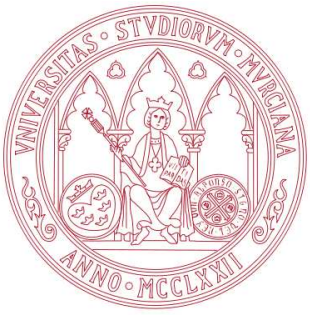


Universidad de Murcia

Facultad de Informática

TÍTULO DE GRADO EN
INGENIERÍA INFORMÁTICA

Fundamentos de Computadores

Tema 3: Sistemas Digitales: Circuitos Combinacionales

Apuntes de teoría

CURSO 2020 / 21

Departamento de Ingeniería y Tecnología de Computadores

Área de Arquitectura y Tecnología de Computadores

Índice general

I. Introducción **2**
I.1. Álgebra de Boole . . . 3
I.1.1. Minitérminos . . . 4
I.1.2. Maxitérminos . . . 4
I.2. Simplificación de funciones mediante mapas de Karnaugh . . . 5
I.2.1. Terminología para la minimización de funciones lógicas . . . 7
I.2.2. Algoritmo de minimización mediante mapas de Karnaugh . . . 8
I.2.3. Simplificación por ceros . . . 8
I.2.4. Salidas no determinadas . . . 9

II. Circuitos combinacionales comunes **11**
II.1. Puertas lógicas básicas . . . 11
II.1.1. Circuitos integrados . . . 12
II.2. Retardos . . . 13
II.3. Implementación de funciones lógicas con puertas NAND/NOR . . . 14
II.4. Bloques lógicos . . . 15
II.5. Codificadores y decodificadores . . . 16
II.6. Multiplexores . . . 18
II.7. Memorias ROM y arrays lógicos programables . . . 20
II.7.1. Memorias ROM . . . 20
II.7.2. PROM, EPROM, EEPROM . . . 21
II.7.3. PLA . . . 22

Introducción

Los ordenadores que se fabrican hoy en día son dispositivos digitales en donde la información se representa de forma discreta, frente a los dispositivos analógicos que manejan formas continuas de información. La electrónica digital opera solamente con dos niveles de tensión de interés: una tensión alta y una tensión baja. Los demás valores de tensión únicamente se presentan durante la transición entre los dos valores anteriores. Por este motivo, los ordenadores utilizan sistemas de numeración binarios, lo que nos permite hacer coincidir la abstracción en la que se basan los ordenadores con la electrónica digital. Según la familia lógica, los valores y relaciones entre los dos niveles de tensión difieren. Por tanto, en lugar de referenciar niveles de tensión, hablamos de señales que son (lógicamente) verdaderas ó 1; y señales que son (lógicamente) falsas ó 0.

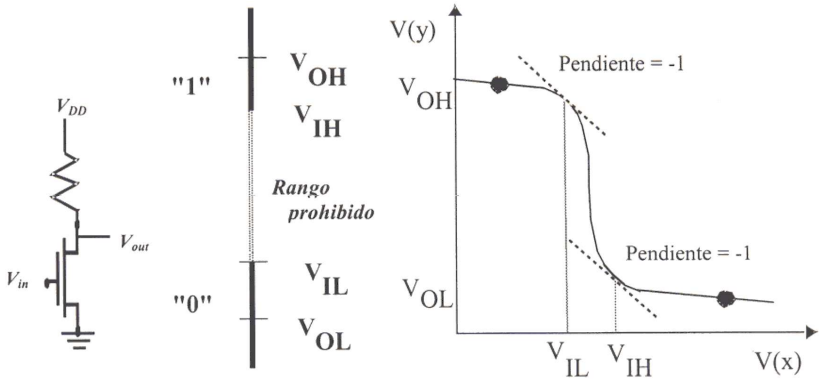


Figura I.1: Implementación de una puerta NOT mediante una resistencia y un transistor.

La figura I.1 muestra la implementación de una puerta NOT mediante el uso de una resistencia y un transistor NMOS. Cuando el voltaje de entrada es menor que un cierto valor V_T el transistor, que funciona a grandes rasgos como un interruptor, no conduce y el voltaje de salida V_{out} es igual a V_{DD} (la alimentación del circuito). Cuando el voltaje de entrada supera ese valor, el interruptor se cierra y la corriente circula por el transistor haciendo que el voltaje de salida V_{out} sea igual a 0.

Para establecer una correspondencia entre niveles de voltaje y valores lógicos usaremos el concepto de valor umbral. Los voltajes por encima de un cierto valor umbral representarán un valor lógico (normalmente el uno lógico), mientras que los valores por debajo de ese umbral representarán el otro valor (normalmente el cero lógico). Dado que en la realidad el voltaje en cualquier punto de un circuito electrónico puede variar levemente por diversos motivos, los valores cercanos al valor umbral son desaconsejables ya que, dependiendo de las circunstancias, pueden ser interpretados de diferente manera. Para evitar esta ambigüedad, se establece un rango prohibido, tal y como se muestra en la misma figura. Este rango viene determinado por el análisis de la llamada función característica del voltaje de salida de un inversor construido con la tecnología a utilizar. En este caso, valores por debajo de V_{IL} representan el valor 0, y voltajes por encima de V_{IH} representan el valor 1.

Aunque la implementación de cualquier función lógica mediante una resistencia de carga y un conjunto de transistores NMOS es factible, presenta dos problemas importantes que hace que este tipo de circuitos no se utilicen en la actualidad. El primero de ellos es el espacio en silicio que ocupa una resistencia, demasiado grande en comparación con el que ocupa un transistor. Esto provoca problemas en la escala de integración de puertas en el chip. El segundo y principal problema es el ligado al consumo de este tipo de circuitos: cuando la salida está a nivel lógico 0, el transistor se encuentra conduciendo y se produce un consumo de energía (y el correspondiente calentamiento) en la resistencia que impediría la fabricación de circuitos integrados con millones de puertas lógicas como los que necesitan los microprocesadores actuales.

La alternativa utilizada hoy en día es la sustitución de la resistencia por un transistor PMOS. Este tipo de transistor se comporta a la inversa del transistor NMOS: cuando se le aplica un voltaje alto al terminal de la

puerta el transistor, que actúa como un interruptor, permanece abierto, cerrándose cuando a la entrada aparece un cero lógico. La figura I.2 muestra la implementación de un inversor, así como la función característica de la salida.

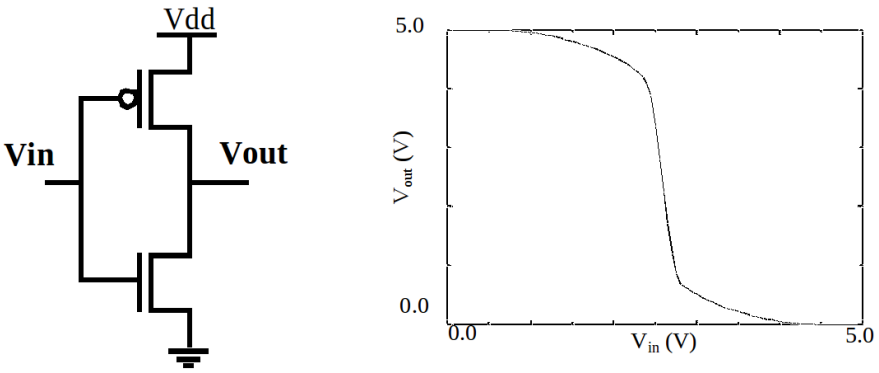


Figura I.2: Inversor CMOS.

La característica clave de este circuito es que los transistores operan de modo complementario; cuando uno conduce, el otro está en corte. Así, siempre existe un camino desde la salida hacia V_{DD} o hacia tierra. No existe camino entre la fuente y tierra, excepto durante un corto periodo donde los transistores están cambiando de estado. Esto significa que el circuito no disipa potencia cuando permanece en un estado estable, sólo cuando cambia de un estado lógico a otro. Por lo tanto, la disipación de potencia del circuito depende del número de cambios que se produce por segundo. Esta es la principal causa de que, con el aumento en la frecuencia de operación de los procesadores, aumenten los problemas de consumo y calentamiento.

Finalizaremos esta introducción comentando que en la electrónica digital podemos diferenciar dos grandes familias de circuitos: *circuitos combinacionales* y *circuitos secuenciales*. Si la salida en un instante dado sólo depende de las entradas en ese instante entonces nos encontramos ante un circuito combinacional. En el caso de que la salida dependa de entradas que se produjeron en instantes anteriores hablaremos de circuito secuencial. En este tema nos dedicaremos a ver algunos de los circuitos combinacionales más usuales, dejando para la asignatura de Estructura y Tecnología de Computadores (ETC) de segundo cuatrimestre el estudio de los circuitos secuenciales.

I.1. Álgebra de Boole

En 1854, George Boole publicó una obra titulada «Investigación de las leyes del pensamiento», sobre las que se basan las teorías matemáticas de la lógica y la probabilidad. En esta publicación se formuló la idea de un álgebra de las operaciones lógicas, que se conoce hoy en día como Álgebra de Boole. El Álgebra de Boole es una forma muy adecuada para expresar y analizar las operaciones de los circuitos lógicos. Claude Shannon fue el primero en aplicar la obra de George Boole al análisis y diseño de circuitos.

Un álgebra booleana es un sistema Algebraico cerrado formado por un conjunto K de dos o más elementos (en nuestro caso, asumiremos que $K = \{0,1\}$); los dos operadores binarios AND (el resultado es 1 si los operandos son 1) y OR (el resultado es 1 si alguno de los operandos es 1), conocidos también como producto lógico ($\cdot$) y suma lógica (+), respectivamente; y el operador unario NOT (el resultado es la inversión o negación del operando, es decir, 0 si el operando es 1 y viceversa) denominado negación lógica ($\bar{}$ o ').

Sobre dicha álgebra es posible definir funciones lógicas. Una función lógica es una función del tipo $F : \{0,1\} \times \{0,1\} \times ... \times \{0,1\} \rightarrow \{0,1\}$, o lo que es lo mismo $F : \{0,1\}^n \rightarrow \{0,1\}$. La función F posee n entradas cada una de las cuales puede tomar un valor 0 ó 1. Dependiendo de estos valores la salida a su vez tomará también uno de los dos valores. Por tanto, tenemos 2^n combinaciones de entrada para cada una de las cuales la función toma un valor.

Cualquier función lógica puede escribirse como una ecuación lógica, con una salida en la parte izquierda de la ecuación, y una fórmula, que consta de variables, complementos de variables y operadores, en la parte

derecha. Para una función lógica se pueden encontrar varias ecuaciones lógicas diferentes equivalentes. De éstas, para facilitar el trabajo, se eligen determinadas formas de suma de productos y producto de sumas, denominadas formas canónicas, que se caracterizan por ser únicas y que se describen a continuación.

I.1.1. Minitérminos

Dada una función de n variables de entrada, si un término producto contiene cada una de las n variables, ya sea en forma complementada o no, es un minitérmino. Si la función se representa como una suma de minitérminos, se dice que la función tiene forma de *suma canónica de productos* o que está en *forma normal disyuntiva*. Por ejemplo:

$$F(A,B,C) = \overline{A}\cdot B\cdot \overline{C} + \overline{A}\cdot B\cdot C + A\cdot B\cdot \overline{C} + A\cdot B\cdot C$$

Para simplificar la escritura en forma de suma canónica de productos, se usa una notación especial. Se elige un orden fijo para las variables en los minitérminos y a cada minitérmino se le asocia un número cuya representación binaria tiene n bits que resultan de considerar como 0 las variables complementadas y como 1 las variables no complementadas. Así la función booleana anterior se representaría como:

$$F(A,B,C) = m_2 + m_3 + m_6 + m_7 = \sum m(2,3,6,7)$$

I.1.2. Maxitérminos

Dada una función de n variables de entrada, si un término suma contiene cada una de las n variables, ya sea en forma complementada o no, es un maxitérmino. Si la función se representa como un producto de maxitérminos, se dice que la función tiene forma de *producto canónico de sumas* o que está en *forma normal conjuntiva*. Por ejemplo:

$$F(A,B,C) = (A+B+C)\cdot(A+B+\overline{C})\cdot(\overline{A}+B+C)\cdot(\overline{A}+B+\overline{C})$$

Análogamente al caso de la forma normal disyuntiva, podemos simplificar la expresión de la función, indicando los maxitérminos que incluye. Sin embargo, en este caso cada variable complementada corresponde a un 1 y viceversa, al contrario que antes:

$$F(A,B,C) = M_0\cdot M_1\cdot M_4\cdot M_5 = \prod M(0,1,4,5)$$

Ejercicio: dada la siguiente función, exprésala en forma de lista de minitérminos.

$$F(A,B,Q,Z) = \overline{A}\cdot\overline{B}\cdot\overline{Q}\cdot\overline{Z} + \overline{A}\cdot\overline{B}\cdot\overline{Q}\cdot Z + \overline{A}\cdot B\cdot Q\cdot\overline{Z} + \overline{A}\cdot B\cdot Q\cdot Z$$

Solución:

$$F(A,B,Q,Z) = m_0 + m_1 + m_6 + m_7 = \sum m(0,1,6,7)$$

Alternativamente, es posible emplear una tabla de verdad. En una tabla de verdad aparecen a la izquierda tantas columnas como entradas tenga la función y una columna para la salida. Cada fila representa uno de los posibles valores de entrada junto con su salida (la tabla de verdad de una función resulta de aplicar la ecuación lógica para todos y cada uno de los valores de entrada). Por tanto, completando todos los valores de la tabla, se tiene definida totalmente la función.

Ejemplo: la función lógica F tiene tres entradas A, B y C, y una salida. La salida es verdadera si exactamente dos entradas son verdaderas. A continuación se muestra la ecuación lógica, y la tabla de verdad para esta función se muestra en la tabla I.1.

$$F = ((A \cdot B) + (B \cdot C) + (A \cdot C)) \cdot \overline{(A \cdot B \cdot C)} \equiv F = (A \cdot B \cdot \overline{C}) + (A \cdot \overline{B} \cdot C) + (\overline{A} \cdot B \cdot C)$$

Entradas			Salida
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Tabla I.1: Tabla de verdad correspondiente a la función F.

Las funciones lógicas que operan sólo sobre 0's y 1's también son llamadas funciones de conmutación.

I.2. Simplificación de funciones mediante mapas de Karnaugh

La simplificación de las funciones lógicas es una meta importante. Su importancia radica en el hecho de que cuanto más sencilla sea la función, más fácil será construir el circuito equivalente. El objetivo de la simplificación es minimizar el costo de implementación de una función mediante componentes electrónicos, donde el costo depende del número y complejidad de los elementos necesarios para construirla.

Los mapas de Karnaugh constituyen un método sencillo y apropiado para la minimización de funciones lógicas, limitado en la práctica hasta cinco o seis variables. Como ya visteis en la asignatura de *Lógica*, un mapa de Karnaugh es una representación gráfica de una tabla de verdad y, por tanto, existe una asociación unívoca entre ambas. La tabla de verdad tiene una fila por cada minitérmino, mientras que el mapa de Karnaugh tiene una celda por cada minitérmino. De manera análoga, también existe una correspondencia unívoca entre las filas de la tabla de verdad y las celdas del mapa de Karnaugh si se utilizan los maxitérminos. En la figura I.3 se muestran los mapas de Karnaugh correspondientes a 2, 3 y 4 variables de entrada. En dicha figura, cada celda aparece etiquetada con su respectivo número de minitérmino, o lo que es lo mismo, con el número de fila de la tabla de verdad con el que se correspondería. Las etiquetas de las filas y las columnas se refieren a los valores de las variables.

La minimización de funciones sobre el mapa de Karnaugh se aprovecha del hecho de que sobre el mapa, los términos adyacentes desde el punto de vista lógico, también son adyacentes físicamente. Definimos los minitérminos adyacentes desde el punto de vista lógico como dos minitérminos que difieren sólo en una variable. Por ejemplo, $AB\overline{C}\overline{D}$ y $AB\overline{C}D$ son minitérminos de cuatro variables adyacentes lógicamente, ya que sólo difieren en la variable D.

Dos términos adyacentes pueden combinarse eliminando la variable en la que difieren, usando las propiedades distributiva y complementaria ($A \cdot B \cdot \overline{C} \cdot \overline{D} + A \cdot B \cdot \overline{C} \cdot D = A \cdot B \cdot \overline{C} \cdot (D + \overline{D}) = A \cdot B \cdot \overline{C}$). En el mapa indicamos los términos que se pueden combinar trazando un anillo alrededor de ellos. Estos términos al combinarse nos darán una expresión más sencilla, o sea, con menos variables.

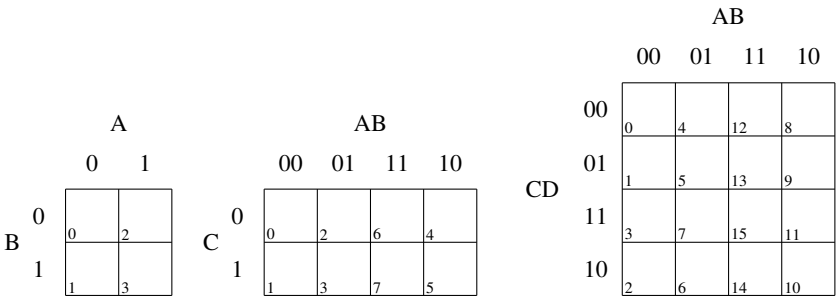
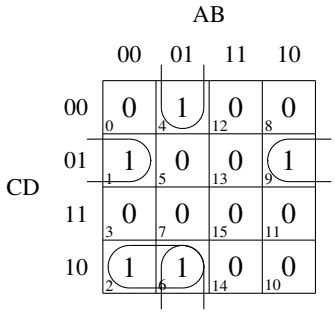


Figura I.3: Mapas de Karnaugh para diferentes números de variables.

Ejercicio: simplifica la función $F(A,B,C,D) = \sum m(1,2,4,6,9)$ mediante su mapa de Karnaugh.
Solución:



La expresión simplificada de la función queda:

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

En el paso 1 combinamos los minitérminos 1 y 9 que son adyacentes lógicamente. La variable que cambia es A que, por tanto, se elimina, quedando $\overline{B} \cdot \overline{C} \cdot D$. Del mismo modo procedemos en los pasos 2 y 3. En el paso 2 combinamos los minitérminos 4 y 6 que se diferencian en la variable C, y en el 3 los minitérminos 2 y 6 que difieren en la variable B.

Criterios para la simplificación de funciones con mapas de Karnaugh

Hay cinco reglas que debemos recordar para poder simplificar funciones representadas sobre mapas de Karnaugh:

1. Cada cuadrado (minitérmino) sobre una mapa de Karnaugh de dos variables tiene dos cuadrados (minitérminos) adyacentes lógicamente; cada cuadrado sobre un mapa de Karnaugh de tres variables, tiene tres cuadrados adyacentes, etc. En general, cada cuadrado en un mapa de Karnaugh de n variables tiene n cuadrados adyacentes lógicamente, de modo que cada par de cuadrados adyacentes difiere precisamente en una variable.

2. Al combinar los cuadrados en un mapa de Karnaugh, agruparemos un número de minitérminos que sea potencia de dos. Al agrupar dos cuadrados eliminamos una variable, al agrupar cuatro cuadrados eliminamos dos variables, etc. En general, al agrupar 2^n cuadrados eliminamos n variables.

3. Debemos agrupar tantos cuadrados como sea posible; cuanto mayor sea el grupo, habrá un número menor de literales en el término producto resultante.

4. Debemos formar el menor número posible de grupos que cubran todos los cuadrados (minitérminos) de la función. Un minitérmino está cubierto si está incluido al menos en un grupo. Si hay menos grupos, será menor el número de términos de la función minimizada. Podemos utilizar cada término tantas veces como sea necesario en los pasos 4 y 5; sin embargo, debemos usarlo al menos una vez. Tan pronto hayamos utilizado todos los minitérminos al menos una vez nos detenemos.

5. La combinación de cuadrados en el mapa se hará siempre empezando por los minitérminos que tienen menor número de cuadrados adyacentes (los más solitarios en el mapa). Los minitérminos con varios términos adyacentes ofrecen más combinaciones posibles y, por tanto, deben combinarse más adelante en el proceso de minimización.

I.2.1. Terminología para la minimización de funciones lógicas

Los términos que vamos a exponer son útiles en los procedimientos de simplificación de funciones lógicas, pero no sólo con mapas de Karnaugh sino también con otras técnicas más generales de minimización.

Implicante: término producto que puede servir para cubrir los minitérminos de una función.

Implicante primo: implicante que no es parte de ningún otro implicante de la función. En el mapa de Karnaugh, un implicante primo equivale a un conjunto de cuadrados que no es subconjunto de algún conjunto con un mayor número de cuadrados.

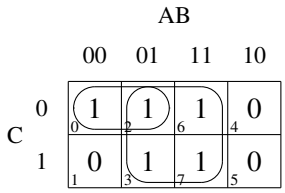
Implicante primo esencial: implicante primo que cubre al menos un minitérmino que no está cubierto por algún otro implicante primo.

Cubierta: conjunto de implicantes primos tal que todos los minitérminos de la función están contenidos en al menos un implicante primo. La cubierta de una función debe incluir, como mínimo, todos los implicantes primos esenciales posibles.

Ejercicio: identifica sobre la siguiente función, mediante su mapa de Karnaugh, los implicantes, los implicantes primos, los implicantes primos esenciales y la cubierta:

$$F(A,B,C) = \sum m(0,2,3,6,7)$$

Solución:



- *Implicante* $\rightarrow$ 11 implicantes: 5 minitérminos, 5 grupos de dos minitérminos y 1 grupo de 4 minitérminos.
- *Implicante primo* $\rightarrow$ 2 implicantes primos, B y $\overline{A}\cdot\overline{C}$.
- *Implicante primo esencial* $\rightarrow$ 2 implicantes primos esenciales, B y $\overline{A}\cdot\overline{C}$.
- *Cubierta* $\rightarrow \{B, \overline{A}\cdot\overline{C}\}$.

I.2.2. Algoritmo de minimización mediante mapas de Karnaugh

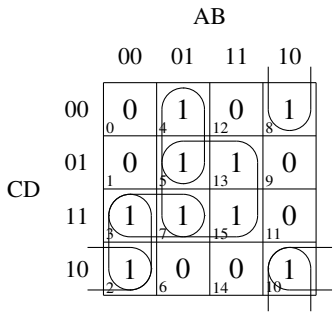
1. Identificar los implicantes primos. Para esto se busca obtener los grupos con mayor cantidad de unos adyacentes. Los grupos deben contener un número de unos que sea potencia de 2.
2. Identificar todos los implicantes primos esenciales.
3. La expresión mínima se obtiene seleccionando todos los implicantes primos esenciales y el menor número de implicantes primos para cubrir los minitérminos no incluidos en los implicantes primos esenciales.

Ejercicio: utiliza el mapa de Karnaugh para simplificar la función:

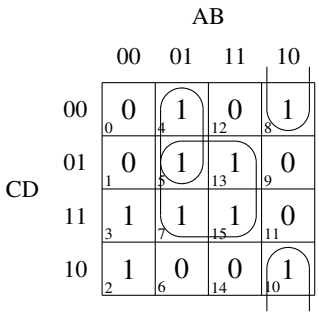
$$F(A,B,C,D) = \sum m(2,3,4,5,7,8,10,13,15)$$

Solución:

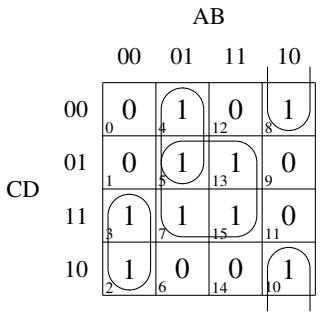
Implicantes primos:



Implicantes primos esenciales:



Cubierta:

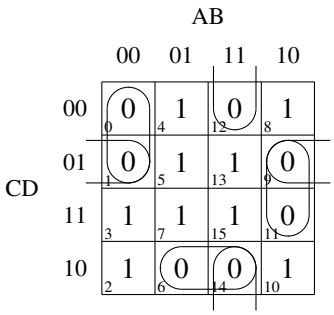


La función minimizada queda:

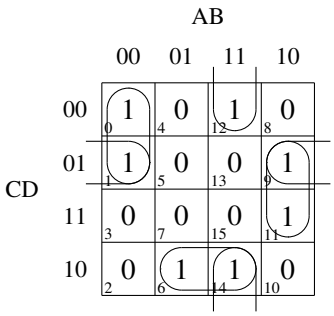
$$F(A,B,C,D) = \overline{A}\cdot\overline{B}\cdot C + \overline{A}\cdot B\cdot\overline{C} + B\cdot D + A\cdot\overline{B}\cdot\overline{D}$$

I.2.3. Simplificación por ceros

A veces puede ser interesante agrupar los ceros en el Mapa de Karnaugh en lugar de los unos, ya se deba a que sea más sencilla dicha agrupación o porque nos interese obtener una expresión simplificada en forma de producto de sumas. Veamos cómo se realiza la simplificación en este caso utilizando la misma función que en el apartado anterior. El primer paso es dibujar el Mapa de Karnaugh marcando todos los implicantes primos:



El problema que nos encontramos es que no sabemos cómo se eliminan las variables que cambian en las agrupaciones, por lo que para obtener las reglas a seguir, escribiremos en este caso concreto (más adelante no hará falta) el Mapa de Karnaugh de la función $\overline{F}$:



Esta función sí sabemos simplificarla siguiendo los pasos habituales, con lo que la expresión simplificada será:

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

Teniendo en cuenta que $\overline{\overline{F}} = F$, la expresión simplificada de F será, por lo tanto (aplicando las leyes de De Morgan):

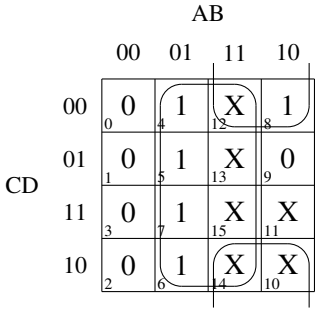
$$\begin{aligned} F = \overline{\overline{F}} &= \overline{\overline{A} \cdot \overline{B} \cdot \overline{C} + B \cdot C \cdot \overline{D} + A \cdot B \cdot \overline{D} + A \cdot \overline{B} \cdot D} \\ &= (A + B + C) \cdot (\overline{B} + \overline{C} + D) \cdot (\overline{A} + \overline{B} + D) \cdot (\overline{A} + B + \overline{D}) \end{aligned}$$

Si observamos el resultado obtenido y vemos el primer Mapa de Karnaugh, observamos que la variable que se elimina sigue siendo la que cambia. De las variables que no cambian, éstas aparecerán negadas cuando en las casillas correspondientes aparezcan a 1 y sin negar cuando aparezcan a 0. Por último, la expresión obtenida es de producto de sumas, es decir, la expresión que representa una agrupación de ceros es una suma de variables negadas o no. Estas serán las reglas que utilizaremos a la hora de simplificar por ceros (sin necesidad ya de dibujar el Mapa de Karnaugh de la función $\overline{F}$).

I.2.4. Salidas no determinadas

Hasta ahora hemos tratado con funciones lógicas que, para todas las combinaciones posibles a la entrada (2^n, siendo n el número de variables de entrada) toman un valor binario determinado en la salida, ya sea 0 o 1. Sin embargo, existen situaciones en las cuales queremos diseñar un circuito en el cual no vamos a utilizar todas las condiciones de entrada, sino que algunas de ellas van a quedar sin utilizar. En ese caso, decimos que a aquellas combinaciones de entrada para las cuales no nos importa la salida (puesto que, en su operación normal, al circuito nunca le presentaremos esas entradas) les corresponde una salida no determinada. Se indicarán en la tabla de verdad y mapa de Karnaugh correspondientes colocando una X, en lugar de un 0 o un 1.

Un ejemplo de utilización se expone a continuación. Imaginemos que queremos simplificar una función booleana de 4 variables de entrada (A, B, C y D) y una de salida (F(A, B, C, D)), que valga uno cuando la ristra de 4 bits de entrada, interpretada en binario natural, esté entre 4 y 8 (ambos inclusive), valga cero para el resto de casos entre 0 y 9 (0, 1, 2, 3, 9) y que nos de igual la salida si está entre 10 y 16. La tabla de verdad correspondiente sería la indicada en tabla I.2.



Entradas				Salida
A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

Tabla I.2: Tabla de verdad correspondiente a la función F con salidas indeterminadas.

Las condiciones de «no importa» (X) pueden ser utilizadas en la simplificación de la función, con la ventaja de que, como nos da igual que la salida efectiva sea 0 o 1 en la función simplificada, podemos decidir cubrir las X o no en el mapa de Karnaugh correspondiente. La únicas obligaciones que tenemos son:

- Que se cubran todos los unos.
- Que no se cubra ningún cero.

Las X pueden utilizarse para hacer los grupos de unos más grandes (así el producto obtenido tendrá menos variables), o bien simplemente dejarse sin cubrir. En el ejemplo que nos ocupa, una solución sería $F(A,B,C,D) = B + A \cdot \overline{D}$. Obsérvese como las X de los minitérminos 12, 13, 14 y 15 se aprovechan para hacer un grupo de 8 unos, correspondiente al implicante primo B, y los términos 10, 12 y 14 se aprovechan para formar un grupo de 4 unos, correspondiente al implicante primo A·D'. La X del minitérmino 11, sin embargo, no se cubre en este caso, lo que significaría que la salida de la función simplificada obtenida valdría cero para esa combinación de entrada (la 1011). En general, las condiciones de no importa suelen resultar bastante útiles para obtener mayores simplificaciones.

Circuitos combinacionales comunes

En este apartado estudiaremos los circuitos combinacionales, que son aquellos en los que la salida en cada instante depende únicamente del valor de las entradas en ese instante, sin importar el comportamiento anterior del circuito. Como veremos posteriormente, la diferencia con el otro tipo básico de circuitos, llamados circuitos secuenciales, radica en que en estos últimos el valor de la salida no dependerá sólo del valor de las entradas en ese instante, sino también de la historia previa del circuito (por ello se dice que ese otro tipo de circuitos poseen memoria).

Veamos a continuación algunos de los circuitos combinacionales más importantes, comenzando por las puertas lógicas.

II.1. Puertas lógicas básicas

Los bloques lógicos se construyen a partir de puertas lógicas que implementan las funciones lógicas básicas, y que se utilizan como elementos constructivos de partida para implementar la mayor parte de las funciones que desempeña la circuitería del ordenador. Por ejemplo, una puerta AND implementa el producto lógico y una puerta OR implementa la suma lógica. Como AND y OR son conmutativas y asociativas, una puerta AND o una puerta OR pueden tener múltiples entradas, siendo la salida igual al producto lógico o suma lógica de todas sus entradas. La negación lógica se implementa con una puerta NOT, llamada también inversor, que tiene una única entrada. La representación simbólica de estas tres puertas lógicas básicas puede verse en la siguiente figura:

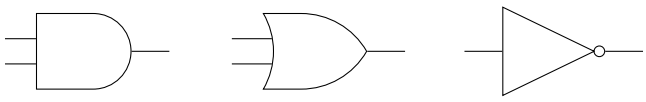


Figura II.1: Símbolo de las puertas lógicas básicas (AND, OR, NOT).

En lugar de dibujar los inversores explícitamente, una práctica común es añadir burbujas a las entradas o salidas de una puerta para lograr que el valor lógico en la línea de entrada o en la línea de salida se invierta. Por ejemplo, la figura II.2, muestra el diagrama lógico para la función $F = \overline{(\overline{A}+B)}$, utilizando inversores explícitamente a la izquierda y utilizando entradas y salidas con burbujas a la derecha.

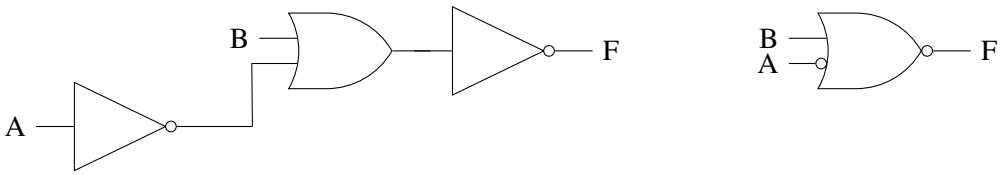


Figura II.2: Implementación con puertas lógicas de $F = \overline{(\overline{A}+B)}$, utilizando explícitamente inversores (izq.) y utilizando entradas y salidas con burbujas (der.).

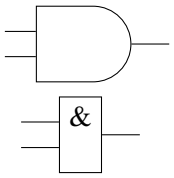
Cualquier función lógica puede ser construida utilizando puertas AND, puertas OR e inversores. Por otro lado, es posible realizar todas las funciones lógicas utilizando un solo tipo de puertas, si esa puerta es inversora. Las dos puertas anteriores invertidas se denominan NAND y NOR. Las puertas NAND y NOR se denominan universales, ya que cualquier función lógica puede ser construida utilizando este tipo de puertas. Como ejemplo, veamos cómo puede expresarse la función $F(A,B) = A+B$ usando sólo la operación NAND (AND negado):

$$A+B = (por\,involución) = \overline{\overline{(A+B)}} = (por\,Ley\,de\,DeMorgan) = \overline{(\overline{A}\cdot\overline{B})}$$

donde las respectivas negaciones de A y B pueden hacerse con una puerta NAND de sólo una entrada, y el posterior producto negado con una NAND de dos entradas. Análogamente, pueden encontrarse equivalencias

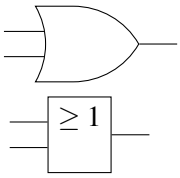
para cada una de las puertas AND, OR y NOT sólo en función de NAND o NOR (se propone como sencillo ejercicio), con lo que quedaría demostrada su universalidad. En la figura II.3 se detalla la tabla de verdad y el símbolo lógico de cada una de las puertas mencionadas.

A	B	$A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1



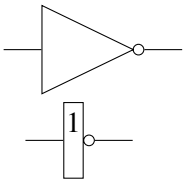
(a) Puerta AND de 2 entradas.

A	B	$A + B$
0	0	0
0	1	1
1	0	1
1	1	1



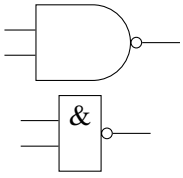
(b) Puerta OR de 2 entradas.

A	$\overline{A}$
0	1
1	0



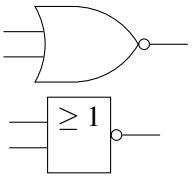
(c) Puerta NOT.

A	B	$\overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0



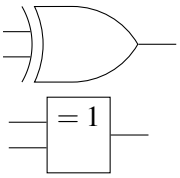
(d) Puerta NAND de 2 entradas.

A	B	$\overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0



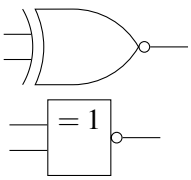
(e) Puerta NOR de 2 entradas.

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0



(f) Puerta XOR de 2 entradas.

A	B	$\overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1



(g) Puerta XNOR de 2 entradas.

Figura II.3: Símbolo y tabla de verdad de diferentes puertas lógicas.

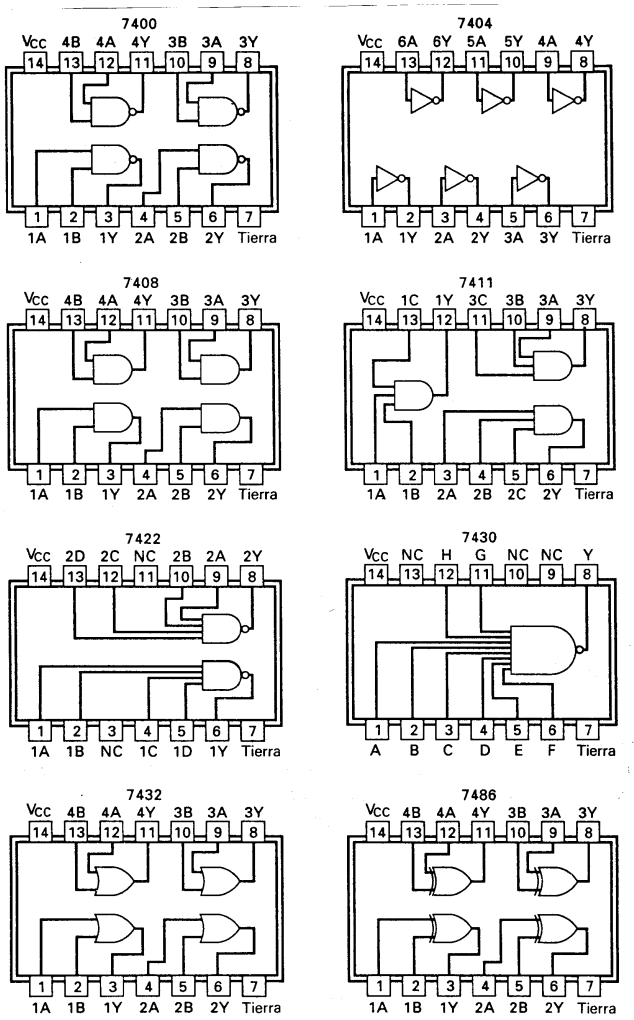
Tan sólo comentar el caso de la operación de OR exclusivo (XOR), que se define de manera funcional como $XOR(A,B) = A \oplus B = A \cdot \overline{B} + \overline{A} \cdot B$. Como vemos, la salida es 1, si y sólo si, sus entradas son diferentes. El OR exclusivo recibe su nombre debido a su relación con la puerta OR. Ambas difieren en la combinación de entradas A = 1 y B = 1. El OR exclusivo da como salida 0, mientras que la puerta OR produce un 1, por lo que se le llama OR inclusivo. Esta operación cumple las propiedades conmutativa y asociativa. En general, para cualquier número de entradas, la salida de una puerta OR exclusivo es la suma módulo dos de sus entradas. Es decir, la salida es 1 si el número de entradas que valen 1 es impar, y 0 en caso contrario.

II.1.1. Circuitos integrados

Las puertas lógicas no se fabrican ni se venden individualmente, sino en unidades llamadas circuitos integrados o chips. Un chip no es más que una pieza de silicio rectangular sobre la que se depositan algunas puertas lógicas. Las pastillas pueden dividirse en varios grupos, en base al número de puertas que contienen:

- SSI (circuitos integrados a escala pequeña): 1 a 10 puertas.

- MSI (circuitos integrados a escala media): 10 a 100 puertas.
- LSI (circuitos integrados a escala grande): 100 a 100.000 puertas.
- VLSI (circuitos integrados a escala muy grande): Más de 100.000 puertas.



Algunas pastillas SSI. Disposiciones de patas extraídas de *The TTL Data Book for Design Engineers* (© D.R. por Texas Instruments Incorporated, 1976).

Figura II.4: Algunas pastillas SSI.

donde SSI, MSI, LSI y VLSI son las iniciales, respectivamente, de *Short*, *Medium*, *Large* y *Very Large Scale of Integration*. En la figura II.4 podemos ver diferentes pastillas SSI que implementan algunas de las puertas lógicas básicas que hemos estudiado hasta ahora.

II.2. Retardos

Puesto que a la hora de implementar funciones con puertas, en última instancia éstas son dispositivos físicos reales, existen una serie de restricciones sobre el funcionamiento de cada puerta, de las cuales la más importante es el retardo temporal que introducen. Este retardo se puede definir como el tiempo que transcurre entre el instante en que un circuito tiene disponibles los valores de señal deseados a la entrada y el instante en que la señal de salida se estabiliza al valor deseado. Por ejemplo, si decimos que las puertas AND y OR tienen un retardo de 10 ns (nanosegundos), las NOT de 5 ns, y los retardos introducidos por las conexiones son despreciables, el circuito de la figura II.5 tendrá un retardo total de 25 ns (puesto que la salida del NOT

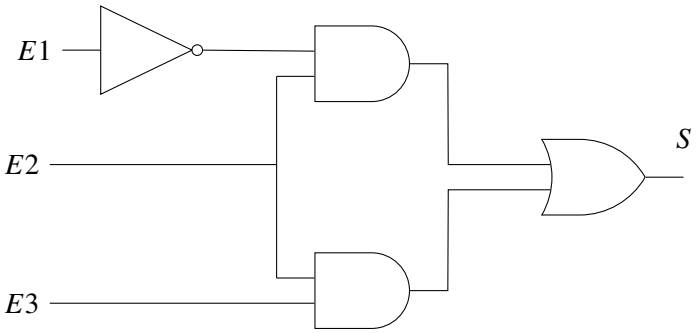


Figura II.5: Ejemplo de retardo total de un circuito.

se obtendrá en 5 ns, la del AND superior en 5+10=15 y la del OR en 5+10+10=25 ns, mientras que el AND inferior funcionará en paralelo al superior y tardará sólo 10 ns en efectuar su operación).

II.3. Implementación de funciones lógicas con puertas NAND/NOR

Una de las características de las puertas NAND (NOR) es su carácter de puerta *universal*. Es decir, es posible implementar cualquier función lógica utilizando únicamente puertas NAND (NOR). La demostración es sencilla, basta con demostrar cómo se implementan las puertas AND, OR y NOT utilizando únicamente puertas NAND (NOR). La figura II.6 muestra la implementación de las puertas AND, OR y NOT utilizando únicamente puertas NAND (la implementación con puertas NOR sería similar).

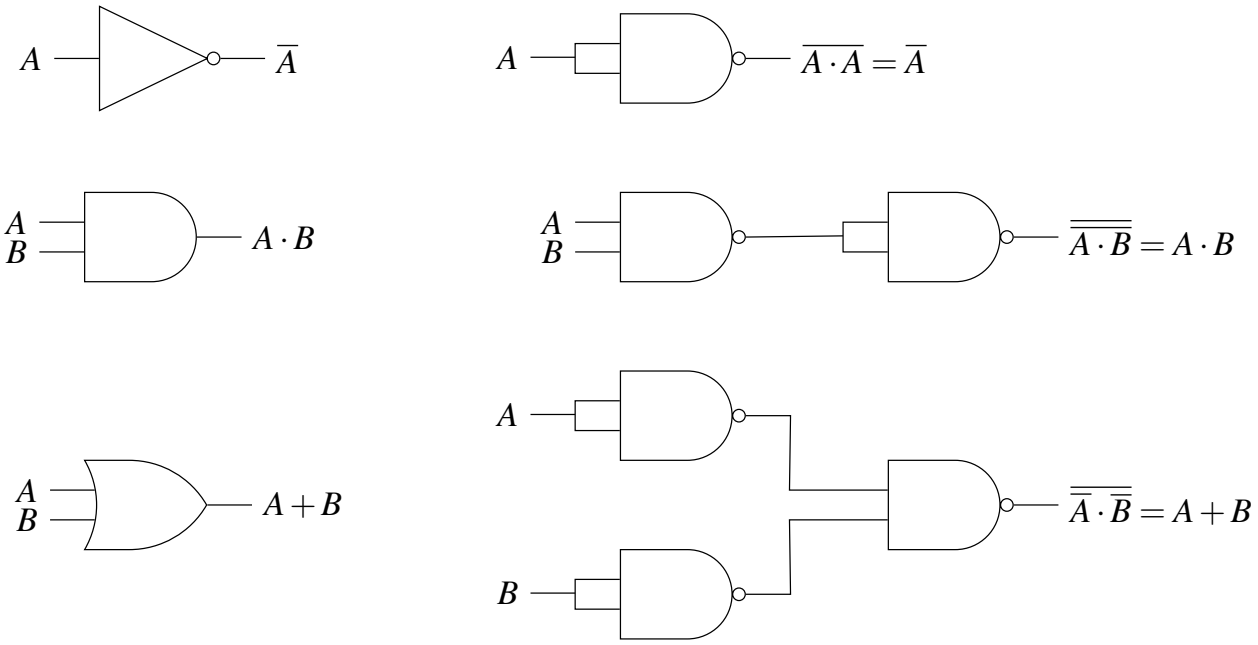


Figura II.6: Equivalencia entre las puertas AND, OR y NOT y la puerta NAND.

Una vez vista la universalidad de ambas puertas veamos cómo podemos implementar una función lógica utilizando únicamente puertas NAND (NOR). La situación más sencilla es la implementación con puertas NAND ya que una vez obtenida la función mínima expresada en forma de suma de productos (suma de minitérminos) sólo debemos negar la expresión dos veces y aplicar las leyes de De Morgan a la negación interior para obtener una expresión en donde sólo aparezcan puertas NAND. Así, por ejemplo, dada la función $F(A,B,C,D) = \sum m(2,3,4,5,7,8,10,13,15)$ cuya minimización ya vimos que era:

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

Si a continuación negamos dos veces y aplicamos De Morgan obtenemos:

$$\begin{aligned} F(A,B,C,D) &= \overline{\overline{\overline{A}\cdot\overline{B}\cdot C + \overline{A}\cdot B\cdot\overline{C} + B\cdot D + A\cdot\overline{B}\cdot\overline{D}}} \\ &= \overline{\overline{(\overline{A}\cdot\overline{B}\cdot C)}\cdot\overline{(\overline{A}\cdot B\cdot\overline{C})}\cdot\overline{(B\cdot D)}\cdot\overline{(A\cdot\overline{B}\cdot\overline{D})}} \end{aligned}$$

que puede implementarse directamente mediante puertas NAND.

En el caso de querer implementar la misma función utilizando puertas NOR sería conveniente obtener una expresión de la función en forma de producto de sumas (producto de maxitérminos), ya que, una vez que estemos en dicha situación, sólo necesitaremos repetir el proceso anterior (negar la expresión dos veces y aplicar De Morgan a la negación interior) para obtener de forma directa la expresión deseada. Por tanto, simplificaremos agrupando los ceros de la función para obtener una expresión en producto de maxitérminos. El resultado obtenido, como ya vimos, es:

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

Si a continuación negamos dos veces obtenemos y aplicamos De Morgan obtenemos:

$$\begin{aligned} F(A,B,C,D) &= \overline{\overline{(A+B+C)\cdot(\overline{B}+\overline{C}+D)\cdot(\overline{A}+\overline{B}+D)\cdot(\overline{A}+B+\overline{D})}} \\ &= \overline{\overline{(A+B+C)}+\overline{(\overline{B}+\overline{C}+D)}+\overline{(\overline{A}+\overline{B}+D)}+\overline{(\overline{A}+B+\overline{D})}} \end{aligned}$$

que puede implementarse directamente mediante puertas NOR.

II.4. Bloques lógicos

Conforme construimos funciones lógicas cada vez más complejas (involucrando varias operaciones AND, OR, NOT, etc, en distintas etapas), se hace inviable representar gráficamente el diagrama de conexión completo con todas las puertas resultantes. Además, en general, el método de diseño de circuitos será jerárquico, es decir, iremos construyendo sencillos circuitos capaces de realizar alguna tarea simple, para después, utilizando estos bloques elementales, construir circuitos cada vez más complejos. Si tenemos clara la función y estructura interna de cada bloque, no hace falta dibujar todas las puertas que componen el mismo, sino solamente un cuadro con el nombre del bloque, y sus entradas y salidas, como se muestra en la figura II.7.

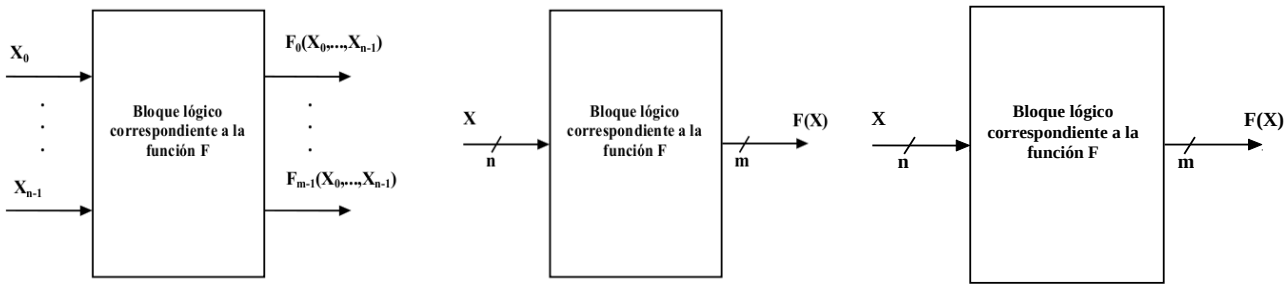


Figura II.7: (Izquierda) Bloque lógico correspondiente a una función de n bits de entrada y m de salida. (Derecha) A veces, si las entradas y/o salidas están relacionadas entre sí, formando una secuencia de un determinado número de bits (n y m en la figura), emplearemos la abreviatura gráfica mostrada.

En los siguientes apartados, donde se construyen algunos circuitos básicos con diversas utilidades, y que serán empleados posteriormente en el diseño de un procesador, se verán numerosos ejemplos del uso de estos bloques para simplificar los diagramas obtenidos.

II.5. Codificadores y decodificadores

Un *codificador* es un circuito lógico combinacional con 2^n líneas de entrada y n líneas de salida, tal y como se muestra en la figura II.8 (en este ejemplo hay 8 entradas y 3 salidas). Cuando una de las entradas es activada y el resto se dejan a cero, en las n líneas de salida aparece el número de entrada activa codificado en binario. Por ejemplo, si se activa la entrada E3, la salida será (S2, S1, S0)=(0, 1, 1), puesto que 011 en binario es 3. La generalización para construir un codificador con otro número de entradas es trivial, conectando cada entrada n a una puerta OR de salida si y sólo si el bit que ocupa la posición correspondiente a dicha puerta de salida está a uno al expresar n en binario.

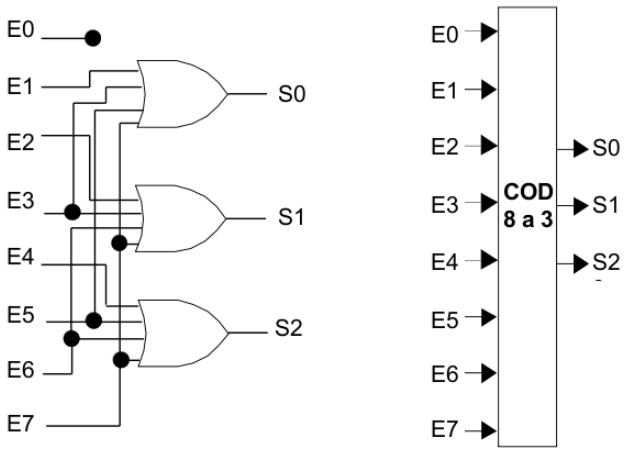


Figura II.8: Implementación con puertas y diagrama lógico equivalente de un codificador de 8 a 3 bits.

La implementación anterior sólo permite que una de las entradas esté activa simultáneamente, como se indica en la siguiente tabla de verdad:

E0	E1	E2	E3	E4	E5	E6	E7	S2	S1	S0
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Inversamente, un *decodificador* es un circuito lógico combinacional con n líneas de entrada y 2^n líneas de salida, tal y como se muestra en la figura II.9.

En este caso, interesa que para cada posible valor de las líneas de entrada, una y sólo una de las señales de salida tenga el valor lógico 1. Por tanto, podemos considerar el decodificador n a 2^n como un generador de minitérminos, donde cada salida corresponde precisamente a un minitérmino diferente. Si en la entrada, por ejemplo, colocamos la secuencia 110, entonces S6 valdrá 1, mientras que el resto de salidas valdrá 0. Al igual que ocurría con los codificadores, la extensión a mayor número de entradas es inmediata, mediante la simple adición de una puerta AND por cada salida n, a la cual se conectan todas y cada una de las entradas C_i, negadas si al escribir n en binario el bit i es 0, y sin negar en caso contrario.

Los decodificadores se usan para tareas tales como seleccionar una palabra de memoria, dada su dirección (lo veremos en el tema siguiente) o convertir códigos, por ejemplo, de binario a decimal. La siguiente tabla de verdad resume el funcionamiento del anterior decodificador:

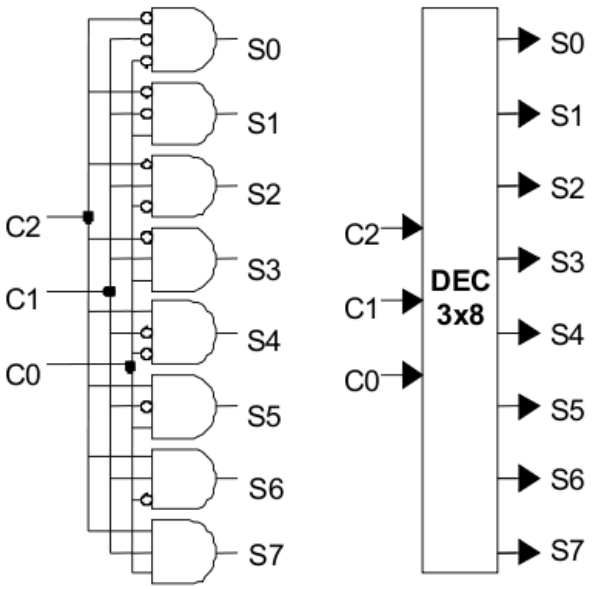


Figura II.9: Diagrama funcional de un decodificador 3 a 8.

C2	C1	C0	S0	S1	S2	S3	S4	S5	S6	S7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Tal y como se indicó anteriormente, un decodificador de n a 2^n se puede ver como un generador de minitérminos, donde cada salida corresponde precisamente a un minitérmino diferente. Así, podemos utilizar un decodificador de n entradas junto a una puerta OR del tamaño adecuado para implementar cualquier función booleana de n variables. Por ejemplo, si quisiéramos implementar la función de cuatro variables

$$F(A,B,C,D) = \sum m(2,3,4,5,7,8,10,13,15),$$

sólo tendríamos que utilizar un decodificador de 4 a 16, conectando las 4 variables a las 4 entradas del decodificador y conectando a la puerta OR las salidas 2, 3, 4, 5, 7, 8, 10, 13 y 15 del decodificador (ver figura II.10).

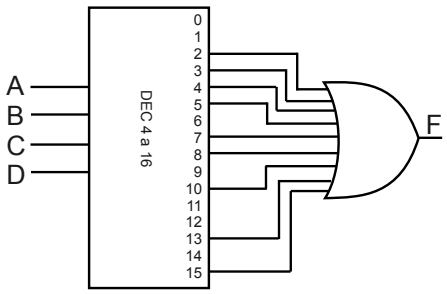


Figura II.10: Implementación de una función booleana mediante el uso de un decodificador.

II.6. Multiplexores

Un multiplexor, también llamado selector de datos, es un dispositivo que selecciona una de varias líneas de entrada para que aparezca en una única línea de salida. La entrada de datos que se selecciona viene indicada por unas entradas especiales, llamadas de control. La figura II.11 (izquierda) muestra el esquema del multiplexor más sencillo, de 2x1 (dos entradas y una salida), con dos entradas de datos y una de control, que selecciona cuál de las dos entradas de datos queremos que se transfiera a la salida. Siguiendo la misma técnica que con los decodificadores, es fácil extender los multiplexores para mayor número de entradas. La figura II.11 (derecha) muestra un multiplexor 4x1.

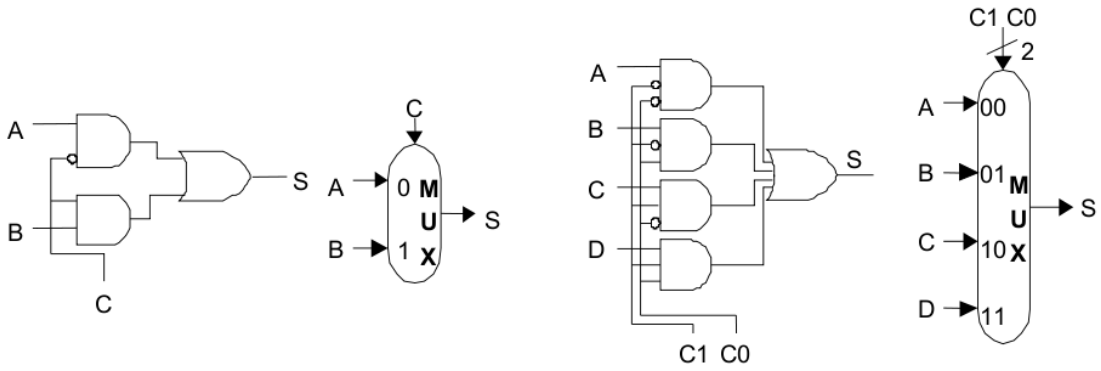


Figura II.11: Implementaciones y bloques lógicos de dos tipos de multiplexores de un bit de ancho: (Izquierda) Multiplexor 2x1, con entradas de datos A y B y de control C. (Derecha) Multiplexor 4x1, con entradas de datos A, B, C y D y de control C1 y C0.

En general, en un multiplexor con n líneas de entrada y una única salida, se determina cuál de las líneas de entrada $(D_{n-1}D_{n-2}D_{n-3}...D_1D_0)$ se conecta a la única línea de salida (Y) mediante un código de selección $(S_{k-1}S_{k-2}S_{k-3}\cdots S_1S_0)$ donde necesariamente $n = 2^k$. Así, por ejemplo, para un multiplexor con $n = 4$ (y por tanto $k = 2$), la ecuación lógica de la salida Y queda:

$$Y = (S1' \cdot S0') \cdot D0 + (S1' \cdot S0) \cdot D1 + (S1 \cdot S0') \cdot D2 + (S1 \cdot S0) \cdot D3$$

Además de en el número de entradas, otra extensión interesante de los multiplexores es en la anchura de los datos seleccionados. Así, en general hablaremos de un multiplexor de $2^n x1$ de m bits de anchura, si es capaz de seleccionar entre 2^n palabras de m bits cada una en función de una señal de control de n bits. En la figura II.12 se muestra cómo crear un multiplexor 2x1 de palabras de 32 bits, a partir de multiplexores elementales de 1 bit. Usando esta misma técnica pueden conseguirse multiplexores con la anchura y el número de entradas deseadas.

Un multiplexor con n bits de control puede servir también para implementar una función cualquiera con n entradas, simplemente conectando las entradas de la función a las entradas de control del multiplexor, y las entradas de datos de éste a unos y/o ceros, según la tabla de verdad de la función. Así, la implementación de la función:

$$F(A,B,C,D) = \sum m(2,3,4,5,7,8,10,13,15)$$

utilizando un multiplexor de 16 a 1 sería la mostrada en la figura II.13.

Por otro lado, también es posible implementar funciones de más de n variables utilizando un multiplexor con n bits de control. En este caso elegimos n de las variables para ser conectadas a las n entradas de control del multiplexor. El resto de variables aparecerán en las entradas de datos en forma de función booleana. Veamos el ejemplo concreto en donde utilizamos un multiplexor con n bits de control para implementar una función de $n+1$ variables.

Para la función implementada en la figura II.13, el multiplexor sería ahora de 8 a 1 (tres variables de control). Debemos seleccionar qué variables vamos a utilizar como control y cuál se quedará fuera. Supongamos que las variables de control elegidas son A, B y C, dejando fuera la variable D. Entonces cada entrada de datos

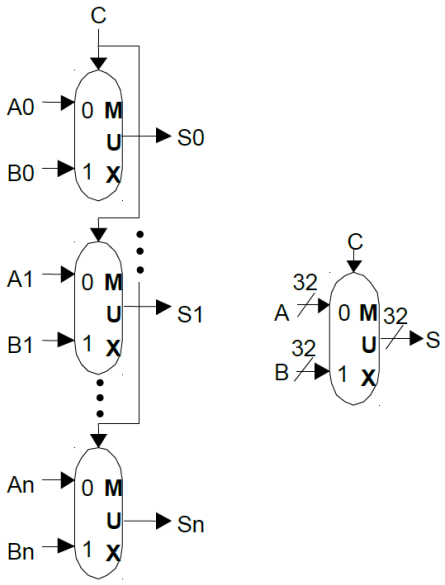


Figura II.12: Implementación de un MUX 2x1 para palabras de 32 bits y bloque lógico correspondiente. Obsérvese que es necesario sólo un bit de control para el multiplexor.

del multiplexor depende del contenido de dos casillas adyacentes en el Mapa de Karnaugh. Así, la entrada de datos 0 debe mostrar los valores de las casillas 0 y 1, la entrada 1 los valores de las casillas 2 y 3, etc. En la figura II.14(izq) se puede observar como quedan agrupadas las casillas en el Mapa de Karnaugh.

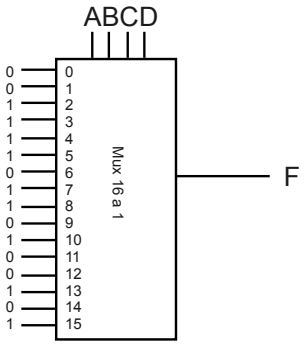


Figura II.13: Implementación de una función booleana de 4 variables mediante el uso de un multiplexor 16 a 1.

Con cada entrada representando a dos casillas existen cuatro posibles opciones que determinan los valores que pondremos en cada una de las entradas del multiplexor. Si en ambas casillas encontramos un 0, la entrada correspondiente será 0. De igual manera si encontramos un 1, asignaremos un 1 a esa entrada. Por último, si las casillas tienen valores distintos, procederemos de la siguiente manera. Observaremos el valor de la variable excluida en esa agrupación de casillas. Si cuando dicha variable vale 0 la casilla contiene un 0 y cuando vale un 1 la casilla contiene un 1, conectaremos esa entrada del multiplexor con la variable excluida (D en nuestro caso). Si la situación es la inversa, asignaremos a dicha entrada la variable excluida negada (el valor $\overline{D}$ en nuestro caso). En la figura II.14(der) se puede observar la implementación final de la función F.

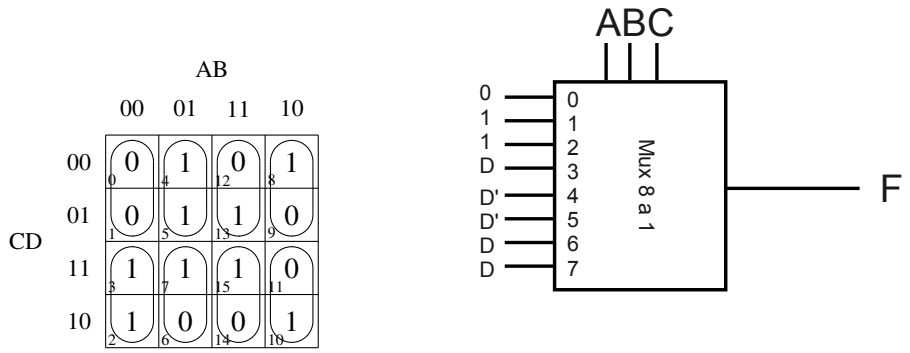


Figura II.14: Implementación de una función booleana de 4 variables mediante el uso de un multiplexor 8 a 1 (*D* es la variable excluida).

II.7. Memorias ROM y arrays lógicos programables

II.7.1. Memorias ROM

La ROM (*Read Only Memory*, memoria de sólo lectura) es una forma de lógica combinacional estructurada que sirve para implementar un conjunto de funciones lógicas. Una ROM se llama memoria porque contiene un conjunto de posiciones que pueden ser leídas; sin embargo, el contenido de estas posiciones es fijo, habitualmente desde el instante en que se crea la ROM. Así, la salida del circuito ante una entrada determinada no depende de su evolución anterior (como ocurrirá en una memoria RAM), sino que depende sólo de la entrada (esto es, la dirección) que se le proporcione en un momento dado. Se trata, pues, de un circuito combinacional.

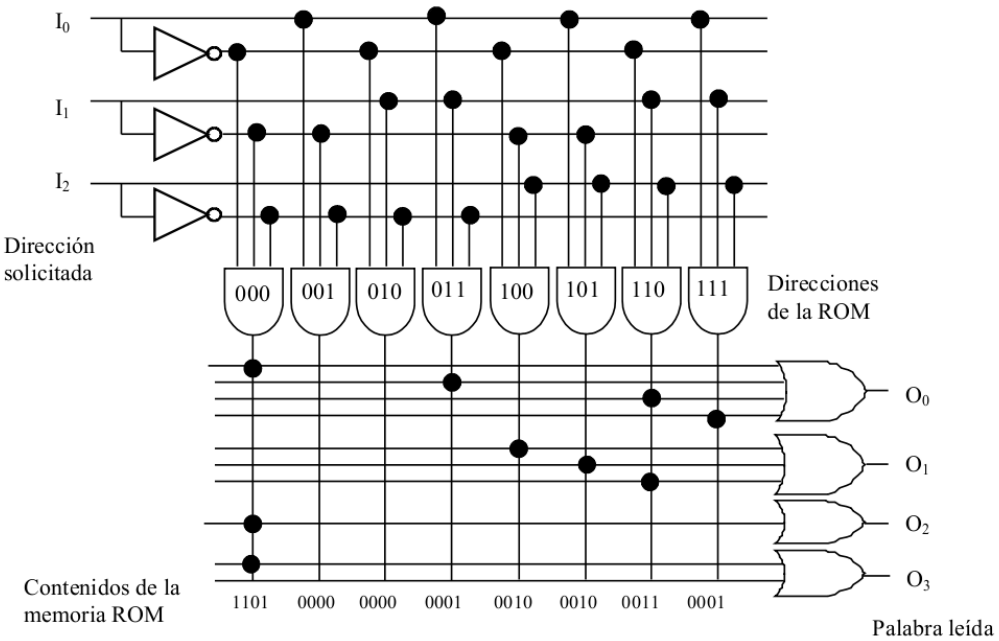


Figura II.15: Esquema de una memoria ROM con 8 posiciones de 4 bits cada una (3 bits de dirección, y anchura de datos 4).

Una ROM tiene un conjunto de líneas de entrada de dirección y un conjunto de salidas. El número de elementos direccionables de la ROM determina el número de líneas de dirección: si la ROM contiene 2^m elementos direccionables, o altura de la ROM, entonces hay m líneas de dirección, o sea, m líneas de entrada.

El número de bits de cada elemento direccionable es igual al número de bits de salida y a veces se denomina anchura de la ROM. El número total de bits de la ROM es igual a la altura por la anchura. La altura y la anchura a veces se denominan forma de la ROM.

Una ROM puede implementar directamente una colección de funciones lógicas a partir de la tabla de verdad. Por ejemplo, si hay n funciones con m entradas, necesitamos una ROM con m líneas de dirección y 2^m elementos, teniendo cada uno n bits de ancho. Las entradas de la parte de entrada de la tabla de verdad representan las direcciones de los elementos contenidos en la ROM, mientras que la parte correspondiente a las salidas de la tabla de verdad constituye el contenido de la ROM para cada dirección. Si se organiza la tabla de verdad para que la secuencia de entradas de la tabla de verdad sea la secuencia de números binarios (tal y como hemos hecho en las tablas de verdad hasta ahora), entonces la parte de salida da también el contenido de la ROM en orden.

El esquema de implementación de una ROM consta de dos niveles de puertas, o planos, a los cuales se les llama plano AND y plano OR, por el tipo de puertas que se utilizan en cada nivel. En general, una ROM de m bits de entrada (dirección) y n bits de anchura (es decir, capaz de guardar 2^m palabras de n bits cada una), tendrá un total de 2^m puertas AND, cada una con m entradas, y n puertas OR, cada una con, como máximo, 2^m entradas. El plano de conexión AND será siempre el mismo para todas las ROM con la misma capacidad, mientras que el OR dependerá de los contenidos concretos de la memoria de sólo lectura que estamos construyendo. La figura II.15 muestra la construcción de una ROM de ejemplo $2^3 \times 4$ (8 posiciones de 4 bits), que implementa la siguiente tabla de verdad:

I2	I1	I0	O3	O2	O1	O0
0	0	0	1	1	0	1
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	1
1	0	0	0	0	1	0
1	0	1	0	0	1	0
1	1	0	0	0	1	1
1	1	1	0	0	0	1

II.7.2. PROM, EPROM, EEPROM

La memoria programable exclusiva para lectura (PROM) es similar a la memoria ROM pero con la particularidad de que el plano OR es programable. El plano AND, al igual que en la memoria ROM, genera los 2^m posibles minitérminos para sus m entradas. El plano OR permite incluir cualquier combinación de los términos producto en cada término suma. Por tanto, es posible construir cualquier función que esté en forma de suma canónica de productos.

Para hacer que el plano OR sea programable, en cada cruce de líneas del plano OR hay un fusible metálico. Un fusible intacto se comporta como un circuito cerrado conectando la línea correspondiente a la puerta OR. Si se funde tal fusible, pasando una corriente alta a través de él, se impide tal contacto. De este modo es posible programar la memoria PROM. En la figura II.16 se muestra un ejemplo de PROM aún sin programar (obsérvese el uso abreviado en el diagrama de las líneas de entrada a las puertas de los planos AND y OR).

Durante el desarrollo de un circuito lógico, la información por almacenar en cada PROM experimenta cambios frecuentes hasta que el diseño ha sido depurado. Por desgracia, las PROM, al igual que las ROM, no se pueden modificar una vez programadas. En este caso, se utilizan las memorias programables exclusivas para lectura borrables (EPROM). El plano OR de una EPROM se programa con un voltaje de programación especial que hace que en determinadas celdas (cruce de líneas) se almacene carga. La presencia o ausencia de carga determina si la línea se conecta o no a la puerta OR correspondiente. Esta carga se puede eliminar al irradiar el circuito con luz ultravioleta a través de una ventana de cuarzo que tiene el chip, lo cual devuelve el plano OR a su condición inicial, es decir, sin programar. Este ciclo de borrado y programación puede repetirse hasta que el contenido sea correcto, lo que permite usar una única EPROM durante todo el desarrollo.

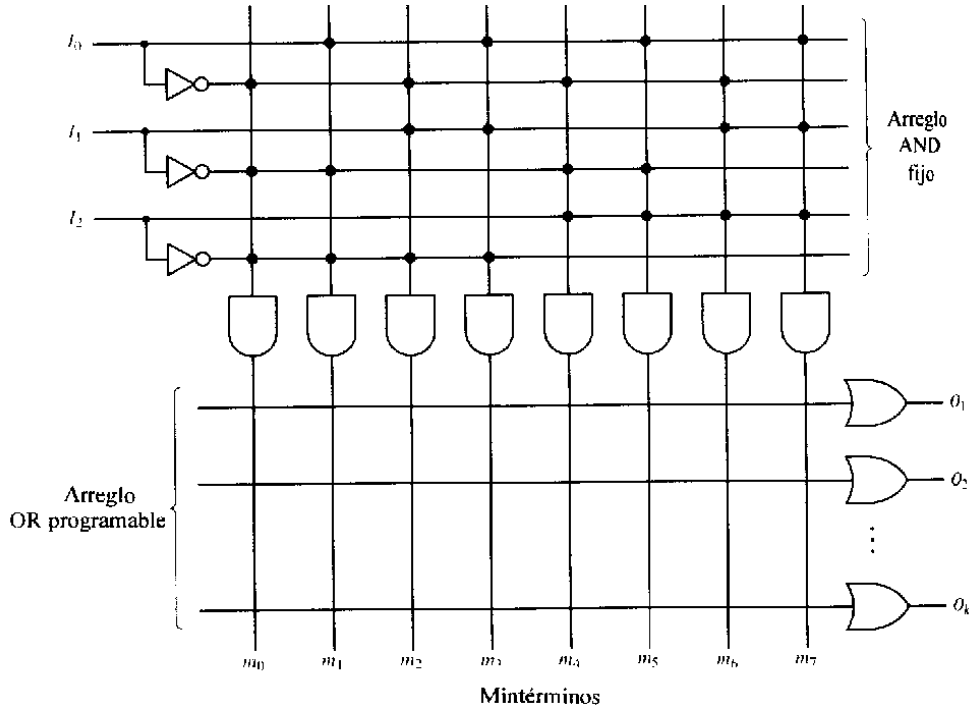


Figura II.16: Esquema de una memoria programable exclusiva para lectura (PROM).

Una memoria programable exclusiva para lectura borrable eléctricamente (EEPROM) es similar a un EPROM en el sentido de que el plano OR se puede programar repetidas veces mediante cargas eléctricas. Sin embargo, en una EEPROM el borrado se hace eléctricamente, aplicando un voltaje especial al circuito. Esto permite borrar y programar un circuito sin tener que aplicarle una luz especial. Por tanto, las EEPROM son atractivas en aplicaciones en las que es posible que haya que modificar la información almacenada a menudo. Muchos dispositivos EEPROM permiten un borrado selectivo del circuito. También existe un tipo de dispositivos EEPROM, llamados memorias flash, que permiten el borrado de todo el circuito, o por bloques, sin necesidad de ser extraídas de la placa en la que se encuentran. En resumen, las memorias EPROM y EEPROM son más flexibles que las PROM y, por tanto, más caras por bit.

II.7.3. PLA

Un array lógico programable (o PLA) es, de alguna manera, una generalización de una ROM. Al igual que aquella, tiene un conjunto de entradas con sus complementos correspondientes, a los que siguen dos etapas de lógica, una AND y una OR. En la primera etapa, cada puerta AND implementa un minitérmino, mientras que en la segunda cada puerta OR forma una suma lógica de cualquier número de los minitérminos realizados. Por tanto, un PLA también puede, como la ROM, implementar directamente la tabla de verdad de un conjunto de funciones lógicas con múltiples entradas y salidas.

La diferencia básica entre un PLA y una ROM radica en el plano AND. Mientras que, como vimos, en una ROM este plano es fijo e implementa todos los minitérminos posibles (2^m, siendo m el número de entradas), en una PLA este plano es programable e implementa un número limitado de minitérminos.

De esta forma, podemos ahorrar puertas AND si elegimos un PLA del tamaño adecuado, es decir, que tenga al menos tantas puertas AND como minitérminos tenga la función que deseamos implementar. El tamaño de los PLAs se indica mediante una expresión de la forma $I \times A \times O$, donde I es el número de entradas, A el número de puertas AND empleadas, y O el número de salidas (que coincide con el de puertas OR).

Por ejemplo, para la función que implementamos en la ROM de la figura II.15, se observaba que las filas correspondientes a los minitérminos 001 y 010 tienen todas sus salidas nulas (0000). Así, en la implementación podemos ver que las correspondientes puertas AND no se utilizan. En este caso, podríamos implementar la función con un PLA de tamaño $3 \times 6 \times 4$, como se muestra en la figura II.17.

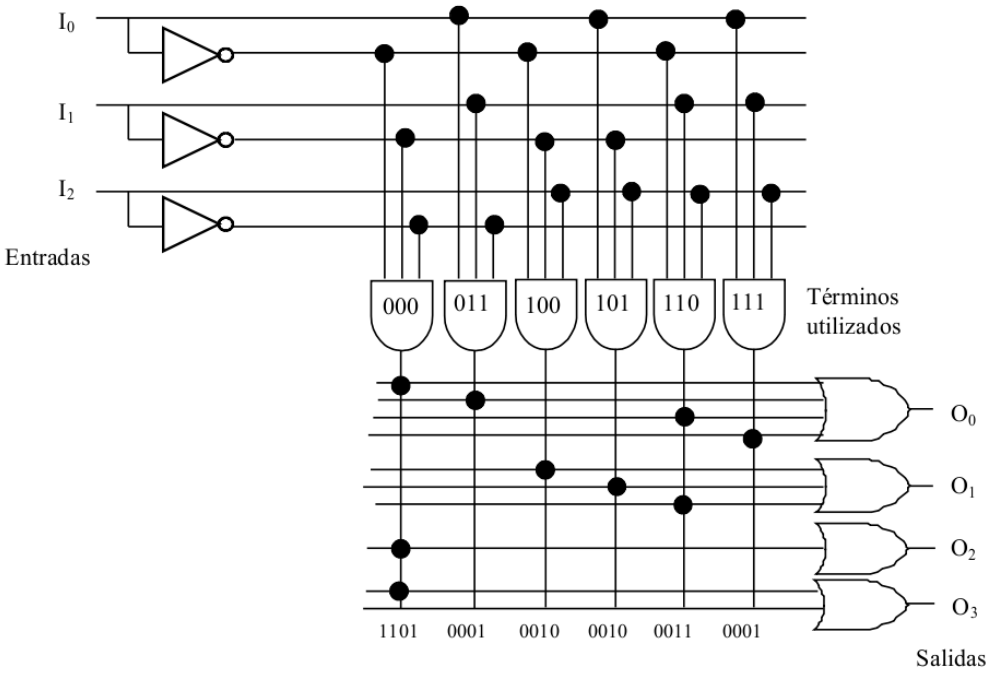


Figura II.17: PLA equivalente a la ROM de la figura II.15.

En general, para funciones con más entradas, donde muchas de las combinaciones de entrada no se utilizan, es muy probable que existan muchos minitérminos no utilizados.