Tema 4: Circuitos combinacionales II.

4.1: Introducción a los bloques funcionales combinacionales (comerciales).

4.2: Circuitos combinacionales MSI.

- 4.2.1: Decodificadores y demultiplexores.
- 4.2.2: Codificadores.
- 4.2.3: Multiplexores.
- 4.2.4: Comparadores.
- 4.2.5: Generadores / detectores de paridad.
- 4.2.6: Convertidores de código.
- 4.2.7: Circuitos aritméticos.

4.3: Análisis y síntesis de circuitos combinacionales utilizando circuitos integrados SSI y MSI.

"La libertad es el derecho a decirle a la gente lo que no quieren escuchar"

"If you hate violence and don't believe in politics, the only major remedy remaining is education".

George Orwell

4.1: Introducción.

- Escala SSI $\Leftrightarrow$ funciones sencillas (pocas).

- Escala MSI (hasta 100 puertas/chip) $\Rightarrow$ realización de una o varias funciones (complejas) de aplicación general.

- Ventajas de la utilización de ICs MSI:

_ Se reduce el número de ICs a utilizar.

_ Se reduce el coste del circuito.

_ Se simplifica el diseño (teórico).

_ Se simplifica el cableado del circuito y su construcción.

_ El circuito tiene un funcionamiento mucho mejor (fenómenos aleatorios y EMC)

- La filosofía de diseño cambió con la aparición de los ICs de la escala MSI:

  _ Objetivo *principal*: minimizar el número de ICs a utilizar.

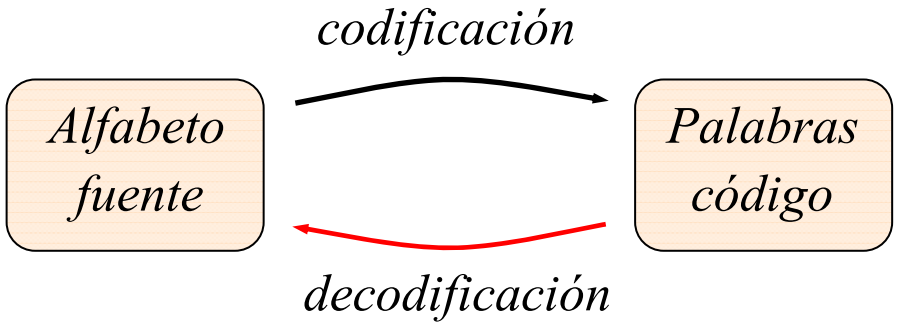
  _ Objetivo *secundario*: minimizar el número de puertas lógicas a utilizar.

- En este tema se estudian los principales bloques funcionales combinacionales pertenecientes a la escala de integración media (MSI). Dichos bloques *se utilizan habitualmente en la descripción de la arquitectura interna de circuitos digitales complejos como son, por ejemplo, los procesadores*.

- Decodificadores de *n* líneas de entrada a *m* líneas de salida
  de 1 línea de salida activa entre *m* líneas de salida

DEF. 1: Un decodificador es un circuito combinacional que decodifica información codificada en un código dado.

Un decodificador X/Y indica para cada palabra código de un código X el símbolo del código fuente Y que le corresponde.



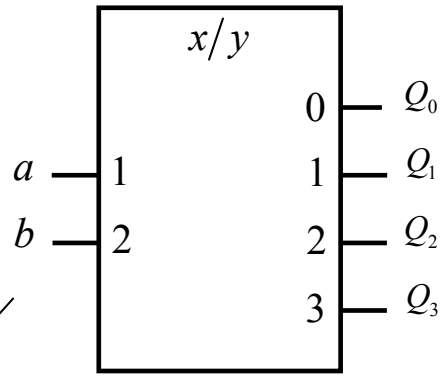
Ejemplo:

Alfabeto Fuente	Palabras código $BCD_{natural}$	Alfabeto código
0	0 0 0 0	0
1	0 0 0 1	1
2	0 0 1 0	
3	0 0 1 1	
4	0 1 0 0	
5	0 1 0 1	
6	0 1 1 0	
7	0 1 1 1	
8	1 0 0 0	
9	1 0 0 1	

DEF. 2: Un decodificador es un circuito combinacional con *n* entradas y *m* salidas. Para cada combinación (válida) de las *n* variables de entrada sólo se activa una de las *m* salidas.

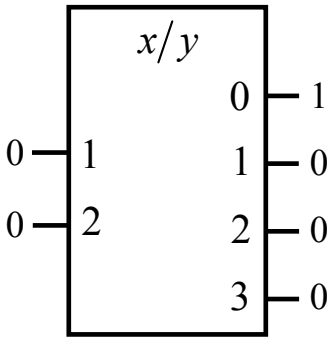
Nota: todos los decodificadores que se estudian en esta asignatura cumplen la definición anterior. Ahora bien, existen decodificadores, que no se estudian en esta asignatura por falta de tiempo, que no cumplen la definición anterior como, por ejemplo, un decodificador de *BCD* a 7 *segmentos*.

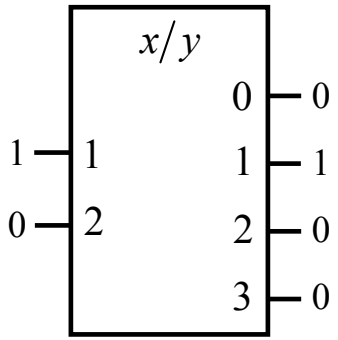
Decodificador de 2 *a* 4 *ó de* 1 *entre* 4

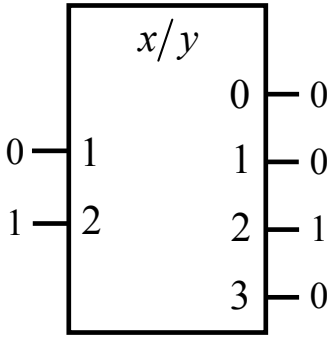


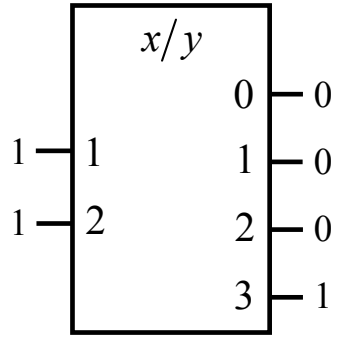
b	a	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Las entradas tienen asignados pesos (la entrada *b* tiene asignado el peso 2 y la entrada *a* tiene asignado el peso 1). Las salidas están numeradas empezando desde 0. Se pone a 1 la salida que tiene asignado un número igual a la suma de los pesos de las entradas que están a 1.

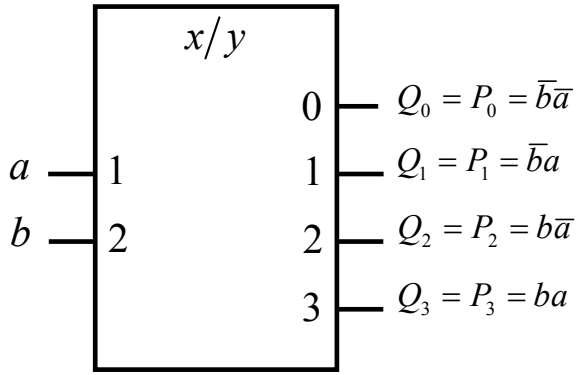


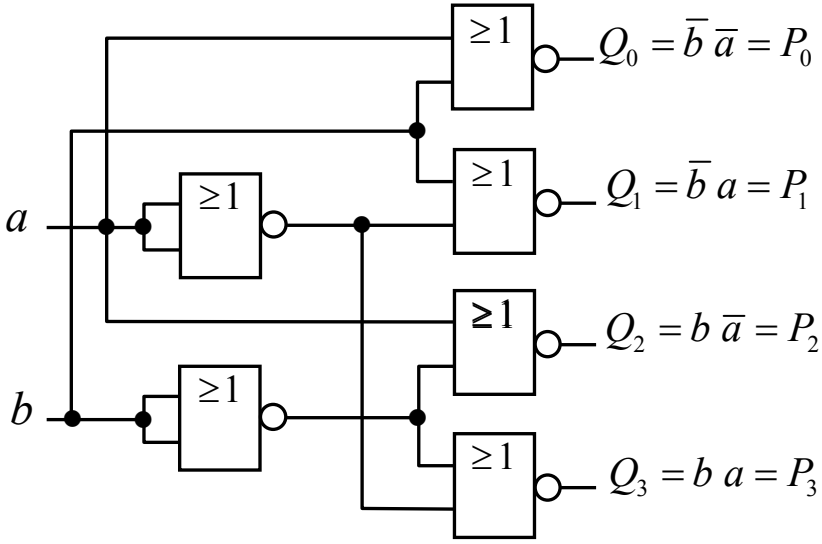




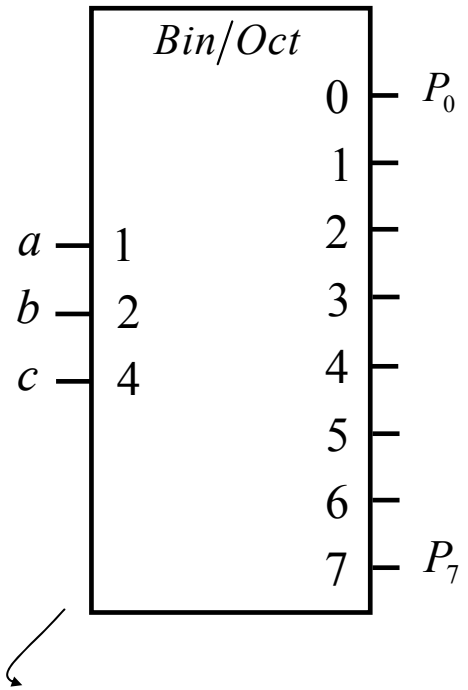


Implementación de un decodificador de 2 *a* 4 utilizando puertas NOR de 2 entradas





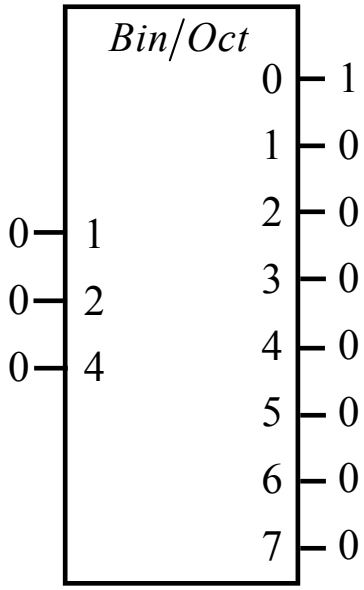
Decodificador de 3 *a* 8 *o de* 1 *entre* 8 ≡ *Decodificador octal*

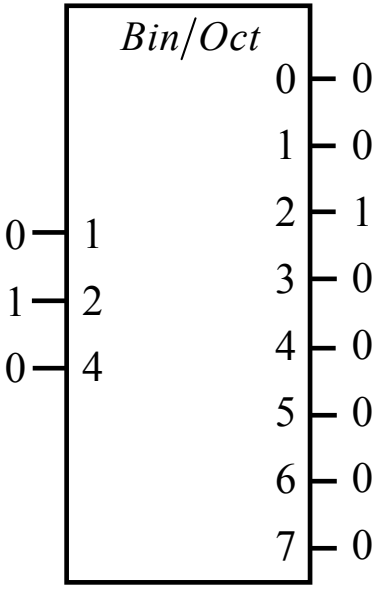


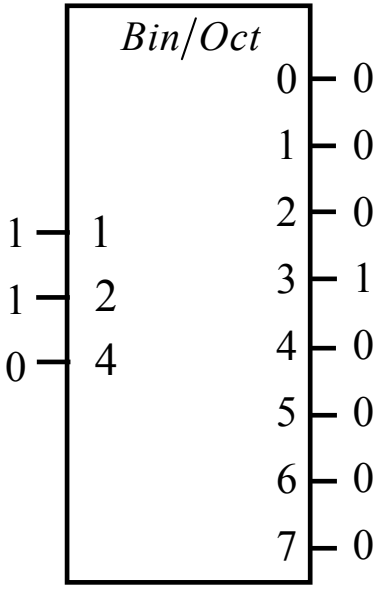
c	b	a	P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

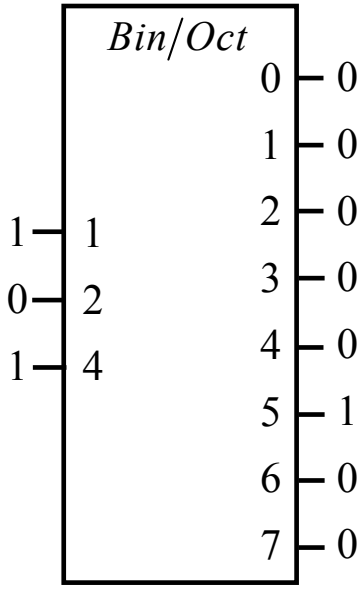
Se pone a 1 la salida que tiene asignado un número igual a la suma de los pesos de las entradas que están a 1. Los pesos asociados a las entradas *c*, *b* y *a* son 4, 2 y 1 respectivamente.

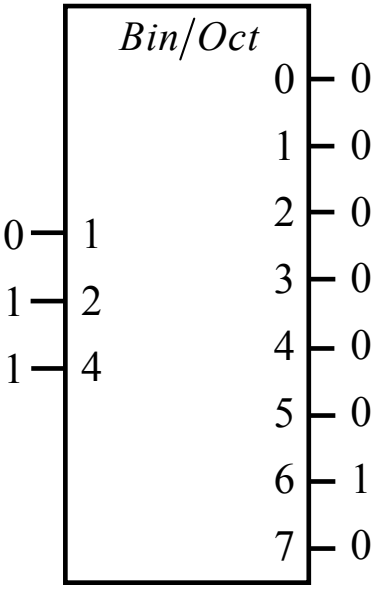
Ejemplos:

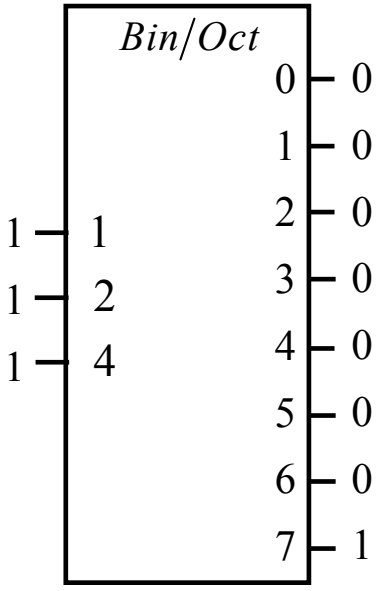

Bin/Oct
0 1
1 0
2 0
3 0
4 0
5 0
6 0
7 0
0 1
0 2
0 4


Bin/Oct
0 0
1 0
2 1
3 0
4 0
5 0
6 0
7 0
0 1
1 2
0 4

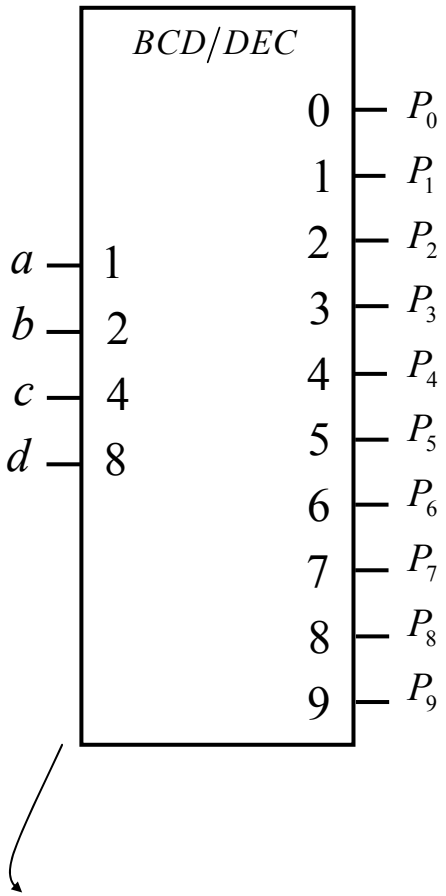

Bin/Oct
0 0
1 0
2 0
3 1
4 0
5 0
6 0
7 0
1 1
1 2
0 4


Bin/Oct
0 0
1 0
2 0
3 0
4 0
5 1
6 0
7 0
1 1
0 2
1 4


Bin/Oct
0 0
1 0
2 0
3 0
4 0
5 0
6 1
7 0
0 1
1 2
1 4


Bin/Oct
0 0
1 0
2 0
3 0
4 0
5 0
6 0
7 1
1 1
1 2
1 4

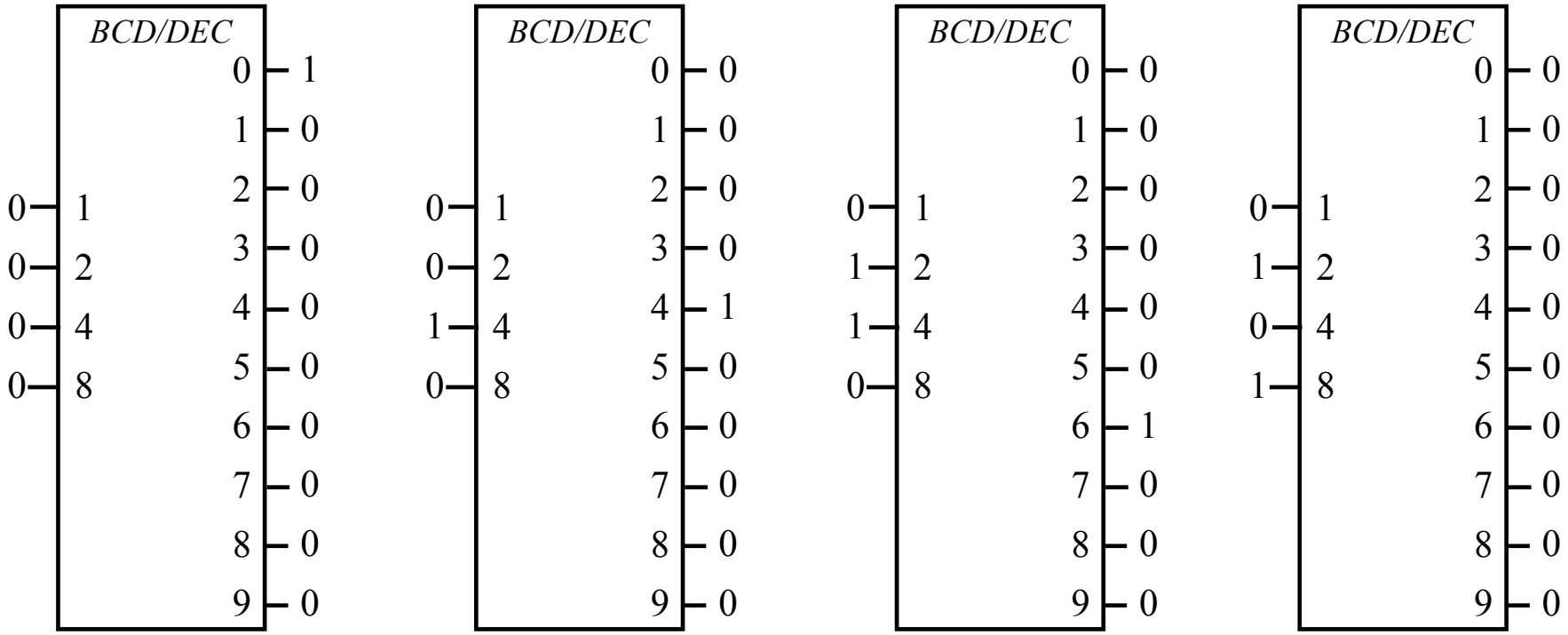
Decodificador decimal



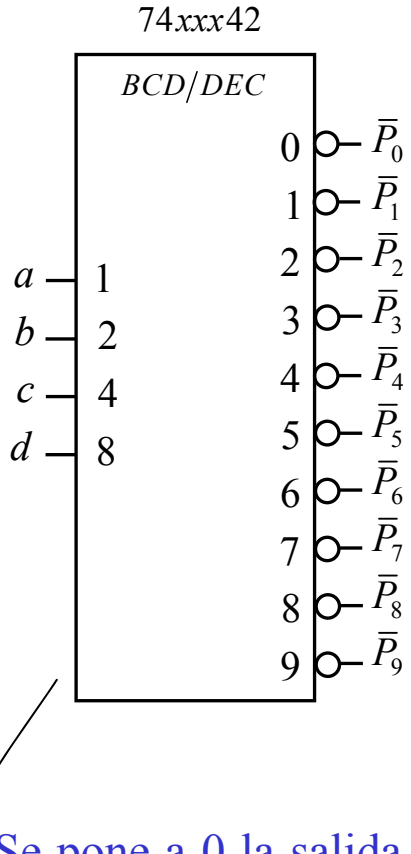
d	c	b	a	P_9	P_8	P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0
0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0

Se pone a 1 la salida que tiene asignado un número igual a la suma de los pesos de las entradas que están a 1. Los pesos asociados a las entradas *d*, *c*, *b* y *a* son 8, 4, 2 y 1 respectivamente. El mayor valor que se puede decodificar es el 1001_2

Ejemplos:


BCD/DEC
0 1
0 2
0 4
0 8
0 1
1 2
2 3
3 4
4 5
5 6
6 7
7 8
8 9
9
BCD/DEC
0 1
0 2
1 4
0 8
0 1
1 2
2 3
3 4
4 1
5 5
6 6
7 7
8 8
9
BCD/DEC
0 1
1 2
1 4
0 8
0 1
1 2
2 3
3 4
4 5
5 6
6 1
7 7
8 8
9
BCD/DEC
0 1
1 2
0 4
1 8
0 1
1 2
2 3
3 4
4 5
5 6
6 7
7 8
8 9
9

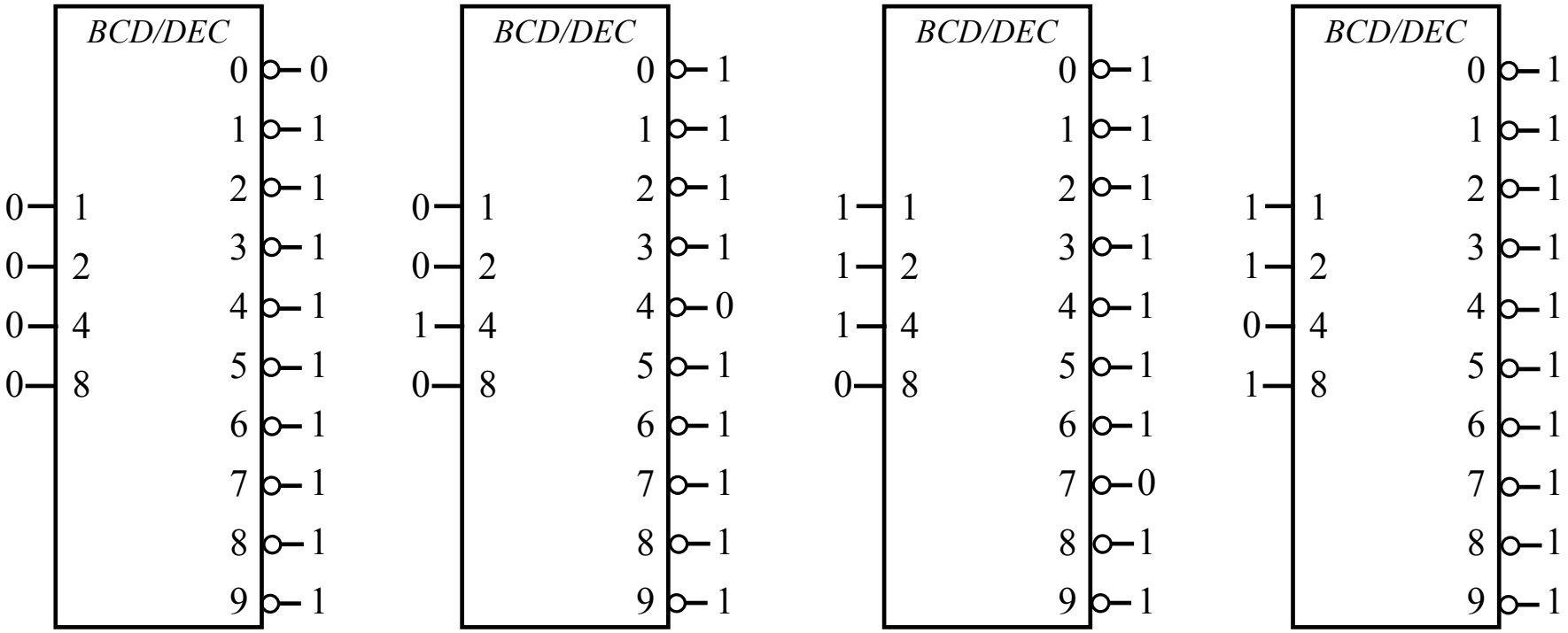
Decodificador decimal



d	c	b	a	$\overline{P}_9$	$\overline{P}_8$	$\overline{P}_7$	$\overline{P}_6$	$\overline{P}_5$	$\overline{P}_4$	$\overline{P}_3$	$\overline{P}_2$	$\overline{P}_1$	$\overline{P}_0$
0	0	0	0	1	1	1	1	1	1	1	1	1	0
0	0	0	1	1	1	1	1	1	1	1	1	0	1
0	0	1	0	1	1	1	1	1	1	1	0	1	1
0	0	1	1	1	1	1	1	1	1	0	1	1	1
0	1	0	0	1	1	1	1	1	0	1	1	1	1
0	1	0	1	1	1	1	1	0	1	1	1	1	1
0	1	1	0	1	1	1	0	1	1	1	1	1	1
0	1	1	1	1	1	0	1	1	1	1	1	1	1
1	0	0	0	1	0	1	1	1	1	1	1	1	1
1	0	0	1	0	1	1	1	1	1	1	1	1	1
1	0	1	0	1	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1	1	1	1	1
1	1	0	0	1	1	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1

Se pone a 0 la salida que tienen asignado un número igual a la suma de los pesos de las entradas que están a 1. Los pesos asociados a las entradas *d*, *c*, *b* y *a* son 8, 4, 2 y 1 respectivamente. El mayor valor que se puede decodificar es el 1001_2

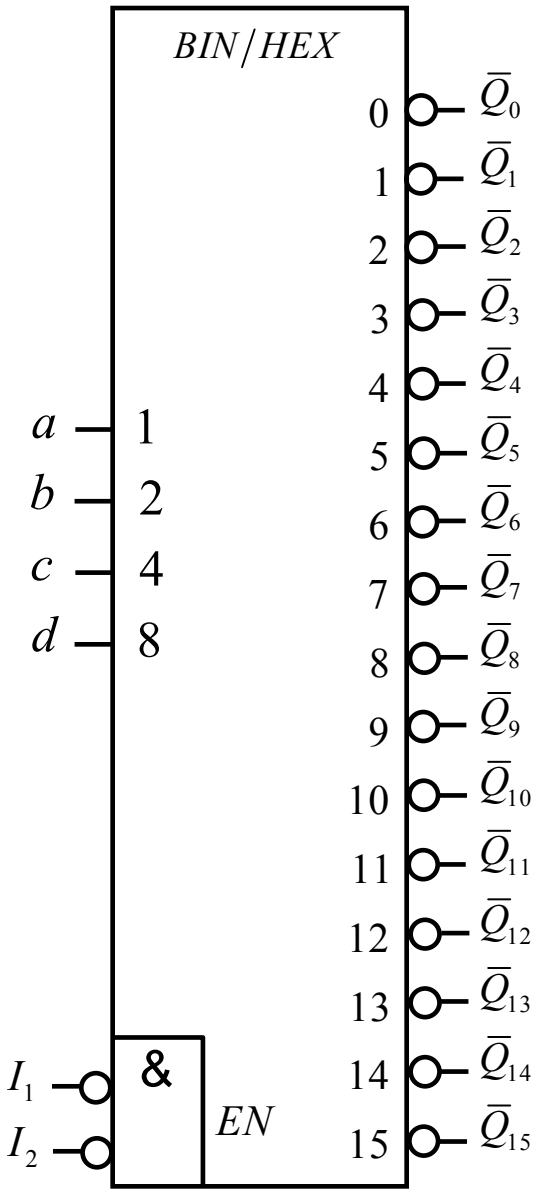
Ejemplos:


BCD/DEC
0 1 2 3 4 5 6 7 8 9
1 2 4 8
0 0 0 0
0 1 1 1 1 1 1 1 1 1
BCD/DEC
0 1 2 3 4 5 6 7 8 9
1 2 4 8
0 0 1 0
1 1 1 1 0 1 1 1 1 1
BCD/DEC
0 1 2 3 4 5 6 7 8 9
1 2 4 8
1 1 0 0
1 1 1 1 1 1 0 1 1 1
BCD/DEC
0 1 2 3 4 5 6 7 8 9
1 2 4 8
1 1 0 1
1 1 1 1 1 1 1 1 1 1

no

Decodificador hexadecimal

74*xxx*154

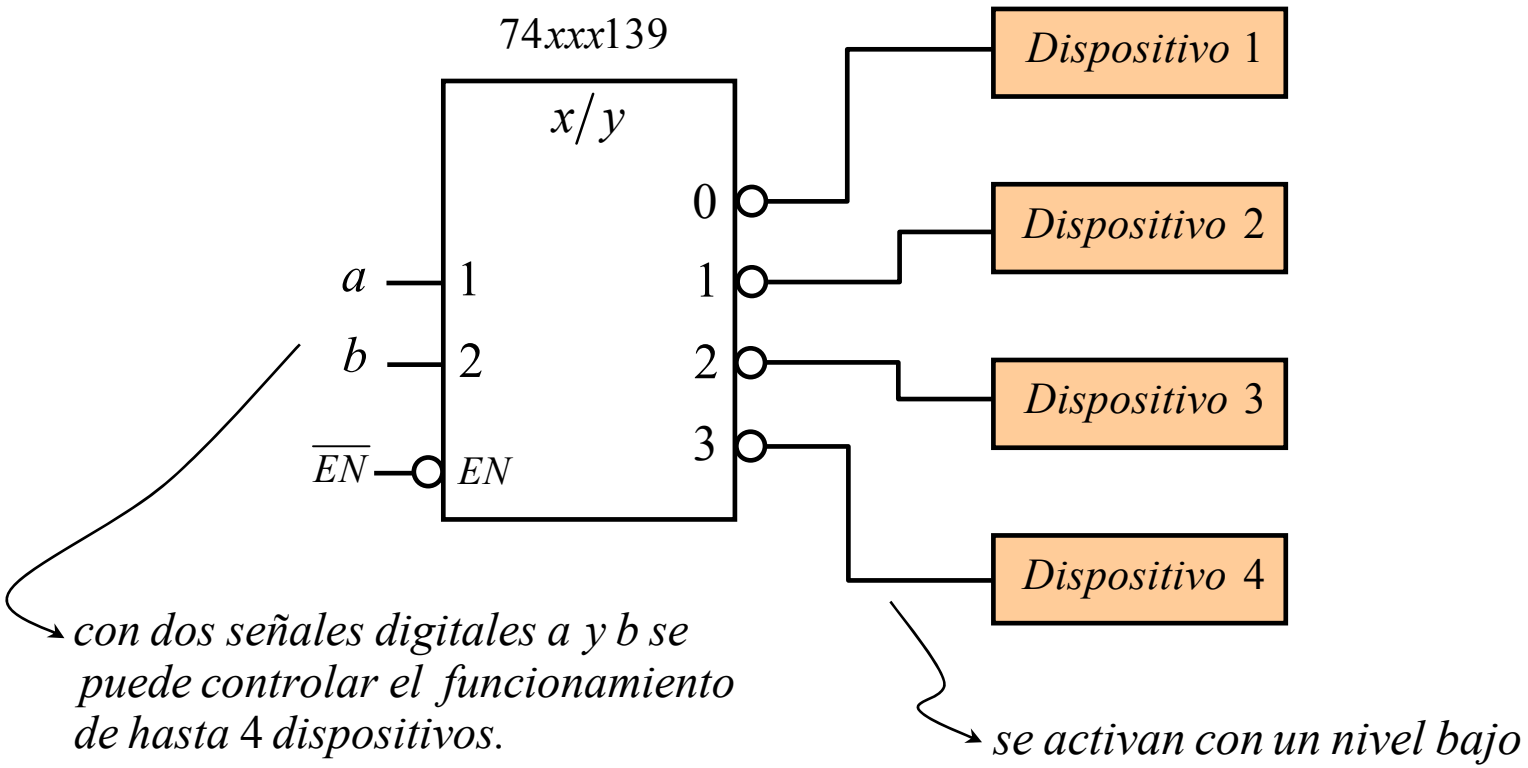


no

I_2	I_1	d	c	b	a	$\bar{Q}_{15}$	$\bar{Q}_{14}$	$\bar{Q}_{13}$	$\bar{Q}_{12}$	$\bar{Q}_{11}$	$\bar{Q}_{10}$	$\bar{Q}_9$	$\bar{Q}_8$	$\bar{Q}_7$	$\bar{Q}_6$	$\bar{Q}_5$	$\bar{Q}_4$	$\bar{Q}_3$	$\bar{Q}_2$	$\bar{Q}_1$	$\bar{Q}_0$
1	1	x	x	x	x	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	x	x	x	x	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	1	x	x	x	x	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
0	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
0	0	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
0	0	1	0	0	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
0	0	1	0	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
0	0	1	0	1	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
0	0	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
0	0	1	1	0	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
0	0	1	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Aplicaciones:

- ✓ Decodificar palabras código.
- ✓ Control del funcionamiento no simultáneo de varios dispositivos.



✓ Construir decodificadores de otros tipos de códigos: por ejemplo, utilizando decodificadores decimales se puede construir:

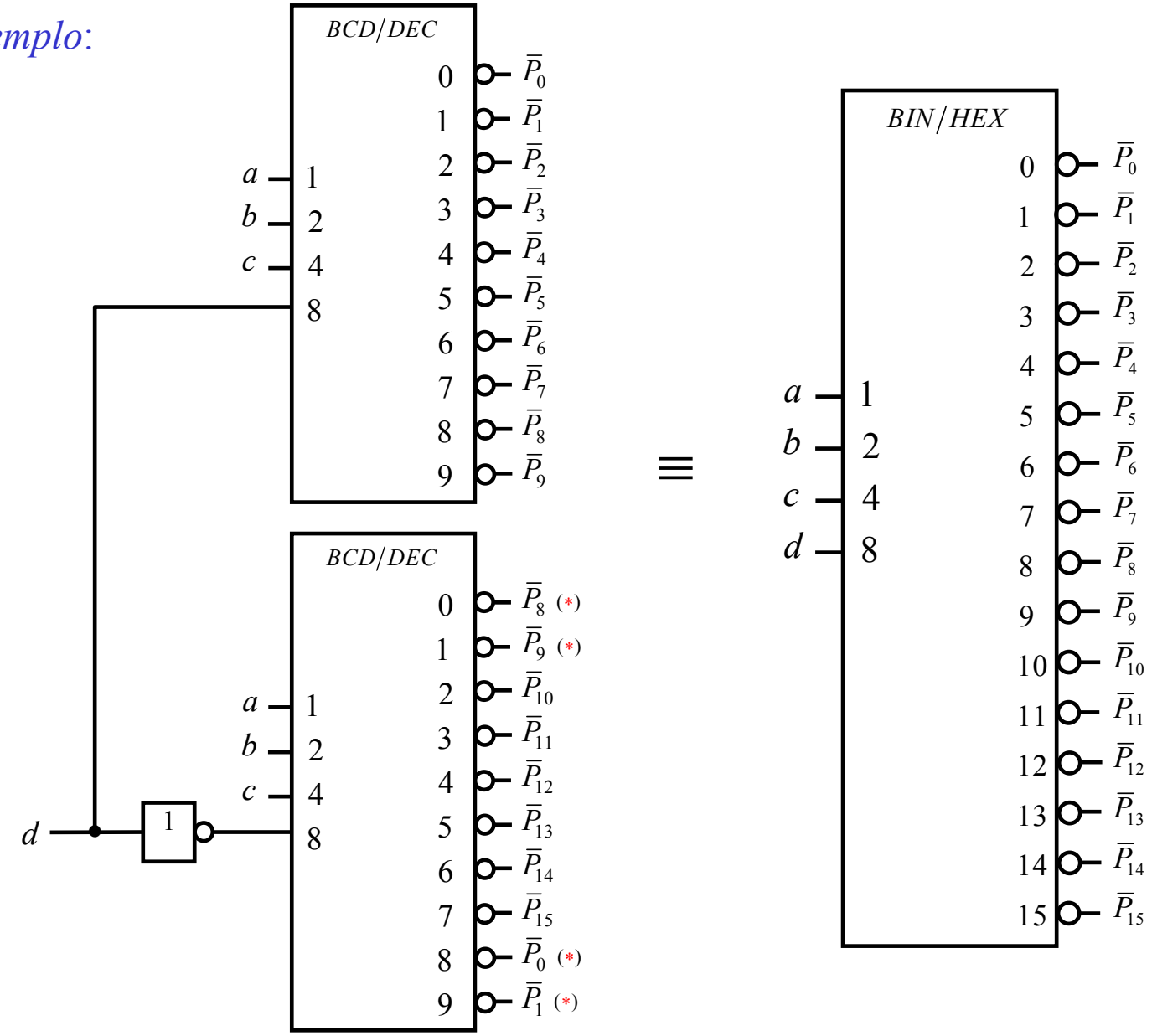
_ Un decodificador de cualquier código BCD (Aiken, exceso 3, etc)

_ Un decodificador hexadecimal

_ etc.

no

Ejemplo:


BCD/DEC
a
b
c
d
1
2
4
8
0
1
2
3
4
5
6
7
8
9
$\overline{P}_0$
$\overline{P}_1$
$\overline{P}_2$
$\overline{P}_3$
$\overline{P}_4$
$\overline{P}_5$
$\overline{P}_6$
$\overline{P}_7$
$\overline{P}_8$
$\overline{P}_9$
BCD/DEC
$\overline{P}_8$ (*)
$\overline{P}_9$ (*)
$\overline{P}_{10}$
$\overline{P}_{11}$
$\overline{P}_{12}$
$\overline{P}_{13}$
$\overline{P}_{14}$
$\overline{P}_{15}$
$\overline{P}_0$ (*)
$\overline{P}_1$ (*)
≡
BIN/HEX
$\overline{P}_0$
$\overline{P}_1$
$\overline{P}_2$
$\overline{P}_3$
$\overline{P}_4$
$\overline{P}_5$
$\overline{P}_6$
$\overline{P}_7$
$\overline{P}_8$
$\overline{P}_9$
$\overline{P}_{10}$
$\overline{P}_{11}$
$\overline{P}_{12}$
$\overline{P}_{13}$
$\overline{P}_{14}$
$\overline{P}_{15}$

✓ Implementar funciones lógicas (en algunos casos se consigue reducir el número de ICs a utilizar).

Ejemplo 1: Diseñar un circuito que implemente las siguientes funciones, utilizando un decodificador adecuado y puertas NAND

$$f_1(c,b,a) = \sum_3 (0,1,3,5)$$

$$f_2(c,b,a) = \sum_3 (5,6,7) + \sum_\phi (1,4)$$

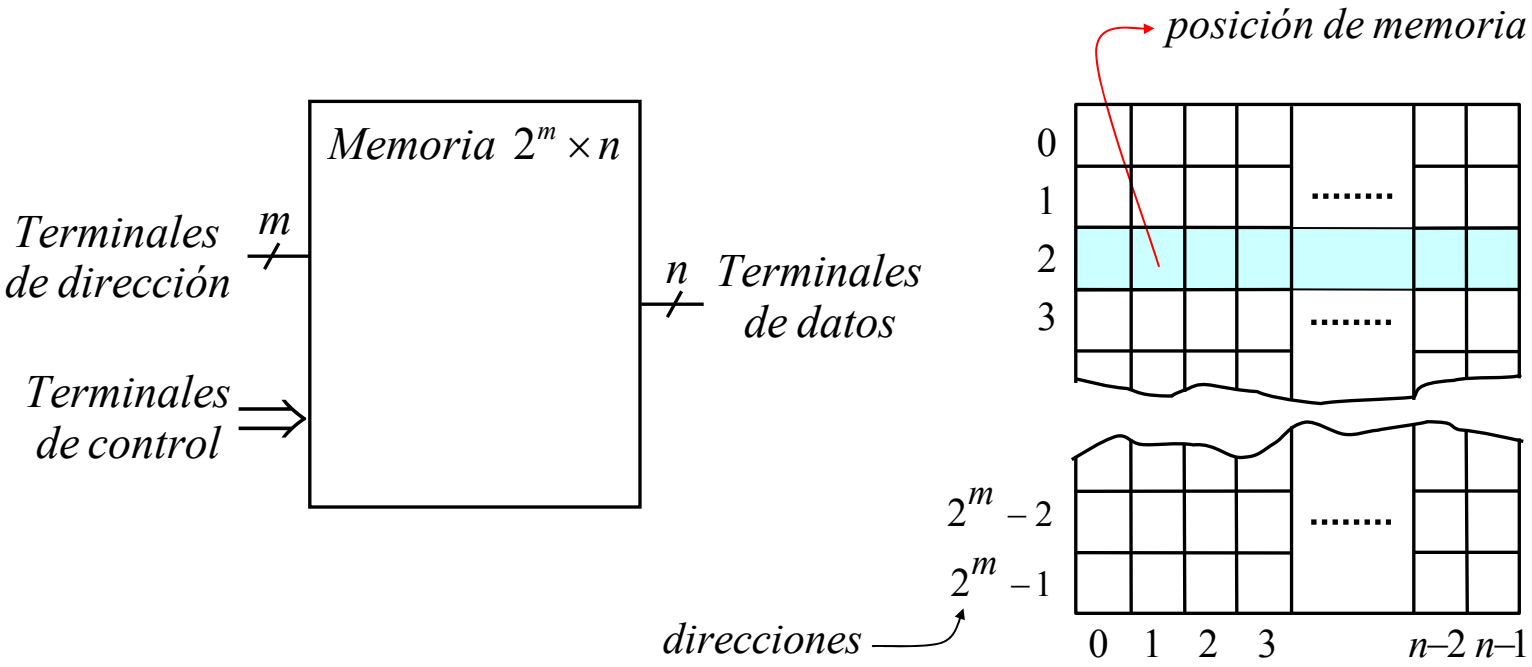
Ejemplo 2: Diseñar un circuito que implemente la siguiente función, utilizando un decodificador decimal y puertas NAND

$$g(b,a) = \sum_2 (0,1,2)$$

Ejemplo 3: Diseñar un circuito que implemente la siguiente función, utilizando un decodificador adecuado y el menor número posible de puertas NAND de 2 entradas.

$$h(d,c,b,a) = \overline{\overline{\overline{d} \cdot c \cdot \overline{a}} \cdot (c + b + \overline{a})}$$

✓ Decodificar las direcciones presentes en el bus de direcciones de una memoria

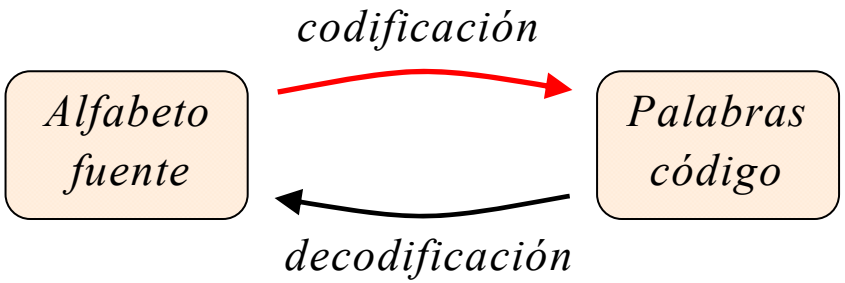


_ *La memoria tiene* 2^m *posiciones. Cada posición tiene asociada una dirección. La posición en la que se lee o se escribe se determina decodificando la dirección presente en los terminales de dirección (bus de direcciones).*

_ *Cada posición o dirección de memoria guarda n bits* ($\equiv 1$ *palabra* $\equiv 1$ *word*)

- Codificadores de *n* líneas de entrada a *m* líneas de salida

DEF.: Un codificador es un circuito combinacional que proporciona para cada símbolo de un alfabeto fuente dado la palabra código que le corresponde según el código que se considere.



Hay dos tipos de codificadores: *con prioridad* y *sin prioridad*. Su funcionamiento es el mismo excepto en el caso de que se activen simultáneamente dos o más entradas. Los codificadores *sin prioridad* no se utilizan.

Ejemplo:

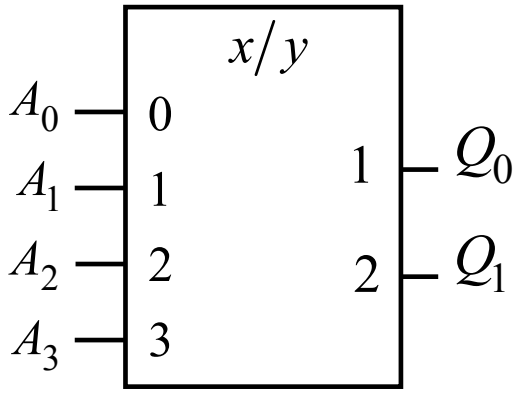
Alfabeto Fuente	Palabras código $BCD_{natural}$	Alfabeto código
0	0 0 0 0	0
1	0 0 0 1	1
2	0 0 1 0	
3	0 0 1 1	
4	0 1 0 0	
5	0 1 0 1	
6	0 1 1 0	
7	0 1 1 1	
8	1 0 0 0	
9	1 0 0 1	

Los codificadores que se estudian en esta asignatura proporcionan la palabra código correspondiente a la entrada activa que tiene el mayor valor asociado (*codificadores con prioridad*).

Nota: un codificador realiza la función inversa de la que realiza un decodificador

no

Codificadores sin prioridad: cuando dos o más entradas están activas simultáneamente, en la salida se muestra la *suma lógica* de las codificaciones correspondientes a cada una de las entradas activas.



A_3	A_2	A_1	A_0	Q_1	Q_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
		⋮			⋮
0	1	1	0	1	1
		⋮			⋮
0	0	0	0	0	0

Ejemplo: si se cumple que $A_3\,A_2\,A_1\,A_0 = 0110$, en la salida aparece el valor $Q_1\,Q_0 = 11$, ya que:

$$\left.\begin{array}{l} A_1 \equiv 01 \\ A_2 \equiv 10 \end{array}\right\} \rightarrow 01+10 = 11$$

suma lógica (bit a bit)

$$\begin{array}{r} 01 \\ +\ 10 \\ \hline 11 \end{array}$$

no

Problemas:

✓ Si se activan simultáneamente dos o más entradas, en la salida aparece una codificación errónea.

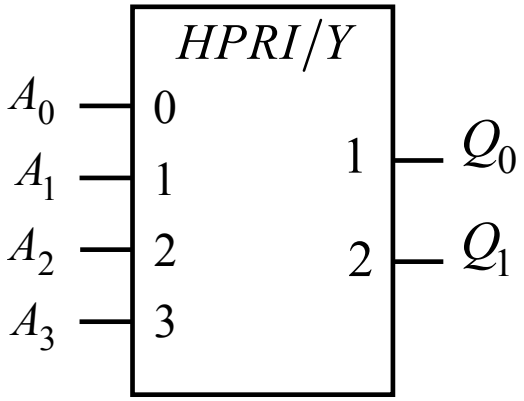
✓ Observando únicamente las salidas (Q_1 y Q_0) no se puede distinguir entre las siguientes condiciones:

A_3	A_2	A_1	A_0	Q_1	Q_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
⋮				⋮	
0	1	1	0	1	1
⋮				⋮	
0	0	0	0	0	0

$$(A_3A_2A_1A_0) = 0000 \rightarrow (Q_1Q_0) = 00$$

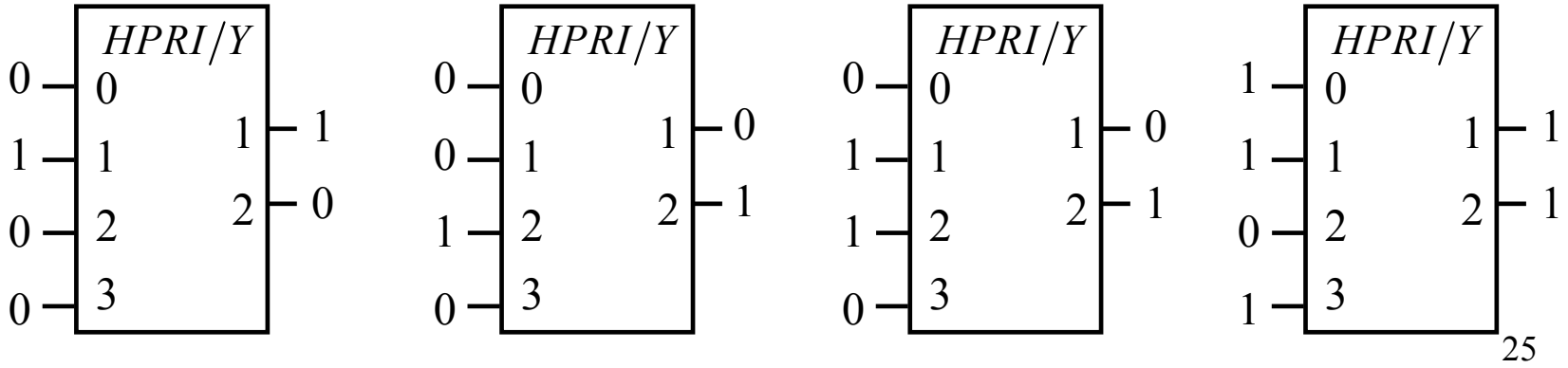
$$(A_3A_2A_1A_0) = 0001 \rightarrow (Q_1Q_0) = 00$$

Codificadores con prioridad (o de *alta prioridad* ≡ *high priority*): cuando se activan simultáneamente dos o más entradas, en la salida aparece la codificación correspondiente a la entrada activa que tenga asignado el valor más alto.

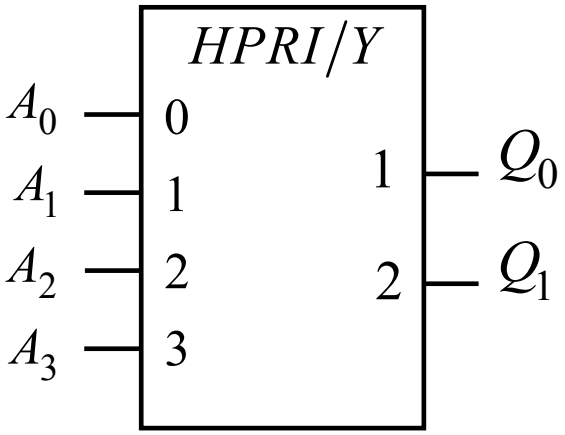


A_3	A_2	A_1	A_0	Q_1	Q_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
⋮				⋮	
0	1	1	0	1	0
⋮				⋮	
0	0	0	0	0	0

Ejemplos:



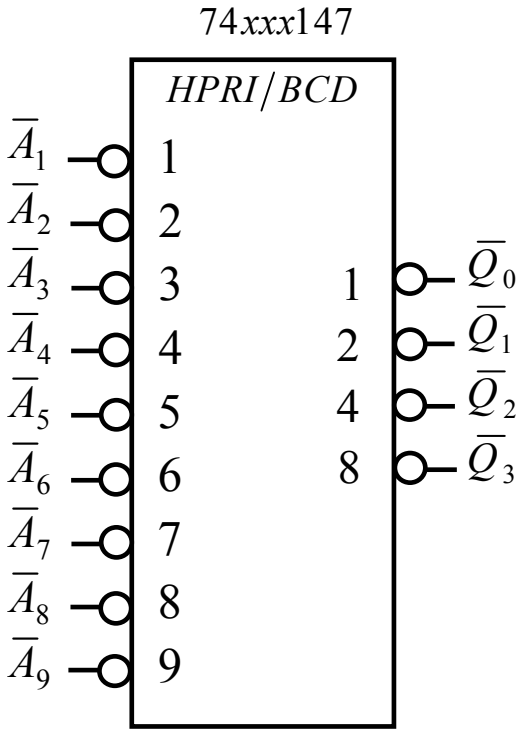
Problema: observando las salidas no se puede distinguir entre la situación en la que sólo está activa la entrada que tiene asignado el valor 0 y la situación en la que no hay ninguna entrada activa.



A_3	A_2	A_1	A_0	Q_1	Q_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
⋮				⋮	
0	1	1	0	1	0
⋮				⋮	
0	0	0	0	0	0

1ª *solución*: Omitir la entrada de menor peso/prioridad

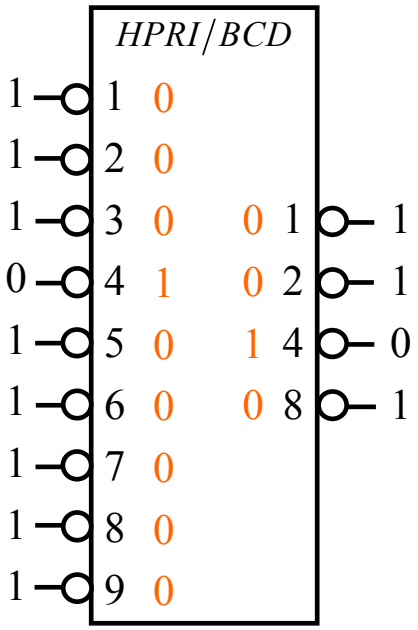
Ejemplo: Codificador con prioridad de Decimal a $BCD_{natural}$

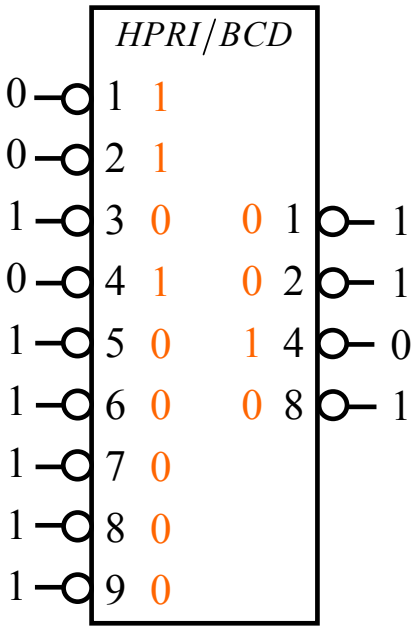


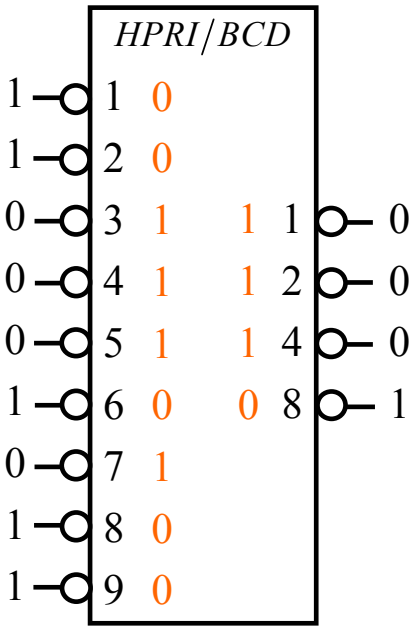
$\overline{A}_9$	$\overline{A}_8$	$\overline{A}_7$	$\overline{A}_6$	$\overline{A}_5$	$\overline{A}_4$	$\overline{A}_3$	$\overline{A}_2$	$\overline{A}_1$	$\overline{Q}_3$	$\overline{Q}_2$	$\overline{Q}_1$	$\overline{Q}_0$
0	x	x	x	x	x	x	x	x	0	1	1	0
1	0	x	x	x	x	x	x	x	0	1	1	1
1	1	0	x	x	x	x	x	x	1	0	0	0
1	1	1	0	x	x	x	x	x	1	0	0	1
1	1	1	1	0	x	x	x	x	1	0	1	0
1	1	1	1	1	0	x	x	x	1	0	1	1
1	1	1	1	1	1	0	x	x	1	1	0	0
1	1	1	1	1	1	1	0	x	1	1	0	1
1	1	1	1	1	1	1	1	0	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1	1	1

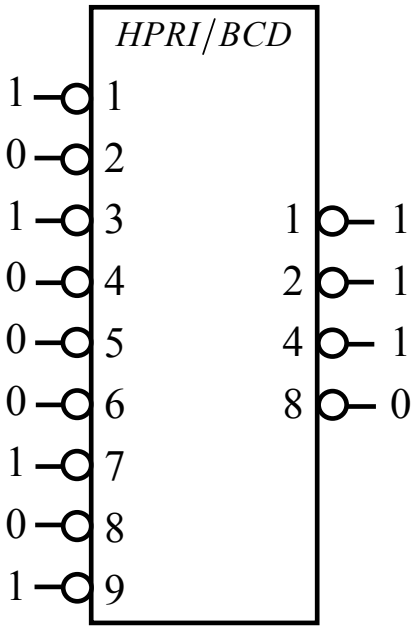
Las *entradas* están numeradas consecutivamente, en este caso, empezando en 1. Las *salidas* tienen asignados *pesos*. Los *estados lógicos internos* de las salidas toman, de acuerdo con los pesos que tienen asignados, el valor correspondiente al número más alto de todas las entradas que estén activas (entradas cuyo estado lógico interno sea igual a 1)

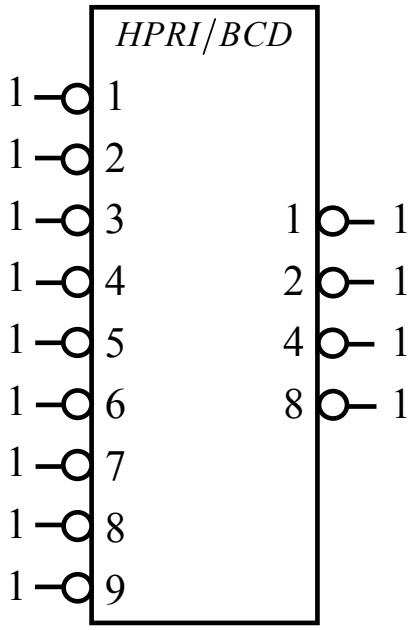
Ejemplos:

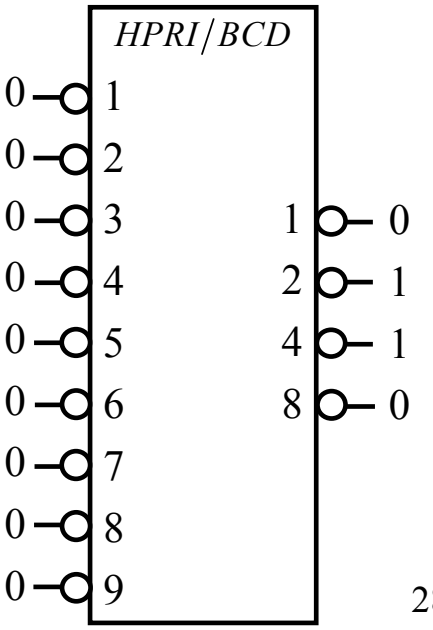

HPRI/BCD
1 1 0
1 2 0
1 3 0
0 4 1
1 5 0
1 6 0
1 7 0
1 8 0
1 9 0
0 1 1
0 2 1
1 4 0
0 8 1


HPRI/BCD
0 1 1
0 2 1
1 3 0
0 4 1
1 5 0
1 6 0
1 7 0
1 8 0
1 9 0
0 1 1
0 2 1
1 4 0
0 8 1


HPRI/BCD
1 1 0
1 2 0
0 3 1
0 4 1
0 5 1
1 6 0
0 7 1
1 8 0
1 9 0
1 1 0
1 2 0
1 4 0
0 8 1


HPRI/BCD
1 1
0 2
1 3
0 4
0 5
0 6
1 7
0 8
1 9
1 1
2 1
4 1
8 0

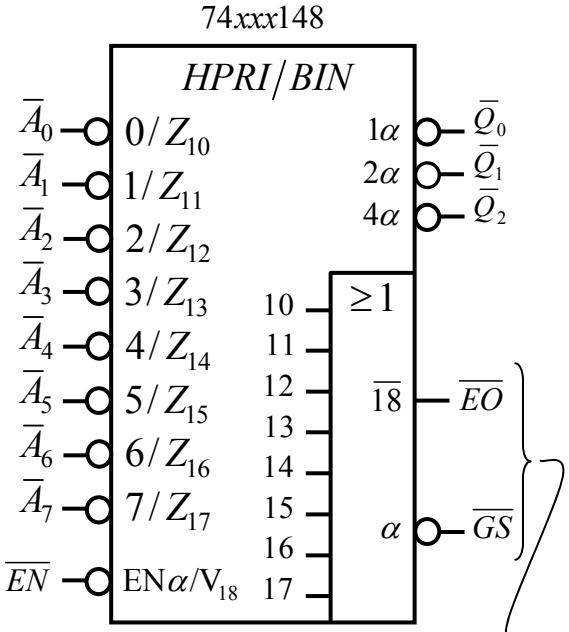

HPRI/BCD
1 1
1 2
1 3
1 4
1 5
1 6
1 7
1 8
1 9
1 1
2 1
4 1
8 1


HPRI/BCD
0 1
0 2
0 3
0 4
0 5
0 6
0 7
0 8
0 9
1 0
2 1
4 1
8 0

2ª *solución*: Añadir salidas auxiliares

no

Ejemplo: Codificador con prioridad de *Octal* a *Binario*



$\overline{EN}$	$\overline{A}_7$	$\overline{A}_6$	$\overline{A}_5$	$\overline{A}_4$	$\overline{A}_3$	$\overline{A}_2$	$\overline{A}_1$	$\overline{A}_0$	$\overline{Q}_2$	$\overline{Q}_1$	$\overline{Q}_0$	$\overline{EO}$	$\overline{GS}$
1	x	x	x	x	x	x	x	x	1	1	1	1	1
0	0	x	x	x	x	x	x	x	0	0	0	1	0
0	1	0	x	x	x	x	x	x	0	0	1	1	0
0	1	1	0	x	x	x	x	x	0	1	0	1	0
0	1	1	1	0	x	x	x	x	0	1	1	1	0
0	1	1	1	1	0	x	x	x	1	0	0	1	0
0	1	1	1	1	1	0	x	x	1	0	1	1	0
0	1	1	1	1	1	1	0	x	1	1	0	1	0
0	1	1	1	1	1	1	1	0	1	1	1	1	0
0	1	1	1	1	1	1	1	1	1	1	1	0	1

salidas auxiliares: *observando estas salidas se puede distinguir entre el caso en el que no hay ninguna entrada activa del caso en el que la sólo está activa la entrada* 0

$$\overline{EO} = A_0 + A_1 + A_2 + A_3 + A_4 + A_5 + A_6 + A_7 + \overline{EN}$$

$$\overline{GS} = \overline{\left(A_0 + A_1 + A_2 + A_3 + A_4 + A_5 + A_6 + A_7\right) \cdot EN} = \overline{A}_0 \cdot \overline{A}_1 \cdot \overline{A}_2 \cdot \overline{A}_3 \cdot \overline{A}_4 \cdot \overline{A}_5 \cdot \overline{A}_6 \cdot \overline{A}_7 + \overline{EN}$$

- Multiplexores de *n* canales (de entrada)

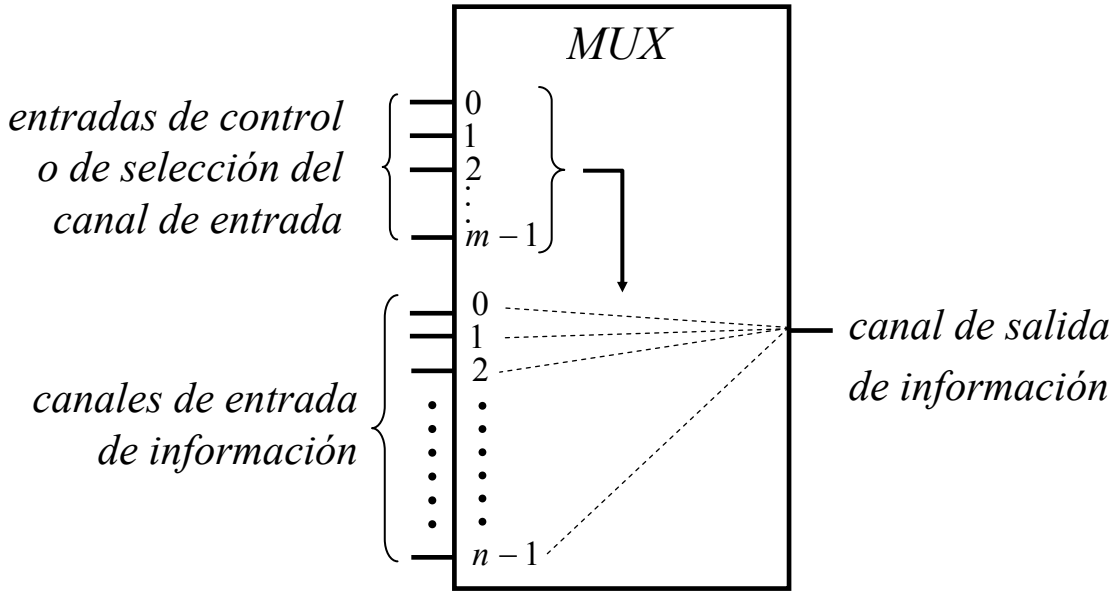
DEF.: Un multiplexor es un circuito combinacional que tiene:

_ 1 salida (canal) de información.

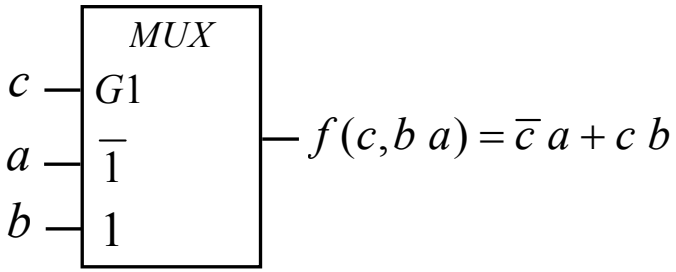
_ *m* entradas de control o de selección (de uno de los canales de entrada).

_ *n* entradas (canales) de información, cumpliéndose que: $2^m = n$

Su funcionamiento se caracteriza por lo siguiente: "*La combinación binaria aplicada a las* **m** *entradas de control selecciona el canal de entrada cuya información lógica* (1 *ó* 0) *aparece en el canal de salida*"

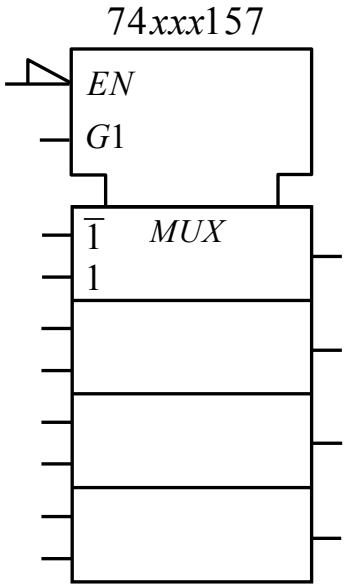


Multiplexor de 2 canales



c	f
0	a
1	b

Cuádruple multiplexor de 2 canales: los cuatro multiplexores comparten la entrada de selección y la entrada de inhibición (*En*: *enable*)

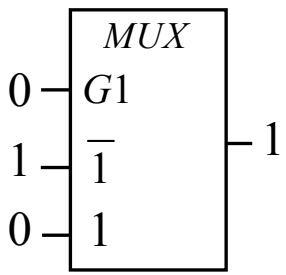


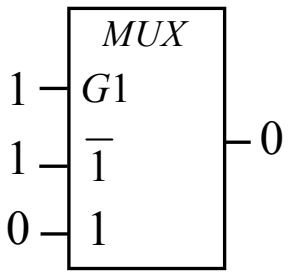
Nota: cuando una entrada *EN* no tiene un subíndice, se cumple lo siguiente:

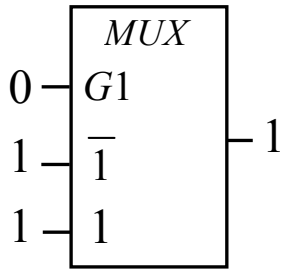
· si el estado lógico interno de dicha entrada (*EN*) es igual a 1, entonces el circuito funciona normalmente.

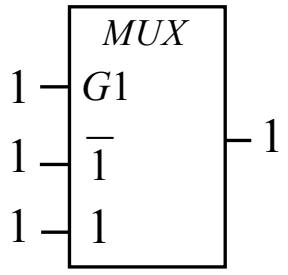
· si el estado lógico interno de dicha entrada (*EN*) es igual a 0, entonces los estados lógicos internos de todas las salidas del circuito se ponen a 0, con independencia de los valores que tomen las demás entradas.

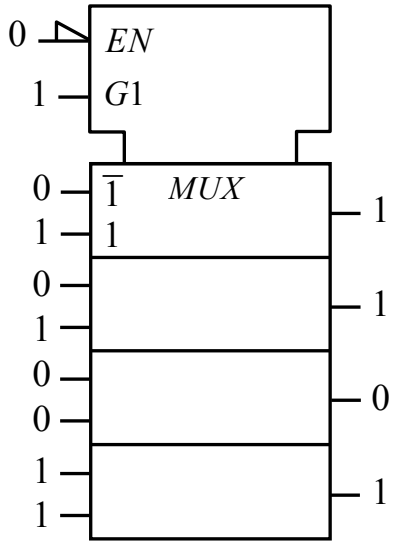
Ejemplos:

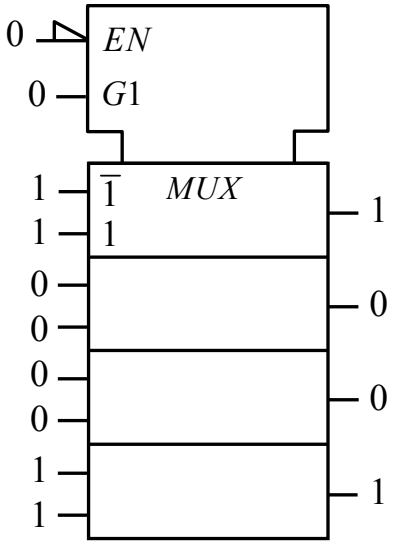

MUX
0
1
0
G1
1
1
1

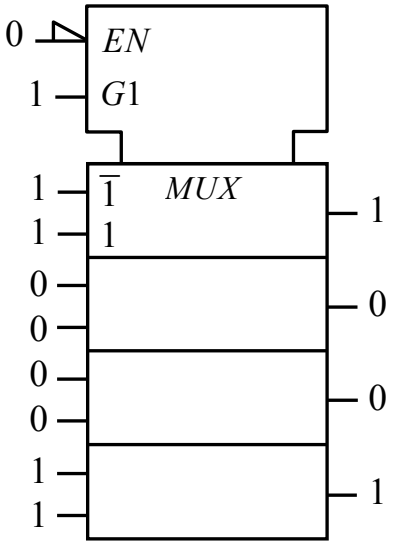

MUX
1
1
0
G1
1
1
0

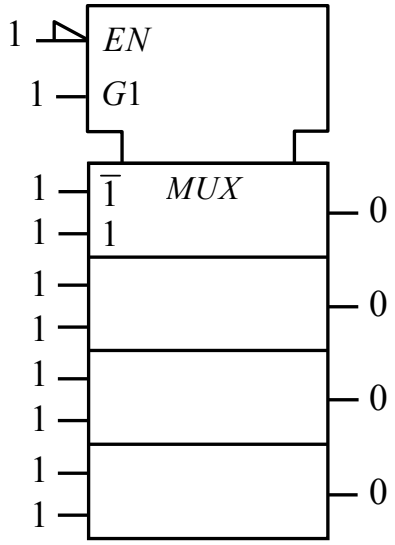

MUX
0
1
1
G1
1
1
1


MUX
1
1
1
G1
1
1
1

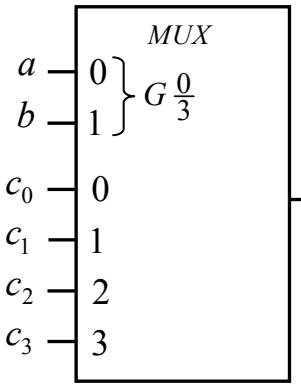

0
EN
1
G1
0
1
MUX
1
0
1
0
0
1
1
1
1
0
0
1
1


0
EN
0
G1
1
1
MUX
1
0
0
0
0
1
1
1
0
0
1


0
EN
1
G1
1
1
MUX
1
0
0
0
0
1
1
1
0
0
1


1
EN
1
G1
1
1
MUX
0
0
0
0
0
1
1
1
0
0
0

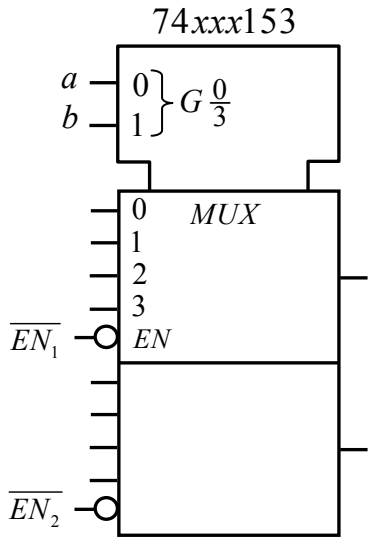
Multiplexor de 4 canales



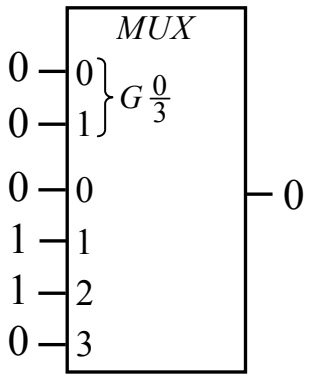
$$f(c_3, c_2, c_1, c_0, b, a) = c_0 \overline{b}\,\overline{a} + c_1 \overline{b}\,a + c_2 b\,\overline{a} + c_3 b\,a$$

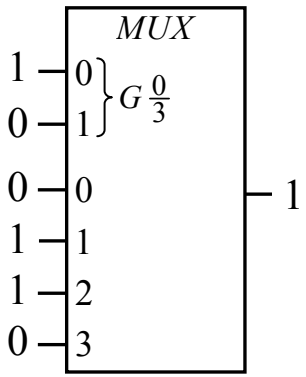
b	**a**	f
0	0	c_0
0	1	c_1
1	0	c_2
1	1	c_3

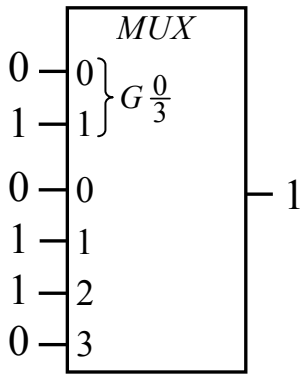
Doble multiplexor de 4 canales: los dos multiplexores comparten las entradas de selección

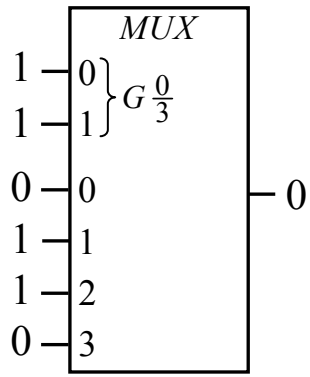


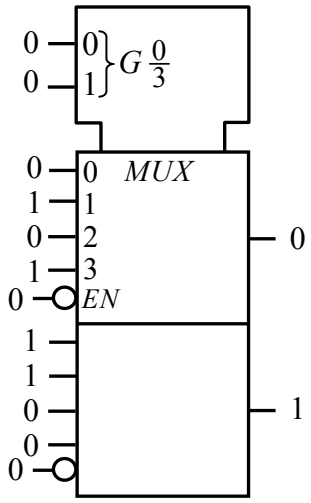
Ejemplos:

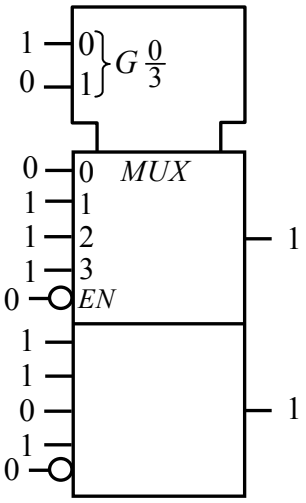

MUX
G 0/3
0
0
0
1
1
0
0
1
2
3
0

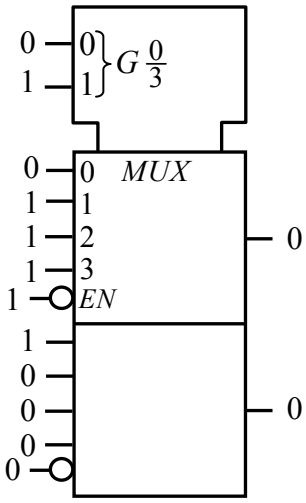

MUX
G 0/3
1
0
0
1
1
0
0
1
2
3
1

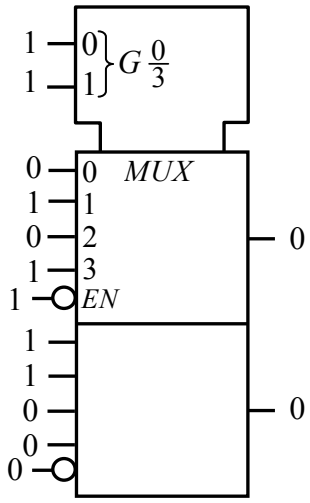

MUX
G 0/3
0
1
0
1
1
0
0
1
2
3
1


MUX
G 0/3
1
1
0
1
1
0
0
1
2
3
0


G 0/3
0
0
MUX
0
1
0
1
0
1
2
3
EN
0
1
1
0
0
0
0
1

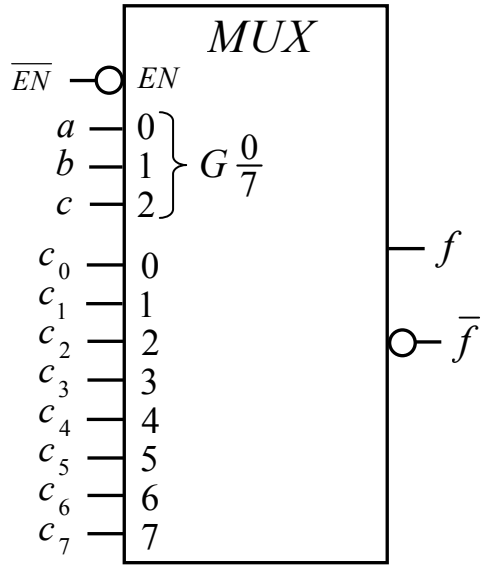

G 0/3
1
0
MUX
0
1
1
1
0
1
2
3
EN
0
1
1
0
1
0
1
1


G 0/3
0
1
MUX
0
1
1
1
1
0
2
3
EN
1
1
0
0
0
0
0
0

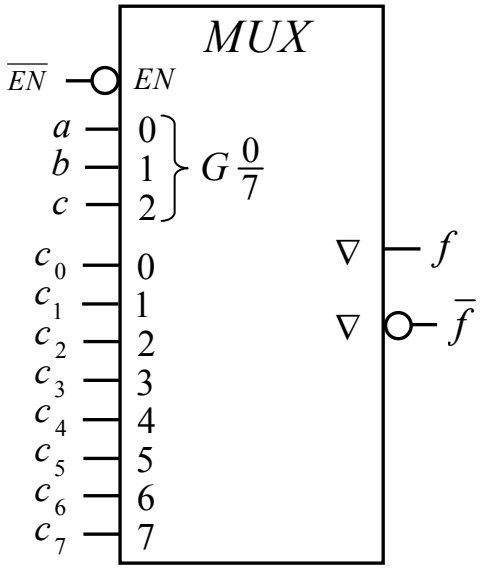

G 0/3
1
1
MUX
0
1
0
1
0
1
2
3
EN
1
1
1
0
0
0
0
0

Multiplexores de 8 canales

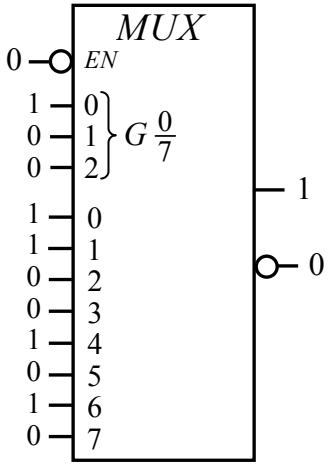
74*xxx*151

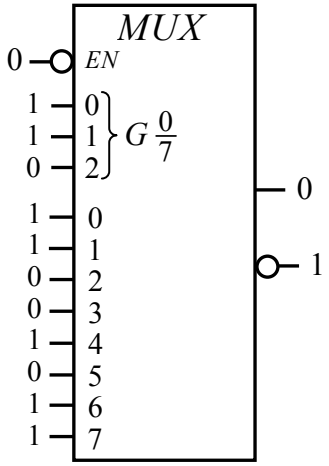


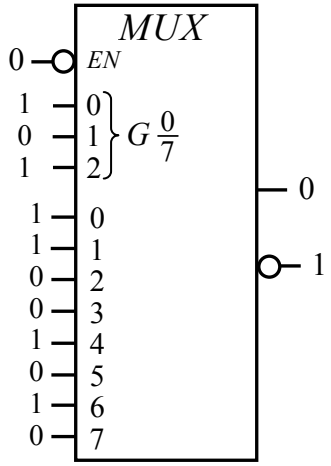
74*xxx*251

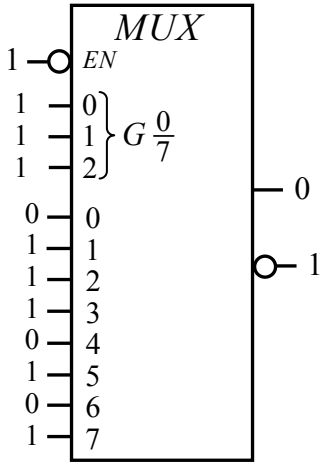


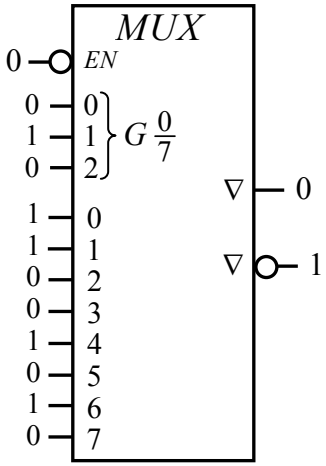
Ejemplos:

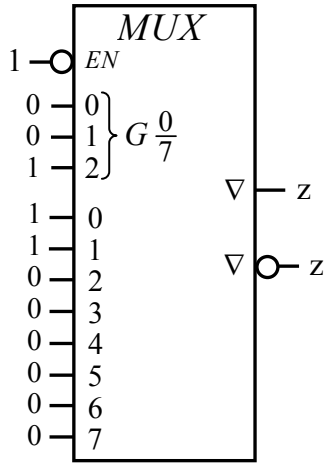

MUX
EN
G 0/7
1
0

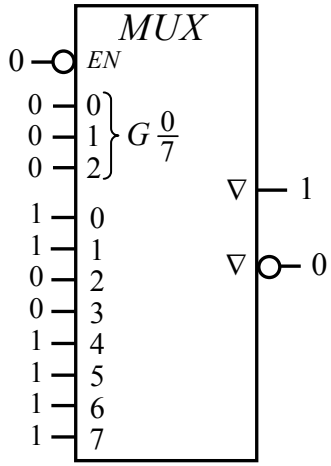

MUX
EN
G 0/7
0
1

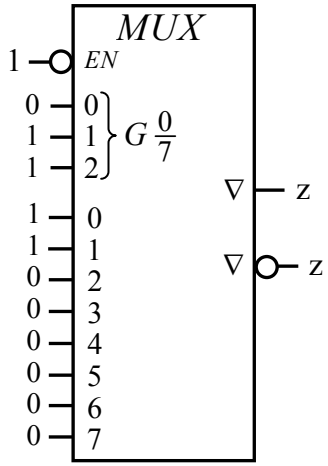

MUX
EN
G 0/7
0
1


MUX
EN
G 0/7
0
1

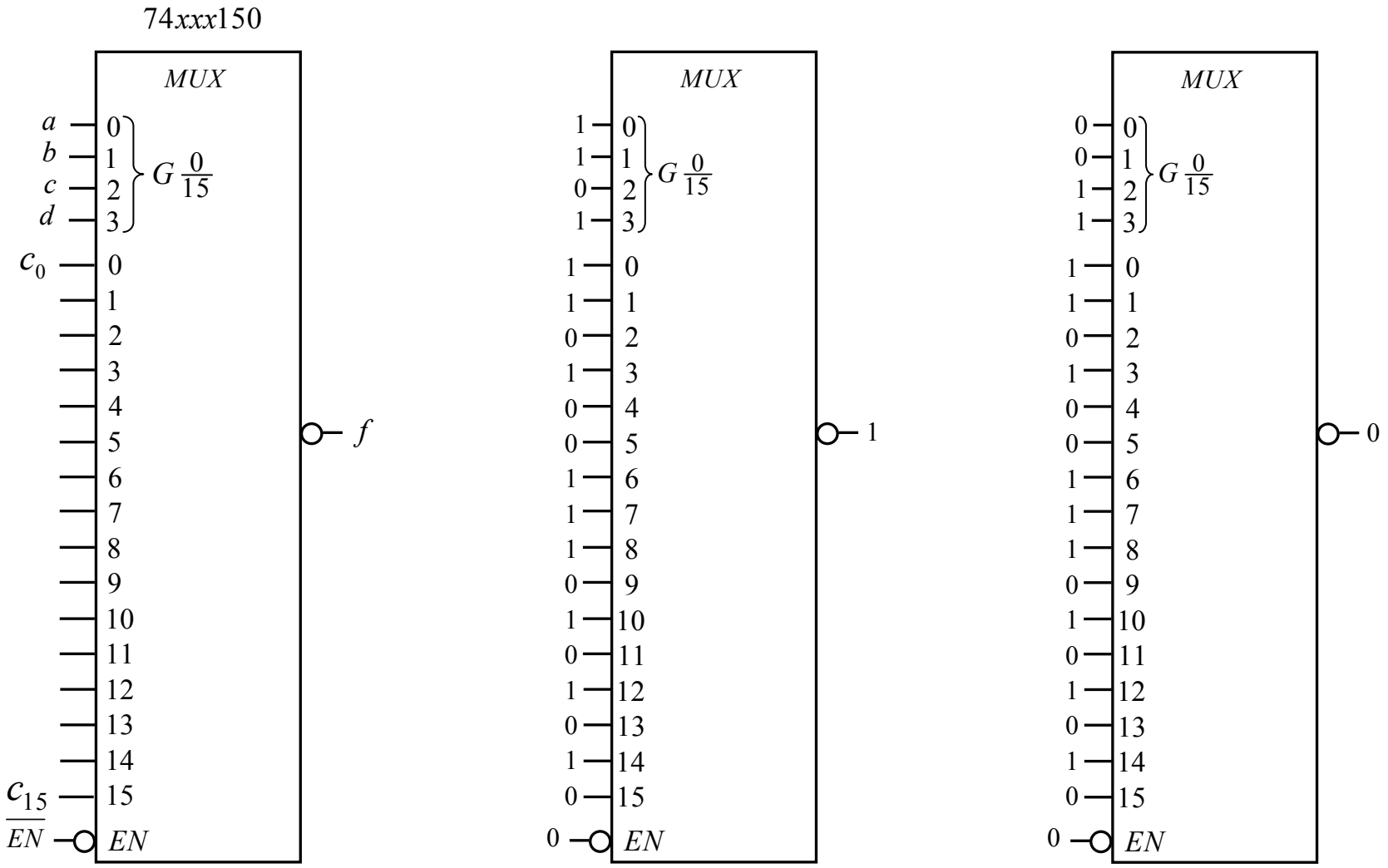

MUX
EN
G 0/7
0
1


MUX
EN
G 0/7
z
z


MUX
EN
G 0/7
1
0

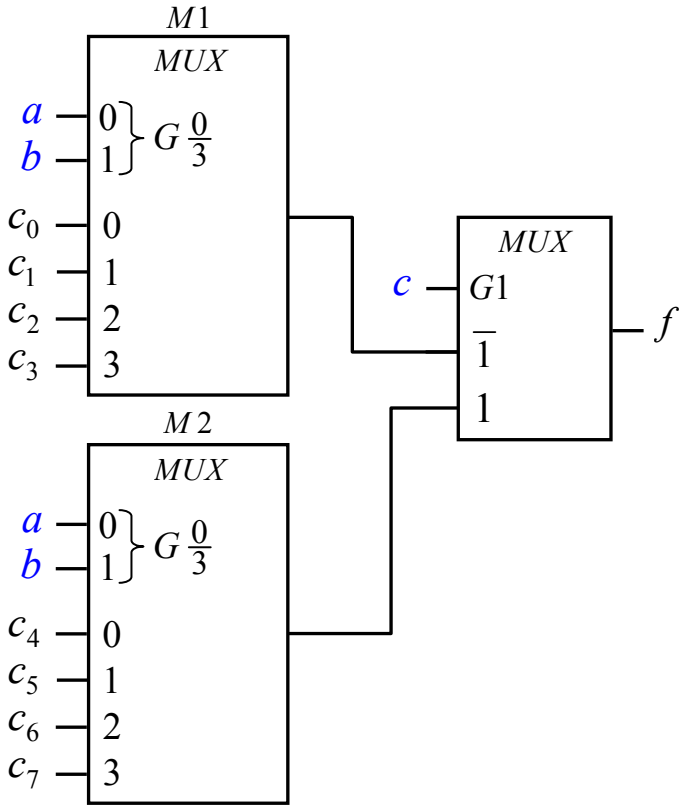

MUX
EN
G 0/7
z
z

Multiplexor de 16 canales



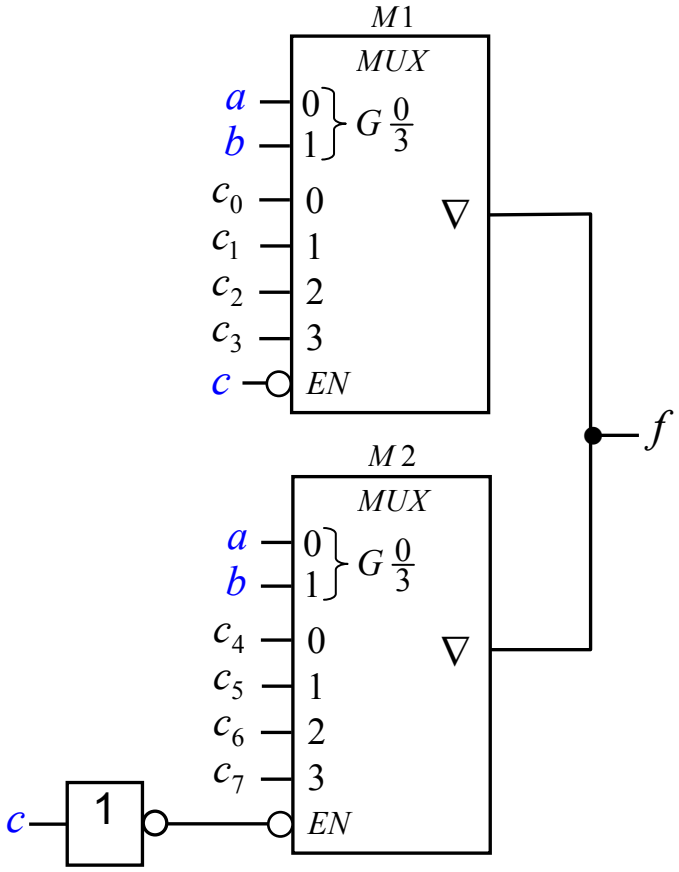
Construcción de multiplexores utilizando multiplexores más sencillos

Construcción de un multiplexor de 8 canales utilizando 2 multiplexores de 4 canales y 1 multiplexor de 2 canales



c	b	a	f
0	0	0	c_0
0	0	1	c_1
0	1	0	c_2
0	1	1	c_3
1	0	0	c_4
1	0	1	c_5
1	1	0	c_6
1	1	1	c_7

Multiplexor de 8 canales



c	*b*	*a*	*f*
0	0	0	c_0
0	0	1	c_1
0	1	0	c_2
0	1	1	c_3
1	0	0	c_4
1	0	1	c_5
1	1	0	c_6
1	1	1	c_7

Aplicación: Implementación de funciones lógicas con multiplexores

- Cualquier función dependiente de n variables, ***siempre*** se puede implementar utilizando:

opción 1: Un multiplexor de n entradas de selección (≡ un multiplexor de 2^n canales)

opción 2: Un multiplexor de $n-1$ entradas de selección y 1 puerta NOT.

- Algunas funciones dependientes de n variables se pueden implementar utilizando únicamente un multiplexor de $n-1$ variables de selección (≡ un multiplexor de 2^{n-1} canales)

Nota 1: el objetivo es implementar funciones utilizando el menor número posible circuitos integrados (ICs).

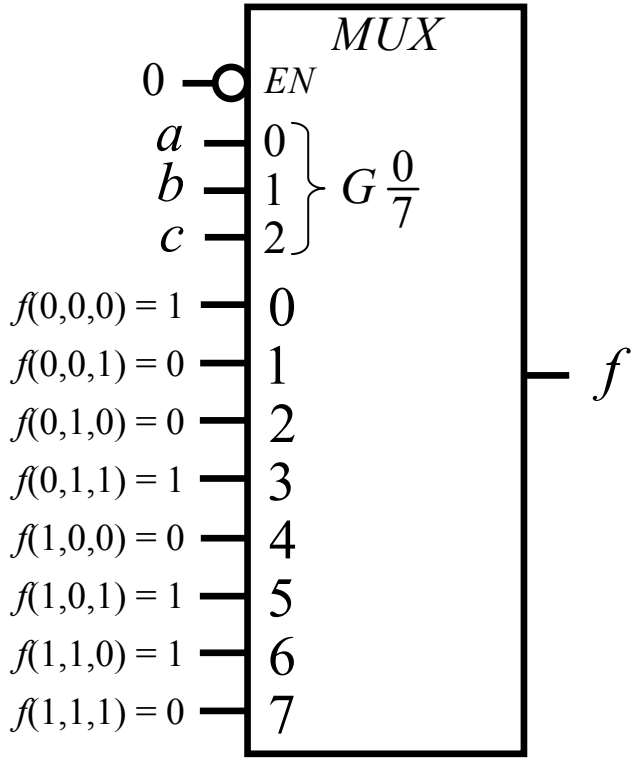
Nota 2: *Xilinx* utiliza memorias para implementar funciones en sus *FPGAs* (ver tema 6), mientras que *Actel* utiliza multiplexores (ver las siguientes diapositivas)

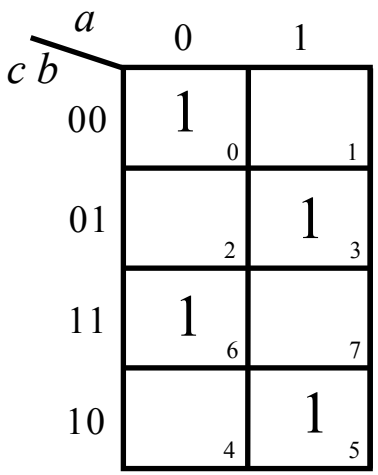
Nota 3: las *LUT* (*look-up table*) de las *logic cells* son las que implementan las funciones lógicas (circuitos combinacionales).

Nota 4: *FPGA*: *field programmable gate array*

Ejemplo 1: no se dispone de ninguna de las variables de las que depende la función tanto en su forma directa como negada.

$$f(c,b,a)=\sum_3(0,3,5,6)$$



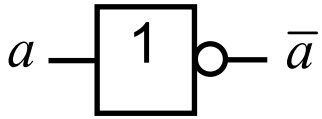


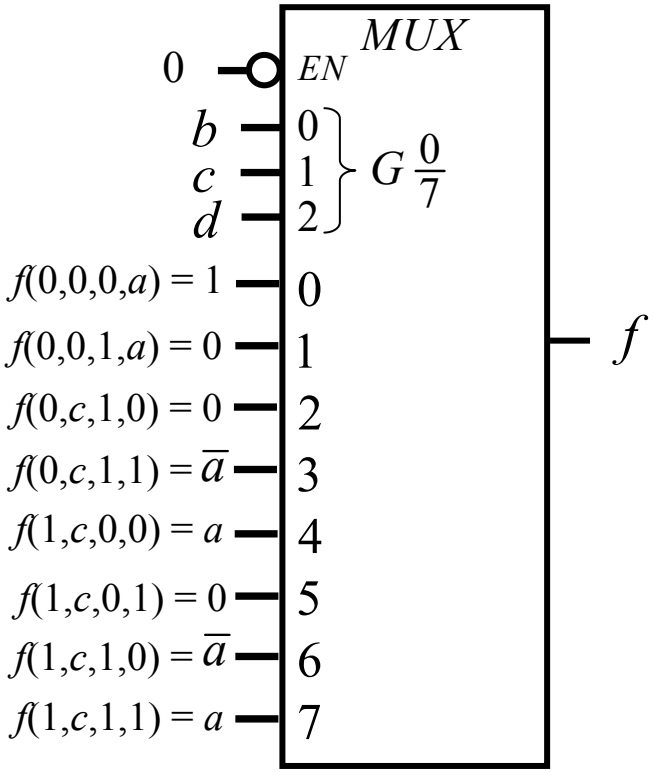
c	b	a	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Ejemplo 2: se dispone de al menos una de las variables de las que depende la función tanto en su forma directa como en su forma negada.

Nota: en este ejemplo se determinan los valores que hay que poner en los canales de entrada utilizando la *tabla de verdad* de la función.

$$f(d,c,b,a) = \sum\nolimits_4 (0,1,6,9,12,15)$$



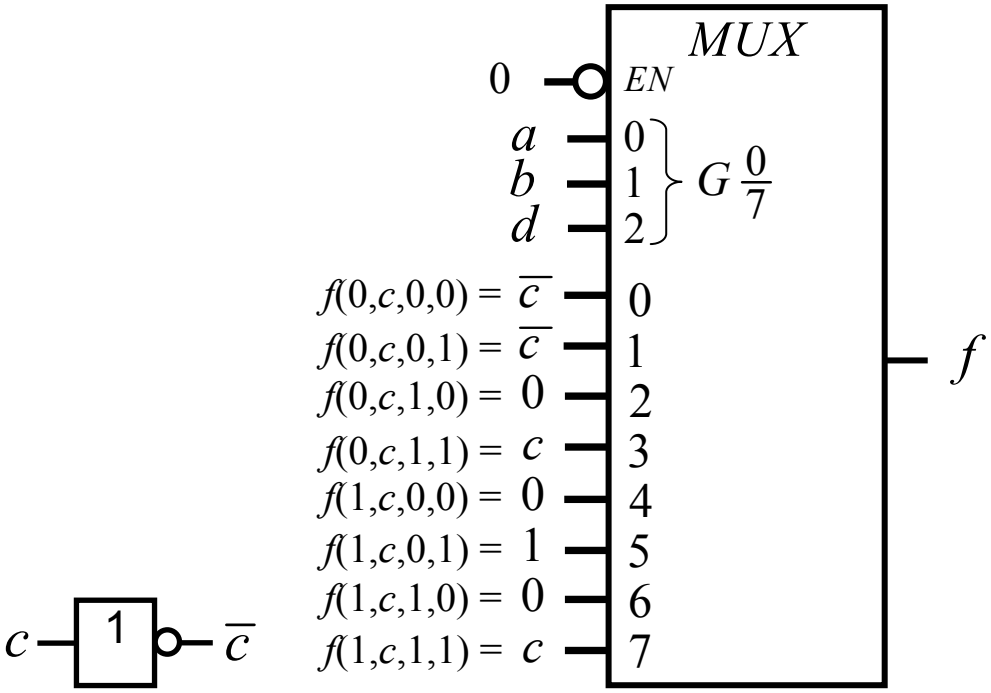


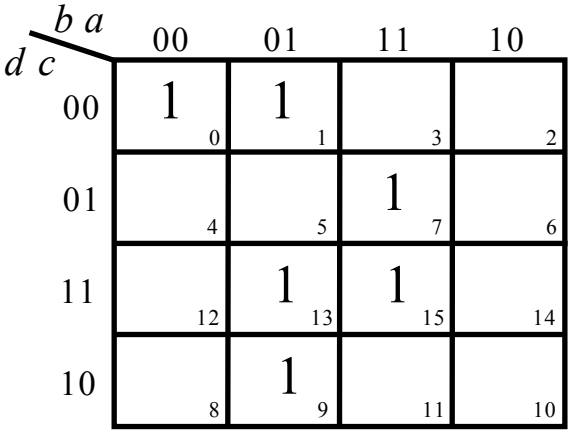
d	c	b	a	f	
0	0	0	0	1	1
0	0	0	1	1	
0	0	1	0	0	0
0	0	1	1	0	
0	1	0	0	0	0
0	1	0	1	0	
0	1	1	0	1	$\overline{a}$
0	1	1	1	0	
1	0	0	0	0	a
1	0	0	1	1	
1	0	1	0	0	0
1	0	1	1	0	
1	1	0	0	1	$\overline{a}$
1	1	0	1	0	
1	1	1	0	0	a
1	1	1	1	1	

Ejemplo 3: en este caso, los valores que hay que poner en los canales de entrada del multiplexor se determinan utilizando una tabla de *Karnaugh-Veitch*.

Solución 1:

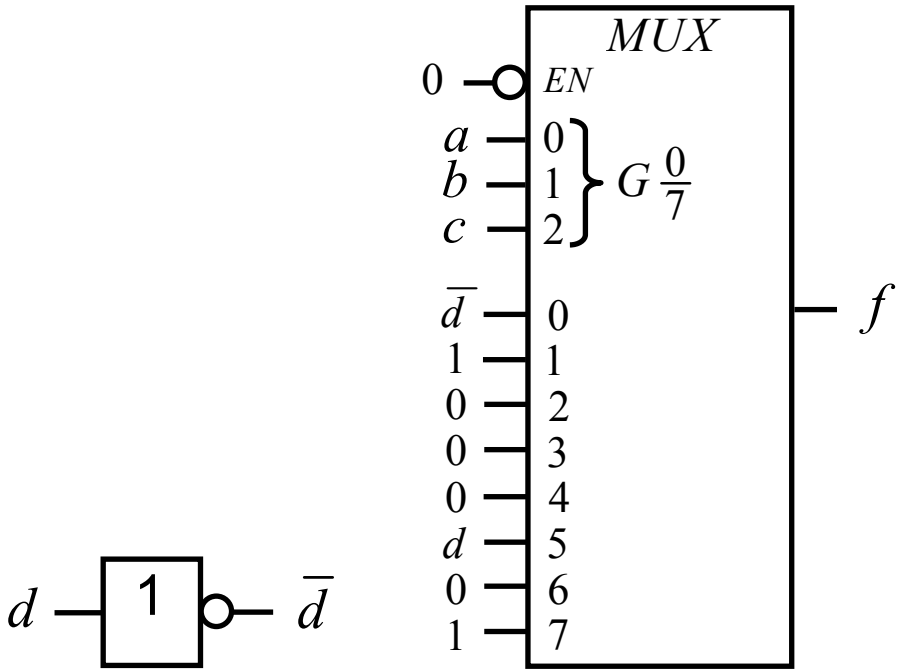
$$f(d,c,b,a) = \sum_4 (0,1,7,9,13,15)$$

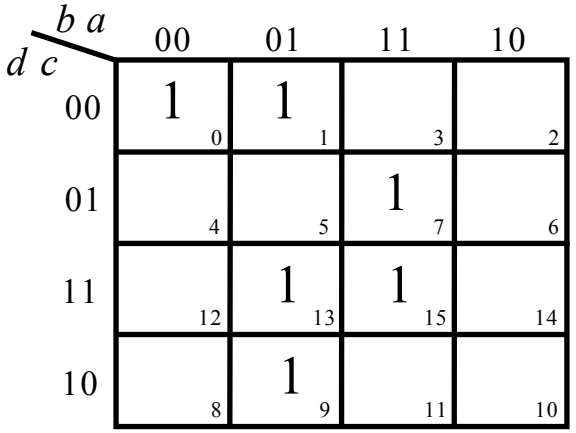




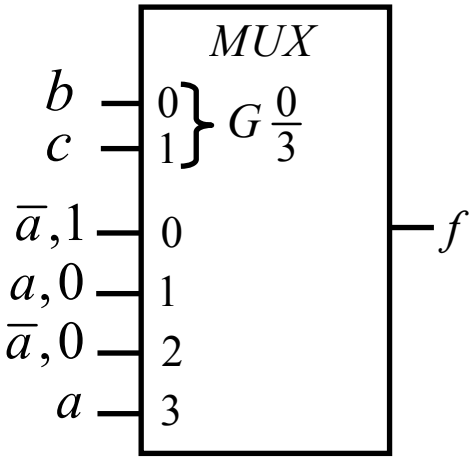
Solución 2: en las entradas de selección del multiplexor se pone un conjunto de variables distinto al de la solución anterior.

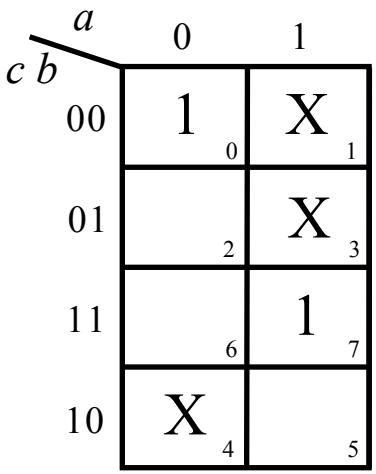
$$f(d,c,b,a) = \sum_4 (0,1,7,9,13,15)$$





Ejemplo 3: la función no está totalmente definida: $f(c,b,a) = \sum_3(0,7) + \sum_\phi(1,3,4)$





c	b	a	f	
0	0	0	1	$\overline{a}$,1
0	0	1	x	
0	1	0	0	0, a
0	1	1	x	
1	0	0	x	0, $\overline{a}$
1	0	1	0	
1	1	0	0	a
1	1	1	1	

Pregunta: En la práctica, ¿qué *valores-señales* se ponen en los canales de entrada?

- Demultiplexores de *n* canales (de salida)

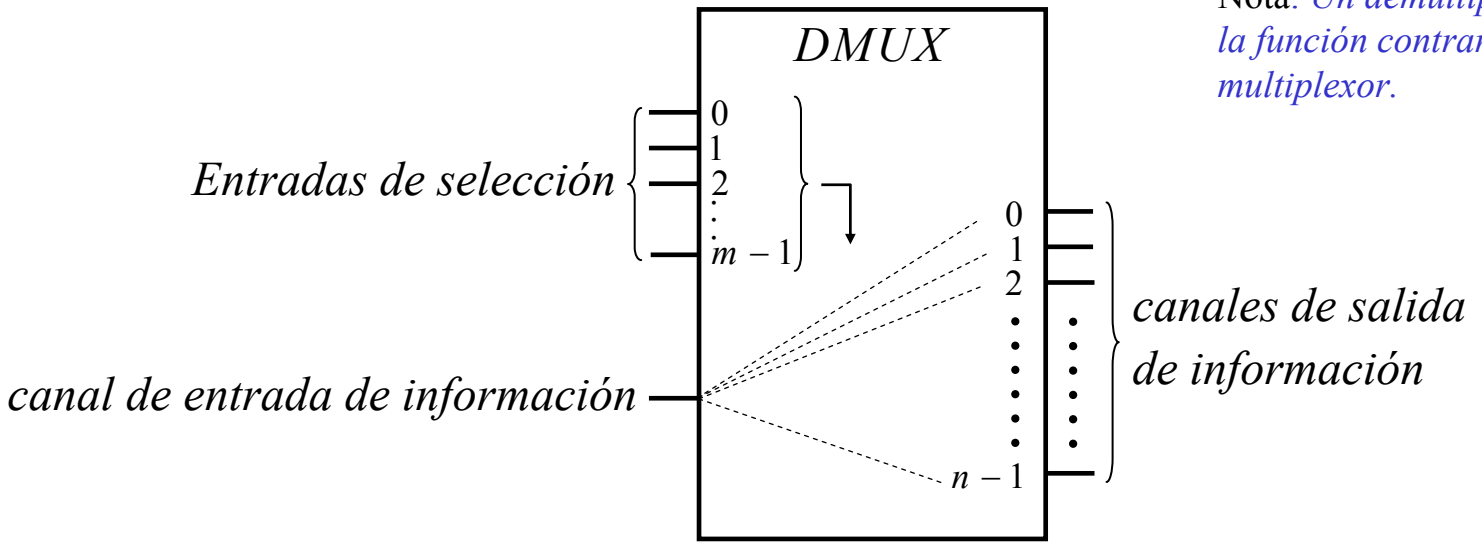
DEF.: Un demultiplexor es un circuito combinacional que tiene:

_ 1 entrada (canal) de *información*.

_ *m* entradas de *control* o de *selección*.

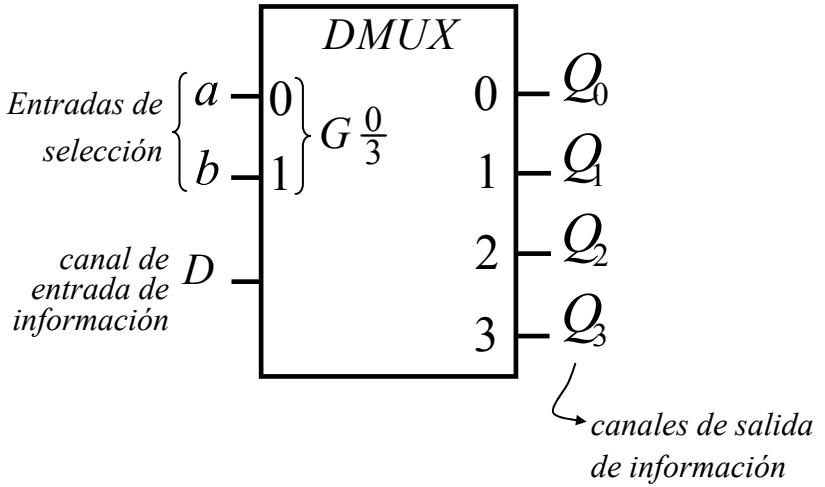
_ *n* salidas (canales) de información, cumpliéndose que: $2^m = n$

Su funcionamiento se caracteriza por lo siguiente: " *La combinación binaria aplicada a las m entradas de control selecciona el canal de salida en el que aparece el valor* (1 *ó* 0) *presente en el canal de entrada*"



Nota: *Un demultiplexor realiza la función contraria a la de un multiplexor.*

Demultiplexor de 4 canales (de salida)

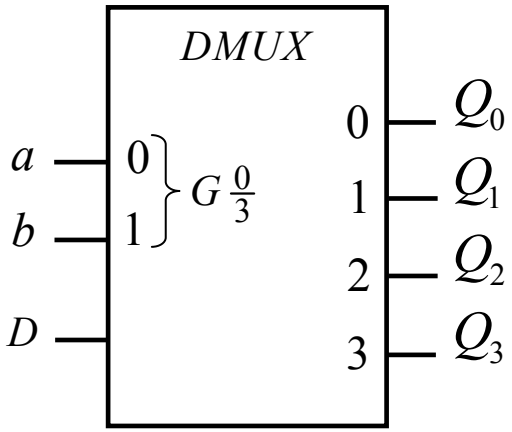


entrada de información — *entradas de selección*

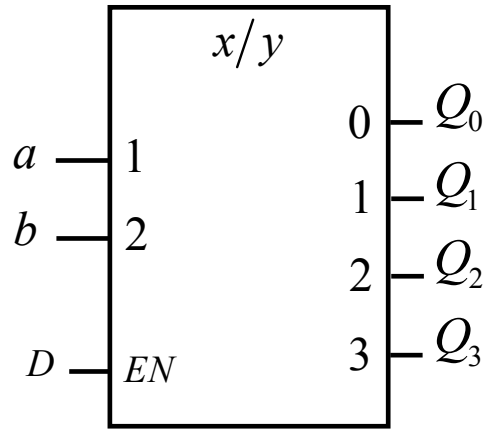
D	b	a	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0	0	0
1	0	0	0	0	0	1
0	0	1	0	0	0	0
1	0	1	0	0	1	0
0	1	0	0	0	0	0
1	1	0	0	1	0	0
0	1	1	0	0	0	0
1	1	1	1	0	0	0

no

Se fabrican circuitos que pueden funcionar como *demultiplexores* y como *decodificadores*



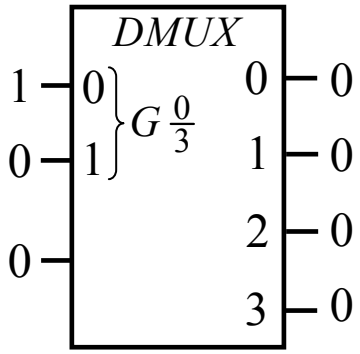
D	b	a	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0	0	0
1	0	0	0	0	0	1
0	0	1	0	0	0	0
1	0	1	0	0	1	0
0	1	0	0	0	0	0
1	1	0	0	1	0	0
0	1	1	0	0	0	0
1	1	1	1	0	0	0

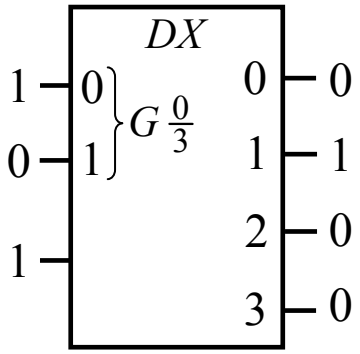


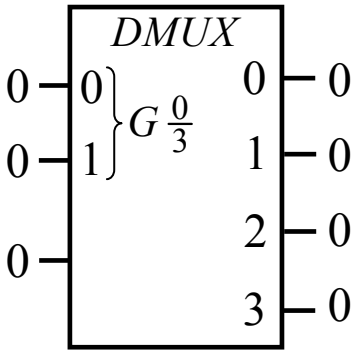
(*decodificador*)

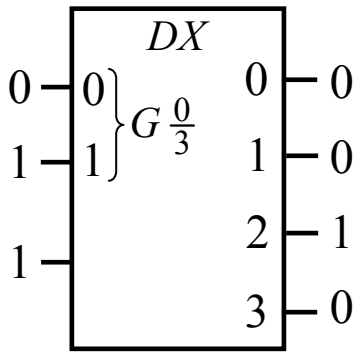
D	b	a	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

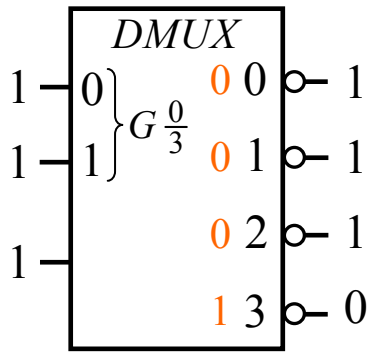
Ejemplos:

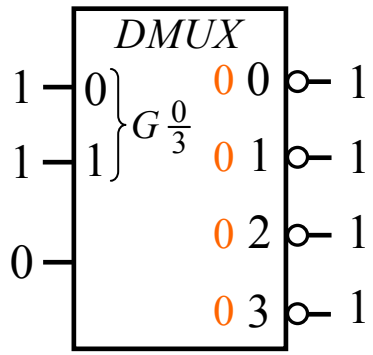

DMUX
1
0
0
0
1
G 0/3
0
1
2
3
0
0
0
0


DX
1
0
1
0
1
G 0/3
0
1
2
3
0
1
0
0

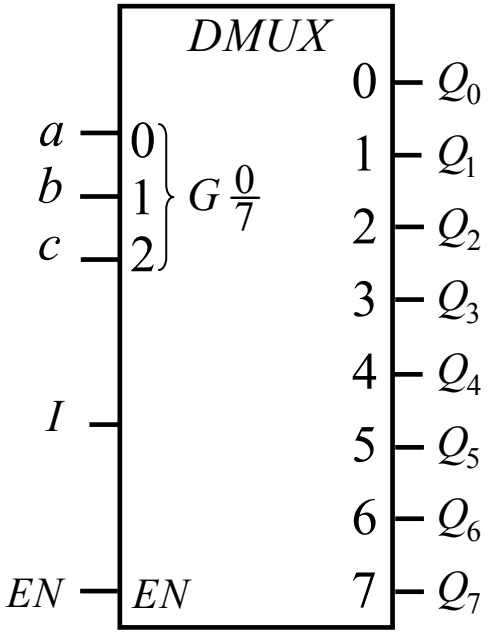

DMUX
0
0
0
0
1
G 0/3
0
1
2
3
0
0
0
0

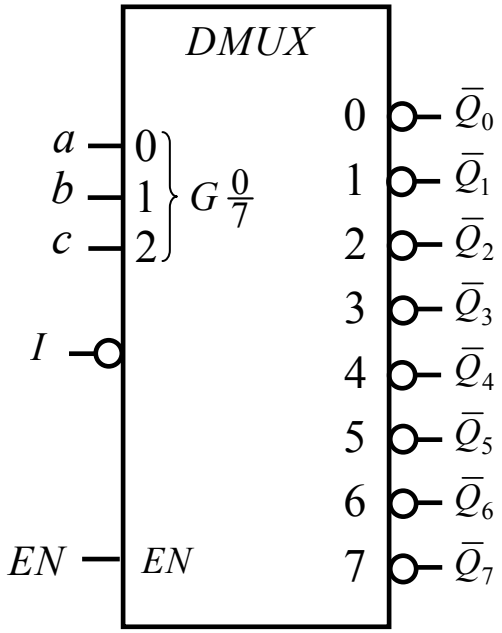

DX
0
1
1
0
1
G 0/3
0
1
2
3
0
0
1
0


DMUX
1
1
1
0
1
G 0/3
0 0
0 1
0 2
1 3
1
1
1
0

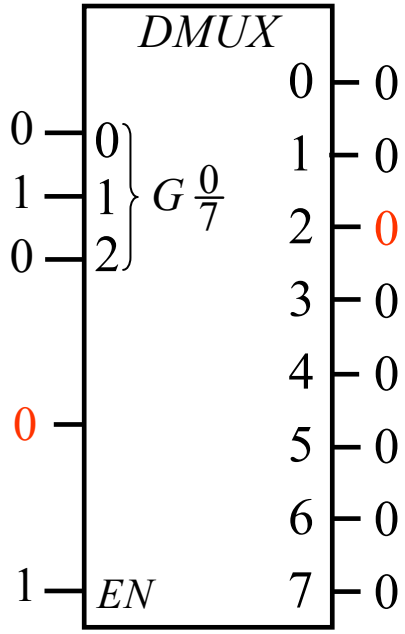

DMUX
1
1
0
0
1
G 0/3
0 0
0 1
0 2
0 3
1
1
1
1

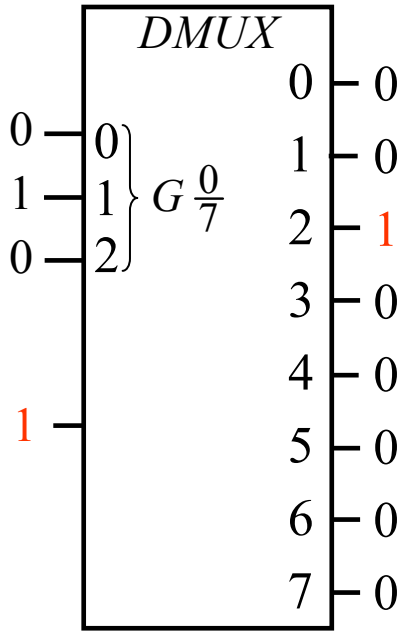
Demultiplexor de 8 canales (de salida)

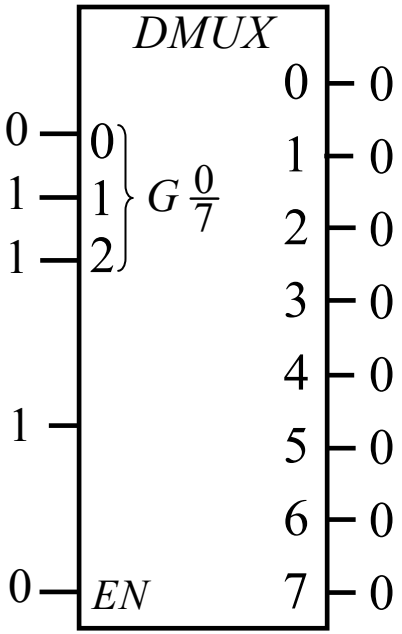


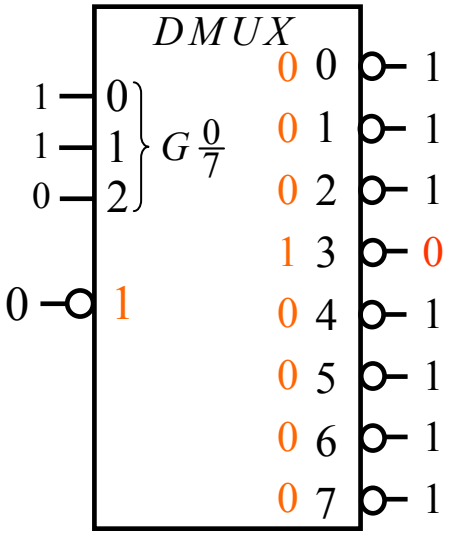


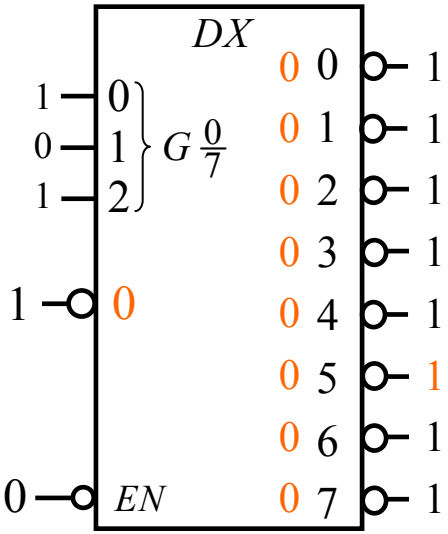
Ejemplos:

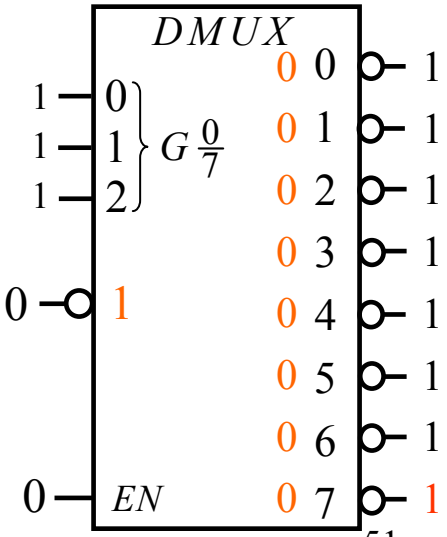

DMUX
0
1
0
0
1
2
G 0/7
0
1
2
3
4
5
6
7
0
0
0
0
0
0
0
0
1
EN


DMUX
0
1
0
0
1
2
G 0/7
0
1
2
3
4
5
6
7
0
0
1
0
0
0
0
0
1
EN


DMUX
0
1
1
0
1
2
G 0/7
0
1
2
3
4
5
6
7
0
0
0
0
0
0
0
0
1
EN

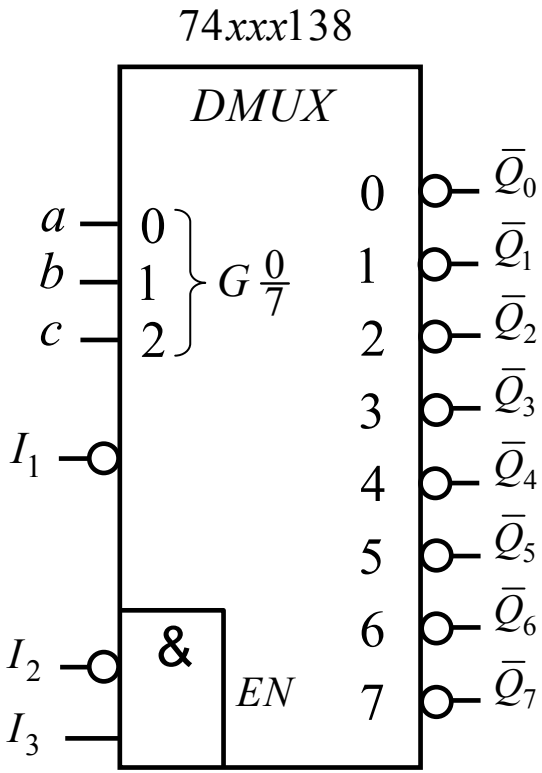

DMUX
1
1
0
0
1
2
G 0/7
0 0
0 1
0 2
1 3
0 4
0 5
0 6
0 7
1
1
1
0
1
1
1
1
0
1
EN

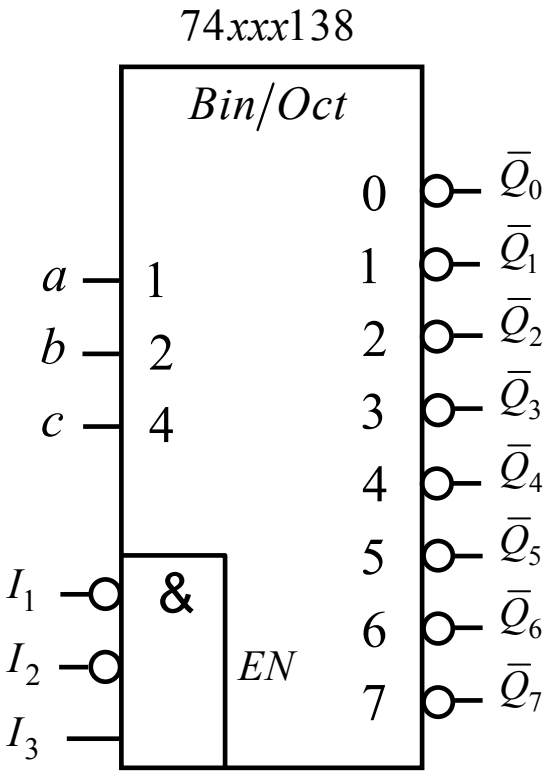

DX
1
0
1
0
1
2
G 0/7
0 0
0 1
0 2
0 3
0 4
0 5
0 6
0 7
1
1
1
1
1
1
1
1
0
1
EN


DMUX
1
1
1
0
1
2
G 0/7
0 0
0 1
0 2
0 3
0 4
0 5
0 6
0 7
1
1
1
1
1
1
1
1
0
1
EN

no

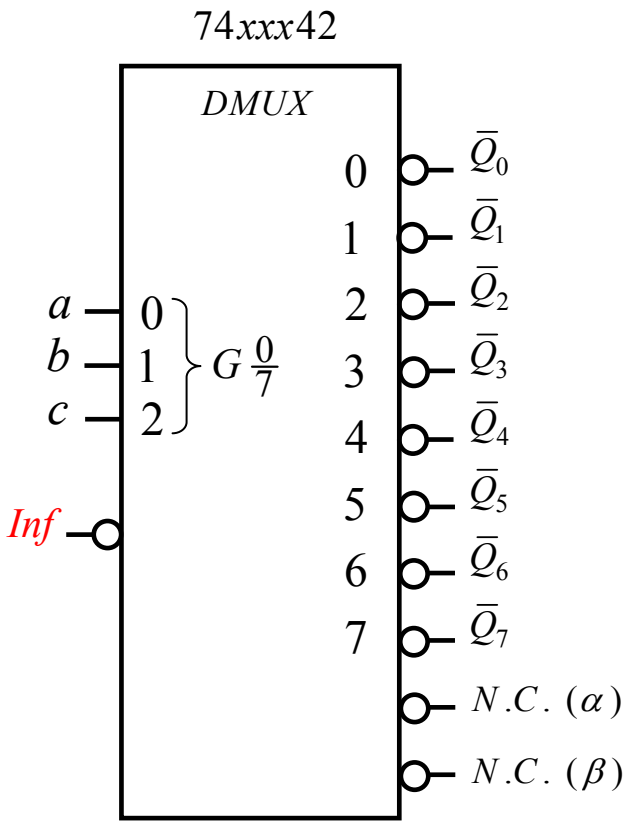
Demultiplexor de 8 canales (comercial)

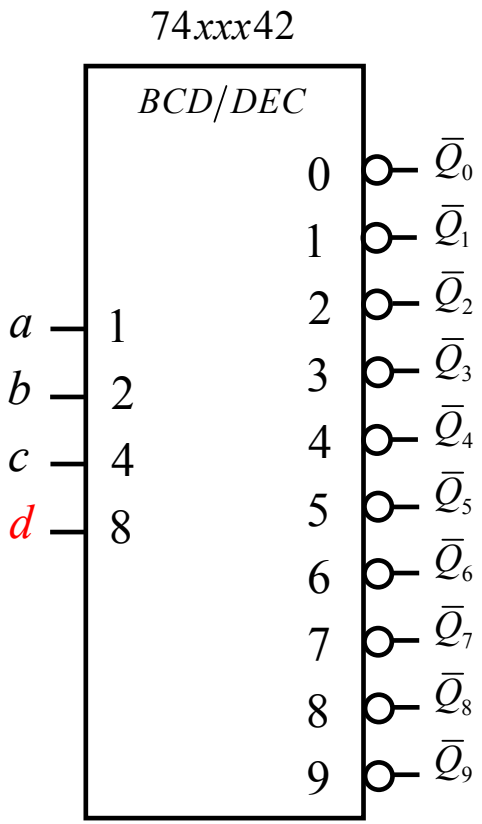




no

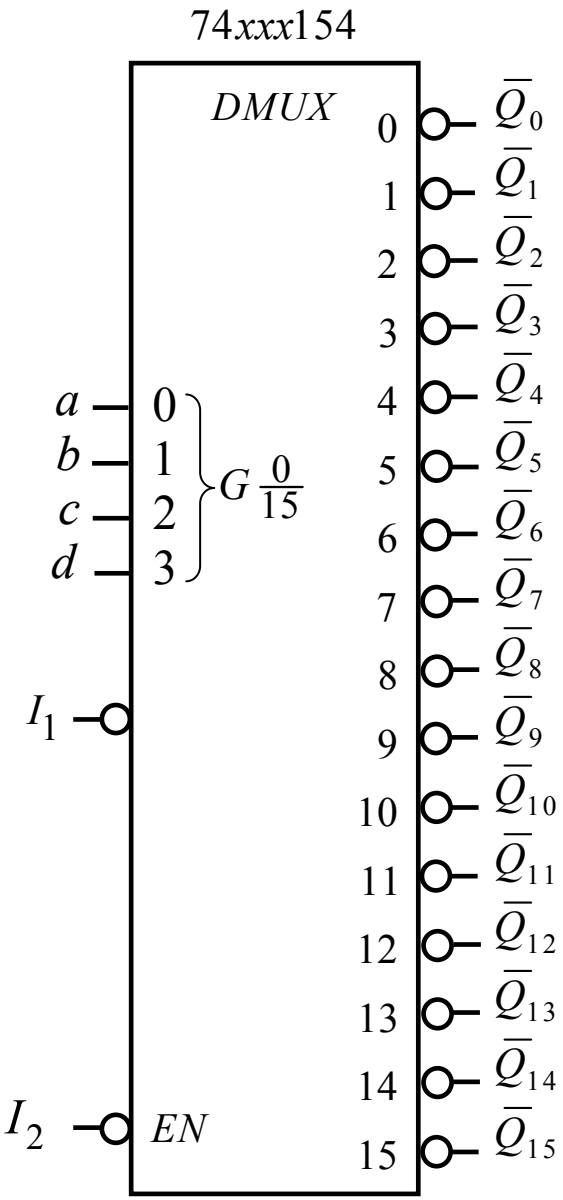
Demultiplexor de 8 canales (comercial)

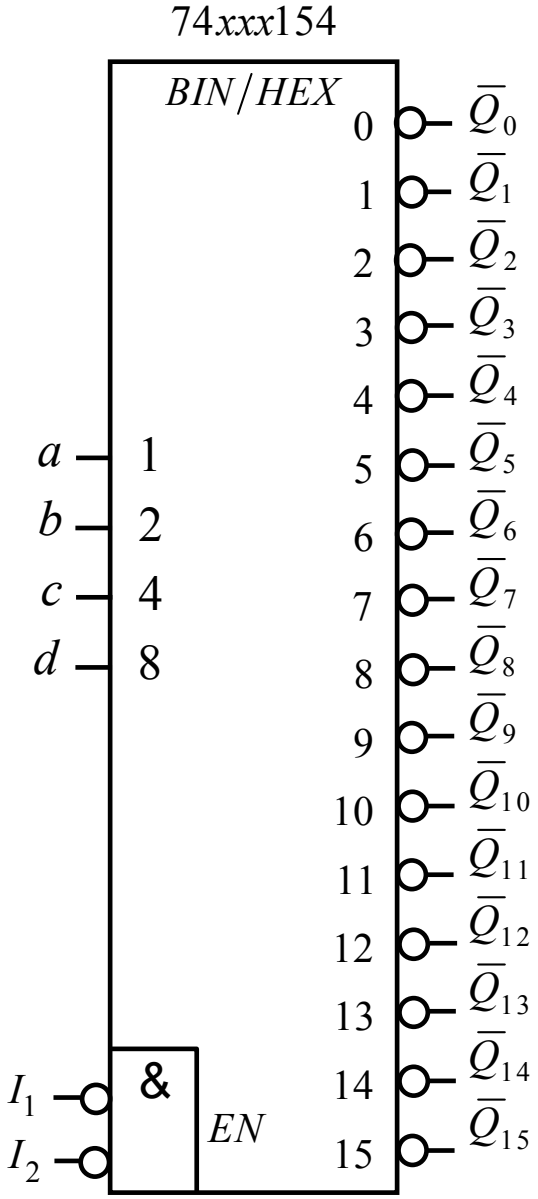




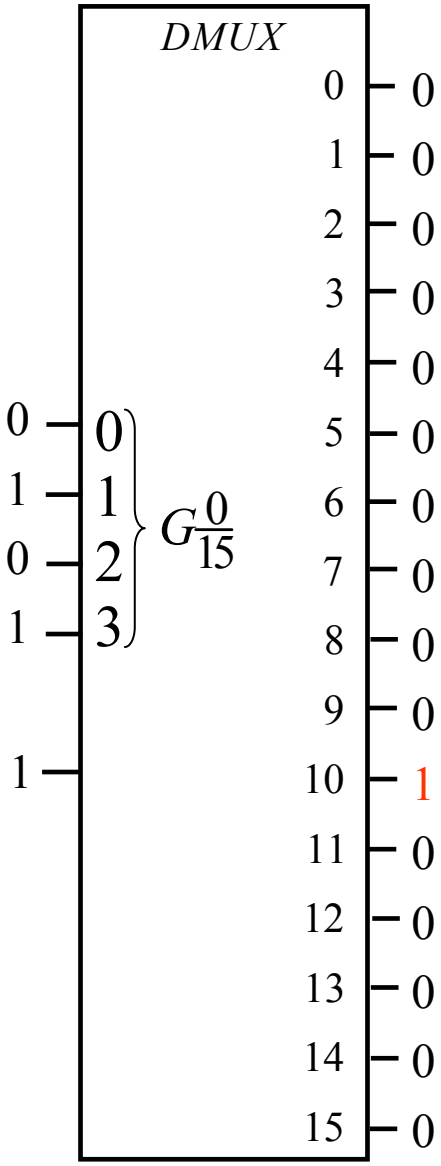
Demultiplexor de 16 canales (de salida)

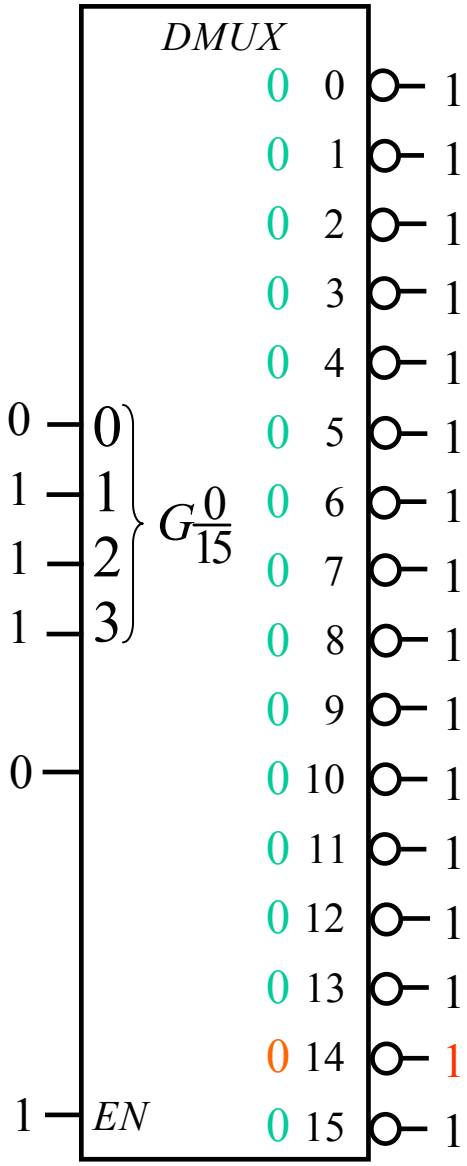
no

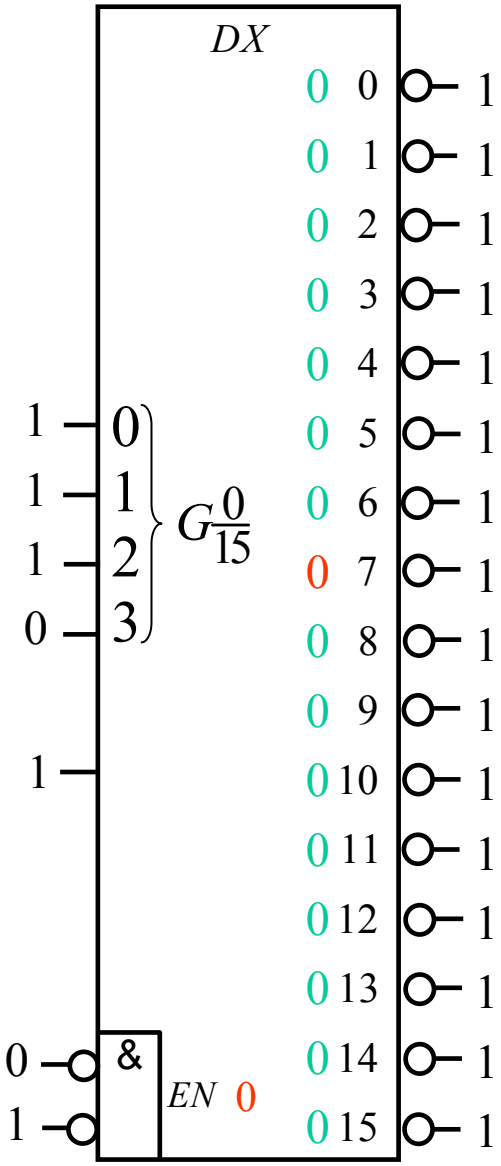




Ejemplos:

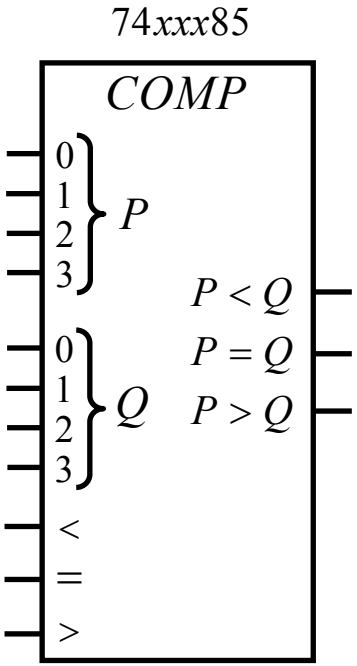

DMUX
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
0
1
0
1
1
G^{0}_{15}


DMUX
0 0
0 1
0 2
0 3
0 4
0 5
0 6
0 7
0 8
0 9
0 10
0 11
0 12
0 13
0 14
0 15
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
0
1
1
1
0
1
G^{0}_{15}
EN


DX
0 0
0 1
0 2
0 3
0 4
0 5
0 6
0 7
0 8
0 9
0 10
0 11
0 12
0 13
0 14
0 15
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
0
1
G^{0}_{15}
&
EN 0

▪ Comparadores de magnitud de *n* bits

DEF.: Un comparador de magnitud es un circuito combinacional que indica si dos magnitudes de *n* bits, representadas en binario natural, son iguales o, en el caso de que no sean iguales, cual de ellas es la mayor.

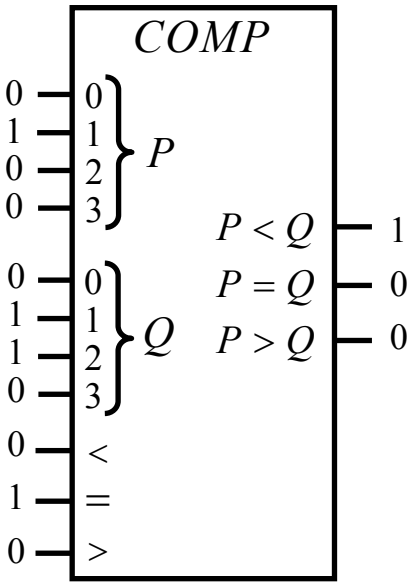


P Q	$<$	$=$	$>$	$P<Q$	$P=Q$	$P>Q$
$P<Q$	X	X	X	1	0	0
$P>Q$	X	X	X	0	0	1
$P=Q$	1	0	0	1	0	0
$P=Q$	0	1	0	0	1	0
$P=Q$	0	0	1	0	0	1

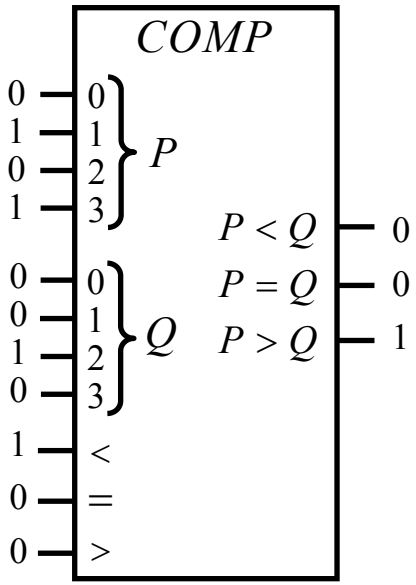
Nota: se comparan magnitudes sin signo (números binarios)

Ejemplos:

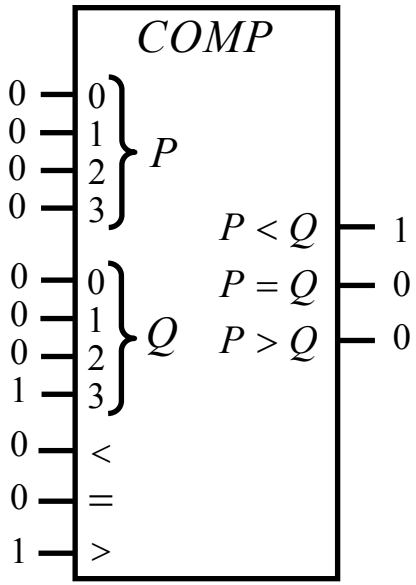
74*xxx*85



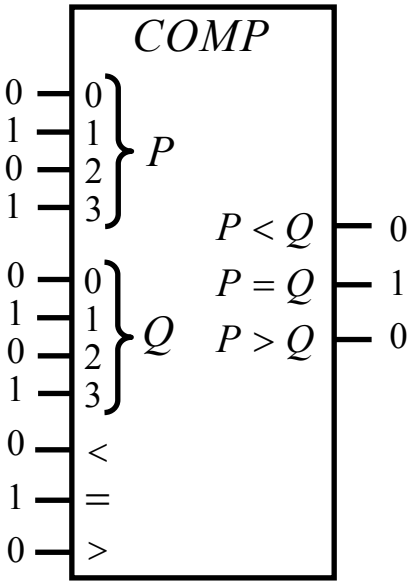
74*xxx*85



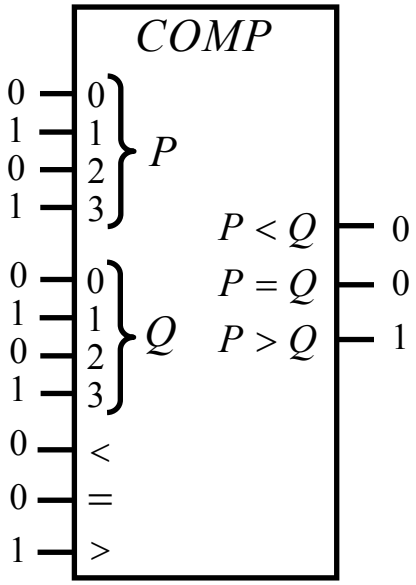
74*xxx*85



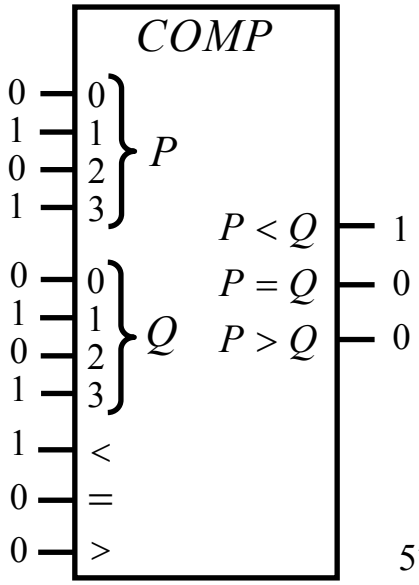
74*xxx*85



74*xxx*85



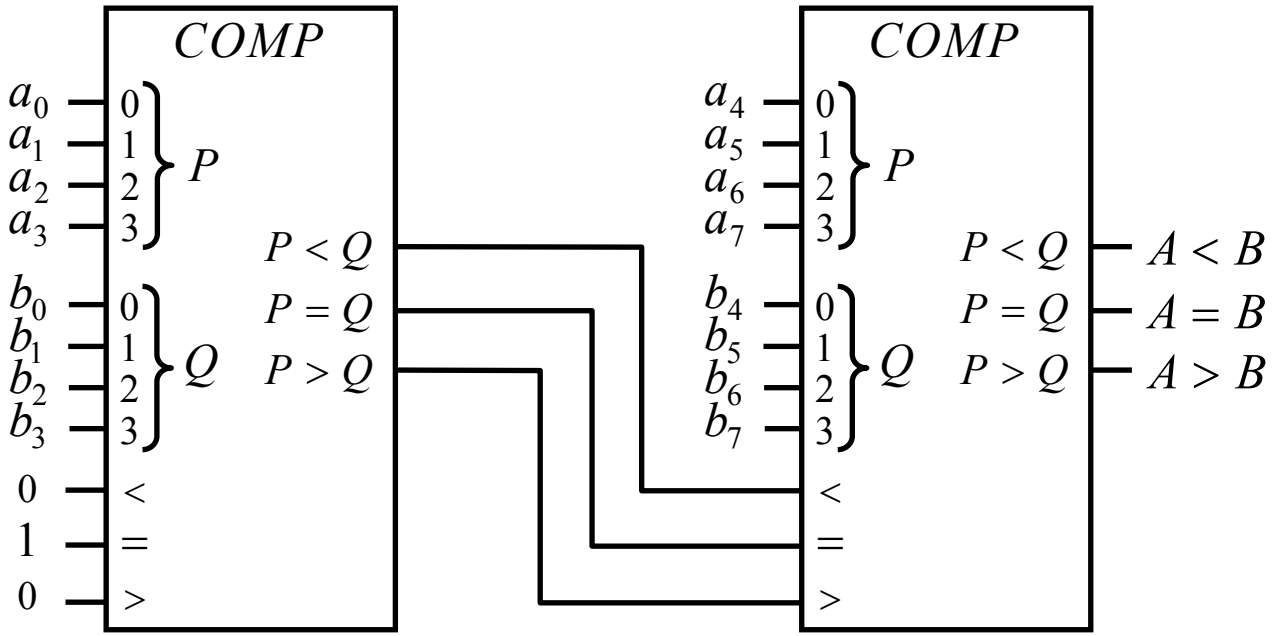
74*xxx*85



no

Diseño de un comparador de magnitud de 8 bits utilizando dos comparadores de magnitud de 4 bits.

Los números a comparar son: $A(a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ a_0)$ y $B(b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0)$



Nota: para determinar cuál de los dos números es el mayor se comparan los *nibbles* altos. En el caso de que sean iguales se comparan los *nibbles* bajos.

- Detectores / Generadores de paridad

DEF.: Un detector de paridad es un circuito combinacional que detecta/indica si en un grupo de *n* bits el número de bits que están a 1 es un número par (→ detector de paridad par) o impar (→ detector de paridad impar).

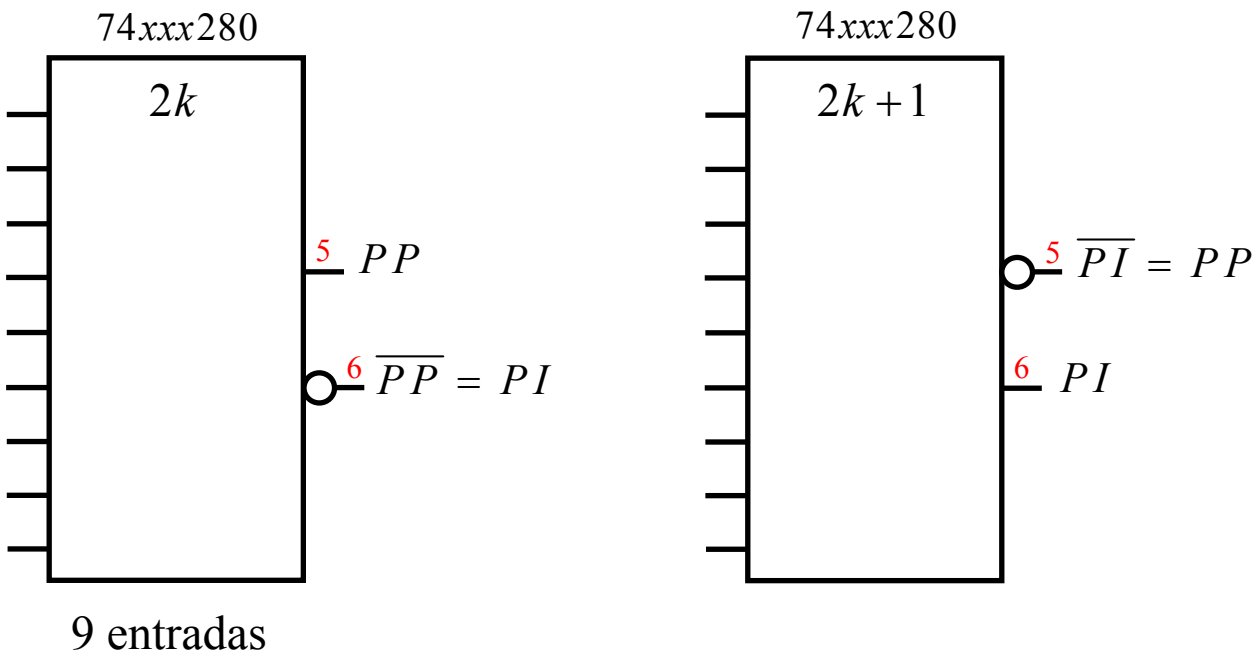
Se cumple que:

$$PP(a_8\,a_7\,a_6\,a_5\,a_4\,a_3\,a_2\,a_1\,a_0) = \overline{a_8 \oplus a_7 \oplus a_6 \oplus a_5 \oplus a_4 \oplus a_3 \oplus a_2 \oplus a_1 \oplus a_0}$$

$$PI(a_8\,a_7\,a_6\,a_5\,a_4\,a_3\,a_2\,a_1\,a_0) = \overline{PP} = a_8 \oplus a_7 \oplus a_6 \oplus a_5 \oplus a_4 \oplus a_3 \oplus a_2 \oplus a_1 \oplus a_0$$

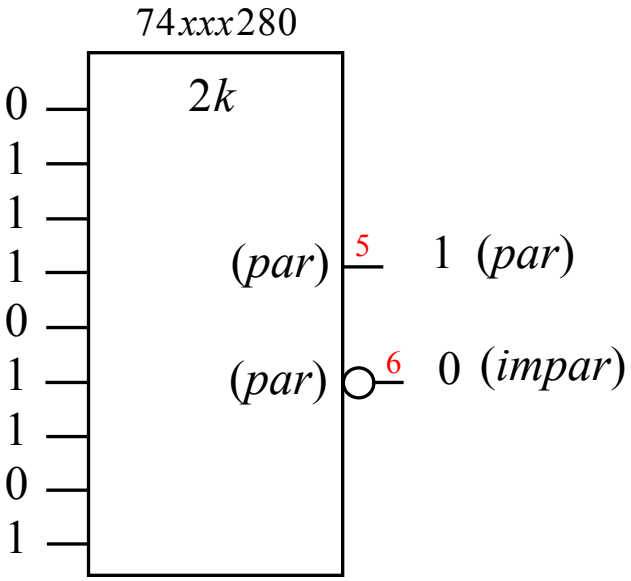
Nota:

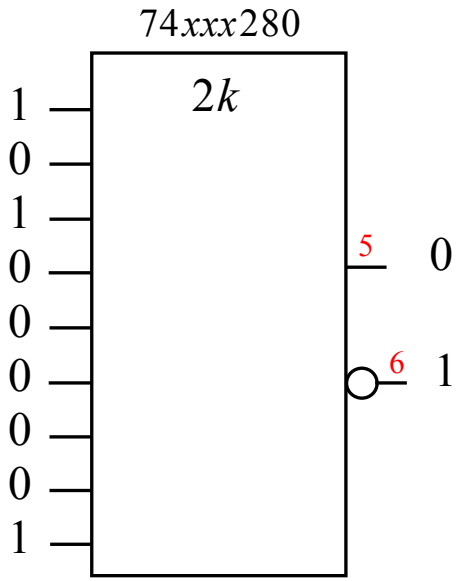
$$x_1 \oplus x_2 \oplus \cdots \oplus x_n = \begin{cases} 1 & \textit{si el número de variables que están a } 1 \textit{ es un número impar.} \\ 0 & \textit{en caso contrario.} \end{cases}$$

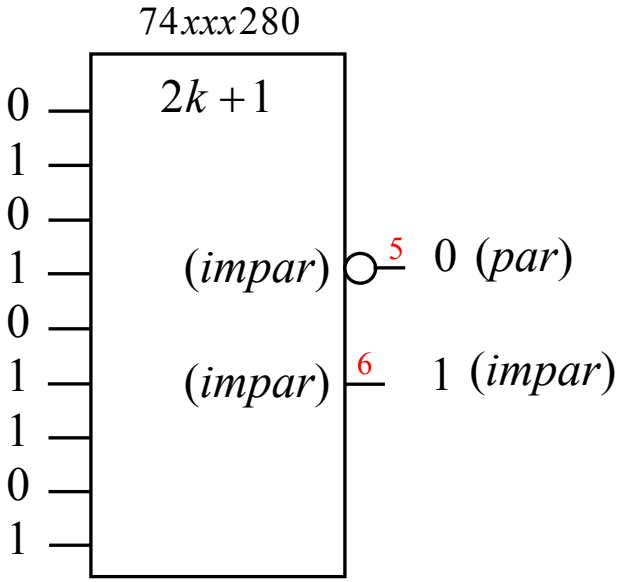


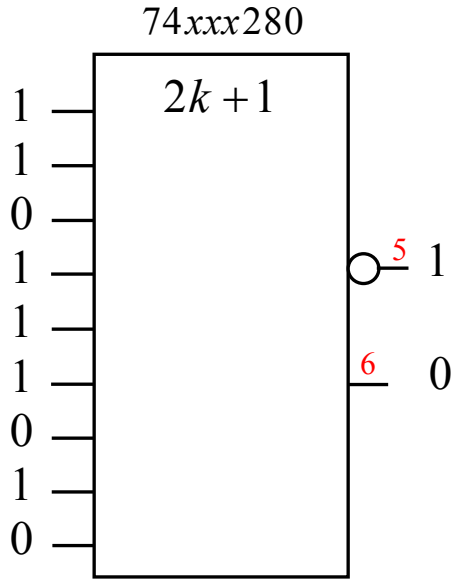
Nota: *los estados lógicos internos de las salidas siempre valen lo mismo, sin embargo los estados lógicos externos de las salidas son siempre distintos (uno es el negado del otro)*

Ejemplos:

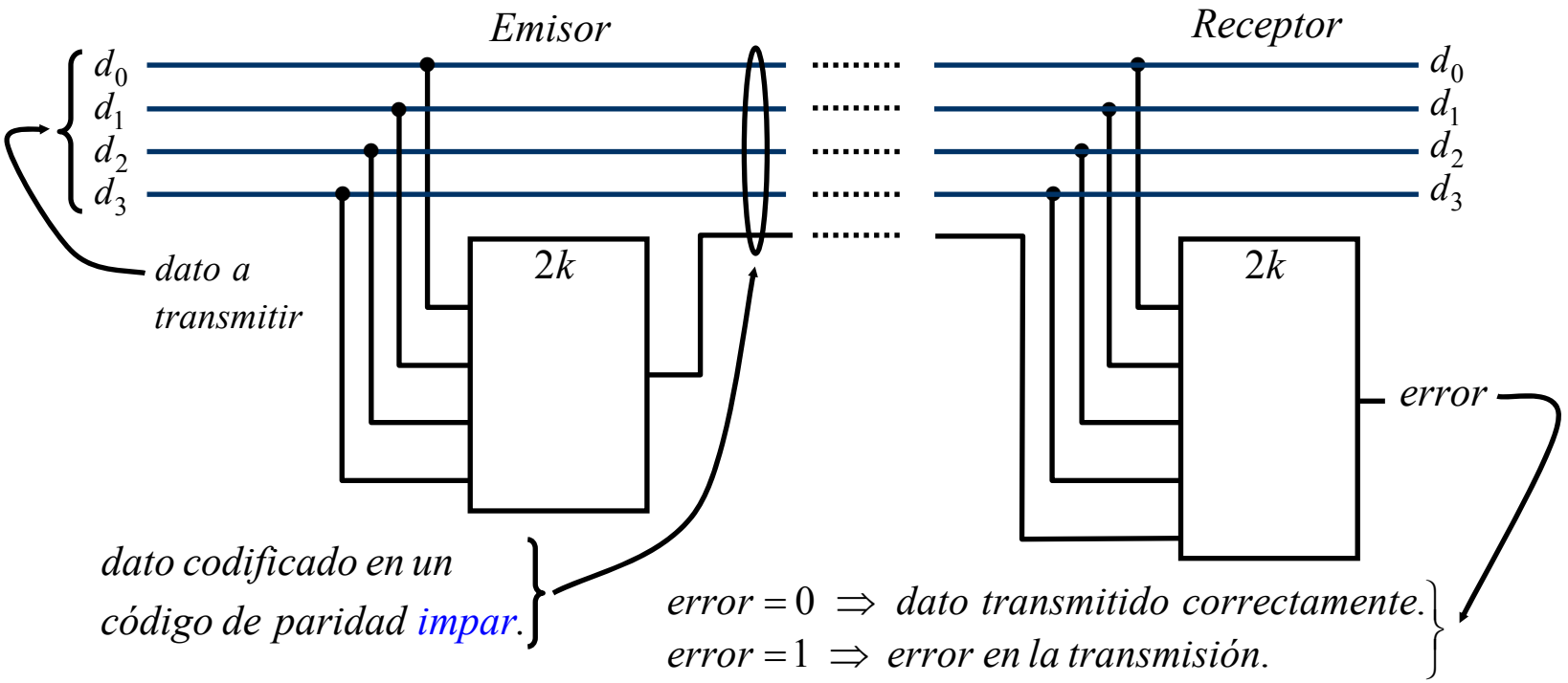

74xxx280
2k
0
1
1
1
0
1
1
0
1
(par)
(par)
5
6
1 (par)
0 (impar)


74xxx280
2k
1
0
1
0
0
0
0
0
1
5
6
0
1


74xxx280
2k +1
0
1
0
1
0
1
1
0
1
(impar)
(impar)
5
6
0 (par)
1 (impar)


74xxx280
2k +1
1
1
0
1
1
1
1
0
1
0
5
6
1
0

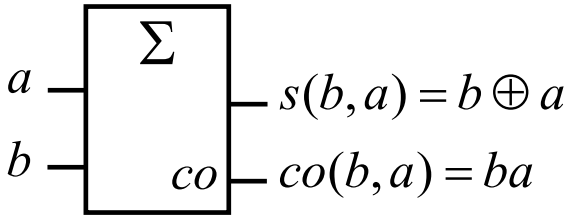
Ejemplo de aplicación:



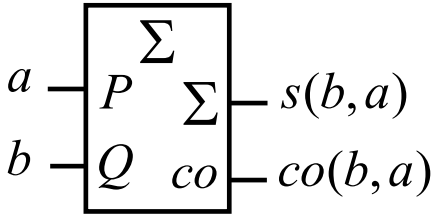
Curiosidad: *los microprocesadores Pentium comprueban la paridad de la información enviada tanto por el bus de datos como por el bus de direcciones.*

- Circuitos sumadores: son circuitos combinacionales que realizan sumas *aritméticas*

_ Semisumador de 1 bit: realiza la suma *aritmética* de dos dígitos binarios (b, a)



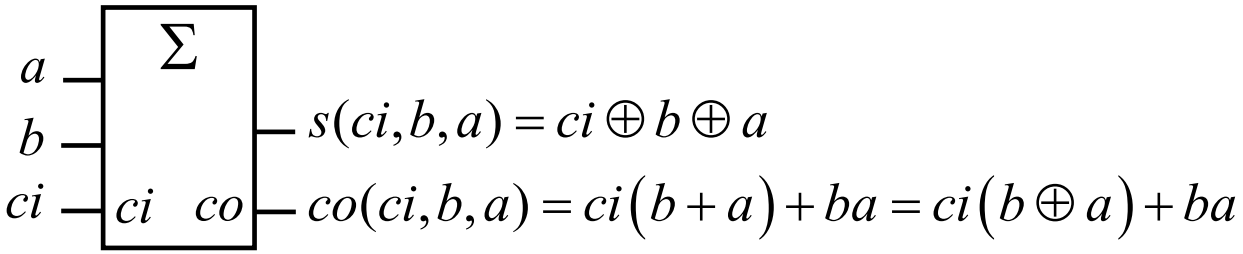
$$\begin{array}{r} b \\ +\ a \\ \hline co\ \ s \end{array}$$



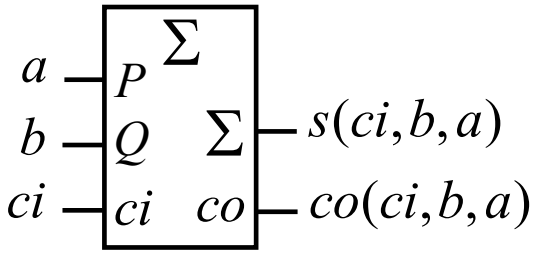
b	a	co	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$\begin{array}{r} b \\ +\ a \\ \hline co\ \ s \end{array} \equiv \begin{array}{r} 0 \\ +\ 1 \\ \hline 0\ \ 1 \end{array}$$

_ Sumador total (o completo) de 1 bit (*full-adder*): los circuitos indicados a continuación realizan la suma *aritmética* de tres dígitos binarios (*ci*, *b*, *a*)



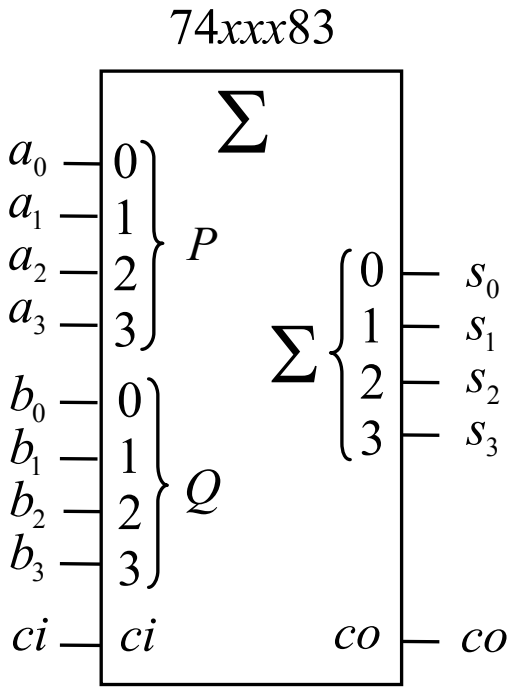
$$\begin{array}{r} ci \\ b \\ +\ a \\ \hline co\ \ s \end{array}$$



$$\begin{array}{r} {}^{2}\ {}^{1} \\ ci \\ b \\ +\ a \\ \hline co\ \ s \end{array} \equiv \begin{array}{r} 1 \\ 0 \\ +\ 1 \\ \hline 1\ \ 0 \end{array}$$

ci	b	a	^{2}co	^{1}s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

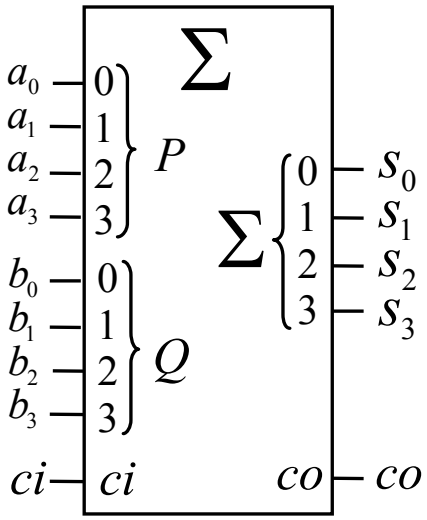
_ Sumador total de 4 bits: el circuito indicado a continuación realiza la suma *aritmética* indicada en la parte derecha



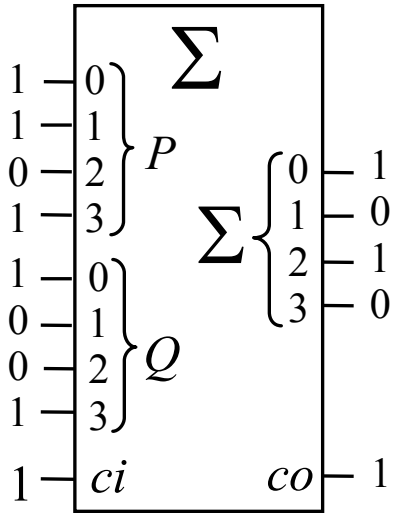
$$
\begin{array}{rcccc}
 & & & & ci \\
 & b_3 & b_2 & b_1 & b_0 \\
+ & a_3 & a_2 & a_1 & a_0 \\
\hline
co & s_3 & s_2 & s_1 & s_0
\end{array}
$$

Nota: los nombres asignados a los terminales de *entrada* y de *salida* no forman parte del símbolo (sólo se han puesto para poder explicar fácilmente la relación que hay entre los valores de las entradas y los valores de las salidas… ver suma indicada en la parte derecha).

Ejemplo:



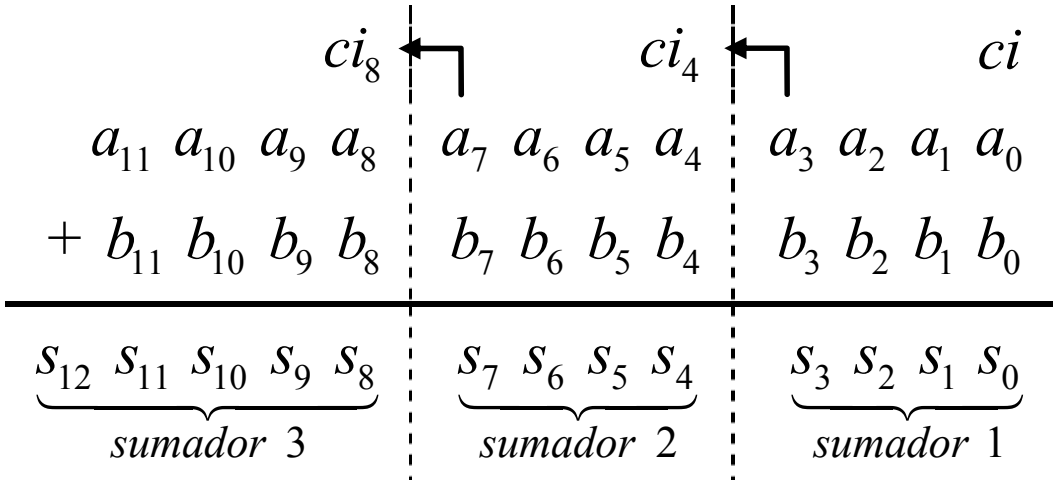
$$\begin{array}{rrrrr} & & & & ci \\ & b_3 & b_2 & b_1 & b_0 \\ + & a_3 & a_2 & a_1 & a_0 \\ \hline co & s_3 & s_2 & s_1 & s_0 \end{array}$$

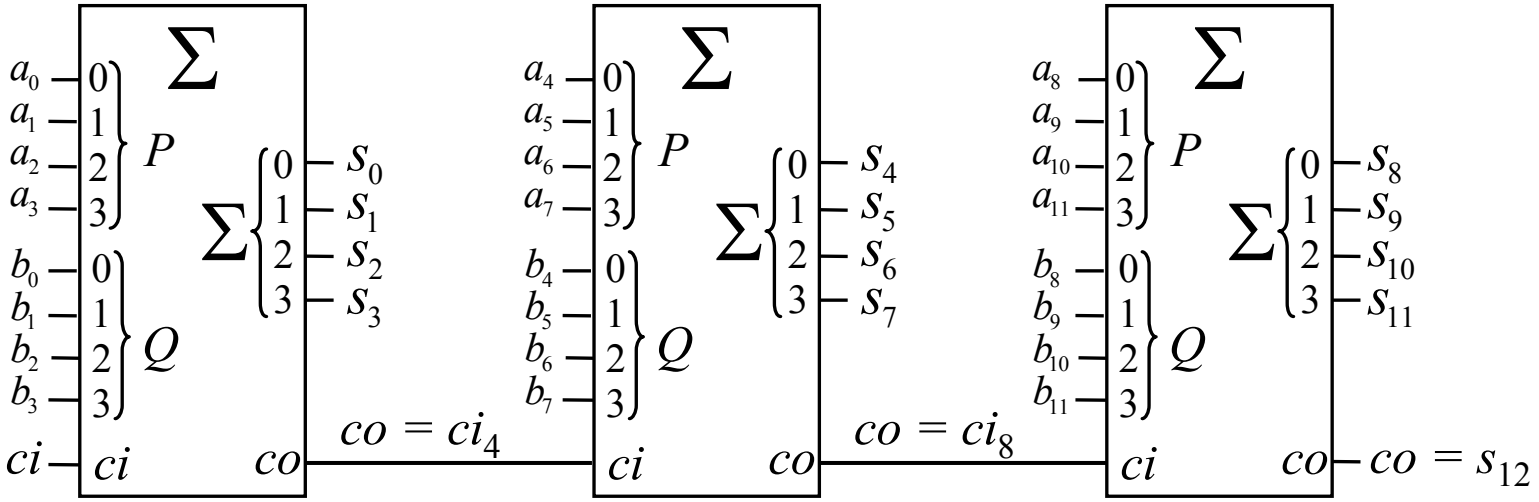


$$\begin{array}{rrrrr} & & & & 1 \\ & 1 & 0 & 1 & 1 \\ + & 1 & 0 & 0 & 1 \\ \hline & 1 & 0 & 1 & 0 & 1 \end{array}$$

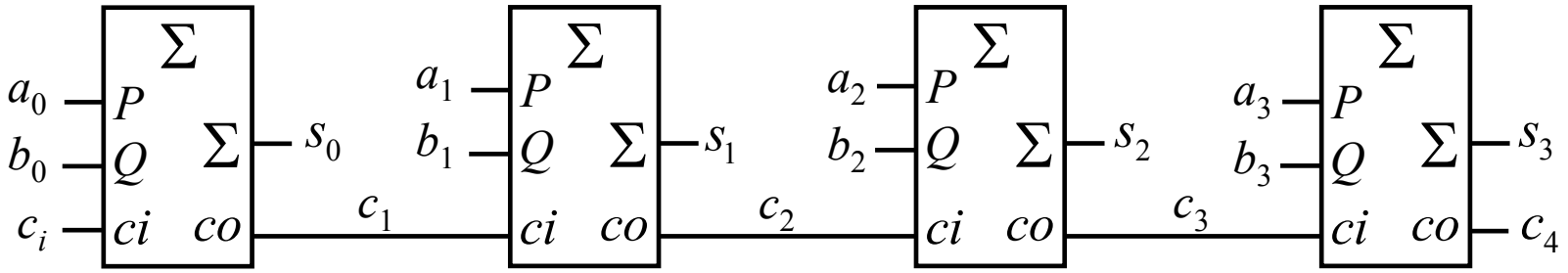
Ejemplo de utilidad de la entrada *ci* (*carry input*): realización de una suma por partes

(el *co* ≡ *carry output* de una suma pasa a ser el *ci* ≡ *carry input* de la siguiente suma)





Ejemplo de implementación de un sumador total de 4 *bits con sumadores totales de* 1 *bit*:



$$s_0 = ci \oplus b_0 \oplus a_0$$

$$c_1 = c_0 \cdot b_0 + c_0 \cdot a_0 + b_0 a_0$$

$$s_1 = c_1 \oplus b_1 \oplus a_1$$

$$c_2 = c_1 \cdot b_1 + c_1 \cdot a_1 + b_1 a_1$$

$$s_2 = c_2 \oplus b_2 \oplus a_2$$

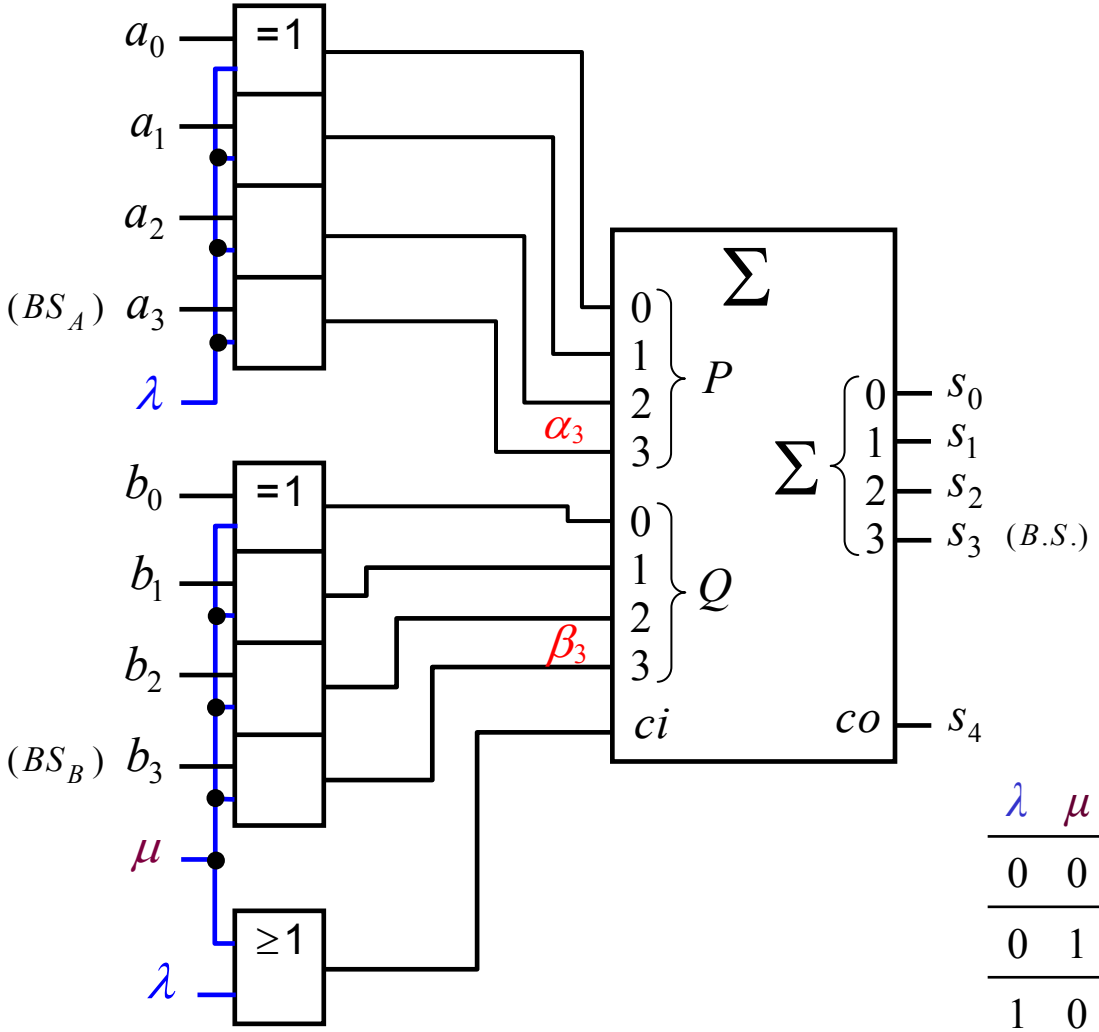
$$c_3 = c_2 \cdot b_2 + c_2 \cdot a_2 + b_2 a_2$$

$$s_3 = c_3 \oplus b_3 \oplus a_3$$

$$c_4 = co = c_3 \cdot b_3 + c_3 \cdot a_3 + b_3 a_3$$

$$\begin{array}{rcccc} & & & & ci \\ & b_3 & b_2 & b_1 & b_0 \\ + & a_3 & a_2 & a_1 & a_0 \\ \hline co & s_3 & s_2 & s_1 & s_0 \end{array}$$

Circuito sumador/restador de números de 4 bits codificados en el *cca*2.



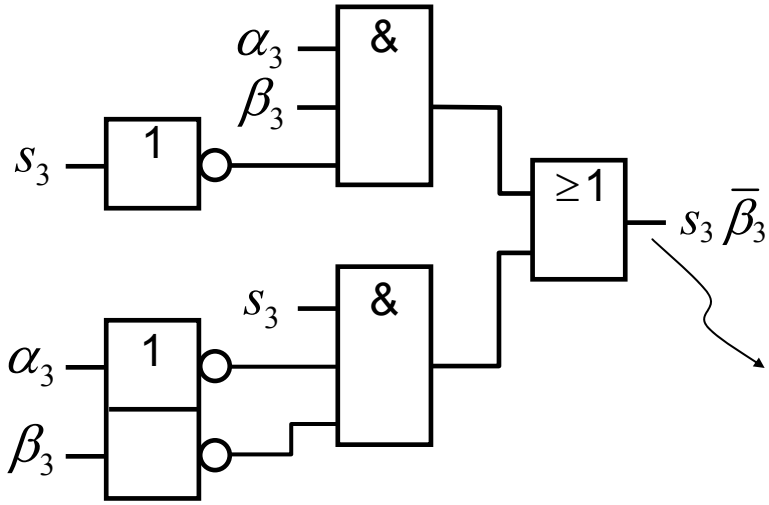
$$A(a_3\,a_2a_1a_0)^{BS}_{cca2}$$

$$B(b_3\,b_2b_1b_0)^{BS}_{cca2}$$

$$S(s_3\,s_2s_1s_0)^{BS}_{cca2}$$

λ	μ	*Operación realizada*
0	0	$A+B$
0	1	$A-B \rightarrow A + ca2(B) = A + ca1(B) +1$
1	0	$-A+B \rightarrow ca2(A)+B= ca1(A) + B +1$
1	1	*prohibida*

Circuito detector de desbordamiento (*overflow*).



$s_3\,\overline{\beta}_3\,\overline{\alpha}_3 + \overline{s}_3\,\beta_3\,\alpha_3 \equiv desbordamiento\ (overflow)$

si en esta salida hay un 1 significa que se ha producido un desbordamiento.

Recordatorio: puede ocurrir un desbordamiento cuando se suman dos números positivos o bien dos números negativos.

Aplicaciones de los circuitos sumadores:

_ Sumar números binarios

_ Restar/sumar números representados en el *cca*2.

_ Diseño de convertidores de código.

_ Implementación de comparadores de magnitud.

_ Implementación de detectores/generadores de paridad.

_ Implementación de algunas funciones lógicas.

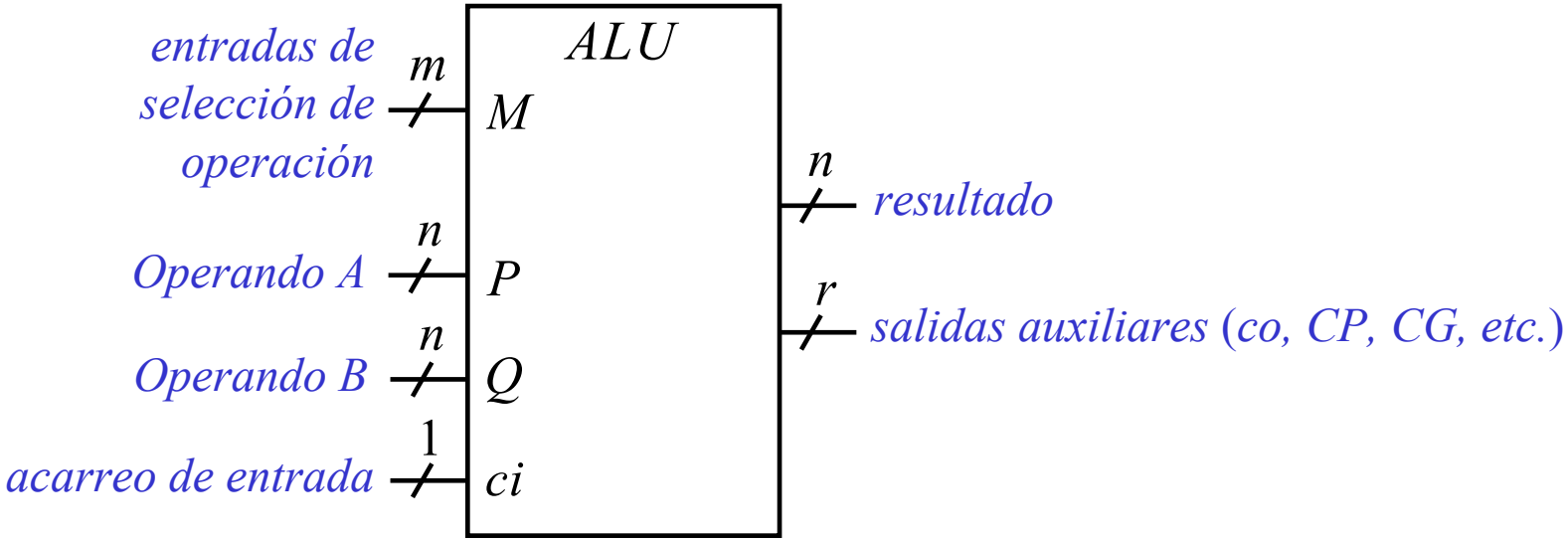
_ etc.

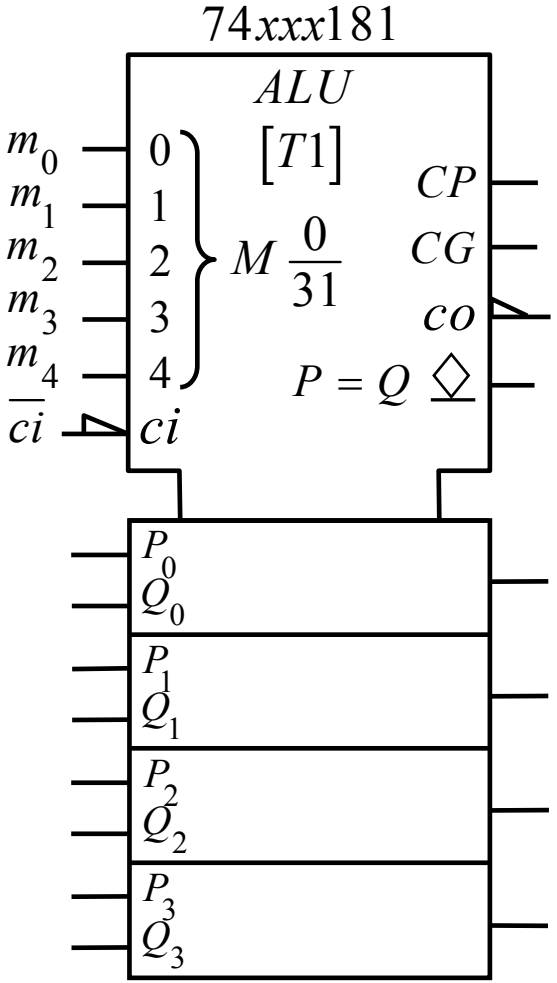
▪ Unidad aritmético-lógica

Es un circuito *combinacional* que puede realizar tanto operaciones *aritméticas* como *lógicas*. Las ALUs tienen dos tipos de entradas:

_ *Entradas de selección*: sirven para indicarle a la ALU la operación (*aritmética* o *lógica*) a realizar. El repertorio de instrucciones que puede realizar una ALU se indica en una tabla (ver siguiente diapositiva)

_ *Entradas de datos*: sirven para proporcionarle a la ALU los datos con los que debe operar (hay operaciones en las que sólo se utiliza un operando).





				Logic	*Arithmetic*
m_3	m_2	m_1	m_0	$m_4 = 1$	$m_4 = 0$
0	0	0	0	$\overline{P}$	*P plus ci*
0	0	0	1	$\overline{P+Q}$	$P+Q$ *plus ci*
0	0	1	0	$\overline{P}Q$	$P+\overline{Q}$ *plus ci*
0	0	1	1	0000	$(-1)_{cca2}$ *plus ci*
0	1	0	0	$\overline{PQ}$	*P plus* $(P\cdot\overline{Q})$ *plus ci*
0	1	0	1	$\overline{Q}$	$P\cdot\overline{Q}$ *plus* $(P+Q)$ *plus ci*
0	1	1	0	$P\oplus Q$	$(-1)_{cca2}$ *plus P minus Q plus ci*
0	1	1	1	$P\cdot\overline{Q}$	$(-1)_{cca2}$ *plus* $P\cdot\overline{Q}$ *plus ci*
1	0	0	0	$\overline{P}+Q$	*P plus* $P\cdot Q$ *plus ci*
1	0	0	1	$\overline{P\oplus Q}$	*P plus Q plus ci*
1	0	1	0	Q	$P\cdot Q$ *plus* $(P+\overline{Q})$ *plus ci*
1	0	1	1	$P\cdot Q$	$(-1)_{cca2}$ *plus* $P\cdot Q$ *plus ci*
1	1	0	0	1111	*P plus P plus ci*
1	1	0	1	$P+\overline{Q}$	*P plus* $(P+Q)$ *plus ci*
1	1	1	0	$P+Q$	*P plus* $(P+\overline{Q})$ *plus ci*
1	1	1	1	P	$(-1)_{cca2}$ *plus P plus ci*

$+ \equiv$ *suma lógica* *plus / minus* $\equiv$ *suma / resta aritmética*

En las operaciones aritméticas, *minus Q* representa la operación *ca*2(*Q*)]

Ejemplo:

m_4	m_3	m_2	m_1	m_0	*Operación realizada*
0	0	0	0	0	$P\ plus\ ci$
1	0	0	0	0	$\overline{P}$

$P : P_3 P_2 P_1 P_0$

(operación *aritmética*: $m_4 = 0$)

$$\left.\begin{array}{r} P:1101 \\ ci:1 \end{array}\right\} \Rightarrow \quad P\ plus\ ci:\ 1110$$

(operación *lógica*: $m_4 = 1$)

$$P:1101 \ \Rightarrow \ \overline{P}:0010$$

Notas:

- $+ \equiv$ *suma lógica*
- *plus*/*minus* $\equiv$ *suma*/*resta aritmética*
- las operaciones lógicas se realizan '*bit a bit*'

Ejercicio 1: Dados 2 números binarios de 4 bits (sin signo), diseñar un circuito que proporcione el mayor de ambos o bien el valor cero en el caso de que sean iguales.
Nota: no se pueden utilizar puertas lógicas

Ejercicio 2: Diseñar un circuito al que llegan 3 números binarios A, B y C de 4 bits y una señal de control d. El circuito funciona de la siguiente manera:

_ Si $d = 1$, entonces el circuito debe indicar la relación entre B y C ($B > C$, $B < C$ o $B = C$)

_ Si $d = 0$, entonces el circuito debe indicar la relación entre B y A ($B > A$, $B < A$ o $B = A$)
Nota: no se pueden utilizar puertas lógicas

Ejercicio 3: Diseñar un circuito que calcule la suma de 3 números binarios $A = 1011_2$, $B = 010_2$ y $C = 100_2$. Nota: no se pueden utilizar puertas lógicas.

Ejercicio 4: Diseñar un convertidor de palabras código que en función del valor de una entrada auxiliar α realice las siguientes conversiones:

_ para $\alpha = 0$ se debe realizar la conversión de $BCD_{exceso\ 3} \rightarrow BCD_{natural}$

_ para $\alpha = 1$ se debe realizar la conversión de $BCD_{natural} \rightarrow BCD_{exceso\ 3}$

Nota: se pueden utilizar bloques funcionales combinacionales y 1 puerta lógica!!!.

Ejercicio 5: Diseñar un convertidor de palabras código que en función del valor de una entrada auxiliar α realice las siguientes conversiones:

_ para $\alpha = 0$ se debe realizar la conversión de $BCD_{exceso\ 3} \rightarrow BCD_{Aiken}$

_ para $\alpha = 1$ se debe realizar la conversión de $BCD_{Aiken} \rightarrow BCD_{exceso\ 3}$

Nota: se pueden utilizar bloques funcionales combinacionales y el menor número posible de puertas XOR.