4 BLOQUES COMBINACIONALES CONFIGURACIONES RETICULARES

Un sistema combinacional es aquel cuyas salidas pueden obtenerse por «combinación booleana» de sus entradas, es decir, a través de un conjunto de operaciones lógicas sobre sus variables de entrada; lo cual sucede siempre que a cada vector de entrada le corresponde un solo vector de salida.

En el diseño de sistemas digitales resulta útil aplicar una estrategia de división modular: cuando un sistema resulta complejo es conveniente descomponerlo en subsistemas y realizar su síntesis por separado. Existe una serie de subsistemas conceptuales que aparecen con mucha frecuencia y conviene identificarlos como módulos típicos. Entre ellos ya hemos visto en el capítulo anterior los bloques aritméticos y lógicos (y en el capítulo segundo los decodificadores y multiplexores, a los que volveremos en el presente tema).

Multiplexores y demultiplexores son bloques dedicados a organizar y seleccionar las líneas de transmisión de la información digital, mientras que los decodificadores seleccionan directamente una de entre n posibilidades. Todos ellos aprovechan la propiedad de selección o filtrado que poseen los términos mínimos en relación con su número binario o vector de entrada.

Los codificadores sirven para cambiar el código en que se expresa la información. Una misma información puede representarse en palabras binarias de muy diferentes formas según el código que se utilice; la conversión entre códigos, aunque no modifique la información, es una labor necesaria y útil (por ejemplo, la conversión de una cifra BCD al código de 7 segmentos permite representarla «gráficamente» en su forma habitual). Cualquier conjunto de n funciones de m variables puede considerarse como un conversor de un código de m bits a otro de n bits (si bien puede suceder que a diferentes vectores de entrada corresponda un mismo vector de salida, lo cual supone pérdida de información).

Todos estos bloques (los considerados en este capítulo y en el anterior) constituyen los módulos habituales de diseño de un sistema combinacional; sin perjuicio de la necesidad de puertas individuales o conjuntos de funciones booleanas particulares (codificadores específicos). El primer paso en un diseño será efectuar la división del sistema en bloques, es decir, detallar una arquitectura propia en forma de diagrama de bloques.

Decodificadores, multiplexores, demultiplexores y codificadores son bloques que suelen presentar un gran número de conexiones internas; la configuración reticular de las mismas simplifica mucho su descripción gráfica.

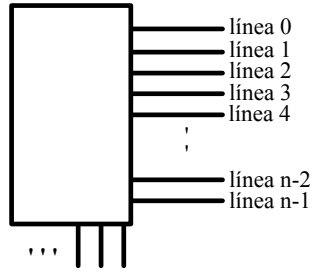
Tomás Pollán Santamaría. Tecnología Electrónica. E.U.I.T.I.Z. Universidad de Zaragoza. tpollan@posta.unizar.es

4.1. Decodificadores, multiplexores y demultiplexores

Una palabra de **n** bits contiene $\mathbf{2^n}$ posibilidades (numeradas de 0 a 2^n-1), de forma que **n** entradas permiten seleccionar una de entre $\mathbf{2^n}$ líneas, asignando a cada una de ellas un número de **n** dígitos. Esta función de selección puede configurarse a través de los términos mínimos de las entradas: un término mínimo adopta valor **1** para su correspondiente vector de entrada y con cualquier otro vector da valor **0**.

4.1.1. Decodificadores

Un *decodificador* es un «selector de línea» con **m** entradas y $\mathbf{2^m}$ líneas de salida (numeradas con **m** dígitos binarios); en cada momento el número binario que recibe como vector de entrada determina que la correspondiente línea de salida se encuentre activa (valor **1**) y el resto de ellas a **0**. Este bloque digital se denomina *decodificador* porque «decodifica» las $\mathbf{2^m}$ posibilidades de un número binario de **m** dígitos sobre $\mathbf{2^m}$ líneas.



La numeración de las líneas de salida comienza por la línea **0**, pues el primer vector de entrada sobre las líneas de selección será el **0...00**, hasta llegar a la línea $\mathbf{2^m-1}$, correspondiente al último vector de selección, **1...11**.

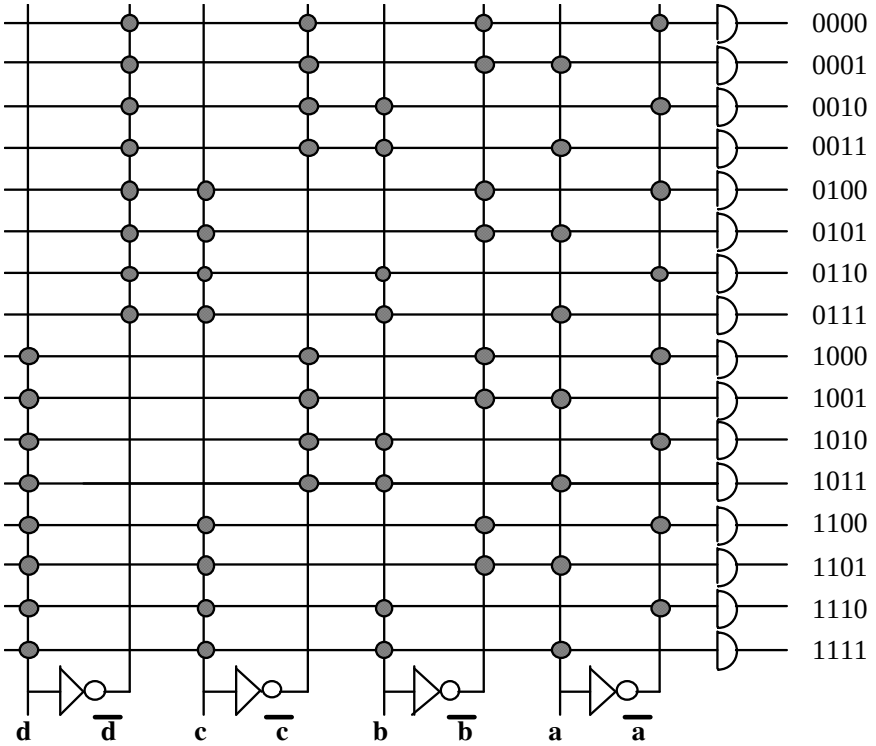
El decodificador es un bloque digital de particular importancia:

- es un selector de línea, que discrimina las líneas de salida según su número binario en las entradas;
- corresponde al conjunto de todos los términos mínimos de sus entradas;
- contiene todas las opciones posibles de sus vectores de entrada y hace corresponder a cada una de ellas con una de las líneas de salida;
- su vector de salida corresponde a un código de «un solo uno» (una sola línea activada, la que corresponde al vector presente en sus entradas).

Asimismo, la disponibilidad de todos los términos mínimos permite utilizar al decodificador para construir funciones booleanas de sus entradas, en su forma canónica; para ello bastará añadir una puerta "**o**" para cada función [precisamente, los decodificadores fueron introducidos con tal finalidad en 2.4.].

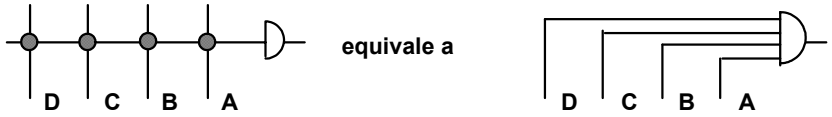
Un decodificador estará configurado por los términos mínimos de sus entradas, cada uno de los cuales dará lugar a una línea de salida: cada salida puede construirse con una simple puerta "**y**" de **m** entradas, formando el término mínimo que corresponde a su numeración.

Consideremos un decodificador de 4 variables de entrada: sus 16 líneas de salida corresponderán a los 16 términos mínimos de 4 variables y serán las salidas de otras tantas puertas "**y**" de 4 entradas que podemos dibujar en la forma siguiente:



Decodificador de 4 variables (16 términos mínimos)

La figura anterior presenta una estructura reticular, gracias a que la representación habitual de puertas con varias entradas ha sido sustituida por una sola «línea de entrada» sobre la cual «se conectan» las diversas entradas:



El dibujo de varias entradas «conectadas» sobre una «línea de entrada» única no significa que tales entradas se interconecten formando un nudo común (pues ello daría lugar a un grave conflicto entre sus valores booleanos cuando fueran diferentes) sino que pretende representar, en forma simplificada, las diversas entradas de la puerta lógica (cada una de las cuales actúa por separado).

Esta forma «reticular» de dibujar bloques de muchas salidas (o, también, de muchas entradas) es sumamente útil porque facilita la «lectura» y la comprensión del circuito (evitando el entrecruzamiento de un gran número de líneas) y permite apreciar la distribución de las variables respecto a las diversas puertas lógicas y las regularidades o simetrías existentes en tal distribución.

Además, esta forma de representar las conexiones sobre las puertas lógicas permite trasladar el conjunto de conexiones (o el equivalente conjunto de entradas) sobre una matriz de «ceros» y «unos», en la cual el 1 representa una conexión y el 0 indica la ausencia de ella. Al decodificador de la figura le corresponderá la siguiente matriz de conexiones:

Matriz de conexiones	**D**	$\overline{\mathbf{D}}$	**C**	$\overline{\mathbf{C}}$	**B**	$\overline{\mathbf{B}}$	**A**	$\overline{\mathbf{A}}$
del decodificador	0	1	0	1	0	1	0	1
de 4 variables	0	1	0	1	0	1	1	0
(16 líneas de salida)	0	1	0	1	1	0	0	1
	0	1	0	1	1	0	1	0
	0	1	1	0	0	1	0	1
	0	1	1	0	0	1	1	0
	0	1	1	0	1	0	0	1
	0	1	1	0	1	0	1	0
	1	0	0	1	0	1	0	1
	1	0	0	1	0	1	1	0
	1	0	0	1	1	0	0	1
	1	0	0	1	1	0	1	0
	1	0	1	0	0	1	0	1
	1	0	1	0	0	1	1	0
	1	0	1	0	1	0	0	1
	1	0	1	0	1	0	1	0

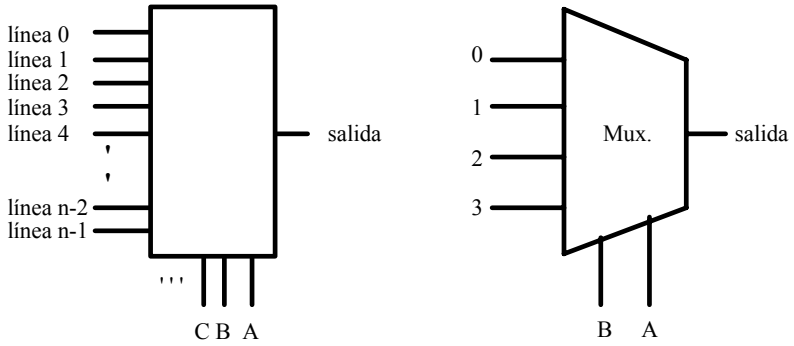
4.1.2. Multiplexores

La función *multiplexado* consiste en seleccionar una de entre varias fuentes de datos para enviarlos por una misma línea; un multiplexor realiza el multiplexado de **n** líneas de entrada a una sola línea de salida.

Un *multiplexor* es un subsistema digital que selecciona una de entre **n** fuentes de datos, comunicándola con la línea de salida del mismo; dispone de **n** líneas de entrada, **1** línea de salida y un conjunto de **m** líneas de control o selección, tal que $\mathbf{n = 2^m}$ (o bien, $\mathbf{2^m \geq n}$). En cada momento el vector de entrada en las líneas de selección determina (por su número) la línea de entrada que «queda conectada» a la salida.

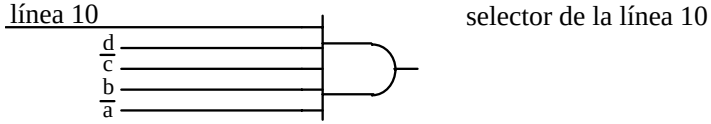
Este tipo de actuación se denomina también *muestreo*, pues selecciona de entre diversas líneas de entrada la que interesa en cada caso: el multiplexor «muestrea» la línea cuyo número binario es establecido sobre las entradas de control del mismo.

Las siguientes figuras representan, respectivamente, un multiplexor genérico de n líneas y un multiplexor de 4 líneas:



Los términos mínimos (por su función selectiva respecto a su propio vector de entrada) sirven para filtrar la línea de entrada que corresponde al vector presente en las líneas de selección; para ello basta realizar la operación "**y**" entre cada línea de entrada y el término mínimo que corresponde a su número en binario.

Por ejemplo, para la línea $10_{(10} = \mathbf{1010}_{(2}$:

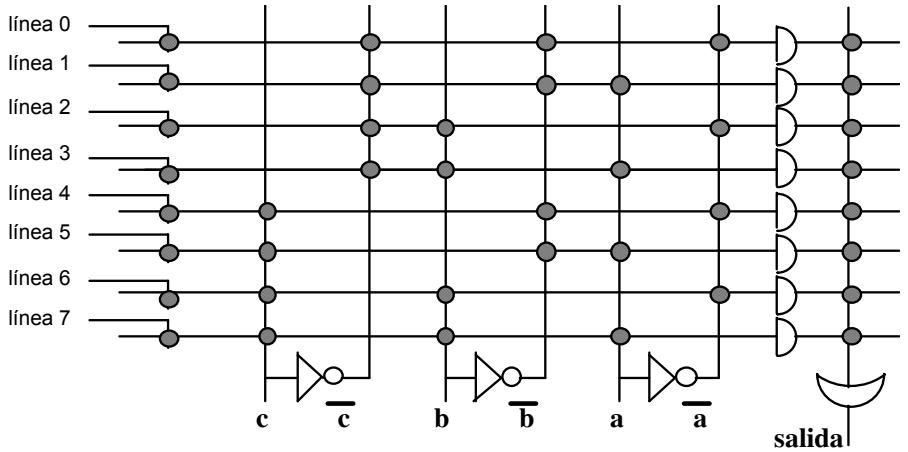


Sólo cuando las variables **dcba** alcanzan el valor $\mathbf{1010}_{(2} = 10_{(10}$ dicha puerta presenta en su salida el valor booleano existente en la línea 10; en cualquier otro caso, su salida es **0**.

De esta forma, un multiplexor estará formado por $\mathbf{2^m}$ puertas "**y**" de **m+1** entradas confluyendo sobre una puerta "**o**" de salida (que dará paso a la única línea seleccionada en cada momento):

$$Y = \sum L_i \,.\, \text{término mínimo}_i$$
$$= L_0 \ldots \bar{c}.\bar{b}.\bar{a} + L_1 \ldots \bar{c}.\bar{b}.a + L_2 \ldots \bar{c}.b.\bar{a} + L_3 \ldots \bar{c}.b.a + L_4 \ldots c.\bar{b}.\bar{a} + \ldots\ldots$$

Por ejemplo, para 8 líneas de entrada (3 líneas de control) el multiplexor resultante, representado en configuración reticular, es el de la figura siguiente:

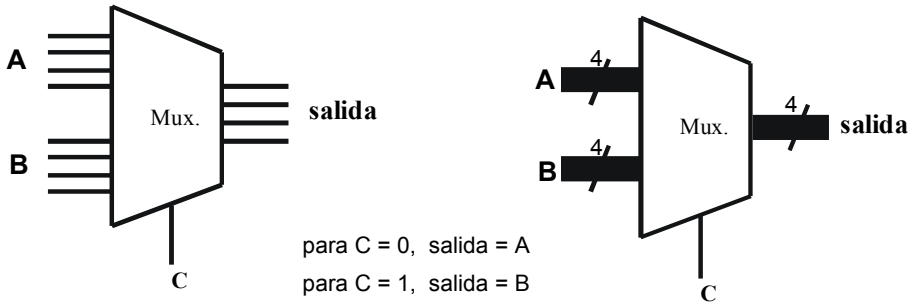


Multiplexor de 8 líneas (3 entradas de control)

Habida cuenta de que $a.b + c.d = (a * b) * (c * d)$ [suma de productos = *Nand de Nands*] las puertas de la figura anterior pueden ser sustituidas, todas ellas a la vez, por puertas "**y-negada**" (*Nand*).

Si asignamos valores booleanos a las líneas de entrada (las conectamos a **0** o a **1**), el multiplexor selecciona uno de dichos valores. De esta forma, podemos construir cualquier función de sus entradas de control, fijando en las líneas de entrada los valores de la «tabla de verdad» de la misma: las variables de control seleccionan sobre la propia tabla de la función el valor booleano que debe adoptar (**look-up-table**) [también los multiplexores fueron introducidos con tal finalidad en II.4]

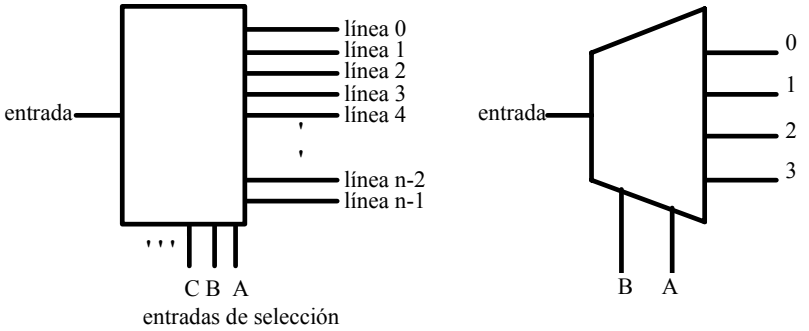
Cuando los datos a multiplexar constan de varios bits y éstos son enviados en paralelo, en lugar de una única línea se tiene un *bus* o conjunto de líneas que comunican una palabra binaria y la función *multiplexado* consiste igualmente en seleccionar uno de entre varios buses de entrada para comunicarlo con el bus de salida; para un bus de longitud **p**, es decir compuesto por **p** líneas, se necesitarán **p** multiplexores en paralelo.



Multiplexor de dos buses de 4 líneas (1 entrada de selección)

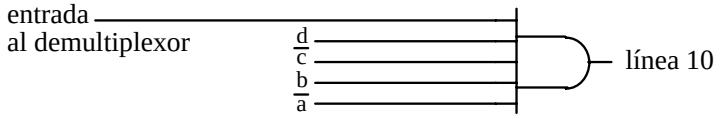
4.1.3. Demultiplexores

Los *demultiplexores* realizan una función contraria a la del multiplexor: reciben los datos por una sola entrada y los dirigen a una de entre **n** líneas de salida, seleccionables numéricamente por **m** líneas de control; en cada momento el dato presente en su entrada aparece en la salida cuyo número binario coincide con el establecido en las entradas de control y el resto de líneas de salida permanecen a **0**.

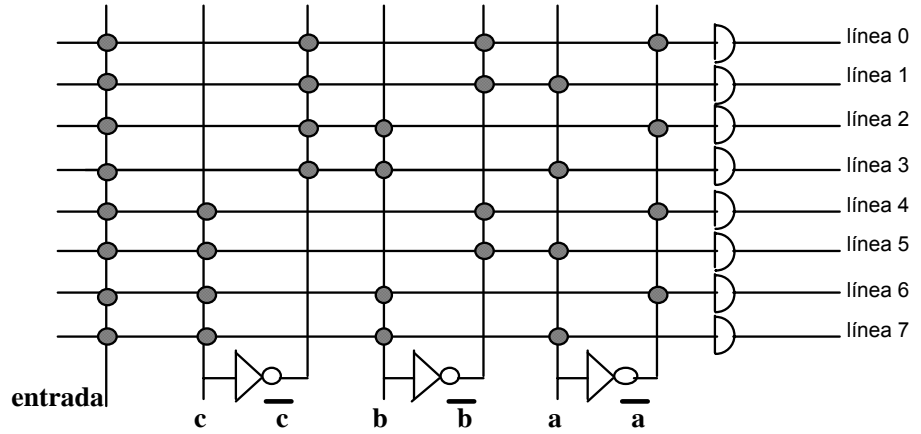


Internamente un demultiplexor está constituido por **n** puertas "**y**" a todas las cuales accede la línea de entrada junto con el término mínimo correspondiente de las entradas de control; dicho término mínimo realiza la selección permitiendo la actuación de una sola de las **n** puertas "**y**": salida_i = entrada . término mínimo_i .

Por ejemplo, para la línea de salida número $10_{(10} = \mathbf{1010}_{(2}$:



La figura siguiente representa el esquema de puertas lógicas que configuran un demultiplexor de 8 líneas:



Demultiplexor de 8 líneas (3 entradas de selección)

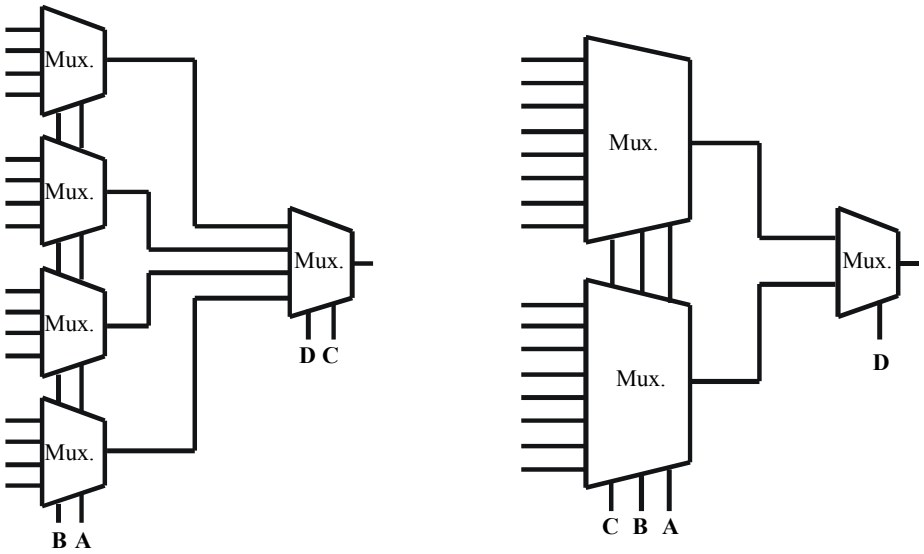
Anulando la línea de entrada del demultiplexor, es decir, conectándola a **1**, se obtiene directamente un decodificador: la línea de salida seleccionada por el número binario establecido en las entradas se encontrará a **1** y el resto de líneas de salida estarán a **0**.

Tanto los multiplexores como los demultiplexores son modularmente ampliables:

- un multiplexor de **n** líneas a cuyas entradas se conectan **n** multiplexores de **k** líneas, conectándose a su vez las entradas de control de éstos en paralelo, se convierte en un multiplexor de **n.k** líneas;
- un demultiplexor de **n** líneas cuyas salidas se conectan a **n** demultiplexores de **k** líneas, conectándose también las entradas de control de éstos en paralelo, se convierte en un demultiplexor de **n.k** líneas;
- en el caso de los decodificadores, su ampliación se realiza conectando sus **n** salidas a otros tantos demultiplexores de **k** líneas, con lo que se obtiene un decodificador de **n.k** líneas.

Por ejemplo, cuatro multiplexores de 4 líneas conectadas sus salidas a las entradas de otro análogo (de 4 líneas) configuran un multiplexor de 16 líneas; el mismo multiplexor de 16 líneas puede configurarse con dos multiplexores de 8 líneas conectados a uno de sólo 2 líneas.

Lo mismo sucede con cuatro demultiplexores de 4 líneas cuyas entradas se conectan a las salidas de otro demultiplexor de 4 líneas o también con dos demultiplexores de 8 líneas conectados a las salidas de uno de 2 líneas.

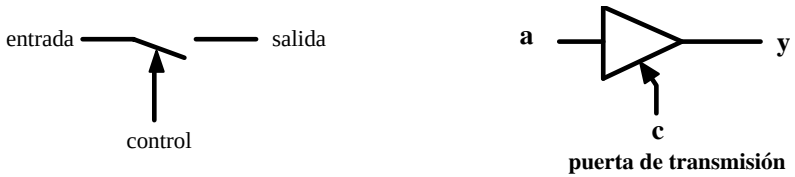


Multiplexor de 16 líneas resultante de la conexión de multiplexores más pequeños

[La figura correspondiente a la conexión de demultiplexores sería la misma, con el flujo de datos de derecha a izquierda: entradas por la derecha, salidas hacia la izquierda.]

4.2. Multiplexado mediante puertas de transmisión: tri-estado

Un interruptor (con una entrada de control) constituye una puerta muy simple: cuando conduce «deja pasar» a su salida el valor booleano presente en su entrada y en corte no permite el paso a su través. Ahora bien, desde el punto de vista de la salida, hay tres situaciones posibles: las dos que corresponden a los valores booleanos **0**/**1** y una tercera de *desconexión* cuando el interruptor se encuentra en corte.



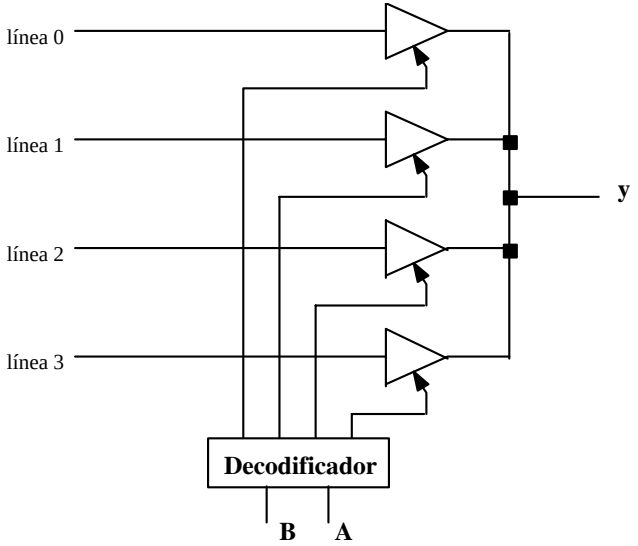
puerta de transmisión

La operación que realiza un interruptor no coincide con las operaciones booleanas básicas (ni con sus derivadas) sino que da lugar a un nuevo tipo de puerta, a la que denominaremos «puerta de transmisión», cuya actuación consiste en permitir/inhibir el paso entre la entrada y la salida.

La salida de una puerta de transmisión presenta tres estados posibles: **0**, **1** y **desconexión**; este tercer estado no corresponde a un valor booleano concreto, sino a una nueva situación física, en la cual la puerta de transmisión no determina (no influye sobre) el valor de la salida.

Se emplea la denominación *tri-estado* para aludir esta triple posibilidad que ofrecen las puertas de transmisión; el estado de desconexión recibe el nombre de «alta impedancia» (pues tal es la situación física a la que corresponde) y se representa en forma abreviada con el símbolo **Z** (o bien, **z**↑ en forma más explícita). Los tres valores del tri-estado son **0**, **1** y **Z**.

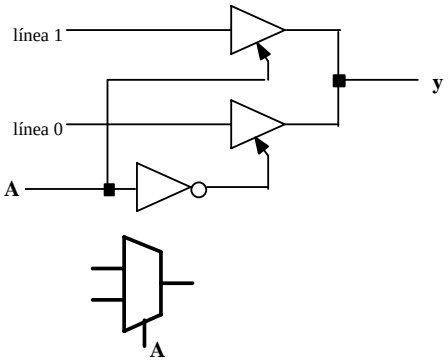
Las puertas de transmisión (o adaptadores tri-estado) no son útiles para configurar funciones lógicas, pero sí para multiplexar señales (para construir multiplexores): podemos conectar las salidas de varias puertas de transmisión y seleccionar una de las señales que reciben, haciendo que la correspondiente puerta esté activada y todas las demás en corte.



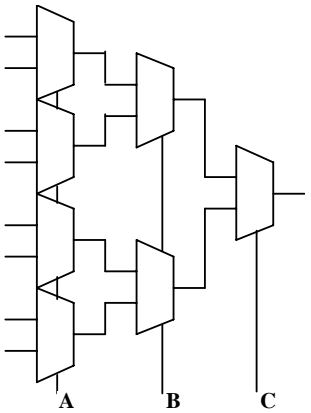
Multiplexor de 4 líneas, construido con 4 puertas de transmisión y un decodificador

Las puertas lógicas no permiten la conexión de varias salidas, unidas en un mismo nudo, porque daría lugar a posibles conflictos entre los dos valores booleanos **0**/**1**; en cambio, las puertas de transmisión pueden agrupar sus salidas, con tal de que en cada momento conduzca una sola de ellas y las demás se encuentren en corte.

Dos puertas de transmisión con un inversor entre sus entradas de control dan lugar a un pequeño multiplexor de 2 líneas; la conexión de tales multiplexores «en cascada» permite construir multiplexores de mayor número de líneas.

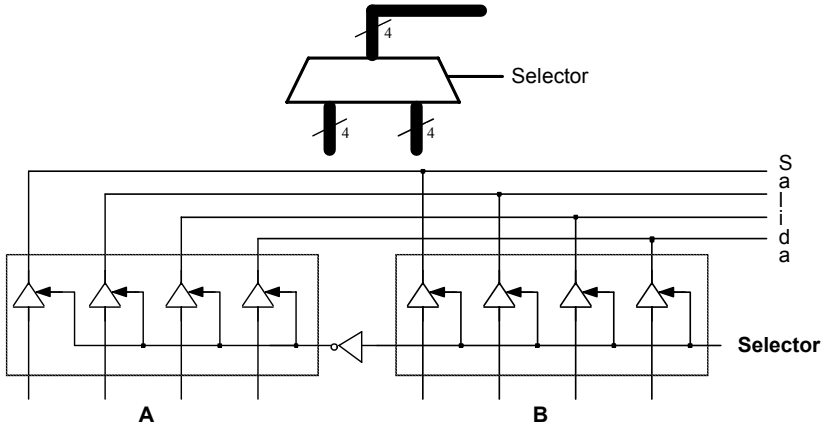


Multiplexor de 2 líneas



Multiplexor de 8 líneas

De igual forma la utilización de varias puertas de transmisión en paralelo, con la entrada de control común, permite el multiplexado de buses; la siguiente figura presenta un multiplexor de dos buses de 4 líneas (cuando Selector = **0**, pasa el bus A a la salida; con Selector = **1**, pasa el B).



Multiplexor de 2 buses de 4 líneas

Los conjuntos de puertas de transmisión en paralelo reciben también el nombre de adaptadores tri-estado (*buffer tri-state)* y pueden ser, asimismo, de tipo inversor (puertas de transmisión con un inversor previo delante de cada una de ellas).

4.3. Codificadores: conversores de código

En los sistemas digitales toda la información se maneja codificada en palabras binarias. Pero existen múltiples maneras de codificar una misma información; por ejemplo, el valor numérico 94 puede ser representado, entre otros, en los siguientes códigos:

- Binario normal (base 2) **1011110**
- Gray **1110001**
- BCD (decimal codificado en binario) **1001 0100**
- ASCII (representación de texto) **0111001 0110100**
- Un solo uno (con 10 niveles) **1000000000 0000010000**
- Barras (de 9 niveles) **111111111 000001111**
- Siete segmentos (para visualizadores) **1111011 0110011**
- …

El código binario es la representación normal de números, utilizando el sistema de numeración base 2, mientras que BCD respeta la estructura decimal del número (base 10). El código ASCII se emplea para la representación de caracteres en el tratamiento de textos y es el adecuado para enviar tal información a una impresora.

Los tres códigos siguientes pueden utilizarse para la representación visual de las cifras decimales. El código de «un solo uno» utiliza palabras binarias que contienen solamente un «uno» y corresponde al vector de salida de un teclado decimal (asimismo, era el utilizado por los tubos «Nixie» de 10 filamentos, uno con la forma de cada cifra). El código de barras representa las cifras decimales mediante barras luminosas (de nueve niveles sucesivos) y el código de siete segmentos las representa en su forma habitual (a través de 7 segmentos, con la forma de un 8).

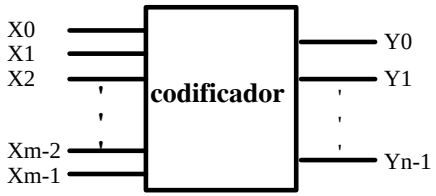
Muchas veces interesa pasar de una codificación a otra; por ejemplo, la entrada de un teclado de 10 pulsadores independientes servirá para introducir cifras decimales en código de «un solo uno» pero, para operar con dichas cifras, generalmente interesará transformarlas a BCD y, una vez obtenidos los resultados, será preciso convertir sus cifras a 7 segmentos para representarlas en un visualizador.

Para efectuar un cambio de código, basta ejecutar la tabla booleana que relaciona ambas codificaciones: si se trata de pasar de una codificación en palabras de **m** dígitos a otra que utiliza **n** bits, dicha tabla representa **n** funciones booleanas (**n** columnas) de **m** variables de entrada ($\mathbf{2^m}$ filas):

$$X_{m-1}\,X_{m-2}\,...\,X_2\,X_1\,X_0 \;\rightarrow\; Y_{n-1}\,Y_{n-2}\,...\,Y_2\,Y_1\,Y_0$$

$$Y_j = f_j(X_i) \qquad \text{n funciones de m variables}$$

Un *codificador* o conversor de código es un bloque digital con **m** entradas y **n** salidas, configurado internamente por **n** funciones booleanas de **m** variables.



Cada codificador es un bloque específico, cuyas **n** funciones corresponden a la conversión de código que efectúa.

Recíprocamente, cualquier conjunto de **n** funciones de **m** variables puede ser considerado como un conversor de un código de **m** bits a otro de **n** bits; ahora bien, en el caso de que un mismo vector de salida se repita para dos vectores de entrada diferentes, existirá pérdida de información.

Consideremos a continuación, a modo de ejemplos, tres codificadores o conversores de código típicos: un conversor BCD a 7 segmentos, un encodificador de 9 líneas a BCD y un codificador de prioridad de 9 líneas de entrada.

4.3.1. Codificador BCD a 7 segmentos

Las cifras decimales (del 0 al 9) se expresan en BCD con cuatro dígitos binarios (del **0000** al **1001**); pero también dichas cifras se pueden representar mediante un visualizador de 7 segmentos para lo cual hay que codificarlas en 7 bits, cada uno de los cuales controla (*enciende*) uno de los segmentos:

El conversor entre el código BCD y el de 7 segmentos ha de tener 4 entradas y 7 salidas; la tabla de conversión representará las 7 funciones booleanas de 4 entradas que realizan dicho cambio de código; a partir de ella, pueden obtenerse directamente las expresiones algebraicas de dichas funciones que servirán para controlar cada uno de los 7 segmentos.

La tabla correspondiente al paso de *BCD* a 7 segmentos es la siguiente:

nº decimal	binario				7 segmentos						
	D	**C**	**B**	**A**	**a**	**b**	**c**	**d**	**e**	**f**	**g**
0	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**0**
1	0	0	0	1	**0**	**1**	**1**	**0**	**0**	**0**	**0**
2	0	0	1	0	**1**	**1**	**0**	**1**	**1**	**0**	**1**
3	0	0	1	1	**1**	**1**	**1**	**1**	**0**	**0**	**1**
4	0	1	0	0	**0**	**1**	**1**	**0**	**0**	**1**	**1**
5	0	1	0	1	**1**	**0**	**1**	**1**	**0**	**1**	**1**
6	0	1	1	0	**1**	**0**	**1**	**1**	**1**	**1**	**1**
7	0	1	1	1	**1**	**1**	**1**	**0**	**0**	**0**	**0**
8	1	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**
9	1	0	0	1	**1**	**1**	**1**	**1**	**0**	**1**	**1**

A partir de ella, utilizando para simplificar los correspondientes diagramas de Karnaugh y expresando las funciones resultantes en términos de puertas "**y-negada**", se obtienen las siguientes expresiones:

a	OO	O1	11	1O
OO	**1**	**0**	**1**	**1**
O1	**0**	**1**	**1**	**1**
11	**X**	**X**	**X**	**X**
1O	**1**	**1**	**X**	**X**

Por términos suma:

$$a = (D + C + B + \overline{A}) \,.\, (D + \overline{C} + B + A) =$$
$$= D + B + C.A + \overline{C}.\overline{A} =$$
$$= \text{Nand}(\overline{D}, \overline{B}, C * A, \overline{C} * \overline{A})$$

b	OO	O1	11	1O
OO	**1**	**1**	**1**	**1**
O1	**1**	**0**	**1**	**0**
11	**X**	**X**	**X**	**X**
1O	**1**	**1**	**X**	**X**

Por términos suma:

$$b = (\overline{C} + B + \overline{A}) \,.\, (\overline{C} + \overline{B} + A) =$$
$$= \overline{C} + B.A + \overline{B}.\overline{A} =$$
$$= \text{Nand}(C, B * A, \overline{B} * \overline{A})$$

c	OO	O1	11	1O
OO	**1**	**1**	**1**	**0**
O1	**1**	**1**	**1**	**1**
11	**X**	**X**	**X**	**X**
1O	**1**	**1**	**X**	**X**

Por términos suma:

$$c = (C + \overline{B} + A) =$$
$$= C + \overline{B} + A =$$
$$= \text{Nand}(\overline{C}, B, \overline{A})$$

d	OO	O1	11	1O
OO	**1**	**0**	**1**	**1**
O1	**0**	**1**	**0**	**1**
11	**X**	**X**	**X**	**X**
1O	**1**	**1**	**X**	**X**

Por términos producto:

$$d = \overline{C}.\overline{A} + \overline{C}.B + C.\overline{B}.A + B.\overline{A} + D =$$
$$= D + C.\overline{B}.A + \overline{C}.B + \overline{C}.\overline{A} + B.\overline{A} =$$
$$= \text{Nand}(\overline{D}, \text{Nand}(C, \overline{B}, A), \overline{C} * B, \overline{C} * \overline{A}, B * \overline{A})$$

e	OO	O1	11	1O
OO	**1**	**0**	**0**	**1**
O1	**0**	**0**	**0**	**1**
11	**X**	**X**	**X**	**X**
1O	**1**	**0**	**X**	**X**

Por términos producto:

$$e = \overline{C}.\overline{A} + B.\overline{A} =$$
$$= \overline{C}.\overline{A} + B.\overline{A} =$$
$$= (\overline{C} * \overline{A}) * (B * \overline{A})$$

f	OO	O1	11	1O
OO	**1**	**0**	**0**	**0**
O1	**1**	**1**	**0**	**1**
11	**X**	**X**	**X**	**X**
1O	**1**	**1**	**X**	**X**

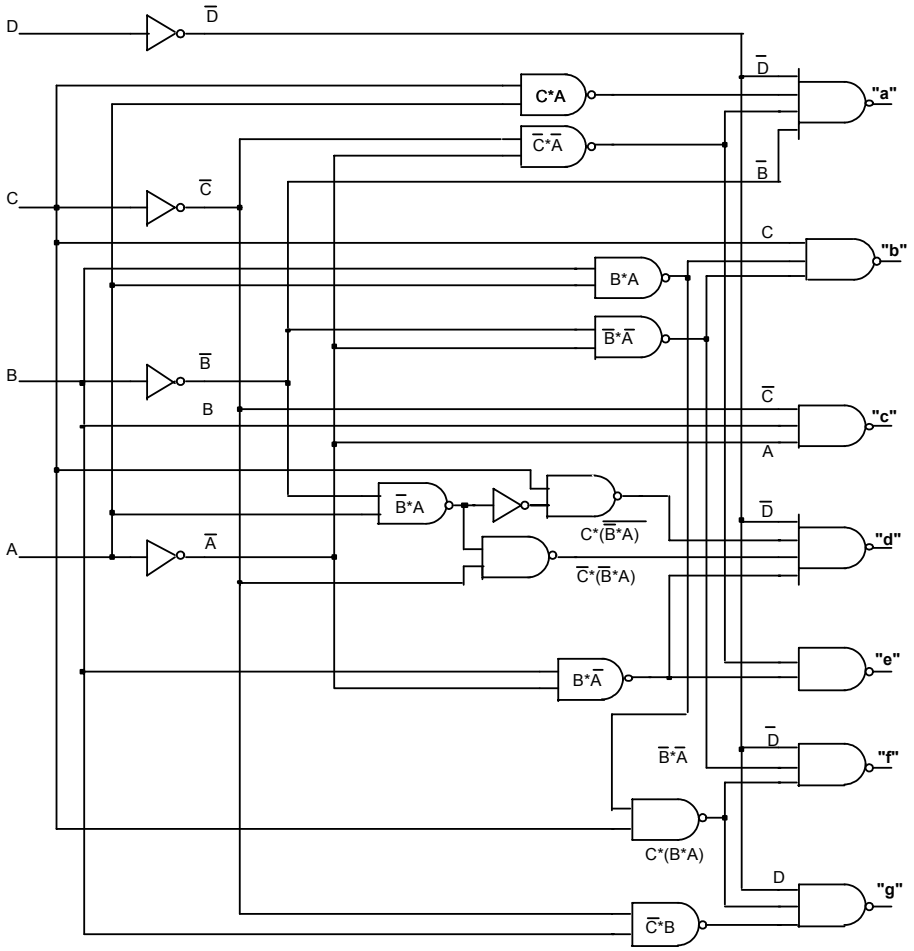
Por términos producto:

$$f = \overline{B}.\overline{A} + C.\overline{B} + C.\overline{A} + D =$$
$$= D + C.\overline{B} + C.\overline{A} + \overline{B}.\overline{A} =$$
$$= \text{Nand}(\overline{D}, C * \overline{B}, C * \overline{A}, \overline{B} * \overline{A})$$

g	OO	O1	11	1O
OO	**0**	**0**	**1**	**1**
O1	**1**	**1**	**0**	**1**
11	**X**	**X**	**X**	**X**
1O	**1**	**1**	**X**	**X**

Por términos producto:

$$g = \overline{C}.B + C.\overline{B} + C.\overline{A} + D =$$
$$= D + C.\overline{B} + \overline{C}.B + C.\overline{A} =$$
$$= \text{Nand}(\overline{D}, C * \overline{B}, \overline{C} * B, C * \overline{A})$$



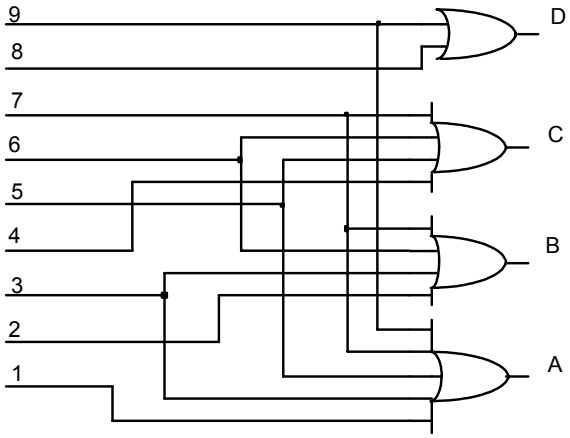
Conversor BCD → 7 segmentos (con puertas *Nand*)

La configuración de la figura utiliza:

- 2 puertas "**y-negada**" de 4 entradas
- 4 puertas "**y-negada**" de 3 entradas
- 11 puertas "**y-negada**" de 2 entradas
- y 5 inversores; en total: 22 puertas booleanas.

4.3.2. Codificador de 9 líneas a BCD (encodificador)

Sean nueve líneas numeradas del 1 al 9, de forma que nunca se activan dos de ellas a la vez (código de «un solo uno»), para convertir dichas 9 entradas en su código BCD correspondiente basta utilizar una puerta "**o**" para cada salida:



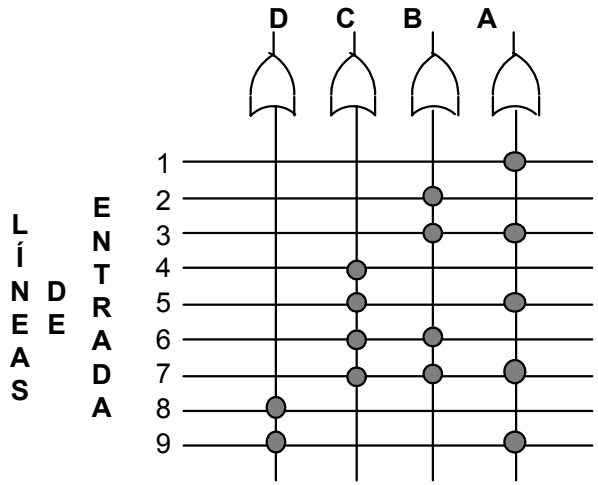
Siempre que el código de entrada es de «un solo uno» el conversor se construye mediante puertas "**o**" (una para cada salida): cada línea de entrada coincide con uno de los códigos de entrada y debe conectar con aquellas puertas "**o**" que deben adoptar valor **1** en el correspondiente código de salida. Por ejemplo, el código de la línea 9 es **1001** y, por tanto, debe incidir en las dos puertas "**o**" de los extremos.

Este tipo de codificador (código de entrada de «un solo uno» - puertas "**o**" de salida) recibe el nombre de *encodificador* (porque llevan las entradas directamente a las puertas de salida).

La tabla funcional de este encodificador es la siguiente:

l_9	l_8	l_7	l_6	l_5	l_4	l_3	l_2	l_1	D	C	B	A
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**1**
0	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**1**	**0**
0	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**1**
0	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
0	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**
0	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**
0	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**
0	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**
1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**

La representación reticular resulta también muy apropiada para los encodificadores:



La matriz de conexiones de un encodificador, en estructura reticular, se corresponde directamente con la tabla funcional del encodificador (con la parte derecha de dicha tabla):

Matriz de conexiones	D	C	B	A	
del encodificador	0	0	0	1	línea 1
de 9 líneas	0	0	1	0	línea 2
(4 salidas)	0	0	1	1	línea 3
	0	1	0	0	línea 4
	0	1	0	1	línea 5
	0	1	1	0	línea 6
	0	1	1	1	línea 7
	1	0	0	0	línea 8
	1	0	0	1	línea 9

De forma que, para dibujar un codificador en configuración reticular, basta establecer la correspondiente retícula líneas de entrada – puertas "**o**" de salida y sobre la misma reflejar la tabla funcional, insertando una conexión cuando el valor de dicha tabla es **1** y dejando sin conexión aquellos cruces que corresponden a valor **0**.

4.3.3. Codificador de prioridad de 9 líneas

Sean nueve líneas numeradas del 1 al 9 como en el caso anterior, pero admitiendo la posibilidad de que se activen varias a la vez: el código de salida será el más alto de los números binarios que corresponden a las líneas que se encuentran activadas. La tabla de conversión simplificada es la siguiente:

l_9	l_8	l_7	l_6	l_5	l_4	l_3	l_2	l_1	D	C	B	A
0	0	0	0	0	0	0	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	1	X	0	0	1	0
0	0	0	0	0	0	1	X	X	0	0	1	1
0	0	0	0	0	1	X	X	X	0	1	0	0
0	0	0	0	1	X	X	X	X	0	1	0	1
0	0	0	1	X	X	X	X	X	0	1	1	0
0	0	1	X	X	X	X	X	X	0	1	1	1
0	1	X	X	X	X	X	X	X	1	0	0	0
1	X	X	X	X	X	X	X	X	1	0	0	1

de la cual resultan las siguientes funciones booleanas (simplificadas por «absorción») :

$$D = l_9 + l_8$$

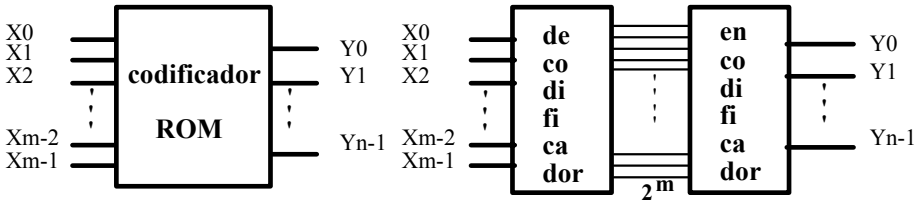
$$C = \overline{l_9}.\overline{l_8}.(l_7 + l_6 + l_5 + l_4)$$

$$B = \overline{l_9}.\overline{l_8}.(l_7 + l_6) + \overline{l_9}.\overline{l_8}.\overline{l_5}.\overline{l_4}.(l_3 + l_2)$$

$$A = l_9 + \overline{l_8}.l_7 + \overline{l_8}.\overline{l_6}.l_5 + \overline{l_8}.\overline{l_6}.\overline{l_4}.l_3 + \overline{l_8}.\overline{l_6}.\overline{l_4}.\overline{l_2}.l_1$$

4.4. Configuración ROM

En general, la tabla funcional correspondiente a un cambio de codificación puede ser muy amplia y la síntesis de las funciones booleanas que corresponden a dicha tabla puede ser sumamente compleja. Sin embargo, la conversión de un código a otro puede resolverse de una manera conceptualmente más sencilla, sin necesidad de construir dichas funciones booleanas, dividiéndolo en dos partes: *decodificación* y *encodificación*.



El primer bloque realiza una *decodificación* completa de los vectores de entrada sobre $\mathbf{2^m}$ líneas intermedias, que adoptarán el código de «un solo uno», propio de las salidas de un decodificador; dichas $\mathbf{2^m}$ líneas intermedias se *encodifican* sobre las **n** líneas de salida.

De hecho, la figura anterior equivale a configurar las **n** funciones booleanas de las salidas en su forma canónica: suma de términos mínimos. Primero, se construyen todos los términos mínimos de las entradas y, luego, cada salida recoge en una puerta "**o**" (suma) los que corresponden a su función, es decir, aquellos cuya columna en la tabla funcional presenta valor **1**.

Los codificadores configurados según este esquema reciben el nombre genérico de **ROM** (*read only memory*, memorias de sólo lectura), nombre confuso que procede de la utilización de estos bloques en los computadores y microprocesadores; sin embargo, estos bloques son claramente combinacionales y no tienen, en absoluto, capacidad de memoria, la cual es propia de los sistemas secuenciales. [Por ello, desde el punto de vista de que son bloques combinacionales y a tenor de su estructura funcional, propongo interpretar las siglas **ROM** como **Row Ordered Minterms**: filas de términos mínimos.]

La configuración **ROM** contiene dos partes diferenciadas:

- un *decodificador*, cuya realización solamente depende del número **m** de entradas, pero es independiente del codificador concreto de que se trate
- y un *encodificador*, constituido por **n** puertas "**o**", cada una de las cuales recibe parte de las líneas de salida del decodificador.

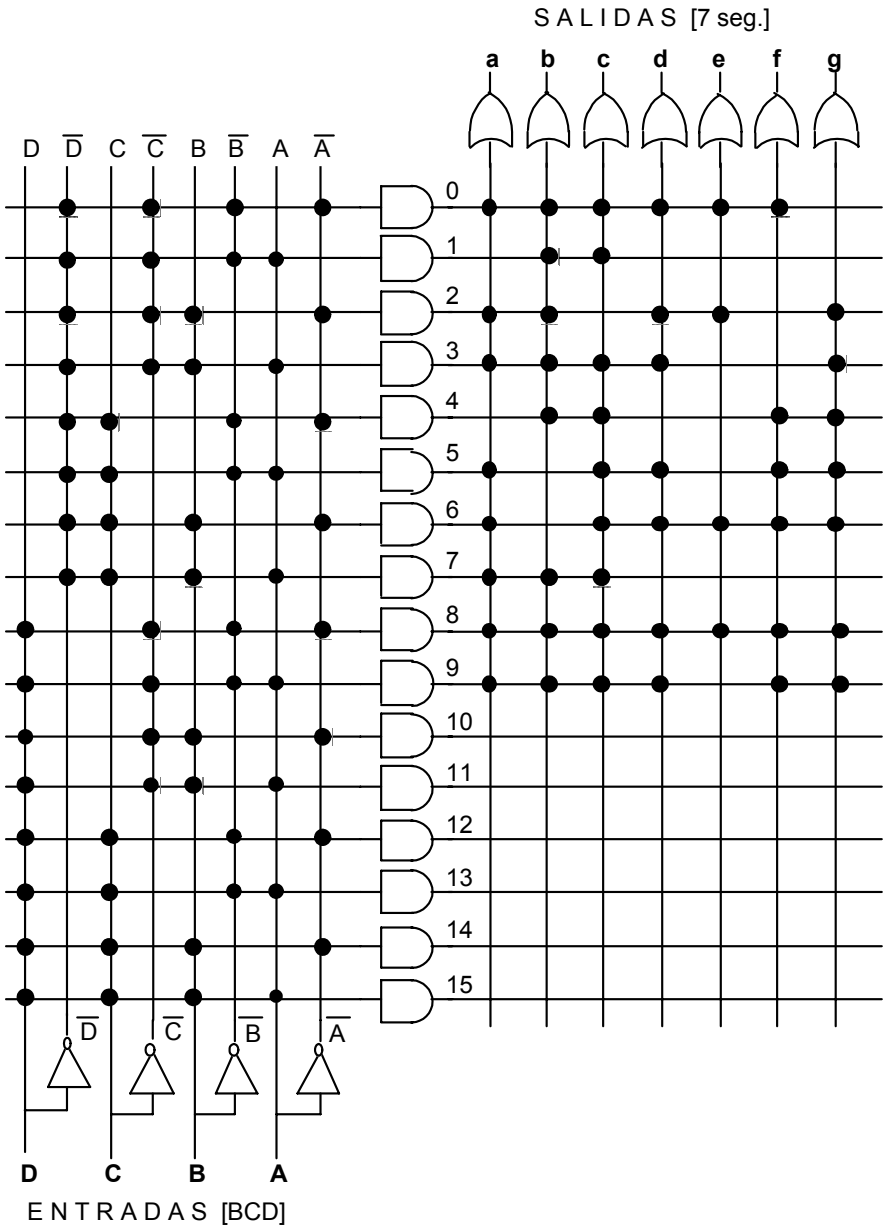
Los subconjuntos de líneas de salida del decodificador conectadas a cada una de las puertas "**o**" del encodificador personalizan el codificador, es decir, son lo único que varía entre dos codificadores diferentes de igual número de entradas y salidas.

En términos de puertas booleanas tal estructura presenta gran simplicidad y regularidad; está constituida modularmente por dos conjuntos de puertas:

- el primero de puertas "**y**" ($\mathbf{2^m}$ puertas "**y**" de **m** entradas que forman el decodif.)
- seguido de un conjunto de puertas "**o**" (**n** puertas "**o**" que forman el encodif.).

Para ambos bloques, la configuración reticular resulta muy apropiada; la matriz de conexiones del encodificador coincide con la tabla funcional (con la parte de la derecha, de asignación de valores a las salidas) de la conversión entre los dos códigos.

Por ejemplo, en el caso del conversor de BCD a 7 segmentos, su configuración ROM (en forma reticular) es la representada en la figura de la página siguiente. [Compruébese en ella que la matriz de conexiones del encodificador es idéntica a la tabla funcional, página 96.]



Codificador ROM BCD → 7 segmentos

Ahora bien, cuando el número de entradas del codificador es pequeño (como sucede en el caso anterior) la configuración **ROM** de un codificador no supone una ventaja apreciable respecto a la obtención de sus funciones booleanas simplificadas.

La estructura **ROM** es realmente apropiada cuando el número de entradas es alto (por encima de las 6 entradas); en tal caso, la configuración **ROM** puede ser construida directamente desde la tabla funcional o tabla de conversión, evitando la necesidad de extraer de ella y de simplificar las funciones booleanas. Dicha construcción consiste en:

- configurar un decodificador genérico de **m** entradas, o sea, construir reticularmente los $\mathbf{2^m}$ términos mínimos,
- seguido de un encodificador, cuya matriz de conexiones ha de ser la propia tabla de conversión.

Dos ejemplos concretos de cambios de código con un número amplio de entradas los constituyen las *tablas funcionales* y los *generadores de caracteres*.

Podemos considerar un codificador **ROM** como una tabla de valores que para cada valor numérico de entrada nos proporciona el resultado de una determinada función aplicada a dicho número; de esta forma los codificadores **ROM** permiten configurar tablas reducidas de funciones, *tablas funcionales*, como, por ejemplo, tablas trigonométricas (senos, cosenos, tangentes,...), tablas de logaritmos, etc.

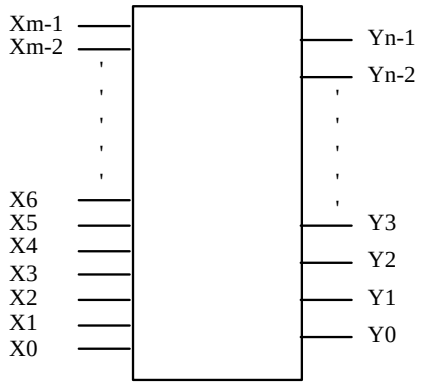
Los *generadores de caracteres* son codificadores que convierten los caracteres alfanuméricos codificados en binario (por ejemplo, en código **ASCII** de 7 bits) en matrices rectangulares de puntos que representan visualmente dichos caracteres. La visualización final puede realizarse bien mediante diodos luminiscentes **LED** o lámparas, bien en monitores o pantallas de rayos catódicos **CRT**, bien sobre papel mediante impresoras de puntos, etc. Los puntos de una matriz **n x m** se activarán (iluminación o impresión) cuando se encuentren con valor **1** y no lo harán si es **0**.

Las pequeñas mini-impresoras por puntos suelen imprimir en columnas de 7 puntos, utilizando 5 columnas para cada letra o número y 2 de separación. El codificador **ROM** para la correspondiente generación de caracteres ha de tener como entradas los 7 dígitos del carácter **ASCII** más 3 dígitos para indicar el número de columna a imprimir y presentará como salidas los 7 bits que van a controlar los solenoides que activan los bastones de impresión.

En cambio, en el barrido de la pantalla de un monitor (**CRT**) las letras y números se dibujan por filas, de forma que el generador de caracteres ha de tener como entradas los 7 dígitos de cada carácter más 3 ó 4 dígitos para indicar el número de la fila y las salidas son los 5 ó 7 bits correspondientes a los puntos que conforman una fila del carácter, caso de representarlos en matrices de 7 x 5 puntos o de 9 x 7 puntos, respectivamente.

4.5. Sistemas combinacionales; diseño modular

Entendemos por sistema lógico o sistema booleano aquel que puede describirse en términos del álgebra de Boole de 2 elementos. En principio, dicho sistema tendrá **m** variables o entradas, cuyos valores booleanos configuran un *vector de entrada* de dimensión **m**, y **n** resultados o salidas, cuyos valores booleanos configuran un *vector de salida* de dimensión **n**.



Un sistema lógico es *combinacional* si a cada *vector de entrada* le corresponde un solo *vector de salida*, es decir, si la salida es única (aplicación unívoca del conjunto de vectores de entrada en el conjunto de vectores de salida): para un vector de entrada dado no existe más de un vector de salida.

La correspondencia unívoca, es decir, el que a un vector de entrada dado le corresponda un solo vector de salida puede parecer obvia y, sin embargo, no lo es: en el capítulo 11, al introducir los sistemas secuenciales (sistemas digitales que poseen «memoria»), se apreciará que para ellos la aplicación vectores de entrada → vectores de salida no es unívoca.

Cada variable de salida **y** es una función booleana de las **m** variables de entrada $\mathbf{x_i}$; tal función puede expresarse mediante su tabla funcional o *tabla de verdad*, haciendo explícito el valor booleano que adopta la variable de salida para cada uno de los $\mathbf{2^m}$ vectores de entrada. En tal sentido, el nombre de *combinacional* proviene de que las salidas o respuestas del sistema son función booleana de sus entradas, es decir, son «combinación» mediante operaciones booleanas de las variables de entrada.

Es posible que el vector de salida correspondiente a uno de entrada sea indeterminado o indiferente (*no importa cual sea*). Esto sucede, por ejemplo, en muchos diseños reales cuando un determinado vector de entrada no se presenta nunca en la práctica o cuando no importa qué respuesta produzca el sistema para un determinado vector de entrada (porque no hay exigencias respecto a tal respuesta y resulte indiferente que sea una u otra). La indeterminación puede referirse a todas las variables de salida o solamente a algunas de ellas: indeterminación parcial.

En los casos de indeterminación se asigna inicialmente a las correspondientes variables de salida el valor **X**, que puede ser **1** ó **0** según interese; esta doble posibilidad se utiliza para simplificar las funciones booleanas resultantes.

Así pues, en un sistema lógico combinacional a un vector de entrada le corresponde uno y sólo un vector de salida; o bien no le corresponde inicialmente ninguno (la salida es indeterminada) y en el proceso de síntesis se le asigna un vector de salida cuyas componentes se eligen según interese.

El análisis de un sistema combinacional pasa por obtener su *tabla de operación* o conjunto de *tablas de verdad*, expresando para cada vector de entrada el vector de salida que le corresponde.

La síntesis de sistemas combinacionales consiste en construir, a partir de las especificaciones o requisitos que se persiguen, la *tabla funcional* que relaciona los $\mathbf{2^m}$ vectores de entrada posibles con los vectores de salida que deben producir; a partir de dicha tabla se obtienen las correspondientes funciones booleanas que permiten construir el sistema por combinación adecuada de puertas lógicas.

En general, en todo proceso de síntesis resulta sumamente útil la aplicación de estrategias de división modular: cuando un sistema combinacional es de una cierta complejidad es necesario descomponerlo en subsistemas y realizar su síntesis por separado. En tal sentido, existe una serie de subsistemas conceptuales que aparecen con mucha frecuencia y es sumamente útil conocerlos; además, la alta densidad de integración que ofrecen los circuitos integrados ha permitido que dichos subsistemas se encuentren disponibles como circuitos integrados.

Tales subsistemas reciben el nombre de *Bloques Combinacionales*; en los dos últimos capítulos hemos estudiado diversos bloques de utilidad general, que pueden clasificarse en tres grupos:

- *Bloques operacionales aritméticos y lógicos*, capaces de efectuar una operación aritmética entre dos números binarios de **m** dígitos o de efectuar una operación lógica (bit a bit) entre dos palabras binarias de longitud **m**.
- *Bloques distribuidores: multiplexores, demultiplexores y decodificadores*, dedicados a organizar y seleccionar las líneas de transmisión de la información.
- *Bloques codificadores o conversores de código,* que realizan la conversión entre códigos, o sea, entre palabras binarias que expresan la misma información.

El interés de los bloques combinacionales no se limita a su disponibilidad en forma de circuitos integrados; incluso la utilización de tales circuitos integrados estándar es, cada vez, menos habitual pues interesa configurar todo el diseño digital en un solo integrado, bien sea éste programable o construido a medida (ASIC).

Los bloques combinacionales interesan por sí mismos, como módulos conceptuales o «piezas de diseño» que facilitan estrategias de división en partes (diagramas de bloques); tal «partición» es imprescindible para abordar el diseño de sistemas complejos.

El tiempo de propagación en los circuitos combinacionales

"El tiempo es un compañero inevitable" en el funcionamiento de los circuitos digitales, al igual que en cualquier otro sistema físico real. "Nada es instantáneo" en la respuesta de los sistemas físicos en relación con su excitación, con las entradas que, al incidir sobre el sistema, causan la respuesta del mismo: la relación causa - efecto requiere un cierto tiempo de ejecución: *estímulo* $\rightarrow \Delta t \rightarrow$ *respuesta*.

En un circuito combinacional se produce un tiempo de retardo entre el vector de entrada y el correspondiente vector de salida, intervalo temporal al que denominamos tiempo de propagación $\mathbf{t_p}$. En la práctica el tiempo de propagación será diferente para cada una de las salidas del circuito combinacional; habida cuenta de que resulta necesario esperar a la más lenta de ellas para obtener la respuesta global, el tiempo de propagación del circuito será el mayor de los tiempos de retraso de sus salidas.

Los tiempos de propagación dependen linealmente del *fan-out* que soportan las puertas, es decir, de la carga capacitiva conectada a su salida; además, aumentan fuertemente con la temperatura. Un circuito electrónico se calienta por la disipación de la energía que consume en su propio funcionamiento que se elimina en forma de calor, con la consiguiente elevación de la temperatura del circuito y disminución de la velocidad de trabajo del mismo; en muchos casos será necesario incluir una refrigeración adecuada para garantizar la frecuencia de trabajo del circuito.

La conexión sucesiva de puertas booleanas implica la correspondiente adición de sus tiempos de propagación: el tiempo de retardo del conjunto será del orden de la suma de los tiempos individuales $\mathbf{\Sigma t_p}$. Si bien dicha suma expresa solamente una cota superior del tiempo de propagación global, ya que los tiempos de propagación de las sucesivas puertas se solapan parcialmente: una puerta no espera para iniciar su conmutación a que la anterior haya completado la suya.

Para valorar, en una primera aproximación, el tiempo de propagación de un bloque combinacional, denominaremos índice de propagación $\mathbf{i_p}$ al número de puertas sucesivas que se presentan entre una salida y una entrada, o bien, al mayor de los números de puertas sucesivas que contiene un circuito combinacional. El índice de propagación proporciona una referencia, al menos cualitativa, del tiempo de propagación global.

Las salidas de un bloque combinacional corresponden a funciones booleanas de las entradas del mismo y, en principio, cualquier función booleana puede expresarse como suma de productos; tales sumas de productos se construyen en la práctica mediante dos niveles de puertas "**y-negada**", *Nand* (*Nand de Nands*), junto con un tercer nivel de inversores para las variables de entrada negadas, de forma que el índice de propagación de una función booleana construida como suma de productos es $\mathbf{i_p = 3}$.

Como veremos en el próximo capítulo (cap. V), cualquier conjunto de funciones booleanas, en particular cuando el número de sus entradas es relativamente alto, puede construirse mediante un esquema: [matriz Y de entradas] [matriz O de salidas]; en la práctica (precisamente para admitir múltiples entradas), este esquema se construye con puertas "**o-negada**", *Nor seudoNMOS,* y requiere cuatro niveles de puertas:

[inversores de las entradas] [matriz NOR de entradas]
[matriz NOR de salidas] [inversores de las salidas]

lo cual supone un índice de propagación $\mathbf{i_p = 4}$.

Así pues, el índice de propagación necesario para construir cualquier sistema combinacional es 4: según este esquema booleano ningún sistema combinacional requiere un índice superior a 4 y, siempre que ello sea necesario por razones de velocidad, cualquier sistema combinacional puede construirse con los 4 niveles antes detallados.

Ahora bien, hay casos de bloques combinacionales en que la construcción directa de cada una de sus salidas presenta alta complejidad y considerable área de integración, porque tales circuitos se adaptan mejor a un esquema recursivo de celdas en cascada; sistemas combinacionales que, en lugar de ajustarse fácilmente a una configuración en paralelo de sus salidas, se acomodan conceptualmente a un cálculo sucesivo (en serie) de las mismas. Por ejemplo, los sumadores y restadores de **n** bits, cuya construcción es muy simple a partir de celdas de 1 bit, conectadas en cascada.

En estos casos de diseño celular recursivo (conexión de pequeños módulos en serie), el índice de propagación puede hacerse relativamente alto y, consiguientemente, el tiempo de propagación global del circuito.

El caso más típico y habitual lo constituyen los sumadores de **n** bits, conformados por **n** celdas sumadoras de 1 bit; en ellos el arrastre (*carry*) ha de propagarse sucesivamente a través de las n celdas: propagación por onda *(ripple carry*):

$$C_{i+1} = a_i.b_i + (a_i + b_i).C_i = \overline{(a_i * b_i) . \left[(a_i \Delta b_i) + \overline{C_i}\right]}$$

El índice de propagación del arrastre en una celda básica de 1 bit es $\mathbf{i_p = 2}$; un sumador de 64 bits presentará un índice de propagación muy alto $\mathbf{i_p = 128}$.

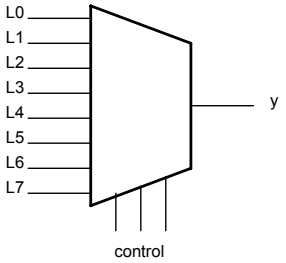
Los esquemas celulares en cascada presentan altos índices de propagación y, por ello, introducen limitaciones fuertes respecto a la frecuencia de trabajo. Cuando interesan velocidades mayores, la forma de conseguirlo se basa en disminuir el número de módulos en cascada, aumentando la «granularidad» de cada módulo; es decir, en lugar de utilizar celdas básicas de 1 bit (granularidad = 1), emplear módulos de **p** bits (granularidad = p), construyendo directamente (en paralelo) las funciones booleanas de los mismos.

Por ejemplo, en el caso de sumadores, si en lugar de celdas básicas de 1 bit se utilizan módulos sumadores de 4 bits el índice de propagación se reduce en un factor 4. Más aún, el apéndice **A2** (referido a la "Propagación rápida de acarreo en los sumadores") muestra cómo la recursividad puede ser aprovechada para conseguir reducciones del índice de propagación aún mayores.

4.5. Los bloques combinacionales en VHDL

Al igual que en el capítulo 1, se incluye aquí la descripción de algunos bloques combinacionales en VHDL a fin de desarrollar una aproximación gradual al diseño con lenguajes de descripción circuital.

Multiplexor de 8 líneas




```
control  : INTEGER range 0 to 7
```

versión 1

```
y <=   L0  when  control = 0  else
       L1  when  control = 1  else
       L2  when  control = 2  else
       L3  when  control = 3  else
       L4  when  control = 4  else
       L5  when  control = 5  else
       L6  when  control = 6  else  L7;
```

versión 2

```
with  control  select
      y <=   L0  when  0,
             L1  when  1,
             L2  when  2,
             L3  when  3,
             L4  when  4,
             L5  when  5,
             L6  when  6,
             L7  when  others;
```

descripción utilizando proceso

```
process (control,L7,L6,L5,L4,L3,L2,L1,L0)
begin
```

versión 1

```
if control = 0  then   y  <= L0;   end if;
if control = 1  then   y  <= L1;   end if;
if control = 2  then   y  <= L2;   end if;
if control = 3  then   y  <= L3;   end if;
if control = 4  then   y  <= L4;   end if;
if control = 5  then   y  <= L5;   end if;
if control = 6  then   y  <= L6;   end if;
if control = 7  then   y  <= L7;   end if;
end process;
```

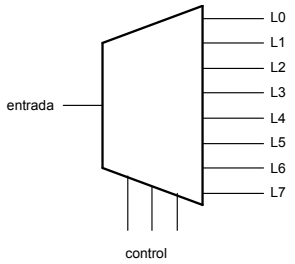
versión 2

```
case control is
      when 0  =>   y <= L0;
      when 1  =>   y <= L1;
      when 2  =>   y <= L2;
      when 3  =>   y <= L3;
      when 4  =>   y <= L5;
      when 5  =>   y <= L5;
      when 6  =>   y <= L6;
      when 7  =>   y <= L7;
end case;
end process;
```


Demultiplexor de 8 líneas

```
control  : INTEGER range 0 to 7

L0 <= entrada when control = 0      else '0';
L1<=  entrada when control = 1      else '0';
L2 <= entrada when control = 2      else '0';
L3 <= entrada when control = 3      else '0';
L4 <= entrada when control = 4      else '0';
L5 <= entrada when control = 5      else '0';
L6 <= entrada when control = 6      else '0';
L7 <= entrada when control = 7      else '0';
```

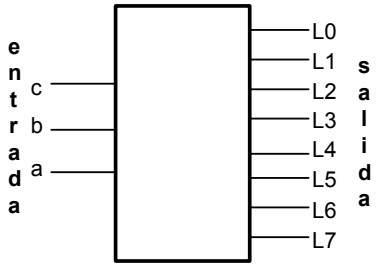


utilizando proceso

```
process (control,entrada)
begin
 L0 <= '0'; L1 <= '0'; L2 <= '0'; L3 <= '0';
 L4 <= '0'; L5 <= '0'; L6 <= '0'; L7 <= '0';
case control is
                when 0  =>   L0 <= entrada;
                when 1  =>   L1 <= entrada;
                when 2  =>   L2 <= entrada;
                when 3  =>   L3 <= entrada;
                when 4  =>   L4 <= entrada;
                when 5  =>   L5 <= entrada;
                when 6  =>   L6 <= entrada;
                when 7  =>   L7 <= entrada;
end case;
end process;
```

Decodificador de 8 líneas

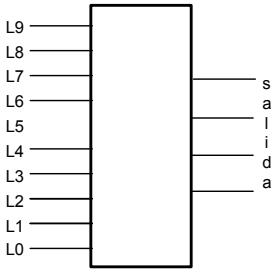
```
with entrada select
salida <=     "10000000" when "000",
              "01000000" when "001",
              "00100000" when "010",
              "00010000" when "011",
              "00001000" when "100",
              "00000100" when "101",
              "00000010" when "110",
              "00000001" when "111";
```



Encodificador de 10 líneas

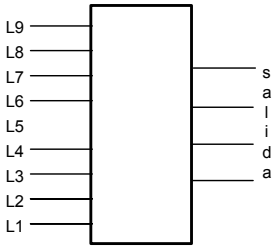
```
entrada <= L9 & L8 & L7 & L6 & L5 & L4 & L3 & L2 & L1 & L0;

with entrada select
        salida <= "1001" when "1000000000",
                  "1000" when "0100000000",
                  "0111" when "0010000000",
                  "0110" when "0001000000",
                  "0101" when "0000100000",
                  "0100" when "0000010000",
                  "0011" when "0000001000",
                  "0010" when "0000000100",
                  "0001" when "0000000010",
                  "0000" when "0000000001",
                  "1111" when others;
```



Codificador de prioridad de 9 líneas.

```
salida <=   "1001" when L9 = '1' else
            "1000" when L8 = '1' else
            "0111" when L7 = '1' else
            "0110" when L6 = '1' else
            "0101" when L5 = '1' else
            "0100" when L4 = '1' else
            "0011" when L3 = '1' else
            "0010" when L2 = '1' else
            "0001" when L1 = '1' else
            "0000";
```



utilizando proceso

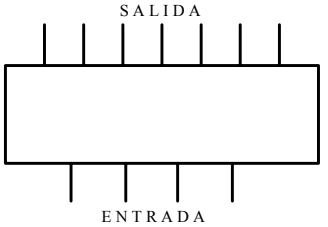
```
process (L9,L8,L7,L6,L5,L4,L3,L2,L1)
begin
if L9 = '1' then            salida <= "1001";
    elsif L8 = '1' then     salida <= "1000"
    elsif L7 = '1' then     salida <= "0111";
    elsif L6 = '1' then     salida <= "0110";
    elsif L5 = '1' then     salida <= "0101";
    elsif L4 = '1' then     salida <= "0100";
    elsif L3 = '1' then     salida <= "0011";
    elsif L2 = '1' then     salida <= "0010";
    elsif L1 = '1' then     salida <= "0001";
    else                    salida <= "0000";
end if;          end process;
```


Conversor BCD a 7 segmentos, ánodo común.

[En un visualizador de ánodo común se «encienden» aquellos segmentos que reciben **0**, ya que la línea de alimentación común a todos ellos corresponde a la tensión positiva Vcc es decir, al **1** booleano.]

```
SALIDA      : Out   BIT_VECTOR(1 to 7);

ENTRADA : In    INTEGER range 0 to 15;
```




```
with entrada select
        SALIDA <=   "0000001" when 0,
                    "1001111" when 1,
                    "0010010" when 2,
                    "0000110" when 3,
                    "1001100" when 4,
                    "0100100" when 5,
                    "0100000" when 6,
                    "0001111" when 7,
                    "0000000" when 8,
                    "0000100" when 9,
                    "1111111" when others;
```

descripción utilizando proceso

```
process (ENTRADA)
begin
case ENTRADA is
            when 0 =>           SALIDA <="0000001";
            when 1 =>           SALIDA <="1001111";
            when 2 =>           SALIDA <="0010010";
            when 3 =>           SALIDA <="0000110";
            when 4 =>           SALIDA <="1001100";
            when 5 =>           SALIDA <="0100100";
            when 6 =>           SALIDA <="0100000";
            when 7 =>           SALIDA <="0001111";
            when 8 =>           SALIDA <="0000000";
            when 9 =>           SALIDA <="0000100";
            when others =>      SALIDA <="1111111";
end case;       end process;
```