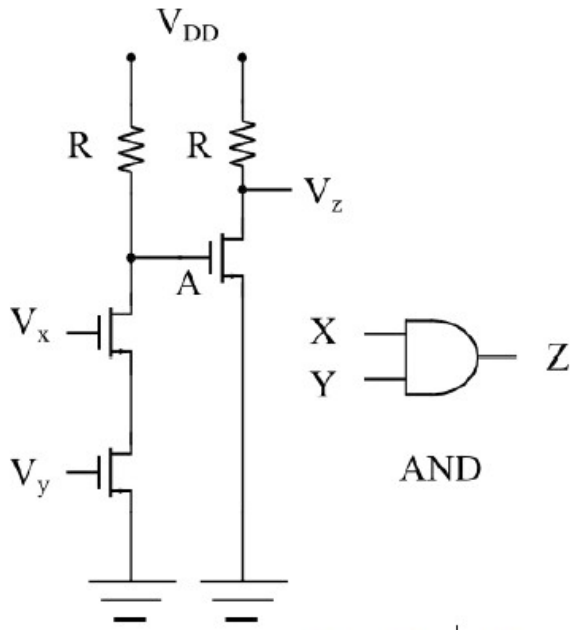
Tema 3:
Bloques combinacionales

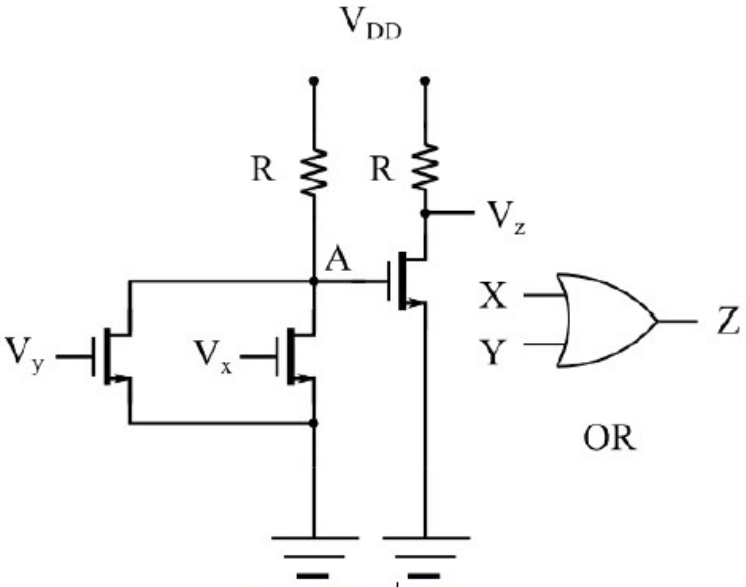
Sistemas combinacionales lógicos

- Los circuitos combinacionales son aquéllos en los que una serie de variables X_i , definen una serie de funciones Z_j.
- Cada combinación de valores de Xi define un sólo valor de Z_j.
- Los valores de Z_i sólo pueden cambiar cuando cambian los valores de X_j.

Circuito AND y circuito OR

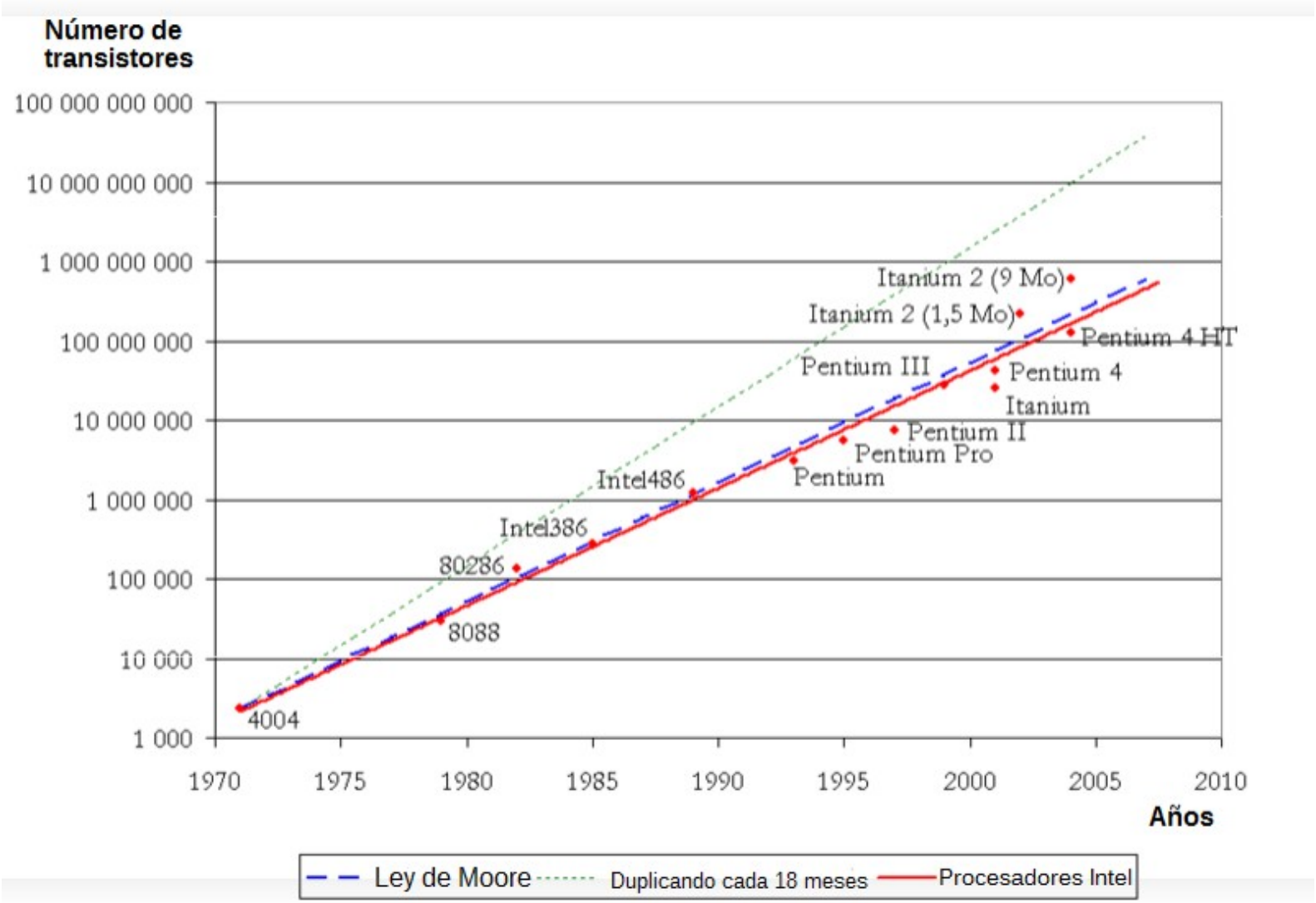


X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1



X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

Ley de Moore


Número de transistores
100 000 000 000
10 000 000 000
1 000 000 000
100 000 000
10 000 000
1 000 000
100 000
10 000
1 000
1970
1975
1980
1985
1990
1995
2000
2005
2010
Años
Itanium 2 (9 Mo)
Itanium 2 (1,5 Mo)
Pentium 4 HT
Pentium III
Pentium 4
Itanium
Pentium II
Pentium Pro
Pentium
Intel486
Intel386
80286
8088
4004
Ley de Moore
Duplicando cada 18 meses
Procesadores Intel

Escala de integración

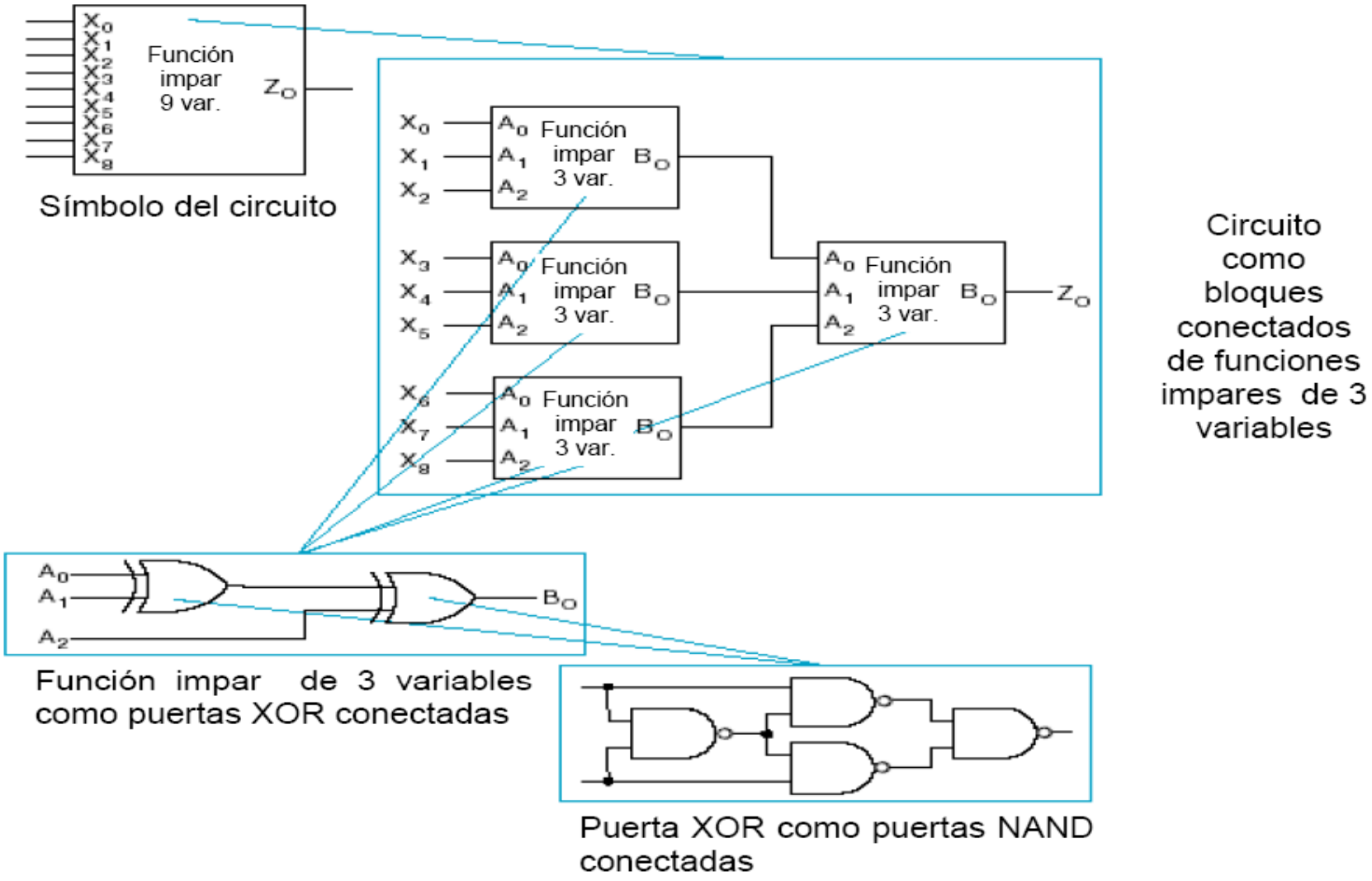
- Los circuitos integrados se clasifican por el número de puertas que incluyen.

- Existen 6 escalas de integración:
  - **SSI** (Small Scale Integration ‹ 10 puertas): Puertas Lógicas y Biestables
  - **MSI** (Medium SI entre 10 y 100 puertas): Codificadores, Multiplexores, Sumadores, etc..
  - **LSI** (Large SI entre 100 y 1000 puertas): Memorias, Circuitos Aritmeticos complejos
  - **VLSI** (Very Large SI menor de 10.000 puertas): µPs y µCs poco complejos, PLDs (Programmable Logic Devices)
  - **ULSI** (Ultra LSI menor de 100.000 puertas.): µPs, µCs y FPGAs (Field Programmable Gate Array) de complejidad media
  - **GLSI** (Giga LSI › 100.000 puertas.): µPs y FPGAs de alta complejidad

Diseño jerárquico

- ¿Pero cómo se puede diseñar un circuito con miles de puertas lógicas?
- La solución es dividir el circuito en funciones más pequeñas (bloques funcionales), de modo que podamos resolver éstas por medio del álgebra **booleana.**
- Organizando los bloques funcionales, podemos resolver el problema más grande.
- Este enfoque se denomina **diseño jerárquico**, porque se establecen niveles de diseño, el más bajo de los cuales se resuelve mediante álgebra booleana y puertas lógicas.

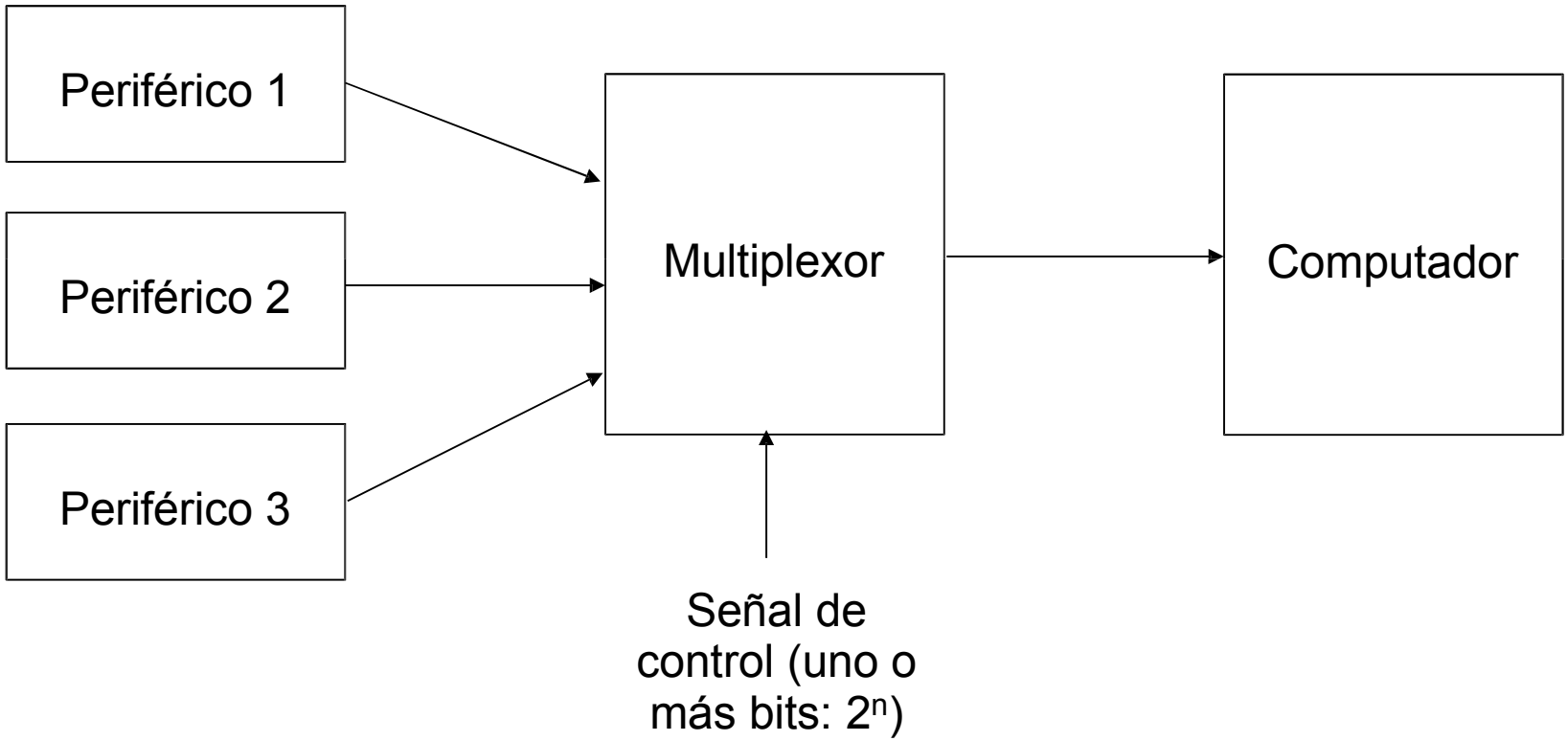
Diseño jerárquico



El circuito final tiene 9 variables, pero sólo hemos usado el álgebra de Boole para dos variables

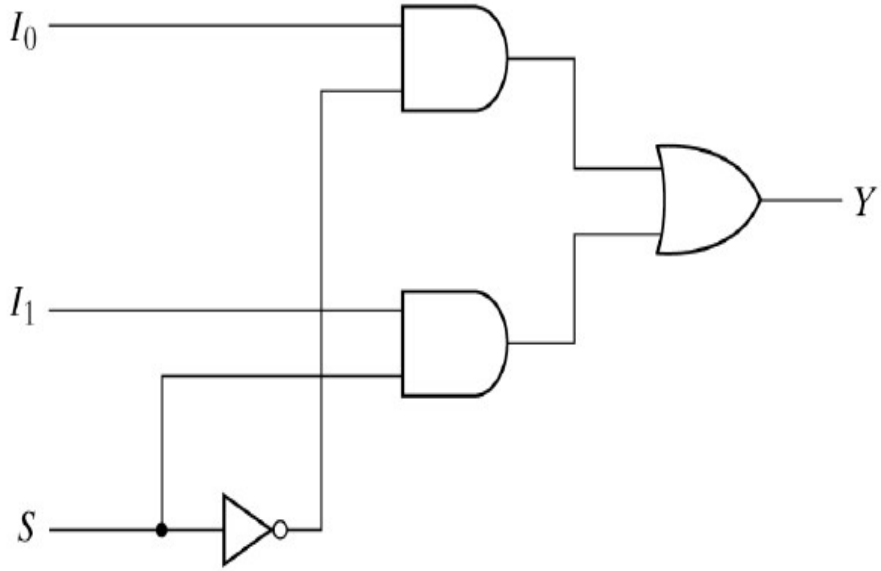
Multiplexor

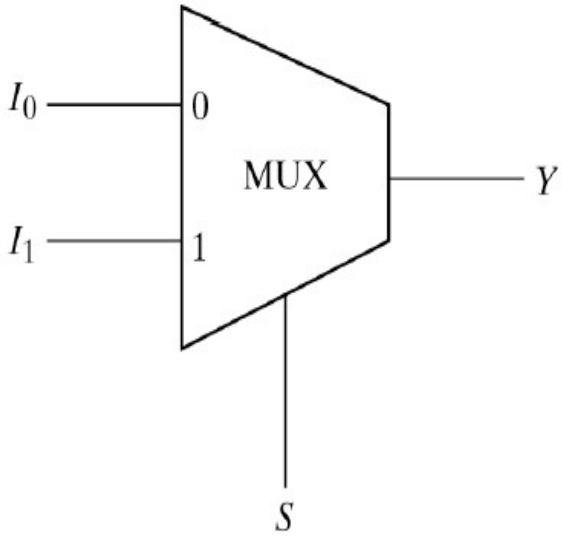
- El multiplexor se usa para elegir una entre varias variables.
- Según el valor de una variable de control, se elige una de entre varias variables de entrada.
- La salida del bloque será la variable elegida.



Multiplexor 2 a 1

S	I_0	I_1	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



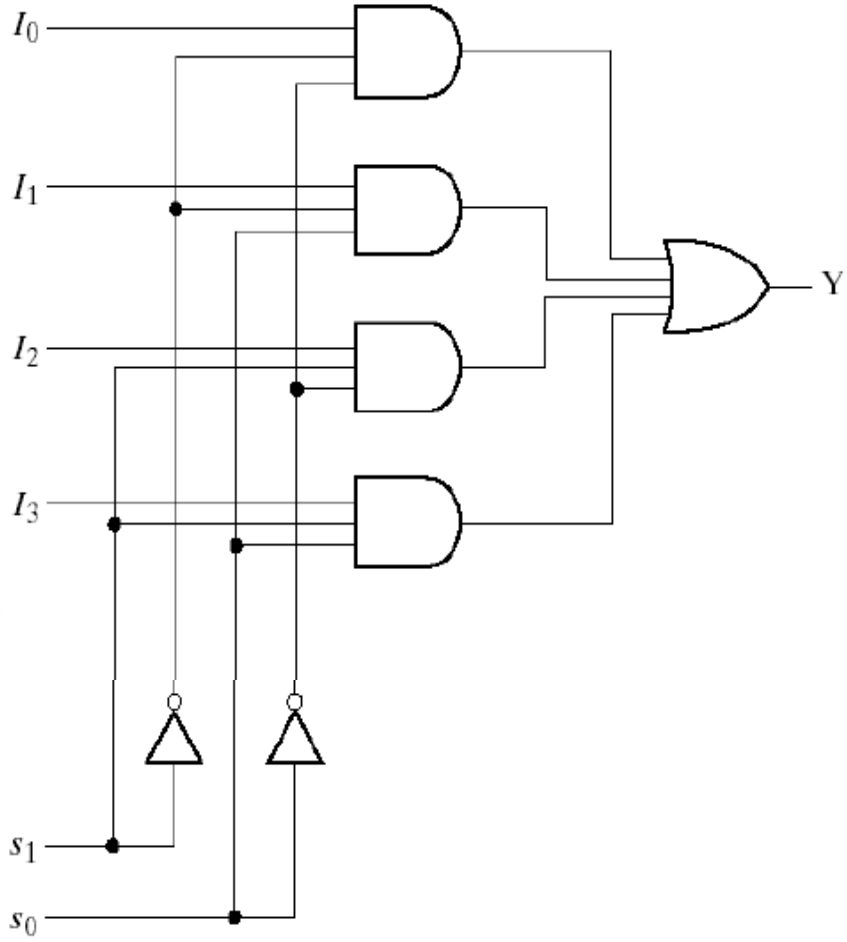


$$Y = \overline{S} \cdot I_0 + S \cdot I_1$$

Multiplexor 4 a 1

s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

$$Y = \overline{S_1} \cdot \overline{S_0} \cdot I_0 + \overline{S_1} \cdot S_0 \cdot I_1 + S_1 \cdot \overline{S_0} \cdot I_2 + S_1 \cdot S_0 \cdot I_3$$



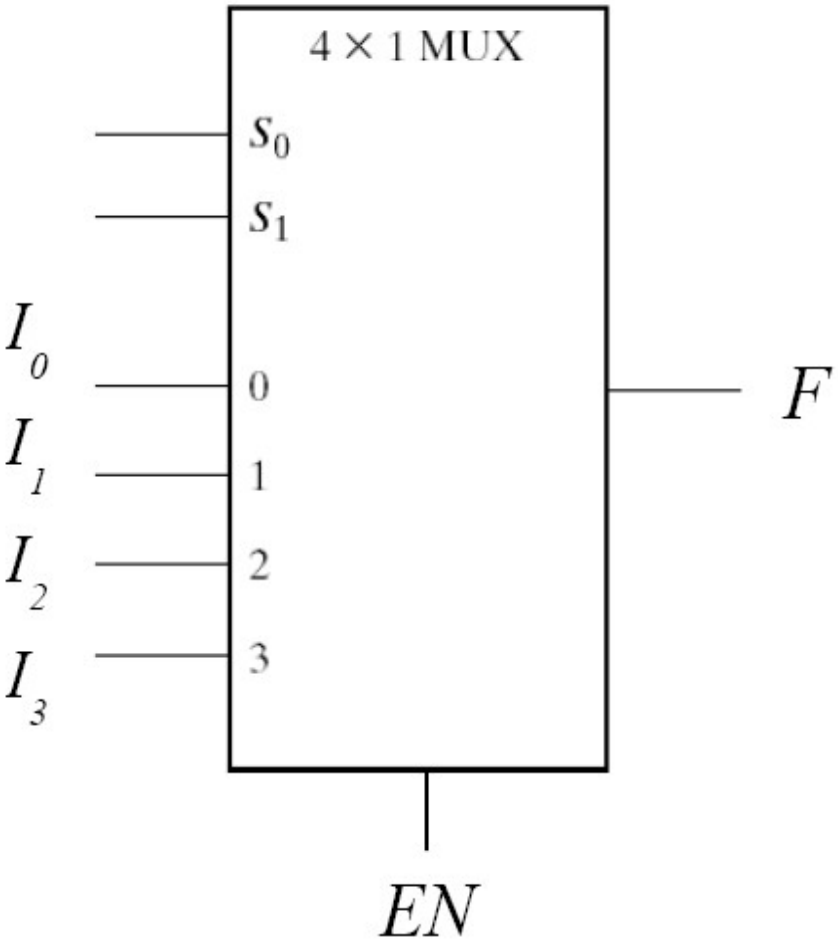
Para cualquier número de entradas 2^n*, el número de señales de control es* ***n.***

Entrada de habilitación

- Los bloques funcionales se usan como elementosde sistemas más grandes.
- Esos sistemas combinan las funciones de estos bloques, usando sólo parte de ellos en distintos momentos.
- Para regular cuándo se usa cada bloque, se usa la entrada de habilitación.
- Cuando está desactivada, el bloque no produce ninguna función, manteniendo la salida constante (a 0 ó 1 según dispositivo).
- Engeneral, suele nombrarse E o EN (abreviatura de *ENABLE* habilitar en inglés).

Multiplexor 4 a 1 con entrada de habilitación

EN	S_1	S_0	F
0	X	X	0
1	0	0	I_0
1	0	1	I_1
1	1	0	I_2
1	1	1	I_3

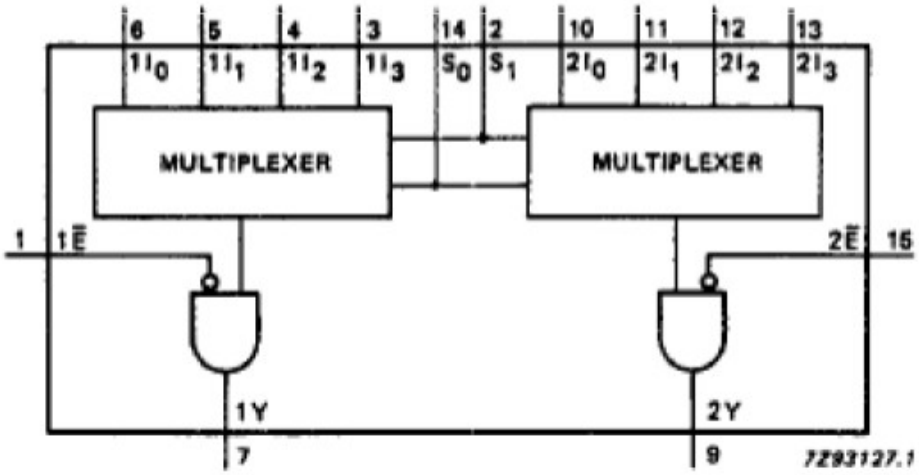


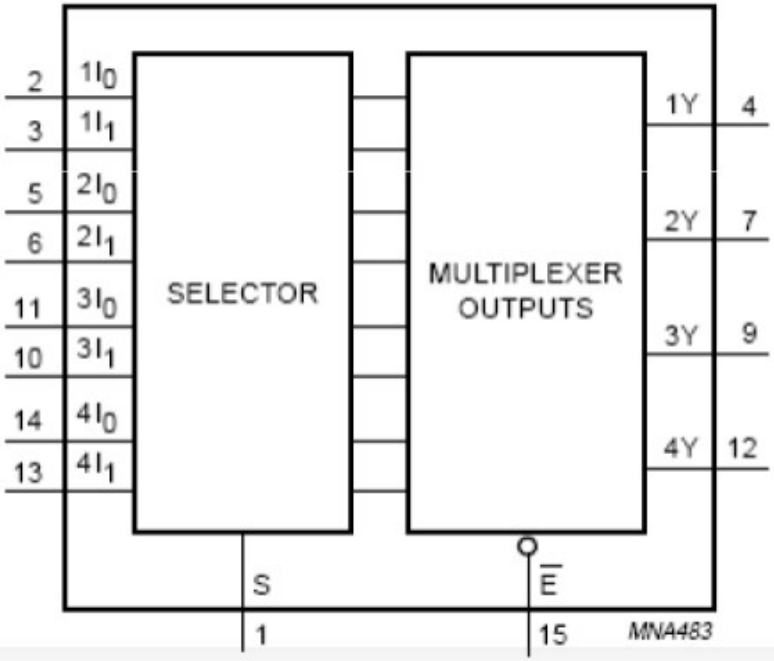
$$Y = EN \cdot \left(\overline{S_1} \cdot \overline{S_0} \cdot I_0 + \overline{S_1} \cdot S_0 \cdot I_1 + S_1 \cdot \overline{S_0} \cdot I_2 + S_1 \cdot S_0 \cdot I_3\right)$$

Aplicaciones: multiplexores con buses

- Un conjunto de señales eléctricas se llama bus.
- Los buses se utilizan para definir datos que supongan más de una variable binaria.
- Tanto las entradas como las salidas de los multiplexores pueden ser buses con más de una variable binaria.
- En ese caso, el número de variables de control será n si el cociente entre el número de señales de entrada y el número de señales de salida es 2^n.

Dos circuitos integrados multiplexores





SELECT INPUTS		DATA INPUTS				OUTPUT ENABLE	OUTPUT
S_0	S_1	nI_0	nI_1	nI_2	nI_3	$n\bar{E}$	nY
X	X	X	X	X	X	H	L
L	L	L	X	X	X	L	L
L	L	H	X	X	X	L	H
H	L	X	L	X	X	L	L
H	L	X	H	X	X	L	H
L	H	X	X	L	X	L	L
L	H	X	X	H	X	L	H
H	H	X	X	X	L	L	L
H	H	X	X	X	H	L	H

74HC/HCT153

INPUT				OUTPUT
$\bar{E}$	S	nI_0	nI_1	nY
H	X	X	X	L
L	L	L	X	L
L	L	H	X	H
L	H	X	L	L
L	H	X	H	H

74AHC157

Multiplexor en VHDL

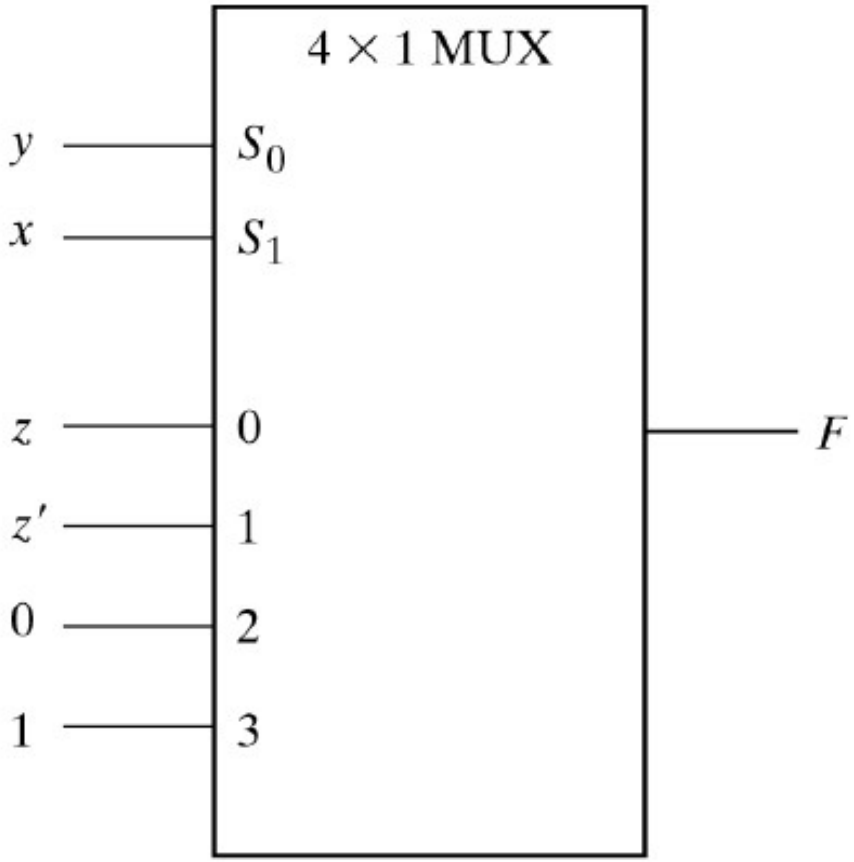
- Esta es la descripción de un multiplexor 4:1 en VHDL:

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

ENTITY mux4to1 IS
PORT (    w0, w1, w2, w3      : IN  STD_LOGIC ;
s      : IN  STD_LOGIC_VECTOR(1 DOWNTO 0) ;
f      : OUT     STD_LOGIC ) ;
END mux4to1 ;

ARCHITECTURE a OF mux4to1 IS
BEGIN
WITH s SELECT
f <= w0 WHEN "00",
             w1 WHEN "01",
             w2 WHEN "10",
             w3 WHEN OTHERS ;
END a ;
```

Aplicaciones: implementación de funciones con multiplexores

- La salida de un multiplexor se expresa siempre como suma de productos multiplicado por cada entrada.
- La expresión canónica de cualquier función es una suma de productos.
- Si utilizamos las señales de control como variables de entrada, podemos representar la expresión canónica de cualquier función.
- Podemos usar las entradas del multiplexor para completar los mintérminos con otra variable.
- La salida del multiplexor representará la función buscada.

Aplicaciones: implementación de funciones con multiplexores



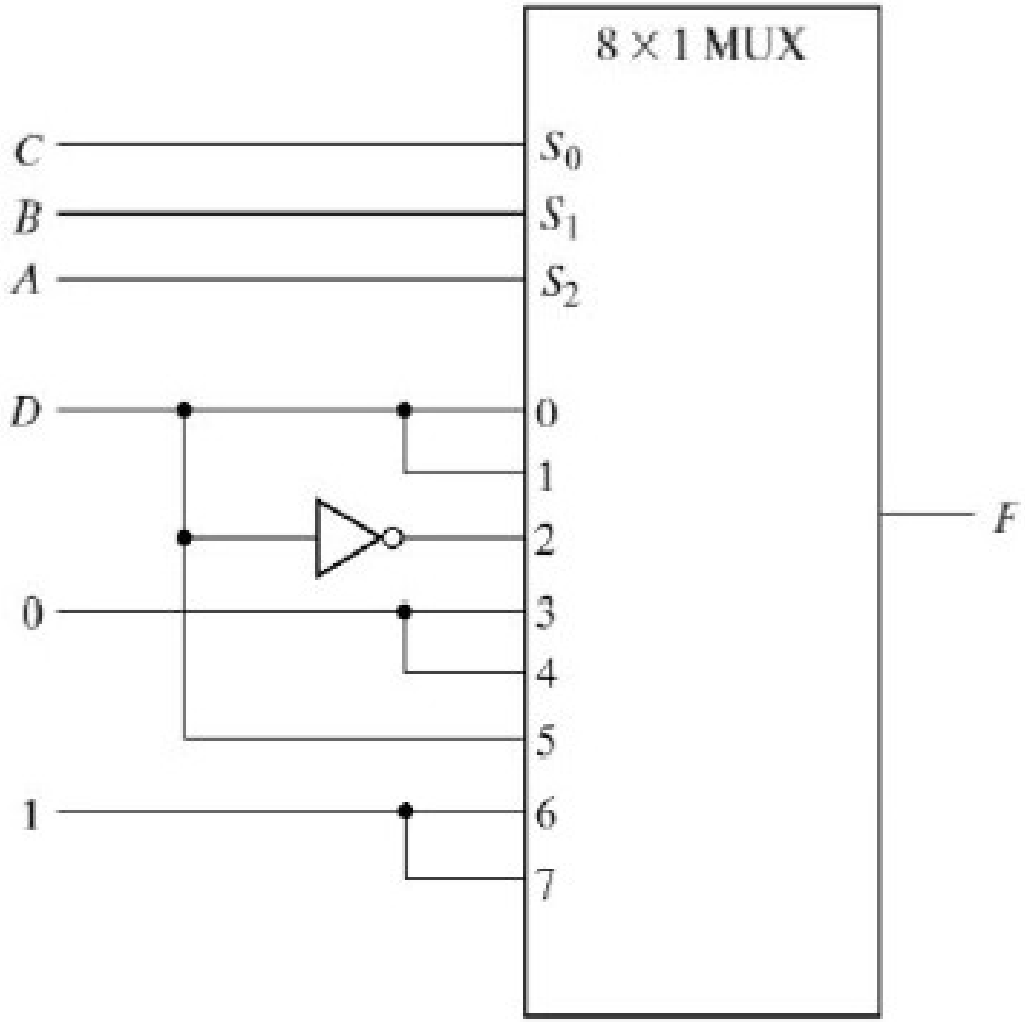
x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

$$F=x'y'z + x'yz' + xyz' + xyz$$

Implementación de funciones con multiplexores: función de 4 variables

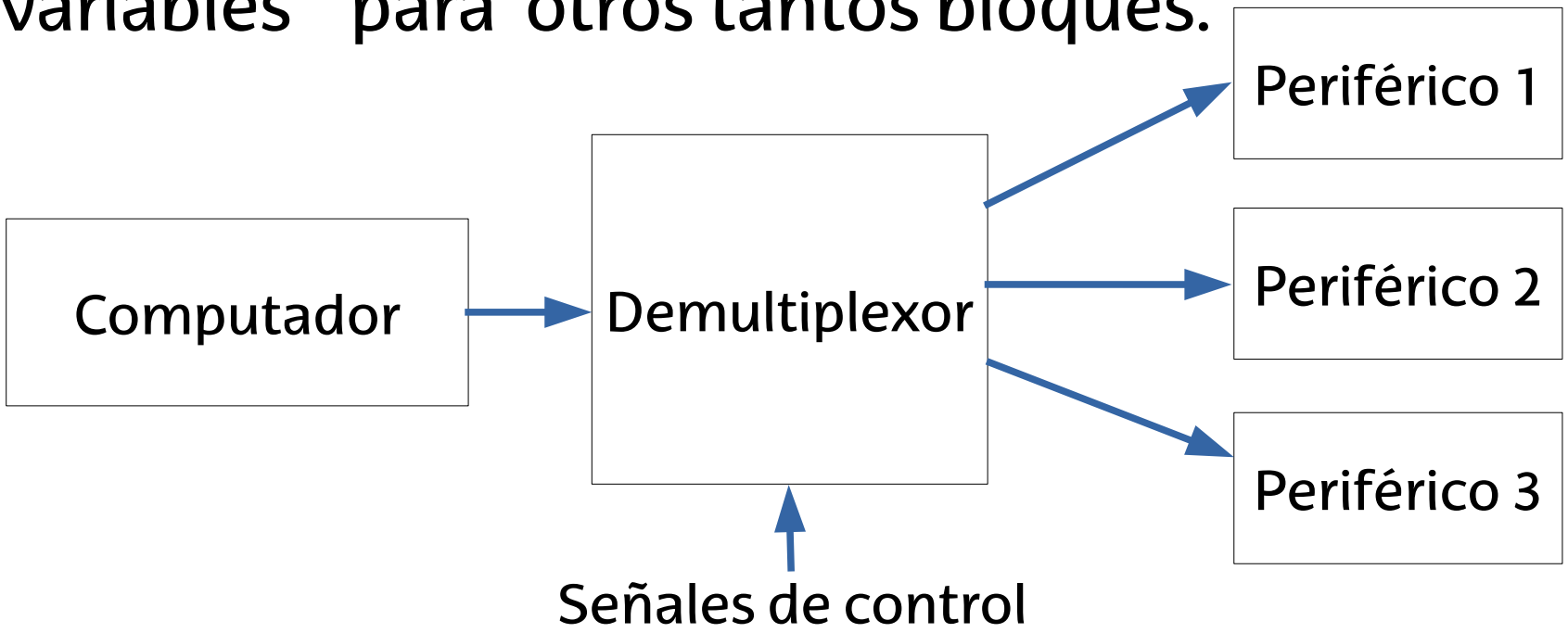
$$F = \overline{A}\cdot\overline{B}\cdot\overline{C}\cdot D + \overline{A}\cdot\overline{B}\cdot C\cdot D + \overline{A}\cdot B\cdot\overline{C}\cdot\overline{D} + A\cdot\overline{B}\cdot C\cdot D +$$
$$A\cdot B\cdot\overline{C}\cdot\overline{D} + A\cdot B\cdot\overline{C}\cdot D + A\cdot B\cdot C\cdot\overline{D} + A\cdot B\cdot C\cdot D$$

A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	



Demultiplexor

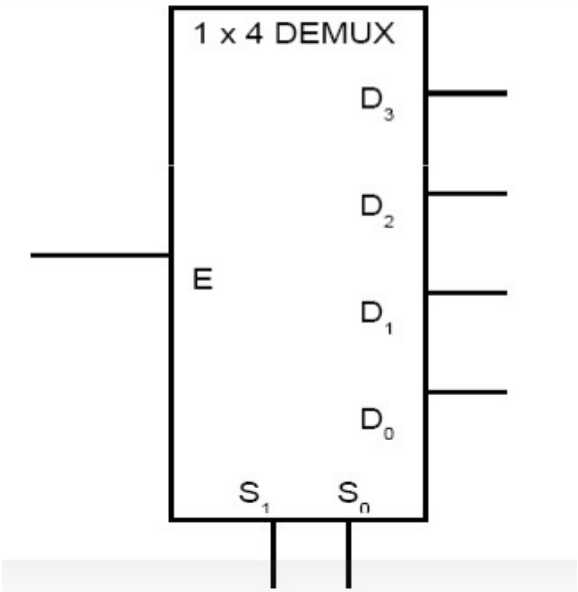
- El demultiplexor se usa para direccionar una variable hacia una de entre varias funciones.
- Según el valor de una variable de control, se transfiere el valor de la variable a una de entre varias funciones.
- Las salidas del bloque serán nuevas variables para otros tantos bloques.

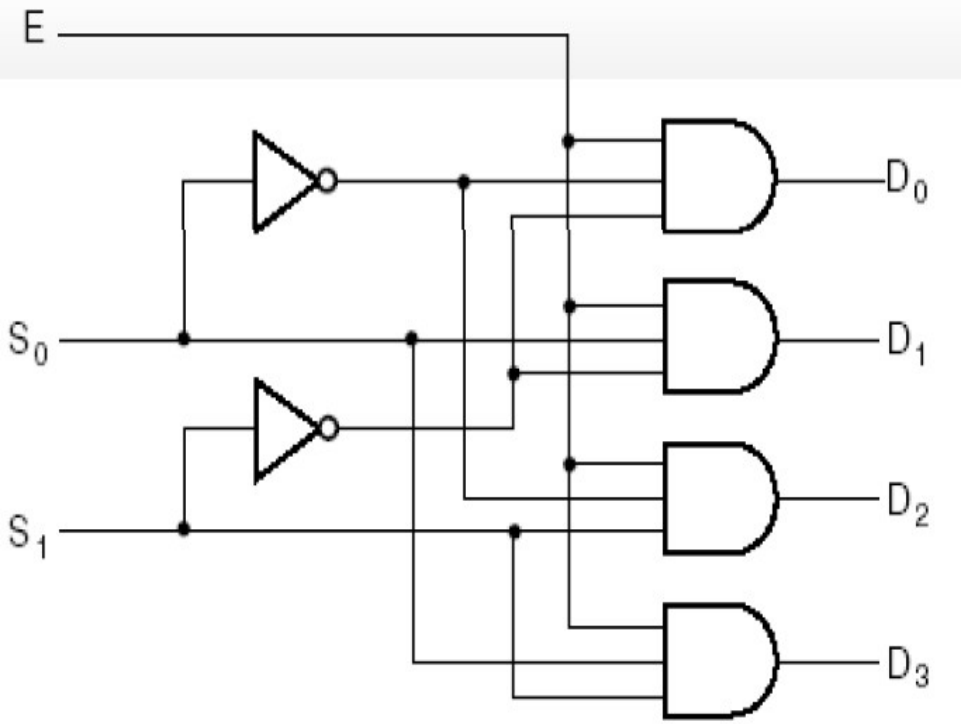


Demultiplexor de 1 a 4

E	S_1	S_0	D_0	D_1	D_2	D_3
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

$$D_0 = E \cdot S_1' \cdot S_0'$$
$$D_1 = E \cdot S_1' \cdot S_0$$
$$D_2 = E \cdot S_1 \cdot S_0'$$
$$D_3 = E \cdot S_1 \cdot S_0$$



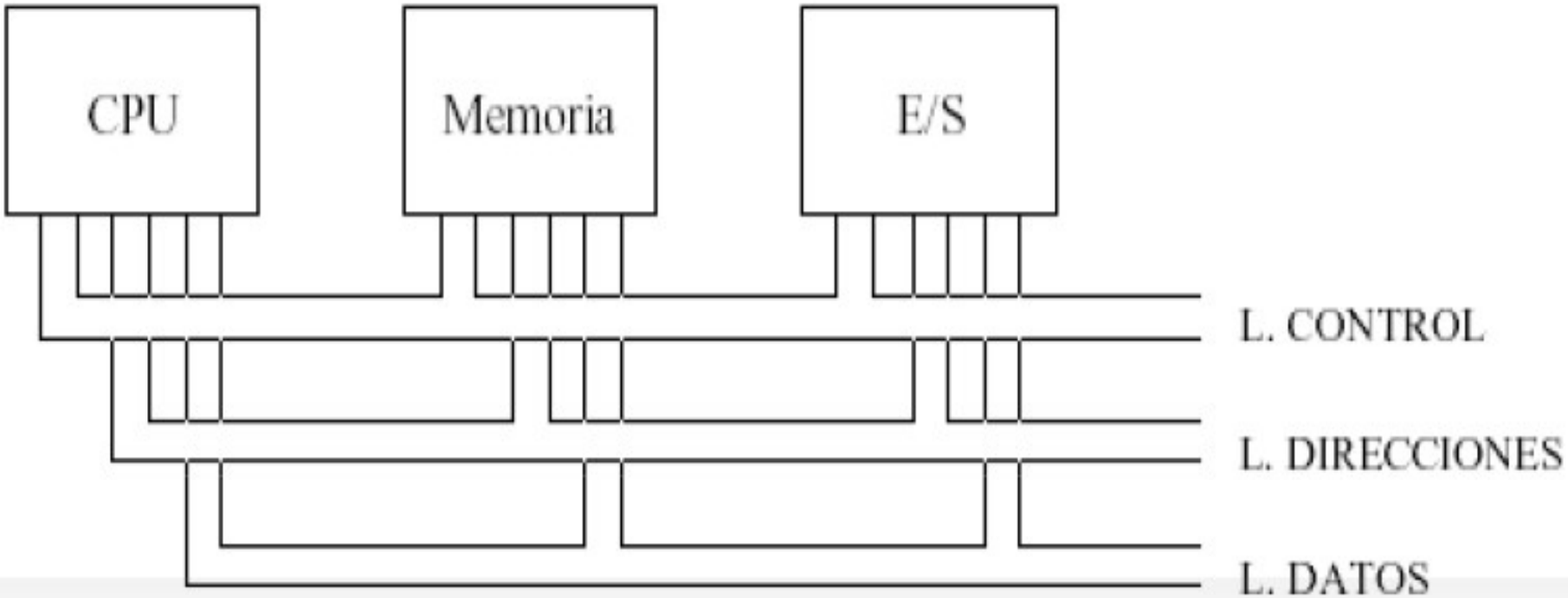


Demultiplexor en VHDL

- Esta es la descripción de un demultiplexor 1:4 en VHDL:

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

ENTITY demux1to4 is
PORT(    w0, w1, w2, w3      : out std_logic;
         f                   : in  std_logic;
         s                   : in  std_logic_vector(1 downto 0));
END demux1to4;

ARCHITECTURE a OF demux1to4 IS
BEGIN
         w0<= f when s="00" else '0';
         w1<= f when s="01" else '0';
         w2<= f when s="10" else '0';
         w3<= f when s="11" else '0';
END a;
```

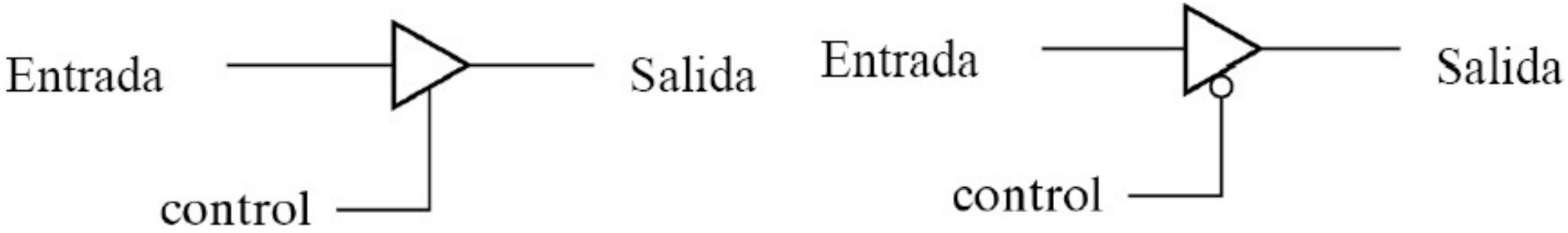
Sistemas electrónicos basados en buses



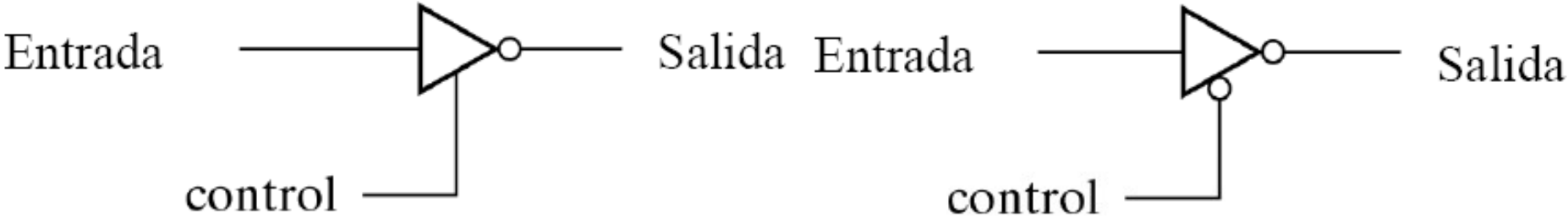
Puertas triestado

- Los buses son a menudo compartidos por varios bloques funcionales, pero el uso de una sola línea por dos bloques a la vez puede llevar a un cortocircuito.
- En ese caso, es preciso asegurar que sólo accedan al bus una pareja de bloques (emisor y receptor de la señal).
- Laconexión de un bloque con el bus se puede cortar eléctricamente (alta impedancia) mediante una puerta triestado.

Puertas triestado



Puertas no inversoras (Entrada= Salida), señal de control de nivel alto y bajo.



Puertas inversoras (Entrada = Salida'), señal de control de nivel alto y bajo.

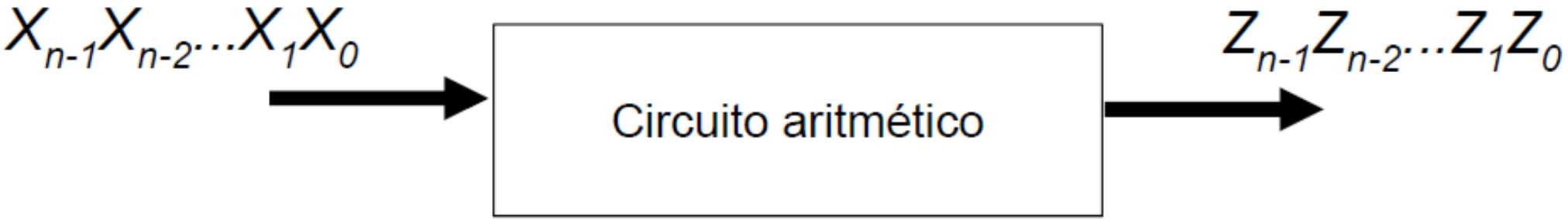
Puerta triestado en VHDL

- Esta es la descripción en VHDL de una puerta triestado no inversora y señal de control a nivel alto:

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

ENTITY tri_st is
PORT(   I      : out std_logic;
        S      : in  std_logic;
        K      : in  std_logic);
END tri_st;

ARCHITECTURE a OF tri_st IS
BEGIN
        I<= S when K='1' else 'Z';
END a;
```

Sistemas combinacionales aritméticos

- Los sistemas combinacionales aritméticos son aquéllos que usan números en entradas o salidas.
- Los números se representan mediante variables lógicas cuyos dos valores representan la cifra 0 o la cifra 1.
- Usaremos el sistema de numeración binaria, ya que sólo utiliza las cifras 0 y 1.
- Las variables numéricas se representan mediante una letra mayúscula y una serie de subíndices para diferenciar cada cifra binaria.



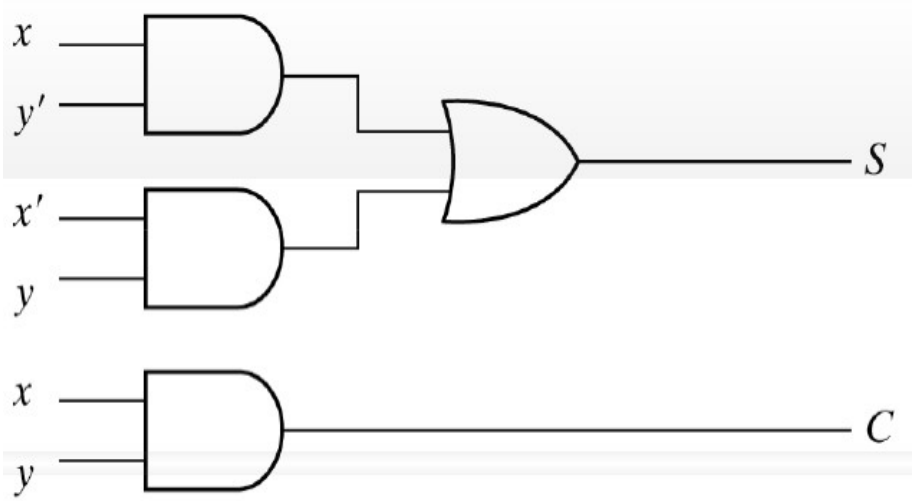
Semisumador

- La suma aritmética de dos cifras binarias es una función lógica de dos variables.
- Para incluir todas las posibilidades de la suma, la respuesta son dos funciones: S (suma) y C (acarreo).

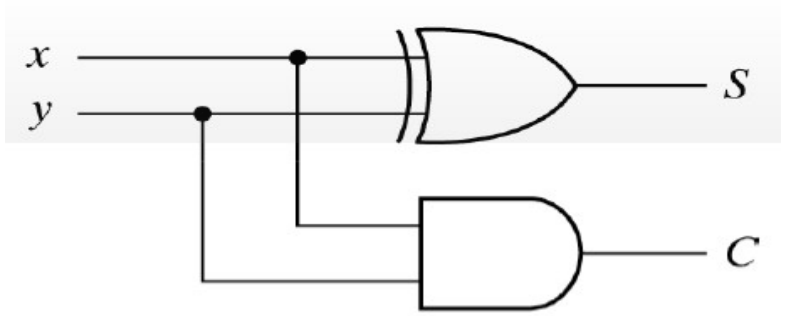
$$\begin{array}{r} 0 \\ +\ 0 \\ \hline 00 \end{array} \qquad \begin{array}{r} 0 \\ +\ 1 \\ \hline 01 \end{array} \qquad \begin{array}{r} 1 \\ +\ 1 \\ \hline 10 \end{array}$$

X	Y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Implementación del semisumador



(a) $S = xy' + x'y$
$C = xy$

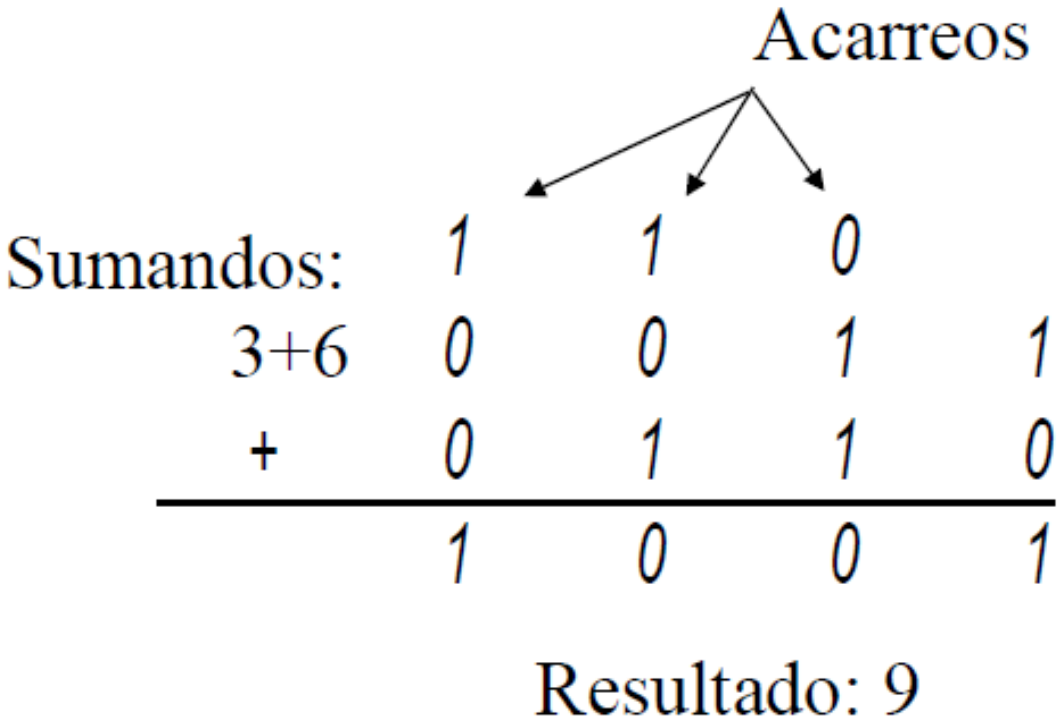


(b) $S = x \oplus y$
$C = xy$

- Esta es la expresión algebraica de las dos funciones S y C de la suma.
- Para que sea práctica, debemos generalizarla a la suma de n bit.

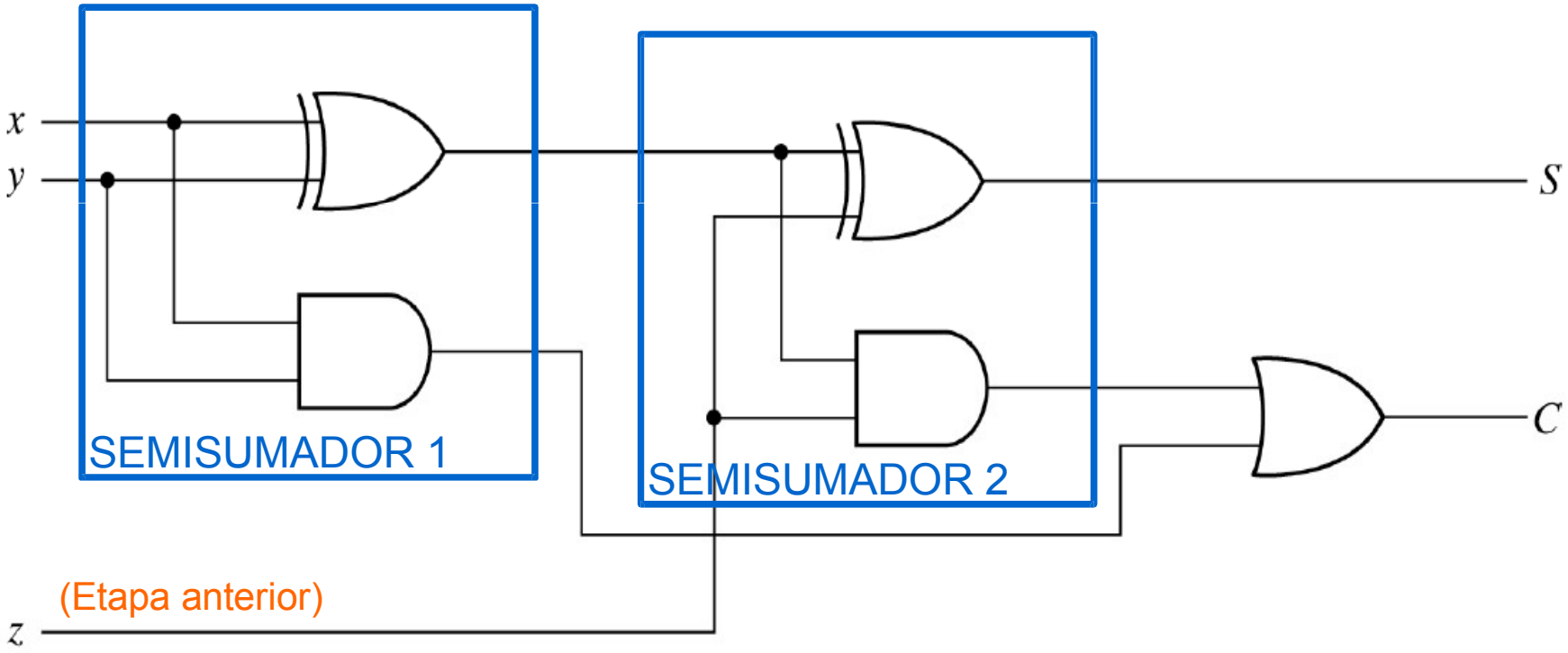
Sumador completo

- Para realizar sumas de n bit, podemos realizar sumas de un bit para cada cifra de los dos sumandos.
- En cada cifra sumaremos dos bits de cada operando y el acarreo de la cifra anterior.
- Por ello definiremos la suma de tres operandos de un bit: sumador completo.



X	Y	Z (C_{i-1})	S	C_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

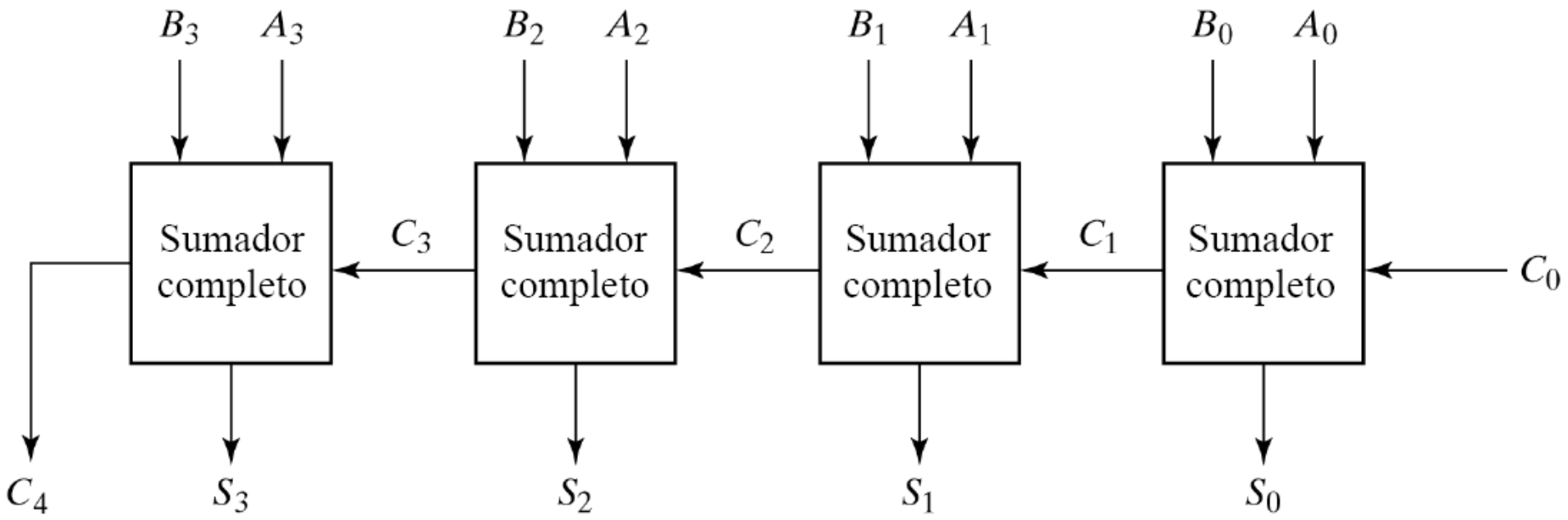
Implementación del sumador completo



$$\begin{cases} s = z \oplus (x \oplus y) \\ c_i = (x \cdot y) + (x \oplus y) \cdot z \end{cases}$$

La solución, obtenida de la tabla de la verdad de S y C, muestra que el sumador completo es la combinación de dos semisumadores.

Sumador de cuatro bits



- Combinando n sumadores completos, se obtiene un sumador de n bit.
- Para obtener la última cifra S_{n-1}, hace falta el acarreo C_{n-1}, obtenido del bloque anterior, y éste del anterior a él-›Acumulación de retardos.

Restador

- La resta binaria puede definirse del mismo modo que la suma binaria.
- Sin embargo, si el minuendo es menor que el sustraendo, produce un resultado negativo.
- En ese caso, la resta se realiza invirtiendo el orden de los operadores, y el resultado se marca como negativo con una cifra binaria añadida (0 si es positivo y 1 si es negativo).
- El sistema de resta en magnitud con signo añadido no se utiliza por emplear muchos más elementos que la suma.

$$\begin{array}{r} 0 \\ -\ 0 \\ \hline 0\ 0 \end{array}$$

$$\begin{array}{r} 1 \\ -\ 0 \\ \hline 0\ 1 \end{array}$$

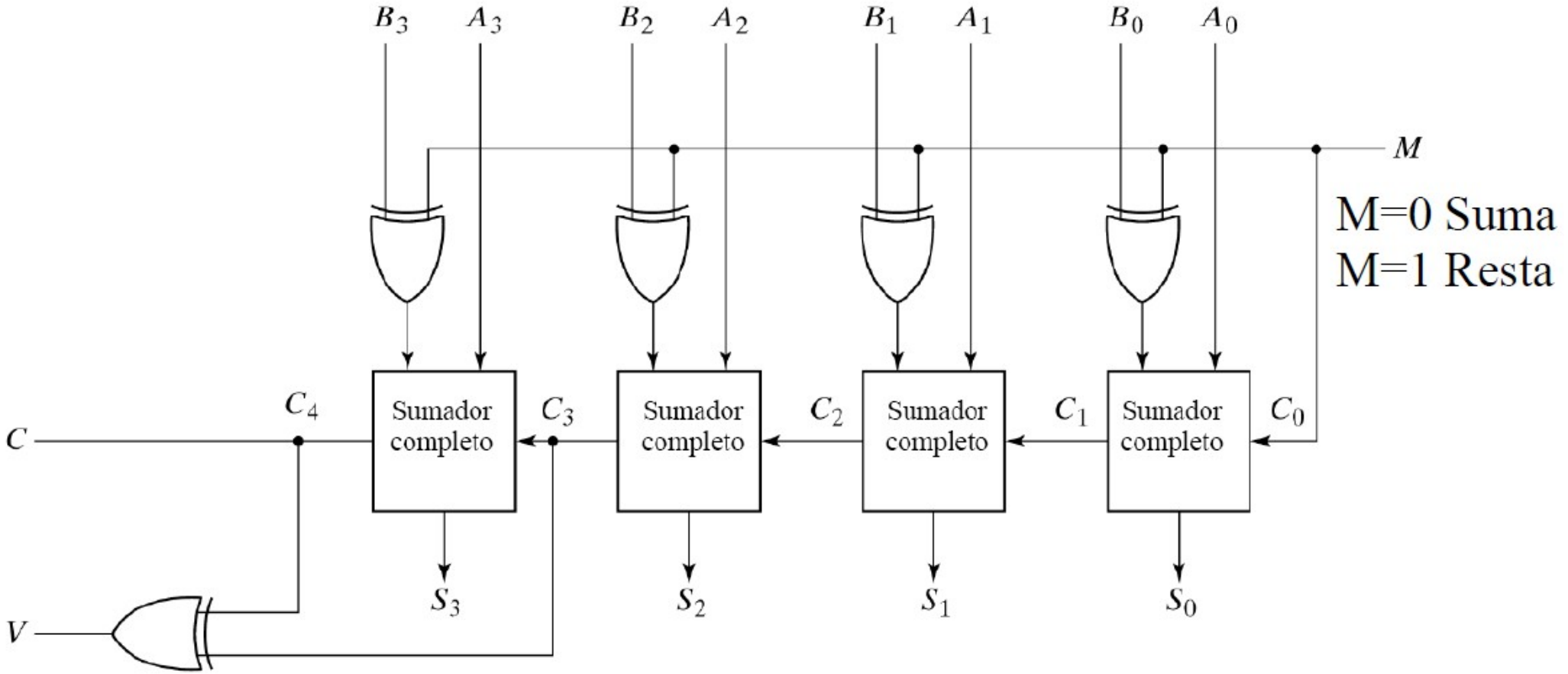
$$\begin{array}{r} 1 \\ -\ 1 \\ \hline 0 \end{array}$$

$$\begin{array}{r} 0 \\ -\ 1 \\ \hline 1\ 1 \end{array}$$

Resta en complemento a dos

- La resta en complemento a dos consiste en aplicar el complemento a dos al sustraendo, y luego realizar la suma: A-B=A+(-B).
- La operación es 2^n - B, y en la práctica consiste en sustituir 0s por 1s y viceversa (complemento a 1), y luego sumarle 1.
- El resultado es coherente con el sistema utilizado (si es negativo, está en complemento a dos) y permite unificar suma y resta en el mismo operador.
- Si el resultado es negativo, realizando la misma operación de complemento a dos, se obtiene la magnitud absoluta.
- Si el resultado es positivo, aparece siempre un acarreo en la última cifra, que descartamos.

Restador-sumador de cuatro bits



- La señal V indica si ha habido desbordamiento en la resta, es decir, si el resultado no se puede expresar en 4 bit.
- La señal C indica si ha habido desbordamiento en la suma.

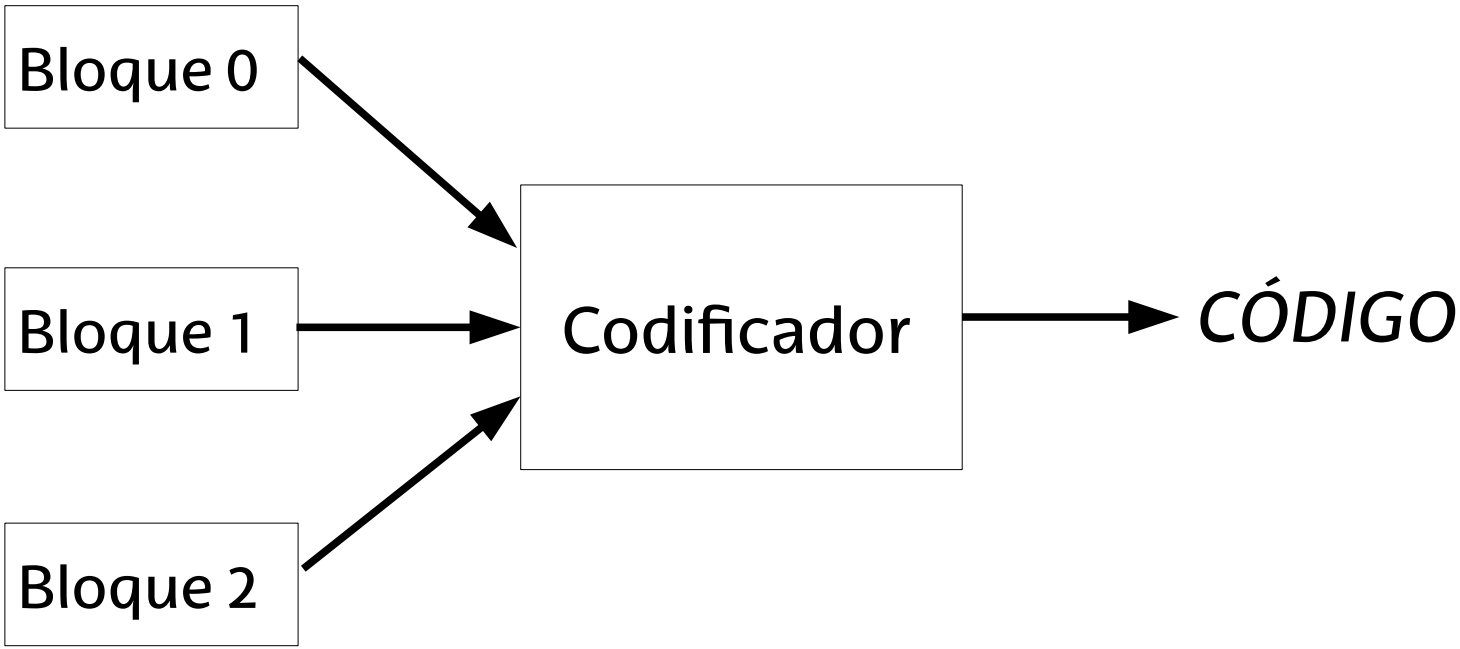
Sumador-restador de 4 bits en VHDL

- Esta es la implementación de un sumador-restador de 4 bits en VHDL:

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

ENTITY restsumIS
PORT (     M            : IN    STD_LOGIC ;
           A, B         :IN    STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           F            : OUT       STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END restsum ;

ARCHITECTURE a OF restsum IS
BEGIN
PROCESS ( M, A, B )
           BEGIN
                  CASE M IS
                  WHEN '0'                =>F <= A+B ;
                  WHEN OTHERS         =>F <= A-B;
           END CASE ;
END PROCESS ;
END a ;
```

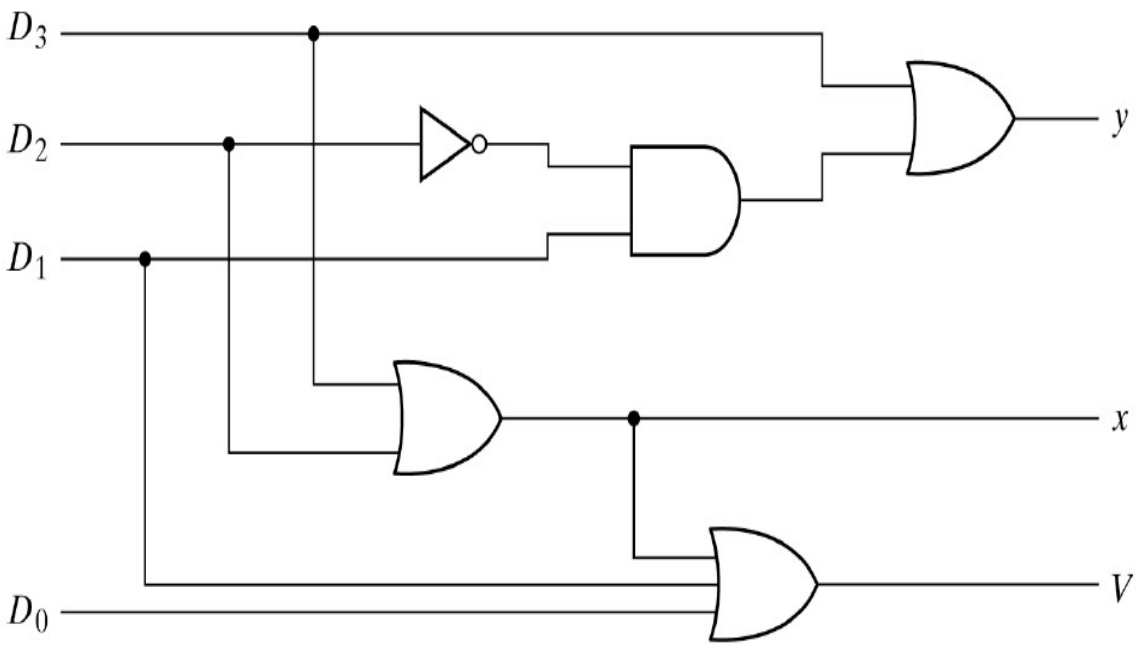
Codificador

- El codificador produce como salida el número (código) correspondiente a la variable que está activada.
- Si el código es el binario natural, el número de entradas es 2^n, siendo n el número de salidas.



Codificador con prioridad

D_0	D_1	D_2	D_3	x	y	v
0	0	0	0	0	0	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1



- Si se activan dos señales al mismo tiempo, la salida debe corresponder sólo a una, por lo que se establece prioridad entre las entradas.
- Normalmente, la entrada asociada al número mayor es la de mayor prioridad.
- La salida adicional V indica si hay alguna entrada activada.

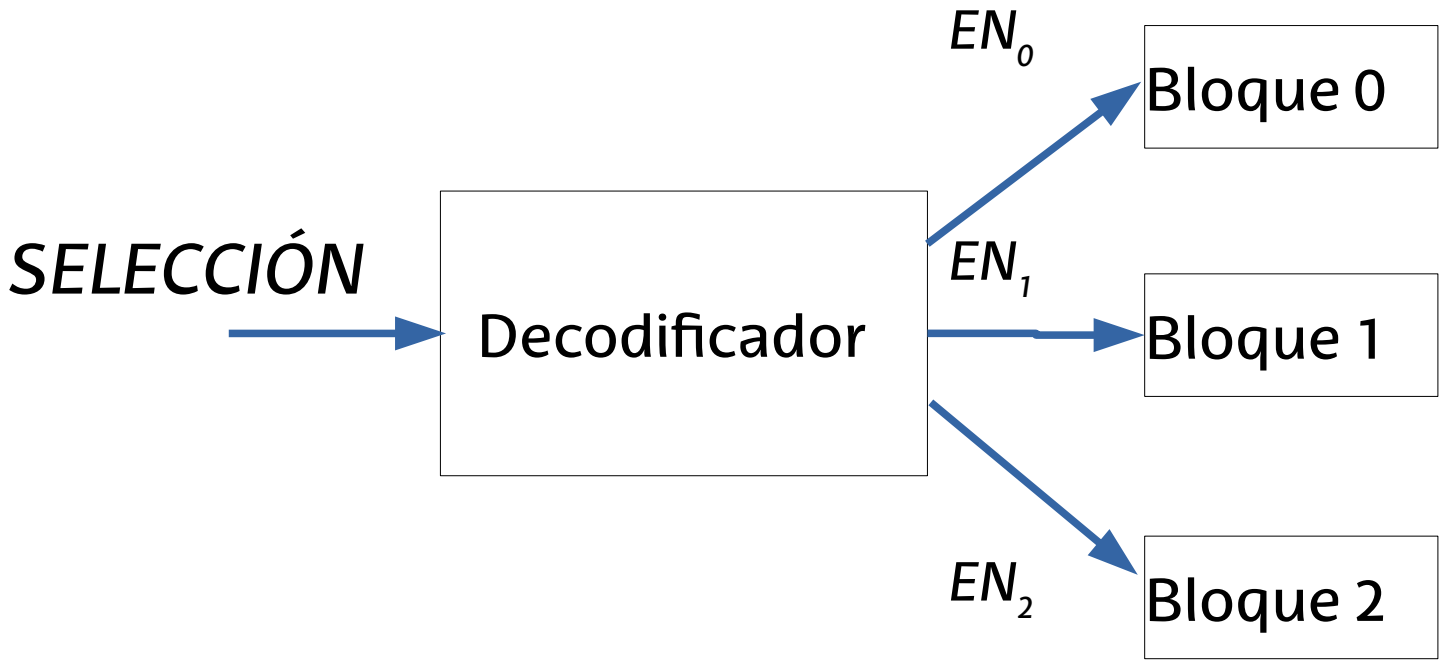
Codificador de 4 bit en VHDL

- Esta sería la descripción de un codificador con prioridad de 4 bit en VHDL:

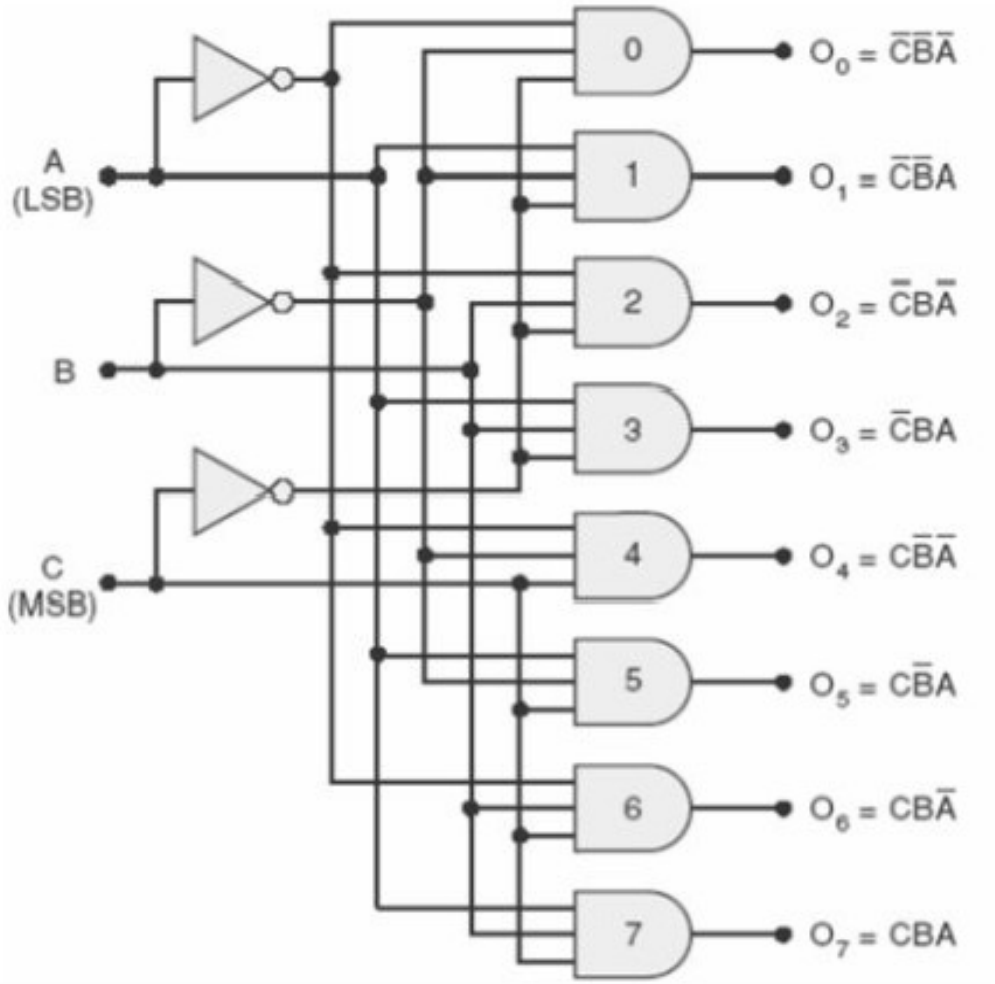
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY cod4to2 IS
PORT (   w    : IN      STD_LOGIC_VECTOR(3 DOWNTO 0) ;
         y    : OUT     STD_LOGIC_VECTOR(1 DOWNTO 0) ;
         v    : OUT     STD_LOGIC ) ;
END cod4to2 ;

ARCHITECTURE a OF cod4to2 IS
BEGIN
y  <= "11"       WHEN w(3) = '1' ELSE
      "10"       WHEN w(2) = '1' ELSE
      "01"       WHEN w(1) = '1' ELSE
      "00" ;
v <=   '0'       WHEN w = "0000" ELSE '1' ;
END a;
```

Decodificador

- El decodificador elige de entre varias funciones cuál se activa al aplicar en la variables de entrada el número (código) asignado a esa función.
- Este circuito tiene 2^n salidas si n es el número de variables de entrada y el código es binario natural.

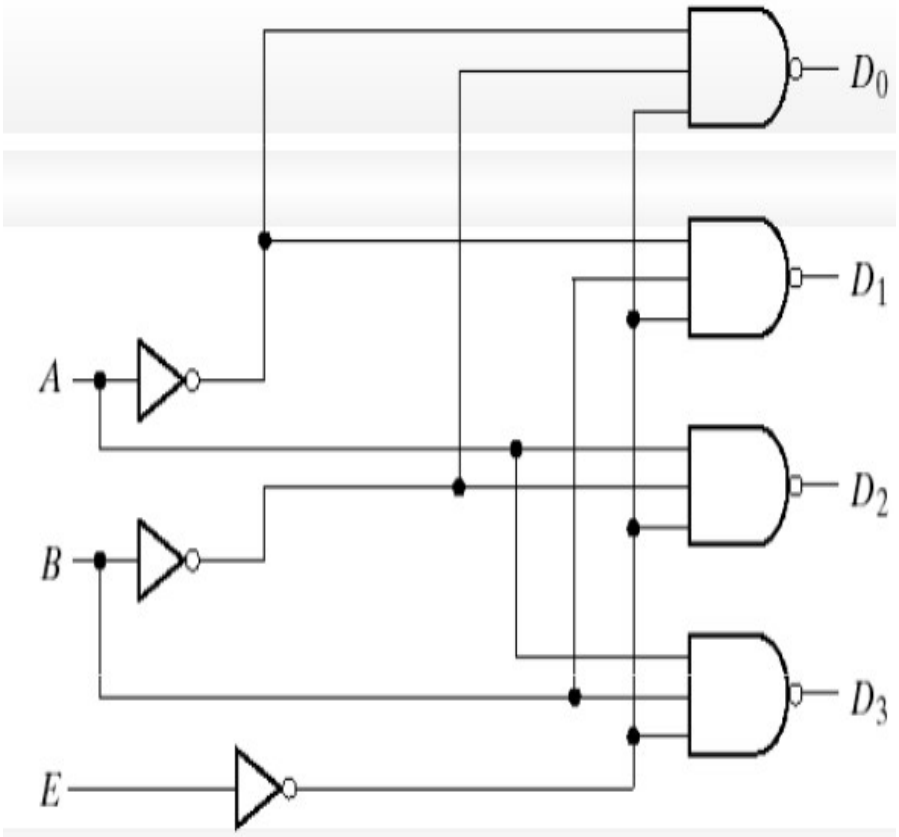


Decodificador de 3 entradas



C	B	A	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

Decodificador de 2 bit con entrada de habilitación



E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

$D_0 = (E \cdot A \cdot B)'$

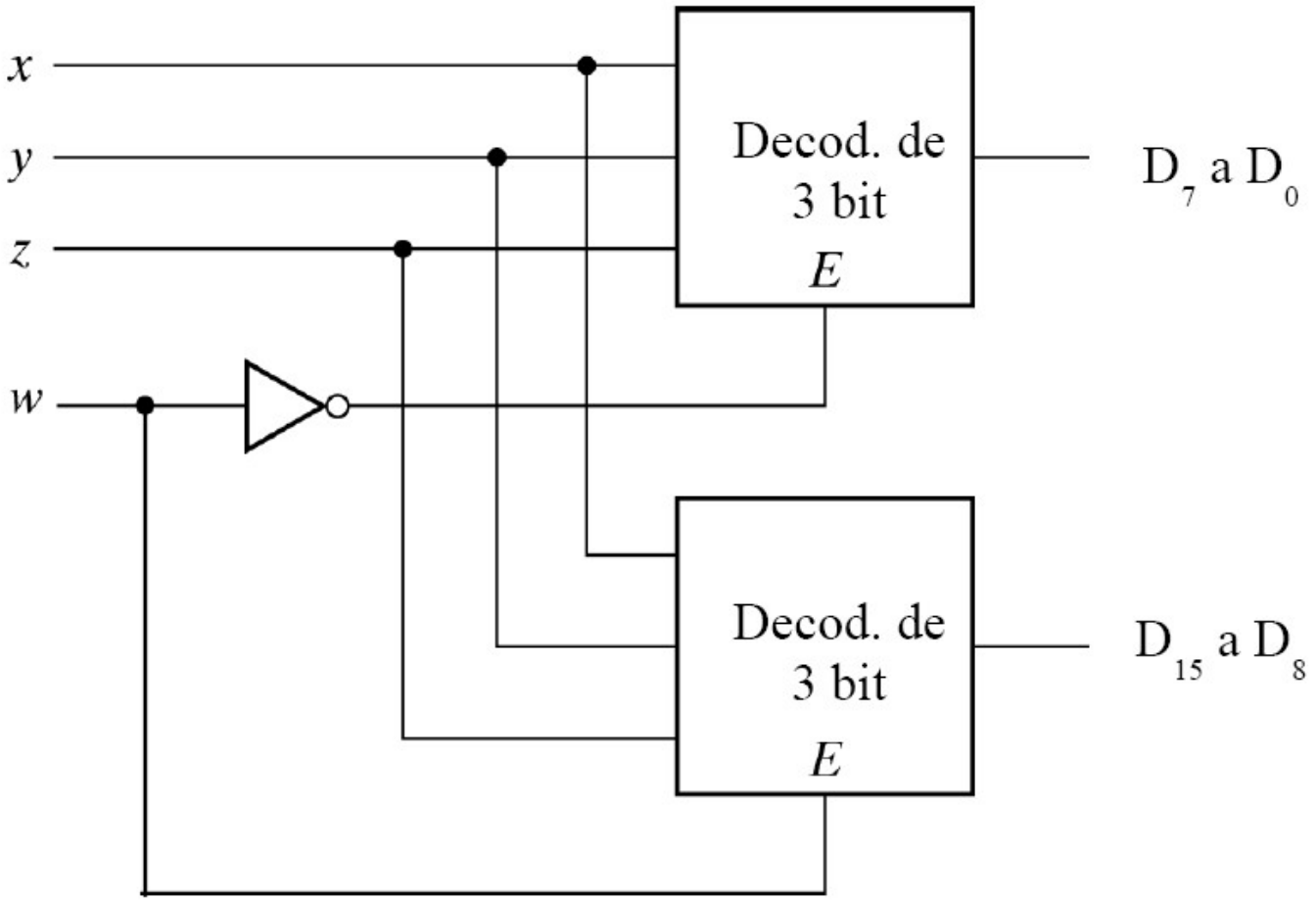
$D_1 = (E \cdot A \cdot B)'$

$D_2 = (E \cdot A \cdot B)'$

$D_3 = (E \cdot A \cdot B)'$

Estas salidas son en nivel bajo, por lo que aparece negada cada función

Decodificador de 4 entradas mediante 2 decodificadores de 3 entradas



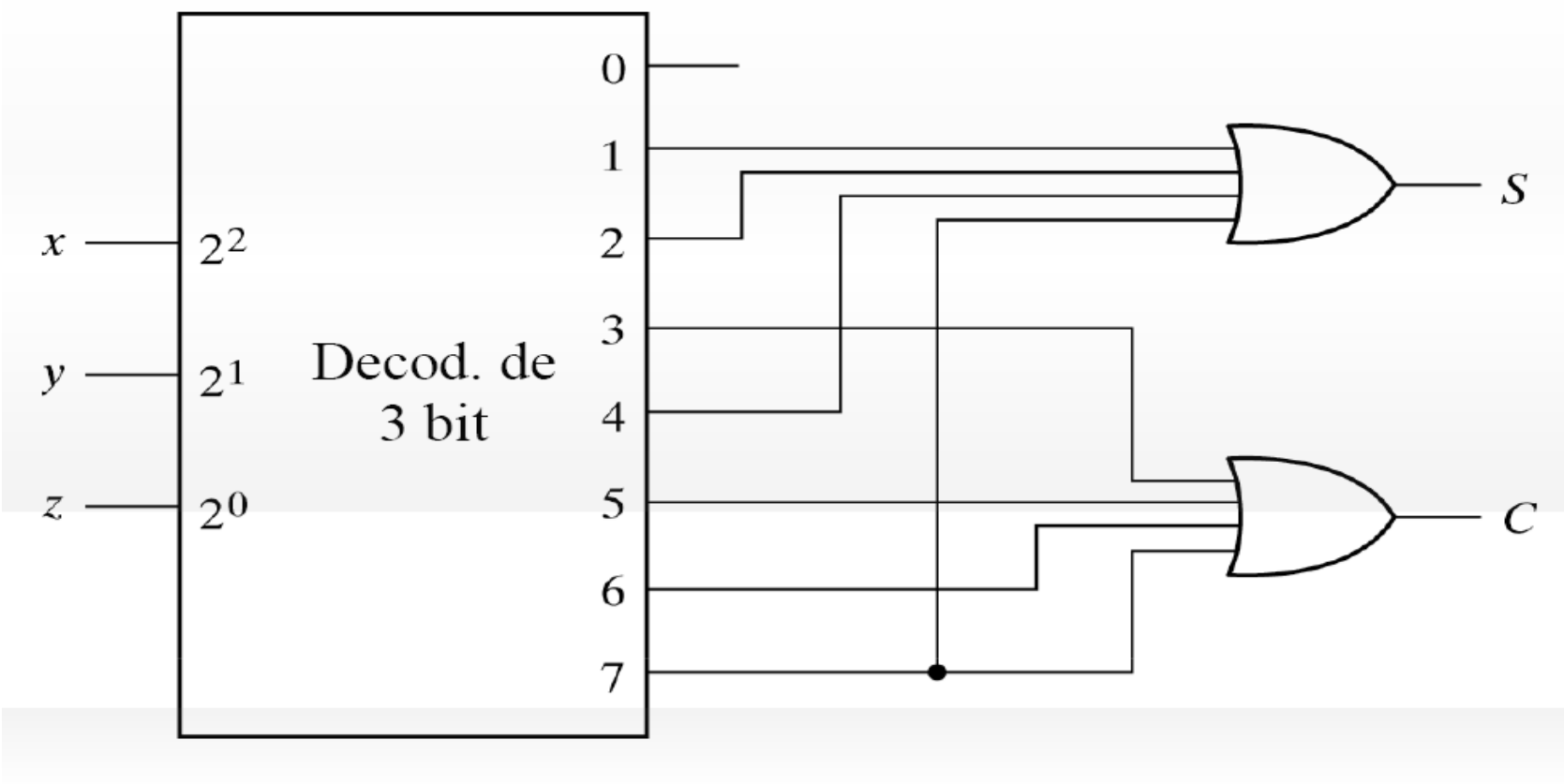
Este método se puede utilizar encualquier bloque funcional (diseño jerárquico).

Decodificador de 4 bits en VHDL

- Esta sería la descripción de un decodificador de 4 bits en VHDL:

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

ENTITY dec2to4 IS
PORT (   w    : IN       STD_LOGIC_VECTOR(1 DOWNTO 0) ;
         y    : OUT      STD_LOGIC_VECTOR(0 TO 3) ) ;
END dec2to4 ;

ARCHITECTURE a OF dec2to4 IS
BEGIN
WITH w SELECT
y <=     "1000" WHEN "00",
         "0100" WHEN "01",
         "0010" WHEN "10",
         "0001" WHEN "11",
         "0000" WHEN OTHERS ;
END a ;
```

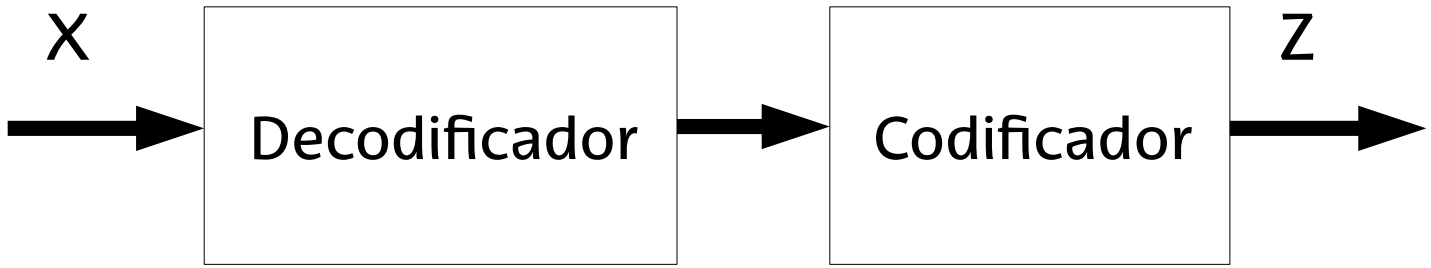
Decodificador para implementación de funciones: sumador completo



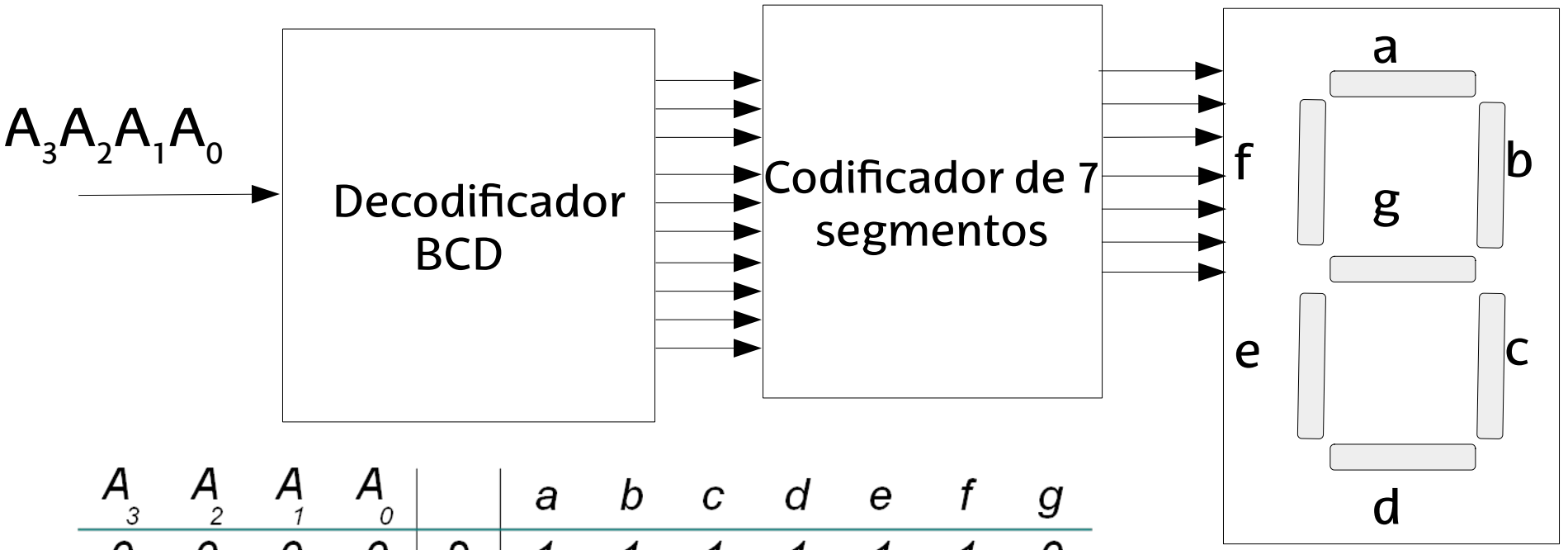
Cada salida del decodificador es un mintérmino, por lo que podemos usarla para representar cualquier expresión canónica, sumando las salidas correspondiente a mintérminos de la función con una puerta OR.

Convertidor de código

- Una aplicación típica de decodificador + codificador es el convertidor de código.
- Un dato numérico es introducido al decodificador para activar la salida correspondiente.
- La señal activada es la entrada para un codificador en otro código, obteniendo a la salida el dato codificado en el nuevo código.



Display de 7 segmentos



A_3	A_2	A_1	A_0		a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0	0	0	0
0	0	1	0	2	1	1	0	1	1	0	1
0	0	1	1	3	1	1	1	1	0	0	1
0	1	0	0	4	0	1	1	0	0	1	1
0	1	0	1	5	1	0	1	1	0	1	1
0	1	1	0	6	1	0	1	1	1	1	1
0	1	1	1	7	1	1	1	0	0	0	0
1	0	0	0	8	1	1	1	1	1	1	1
1	0	0	1	9	1	1	1	1	0	1	1

Display de 7 segmentos