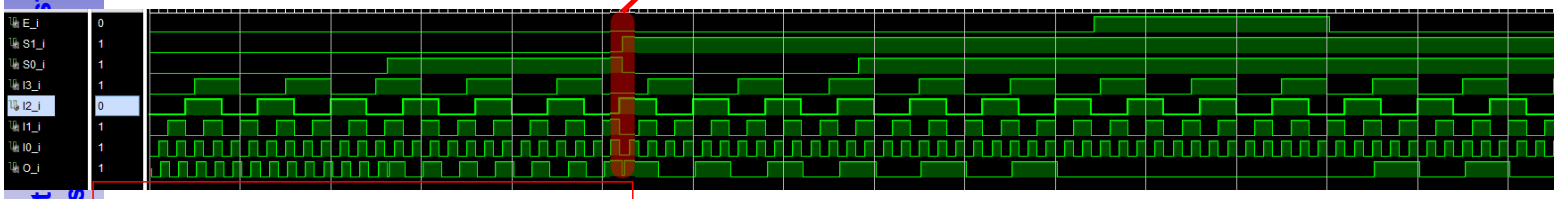




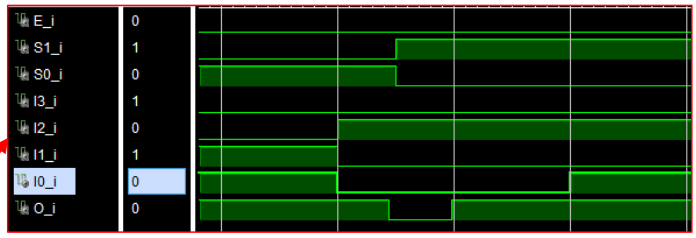
❑ Simulación temporal.

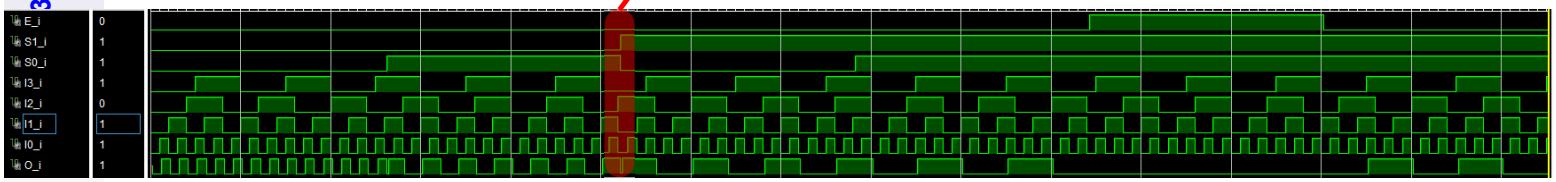
```
SEL <= S1&S0;

  process(SEL, E, I1, I2, I3, I0)
  begin
    if E = '0' then
      if SEL = "00" then
        O <= I0;
      elsif SEL = "01" then
        O <= I1;
```



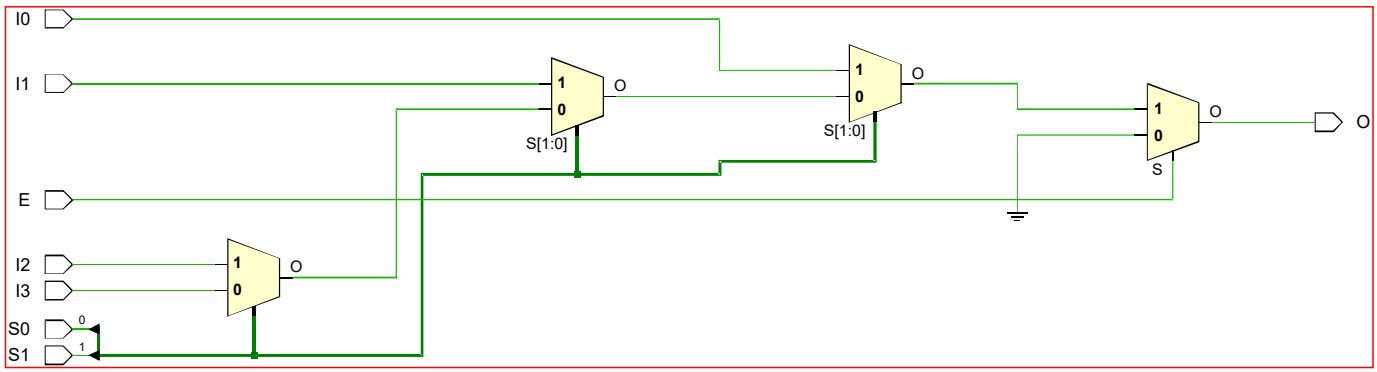


```
process(S1,S0,E,I1,I2,I3,I0)
begin
SEL <= S1&S0;
  if E = '0' then
    if SEL = "00" then
      O <= I0;
    elsif SEL = "01" then
      O <= I1;
    elsif SEL = "10"
```





¡Todos los modelos generan la misma estructura!



El modelo correcto es:

```
SEL <= S1&S0;

  process(SEL, E, I1, I2, I3, I0)
  begin
    if E = '0' then
      if SEL = "00" then
        O <= I0;
      elsif SEL = "01" then
        O <= I1;
. . . .
```

- Modelado de un Multiplexor de 4:1 con VHDL 2008.

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

entity mux4_1 is
  port( E, S1, S0       : in  std_logic;
        I1, I2, I3, I0  : in  std_logic;
        O               : out std_logic);
end mux4_1;

architecture RTL of mux4_1 is

begin

  process(all)
  begin
    if E = '0' then
      if s1 = '0' and s0 = '0' then
        O <= I0;
      elsif s1 = '0' and s0 = '1' then
        O <= I1;
      elsif s1 = '1' and s0 = '0' then
        O <= I2;
      else
        O <= I3;
      end if;
    else
      O   <= '0';
    end if;
  end process;
end RTL;
```

VHDL-08 permite simplificar la lista de sensibilidad de los procesos utilizando la palabra reservada ***all*** para declarar todas las entradas

Esto permite reducir notablemente los errores de síntesis.

- Cosas que se pueden hacer y no.

```
library ieee;
use ieee.std_logic_1164.all;
entity mux4_1 is
  port( E, S1, S0       : in  std_logic;
        I1, I2, I3, I0  : in  std_logic;
        O               : out std_logic);
end mux4_1;
architecture RTL of mux4_1 is
Begin
process(S1, S0, E, I1, I2, I3, I0)
  begin
    if E = '0' then
      if   S1&S0 = "00" then
        O <= I0;
      elsif S1&S0 = "01" then
        O <= I1;
      elsif S1&S0 = "10" then
        O <= I2;
      else
        O <= I3;
      end if;
    else
      O   <= '0';
    end if;
  end process;
```

Vivado(AMD_Xilinx)

```
Error: found "2" definition of operator, cannot determine exact overloaded   matching definition for "="
```

Quartus(Intel-Altera)

```
Error (10327): can't determine definition of operator ""="" -- found 2 possible definitions
```

Hay ambigüedad en cuanto al tipo de datos de la expresión

```
if std_logic_vector'(S1&S0) = "00" then
```

```
if std_logic_vector'(S1, S0) = "00" then
```

La sintaxis *std_logic_vector'( )*es una expresión VHDL cualificada (*qualified expression*).

La expresión entre paréntesis se interpreta como una agregación que se convierte en un vector. resultante que debe ser *std_logic_vector*. Con la notación de la expresión cualificada, se define explícitamente el tipo de dato resultante.

Sentencia de asignación condicional

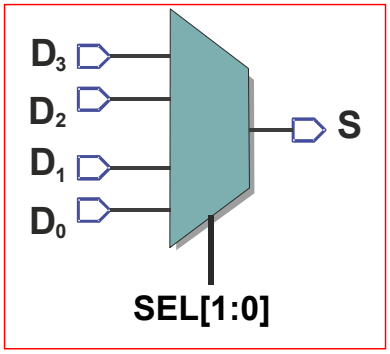
Sentencia de asignación condicional

- Permite desviar el flujo de la ejecución de forma similar a las sentencia if.
- Es sensible a las señales que aparecen en las expresiones de las formas de ondas o en las propias condiciones.

```
[ etiqueta:  ] nombre_señal <=
            {forma_de_onda  when expresión_booleana else}
             [forma_de_onda when expresión_booleana ];
```

- En VHDL-93 es una sentencia concurrente y va siempre fuera de los procesos.

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

entity mux4_1_conc is
  port( sel             : in  std_logic_vector(1 downto 0);
        D0, D1, D2, D3  : in  std_logic_vector(7 downto 0);
        S               : out std_logic_vector(7 downto 0));
end mux4_1_conc;

architecture rtl_1 of mux4_1_conc is

begin

  S <= D0 when (sel = "00" ) else
       D1 when (sel = "01" ) else
       D2 when (sel = "10")  else
       D3 when (sel = "11")  else (others=>'0');
end architecture;
```



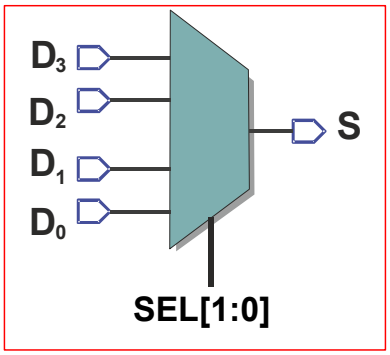
Tema 3:Modelado para síntesis de subsistemas combinacionales en VHDL.

Sentencia de asignación condicional

- En VHDL-08 puede ser una sentencia concurrente o secuencial según si va fuera o dentro de los procesos.

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

entity mux4_1_sec is
  port( sel             : in  std_logic_vector(1 downto 0);
        D0, D1, D2, D3  : in  std_logic_vector(7 downto 0);
        S               : out std_logic_vector(7 downto 0));
end mux4_1_sec;

architecture rtl_1 of mux4_1_sec is

begin
  process (all)
  begin
    S <= D0 when (sel = "00" ) else
         D1 when (sel = "01" ) else
         D2 when (sel = "10")  else
         D3 when (sel = "11")  else (others => '0');
  end process;

end architecture;
```

La asignación también puede ser a una variable

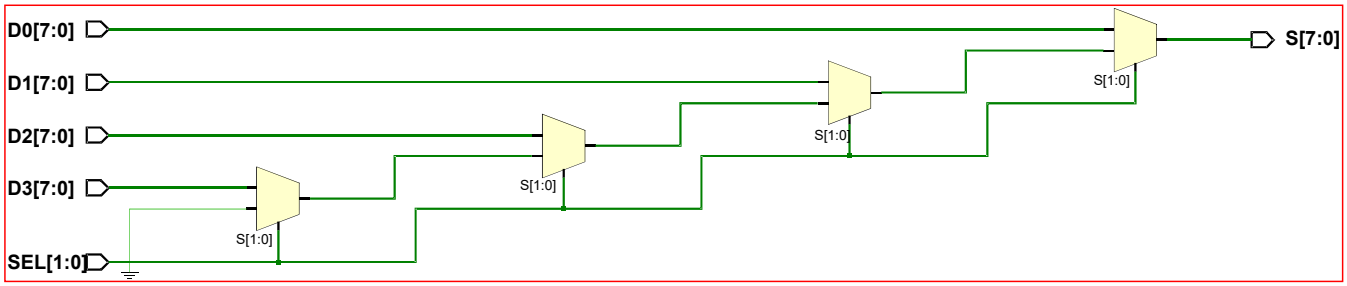


Comparativa.

```
S <= D0 when (sel = "00" ) else
     D1 when (sel = "01" ) else
     D2 when (sel = "10")  else
     D3 when (sel = "11")  else (others=>'0');
```

```
process (all)
  begin
    S <= D0 when (sel = "00" ) else
         D1 when (sel = "01" ) else
         D2 when (sel = "10")  else
         D3 when (sel = "11")  else (others => '0');
  end process;
```

```
process(all)
 begin
   if SEL = "00" then
     S <= D0;
   elsif SEL = "01" then
     S <= D1;
   elsif SEL = "10" then
     S <= D2;
   elsif SEL = "11" then
     S <= D3;
     else
     S<=(others=>'0');
   end if;
 end process;
```

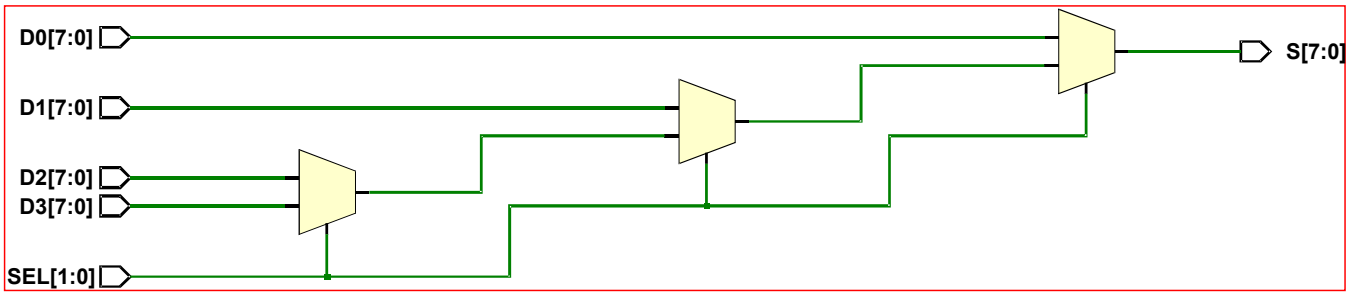


Comparativa.

```
S <= D0 when (sel = "00" ) else
     D1 when (sel = "01" ) else
     D2 when (sel = "10")  else
     D3;
```

```
process (all)
  begin
    S <= D0 when (sel = "00" ) else
         D1 when (sel = "01" ) else
         D2 when (sel = "10")  else
         D3;
  end process;
```

```
process(all)
 begin
   if SEL = "00" then
     S <= D0;
   elsif SEL = "01" then
     S <= D1;
   elsif SEL = "10" then
     S <= D2;
   elsif
  end if;
 end process;
```



- Operadores relacionales
  - Permiten comparar objetos y datos.
  - Admiten cualquier tipo de datos.
  - Devuelven un valor ***boolean*** (*true* ó *false*).
  - Los objetos comparados deben ser del mismo tipo.

OPERACIÓN	OPERADOR
Igual	=
Diferente	/=
Mayor que	>
Menor que	<
Mayor o igual que	>=
Menor o igual que	<=

```
signal s1,s2,s3 : std_logic_vector(3 downto 0);
signal s4 : std_logic_vector(2 downto 0);
signal r1,r2 : std_logic;
signal ROW : in std_logic_vector(3 downto 0);
…

if(ce = '1') and (ROW = x"F") then
...

if (s1=s2) then
     ...
y <= c when (s1=s2) else. .

...
if (s3<s4) then

...

y <= c when (s3<s4) else. .
```

```
y <= c when (a = '1' ) else
     d when (b = '1' ) ;
```

```
ROW = 4x"F"
ROW = 4d"15"
```

Para VHDL-2008

Alinea a la izquierda

Compara a derechas

s3	1	0	1	1
s4	1	1	0	

- El paquete ***numeric_std*** permiten comparar ***un/signed*** con enteros.

```
signal  counter_bit: unsigned(4 downto 0);
signal CE : std_logic;

---
 if (CE = '1') and (counter_bit < 8) then
     counter_bit <= counter_bit+1;
 elsif (busy = '0') and (counter_bit = 8)  then
     counter_bit <= (others => '0');
 end if;
```

```
  H <= D0 when (counter_bit < 8) else
       D1 when (counter_bit >7) and (counter_bit < 12) else
       D2 when (counter_bit >11)and (counter_bit < 14)  else
       D3;
```

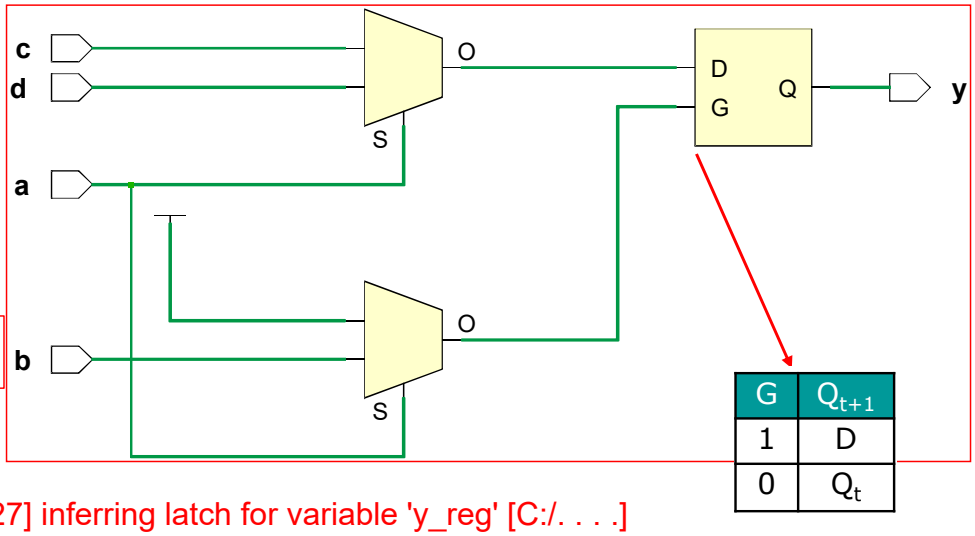
```
counter_bit <= 0;
```

Error, tipos diferentes

- Hay que tener mucho cuidado con las construcciones ***if*** incompletas. Pueden generar (inferir) latches.

```
process(all)
begin
  if (a='1')  then
    y <= c;
  elsif (b='1') then
    y <= d;
  end if;
end process;
```

```
y <= c when (a = '1' ) else
     d when (b = '1' ) ;
```



G	Q_{t+1}
1	D
0	Q_t

WARNING: [Synth 8-327] inferring latch for variable 'y_reg' [C:/. . . .]

¿Cuánto vale ***y*** para a='0' y b='0'? ⇨ El valor que tenía

¡Es un circuito secuencial ya que tiene memoria!

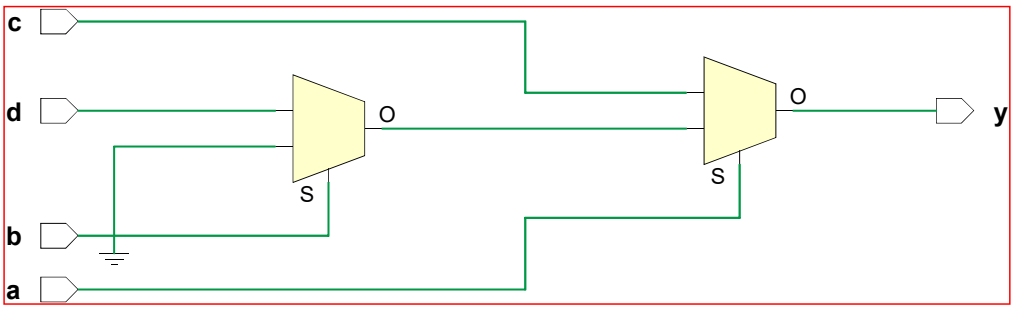
¡Es un error que se comete a menudo!

- Solución.

```
process(all)
begin
  if (a='1') then
    y <= c;
  elsif ( b = '1') then
    y <= d;
  else
    y<= '0';
  end if;
```

```
process(all)
begin
    y<= '0';
  if (a='1') then
    y <= c;
  elsif ( b = '1') then
    y <= d;
  end if;
end process;
```

```
y <= c when (a = '1' ) else
     d when (b = '1' ) else
     '0';
```

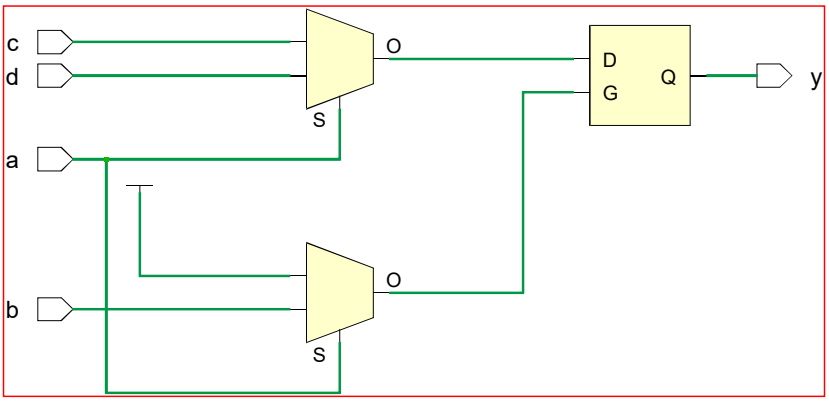


- Solución incorrecta.

```
process(all)
begin
  if (a='1') then
    y <= c;
  elsif ( b = '1') then
    y <= d;
  else
    y<= y;
  end if;
```

```
process(all)
begin
    y<= y;
  if (a='1') then
    y <= c;
  elsif ( b = '1') then
    y <= d;
  end if;
end process;
```

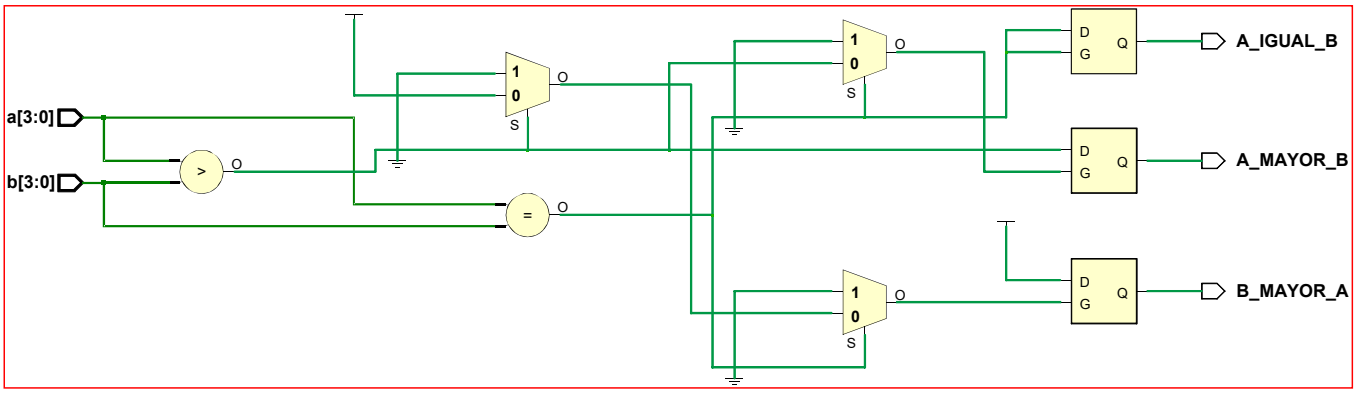
```
y <= c when (a = '1' ) else
     d when (b = '1' ) else
     y;
```

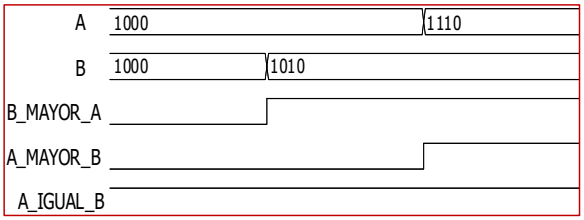


- Otra forma generar (inferir) latches.

ENTRADAS	SALIDAS		
	A_MAYOR_B	*B_MAYOR_A*	*A_IGUAL_B*
A > B	1	0	0
A < B	0	1	0
A = B	0	0	1

```
process (A, B)
  begin
    if A = B then
      A_IGUAL_B <= '1';
    elsif A > B then
      A_MAYOR_B <= '1';
    else
      B_MAYOR_A <= '1';
    end if;
end process;
```

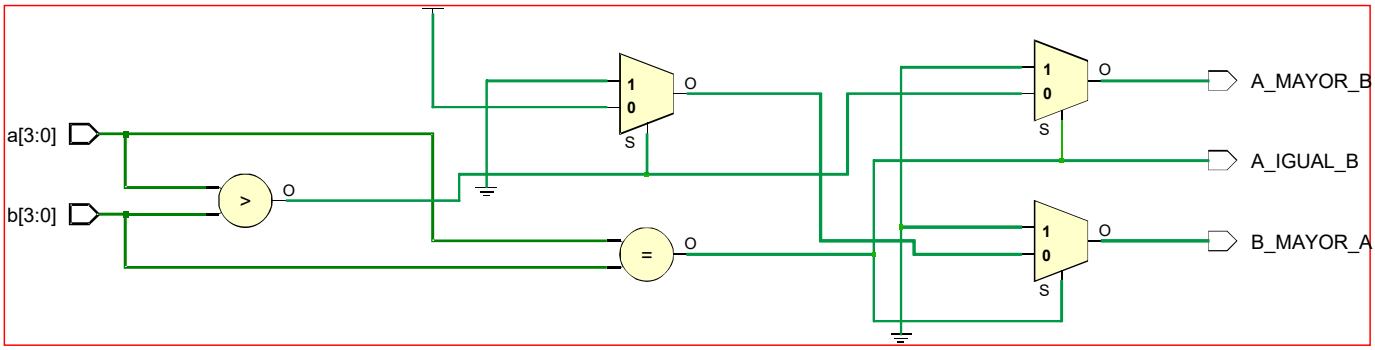




Solución.

```
process (A, B)
  begin
    A_IGUAL_B   <= '0';
    A_MAYOR_B   <= '0';
    B_MAYOR_A   <= '0';
    if A = B then
      A_IGUAL_B <= '1';
    elsif A > B then
      A_MAYOR_B <= '1';
    else
      B_MAYOR_A <= '1';
    end if;
  end process;
```

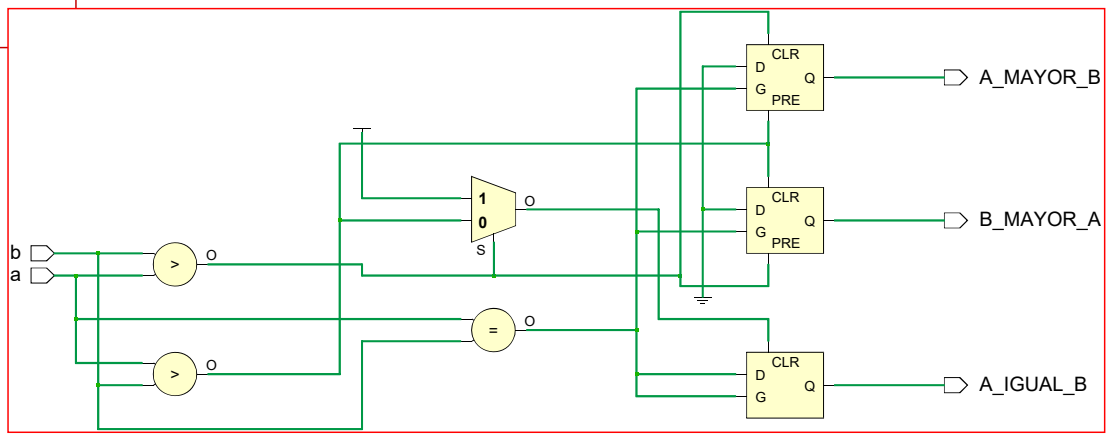
```
process (A, B)
  begin
    if A = B then
      A_IGUAL_B <= '1';
      A_MAYOR_B <= '0';
      B_MAYOR_A <= '0';
    elsif A > B then
      A_IGUAL_B <= '0';
      A_MAYOR_B <= '1';
      B_MAYOR_A <= '0';
    else
      A_IGUAL_B <= '0';
      A_MAYOR_B <= '0';
      B_MAYOR_A <= '1';
    end if;
  end process;
```



Para evitar los latchs: ¡a todas las señales DE SALIDA DE UN PROCESO COMBINACIONAL siempre se le debe asignar un valor, independientemente del camino seguido en la ejecución del proceso!

Solución incorrecta.

```
process (A, B)
  begin
    if A = B then
      A_IGUAL_B <= '1';
      A_MAYOR_B <= '0';
      B_MAYOR_A <= '0';
    end if;
    if A > B then
      A_IGUAL_B <= '0';
      A_MAYOR_B <= '1';
      B_MAYOR_A <= '0';
    end if;
    if B > A then
      A_IGUAL_B <= '0';
      A_MAYOR_B <= '0';
      B_MAYOR_A <= '1';
    end if;
  end process;
```



Sentencias condicionales

- Antes de VHDDL-08 era necesario que las expresiones condicionales devolvieran un valor tipo *boolean* (*true* o *false*) ahora pueden devolver un valor tipo *bit* o *std_ulogic* (1= *true* y 0= *false*). También *std_ulogic.*

```
SUBTYPE std_logic IS resolved std_ulogic;
```

```
 process (A, B, SEL)
begin
   if SEL = '0' then
      Z1 <= A;
   else
      Z1 <= B;
   end if;
end process
```

VHDL93

```
 process (all)
begin
   if not SEL then
      Z1 <= A;
   else
      Z1 <= B;
   end if;
end process
```

VHDL2008

```
if (A='1' and B='0' and C='1) then
```

```
if (A and not B and C) then
```

```
if (A and not B and C='1') then
```

¡Error!

```
if a = '1' and b = '1' and d = x"A" then
```

```
if a and b and d = 4d"10" then
```

¡Error!

No se puede realizar operaciones lógicas entre tipo bit/std_ulogic con boolean.

Sentencias condicionales

- Es lo mismo para la asignación condicional.

```
O<= A when (B and not C) else E;
```

=

```
O<= A when (B='1' and  C='0') else E;
```

```
O<= A when (B and C='0') else E;
```

¡Error!

```
O<= A when (B and not C and d = 4d"10" ) else E;
```

¡Error!

```
O1 <= (C and E) when (A = '0' and B = '0') else
      (C xor E) when(A = '0'and B = '1')   else
      (C or E)  when(A = '1'and B = '0')   else
       C nor E;
```

```
O1 <= (C and E) when (not A and not B) else
      (C xor E) when(not A and B)      else
      (C or E)  when(A and not B)      else
      C nor not E;
```

- VHDL-08 define nuevos operadores relacionales que devuelven un valor tipo *bit* o *std_ulogic* (1 o 0).

Resultado	
boolean	bit/std_ulogic
=	?=
/=	?/=
<	?<
<=	?<=
>	?>
>=	?>=

```
if a and b and d ?= 4d"10" then
```

El operador **?=**devuelve un valor std_logic.

```
O<= A when (B and not C and d ?= 4d"10" ) else E;
```

OPERANDOS		RESULTADO	
X	Y	X=Y	X=?Y
'0'	'0'	true	'1'
'0'	'1'	false	'0'
'0'	'L'	false	'1'
'0'	'H'	false	'0'
'0'	'-'	false	'1'
'1'	'1'	true	'1'
'1'	'L'	false	'0'
'1'	'H'	false	'1'
'1'	'-'	false	'1'

```
signal aux_1:std_logic_vector(3 downto 0);
signal aux_2:unsigned(3 downto 0);
. . .
O2<= A when (aux_1(3) and aux_2(3))  else E;
```

```
O2<= a when (b or not c or f) else e;
```

```
O3<= a when (b or not c xor f)  else e;
```

¡Error!

```
O3<= a when (b or not c) xor f  else e;
```

```
O1<= a when (b xor f ) else e;
```


- Al devolver un valor tipo *bit* o *std_ulogic* (1 o 0) los operandos relacionales con ? se pueden utilizar en las asignaciones.

```
S <= A ?= B;
```

=

```
signal A, B   : std_logic_vector(3 downto 0);
signal S: std_logic;
begin
  . . .
  S <= '1' when A = B else '0';
```

```
S <= C ?> D;
```

=

```
signal C, D   : unsigned(3 downto 0);
signal S: std_logic;
begin
  . . .
  S <= '1' when C > D else '0';
```

Otros operadores

- VHDL-08 define las funciones *maximum* y *minimum* *que devuelven el mayor o el menor respectivamente de dos valores.*

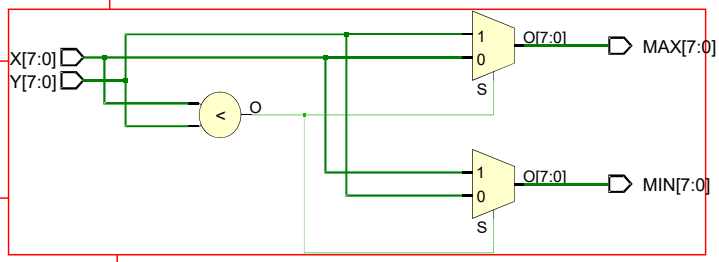
  Pueden trabajar con escalares y vectores.

```
signal MAX,MIN,X,Y : std_logic_vector(7 downto 0);
. . .
begin
. . .
```

```
MAX<=maximum( X,Y);
```

=

```
process(all)
 begin
   if X > Y then
     MAX <= X;
   else
     MAX <= Y;
   end if;
 end process;
```




```
MIN<=minimum( X,Y);
```

=

```
process(all)
 begin
   if X > Y then
     MAX <= X;
   else
     MAX <= Y;
   end if;
 end process;
```


Otros operadores

- Funciones *maximum* y *manimum.*

```
signal MAX,MIN,X,Y : std_logic_vector(7 downto 0);
. . .
begin
. . .
```

```
MAX<=  maximum("100",X);
```

¡Error, tamaños diferentes!

```
MAX<=  maximum(123,X);
```

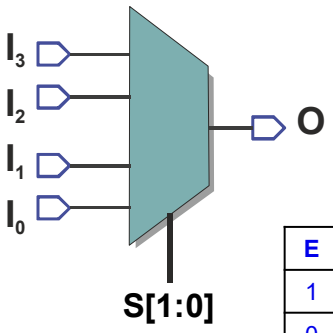
¡Error, tipos diferentes!

```
signal MAX,X,Y : unsigned(7 downto 0);
. . .
begin
. . .
```

```
MAX<=  maximum(123,X);
```

Asignación correcta.

- Multiplexor de 4 canales, otro modelado .



E	S_1 S_0	O
1	X X	0
0	0 0	I0
0	0 1	I1
0	1 0	I2
0	1 1	I3

```
type std_ulogic is ( 'U', -- Uninitialized
                     'X', -- Forcing Unknown
                     '0', -- Forcing 0
                     '1', -- Forcing 1
                     'Z', -- High Impedance
                     'W', -- Weak Unknown
                     'L', -- Weak 0
                     'H', -- Weak 1
                     '-' -- Don't Care
                   );
```

Se deben contemplar todas las combinaciones

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

entity mux_4_1 is
  port( E, S1, S0       : in  std_logic;
        I1, I2, I3, I0  : in  std_logic;
        O               : out std_logic);
end entity mux_4_1;

architecture rtl of mux_4_1 is
  signal sel : std_logic_vector(1 downto 0);
begin
  sel <= S1 & S0;

  process(sel, I0, I1, I2, I3)
  begin
    if E = '0' then
      case sel is
        when "00"   =>
          O <= I0;
        when "01"   =>
          O <= I1;
        when "10"   =>
          O <= I2;
        when "11"   =>
          O <= I3;
        when others =>
          O<= '0';
      end case;
    else
      O     <= '0';
    end if;
  end process;
end architecture;
```

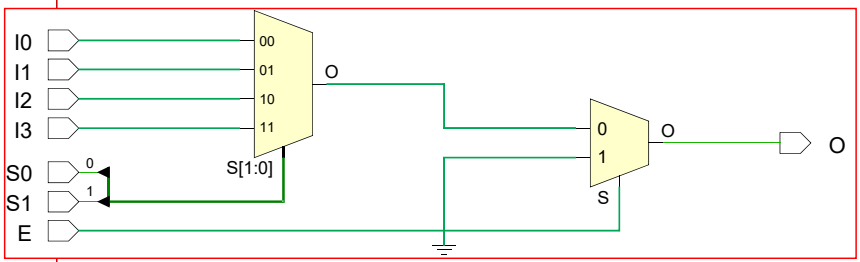
Para VHDL-2008

```
process(all)
```

```
when others =>
  O <= I3;
end case;
```


- Multiplexor de 4 canales, otro modelado .

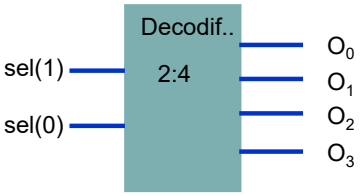
```
library ieee;
use ieee.std_logic_1164.all;

entity mux_4_1 is
  port( E, S1, S0       : in  std_logic;
        I1, I2, I3, I0  : in  std_logic;
        O               : out std_logic);
end entity mux_4_1;

architecture rtl of mux_4_1 is
  signal sel : std_logic_vector(1 downto 0);
begin
  sel <= S1 & S0;

  process(all)
  begin
    if E = '0' then
      case sel is
        when "00"   =>
          O <= I0;
        when "01"   =>
          O <= I1;
        when "10"   =>
          O <= I2;
        when "11"   =>
          O <= I3;
        when others =>
          O<= '0';
      end case;
    else
      O     <= '0';
    end if;
  end process;
end architecture;
```



Decodificadores

Decodificadores.

Es un circuito combinacional de ***m*** entradas y ***N*** salidas ($2^m \geq N$) de las que en cada momento se activa "sólo" una de ellas.



Sel[1:0]	O[3..0]
00	0001
01	0010
10	0100
11	1000

```
library ieee;
use ieee.std_logic_1164.all;
entity decoder is
  port (
    sel  : in  std_logic_vector(1 downto 0);
    O    : out std_logic_vector(3 downto 0));
end decoder;

architecture rtl of decoder is
begin
  process (sel) is
  begin
    case sel is
      when "00"   =>
        O <= "0001";
      when "01"   =>
        O <= "0010";
      when "10"   =>
        O <= "0100";
      when others =>
        O <= " 1000 ";
     end case;
  end process;
end rtl;
```


Decodificadores

Sentencia ***null***. Es una sentencia secuencial

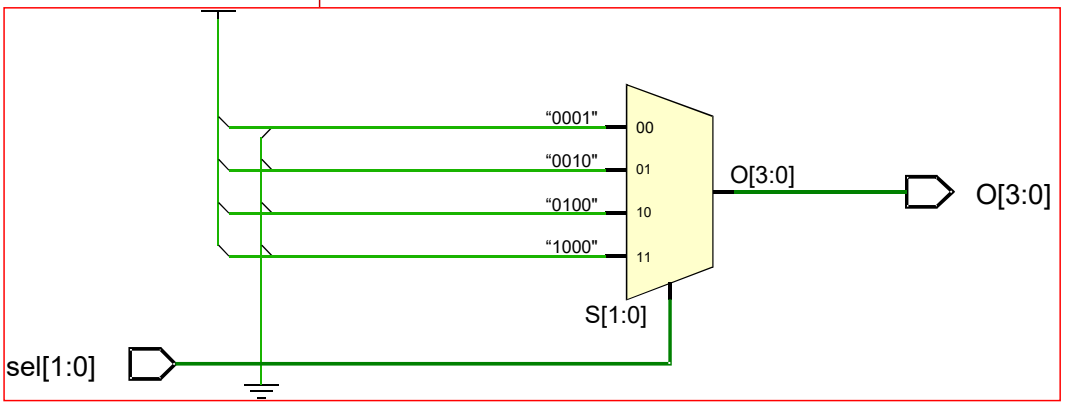
```
library ieee;
use ieee.std_logic_1164.all;
entity decoder is
  port (
    sel  : in  std_logic_vector(1 downto 0);
    O    : out std_logic_vector(3 downto 0));
end decoder;

architecture rtl of decoder is
begin
  process (sel) is
  begin
    case sel is
      when "00"   =>
        O <= "0001";
      when "01"   =>
        O <= "0010";
      when "10"   =>
        O <= "0100";
      when "11"   =>
        O <= "1000";
      when others =>
        null;
    end case;
  end process;
end rtl;
```

```
when others =>
      O <= " 1000 ";
end case;
```

```
when others =>
      O <= "0000 ";
end case;
```

¡La sentencia ***null*** puede inferir latches!

```
library ieee;
use ieee.std_logic_1164.all;
entity decoder is
  port (
    sel  : in  std_logic_vector(1 downto 0);
    O    : out std_logic_vector(3 downto 0));
end decoder;

architecture rtl of decoder is
begin
  process (sel) is
  begin
    case sel is
      when "00"   =>
        O <= "0001";
      when "01"   =>
        O <= "0010";
      when "10"   =>
        O <= "0100";
      when others =>
        O <= "1000";
    end case;
  end process;
end rtl;
```



- Sentencia ***case***.
  - Permiten seleccionar el grupo de sentencias que deben ejecutarse entre un conjunto de posibilidades.
  - Las alternativas son excluyentes (sólo se debe satisfacer una).
  - Se deben contemplar todas las posibilidades.

```
[etiqueta:]case expresión is
   when elección =>
         sentencias_secuenciales
   {when elección =>
         sentencias_secuenciales }
    end case  [etiqueta] ;
```

¡La sentencia ***case*** sólo se pueden utilizar dentro de los procesos.!

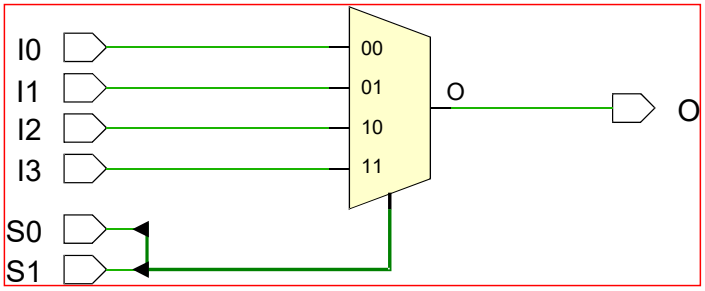
Sentencias de selección condicional

- Sentencia ***with select***.
  - Se deben contemplar todas las posibilidades.
  - En VHDL-93 es una sentencia concurrente y va siempre fuera de los procesos.

```
[etiqueta:] with expresión select
                nombre_señal <=
                        {forma_de_onda when valor}
                    ....
                        [{forma_de_onda when valor}];
```

- Modelado de un multiplexor 4:1.

```
with sel select
  O <=
  I0 when "00",
  I1 when "01",
  I2 when "10",
  I3 when others;
```

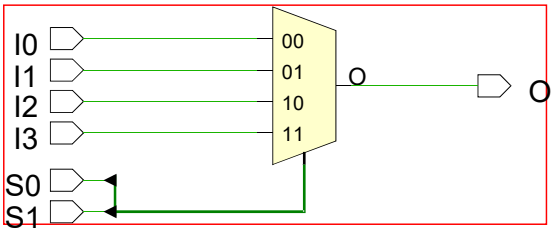


Tema 3:Modelado para síntesis de subsistemas combinacionales en VHDL.

Sentencias de selección condicional

- En VHDL-08 puede ser una sentencia concurrente o secuencial según si va fuera o dentro de los procesos.

```
process (all) is
  begin
   with sel select
    O <=
    I0 when "00",
    I1 when "01",
    I2 when "10",
    I3 when others;
end process;
```



- Dentro de un proceso, también se puede utilizar con una variable.

```
process (all) is
variable R: std_logic;
  begin
  . . .
   with AUX select
    R :=
    A when "00",
    B when "01",
    C when "10",
    D when others;
          . . .
end process;
```

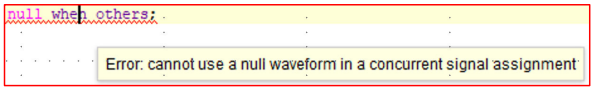
Sentencias de selección condicional

- Particularidades de las herramientas.

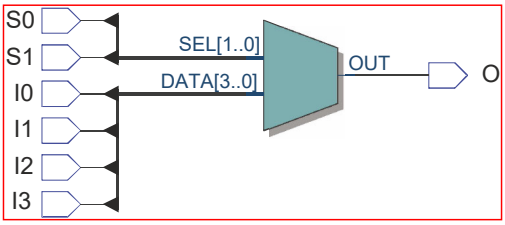
```
with sel select
   O <=
   I0   when "00",
   I1   when "01",
   I2   when "10",
   I3   when "11",
   null when others;
```

```
process (all) is
  begin
   with sel select
      O <=
      I0   when "00",
      I1   when "01",
      I2   when "10",
      I3   when "11",
      null when others;
end process;
```

Vivado 22.2 dice:



Quartus Prime Lite 19.1:

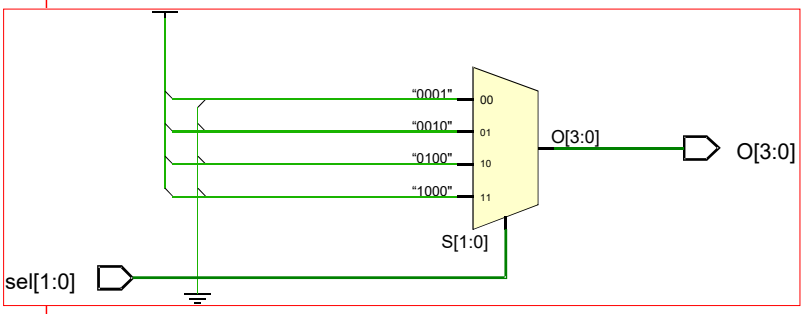


Sentencias de selección condicional

- Modelado de un decodificador 2:4.

```
  with sel select
    O <=
    "0001" when "00",
    "0010" when "01",
    "0100" when "10",
    "1000" when others;
```

```
process (all) is
  begin
 with sel select
    O <=
    "0001" when "00",
    "0010" when "01",
    "0100" when "10",
    "1000" when others;
end process;
```



Sentencias de selección condicional

Sel[1:0]	O[3..0]
00	0001
01	0010
10	0100
11	1000

```
architecture rtl of decoder is
begin
  process (sel) is
  begin
    case sel is
      when "00"   =>
        O(0) <= '1';
      when "01"   =>
        O(1) <= '1';
      when "10"   =>
        O(2) <= '1';
      when others =>
        O(3) <= '1';
    end case;
  end process;
end rtl;
```

¿Qué problema plantea este código?

¡Infiere latches!

Sentencias de selección condicional

- Solución.

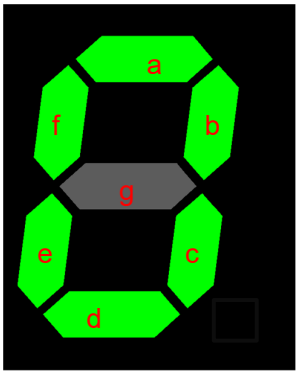
Sel[1:0]	O[3..0]
00	0001
01	0010
10	0100
11	1000

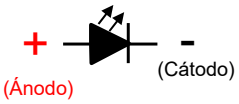
```
process (sel) is
  begin
    O <=(others=>'0');
    case sel is
      when "00"   =>
        O(0) <= '1';
      when "01"   =>
        O(1) <= '1';
      when "10"   =>
        O(2) <= '1';
      when others =>
        O(3) <= '1';
    end case;
  end process;
```

```
process (selec) is
  begin
    case selec is
      when "00"   =>
        O <= (0=>'1',others=>'0');
      when "01"   =>
        O <= (1=>'1',others=>'0');
      when "10"   =>
        O <= (2=>'1',others=>'0');
      when others =>
        O <= (3=>'1',others=>'0');
    end case;
end process;
```

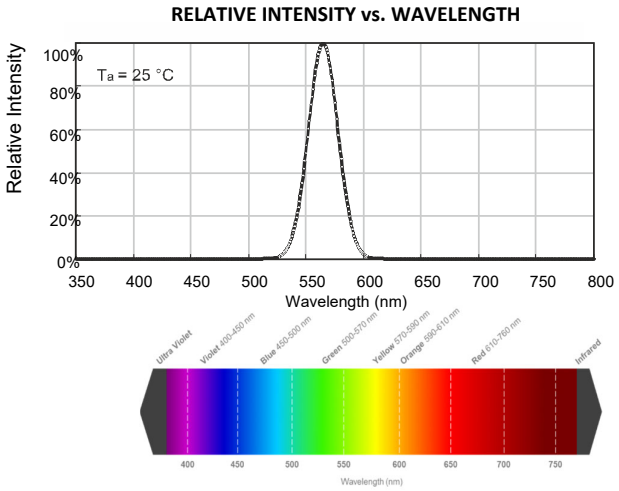
Decodificadores BCD a 7 segmentos

- Se utilizan para controlar la visualización de datos en un display de 7 segmentos
- Cada segmento (a,b,c…g) a iluminar es un diodo led.

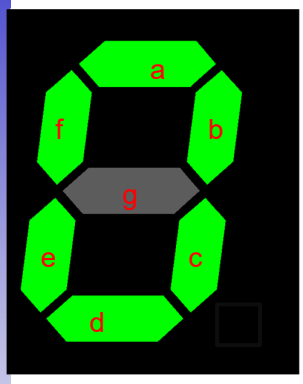


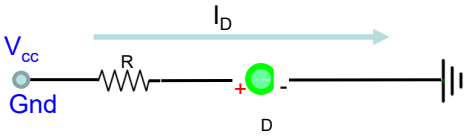


Wavelength	Colour
850-940nm	Infra-Red
630-660nm	Red
605-620nm	Amber
585-595nm	Yellow
550-570nm	Green
430-505nm	Blue
450nm	White



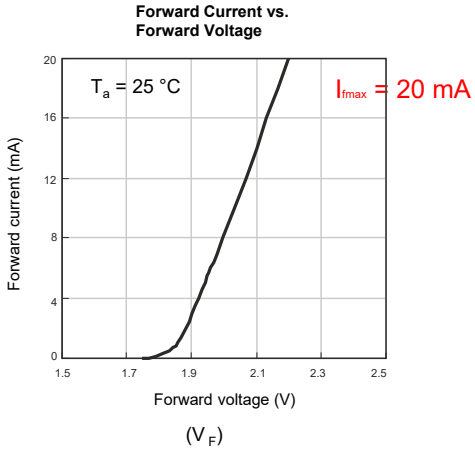
Displays

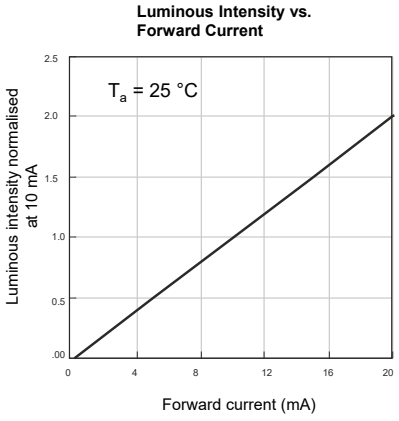




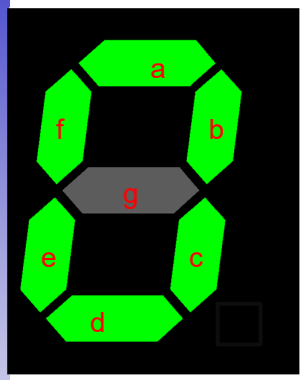
$$I_D = \frac{V_{cc} - V_F}{R}$$

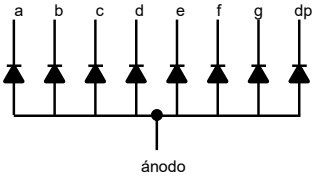
$$I_D = 0$$



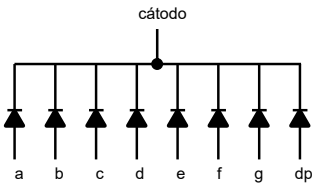


❑ Displays

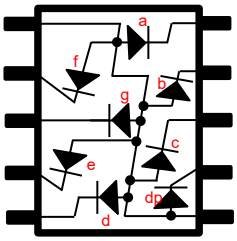




¡ El ánodo común se conectan a Vcc!

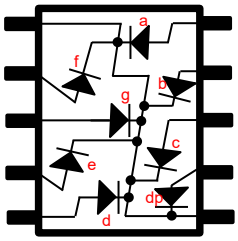


¡ El cátodo común se conectan a Gnd!



Display de ánodo común

¡Con un nivel bajo en los cátodos se ilumina el segmento!

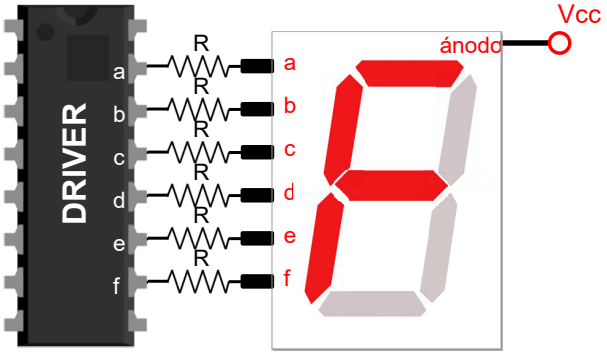


Display de cátodo común

¡Con un nivel alto en los ánodos se ilumina el segmento!

❑ Displays

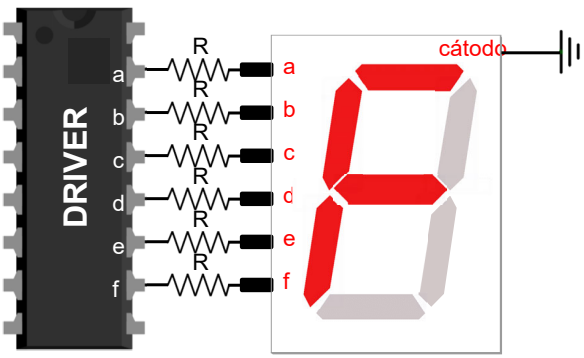
Display de ánodo común



¡Las salidas del driver son activas a nivel bajo!

¡Con un nivel bajo se ilumina el segmento!

Display de cátodo común

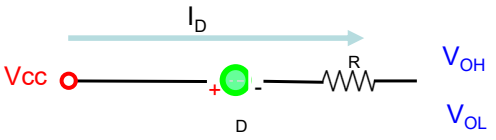


¡Las salidas del driver son activas a nivel alto!

¡Con un nivel alto se ilumina el segmento!

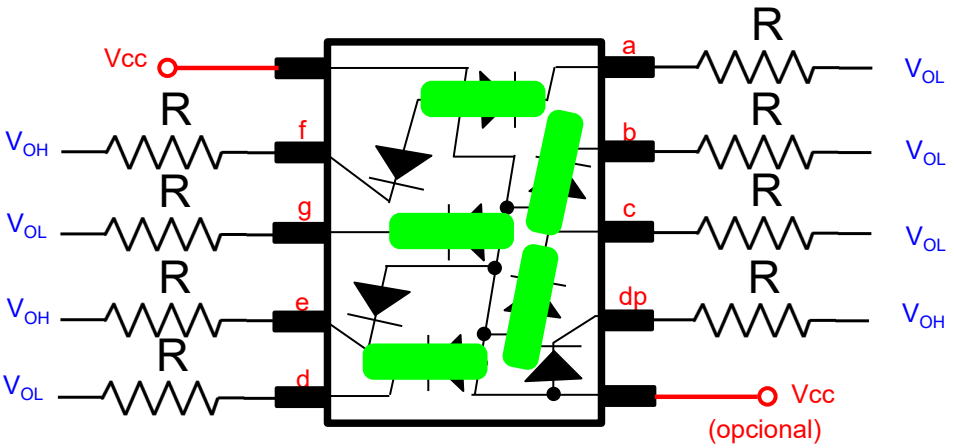
Displays

Displays de ánodo Común.



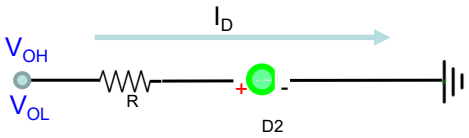
$$I_D = \frac{V_{cc} - V_{OL} - V_F}{R}$$

$$I_D = 0$$



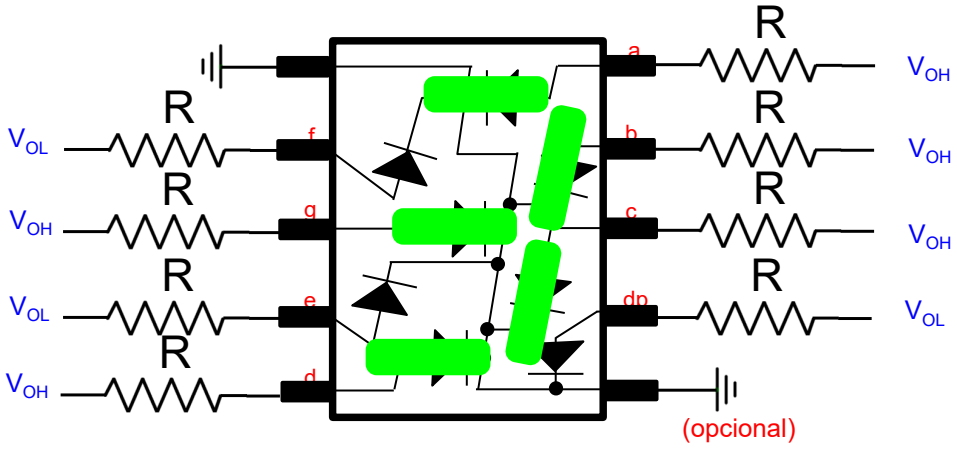
Displays

Displays de cátodo Común.



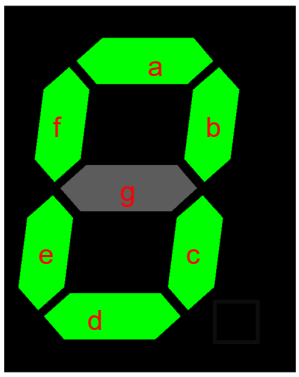
$$I_D = \frac{V_{OH} - V_F}{R}$$

$$I_D = 0$$



Sentencia CASE

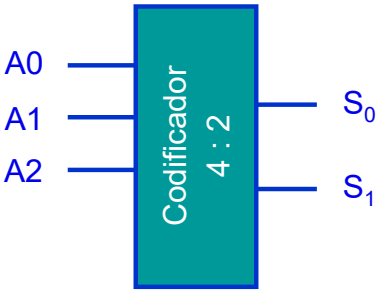
- Decodificadores BCD a 7 segmentos para un display de cátodo común




```
process (dato) is
  begin
    case dato is --gfedcba
      when x"0"   =>
        display <= "0111111";
      when x"1"   =>
        display<= "0000110";
      when x"2"   =>
        display <= "1011011";
      when x"3"   =>
        display <= "1001111";
      when x"4"   =>
        display <= "1100110";
      when x"5"   =>
        display <= "1101101";
      when x"6"   =>
        display <= "1111101";
      when x"7"   =>
        display <= "0000111";
      when x"8"   =>
        display <= "1111111";
      when x"9"   =>
        display <= "1100111";
      when x"A"   =>
        display <= "1110111";
      when x"B"   =>
        display <= "1111100";
      when x"C"   =>
        display <= "0111001";
      when x"D"   =>
        display <= "1011110";
      when x"E"   =>
        display <= "1111001";
      when x"F"   =>
        display <= "1110001";
      when others =>
        display <= "0000000";
    end case;
  end process;
```


Codificadores

- Son circuitos combinacionales que proporcionan una combinación en las salidas que permiten identificar la entrada activa.

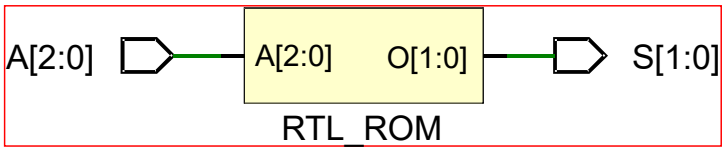


A[2..0]	S[1..0]
001	00
010	01
100	10

A[2..0]	S[1..0]
000	00
001	00
010	01
011	01
100	10
101	10
110	10
111	10

m entradas y N salidas ($2^N \geq m$)

```
process (A)  process (all)        <-- Para VHDL_08
  begin
    case A is
      when "100"|"101"|"110"|"111" =>
        S <= "10";
      when "010"|"011"             =>
        S <= "01";
      when others                  =>
        S <= "00";
    end case;
  end process;
```




```
S <= "10" when A(2) = '1' else
     "01" when A(1) = '1' else
     "00";
```

```
with A select
  S <= "10" when "100"|"101"|"110"|"111",
       "01" when "010"|"011",
       "00" when others;
```

- Modelado de un codificador con prioridad 4:2

Entradas				Salidas		
E3	E2	E1	E0	S1	S0	P
0	0	0	0	0	0	0
X	X	X	1	0	0	1
X	X	1	0	0	1	1
X	1	0	0	1	0	1
1	0	0	0	1	1	1

```
S <= "11" when E(3) = '1' else
     "10" when E(2) = '1' else
     "01" when E(1) = '1' else
     "00" when E(0) = '1' else
     "00";
P <= '1' when E(0)='1' or E(1)='1' or
               E(2) = '1' or E(3) = '1' else
               '0';
```

```
library ieee;
use ieee.std_logic_1164.all;
entity codif_4_2 is
  port(
    E : in  std_logic_vector(3 downto 0);
    P : out std_logic;
    S : out std_logic_vector(1 downto 0));
end codif_4_2;
architecture RTL of codif_4_2 is
begin
  process(E)
  begin
    if E(0) = '1' then
      P <= '1';
      S <= "00";
    elsif E(1) = '1' then
      P <= '1';
      S <= "01";
    elsif E(2) = '1' then
      P <= '1';
      S <= "10";
    elsif E(3) = '1' then
      P <= '1';
      S <= "11";
    else
      P <= '0';
      S <= "00";
    end if;
  end process;
end RTL;
```


- Modelado de un codificador con prioridad 8:3.

Entradas								Salidas			
E7	E6	E5	E4	E3	E2	E1	E0	S2	S1	S0	P
0	0	0	0	0	0	0	0	0	0	0	0
X	X	X	X	X	X	X	1	0	0	0	1
X	X	X	X	X	X	1	0	0	0	1	1
X	X	X	X	X	1	0	0	0	1	0	1
X	X	X	X	1	0	0	0	0	1	1	1
X	X	X	1	0	0	0	0	1	0	0	1
X	X	1	0	0	0	0	0	1	0	1	1
X	1	0	0	0	0	0	0	1	1	0	1
1	0	0	0	0	0	0	0	1	1	1	1

```
with E select?
  S <= 3d"0" when "-------1",
       3d"1" when "------10",
       3d"2" when "-----100",
       3d"3" when "----1000",
       3d"4" when "---10000",
       3d"5" when "--100000",
       3d"6" when "-1000000",
       3d"7" when "10000000",
       3d"0"when others;

P <= '0' when E = 8d"0" else '1';
```

```
P<= or E;
```

VHDL-2008 aporta una nueva forma de la sentencia case llamada *matching case* que utiliza el operador ?=.

El operador ?= permite comparar elementos *std_ulogic* y todos los tipos derivados.

```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;
entity codif_38 is
port(E: in std_logic_vector(7 downto 0);
     S: out std_logic_vector(2 downto 0);
     P: out std_logic);
end codif_38;

architecture RTL of codif_38 is
begin
process(all)  begin
    P<='0';
    case ? E is
        when "-------1" => S<=3d"0"; P<='1';
        when "------10" => S<=3d"1"; P<='1';
        when "-----100" => S<=3d"2"; P<='1';
        when "----1000" => S<=3d"3"; P<='1';
        when "---10000" => S<=3d"4"; P<='1';
        when "--100000" => S<=3d"5"; P<='1';
        when "-1000000" => S<=3d"6"; P<='1';
        when "10000000" => S<=3d"7"; P<='1';
        when others     => S<=3d"0";
     end case?;
end process;
end RTL;
```

- Modelado de un codificador con prioridad 8:3.

Entradas								Salidas			
E7	E6	E5	E4	E3	E2	E1	E0	S2	S1	S0	P
0	0	0	0	0	0	0	0	0	0	0	0
X	X	X	X	X	X	X	1	0	0	0	1
X	X	X	X	X	X	1	0	0	0	1	1
X	X	X	X	X	1	0	0	0	1	0	1
X	X	X	X	1	0	0	0	0	1	1	1
X	X	X	1	0	0	0	0	1	0	0	1
X	X	1	0	0	0	0	0	1	0	1	1
X	1	0	0	0	0	0	0	1	1	0	1
1	0	0	0	0	0	0	0	1	1	1	1

¿Qué sucede si se pone?

```
for i in 0 to 7 loop
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity codificador_for is
  port(
    E : in  std_logic_vector(7 downto 0);
    P : out std_logic;
    S : out std_logic_vector(2 downto 0));
end codificador_for;
architecture RTL of codificador_for is
begin
  process(E)
  begin
    P     <= '0';
    S     <= "000";
    for j in 7 downto 0 loop
      if E(j) = '1' then
        P <= '1';
        S <= std_logic_vector(to_unsigned(j, 3));
      end if;
    end loop;
  end process;
end RTL;
```


- Permite ejecutar un grupo de sentencias secuenciales de forma repetitiva.
  - La sentencia básica es ***loop***
  - Hay tres formas de crear un bucle:
    - Sólo con ***loop***, siempre se obtiene un bucle infinito (no es sintetizable)
    - Con ***for***, siempre se obtiene un bucle finito (es sintetizable)
    - Con ***while*** el bucle puede ser finito o infinito según la sentencia asociada.

```
[etiqueta:][for control_repeticion][while condición] loop
                secuencia_de_sentencias
            end loop  [etiqueta];
```

Bucle *for*:

Permite ejecutar un bucle un número fijo de veces.

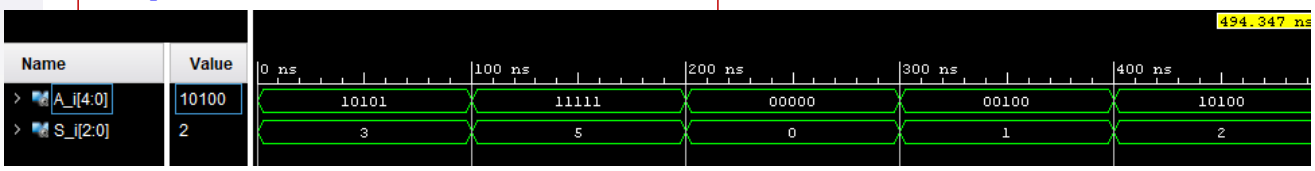
```
[ etiqueta:] for indice in rango loop
                secuencia_de_sentencias
             end loop  [ etiqueta ];
```

El índice es una variable entera que no se declara y que cambia en una unidad por cada ejecución del bucle a lo largo del rango especificado.

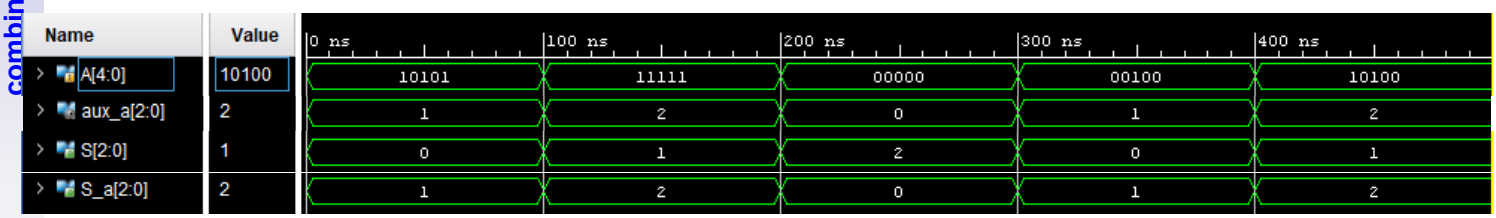
```
architecture rtl of num_1s_a is

  signal A : std_logic_vector(4 dowto 0);
  signal S : std_logic_vector(2 dowto 0);
. . .
begin

process (A)                          -- process (all)   Para VHDL-2008
  variable aux_a : unsigned(2 downto 0);
begin
   aux_a := (others=>'0');
   for j in 4 downto 0 loop
     if A(j)='1' then
       aux_a:=aux_a+1;
     end if;
   end loop;  -- j
   S<=std_logic_vector(aux_a);
end process;
```

¿Qué sucede si se pone?

```
for j in 0 to 4 loop
```

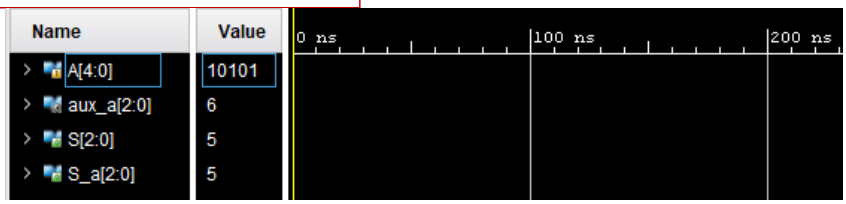



```
architecture rtl of num_1s_b is
 signal aux_a : unsigned(2 downto 0);
begin  -- rtl

process (A)
begin
   aux_a <= (others=>'0');
   for j in 4 downto 0 loop
     if A(j)='1' then
       aux_a<=aux_a+1;
     end if;
   end loop;  -- j
   S<=std_logic_vector(aux_a);
end process;
  S_a<=std_logic_vector(aux_a);

end rtl;
```



```
architecture rtl of num_1s_b is
 signal aux_a : unsigned(2 downto 0);
begin  -- rtl

process (A,aux_a)          process (all)
begin
   aux_a <= (others=>'0');
   for j in 4 downto 0 loop
     if A(j)='1' then
       aux_a<=aux_a+1;
     end if;
   end loop;  -- j
   S<=std_logic_vector(aux_a);
end process;
  S_a<=std_logic_vector(aux_a);

end rtl;
```

Para VHDL-2008




```
Vivado Simulator 2017.4
Time resolution is 1 ps
FATAL_ERROR: Iteration limit 10000 is reached. Possible zero delay oscillation detected where simulation time can not advance. Please check your source code.
Time: 0 ps  Iteration: 10000
```


- Para que sea sintetizable, El rango debe ser estático;

```
signal A : std_logic_vector(3 downto 0);
signal S : std_logic_vector(15 downto 0);
…

  process (A)          process (all)
   begin  -- process
     S <= (others=>'0');
     for J in 0 to to_integer(unsigned(A)) loop
         S(J) <= '1';
     end loop;
  end process;
```

Para VHDL-2008

a	0011	1010	1001	1111	0000
s	0000000000001111	0000011111111111	0000001111111111	1111111111111111	0000000000000001

¡ El código es válido, pero no para síntesis!

Atributos

- Es una característica que se puede asociar a un elemento del lenguaje:
  - Tipos, subtipos, procedimientos, funciones, señales, variables, constantes, entidades, arquitecturas, configuraciones, paquetes y componentes.
- Proporcionan información de esos elementos
- Atributo de un elemento ≠ valor de ese elemento.
- Se utilizan para:
  - Mejorar la portabilidad del código.
  - Controlar el proceso de síntesis.
- Pueden ser:
  - Atributos predefinidos
  - Definibles por el usuario

Atributos

- Atributos de rango de vectores.

  Todos son sintetizables

Atributo	Descripción
A'left(n)	Valor izquierdo del rango n del vector A
A'right(n)	Valor derecho del rango n del vector A
A'low(n)	Valor mínimo del rango n del vector A
A'high(n)	Valor máximo del rango n del vector A
A'ascending(n)	Verdadero si el rango n del vector A es ascendente
A'range(n)	Rango n del vector A
A'regverse_range(n)	Rango n del vector A invertido
A'length(n)	Número de valores del rango n del vector A

- Atributos de tipo vector.

  Algunos son sintetizables

Atributo	Descripción
T'base	Tipo base de T.
T'left	Valor más a la izquierda de T.
T'right	Valor más a la derecha de T.
T'low	Valor mínimo de T.
T'high	Valor máximo de T.
T'ascending	Verdadero si T tiene rango ascendente.
T'image(x)	Representación textual del valor x del tipo T.
T'value(x)	Valor expresado por la cadena de caracteres.
T'pos(x)	Posición ocupada por x en T.
T'val(x)	Valor de la posición x en T.
T'succ(x)	Valor de la posición siguiente a x en T.
T'pred(x)	Valor de la posición anterior a x en T.
T'leftof(x)	Valor de la posición derecha a x en T.

Atributos

```
signal A : std_logic_vector(3 downto 0);
```

```
process (A)
  variable aux_a : unsigned(2 downto 0);
begin
   aux_a := (others=>'0');
   for j in 3 downto 0 loop
     if A(j)='1' then
       aux_a:=aux_a+1;
     end if;
   end loop;  -- j
   S<=std_logic_vector(aux_a);
end process;
```

```
process (A)
  variable aux_a : unsigned(2 downto 0);
begin
   aux_a := (others=>'0');
   for j in A'range loop
     if A(j)='1' then
       aux_a:=aux_a+1;
     end if;
   end loop;  -- j
   S<=std_logic_vector(aux_a);
end process;
```

¿Cuáles son más versátiles?

```
signal S : std_logic_vector(2 downto 0);
```

```
process (A)
  variable aux_a : integer;
begin
 aux_a := 0;
  for j in A'range loop
    if A(j)='1' then
      aux_a:=aux_a+1;
    end if;
  end loop;  -- j
 S<=std_logic_vector(to_unsigned(aux_a,3));
end process;
```

```
process (A)
  variable aux_a : integer;
begin
 aux_a := 0;
  for j in A'range loop
    if A(j)='1' then
      aux_a:=aux_a+1;
    end if;
  end loop;  -- j
 S<=std_logic_vector(to_unsigned(aux_a,s'length));
end process;
```


Atributos

- Atributos de señales.

Atributo	Descripción
S'delayed(t)	Señal S demorada t unidades de tiempo.
S'stable(t)	Señal booleana verdadera si S es estable hace t unidades de tiempo.
S'quiet(t)	Señal booleana verdadera si no ha habido ninguna asignación a S es desde t unidades de tiempo.
S'transaction	Señal de tipo bit, vale '1' cuando hay una asignación a S.
S'event	Verdadero si ocurre un evento en S.
S'active	Verdadero si ocurre una asignación sobre S.
S'last_event	Unidades de tiempo desde el último evento en S.
S'last_active	Unidades de tiempo desde la última asignación sobre S.
S'last_value	Valor anterior de S.
S'driving	Verdadero si el proceso actual excita la señal S.
S'driving_value	Valor inyectado por el proceso actual sobre la señal S.

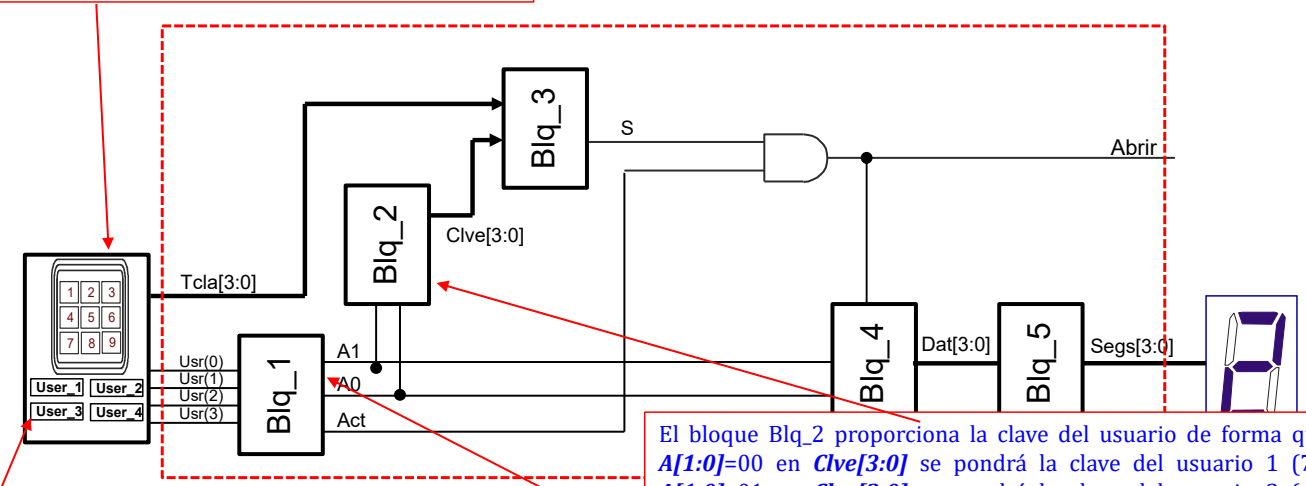
```
if clk'event and clk = '1' then
```

Ejemplo I

Crear el código VHDL que modele un sistema digital que realiza el control de apertura de una puerta mediante dos datos: usuario y clave seleccionados con sendos teclados.

Teclado numérico (1 a 9) : permiten introducir la clave de usuario, de un solo dígito. Entregan en la salida ***Tcla[3:0]*** el código binario correspondiente a la tecla pulsada mientras que ésta está presionada. En caso de que no se pulse ninguna tecla estas salidas están a nivel bajo.

Para que un usuario pueda abrir la puerta deberá pulsar simultáneamente su tecla de usuario, ***User_i,*** y su **clave** de usuario. De esa manera se activará a nivel alto la salida ***Abrir*** y esta señal permitirá abrir la puerta.

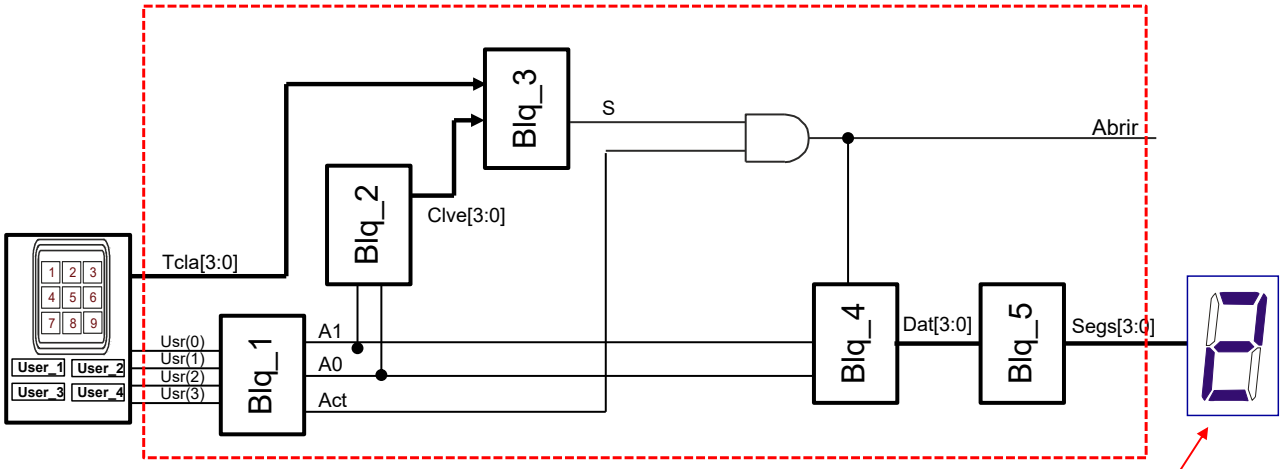


Teclas ***User_1***, ***User _2***, ***User _3*** y ***User _4***: Identifican el usuario que quiere acceder. Al presionar ***User_1*** la salida ***Usr*** toma el valor 1110, si se presiona ***User_2 Usr*** toma el valor 1101, si se presiona ***User_3 Usr*** toma el valor 1011 y si se presiona ***User_4 Usr*** toma el valor 0111
Cada vez que se pulsa una tecla de usuario, ***Act*** se pone a nivel alto

El bloque Blq_2 proporciona la clave del usuario de forma que si ***A[1:0]***=00 en ***Clve[3:0]*** se pondrá la clave del usuario 1 (7), si ***A[1:0]***=01 en ***Clve[3:0]*** se pondrá la clave del usuario 2 (4), si ***A[1:0]***=10 en ***Clve[3:0]*** se pondrá la clave del usuario 3 (6) y si ***A[1:0]***=11 en ***Clve[3:0]*** se pondrá la clave del usuario 4 (5).

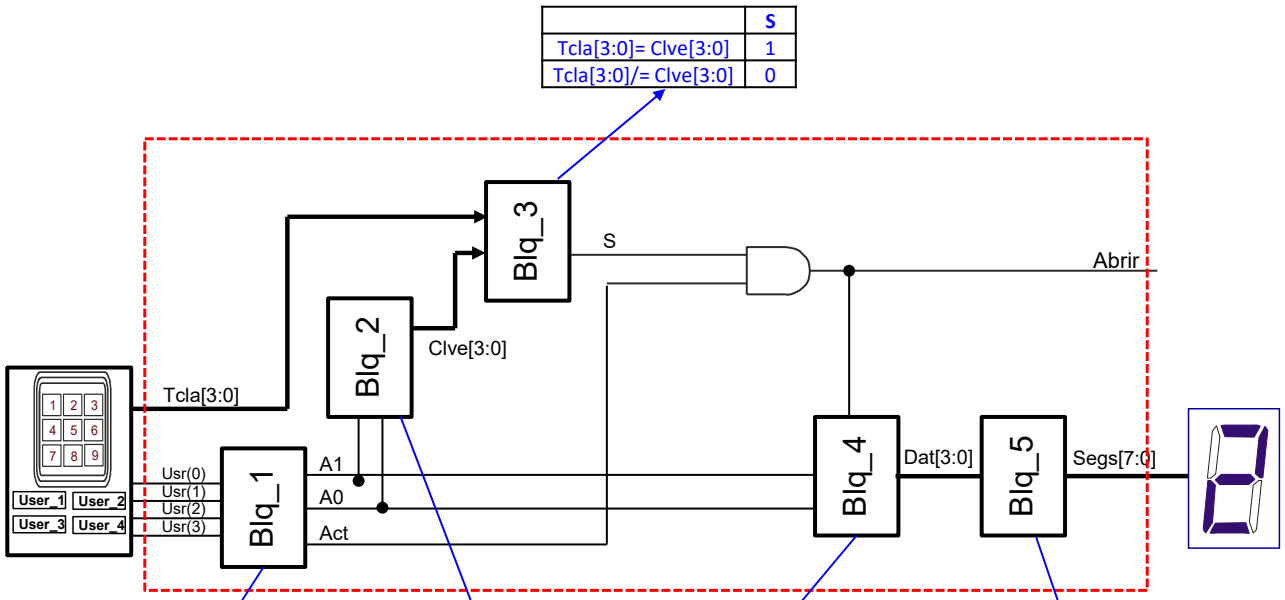
El bloque Blq_1 proporciona un código binario que informa de la tecla ***User_i*** activada. Si se pulsa ***User_1*** el valor de ***A[1:0]*** será 1, 2 si se pulsa ***User_2***, 3 si lo es ***User_3*** y 4 si lo es ***User_4***. Por su parte ***Act*** se pone a nivel alto para indicar que se ha pulsado una tecla ***User_i*** y a 0 en caso contrario.

Ejemplo I



Cuando Abrir se pone a nivel alto, en el display, de ánodo común, se muestra el número del usuario, en caso contrario el display permanecerá apagado.

Ejemplo I



	S
Tcla[3:0]= Clve[3:0]	1
Tcla[3:0]/= Clve[3:0]	0

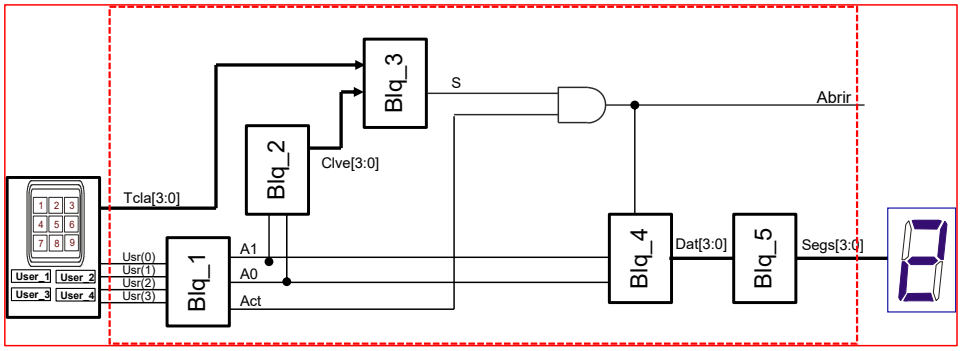
Usr[3:0]	A[1:0]	Act
1110	00	1
1101	01	1
1011	10	1
0111	11	1
Resto	00	0

A[1:0]	Clve[3:0]
00	0111
01	0100
10	0110
11	0101

Abrir	A[1:0]	Dat[3:0]
1	00	0001
1	01	0010
1	10	0011
1	11	0100
0	XX	1111

Dat[3:0]	Segs[6:0]
0001	1111001
0010	0100100
0011	0110000
0100	0011001
resto	1111111

Ejemplo I



Para el tipo std_logic

Para hacer operaciones aritméticas

Entadas Y salidas

Señales para interconectar los bloques

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

entity ejemplo_CC is
  port( Tcla            : in  std_logic_vector(3 downto 0);
        Usr             : in  std_logic_vector(3 downto 0);
        Segs            : out std_logic_vector(6 downto 0);  --gf..a
        Abrir           : out std_logic);
end;
architecture rtl of ejemplo_CC is
  signal   A            :     std_logic_vector(1 downto 0);
  signal   Act, S, aux  :     std_logic;
  signal   Clve, Dat    :     std_logic_vector(3 downto 0);
  constant clv_U1       :     std_logic_vector(3 downto 0) := x"7";
  constant clv_U2       :     std_logic_vector(3 downto 0) := x"4";
  constant clv_U3       :     std_logic_vector(3 downto 0) := x"6";
  constant clv_U4       :     std_logic_vector(3 downto 0) := x"5";
begin
```

Ejemplo I

Usr[3:0]	A[1:0]	EN
1110	00	1
1101	01	1
1011	10	1
0111	11	1
Resto	00	0

```
--BLq_1 (codificador)
  process (Usr)
  begin  -- process
    case Usr is
      when "1110" =>
        A   <= "00";
        Act <= '1';
      when "1101" =>
        A   <= "01";
        Act <= '1';
      when "1011" =>
        A   <= "10";
        Act <= '1';
      when "0111" =>
        A   <= "11";
        Act <= '1';
      when others =>
        A   <= "00";
        Act <= '0';
    end case;
  end process;
```

	S
Usr[3:0]= Usr[3:0]	1
Usr[3:0]/= Usr[3:0]	0

```
--BLq_3 (comparador)
  S      <= '1'    when Clve = Tcla else '0';
```

A[1:0]	Clve[3:0]
00	0111
01	0100
10	0110
11	0101

```
--BLq_2 (multiplexor)
  with a select
    Clve <= clv_U1 when "00",
    clv_U2         when "01",
    clv_U3         when "10",
    clv_U4         when "11",
    x"0"           when others;
```


Ejemplo I

Abrir	A[1:0]	Dat[3:0]
1	00	0001
1	01	0010
1	10	0011
1	11	0100
0	XX	1111

```
--BLq_4 (Circ Combinacional)
  Dat<= std_logic_vector(unsigned("00"&A)+1)when aux = '1' else (others => '1');
```

Dat[3:0]	Segs[6:0]
0001	1111001
0010	0100100
0011	0110000
0100	0011001
resto	1111111

```
--BLq_5 (Decodificador a 7 segmentos)
  process (Dat) is
  begin
    case Dat is                 --gfedcba
      when x"1"=>    Segs <= "1111001";
      when x"2"=>    Segs <= "0100100";
      when x"3"=>    Segs <= "0110000";
      when x"4"=>    Segs <= "0011001";
      when others=> Segs <= "1111111";
    end case;
  end process;
end rtl;
```

Ejemplo I

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

entity ejemplo_CC is
  port( Tcla            : in  std_logic_vector(3 downto 0);
        Usr             : in  std_logic_vector(3 downto 0);
        Segs            : out std_logic_vector(6 downto 0);
        Abrir           : out std_logic);
end;
architecture rtl of ejemplo_CC is
  signal   A            :std_logic_vector(1 downto 0);
  signal   Act, S, aux  :std_logic;
  signal   Clve, Dat    :std_logic_vector(3 downto 0);
  constant clv_U1:std_logic_vector(3 downto 0) := x"7";
  constant clv_U2:std_logic_vector(3 downto 0) := x"4";
  constant clv_U3:std_logic_vector(3 downto 0) := x"6";
  constant clv_U4:std_logic_vector(3 downto 0) := x"5";
begin

--BLq_1 (codificador)
  process (Usr)
  begin  -- process
    case Usr is
      when "1110" =>
        A   <= "00";
        Act <= '1';
      when "1101" =>
        A   <= "01";
        Act <= '1';
      when "1011" =>
        A   <= "10";
        Act <= '1';
      when "0111" =>
        A   <= "11";
        Act <= '1';
      when others =>
        A   <= "00";
        Act <= '0';
    end case;
  end process;
```

```
 --BLq_2 (multiplexor)
  with a select
    Clve <= clv_U1 when "00",
    clv_U2         when "01",
    clv_U3         when "10",
    clv_U4         when "11",
    x"0"           when others;
  --BLq_3 (comparador)
  S      <= '1'when Clve = Tcla else '0';

  aux    <= S and Act;
  Abrir <= aux;

 --BLq_4 (Circ Combinacional)
  Dat<= std_logic_vector( unsigned("00"&A)+1)
        when aux = '1' else (others => '1');

  --BLq_5 (Decodificador a 7 segmentos)
  process (Dat) is
  begin
    case Dat is              --gfedcba
      when x"1"=> Segs    <= "1111001";
      when x"2"=> Segs    <= "0100100";
      when x"3"=> Segs    <= "0110000";
      when x"4"=> Segs    <= "0011001";
      when others=> Segs <= "1111111";
    end case;
  end process;
end rtl;
```


Ejemplo I

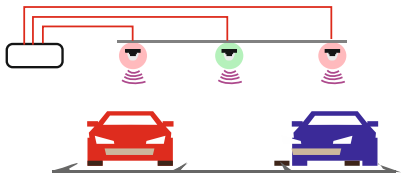
- Con VHDL-08

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

entity ejemplo_CC is
  port( Tcla            : in  std_logic_vector(3 downto 0);
        Usr             : in  std_logic_vector(3 downto 0);
        Segs            : out std_logic_vector(6 downto 0);
        Abrir           : out std_logic);
end;
architecture rtl of ejemplo_CC is
  signal   A            :std_logic_vector(1 downto 0);
  signal   Act, S       :std_logic;
  signal   Clve, Dat    :std_logic_vector(3 downto 0);
  constant clv_U1:std_logic_vector(3 downto 0) := x"7";
  constant clv_U2:std_logic_vector(3 downto 0) := x"4";
  constant clv_U3:std_logic_vector(3 downto 0) := x"6";
  constant clv_U4:std_logic_vector(3 downto 0) := x"5";
begin

--BLq_1 (codificador)
  process (all)
  begin  -- process
    case Usr is
      when "1110" =>
        A   <= "00";
        Act <= '1';
      when "1101" =>
        A   <= "01";
        Act <= '1';
      when "1011" =>
        A   <= "10";
        Act <= '1';
      when "0111" =>
        A   <= "11";
        Act <= '1';
      when others =>
        A   <= "00";
        Act <= '0';
    end case;
  end process;
```

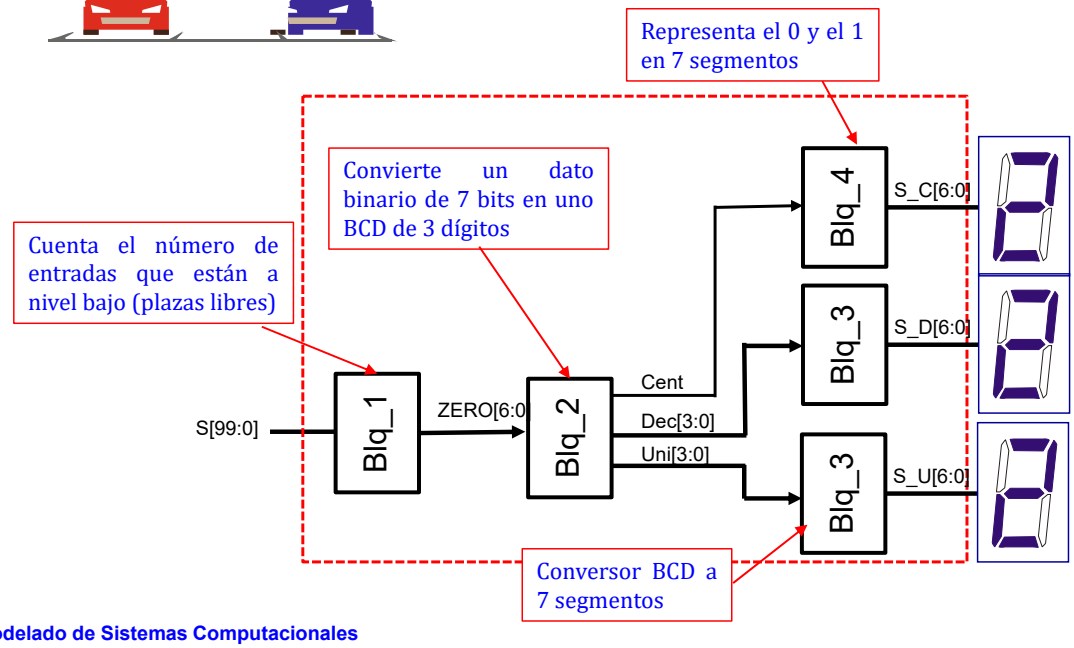
```
 --BLq_2 (multiplexor)
  with a select
    Clve <= clv_U1 when "00",
    clv_U2         when "01",
    clv_U3         when "10",
    clv_U4         when "11",
    x"0"           when others;
--BLq_3 (comparador)
  S      <=  Clve ?= Tcla ;
  Abrir   <= S and Act;
 --BLq_4 (Circ Combinacional)
  Dat<= std_logic_vector(unsigned("00"&A)+1)
        when Abrir ='1' else (others =>'1');

  --BLq_5 (Decodificador a 7 segmentos)
  process (all) is
  begin
    case Dat is              --gfedcba
      when x"1"=> Segs    <= "1111001";
      when x"2"=> Segs    <= "0100100";
      when x"3"=> Segs    <= "0110000";
      when x"4"=> Segs    <= "0011001";
      when others=> Segs <= "1111111";
    end case;
  end process;
end rtl;
```

Ejemplo II

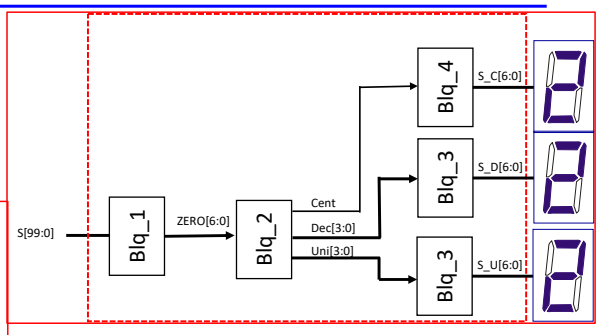
Se pretende realizar un sistema que permite determinar el número de plazas libres de un aparcamiento y visualizarlas en 3 displays, siendo el número de plazas es 100.



Para determinar si una plaza de aparcamiento está libre u ocupada, en los sistemas comerciales, cada plaza tiene instalado un sensor de presencia ultrasónico que proporciona un nivel alto cuando está ocupada y un nivel bajo cuando está libre.



Ejemplo II




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

entity ejemplo_CC2 is
  port( S          : in  std_logic_vector(99 downto 0);
        S_U        : out std_logic_vector(6 downto 0);
        S_D        : out std_logic_vector(6 downto 0);
        S_C        : out std_logic_vector(6 downto 0));
end;
architecture rtl of ejemplo_CC2 is

  signal ZEROS    :      std_logic_vector(6 downto 0);
  signal Cent     :      std_logic;
  signal Uni, Dec :      std_logic_vector(3 downto 0);

Begin

--BLq_1 (contador de ceros)

 process (S)
    variable cuenta : unsigned(6 downto 0);
   begin  -- process
     cuenta      := (others => '0');
     for j in s'range loop
       if S(j) = '0' then
         cuenta := cuenta+1;
       end if;
     end loop;  -- j
     ZEROS <= std_logic_vector(cuenta);
   end process;
```

Ejemplo II

- Conversión de binario a BCD. Algoritmo de suma y desplazamiento

Paso	Operación	Centenas	Decenas	Unidades	Dato Binario
					1 1 1 1 1 1 1 1
1	Desplazar 1			1	1 1 1 1 1 1 1 0
2	Desplazar 2			1 1	1 1 1 1 1 1 0 0
3	Desplazar 3			1 1 1	1 1 1 1 1 0 0 0
4	Sumar 3			1 0 1 0	1 1 1 1 1 0 0 0
	Desplazar 4		1	0 1 0 1	1 1 1 1 0 0 0 0
5	Sumar 3		1	1 0 0 0	1 1 1 1 0 0 0 0
	Desplazar 5		1 1	0 0 0 1	1 1 1 0 0 0 0 0
6	Desplazar 6		1 1 0	0 0 1 1	1 1 0 0 0 0 0 0
7	Sumar 3		1 0 0 1	0 0 1 1	1 1 0 0 0 0 0 0
	Desplazar 7	1	0 0 1 0	0 1 1 1	1 0 0 0 0 0 0 0
8	Sumar 3	1	0 0 1 0	1 0 1 0	1 0 0 0 0 0 0 0
	Desplazar 8	**1 0**	**0 1 0 1**	**0 1 0 1**	
		2	**5**	**5**	

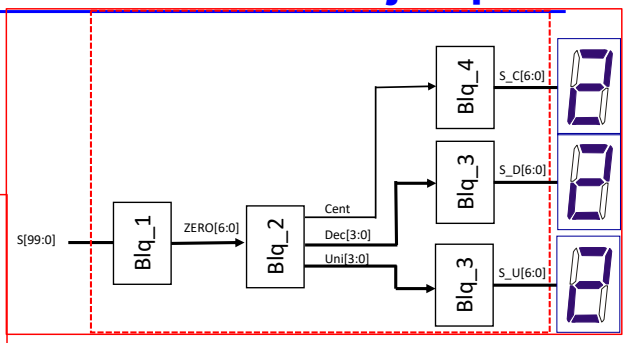
1. Se determina el número de columnas de 4 bits que debe tener el dato BCD resultante.
2. Si el valor de alguna de las columnas es mayor o igual a 5 se le suma 3.
3. Se desplaza un bit a la izquierda.
4. Si el número de desplazamientos realizados es igual al número de bits del dato binario se da por finalizada la conversión y el dato BCD es el mostrado en las columnas del dato BCD.

Ejemplo II

```
--BLq_2 (conversor binario BCD)

process (ZEROS)
    variable aux : std_logic_vector(15 downto 0);
  begin  -- process
    aux                    := (others => '0');
    aux(6 downto 0)        := ZEROS;
    for i in 1 to 7 loop
      if unsigned(aux(10 downto 7)) > 4 then
        aux(10 downto 7)   := std_logic_vector(unsigned(aux(10 downto 7))+3);
      end if;
      if unsigned(aux(14 downto 11)) > 4 then
        aux(14 downto 11) := std_logic_vector(unsigned(aux(14 downto 11))+3);
      end if;
       aux                  := aux(14 downto 0)&'0';
    end loop;  -- i
    Uni  <= aux(10 downto 7);
    Dec  <= aux(14 downto 11);
    Cent <= aux(15);
  end process;
```

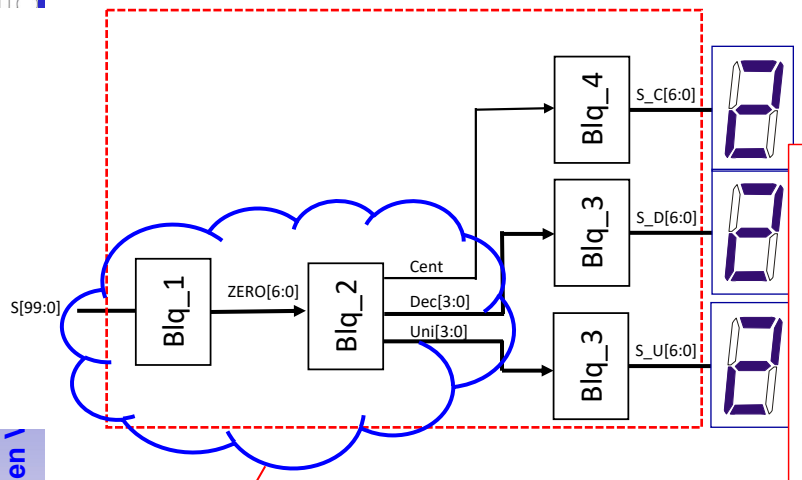
1. Se determina el número de columnas de 4 bits que debe tener el dato BCD resultante.
2. Si el valor de alguna de las columnas es mayor o igual a 5 se le suma 3.
3. Se desplaza un bit a la izquierda.
4. Si el número de desplazamientos realizados es igual al número de bits del dato binario se da por finalizada la conversión y el dato BCD es el mostrado en las columnas del dato BCD.

Ejemplo II




```
--BLq_3 (Decodificador a 7 segmentos)

 with Uni select
   S_U <= "1000000" when "0000",   --0   --gfedcba
   "1111001"        when "0001",         --1
   "0100100"        when "0010",         --2
   "0110000"        when "0011",         --3
   "0011001"        when "0100",         --4
   "0010010"        when "0101",         --5
   "0000010"        when "0110",         --6
   "1111000"        when "0111",         --7
   "0000000"        when "1000",         --8
   "0011000"        when "1001",         --9
   "0110110"        when others;

 with Dec select
   S_D <= "1000000" when "0000",   --0   --gfedcba
   "1111001"        when "0001",         --1
   "0100100"        when "0010",         --2
   "0110000"        when "0011",         --3
   "0011001"        when "0100",         --4
   "0010010"        when "0101",         --5
   "0000010"        when "0110",         --6
   "1111000"        when "0111",         --7
   "0000000"        when "1000",         --8
   "0011000"        when "1001",         --9
   "0110110"        when others;

 --BLq_4 (Decodificador a 7 segmentos  1 bit)

 S_C <= "1000000" when Cent='0' else "1111001" ;

end rtl;
```


Ejemplo II



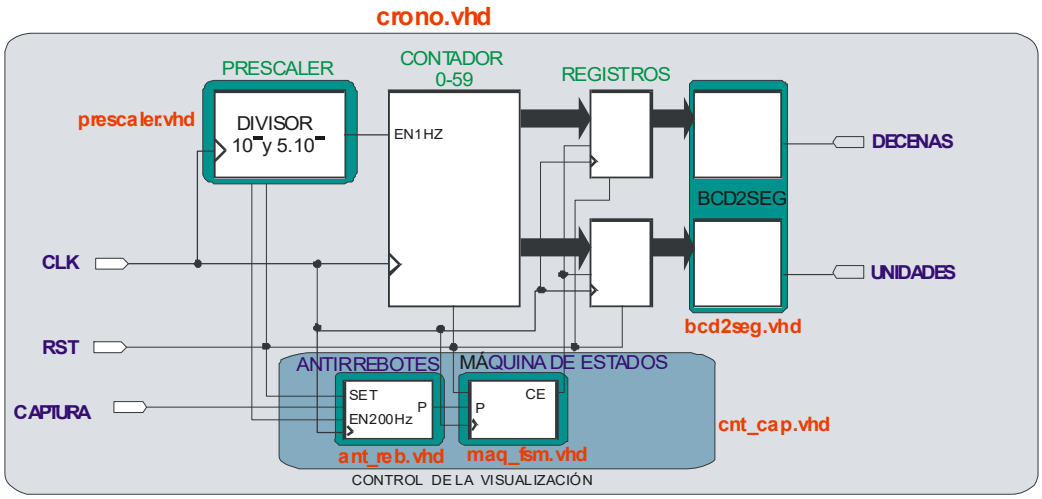
¿Por qué contar en binario y después pasar a BCD?

¡Se cuenta en BCD!

```
architecture rtl of ejemplo_CC2 is
 signal Cent     :     std_logic;
 signal Uni, Dec :     unsigned(3 downto 0);

begin

  --BLq_1-2 (contador de ceros en BCD)
process (S)
  variable cuenta : unsigned(8 downto 0);
begin  -- process
 cuenta                        := (others => '0');
 for j in s'range loop
  if S(j) = '0' then
   if cuenta(3 downto 0) = 9 then
    cuenta(3 downto 0)    := (others => '0');
    if cuenta(7 downto 4) = 9 then
     cuenta(7 downto 4)  := (others => '0');
     cuenta(8)           := '1';
    else
     cuenta(7 downto 4)  := cuenta(7 downto 4)+1;
    end if;
   else
    cuenta(3 downto 0)    := cuenta(3 downto 0)+1;
   end if;
  end if;
 end loop;  -- j
  Uni  <= cuenta(3 downto 0);
  Dec  <= cuenta(7 downto 4);
  Cent <= cuenta(8);
end process;
```

Descripción estructural

- Describe la entidad como un conjunto de componentes conectados entre sí por señales.
- Un componente es una entidad previamente elaborada (compilada).
- Esto forma la base de un sistema de diseño jerárquico.
- Pueden coexistir con otros tipos de descripciones (algorítmico, flujo de datos)
- Proporcionan un mecanismo para particionar un diseño:
  - Favorecen la legibilidad de los diseños.
  - Favorece la simplificación de los diseños.



Componentes.

- Representa una entidad declarada en un diseño o librería previamente compilada.
- Hay que realizar dos acciones, una en VHDL-93:
  - Declaración:
  - Instanciación.
- La instanciación es la sentencia concurrente, En ella se especifica la interconexión de las señales del componente dentro del diseño en el que está siendo utilizado.

Declaración

```
component identificador [is]
    [ generic (lista_genéricos);]
    [ port (lista_puertos);]
end component [identificador];
```

Instanciación

```
etiqueta: nombre_componente
        [ genenic map lista_asociación];
        [ port map lista_asociación];
```

Para VHDL-87

```
etiqueta: entity libreria.nombre_componente
                    [ genenic map lista_asociación];
                    [ port map lista_asociación];
```

Para VHDL-93

decodificador.vhd

```
entity decodificador is
port(
  S: in std_logic_vector(3 downto 0);
  I: in std_logic;
  Z: out std_logic_vector(7 downto 0));
end entity;
```

raiz.vhd

```
architecture RTL of raiz is
---
component decodificador
 port(
  S:in std_logic_vector(3 downto 0);
  I:in std_logic;
  Z:out std_logic_vector(7 downto 0));
end component;
---
begin
```

```
ref1: decodificador
            port map (
                 S => S_i,
                 I => I_i,
                 Z => Z_i);
```

VHDL-87

```
ref1:entity work.decodificador
            port map (
                 S => S_i,
                 I => I_i,
                 Z => Z_i);
```

VHDL-93

- La asociación (conexión) de puertos y genéricos puede realizarse:
  - **Por identificador**: En cada elemento de la lista de asociación se indica el nombre del puerto y el objeto a que se conecta: *puerto => objeto*
  - **Por posición**: Se indica la lista de elementos a que se conecta siguiendo el orden en que fueron declarados los puertos (o genéricos).

```
entity dec_counter is
 port (
  clk   : in  std_logic;
  rst   : in  std_logic;
  cnt   : out std_logic_vector(3 downto 0));
end dec_counter;
```

```
U1 : dec_counter
    port map (clock, reset, cnt_i);
```

Asociación por posición: el orden es importante

```
U1 : dec_counter
    port map (
      rst => reset,
      clk => clock,
      cnt => cnt_i);
```

Asociación por identificador: el orden es irrelevante

¡Esto también es aplicable a los genéricos!

```
ibrary ieee;use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity decoder is
    generic(n:integer:=3;
            m:integer:=8);
    port(a:in std_logic_vector(n-1 downto 0);
         y: out std_logic_vector(m-1 downto 0));
end decoder;
```

```
architecture rtl of diseno is

    constant cte_1:natural:=8;
    constant cte_2:natural:=256;
    signal sgn_1  :std_logic_vector(N-1 downto 0);
    signal sgn_2  :std_logic_vector(M-1 downto 0);

Begin

 UX: entity work.decoder
             generic map (n=>cte_1,
                          m=>cte_2)
             port map(a=> sgn_1,
                      y=> sgn_2);
```


- En la instanciación de un componente, a un puerto se le puede asignar:
  - Un puerto de la entidad en la que se instancia
  - Una señal.
  - Un valor fijo.
  - El valor ***open*** (no conectar).

```
entity cmp is
port(P1: in std_logic_vector(3 downto 0);
     P2: in std_logic_vector(3 downto 0);
     P3: out std_logic_vector(3 downto 0);
     P4: out std_logic_vector(3 downto 0);
     P5: out std_logic;
     P6: out std_logic_vector(3 downto 0));
end cmp;
```

```
entity diseno is
  port( IN_1 : in  std_logic_vector(3 downto 0);
        IN_2 : in  std_logic_vector(3 downto 0);
        OUT_1: out std_logic_vector(3 downto 0);
        OUT_2: out std_logic_vector(6 downto 0);
        OUT_3: out std_logic_vector(6 downto 0));
end;
architecture rtl of diseno is
  signal S1:     std_logic;
  signal S2:     unsigned(3 downto 0);
begin
. . .
UX: entity work.cmp
     port map(P1=> IN_1,
              P2=> x"3",
              P3=> open,
              P4=> OUT_2(6 downto 3),
              P5=> S1,
              P6=> std_logic_vector(S2);
. . .
```

```
. . .
P3=> IN_1,
. . .
```
¡Error!

```
. . .
P3=> x"3",
. . .
```
¡Error!

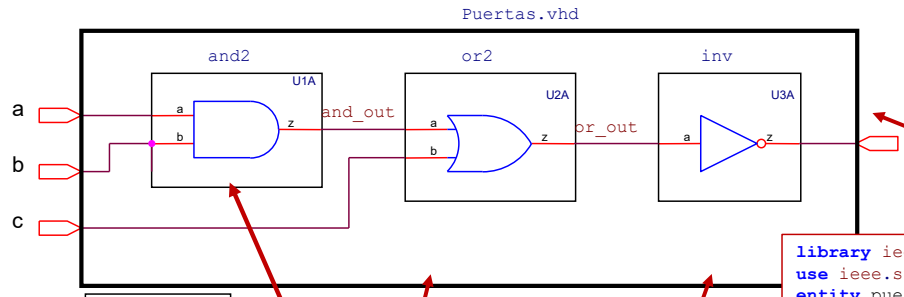
```
. . .
P1=> open,
. . .
```
¡Error!

```
. . .
P1=> S2,
. . .
```
¡Error!

- VHDL-08 permite asignar a un puerto una expresión.

```
entity cmp2 is
  port (
     clk    : in  std_logic;
     rst    : in  std_logic;
     ce     : in  std_logic;
     Q_out  : out std_logic_vector(3 downto 0));
end shifter16;
```

```
entity diseno2 is
  port( E     : in  std_logic;
        RELOJ : in  std_logic;
        INI   : out std_logic;
        SHF   : out std_logic_vector(3 downto 0));
end;
architecture rtl of diseno is
. . .
 signal EN:     std_logic;
 signal S2: std_logic_vector(3 downto 0);
begin
. . .
UX: entity work.cmp
       port map(
              clk  => RELOJ,
              rst  => INI,
              ce   => E and EN,
              Q_out => S2);
                     . . .
```

and2.vhd

```
library ieee;
use ieee.std_logic_1164.all;
entity and2 is
  port(a : in std_logic;
       b : in std_logic;
       z : out std_logic);
end and2;
architecture rtl of and2 is
begin  -- rtl
  z<=a and b;
end rtl;
```

inv.vhd

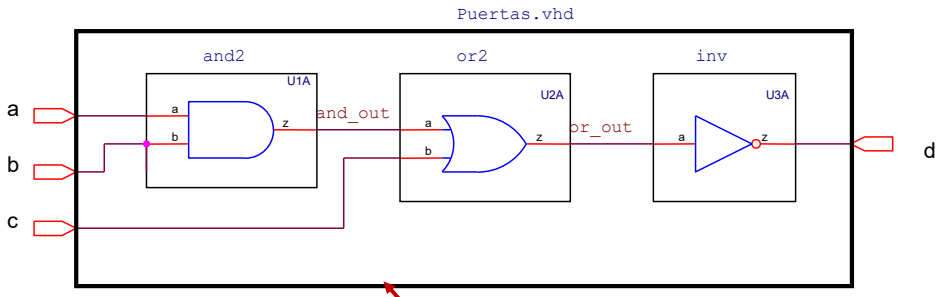
```
library ieee;
use ieee.std_logic_1164.all;
entity inv is
  port(a : in std_logic;
       z : out std_logic);
end inv ;
architecture rtl of inv is
begin  -- rtl
  z<=not a;
end rtl;
```

or2.vhd

```
library ieee;
use ieee.std_logic_1164.all;
entity or2 is
  port(a : in std_logic;
       b : in std_logic;
       z : out std_logic);
end or2 ;
architecture rtl of or2 is
begin  -- rtl
  z<=a or b;
end rtl;
```

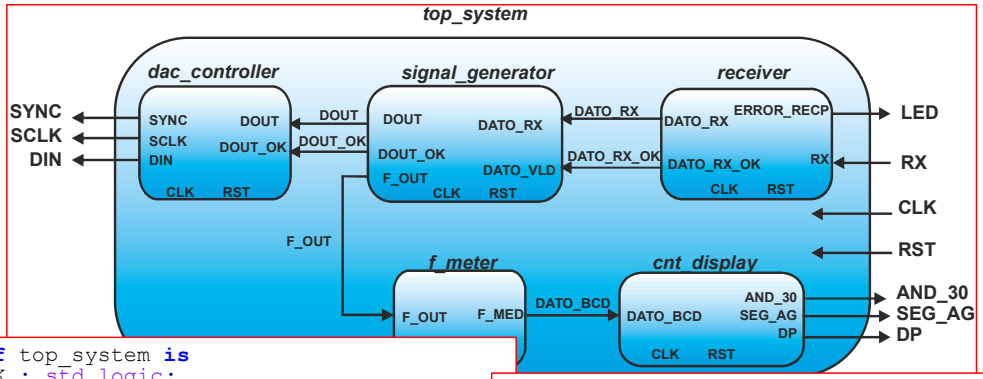
puertas.vhd

```
library ieee;
use ieee.std_logic_1164.all;
entity puertas is
  port(a         : in  std_logic;
       b         : in  std_logic;
       c         : in  std_logic;
       d         : out std_logic);
end puertas;
architecture structural of puertas is
  component and2
    port(a       : in  std_logic;
         b       : in  std_logic;
         z       : out std_logic);
  end component;
  component or2
    port(a       : in  std_logic;
         b       : in  std_logic;
         z       : out std_logic);
  end component;
  component inv
    port(a       : in  std_logic;
         z       : out std_logic);
  end component;
  signal and_out :     std_logic;
  signal or_out  :     std_logic;
begin
  u1A : and2 port map(a => a, b => b, z => and_out);
  u2A : or2 port map(a  => and_out, b => c, z => or_out);
  u3A : inv port map(a  => or_out, z => d);
end structural;
```



puertas.vhd

```
library ieee;
use ieee.std_logic_1164.all;
entity puertas is
  port(a          : in  std_logic;
       b          : in  std_logic;
       c          : in  std_logic;
       d          : out std_logic);
end puertas;
architecture structural of puertas is
signal and_out :     std_logic;
  signal or_out  :     std_logic;
begin
  u1A : entity work.and2 port map(a => a, b => b, z => and_out);
  u2A : entity work.or2 port map(a  => and_out, b => c, z => or_out);
  u3A : entity work.inv port map(a  => or_out, z => d);
end structural;
```


```
architecture rtl of top_system is
  signal DATO_RX_OK : std_logic;
  signal DATO_RX    : std_logic_vector(7 downto 0);
  signal DOUT       : std_logic_vector (11 downto 0);
  signal DOUT_OK    : std_logic;
  signal F_OUT      : std_logic;
  signal DATO_BCD   : std_logic_vector(15 downto 0);
begin  -- rtl

  u_rx : entity work.receiver
    port map (
      clk         => CLK,
      rst         => RST,
      rx          => RX,
      DATO_RX     => DATO_RX,
      error_recep => LED,
      DATO_RX_OK  => DATO_RX_OK);


  u_sg : entity work.signal_generator
    port map (
      RST         => RST,
      CLK         => CLK,
      DATO_RX     => DATO_RX,
      DATO_RX_OK  => DATO_RX_OK,
      DOUT        => DOUT,
      F_OUT       => F_OUT,
      DOUT_OK     => DOUT_OK);
```

```
  u_dac : entity work.dac_controller
    port map (
      CLK      => CLK,
      RST      => RST,
      DOUT     => DOUT,
      DOUT_OK  => DOUT_OK,
      SYNC     => SYNC,
      SCLK     => SCLK,
      DIN      => DIN);


  u_fm : entity work.f_meter
    port map (
      CLK    => CLK,
      RST    => RST,
      F_OUT  => F_OUT,
      F_MED  => DATO_BCD);


  u_cd : entity work.cnt_display
    port map (
      CLK      => CLK,
      RST      => RST,
      DATO_BCD => DATO_BCD,
      AND_30   => AND_30,
      DP       => DP,
      SEG_AG   => SEG_AG);

end rtl;
```

- En una entidad pueden coexistir todos los tipos de estilos descriptivos:
  - Algorítmico.
  - Flujo de datos.
  - Estructural

Estructural

Algorítmico

Flujo de datos

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity test_dcm is

  port (
    RST, RELOJ : in  std_logic;
    Q          : out std_logic_vector(3 downto 0));

end test_dcm;
architecture rtl of test_dcm is
signal cnt : unsigned(3 downto 0);
signal CLK : std_logic;
begin  -- rtl

  u1 :entity work.clocking
  port map
   (-- Clock in ports
    CLK_IN1 => RELOJ,
    -- Clock out ports
    CLK_OUT1 => CLK);

  process (CLK, RST)
  begin  -- process
    if RST = '1' then
      cnt<=(others=>'0');
    elsif CLK'event and CLK = '1' then
      cnt<=cnt+1;
    end if;
  end process;

Q<=std_logic_vector(cnt);
end rtl;
```