5 CONJUNTOS DE FUNCIONES BOOLEANAS ESTRUCTURAS MATRICIALES Y BLOQUES PROGRAMABLES

El presente tema trata de dar respuesta a la pregunta: ¿Cómo construir n funciones booleanas de m variables cuando tal número de variables de entrada es un poco alto? O lo que es lo mismo: ¿Cómo manejar conjuntos de varias funciones de muchas variables? En tales casos, el alto número de entradas supone una dificultad importante a la hora de obtener las expresiones algebraicas de las funciones y, también, a la hora de transformar dichas expresiones en circuitos con puertas lógicas.

Lo grande, lo complejo requiere, para ser manejable, de un plus de organización, de una «estructura» que oriente y facilite su manejo. «Estructura = distribución, orden y enlace de las partes para formar el todo.» A lo largo de este tema se muestra cómo la estructura matricial (que se representa mediante una configuración reticular) permite abordar con relativa facilidad conjuntos de funciones de muchas variables.

Además, tal tipo de organización ha permitido construir bloques integrados «programables», en cuyo interior es posible configurar las funciones booleanas por medio de la eliminación de conexiones (fusibles). De manera que disponemos de circuitos integrados programables que constituyen, actualmente, la forma eficaz y habitual de construir los prototipos de sistemas digitales o de fabricar series de pocas unidades.

Se utilizan tres tipos de estructura matricial (ROM, PAL y PLA) que son útiles en cuanto a modelos o esquemas conceptuales para la realización de conjuntos de funciones booleanas y, también, como configuraciones físicas concretas de los circuitos integrados programables.

Los codificadores ROM (ya estudiados en el capítulo anterior) expresan las funciones booleanas en su forma canónica, como suma de términos mínimos, diferenciando el decodificador (los términos mínimos, comunes a las diversas funciones) del encodificador (la suma de los mismos, que identifica a cada una de ellas).

La estructura PAL se corresponde con la forma habitual de expresar las funciones booleanas como suma de productos, una vez simplificada su expresión algebraica; cada función se construye por separado, con sus propios términos producto.

Por último, la configuración PLA es también del tipo suma de productos, pero los términos producto son compartidos por las diversas funciones, lo cual permite minimizar el número de términos producto necesarios (a través de una simplificación multifunción).

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

5.1. Formas diversas de construir una función booleana

Las funciones booleanas son multiformes; pueden representarse y expresarse en formas diversas:

enunciado → tabla funcional → forma canónica → expresión algebraica simplificada

De igual manera, una función booleana puede construirse en formas diversas:

- con un multiplexor cuyas entradas reciben los valores de la tabla funcional: estructura LUT
- con un decodificador, recogiendo sobre una puerta "**o**" los términos mínimos que corresponden a vectores de entrada que dan valor **1** en la tabla funcional: forma canónica ($\sum$ m)
- con puertas básicas, una vez simplificada la expresión algebraica de la función: suma de productos ($\sum$ p)
- con puertas unitarias (en concreto, con puertas *Nand* o con puertas *Nor*), utilizando un solo tipo de puertas
- …

Veamos estas cuatro formas diferentes de construir una función booleana, aplicándolas a una función concreta de 4 variables:

Enunciado: De entre los números binarios de 4 dígitos **dcba** (del 0 al 15 decimales) se excluyen los números 3, 5, 10, 11, 12 y 14, y con el resto se forma el subconjunto **R**; la función "**y**" indicará la pertenencia (**y = 1**) de un número binario de 4 dígitos al subconjunto **R**.

A continuación, se representa esta función en las cuatro configuraciones citadas:

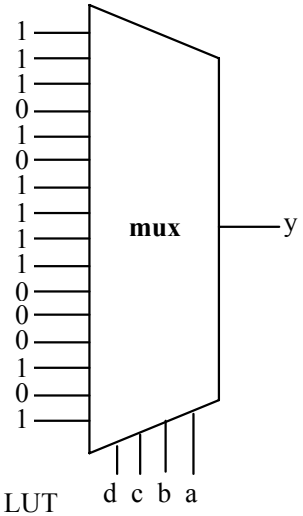
a) Un multiplexor de 4 líneas de control cuyas entradas reciben los valores booleanos de la «tabla de verdad» (o tabla funcional) de la función.

Esta manera de configurar funciones booleanas, con multiplexores que «muestrean» la tabla de la función, se denomina **look-up-table** (LUT): construcción tabular (*mirar sobre su tabla).* [Ver figura en la página siguiente.]

Si bien esta forma de construir funciones no corresponde a las estructuras matriciales consideradas en el presente tema, es utilizada en algunos tipos de circuitos integrados programables (FPGAs), ya que permite la programación de funciones booleanas sin más que almacenar en un registro su tabla funcional.

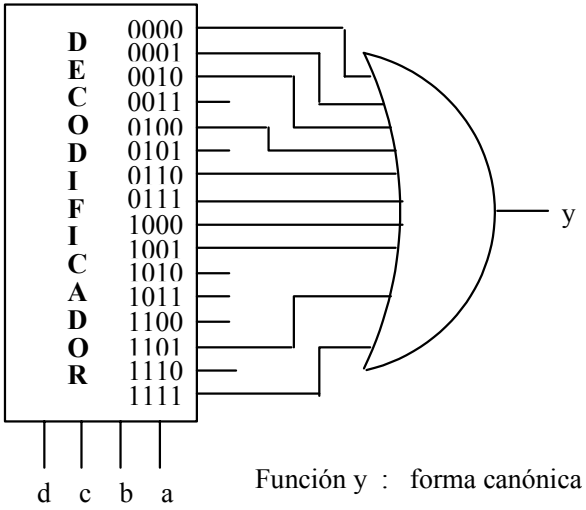
Tabla funcional

d c b a	y
0 0 0 0	1
0 0 0 1	1
0 0 1 0	1
0 0 1 1	0
0 1 0 0	1
0 1 0 1	0
0 1 1 0	1
0 1 1 1	1
1 0 0 0	1
1 0 0 1	1
1 0 1 0	0
1 0 1 1	0
1 1 0 0	0
1 1 0 1	1
1 1 1 0	0
1 1 1 1	1



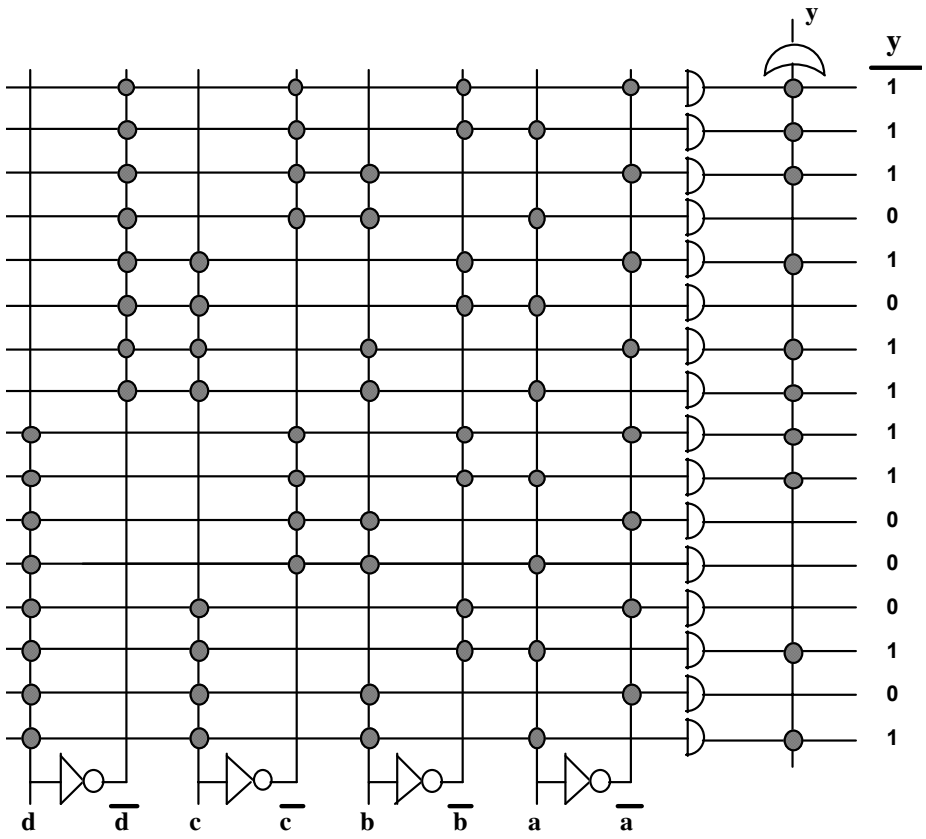
Función y : configuración LUT

b) Un decodificador de 4 líneas de control seguido de una puerta "**o**" que recibe aquellas salidas que corresponden a valor **1** en la «tabla de verdad» de la función; no es sino la construcción directa de la forma canónica de la función y equivale a la configuración **ROM** introducida en el capítulo anterior.



Función y : forma canónica

La siguiente figura muestra la representación reticular de esta configuración: un decodificador genérico de 4 variables, más una puerta "**o**" que recoge los términos mínimos con valor **1** en la tabla funcional. De forma que la línea de conexiones de dicha puerta "**o**" coincide con la columna de valores de salida de la tabla funcional (entendiendo el valor **1** como conexión y el **0** como ausencia de ella).



Las dos formas anteriores de construir la función (LUT y forma canónica) no precisan de la expresión algebraica de la función booleana y menos aún de su simplificación; se obtienen directamente de su tabla funcional.

En cambio, las siguientes formas de construir la función "**y**" corresponden a expresiones algebraicas de la misma; pasemos a obtenerlas, en forma simplificada, empleando su mapa de Karnaugh.

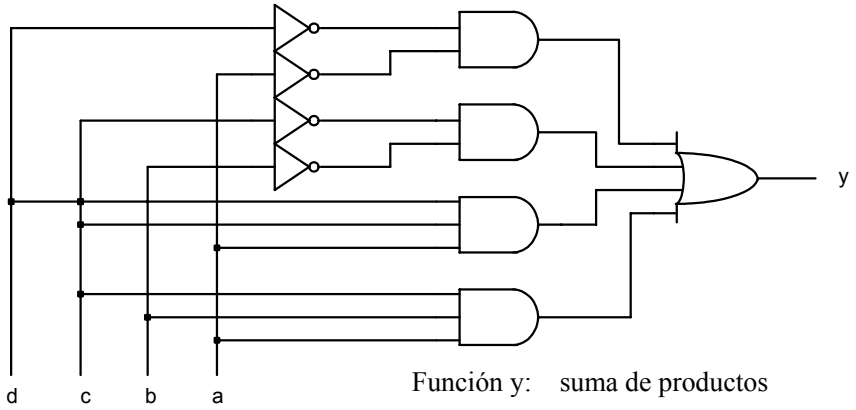
D C \ B A	*0 0*	*0 1*	*1 1*	*1 0*
0 0	**1**	**1**	**0**	**1**
0 1	**1**	**0**	**1**	**1**
1 1	**0**	**1**	**1**	**0**
1 0	**1**	**1**	**0**	**0**

$y = \bar{d}.\bar{a} + \bar{c}.\bar{b} + d.c.a + c.b.a \quad \rightarrow \mathbf{\Sigma p}$

$= \bar{d}.\bar{a} + \bar{c}.\bar{b} + c.a.(d+b)$

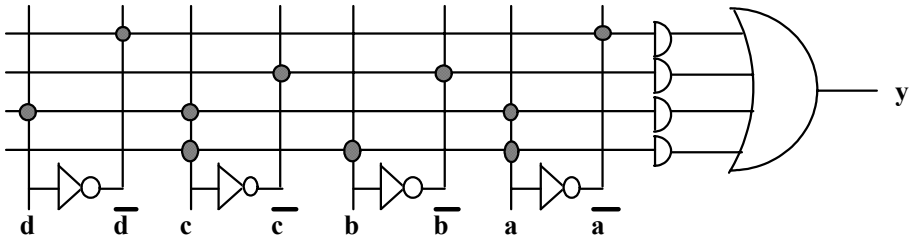
$= \text{Nand}[\bar{d} * \bar{a}, \bar{b} * \bar{c}, \text{Nand}(c, a, \bar{d} * \bar{b})] \quad \rightarrow \mathbf{Nand}$

c) La conexión de puertas básicas (inversores, puertas "**o**" y puertas "**y**") que corresponde a la expresión algebraica de la función como suma de productos, una vez simplificada ($\sum \mathbf{p}$). Es la forma habitual de expresar una función y dará lugar a la estructura matricial que denominaremos **PAL**.

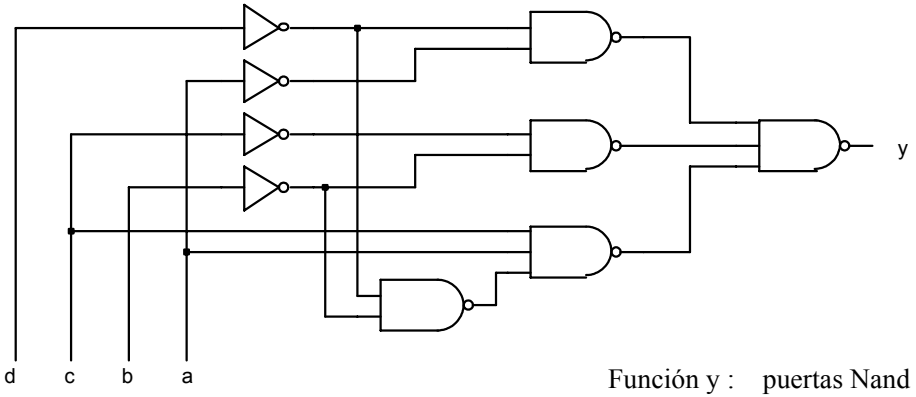


Función y: suma de productos

Podemos representar la misma expresión algebraica en configuración reticular en la forma siguiente:



d) La expresión de la función trasladada a puertas "**y-negada**" (*Nand*) para utilizar un solo tipo de puertas. Es la forma típica de construir una función con circuitos integrados estándar: el uso de un solo tipo de puertas para minimizar el número de circuitos integrados necesarios.



Función y : puertas Nand

La utilización de circuitos integrados estándar, aun cuando puede resultar útil en laboratorio (sobre todo para funciones no muy complejas) está siendo sustituida en la práctica por el empleo de circuitos integrados programables, los cuales permiten insertar en el interior de un solo circuito el conjunto completo de funciones booleanas necesarias, evitando el cableado entre las diversas puertas lógicas.

Respecto al presente tema nos interesan, en particular, las formas **b)** y **c)** de construir una función, es decir su forma canónica a partir de un decodificador (suma de términos mínimos $\sum \mathbf{m}$) y su forma algebraica como suma de productos $\sum \mathbf{p}$, que dan lugar, respectivamente, a las configuraciones **ROM** y **PAL**. Consideraremos también un tercer tipo de configuración, **PLA** (suma de productos compartidos) que solamente tiene sentido cuando tratamos con conjuntos de funciones y no para funciones individuales.

Las siglas utilizadas para estas configuraciones proceden de denominaciones que aportan poco (e incluso son confusas) en relación con la estructura a la que dan nombre: **ROM** (*read only memory* –memorias de sólo lectura–), **PAL** (*programmable and logic* –lógica de puertas "**y**" programables–) y **PLA** (*programmable logic array* –matriz lógica programable–).

Por ello, a fin de hacer una referencia más directa a la propia estructura de los bloques, propongo una lectura diferente de dichas siglas: **ROM** (*row ordered minterms* –filas de términos mínimos–), **PAL** (*product adding layers* –estratos o módulos sumadores de productos–) y **PLA** (*product linking adders* –sumadores que enlazan o comparten productos–).

5.2. Conjuntos de funciones booleanas: estructuras PAL y ROM

Una advertencia previa: para facilitar la comprensión de los conceptos y, también, por razones obvias de facilidad de realización y de comprensión de los dibujos, en el desarrollo de este capítulo se utilizan como ejemplos funciones de reducido número de variables; ahora bien, debe tenerse en cuenta que las estructuras que estamos estudiando son realmente interesantes y útiles para funciones de amplio número de entradas.

La expresión algebraica más habitual de una función booleana, una vez simplificada, es la forma de suma de términos producto $\sum \mathbf{p}$. El circuito correspondiente a tal expresión es un módulo conformado por una puerta "**o**", que proporciona la salida de la función, conectada a varias puertas "**y**", que realizan los diversos términos producto. Las conexiones de las entradas (y de sus negadas) sobre las puertas "**y**" pueden ser representadas gráficamente en forma reticular o matricial.

Veámoslo con el siguiente ejemplo referido a un conjunto de cuatro funciones de cuatro entradas:

Tabla funcional

d c b a	Y1	Y2	Y3	Y4
0 0 0 0	0	0	1	0
0 0 0 1	1	1	1	1
0 0 1 0	0	0	0	0
0 0 1 1	1	1	0	1
0 1 0 0	1	0	1	1
0 1 0 1	1	0	1	1
0 1 1 0	1	1	0	0
0 1 1 1	1	1	0	0
1 0 0 0	0	1	1	1
1 0 0 1	0	1	1	1
1 0 1 0	0	0	0	0
1 0 1 1	1	0	1	1
1 1 0 0	1	0	0	1
1 1 0 1	1	0	0	1
1 1 1 0	1	1	0	0
1 1 1 1	1	1	1	1

Mapas de Karnaugh

Y1

dc \ ba	00	01	11	10
00	0	1	1	0
01	1	1	1	1
11	1	1	1	1
10	0	0	1	0

Y2

dc \ ba	00	01	11	10
00	0	1	1	0
01	0	0	1	1
11	0	0	1	1
10	1	1	0	0

Y3

dc \ ba	00	01	11	10
00	1	1	0	0
01	1	1	0	0
11	0	0	1	0
10	1	1	1	0

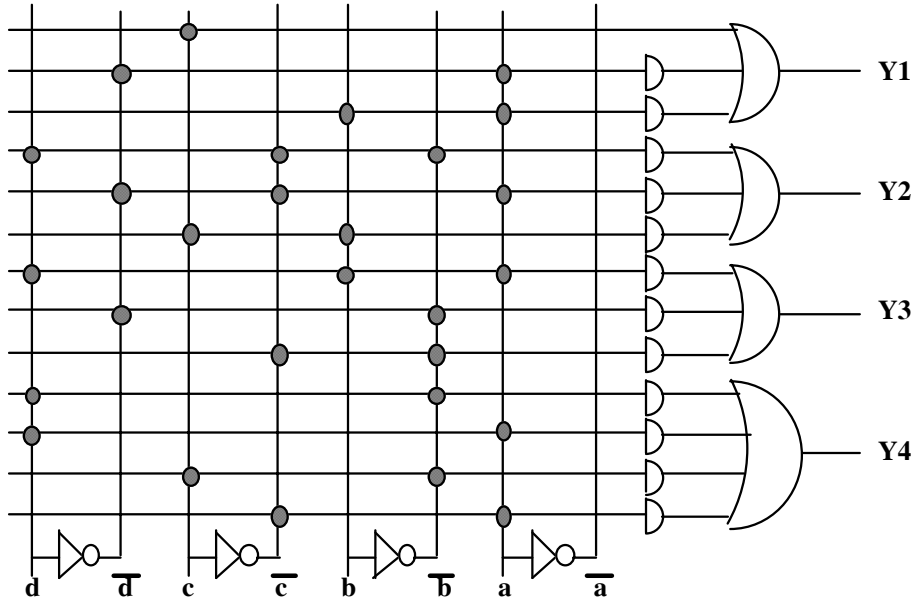
Y4

dc \ ba	00	01	11	10
00	0	1	1	0
01	1	1	0	0
11	1	1	1	0
10	1	1	1	0

$Y1 = c + \bar{d}.a + b.a$ $\qquad Y2 = d.\bar{c}.\bar{b} + \bar{d}.\bar{c}.a + c.b$

$Y3 = d.b.a + \bar{d}.\bar{b} + \bar{c}.\bar{b}$ $\qquad Y4 = d.\bar{b} + d.a + c.\bar{b} + \bar{c}.a$

La siguiente figura representa, en forma reticular, las cuatro funciones booleanas en configuración suma de productos $\sum \mathbf{p}$: cada función ocupa un módulo separado, constituido por una puerta "**o**" que recibe las salidas de varias puertas "**y**"; las conexiones de las variables se ajustan a la retícula formada por las entradas y sus negadas y las «líneas de entrada» de las puertas "**y**".



Esta forma reticular de dibujar la configuración $\sum \mathbf{p}$ de un conjunto de funciones recibe el nombre de estructura **PAL** (*product adding layers*: estratos o módulos sumadores de productos); en ella, cada función dispone de sus propios términos producto y las diversas funciones no comparten ninguna puerta booleana, salvo los inversores de las variables de entrada.

Es claro que la configuración **PAL** no es sino una forma de organizar el dibujo de las funciones y no añade nada a su expresión en suma de productos. Pero, si las funciones booleanas son complejas, esta forma facilita el dibujo y la comprensión del mismo; además, esta estructura da lugar a un tipo de bloques integrados programables que permiten construir directamente las funciones booleanas en su interior, mediante la programación de sus conexiones.

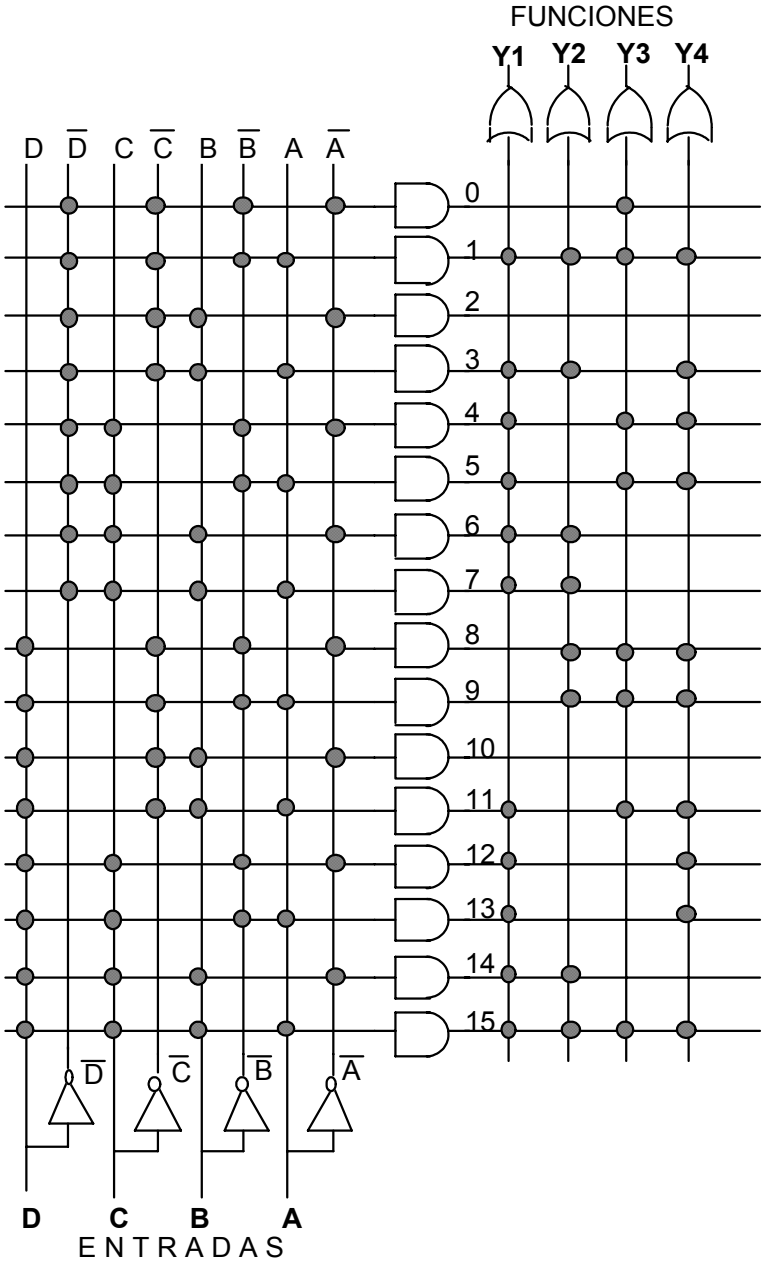
La retícula de conexiones sobre las puertas "**y**" permite describir tales conexiones en forma de matriz de «ceros» y «unos»; es decir, la configuración reticular equivale a una descripción matricial de las funciones, a la cual podemos denominar *Matriz Y de términos producto*:

Matriz de conexiones:	**d**	$\overline{\mathbf{d}}$	**c**	$\overline{\mathbf{c}}$	**b**	$\overline{\mathbf{b}}$	**a**	$\overline{\mathbf{a}}$
- función Y1:	0	0	1	0	0	0	0	0
	0	1	0	0	0	0	1	0
	0	0	0	0	1	0	1	0
- función Y2:	1	0	0	1	0	1	0	0
	0	1	0	1	0	0	1	0
	0	0	1	0	1	0	0	0
- función Y3:	1	0	0	0	1	0	1	0
	0	1	0	0	0	1	0	0
	0	0	0	1	0	1	0	0
- función Y4:	1	0	0	0	0	1	0	0
	1	0	0	0	0	0	1	0
	0	0	1	0	0	1	0	0
	0	0	0	1	0	0	1	0

La configuración **ROM** (*row ordered minterms*: filas de términos mínimos) constituye otra forma de construir funciones, utilizando un decodificador seguido del encodificador que corresponde a la tabla de las funciones:

- las **m** entradas del decodificador actúan como variables de entrada para todas las funciones,
- las $\mathbf{2^m}$ líneas intermedias corresponden a los términos mínimos de las variables de entrada,
- y cada salida realizará una de las funciones booleanas, mediante una puerta "**o**", haciendo confluir sobre ella los términos mínimos que intervienen en dicha función (forma canónica de la misma).

La figura de la página siguiente representa la configuración **ROM** de las cuatro funciones booleanas que estamos considerando como ejemplo: el decodificador construye físicamente los 16 términos mínimos de las 4 variables de entrada y el encodificador efectúa, para cada una de las funciones, la suma de los términos mínimos que la activan (que producen valor **1** para dicha función).



En términos de puertas booleanas la estructura **ROM** presenta gran simplicidad y regularidad; está constituida modularmente por dos conjuntos de puertas, cuyas conexiones adoptan una configuración reticular, que puede describirse mediante matrices de «ceros» y «unos».

Matriz del decodificador:
Matriz Y completa

d	$\bar{\mathbf{d}}$	**c**	$\bar{\mathbf{c}}$	**b**	$\bar{\mathbf{b}}$	**a**	$\bar{\mathbf{a}}$
0	1	0	1	0	1	0	1
0	1	0	1	0	1	1	0
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

Matriz del encodificador:
Matriz O funcional

Y1	**Y2**	**Y3**	**Y4**
0	0	1	0
1	1	1	1
0	0	0	0
1	1	0	1
1	0	1	1
1	0	1	1
1	1	0	0
1	1	0	0
0	1	1	1
0	1	1	1
0	0	0	0
1	0	1	1
1	0	0	1
1	0	0	1
1	1	0	0
1	1	1	1

La denominación Y/O de las matrices alude al tipo de puertas que las conforman. La primera matriz es de dimensión $\mathbf{2^m \times 2m}$ y la segunda es de $\mathbf{2^m \times n}$, siendo **m** el número de entradas y **n** el número de salidas o funciones ($\mathbf{2^m}$ es el número de términos mínimos o líneas intermedias).

El decodificador, formado por puertas "**y**" contiene todas las posibilidades de sus variables de entrada, cada una de las cuales corresponde a un término mínimo; por ello la configuración reticular de sus conexiones recibe el nombre de *Matriz Y completa*.

El encodificador corresponde directamente a la «tabla de verdad» de las funciones ya que sus puertas "**o**" han de recibir las conexiones de aquellos términos mínimos cuyo valor en dicha tabla es **1**: *Matriz O funcional*.

Este procedimiento de configurar funciones booleanas (formando todos los términos mínimos de sus entradas y construyendo a partir de ellos la forma canónica de la función) es sumamente útil cuando el número de entradas es grande (**m > 6**) y cuando son varias las funciones a construir. La estructura **ROM** facilita en gran manera la síntesis de las funciones, evitando la necesidad de obtener sus expresiones algebraicas y de simplificar dichas expresiones.

5.3. Simplificación multifunción: estructura PLA

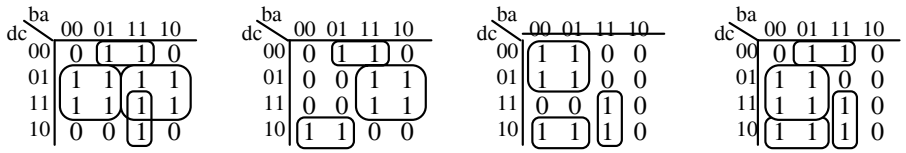
La configuración **ROM** consta de dos matrices de conexiones (Matriz Y completa - Matriz O funcional); el circuito resultante es amplio por cuanto que contiene todos los términos mínimos de las entradas y tantas conexiones en la Matriz O como «unos» hay en la tabla funcional. Las funciones comparten los términos mínimos proporcionados por el decodificador inicial, pero su número es grande: $\mathbf{2^m}$ siendo **m** el número de entradas.

La configuración **PAL** representa a las funciones una vez simplificadas, pero trata a las funciones individualmente: cada módulo PAL es independiente de los demás; no aprovecha la posibilidad de compartir términos producto entre dos o más funciones.

Una tercera posibilidad consiste en utilizar términos productos simplificados, pero no de cada función individual, sino en compartir los términos resultantes de simplificar las **n** funciones conjuntamente: es lo que se conoce como simplificación multifunción.

De esta forma se aprovecha cada término producto para varias funciones a la vez y el número de términos producto (y por tanto el circuito a construir) será menor. Ahora bien, para ello es necesario utilizar un proceso de simplificación multifunción que es complicado: en general es muy laborioso (e, incluso, imposible) de realizar a mano, pero se dispone de aplicaciones informáticas para ejecutarlo en computador.

En el caso del conjunto de cuatro funciones de cuatro entradas consideradas en el apartado anterior, la simplificación multifunción permite reducir los 13 términos producto que intervienen en sus expresiones algebraicas simplificadas individuales a solamente 6 términos producto compartidos.



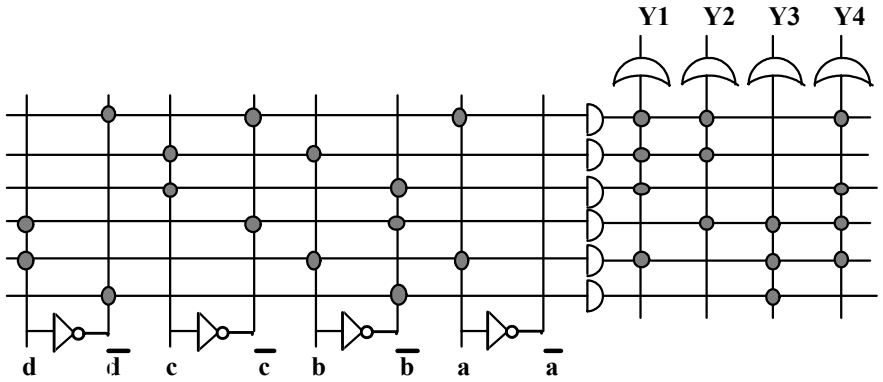
$$Y1 = \bar{d}.\bar{c}.a + c.\bar{b} + c.b + d.b.a$$

$$Y2 = \bar{d}.\bar{c}.a + c.b + d.\bar{c}.\bar{b}$$

$$Y3 = + d.b.a + d.\bar{c}.\bar{b} + \bar{d}.\bar{b}$$

$$Y4 = \bar{d}.\bar{c}.a + c.\bar{b} + d.b.a + d.\bar{c}.\bar{b}$$

La representación gráfica de estas expresiones da lugar a la siguiente configuración reticular:



La figura anterior representa a las cuatro funciones booleanas compartiendo términos producto como resultado de una simplificación multifunción $\sum$**pc** (suma de productos compartidos); tal simplificación permite reducir en gran medida el número de términos producto necesarios para el conjunto de funciones, aunque la expresión resultante para cada función no es la más simplificada posible.

Esta forma de construir un conjunto de funciones recibe el nombre de estructura **PLA** (*product linking adders*: sumadores que enlazan o comparten productos); las conexiones conforman dos matrices: la *Matriz Y de términos producto compartidos* y la *Matriz O de salidas*. [Una vez más conviene hacer notar que estos tipos de organización reticular o matricial son particularmente útiles para funciones de un amplio número de entradas.]

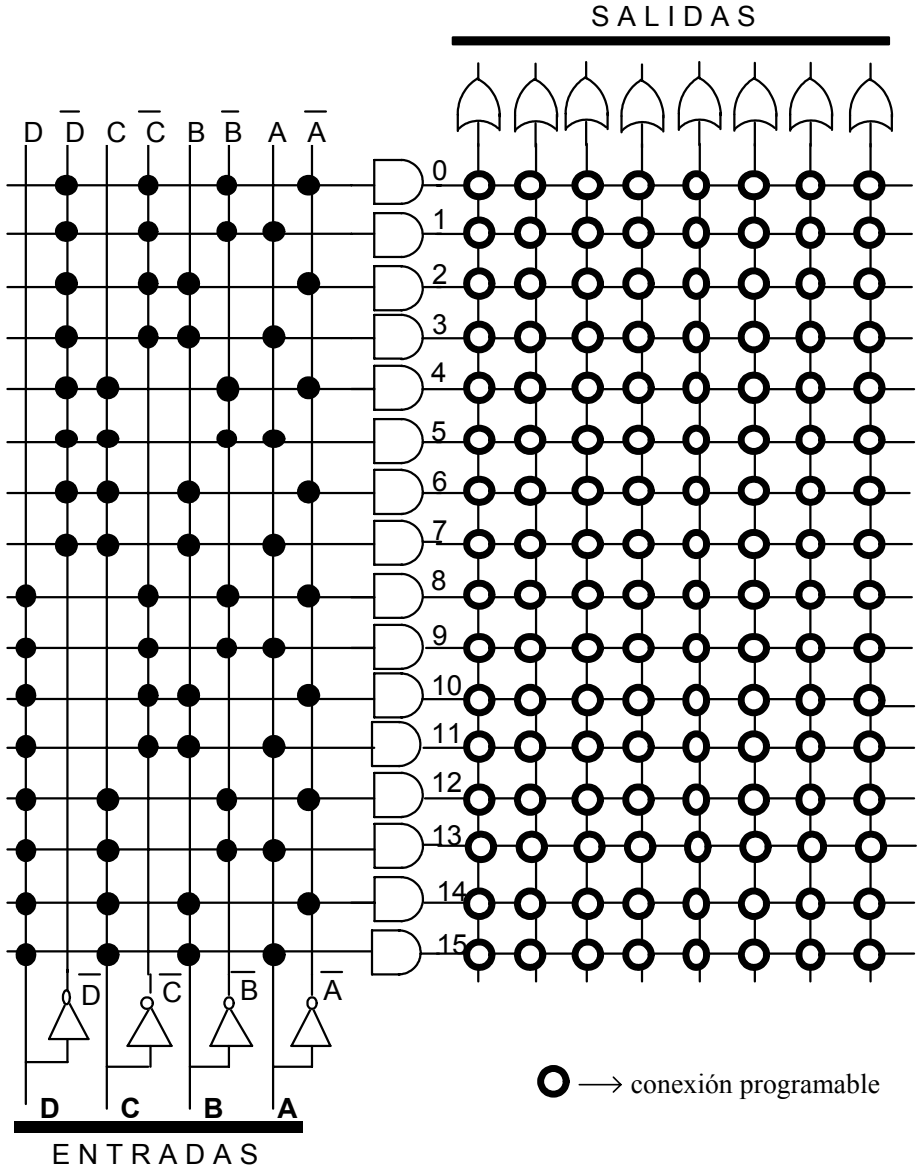
La configuración **PLA** supone un claro ahorro circuital por cuanto que minimiza el conjunto de términos producto necesarios, pero, en cambio, requiere un complejo proceso previo de simplificación multifunción.

5.4. Bloques integrados programables

Consideremos un bloque **ROM** de **m** entradas y **n** salidas que, por tanto, contiene **n** funciones booleanas de **m** entradas; particularizar dicho bloque a **n** funciones booleanas dadas consiste en determinar en cuáles de los puntos de la *Matriz O funcional* debe haber conexión y en cuáles no.

Supongamos que todos los puntos de la Matriz O de conexiones sobre las puertas "**o**" de salida se encuentren conectados (por ejemplo, mediante fusibles) y que el usuario pueda eliminar aquellas conexiones que no interesen (fundiendo los fusibles); de esta forma dispondremos de un bloque **ROM** *programable* por el usuario: **PROM** [véase la figura de la página siguiente].

La siguiente figura representa un bloque **PROM** de 4 entradas y 8 salidas, apto para la programación de 8 funciones de 4 variables.



Codificador PROM de 4 líneas de entrada y 8 líneas de salida

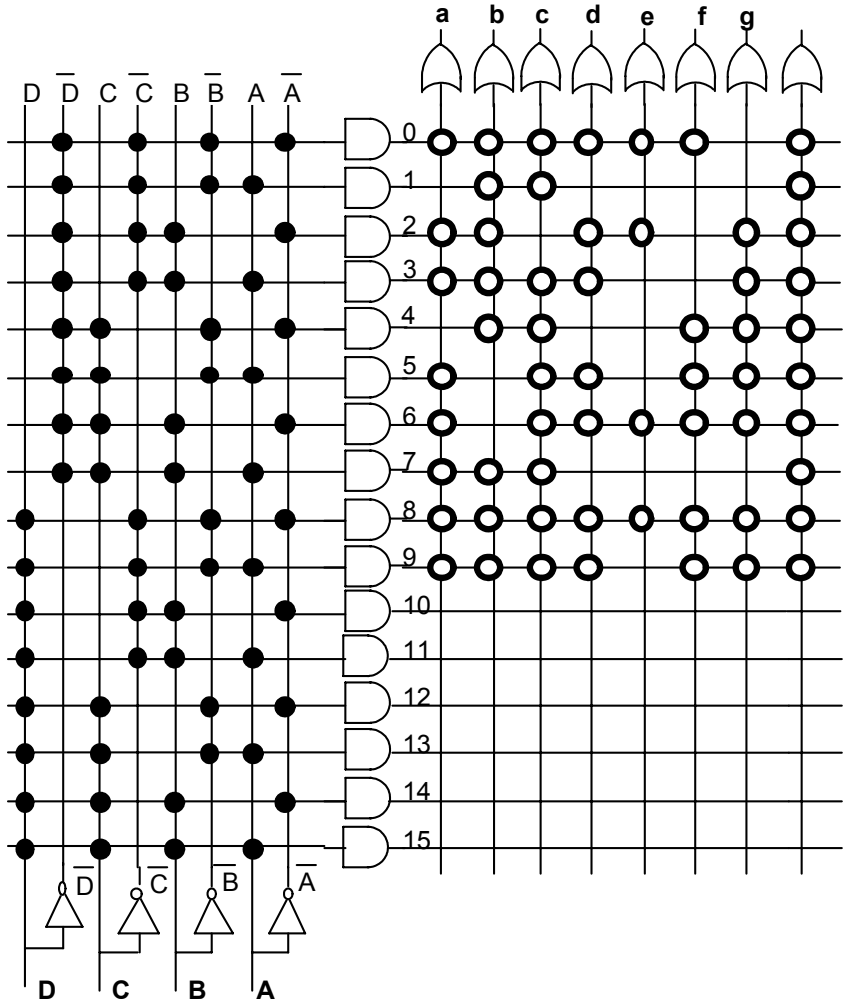
La tecnología electrónica de integración de circuitos ha permitido construir este tipo de bloques **PROM**, programables eléctricamente mediante procesos análogos a la ruptura de fusibles. Su programación consiste en eliminar aquellas conexiones de la Matriz O que no deben estar presentes conforme a la tabla funcional del conjunto de funciones booleanas deseado, es decir, eliminar (*fundir los fusibles de*) aquellas conexiones que corresponden a valor **0** en la tabla funcional.

La programación de un bloque **PROM**, a partir de la tabla de las **n** funciones booleanas equivale a reflejar sobre la Matriz O la tabla de dichas funciones:

- Cada una de las líneas intermedias del bloque **PROM** (salidas del decodificador que conforman las filas de la Matriz O) se corresponde con un término mínimo, al igual que cada fila de la tabla funcional, y cada una de las columnas de la Matriz O se corresponde con una de las funciones, igual que cada columna de dicha tabla.
- Existe una correspondencia directa entre la matriz de valores **0** y **1** que configura la parte de la derecha de la tabla funcional (vectores de salida) y la matriz de conexiones sobre las puertas "**o**" de salida, Matriz O, asociando el valor **0** a la eliminación o ausencia de conexión y el valor **1** a la conservación o presencia de la misma.
- Para cada columna de la Matriz O los valores **1** presentes en la correspondiente columna de la tabla funcional determinan los términos mínimos que deben estar presentes en la función booleana que tal columna configura y, por tanto, las conexiones que deben incidir sobre la puerta "**o**" que realiza dicha función.

La figura de la página siguiente corresponde a la programación de las 7 funciones de un conversor BCD → 7 segmentos [coincide con la figura de la página 103 del tema anterior, apartado 4.4.]. La tabla de dichas funciones es la siguiente:

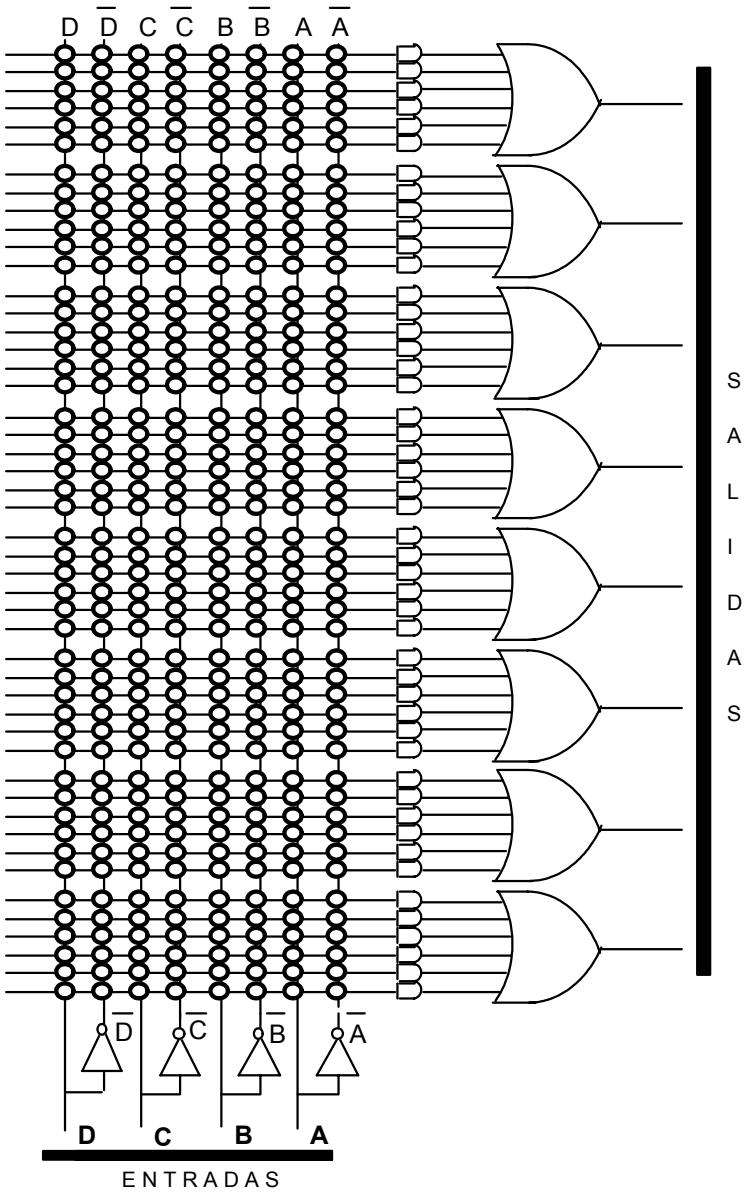
D	C	B	A	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1



Programación del conversor BCD → 7 segmentos en la PROM anterior

En los bloques programables tipo **PAL** (véase la figura de la página siguiente) cada puerta "**o**" de salida está conectada a un subconjunto independiente de puertas "**y**", siendo programables las conexiones de las entradas sobre dichas puertas "**y**"; de forma que cada función de salida posee su propia puerta "**o**" y su propio subconjunto de puertas "**y**" que inciden sobre ella.

La siguiente figura representa un bloque **PAL** de 4 entradas y 8 salidas, con 6 términos producto en cada módulo.



PAL programable de 4 líneas de entrada y 8 de salida con 6 términos producto

De forma que un bloque **PAL** es un conjunto de **n** módulos independientes, cada uno de los cuales está constituido por la suma de **q** términos producto de las **m** variables de entrada: cada una de las **n** funciones se programa con independencia de las demás, pues no poseen términos producto compartidos.

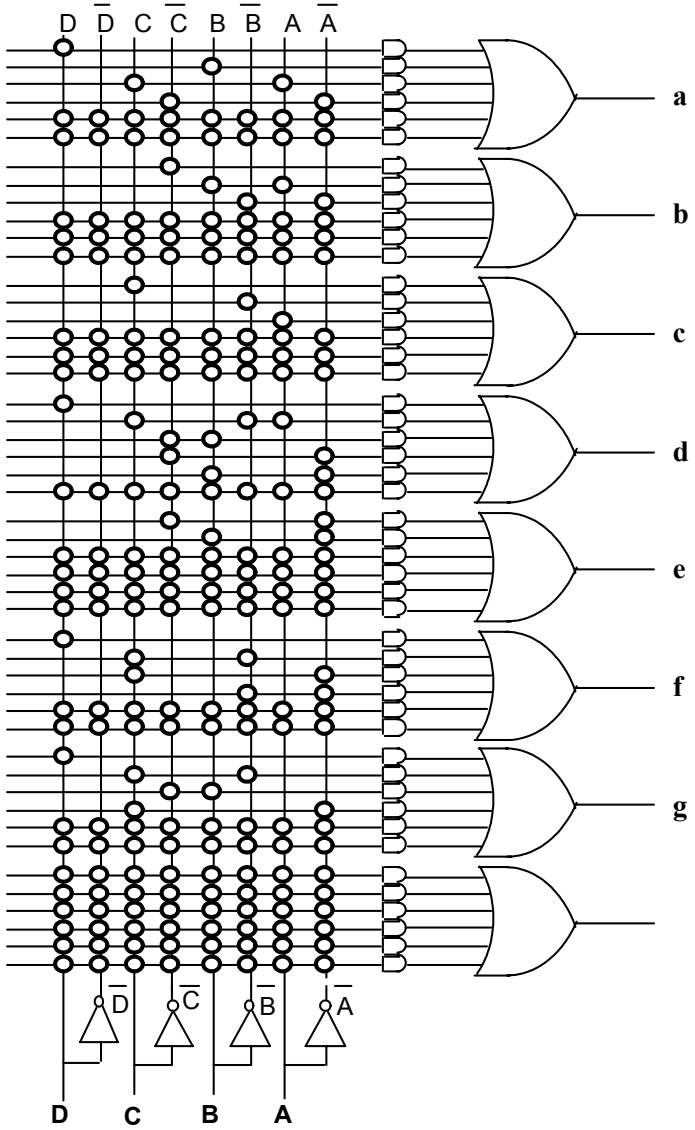
La estructura de los bloques **PAL** coincide con las funciones expresadas como suma de términos producto $\sum \mathbf{p}$, sin necesidad de preocuparse por las coincidencias de términos entre las diversas funciones. Por ello su programación es muy sencilla y se deduce directamente de la expresión algebraica simplificada de cada una de las funciones a programar.

Ahora bien, un bloque programable **PAL** se encuentra limitado por el número de términos producto disponibles en cada módulo. No es posible programar cualquier función booleana de **m** variables: no podrán programarse aquellas funciones cuya expresión algebraica, una vez simplificada al máximo, incluya un número de términos producto mayor.

La programación del conversor BCD $\rightarrow$ 7 segmentos sobre un bloque **PAL** consiste en configurar directamente cada una de sus siete funciones, una vez simplificadas:

$$a = D + B + C.A + \overline{C}.\overline{A}$$
$$b = \overline{C} + B.A + \overline{B}.\overline{A}$$
$$c = C + \overline{B} + A$$
$$d = D + C.\overline{B}.A + \overline{C}.B + \overline{C}.\overline{A} + B.\overline{A}$$
$$e = \overline{C}.\overline{A} + B.\overline{A}$$
$$f = D + C.\overline{B} + C.\overline{A} + \overline{B}.\overline{A}$$
$$g = D + C.\overline{B} + \overline{C}.B + C.\overline{A}$$

Las funciones **a**, **f** y **g** requieren 4 términos producto, la **b** y la **c** 3 términos, mientras que la función **d** requiere 5 términos y la función **e** solamente 2 términos producto.

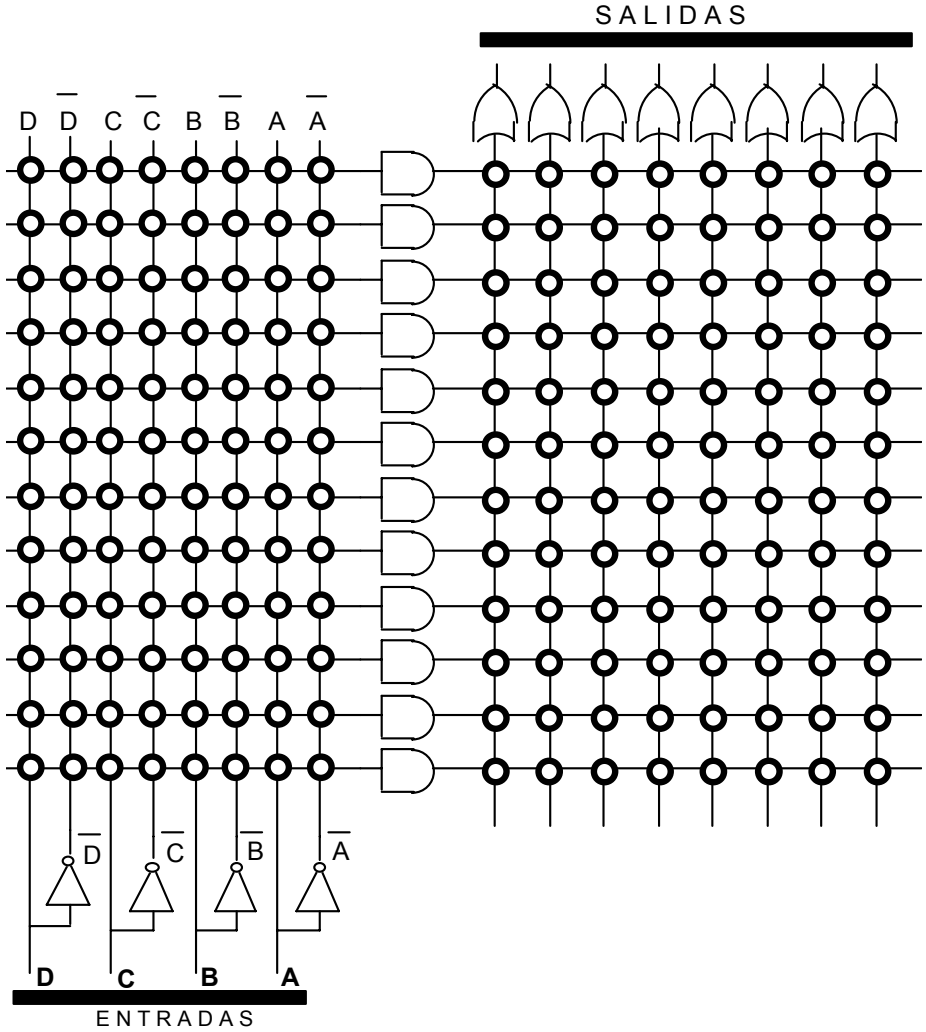


Conversor BCD → 7 segmentos programado en la PAL anterior

Obsérvese que en esta figura no se han programado *(no se han eliminado las conexiones de*) aquellos términos producto que no han sido necesarios; un término producto sin programar se anula por sí mismo, ya que contiene el producto de variables por sus negadas ($a \cdot \bar{a} = 0$).

La configuración **PLA** requiere que ambas matrices sean programables, tanto la Matriz Y de entradas como la Matriz O de salidas: sobre la Matriz Y se programarán los términos producto que son necesarios para realizar las **n** funciones y sobre la Matriz O se programará la inclusión o no de cada término producto en cada función.

La siguiente figura presenta una **PLA** de 4 entradas, 8 salidas y 12 términos producto.



PLA de 4 líneas de entrada, 4 líneas de salida y 12 términos producto.

La programación de un bloque **PLA** resulta relativamente compleja por cuanto que es preciso simplificar conjuntamente las **n** funciones a programar, a fin de minimizar el número de términos producto necesarios.

No basta la simplificación individual de las **n** funciones; al contrario, la minimización de cada una de ellas por separado impedirá, en la mayoría de los casos, encontrar términos producto coincidentes en varias que permitan minimizar el conjunto: el objetivo no es buscar el menor número de términos producto para cada una de las funciones, sino para el conjunto global de ellas.

La programación del conversor BCD $\rightarrow$ 7 segmentos corresponde a las siete funciones siguientes, obtenidas mediante simplificación multifunción:

$$a = D + C.A + \overline{C}.\overline{A} + B.A + B.\overline{A}$$

$$b = \overline{C}.A + \overline{C}.\overline{A} + B.A + \overline{B}.\overline{A}$$

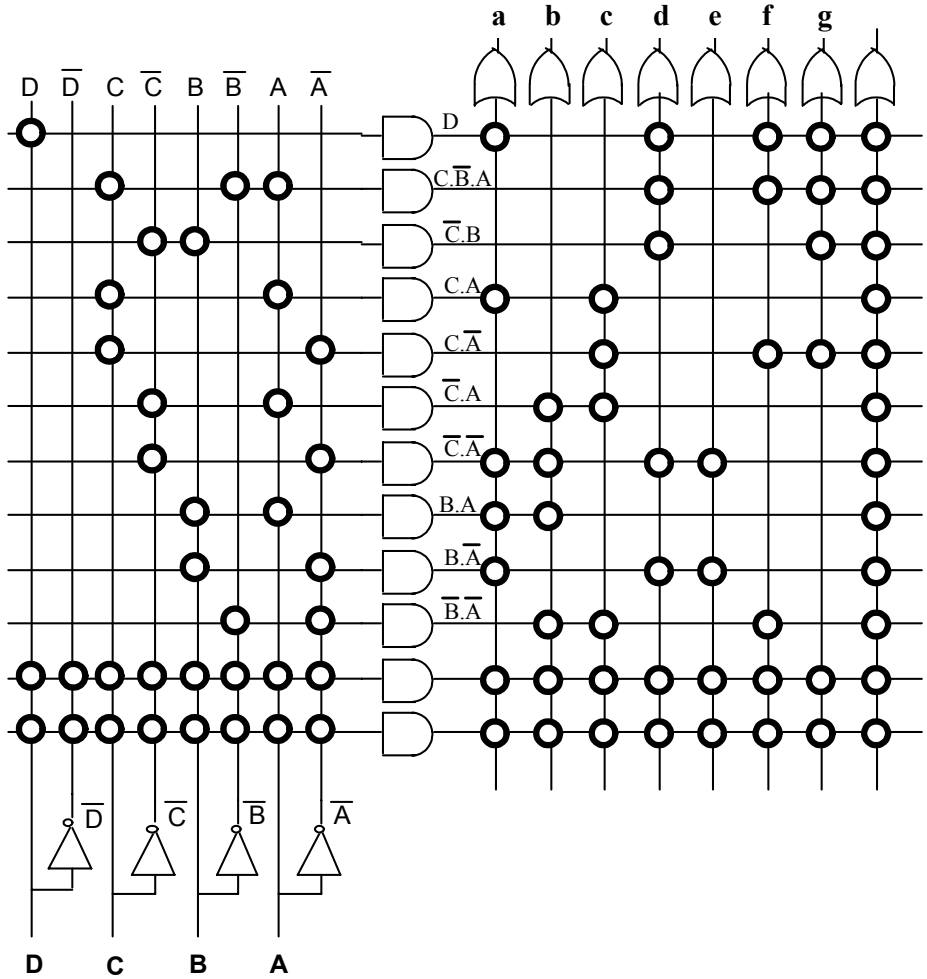
$$c = C.A + C.\overline{A} + \overline{C}.A + \overline{B}.\overline{A}$$

$$d = D + C.\overline{B}.A + \overline{C}.B + \overline{C}.\overline{A} + B.\overline{A}$$

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

$$f = D + C.\overline{B}.A + C.\overline{A} + \overline{B}.\overline{A}$$

$$g = D + C.\overline{B}.A + \overline{C}.B + C.\overline{A}$$



Conversor BCD $\rightarrow$ 7 segmentos programado en la PLA anterior

Al igual que en el caso de la **PAL**, no se han programado *(no se han eliminado las conexiones de)* aquellos términos producto que no han sido necesarios, ya que se anulan por sí mismos (producto de variables por sus negadas, $a \,.\, \overline{a} = 0$).

En resumen, se dispone de tres bloques de lógica programable que permiten realizar **n** funciones de **m** entradas, expresadas éstas como suma de términos producto (en el caso del bloque **PROM** como suma de términos mínimos). Los bloques **PROM** son programables en cuanto a su Matriz O de salidas, los bloques **PAL** lo son en cuanto a su Matriz Y de entradas y los bloques **PLA** son programables respecto a sus dos matrices Y, O.

Los tres bloques se encuentran configurados por una *Matriz Y* de conexiones de las entradas y de sus negadas sobre puertas "**y**" y una *Matriz O* de conexiones de las salidas de las puertas "**y**" sobre las puertas "**o**" que conforman las salidas del bloque; difieren en cuanto a la capacidad de programación de dichas matrices, conforme a la siguiente tabla:

	Matriz Y	**Matriz O**	**Nº de términos producto**
PROM	fija	programable	$\mathbf{2^m}$ = nº términos mínimos
PLA	programable	programable	$\mathbf{p << 2^m}$
PAL	programable	fija	**n x q** (**q < p**)

Un bloque **PROM** admite la programación de **n** funciones cualesquiera de **m** entradas; los bloques **PLA** se encuentran limitados por el número total de términos producto ($\mathbf{p << 2^m}$) disponibles en la Matriz Y, mientras que los bloques **PAL** están limitados en el número de términos producto (**q < p**) de cada módulo.

La programación **PROM** consiste en reflejar sobre la Matriz O la tabla de las funciones (de forma que se eliminen aquellas conexiones que corresponden a valor **0** en dicha tabla). La programación de cada módulo **PAL** corresponde a la expresión algebraica simplificada de la función en forma de suma de términos producto (con tal de que el número de términos sea igual o inferior a los disponibles en el módulo **PAL**). La programación **PLA** supone configurar en su Matriz Y los términos producto resultantes de la simplificación multifunción de las funciones a programar y, posteriormente, configurar sobre la Matriz O las sumas de aquellos términos que dan lugar a cada una de las funciones.

Tanto en el caso PAL como PLA no es necesario programar (*eliminar*) los términos producto no utilizados ya que se anulan ellos solos, por contener productos de una variable por su negada ($a \cdot \bar{a} = 0$)

El número de conexiones de las matrices puede servir para comparar las tres estructuras **PROM**, **PLA**, **PAL**, en cuanto a sus dimensiones físicas (en su realización electrónica el número de conexiones equivale aproximadamente al número de transistores, ya que cada conexión se realiza a través de un transistor):

	Matriz Y	**Matriz O**	
PROM	$\mathbf{m \times 2^m}$	$\mathbf{n \times 2^m}$ programables	
PLA	**2m x p** programables	**n x p** programables	$\mathbf{p << 2^m}$
PAL	**2m x q x n** programables	**n x q**	**q < p**

Por ejemplo, comparando una **PROM** de 12 entradas y 8 salidas, una **PLA** del mismo número de entradas y salidas con 28 términos producto y una **PAL** análoga con 6 términos para cada salida:

	Matriz Y		**Matriz O**	
PROM	12 x 4.096		8 x 4.096	programables
PLA	24 x 28	programables	8 x 28	programables
PAL	24 x 6 x 8	programables	8 x 6	

resulta que la configuración **PROM** requiere *81.920* conexiones, de las cuales *32.768* son programables, mientras que la **PLA** necesita *896* conexiones (la centésima parte de las que precisa la **PROM**), todas ellas programables, y la **PAL** utiliza *1.200* conexiones, de las cuales *1.152* son programables.

En la anterior **PROM** pueden programarse cualquier conjunto de 8 funciones de 12 entradas, mientras que en la **PAL** no caben funciones cuya expresión algebraica contenga más de 6 términos producto y la **PLA** puede admitir hasta 8 funciones de 12 entradas con tal de que su simplificación multifunción permita reducir el número total de términos producto necesarios a 28 o menos.

Forma de efectuar la programación

La programación de los circuitos integrados programables se realiza habitualmente mediante un programador conectado a un computador; se requieren dos aplicaciones informáticas: un compilador que traduce la descripción del diseño al «mapa de fusibles» del circuito integrado y un programa de control del programador que «ejecuta» el mapa de fusibles sobre el propio circuito integrado.

Se parte de la descripción del diseño específico a programar; tal descripción puede hacerse en forma gráfica (esquemático de puertas lógicas o de bloques), pero lo habitual es hacerla en forma de texto utilizando un lenguaje de descripción circuital (VHDL, Verilog, ABEL, ..., siendo el VHDL el más empleado).

Un compilador apropiado traslada la descripción a funciones booleanas y las «encaja» sobre la configuración del dispositivo programable, obteniendo el «mapa de fusibles» que indica cuales de las conexiones programables se han de eliminar y cuales deben permanecer.

El programador ejecuta el mapa de fusibles sobre el circuito integrado programables, actuando «una a una» sobre las conexiones que deben eliminarse y verificando, posteriormente, que tales conexiones han quedado suprimidas y el resto de ellas mantienen su unión eléctrica.

Ejemplo: conversor BCD → 7 segmentos, de cátodo común

a) descripción VHDL del circuito

```
entity CONVERSOR is
port(  SALIDA        : Out   BIT_VECTOR(1 to 7);
       D,C,B,A       : In    BIT);
end CONVERSOR;

architecture CATODOCOMUN of CONVERSOR is
begin
with D & C & B & A select
     SALIDA <=   "1111110" when "0000",
                 "0110000" when "0001",
                 "1101101" when "0010",
                 "1111001" when "0011",
                 "0110011" when "0100",
                 "1011011" when "0101",
                 "1011111" when "0110",
                 "1110000" when "0111",
                 "1111111" when "1000",
                 "1111011" when "1001",
                 "0000000" when others;
end CATODOCOMUN;
```

b) funciones booleanas y mapa de fusibles resultantes de la compilación

A continuación se adjuntan las funciones relativas a las 7 salidas del conversor obtenidas por el compilador WARP sobre la descripción VHDL anterior; inmediatamente debajo de cada una de las funciones se expresa el mapa de fusibles correspondiente para un módulo PAL de 10 entradas y 6 términos producto.

*Comentario: las funciones que siguen son diferentes de las indicadas para el mismo conversor en el capítulo anterior (apartado 4.3.1.) y reproducidas en este capítulo en la página 132; ello es debido a que en este ejemplo se asigna vector de salida **0000000** para números de entrada superiores a 9, mientras que en el capítulo 4 se les asignaba vector de salida **XXXXXXX** para simplificar en lo posible las funciones resultantes.*

salida_1 = /c * /b * /a + /d * c * a + d * /c * /b + /d * b

```
00010101000000000000
01100010000000000000
10010100000000000000
01001000000000000000
11111111111111111111
11111111111111111111
```


salida_2 = d * b * a + /d * /b * /a + /c * /b + /d * /c

```
10001010000000000000
01000101000000000000
00010100000000000000
01010000000000000000
11111111111111111111
11111111111111111111
```

salida_3 = /c * /b + /d * a + /d * c

```
00010100000000000000
01000010000000000000
01100000000000000000
11111111111111111111
11111111111111111111
11111111111111111111
```

salida_4 = /d * c * /b * a + /c * /b * /a + /d * b * /a + d * /c * /b + /d * /c * b

```
01100110000000000000
00010101000000000000
01001001000000000000
10010100000000000000
01011000000000000000
11111111111111111111
```

salida_5 = /c * /b * /a + /d * b * /a

```
00010101000000000000
01001001000000000000
11111111111111111111
11111111111111111111
11111111111111111111
11111111111111111111
```

salida_6 = /c * /b * /a + d * /c * /b + /d * c * /a + /d * c * /b

```
00010101000000000000
10010100000000000000
01100001000000000000
01100100000000000000
11111111111111111111
11111111111111111111
```

salida_7 = /d * b * /a + d * /c * /b + /d * /c * b + /d * c * /b

```
01001001000000000000
10010100000000000000
01011000000000000000
01100100000000000000
11111111111111111111
11111111111111111111
```