FUNDAMENTOS DE LÓGICA DIGITAL CON DISEÑO VHDL

SEGUNDA EDICIÓN

Stephen Brown y Zvonko Vranesic
Departamento de Ingeniería Eléctrica y Computación
University of Toronto

Traducción
Lorena Peralta Rosales
Víctor Campos Olguín
Traductores profesionales

Revisión técnica
Jorge Valeriano Assem
Coordinador de la carrera de Ingeniería en Computación
Universidad Nacional Autónoma de México

Felipe Antonio Trujillo Fernández
Profesor del Departamento de Ingenierías
Universidad Iberoamericana, Campus Santa Fe

MÉXICO • BOGOTÁ • BUENOS AIRES • CARACAS • GUATEMALA • LISBOA
MADRID • NUEVA YORK • SAN JUAN • SANTIAGO
AUCKLAND • LONDRES • MILÁN • MONTREAL • NUEVA DELHI
SAN FRANCISCO • SINGAPUR • ST. LOUIS • SIDNEY • TORONTO

capítulo

4

Implementación optimizada de funciones lógicas

Objetivos del capítulo

En este capítulo se estudian los temas siguientes:

- Síntesis de las funciones lógicas
- Análisis de los circuitos lógicos
- Técnicas para derivar implementaciones de costo mínimo de las funciones lógicas
- Representación gráfica de funciones lógicas mediante mapas de Karnaugh
- Representación cúbica de funciones lógicas
- El uso de las herramientas CAD y de VHDL para implementar funciones lógicas

En el capítulo 2 mostramos que la manipulación algebraica permite hallar la implementación de menor costo de las funciones lógicas. El propósito de ese capítulo era introducir los conceptos básicos en el proceso de síntesis. Probablemente el lector esté convencido de que es fácil derivar una realización directa de una función lógica en una forma canónica, pero no es del todo obvio cómo elegir y aplicar los teoremas y propiedades de la sección 2.5 a fin de hallar un circuito de costo mínimo. De hecho, la manipulación algebraica es más bien tediosa y muy impráctica para funciones de muchas variables.

Si se usan herramientas CAD para diseñar circuitos lógicos, la tarea de minimizar el costo de implementación no recae en el diseñador; las herramientas realizan automáticamente la optimación necesaria. Aun así, es esencial saber algo acerca de este proceso. La mayor parte de las herramientas CAD tiene muchas características y opciones que quedan bajo el control del usuario. Para saber cuándo y cómo aplicarlas, éste debe conocer qué hacen las herramientas.

En este capítulo expondremos algunas de las técnicas de optimación incluidas en las herramientas CAD y mostraremos cómo automatizarlas. En primer lugar explicaremos un enfoque gráfico, conocido como *mapa de Karnaugh*, que ofrece una forma elegante de derivar manualmente implementaciones de costo mínimo de funciones lógicas simples. Aunque no está disponible para implementarlo en las herramientas CAD, sí ilustra varios conceptos clave. Mostraremos cómo diseñar circuitos de dos y varios niveles. Luego describiremos una representación cúbica de las funciones lógicas, que está disponible para emplearla en las herramientas CAD. También proseguiremos con nuestra explicación del lenguaje VHDL.

4.1 MAPA DE KARNAUGH

En la sección 2.6 vimos que la clave para hallar una expresión de costo mínimo para una función lógica consiste en reducir el número de términos producto (o suma) necesarios en la expresión. Ello se logra mediante la aplicación de la propiedad combinatoria 14*a* (o 14*b*) cuan juiciosamente sea posible. El enfoque de mapa de Karnaugh brinda una forma sistemática de realizar esta optimación. Para entender cómo funciona será útil revisar el enfoque algebraico del capítulo 2. Considérese la función f de la figura 4.1. La expresión canónica en suma de productos para f consta de los mintérminos m_0, m_2, m_4, m_5 y m_6, de modo que

$$f = \overline{x}_1\overline{x}_2\overline{x}_3 + \overline{x}_1x_2\overline{x}_3 + x_1\overline{x}_2\overline{x}_3 + x_1\overline{x}_2x_3 + x_1x_2\overline{x}_3$$

La propiedad combinatoria 14*a* permite sustituir dos mintérminos que difieren sólo en el valor de una variable con un solo término producto que no incluye esa variable. Por ejemplo, tanto m_0 como m_2 incluyen $\overline{x}_1$ y $\overline{x}_3$, pero difieren en el valor de x_2 porque m_0 incluye $\overline{x}_2$ mientras que m_2 incluye x_2. Por ende

$$\begin{aligned}\overline{x}_1\overline{x}_2\overline{x}_3 + \overline{x}_1x_2\overline{x}_3 &= \overline{x}_1(\overline{x}_2 + x_2)\overline{x}_3\\ &= \overline{x}_1 \cdot 1 \cdot \overline{x}_3\\ &= \overline{x}_1\overline{x}_3\end{aligned}$$

Número de fila	x_1	x_2	x_3	f
0	0	0	0	1
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

Figura 4.1 La función $f(x_1, x_2, x_3) = \sum m(0, 2, 4, 5, 6)$.

Por tanto, m_0 y m_2 pueden sustituirse por el término producto $\overline{x}_1\overline{x}_3$. De manera similar, m_4 y m_6 difieren sólo en el valor de x_2 y pueden combinarse usando

$$\begin{aligned} x_1\overline{x}_2\overline{x}_3 + x_1x_2\overline{x}_3 &= x_1(\overline{x}_2 + x_2)\overline{x}_3 \\ &= x_1 \cdot 1 \cdot \overline{x}_3 \\ &= x_1\overline{x}_3 \end{aligned}$$

Ahora los dos términos recién generados, $\overline{x}_1\overline{x}_3$ y $x_1\overline{x}_3$ pueden combinarse todavía más como

$$\begin{aligned} \overline{x}_1\overline{x}_3 + x_1\overline{x}_3 &= (\overline{x}_1 + x_1)\overline{x}_3 \\ &= 1 \cdot \overline{x}_3 \\ &= \overline{x}_3 \end{aligned}$$

Estos pasos de optimación indican que podemos sustituir los cuatro mintérminos m_0, m_2, m_4 y m_6 con el término producto $\overline{x}_3$. En otras palabras, los mintérminos m_0, m_2, m_4 y m_6 están todos *incluidos* en el término $\overline{x}_3$. El mintérmino restante en f es m_5. Puede combinarse con m_4, lo que produce

$$x_1\overline{x}_2\overline{x}_3 + x_1\overline{x}_2x_3 = x_1\overline{x}_2$$

Recuérdese que el teorema 7*b* de la sección 2.5 postula que

$$m_4 = m_4 + m_4$$

lo que significa que es posible usar el mintérmino m_4 dos veces: para combinarse con los mintérminos m_0, m_2 y m_6 para producir el término $\overline{x}_3$ como se explicó anteriormente, y también para combinarse con m_5 para producir el término $x_1\overline{x}_2$.

Ahora hemos dado cuenta de todos los mintérminos en f; por tanto, las cinco combinaciones de entrada para las que $f = 1$ están cubiertas por la expresión de costo mínimo

$$f = \overline{x}_3 + x_1\overline{x}_2$$

La expresión tiene el término producto $\overline{x}_3$ porque $f = 1$ cuando $x_3 = 0$ independientemente de los valores de x_1 y x_2. Los cuatro mintérminos m_0, m_2, m_4 y m_6 representan todos los mintérminos posibles para los que $x_3 = 0$; incluyen las cuatro combinaciones, 00, 01, 10 y 11, de las variables x_1 y x_2. Por ende, si $x_3 = 0$, entonces está garantizado que $f = 1$. Esto puede no ser fácil de ver directamente a partir de la tabla de verdad de la figura 4.1, pero es obvio si escribimos agrupadas las combinaciones correspondientes:

	x_1	x_2	x_3
m_0	0	0	0
m_2	0	1	0
m_4	1	0	0
m_6	1	1	0

De forma similar, si se observan m_4 y m_5 como un grupo de dos

	x_1	x_2	x_3
m_4	1	0	0
m_5	1	1	1

es claro que cuando $x_1 = 1$ y $x_2 = 0$, entonces $f = 1$ independientemente de los valores de x_3.

La explicación anterior sugiere que sería ventajoso elaborar un método que permita el descubrimiento rápido de grupos de mintérminos para los que $f = 1$ y que puedan combinarse en términos individuales. El mapa de Karnaugh es un vehículo útil para tal propósito.

El *mapa de Karnaugh* [1] es una alternativa a la forma de tabla de verdad para representar una función. Consta de *celdas* que corresponden a las filas de la tabla de verdad. Considérese el ejemplo de dos variables de la figura 4.2. En el inciso *a*) se muestra la forma en tabla de verdad, donde cada una de las cuatro filas se identifica mediante un mintérmino. En el inciso *b*) se observa el mapa de Karnaugh, que tiene cuatro celdas. Las columnas están etiquetadas con el valor de x_1, y las filas con el de x_2. Esta forma de etiquetar conduce a la ubicación de los mintérminos, como se advierte en la figura. Si se compara con la tabla de verdad, la ventaja del mapa de Karnaugh es que permite reconocer con facilidad los mintérminos que pueden combinarse aplicando la propiedad 14*a* de la sección 2.5. Los mintérminos en cualesquiera dos celdas adyacentes, ya sea en la misma fila o en la misma columna, pueden combinarse. Por ejemplo, los mintérminos m_2 y m_3 pueden combinarse como

$$\begin{aligned} m_2 + m_3 &= x_1\overline{x}_2 + x_1x_2 \\ &= x_1(\overline{x}_2 + x_2) \\ &= x_1 \cdot 1 \\ &= x_1 \end{aligned}$$

x_1	x_2	
0	0	m_0
0	1	m_1
1	0	m_2
1	1	m_3

a) Tabla de verdad

x_2 \ x_1	0	1
0	m_0	m_2
1	m_1	m_3

b) Mapa de Karnaugh

Figura 4.2 Ubicación de los mintérminos de dos variables.

El mapa de Karnaugh no sólo es útil para combinar pares de mintérminos. Como veremos en varios ejemplos más grandes, puede usarse directamente para derivar un circuito de costo mínimo de una función lógica.

Mapa de dos variables

En la figura 4.3 se ilustra un mapa de Karnaugh para una función de dos variables. Corresponde a la función f de la figura 2.15. El valor de f para cada combinación de las variables x_1 y x_2 se indica en la celda correspondiente del mapa. Como aparece 1 en ambas celdas de la fila inferior, las cuales son adyacentes, hay un solo término producto que puede hacer que f sea igual a 1 cuando las variables de entrada tienen los valores correspondientes a cualquiera de esas celdas. Para indicar este hecho, hemos encerrado en un círculo las entradas de las celdas en el mapa. En vez de aplicar la propiedad combinatoria de manera formal, es posible derivar el término producto intuitivamente. Ambas celdas se identifican mediante $x_2 = 1$, pero $x_1 = 0$ para la celda izquierda y $x_1 = 1$ para la celda derecha. Por tanto, si $x_2 = 1$, entonces $f = 1$ independientemente de que x_1 sea igual a 0 o a 1. El término producto que representa las dos celdas es simplemente x_2.

De manera similar, $f = 1$ para ambas celdas en la primera columna. Dichas celdas se identifican mediante $x_1 = 0$. En consecuencia, conducen al término producto $\bar{x}_1$. En virtud de que esto considera todos los casos en los que $f = 1$, se deduce que la realización de costo mínimo de la función es

$$f = x_2 + \bar{x}_1$$

Evidentemente, para encontrar una implementación de costo mínimo de una función es preciso hallar el número más pequeño de términos producto que producen un valor de 1 para todos

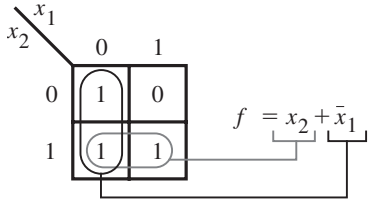


Figura 4.3 La función de la figura 2.15.

los casos donde $f = 1$. Más aún, el costo de estos términos producto debe ser lo más bajo posible. Nótese que es más barato implementar un término producto que abarca dos celdas adyacentes que un término que cubre una sola celda. Para nuestro ejemplo, una vez que el término producto x_2 cubre las dos celdas en la fila inferior, sólo queda una celda (la superior izquierda). Aunque podría abarcarse mediante el término $\overline{x}_1\overline{x}_2$, es mejor combinar las dos celdas de la columna izquierda para producir el término producto $\overline{x}_1$ porque es más barato implementarlo.

Mapa de tres variables

Un mapa de Karnaugh de tres variables se construye colocando dos mapas de dos variables lado a lado. En la figura 4.4 se muestra el mapa y se indican las ubicaciones de los mintérminos. En este caso cada combinación de x_1 y x_2 identifica una columna del mapa, mientras que el valor de x_3 distingue las dos filas. Para asegurar que los mintérminos de las celdas adyacentes del mapa siempre puedan combinarse en un solo término producto, tales celdas deben diferir en el valor de sólo una variable. Por tanto, las columnas se identifican mediante la sucesión de valores (x_1, x_2) de 00, 01, 11 y 10, en vez de los más obvios 00, 01, 10 y 11. Esto hace que la segunda y la tercera columnas sean diferentes sólo en la variable x_1. Además, la primera y la cuarta columnas difieren sólo en la variable x_1, lo que significa que esas columnas pueden considerarse adyacentes. El lector puede encontrar útil imaginar el mapa como un rectángulo plegado en un cilindro donde se tocan los bordes izquierdo y derecho de la figura 4.4*b*. (Una secuencia de códigos, o combinaciones, donde códigos consecutivos difieren sólo en una variable se conoce como *código Gray*, que tiene varios propósitos, algunos de los cuales se abordan más adelante.)

En la figura 4.5*a* se representa la función de la figura 2.18 en forma de mapa de Karnaugh. Para sintetizar esta función es necesario cubrir los cuatro 1 del mapa tan eficientemente como sea posible. No es difícil ver que para ello bastan dos términos producto. El primero cubre los 1 de la fila superior, los cuales están representados por el término $x_1\overline{x}_3$. El segundo término es $\overline{x}_2x_3$, que cubre los 1 de la fila inferior. Por ende, la función se implementa como

$$f = x_1\overline{x}_3 + \overline{x}_2x_3$$

que describe el circuito obtenido en la figura 2.19*a*.

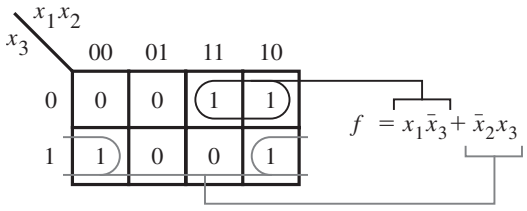
x_1	x_2	x_3	
0	0	0	m_0
0	0	1	m_1
0	1	0	m_2
0	1	1	m_3
1	0	0	m_4
1	0	1	m_5
1	1	0	m_6
1	1	1	m_7

a) Tabla de verdad

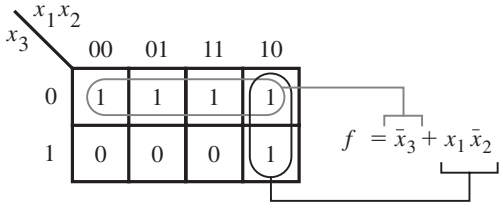
x_3 \ x_1x_2	00	01	11	10
0	m_0	m_2	m_6	m_4
1	m_1	m_3	m_7	m_5

b) Mapa de Karnaugh

Figura 4.4 Ubicación de mintérminos de tres variables.



a) La función de la figura 2.18



b) La función de la figura 4.1

Figura 4.5 Ejemplos de mapas de Karnaugh de tres variables.

En un mapa de tres variables es posible combinar celdas para producir términos producto que correspondan a una sola celda, a dos celdas adyacentes o a un grupo de cuatro celdas adyacentes. La realización de un grupo de cuatro celdas adyacentes que usan un solo término producto se ilustra en la figura 4.5*b,* usando la función de la figura 4.1. Las cuatro celdas de la fila superior corresponden a las combinaciones 000, 010, 110 y 100 de (x_1, x_2, x_3). Como ya explicamos, esto indica que si $x_3 = 0$, entonces $f = 1$ para las cuatro posibles combinaciones de x_1 y x_2, lo que significa que el único requisito es que $x_3 = 0$. Por ende, el término producto $\bar{x}_3$ representa esas cuatro celdas. El 1 restante, que corresponde al mintérmino m_5, se cubre mejor con el término $x_1\bar{x}_2$, que se obtiene al combinar las dos celdas de la columna de la derecha. La realización completa de f es

$$f = \bar{x}_3 + x_1\bar{x}_2$$

También es posible tener un grupo de ocho 1 en un mapa de tres variables. Éste es el caso trivial donde $f = 1$ para todas las combinaciones de variables de entrada; en otras palabras, f es igual a la constante 1.

El mapa de Karnaugh ofrece un mecanismo simple para generar los términos producto que han de usarse a fin de implementar una función. Un término producto debe incluir sólo las variables que tengan el mismo valor para todas las celdas en el grupo representado por ese término. Si la variable es igual a 1 en el grupo, aparece sin complementar en el término producto; si es igual a 0, aparece complementada. Cada variable que a veces es 1 y a veces 0 en el grupo no aparece en el término producto.

Mapa de cuatro variables

Un mapa de cuatro variables se construye colocando juntos dos mapas de tres variables para crear cuatro filas de la misma forma que empleamos dos mapas de dos variables para formar las cuatro columnas de un mapa de tres variables. En la figura 4.6 se muestra la estructura del mapa

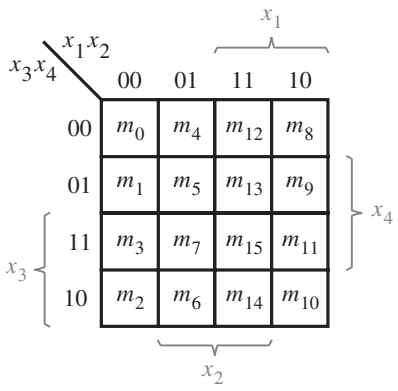


Figura 4.6 Mapa de Karnaugh de cuatro variables.

de cuatro variables y la ubicación de los mintérminos. En esta figura hemos incluido otra forma muy usada de diseñar las filas y las columnas. Como se muestra en gris, basta indicar las filas y las columnas para las que una variable es igual a 1. Por tanto, $x_1 = 1$ para las dos columnas de la extrema derecha, $x_2 = 1$ para las dos columnas de en medio, $x_3 = 1$ para las dos filas de la parte inferior y $x_4 = 1$ para las dos filas de en medio.

En la figura 4.7 se presentan cuatro ejemplos de funciones de cuatro variables. La función f_1 tiene un grupo de cuatro 1 en celdas adyacentes en las dos filas inferiores, para las cuales $x_2 = 1$ y $x_3 = 1$; se representan con el término producto $\overline{x}_2x_3$. Esto deja por cubrir los dos 1 de la segunda fila, lo que se logra con el término $x_1\overline{x}_3x_4$. En consecuencia, la implementación de costo mínimo de la función es

$$f_1 = \overline{x}_2x_3 + x_1\overline{x}_3x_4$$

La función f_2 incluye un grupo de ocho 1 que pueden implementarse mediante un solo término, x_3. De nuevo el lector debe notar que si los dos 1 restantes se implementasen por separado, el resultado sería el término producto $x_1\overline{x}_3x_4$. La implementación de estos 1 como parte de un grupo de cuatro 1, como se muestra en la figura, produce el término producto menos costoso x_1x_4.

Igual que los bordes izquierdo y derecho del mapa son adyacentes en términos de la asignación de las variables, los bordes superior e inferior también lo son. De hecho, las cuatro esquinas del mapa son adyacentes una a otra y, por tanto, pueden formar un grupo de cuatro 1, que puede implementarse con el término producto $\overline{x}_2\overline{x}_4$. Este caso queda descrito con la función f_3. Aparte de este grupo de 1, hay otros cuatro 1 que deben cubrirse para implementar f_3. Ello se logra como se muestra en la figura.

En todos los ejemplos considerados hasta el momento existe una solución única que conduce al circuito de costo mínimo. La función f_4 brinda un ejemplo en el que se tienen algunas opciones. Los grupos de cuatro 1 de las esquinas superior izquierda e inferior derecha del mapa se realizan mediante los términos $\overline{x}_1\overline{x}_3$ y x_1x_3, respectivamente. Esto deja los dos 1 correspondientes al término $x_1x_2\overline{x}_3$, pero éstos pueden realizarse de manera más económica si se tratan como parte de un grupo de cuatro 1. Pueden ser incluidos en dos diferentes grupos de cuatro,

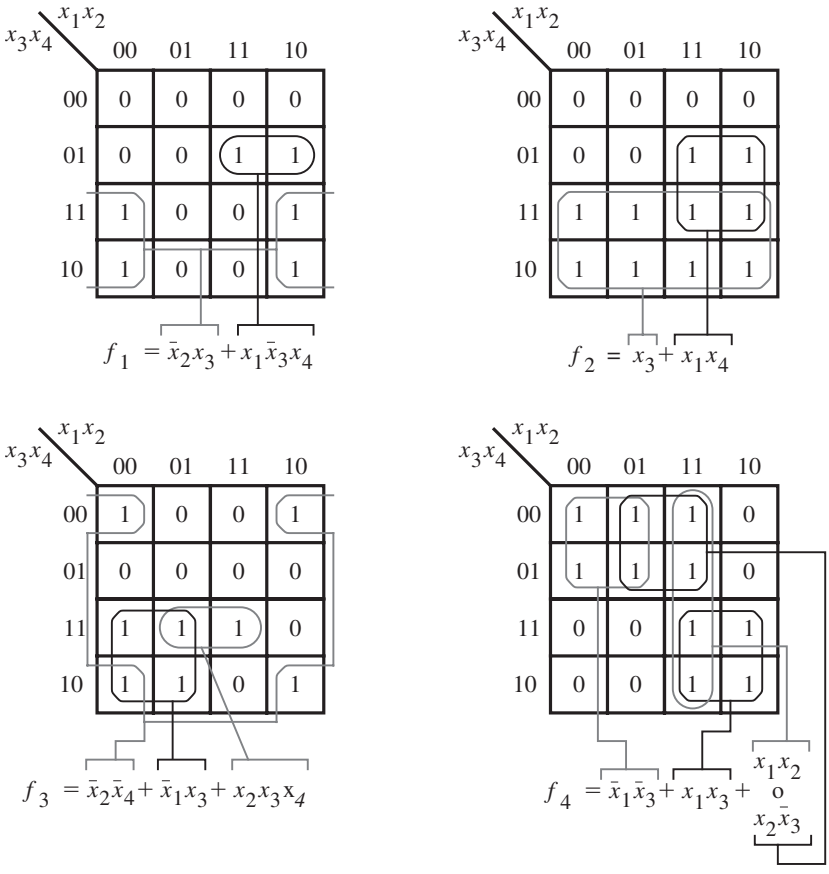


Figura 4.7 Ejemplos de mapas de Karnaugh de cuatro variables.

como se indica en la figura. Una posibilidad conduce al término producto x_1x_2 y la otra a $x_2\overline{x}_3$. Ambos términos tienen el mismo costo; por tanto, no importa cuál se elija en el circuito final. Nótese que el complemento de x_3 en el término $x_2\overline{x}_3$ no implica un aumento en el costo en comparación con x_1x_2, ya que este complemento debe generarse de cualquier forma para producir el término $\overline{x}_1\overline{x}_3$, el cual se incluye en la implementación.

Mapa de cinco variables

Pueden usarse dos mapas de cuatro variables para construir un mapa de cinco variables. Es fácil imaginar una estructura donde un mapa se halle directamente detrás de otro y ambos se distingan por $x_5 = 0$ para un mapa y $x_5 = 1$ para el otro. Como resulta complicado dibujar tal estructura, los dos mapas simplemente se colocan lado a lado, como se muestra en la figura 4.8. Para la función lógica dada en este ejemplo, dos grupos de cuatro 1 aparecen en el mismo lugar en los dos mapas de cuatro variables, en consecuencia su realización no depende del valor de x_5. Lo mismo es cierto para los dos grupos de dos 1 de la segunda fila. El 1 de la esquina superior

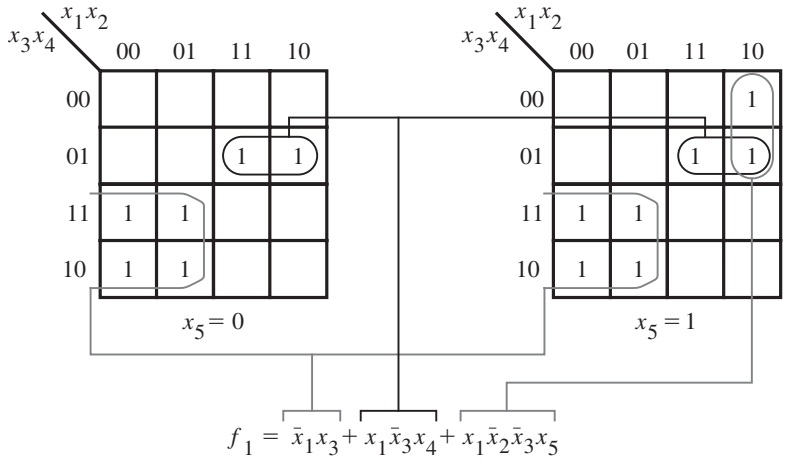


Figura 4.8 Mapa de Karnaugh de cinco variables.

derecha sólo aparece en el mapa de la derecha, donde $x_5 = 1$; forma parte del grupo de dos 1 realizado por el término $x_1\bar{x}_2\bar{x}_3x_5$. Nótese que en este mapa se dejan en blanco las celdas para las que $f = 0$ a fin de hacer que la figura sea más legible. Del mismo modo se hará en varios de los mapas que siguen.

El uso de un mapa de cinco variables es naturalmente más complicado que el de mapas con menos variables. La extensión del concepto de mapa de Karnaugh a más variables no resulta útil desde el punto de vista práctico. Esto no es problemático, puesto que la síntesis práctica de las funciones lógicas se realiza con herramientas CAD que llevan a cabo la minimización necesaria de forma automática. Aunque los mapas de Karnaugh ocasionalmente son útiles para diseñar circuitos lógicos pequeños, la razón principal de exponerlos es ofrecer un vehículo simple para ilustrar las ideas que el proceso de minimización supone.

4.2 ESTRATEGIA DE MINIMIZACIÓN

Para los ejemplos de la sección anterior recurrimos a un enfoque intuitivo a fin de decidir cómo han de agruparse los 1 en un mapa de Karnaugh para obtener la implementación de costo mínimo de una función. Nuestra estrategia intuitiva fue encontrar tan pocos y tan grandes grupos de 1 como fuera posible para cubrir todos los casos en que la función tuviese un valor de 1. Cada grupo de 1 debe abarcar celdas que puedan representarse mediante un solo término producto. Cuanto más grande sea el grupo de 1, menor será el número de variables en el término producto correspondiente. Este enfoque sirvió bien porque los mapas de Karnaugh de los ejemplos eran pequeños. Es inadecuado para funciones lógicas más grandes, con muchas variables. En vez de ello hay que contar con un método organizado para derivar una implementación de costo mínimo. En esta sección expondremos un posible método, similar a las técnicas automatizadas en

las herramientas CAD. Para ilustrar las ideas principales emplearemos mapas de Karnaugh. Más adelante, en la sección 4.8, describiremos otra forma de representar funciones lógicas, la cual se utiliza en las herramientas CAD.

4.2.1 TERMINOLOGÍA

En el desarrollo de técnicas para la síntesis de funciones lógicas se ha realizado una gran cantidad de trabajo de investigación. Sus resultados están publicados en varios documentos. En aras de facilitar la presentación de los resultados, ha evolucionado cierta terminología que evita usar frases muy descriptivas. En los párrafos siguientes definimos parte de esa terminología porque es útil para describir el proceso de minimización.

Literal

Un término producto consta de cierto número de variables, cada una de las cuales puede aparecer en forma complementada o sin complementar. Cada aparición de una variable, ya sea sin complementar o complementada, se llama *literal*. Por ejemplo, el término producto $x_1\overline{x}_2x_3$ tiene tres literales, y $\overline{x}_1x_3\overline{x}_4x_6$ cuatro.

Implicante

Un término producto que indica la combinación de entrada para la que una función es igual a 1 se llama *implicante* de la función. Los implicantes más básicos son los mintérminos, que explicamos en la sección 2.6.1. Para una función de n variables, un mintérmino es un implicante que consta de n literales.

Considérese la función de tres variables de la figura 4.9. Tiene 11 posibles implicantes, los cuales incluyen los cinco mintérminos: $\overline{x}_1\overline{x}_2\overline{x}_3$, $\overline{x}_1\overline{x}_2x_3$, $\overline{x}_1x_2\overline{x}_3$, $\overline{x}_1x_2x_3$ y $x_1x_2x_3$. Luego están los implicantes correspondientes a todos los posibles pares de mintérminos que pueden combinarse: $\overline{x}_1\overline{x}_2$($m_0$ y m_1), $\overline{x}_1\overline{x}_3$ (m_0 y m_2), $\overline{x}_1x_3$ (m_1 y m_3), $\overline{x}_1x_2$ (m_2 y m_3) y x_2x_3 (m_3 y m_7). Finalmente, existe un implicante que abarca un grupo de cuatro mintérminos, el cual consta de una sola literal $\overline{x}_1$.

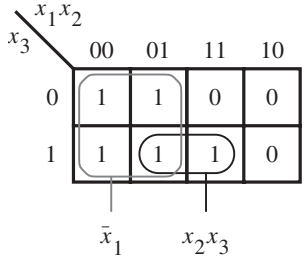


Figura 4.9 Función de tres variables $f(x_1, x_2, x_3) = \sum m(0, 1, 2, 3, 7)$.

Implicante primo

Un implicante se llama *implicante primo* si no puede combinarse en otro implicante que tenga menos literales. Dicho de otra forma, es imposible borrar alguna literal en un implicante primo y aún así tener un implicante válido.

En la figura 4.9 hay dos implicantes primos: $\overline{x}_1$ y x_2x_3. No es posible borrar una literal en cualquiera de ellos. Hacerlo para $\overline{x}_1$ lo haría desaparecer. Para x_2x_3, borrar una literal dejaría x_2 o x_3. Pero x_2 no es un implicante porque incluye la combinación $(x_1, x_2, x_3) = 110$ para la que $f = 0$, y x_3 tampoco lo es porque incluye $(x_1, x_2, x_3) = 101$ para la que $f = 0$.

Cobertura

Un conjunto de implicantes que abarca todas las combinaciones para las que una función es igual a 1 se denomina *cobertura* de dicha función. Para la mayor parte de las funciones existen varias coberturas. Obviamente, un conjunto de todos los mintérminos para los que $f = 1$ es una cobertura. También es evidente que un conjunto de todos los implicantes primos es una cobertura.

Una cobertura define una implementación en particular de la función. En la figura 4.9 una cobertura que consta de mintérminos lleva a la expresión

$$f = \overline{x}_1\overline{x}_2\overline{x}_3 + \overline{x}_1\overline{x}_2x_3 + \overline{x}_1x_2\overline{x}_3 + \overline{x}_1x_2x_3 + x_1x_2x_3$$

Otra cobertura válida está dada por la expresión

$$f = \overline{x}_1\overline{x}_2 + \overline{x}_1x_2 + x_2x_3$$

La cobertura que comprende los implicantes primos es

$$f = \overline{x}_1 + x_2x_3$$

Si bien todas estas expresiones representan la función f correctamente, la cobertura que consta de implicantes primos conduce a la implementación de menor costo.

Costo

En el capítulo 2 sugerimos que una buena indicación del costo de un circuito lógico es el número de compuertas en el circuito más el número total de entradas a todas ellas. A lo largo del libro usaremos esta definición de costo, pero supondremos que las entradas primarias, es decir, las variables de entrada, están disponibles en formas tanto verdadera como complementada sin ningún costo. Por tanto, la expresión

$$f = x_1\overline{x}_2 + x_3\overline{x}_4$$

tiene un costo de nueve porque puede implementarse con dos compuertas AND y una OR, con seis entradas a ellas.

Si dentro de un circuito se necesita una inversión, entonces la correspondiente compuerta NOT y su entrada se incluyen en el costo. Por ejemplo, la expresión

$$g = \overline{x_1\overline{x}_2 + x_3}(\overline{x}_4 + x_5)$$

se implementa con dos compuertas AND, dos OR y una NOT para complementar $(x_1\overline{x}_2 + x_3)$, con nueve entradas. Por ende, el costo total es 14.

4.2.2 PROCEDIMIENTO DE MINIMIZACIÓN

Hemos visto que es posible implementar una función lógica con varios circuitos, los cuales pueden tener diferentes estructuras y costos. Cuando se diseña un circuito lógico suele haber ciertos criterios que han de satisfacerse. Uno de ellos es el costo del circuito, tema que consideramos en la explicación previa. En general, cuanto más grande sea el circuito, mayor importancia tendrá el tema del costo. En esta sección supondremos que el objetivo central es obtener un circuito de costo mínimo.

Luego de decir que el costo es la preocupación principal, cabe señalar que otros criterios de optimación pueden ser más apropiados en ciertos casos. Por ejemplo, en el capítulo anterior describimos varios tipos de dispositivos lógicos programables (PLD) que tienen una estructura básica predefinida y pueden programarse para realizar varios circuitos. Para ellos el objeto principal radica en diseñar un circuito específico de modo que encaje en el dispositivo que se busca. Si ese circuito tiene o no el costo mínimo es algo irrelevante siempre que pueda realizarse bien en el dispositivo. Una herramienta CAD que tiene el propósito de diseñar con un dispositivo específico en mente desarrollará de manera automática optimaciones adecuadas para él. En la sección 4.6 mostraremos que la forma en la que debe optimarse un circuito puede ser diferente de acuerdo con el tipo de dispositivo.

En la subsección anterior llegamos a la conclusión de que la implementación de costo más bajo se logra cuando la cobertura de una función consta de implicantes primos. Cabe entonces preguntar cómo se determina el subconjunto de costo mínimo de implicantes primos que cubrirán la función. Es posible que algunos implicantes primos deban ser incluidos en la cobertura, mientras que para otros puede haber opciones. Si un implicante primo incluye un mintérmino para el que $f = 1$ que no está incluido en algún otro implicante primo, entonces se le debe incluir en la cobertura y se le llama *implicante primo esencial*. En el ejemplo de la figura 4.9, ambos implicantes primos son esenciales. El término x_2x_3 es el único implicante que cubre el mintérmino m_7, y $\overline{x}_1$ es el único que cubre los mintérminos m_0, m_1 y m_2. Nótese que el mintérmino m_3 se cubre con estos dos implicantes primos. La realización de costo mínimo de la función es

$$f = \overline{x}_1 + x_2x_3$$

Ahora presentaremos varios ejemplos en los que hay una opción respecto de qué implicantes primos incluir en la cobertura final. Considérese la función de cuatro variables de la figura 4.10. Hay cinco implicantes primos: $\overline{x}_1x_3$, $\overline{x}_2x_3$, $\underline{x}_3\overline{x}_4$, $\overline{x}_1x_2x_4$ y $x_2\overline{x}_3x_4$. Los esenciales (resaltados en gris) son $\overline{x}_2x_3$ (debido a m_{11}), $x_3\overline{x}_4$ (debido a m_{14}) y $x_2\overline{x}_3x_4$ (debido a m_{13}). Estos mintérminos deben incluirse en la cobertura. Estos tres implicantes primos cubren todos los mintérminos para los que $f = 1$, excepto m_7. Es claro que m_7 puede cubrirse mediante $\overline{x}_1x_3$ o bien por $\overline{x}_1x_2x_4$. Puesto que $\overline{x}_1x_3$ tiene el costo más bajo, se elige para la cobertura. En consecuencia, la realización de costo mínimo es

$$f = \overline{x}_2x_3 + x_3\overline{x}_4 + x_2\overline{x}_3x_4 + \overline{x}_1x_3$$

A partir de lo anterior el proceso de encontrar un circuito de costo mínimo comprende los pasos siguientes:

1. Generar todos los implicantes primos para la función f.
2. Encontrar el conjunto de implicantes primos esenciales.

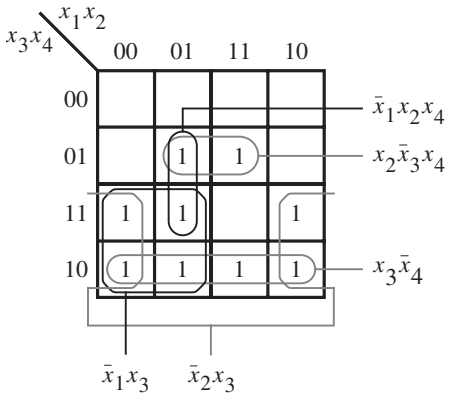


Figura 4.10 Función de cuatro variables $f(x_1, \ldots, x_4) = \sum m(2, 3, 5, 6, 7, 10, 11, 13, 14)$.

3. Si el conjunto de implicantes primos esenciales cubre todas las combinaciones para las que $f = 1$, entonces es la cobertura deseada de f. De otro modo hay que determinar los implicantes primos no esenciales que deben agregarse para formar una cobertura completa de costo mínimo.

La elección de los implicantes primos no esenciales que han de incluirse en la cobertura está regida por consideraciones de costo. Esta elección no siempre es obvia. De hecho, para funciones grandes podría haber muchas posibilidades y entonces hay que emplear un enfoque *heurístico* (es decir, que si bien considera sólo un subconjunto de posibilidades, brinda buenos resultados la mayor parte de las veces). Uno de tales enfoques consiste en elegir arbitrariamente un implicante primo no esencial, incluirlo en la cobertura y luego determinar el resto de ella. Después se determina otra cobertura con la suposición de que ese implicante no está en ella. Se comparan los costos de las coberturas resultantes y se elige implementar la menos costosa.

Podemos ilustrar el proceso con la función de la figura 4.11. De los seis implicantes primos, sólo $\bar{x}_3\bar{x}_4$ es esencial. Considérese a continuación $x_1x_2\bar{x}_3$ y supóngase primero que se incluirá en

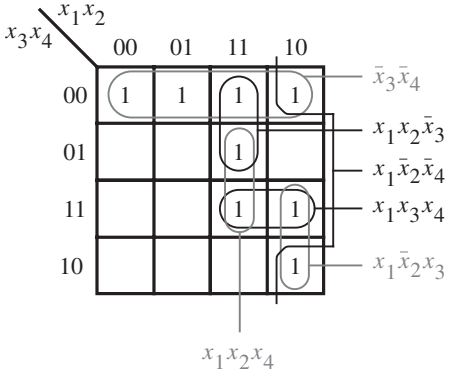


Figura 4.11 La función $f(x_1, \ldots, x_4) = \sum m(0, 4, 8, 10, 11, 12, 13, 15)$.

la cobertura. Luego los restantes tres mintérminos, m_{10}, m_{11} y m_{15}, requerirán dos implicantes primos más para incluirse en la cobertura. Una posible implementación es

$$f = \bar{x}_3\bar{x}_4 + x_1x_2\bar{x}_3 + x_1x_3x_4 + x_1\bar{x}_2x_3$$

La segunda posibilidad es que $x_1x_2\bar{x}_3$ no se incluya en la cobertura. Entonces $x_1x_2x_4$ se vuelve esencial porque no hay otra forma de cubrir m_{13}. Puesto que $x_1x_2x_4$ también cubre m_{15}, sólo m_{10} y m_{11} permanecen sin cubrir, lo que se logra con $x_1\bar{x}_2x_3$. Por tanto, la implementación alternativa es

$$f = \bar{x}_3\bar{x}_4 + x_1x_2x_4 + x_1\bar{x}_2x_3$$

Es evidente que esta implementación es la mejor opción.

A veces puede no haber implicantes primos esenciales en absoluto. En la figura 4.12 se da un ejemplo. Elegir cualquiera de los implicantes primos, incluirlo y luego excluirlo de la cobertura lleva a dos posibilidades de igual costo. Una comprende los implicantes primos indicados en negro, lo que produce

$$f = \bar{x}_1\bar{x}_3\bar{x}_4 + x_2\bar{x}_3x_4 + x_1x_3x_4 + \bar{x}_2x_3\bar{x}_4$$

La otra incluye los implicantes primos indicados en gris, lo que produce

$$f = \bar{x}_1\bar{x}_2\bar{x}_4 + \bar{x}_1x_2\bar{x}_3 + x_1x_2x_4 + x_1\bar{x}_2x_3$$

Este procedimiento puede aplicarse para encontrar las implementaciones de costo mínimo de funciones lógicas pequeñas o grandes. Para nuestros ejemplos pequeños fue conveniente usar mapas de Karnaugh a fin de determinar los implicantes primos de una función y luego elegir la cobertura final. En las herramientas CAD hay otras técnicas basadas en los mismos principios cuyo uso es más adecuado; las expondremos en las secciones 4.9 y 4.10.

Los ejemplos anteriores se basaron en la forma de suma de productos. A continuación ilustraremos los mismos conceptos para la forma de producto de sumas.

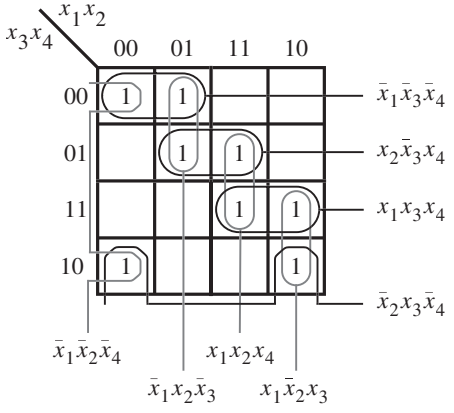


Figura 4.12 La función $f(x_1, \ldots, x_4) = \sum m(0, 2, 4, 5, 10, 11, 13, 15)$.

4.3 MINIMIZACIÓN DE FORMAS DE PRODUCTO DE SUMAS

Ahora que sabemos cómo encontrar las implementaciones de funciones de costo mínimo en suma de productos (SOP, *sum-of-products*), podemos aplicar las mismas técnicas y el principio de dualidad para obtener implementaciones de costo mínimo en producto de sumas (POS, *product-of-sums*). En este caso son los maxitérminos para los que $f = 0$ los que deben combinarse en términos suma que sean lo más grande posible. De nuevo, un término suma se considera más grande si cubre más maxitérminos, y cuanto más grande el término, menos costosa su implementación.

En la figura 4.13 se muestra la misma función descrita en la figura 4.9. Hay tres maxitérminos que deben cubrirse: M_4, M_5 y M_6. Pueden cubrirse mediante los dos términos suma mostrados en la figura, lo que conduce a la implementación siguiente:

$$f = (\bar{x}_1 + x_2)(\bar{x}_1 + x_3)$$

Un circuito correspondiente a esta expresión tiene dos compuertas OR y una AND, con dos entradas por cada una de ellas. Su costo es mayor que el de la implementación SOP equivalente derivada en la figura 4.9, que sólo requiere dos compuertas, una OR y una AND.

La función de la figura 4.10 se reproduce en la figura 4.14. Los maxitérminos para los que $f = 0$ pueden cubrirse como se muestra, llevan a la expresión

$$f = (x_2 + x_3)(x_3 + x_4)(\bar{x}_1 + \bar{x}_2 + \bar{x}_3 + \bar{x}_4)$$

Esta expresión representa un circuito con tres compuertas OR y una AND. Dos de las compuertas OR tienen dos entradas, y la tercera cuatro; la compuerta AND tiene tres entradas. Si se supone que las versiones complementada y sin complementar de las variables de entrada x_1 a x_4 están disponibles sin costo adicional, el costo de este circuito es 15, lo cual resulta mejor que la implementación SOP derivada de la figura 4.10, que requiere cinco compuertas y 13 entradas a un costo total de 18.

En general, como ya aprendimos en la sección 2.6.1, las implementaciones SOP y POS de una función pueden no representar el mismo costo. Se alienta al lector a que halle las implementaciones POS para las funciones de las figuras 4.11 y 4.12, y compare los costos con las formas SOP.

Hemos mostrado cómo obtener implementaciones POS de costo mínimo encontrando los términos suma más grandes que cubran todos los maxitérminos para los que $f = 0$. Otra forma

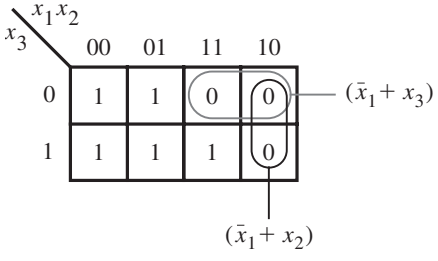


Figura 4.13 Minimización en POS de $f(x_1, x_2, x_3) = \Pi M(4, 5, 6)$.

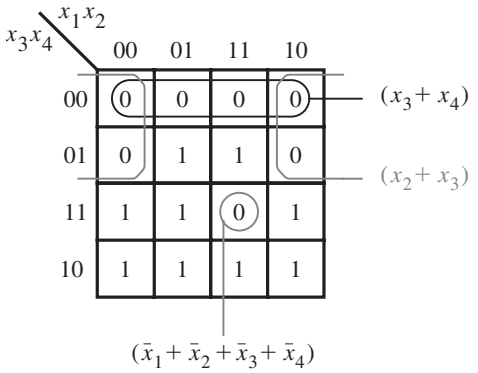


Figura 4.14 Minimización en POS de $f(x_1, \ldots, x_4) = \Pi M(0, 1, 4, 8, 9, 12, 15)$.

de obtener el mismo resultado radica en hallar una implementación SOP de costo mínimo del complemento de f. Luego puede aplicarse el teorema de DeMorgan a esta expresión para obtener la realización POS más simple porque $f = \overline{\overline{f}}$. Por ejemplo, la implementación SOP más simple de $\overline{f}$ en la figura 4.13 es

$$\overline{f} = x_1\overline{x}_2 + x_1\overline{x}_3$$

Al complementar esta expresión mediante el teorema de DeMorgan se produce

$$\begin{aligned} f = \overline{\overline{f}} &= \overline{x_1\overline{x}_2 + x_1\overline{x}_3} \\ &= \overline{x_1\overline{x}_2} \cdot \overline{x_1\overline{x}_3} \\ &= (\overline{x}_1 + x_2)(\overline{x}_1 + x_3) \end{aligned}$$

que es el mismo resultado obtenido antes.

Con este enfoque, para la función de la figura 4.14 se obtiene

$$\overline{f} = \overline{x}_2\overline{x}_3 + \overline{x}_3\overline{x}_4 + x_1x_2x_3x_4$$

Si se complementa esta expresión se obtiene

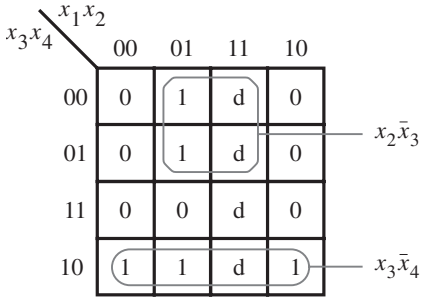
$$\begin{aligned} f = \overline{\overline{f}} &= \overline{\overline{x}_2\overline{x}_3 + \overline{x}_3\overline{x}_4 + x_1x_2x_3x_4} \\ &= \overline{\overline{x}_2\overline{x}_3} \cdot \overline{\overline{x}_3\overline{x}_4} \cdot \overline{x_1x_2x_3x_4} \\ &= (x_2 + x_3)(x_3 + x_4)(\overline{x}_1 + \overline{x}_2 + \overline{x}_3 + \overline{x}_4) \end{aligned}$$

que casa con la implementación antes derivada.

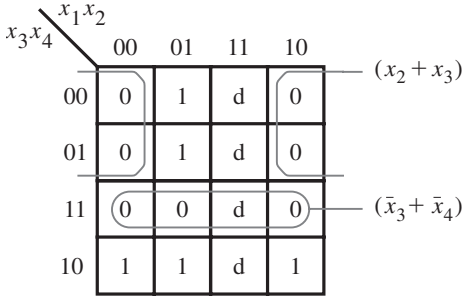
4.4 FUNCIONES ESPECIFICADAS DE MANERA INCOMPLETA

En los sistemas digitales suele ocurrir que ciertas condiciones de entrada nunca pueden suceder. Supóngase que x_1 y x_2 controlan dos interruptores interconectados de modo que ambos no pueden estar cerrados al mismo tiempo. Por tanto, los únicos tres estados posibles de los interruptores son que ambos estén abiertos o que uno esté abierto y el otro cerrado. Son posibles las combinaciones de entrada $(x_1, x_2) = 00$, 01 y 10, pero es seguro que 11 no ocurrirá. Entonces se dice que $(x_1, x_2) = 11$ es una *condición no-importa*, lo que significa que puede diseñarse un circuito con x_1 y x_2 como entradas e ignorar esta condición. Se dice que una función que tenga condiciones no-importa está *especificada de manera incompleta*.

Las condiciones no-importa, o simplemente los *no-importa*, sirven para sacar ventaja en el diseño de circuitos lógicos. Como tales combinaciones de entrada nunca ocurrirán, el diseñador puede suponer que el valor de la función para ellas es 1 o 0, lo que sea de mayor utilidad para encontrar una implementación de costo mínimo. En la figura 4.15 se ilustra esta idea. La función requerida tiene un valor de 1 para los mintérminos m_2, m_4, m_5, m_6 y m_{10}. Si se suponen los interruptores interconectados mencionados con antelación, las entradas x_1 y x_2 nunca serán iguales a 1 al mismo tiempo; por tanto, los mintérminos m_{12}, m_{13}, m_{14} y m_{15} se pueden usar como no-



a) Implementación SOP



b) Implementación POS

Figura 4.15 Dos implementaciones de la función $f(x_1, \ldots, x_4) = \sum m(2, 4, 5, 6, 10) + D(12, 13, 14, 15)$.

importa. Los no importa se denotan con la letra *d* en el mapa. En notación abreviada la función *f* se especifica como

$$f(x_1, \ldots, x_4) = \sum m(2, 4, 5, 6, 10) + D(12, 13, 14, 15)$$

donde *D* es el conjunto de no-importa.

En el inciso *a*) de la figura se indica la mejor implementación en suma de productos. Para formar los grupos más grandes posibles de 1 y, por ende, generar los implicantes primos de costo más bajo es preciso suponer que los no-importa D_{12}, D_{13} y D_{14} (que corresponden a los mintérminos m_{12}, m_{13} y m_{14}) tienen el valor de 1, mientras que D_{15} tiene el valor de 0. Entonces sólo existen dos implicantes primos, los cuales ofrecen una cobertura completa de *f*. La implementación resultante es

$$f = x_2\overline{x}_3 + x_3\overline{x}_4$$

En el inciso *b*) se indica cómo obtener la mejor implementación en producto de sumas. Los mismos valores se suponen para los no-importa. El resultado es

$$f = (x_2 + x_3)(\overline{x}_3 + \overline{x}_4)$$

La libertad para elegir el valor de los no-importa conduce a realizaciones muy simplificadas. Si inocentemente excluyéramos los no-importa de la síntesis de la función, al suponer que siempre tienen un valor de 0, la expresión SOP resultante sería

$$f = \overline{x}_1x_2\overline{x}_3 + \overline{x}_1x_3\overline{x}_4 + \overline{x}_2x_3\overline{x}_4$$

y la expresión POS sería

$$f = (x_2 + x_3)(\overline{x}_3 + \overline{x}_4)(\overline{x}_1 + \overline{x}_2)$$

Ambas expresiones tienen costos mayores que las obtenidas con una asignación más apropiada de valores para los no-importa.

Aunque los valores no-importa pueden asignarse de manera arbitraria, ello podría no desembocar en una implementación de costo mínimo de una función. Si existen *k* no-importa, entonces hay 2^k posibles formas de asignarles los valores 0 o 1. En el mapa de Karnaugh puede verse cómo hacer mejor esta asignación para hallar la implementación más simple.

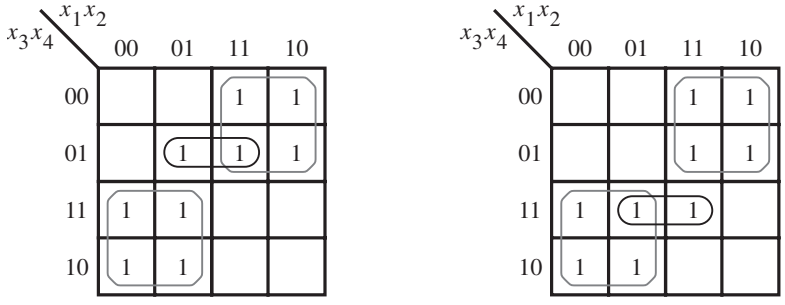
En el ejemplo anterior elegimos los no-importa D_{12}, D_{13} y D_{14} como iguales a 1 y D_{15} igual a 0 para ambas implementaciones, SOP y POS. Por ende, las expresiones derivadas representan la misma función, que también podría especificarse como $\sum m(2, 4, 5, 6, 10, 12, 13, 14)$. Asignar los mismos valores a los no-importa para ambas implementaciones, SOP y POS, no siempre es lo mejor. En ocasiones puede ser ventajoso dar a un no-importa en particular el valor 1 para la implementación SOP y el 0 para la POS, o viceversa. En tales casos, las expresiones óptimas SOP y POS representarán funciones diferentes, pero sólo diferirán por las combinaciones que correspondan a dichos no-importa. En el ejemplo 4.24 de la sección 4.14 se ilustra esta posibilidad.

El uso de interruptores interconectados para ilustrar cómo pueden ocurrir en un sistema real las condiciones no-importa puede parecer un tanto artificioso. Sin embargo, en los capítulos 6, 8 y 9 presentaremos numerosos ejemplos de no-importa que ocurren en el curso del diseño práctico de circuitos digitales.

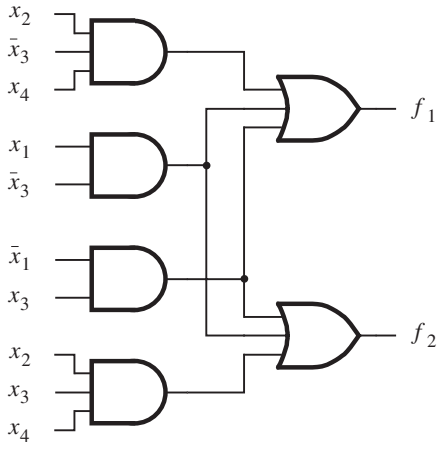
4.5 CIRCUITOS DE SALIDA MÚLTIPLE

En todos los ejemplos previos hemos considerado funciones solas y sus implementaciones en circuito. En los sistemas digitales prácticos es preciso implementar varias funciones como parte de algún circuito lógico grande. Los circuitos que implementan tales funciones suelen combinarse en un solo circuito menos costoso con varias salidas compartiendo algunas de las compuertas necesarias en la implementación de funciones individuales.

Ejemplo 4.1 En la figura 4.16 se ofrece un ejemplo de cómo se comparte una compuerta. Se deben implementar dos funciones, f_1 y f_2, de las mismas variables. Las implementaciones de costo mínimo



a) Función f_1 *b*) Función f_2



c) Circuito combinado para f_1 y f_2

Figura 4.16 Ejemplo de síntesis de salida múltiple.

para ellas se obtienen como se muestra en los incisos *a*) y *b*) de la figura. Esto resulta en las expresiones

$$f_1 = x_1\overline{x}_3 + \overline{x}_1x_3 + x_2\overline{x}_3x_4$$
$$f_2 = x_1\overline{x}_3 + \overline{x}_1x_3 + x_2x_3x_4$$

El costo de f_1 es cuatro compuertas y 10 entradas, para un total de 14. El costo de f_2 es el mismo. Por ende, el costo total es 28 si ambas funciones se implementan en circuitos separados. Si los dos circuitos se combinan en uno solo con dos salidas, es posible una realización menos costosa. Puesto que los dos primeros términos producto son idénticos en ambas expresiones, no se necesita duplicar las compuertas AND que las implementan. El circuito combinado se muestra en la figura 4.16*c*. Su costo es de seis compuertas y 16 entradas, para un total de 22.

En este ejemplo redujimos el costo global encontrando las realizaciones de costo mínimo de f_1 y f_2, y luego compartiendo las compuertas que implementan los términos producto comunes. Esta estrategia no siempre resulta la mejor, como se muestra en el ejemplo siguiente.

Ejemplo 4.2

En la figura 4.17 se presentan dos funciones que hay que implementar con un solo circuito. Las realizaciones de costo mínimo de las funciones individuales f_3 y f_4 se obtienen de los incisos *a*) y *b*) de la figura.

$$f_3 = \overline{x}_1x_4 + x_2x_4 + \overline{x}_1x_2x_3$$
$$f_4 = x_1x_4 + \overline{x}_2x_4 + \overline{x}_1x_2x_3\overline{x}_4$$

Ninguna de las compuertas AND puede compartirse, lo que significa que el costo del circuito combinado sería de seis compuertas AND, dos compuertas OR y 21 entradas, para un total de 29.

Pero varias otras realizaciones son posibles. En lugar de derivar las expresiones para f_3 y f_4 usando sólo implicantes primos, se buscan otros implicantes que puedan compartirse de ventajosamente en la realización combinada de las funciones. En la figura 4.17*c* se muestra la mejor elección de implicantes, lo que produce la realización

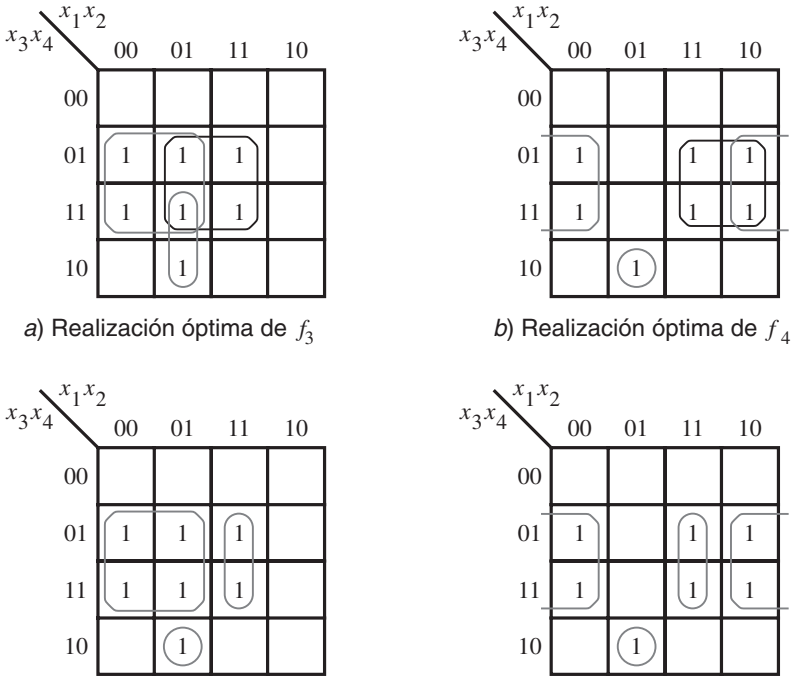
$$f_3 = x_1x_2x_4 + \overline{x}_1x_2x_3\overline{x}_4 + \overline{x}_1x_4$$
$$f_4 = x_1x_2x_4 + \overline{x}_1x_2x_3\overline{x}_4 + \overline{x}_2x_4$$

Los primeros dos implicantes son idénticos en ambas expresiones. El circuito resultante se muestra en la figura 4.17*d*. Tiene el costo de seis compuertas y 17 entradas, para un total de 23.

Ejemplo 4.3

En el ejemplo 4.1 se buscaba la mejor implementación en SOP para las funciones f_1 y f_2 de la figura 4.16. Ahora consideraremos la implementación en POS de las mismas funciones. Las expresiones en POS de costo mínimo para f_1 y f_2 son

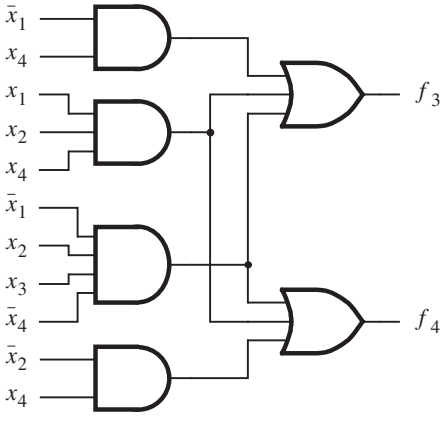
$$f_1 = (\overline{x}_1 + \overline{x}_3)(x_1 + x_2 + x_3)(x_1 + x_3 + x_4)$$
$$f_2 = (x_1 + x_3)(\overline{x}_1 + x_2 + \overline{x}_3)(\overline{x}_1 + \overline{x}_3 + x_4)$$



a) Realización óptima de f_3

b) Realización óptima de f_4

c) Realización óptima de f_3 y f_4 juntas



d) Circuito combinado para f_3 y f_4

Figura 4.17 Otro ejemplo de síntesis de salida múltiple.

En estas expresiones no hay términos suma comunes que puedan compartirse en la implementación. Más aún, a partir de los mapas de Karnaugh de la figura 4.16 es claro que no hay término suma (que cubra las celdas donde $f_1 = f_2 = 0$) que pueda usarse provechosamente en la realización tanto de f_1 como de f_2. Por tanto, la mejor elección es implementar cada función por separado, de acuerdo con las expresiones precedentes. Cada función requiere tres compuertas OR, una AND y 11 entradas. En consecuencia, el costo total del circuito que implementa ambas funciones es 30. Esta realización es más costosa que la realización en SOP derivada en el ejemplo 4.1.

Ejemplo 4.4

Considere ahora la realización en POS de las funciones f_3 y f_4 de la figura 4.17. Las expresiones en POS de costo mínimo para f_3 y f_4 son

$$f_3 = (x_3 + x_4)(x_2 + x_4)(\overline{x}_1 + x_4)(\overline{x}_1 + x_2)$$
$$f_4 = (x_3 + x_4)(x_2 + x_4)(\overline{x}_1 + x_4)(x_1 + \overline{x}_2 + \overline{x}_4)$$

Los primeros tres términos suma son los mismos tanto en f_3 como en f_4; pueden compartirse en un circuito combinado. Esos términos requieren tres compuertas OR y seis entradas. Además, para f_3 se necesita una compuerta OR de dos entradas y una compuerta AND de cuatro entradas, y para f_4 se requiere una compuerta OR de tres entradas y una AND de cuatro. Por tanto, el circuito combinado comprende cinco compuertas OR, dos AND y 19 entradas, para un costo total de 26. Este costo es ligeramente superior que el del circuito derivado en el ejemplo 4.2.

Estos ejemplos muestran que las complejidades de las mejores implementaciones en SOP o POS de funciones concretas pueden ser muy diferentes. Para las funciones de las figuras 4.16 y 4.17 la forma SOP da mejores resultados. Pero si uno está interesado en la implementación de los complementos de las cuatro funciones de estas figuras, entonces la forma POS sería menos costosa.

Las más modernas herramientas CAD para sintetizar funciones lógicas automáticamente desarrollarán los tipos de optimaciones que se ilustran en los ejemplos precedentes.

4.6 SÍNTESIS MULTINIVEL

En las secciones anteriores nuestro objetivo consistió en encontrar una realización de costo mínimo, en suma de productos o en producto de sumas, de una función lógica. Los circuitos lógicos de este tipo tienen *dos niveles* (etapas) de compuertas. En la forma de suma de productos, el primer nivel comprende compuertas AND conectadas a una compuerta OR de segundo nivel. En la forma de producto de sumas, las compuertas OR de primer nivel alimentan la compuerta AND de segundo nivel. Hemos supuesto que ambas versiones de las variables de entrada, verdaderas y complementadas, están disponibles, de modo que no se necesitan compuertas NOT para complementar las variables.

Una realización de dos niveles es eficiente para funciones de unas cuantas variables. Sin embargo, conforme el número de entradas aumenta, un circuito de dos niveles puede resultar en

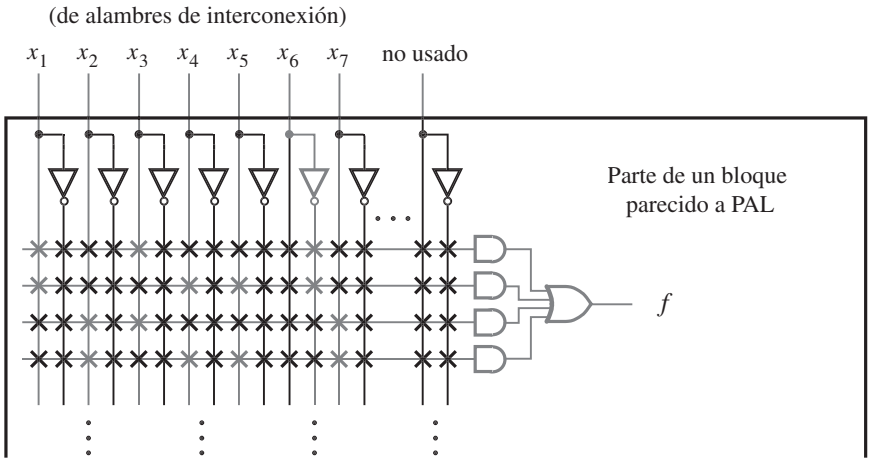


Figura 4.18 Implementación en un CPLD.

problemas de carga de entrada. Si éste es o no un conflicto depende del tipo de tecnología que se use para implementar el circuito. Considérese la función siguiente:

$$f(x_1, \ldots, x_7) = x_1x_3\overline{x}_6 + x_1x_4x_5\overline{x}_6 + x_2x_3x_7 + x_2x_4x_5x_7$$

Se trata de una expresión SOP de costo mínimo. Considérese ahora la implementación de *f* en dos tipos de PLD: un CPLD y un FPGA. En la figura 4.18 se muestra una parte de uno de los bloques parecidos a PAL de la figura 3.33. En la figura se indica en gris los circuitos utilizados para realizar la función *f*. Es claro que la forma SOP de la función es la adecuada para la arquitectura de chip del CPLD.

A continuación considérese la implementación de *f* en un FPGA. Para este ejemplo usaremos el FPGA presentado en la figura 3.39, que contiene dos entradas LUT. Como la expresión SOP para *f* requiere operaciones AND de tres y cuatro entradas, y una OR de cuatro, no puede implementarse directamente en este FPGA. El problema es que la carga de entrada requerida para implementar la función es muy alta para la arquitectura del chip objetivo.

Para resolver el problema de la carga de entrada, *f* debe expresarse en una forma que tenga más de dos niveles de operaciones lógicas. Tal forma se llama *expresión lógica multinivel*. Hay varios enfoques para sintetizar circuitos multinivel. Explicaremos dos importantes técnicas conocidas como *factorización* y *descomposición funcional*.

4.6.1 FACTORIZACIÓN

La propiedad distributiva presentada en la sección 2.5 permite factorizar la expresión anterior para *f* del modo siguiente

$$\begin{aligned} f &= x_1\overline{x}_6(x_3 + x_4x_5) + x_2x_7(x_3 + x_4x_5) \\ &= (x_1\overline{x}_6 + x_2x_7)(x_3 + x_4x_5) \end{aligned}$$

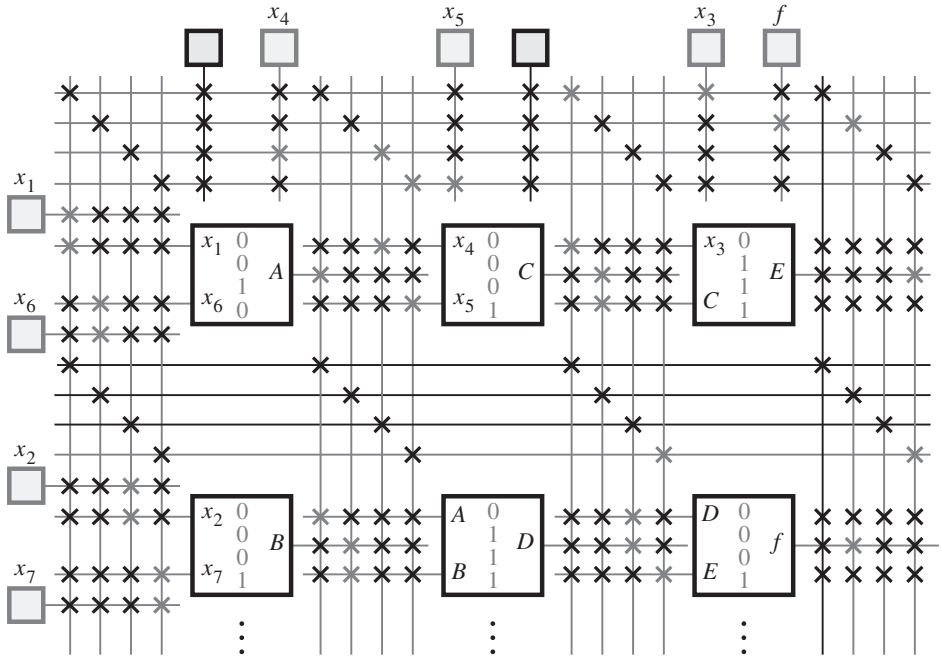


Figura 4.19 Implementación en un FPGA.

El circuito correspondiente tiene una carga de entrada máxima de dos; por tanto, se puede realizar usando LUT de dos entradas. En la figura 4.19 se brinda una posible implementación con el FPGA de la figura 3.39. Nótese que una función de dos variables que deba realizarse por cada LUT se indica en la caja que representa a la LUT.

Problema de carga de entrada

En el ejemplo anterior, las restricciones de carga de entrada fueron ocasionadas por la estructura fija del FPGA, donde cada LUT tiene sólo dos entradas. Sin embargo, aun cuando la arquitectura del chip objetivo no sea fija, la carga de entrada puede seguir siendo un problema. Para ilustrar esta situación, considérese la implementación de un circuito en un chip a la medida. Recuérdese que los chips a la medida contienen un gran número de compuertas. Si el chip se fabrica con tecnología CMOS, entonces habrá limitaciones de carga de entrada, como se expuso en la sección 3.8.8. En esta tecnología el número de entradas a una compuerta lógica debe ser pequeño. Por ejemplo, tal vez se quiera limitar la cantidad de entradas a una compuerta AND para que sean menos de cinco. Con esta restricción, si una expresión lógica incluye un término producto de siete entradas tendríamos que utilizar dos compuertas AND de cuatro entradas, como se indica en la figura 4.20.

Se puede usar factorización para enfrentar el problema de la carga de entrada. Supóngase de nuevo que las compuertas disponibles tienen una carga de entrada máxima de cuatro y que se quiere realizar la función

$$f = x_1\overline{x}_2x_3\overline{x}_4x_5x_6 + x_1x_2\overline{x}_3\overline{x}_4\overline{x}_5x_6$$

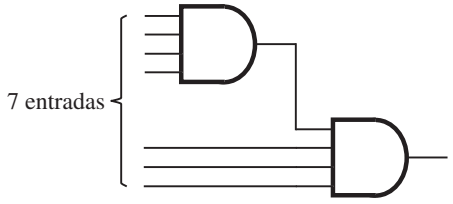


Figura 4.20 Uso de compuertas AND de cuatro entradas para realizar un término producto de siete entradas.

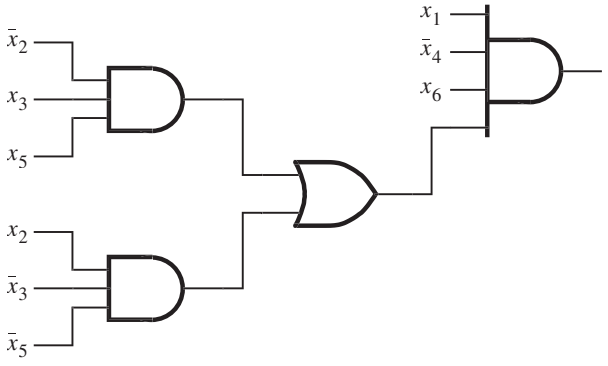


Figura 4.21 Circuito factorizado.

Ésta es una expresión mínima de suma de productos. Si empleamos el enfoque de la figura 4.20, se necesitarían cuatro compuertas AND y una OR para implementar esta expresión. Una mejor solución es factorizar la expresión como sigue

$$f = x_1\bar{x}_4x_6(\bar{x}_2x_3x_5 + x_2\bar{x}_3\bar{x}_5)$$

Entonces bastan tres compuertas AND y una OR para la realización de la función requerida, como se muestra en la figura 4.21.

Ejemplo 4.5 En situaciones prácticas, un diseñador de circuitos lógicos con frecuencia encuentra especificaciones que conducen naturalmente a un diseño inicial donde las expresiones lógicas están en forma factorizada. Suponga que se necesita un circuito que satisface los requisitos siguientes. Existen cuatro entradas: x_1, x_2, x_3 y x_4. Una salida, f_1, debe tener el valor 1 si al menos una de las entradas x_1 y x_2 es igual a 1 y tanto x_3 como x_4 son iguales a 1; también debe ser 1 si $x_1 = x_2 = 0$ y x_3 o x_4 es 1. En todos los demás casos $f_1 = 0$. Una salida diferente, f_2, será igual a 1 en todos los casos excepto cuando x_1 y x_2 sean iguales a 0 o cuando x_3 y x_4 sean iguales a 0.

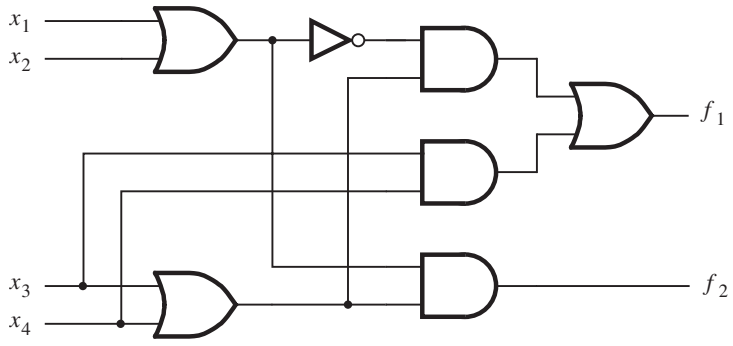


Figura 4.22 Circuito del ejemplo 4.5.

A partir de estas especificaciones, la función f_1 puede expresarse como

$$f_1 = (x_1 + x_2)x_3x_4 + \overline{x}_1\overline{x}_2(x_3 + x_4)$$

Esta expresión puede simplificarse en

$$f_1 = x_3x_4 + \overline{x}_1\overline{x}_2(x_3 + x_4)$$

que el lector puede comprobar con un mapa de Karnaugh.

La segunda función, f_2, se define con más facilidad en términos de su complemento, tal que

$$\overline{f}_2 = \overline{x}_1\overline{x}_2 + \overline{x}_3\overline{x}_4$$

Luego, el uso del teorema de DeMorgan produce

$$f_2 = (x_1 + x_2)(x_3 + x_4)$$

que es la expresión de costo mínimo para f_2; el costo aumenta de manera significativa si se usa la forma SOP.

Puesto que el objetivo es diseñar el circuito combinado de costo más bajo que implemente f_1 y f_2, parece que el mejor resultado se puede lograr si se usan las formas factorizadas para ambas funciones, caso en el que el término suma $(x_3 + x_4)$ puede compartirse. Más aún, al observar que $\overline{x}_1\overline{x}_2 = \overline{x_1 + x_2}$, el término suma $(x_1 + x_2)$ también puede compartirse si se expresa f_1 en la forma

$$f_1 = x_3x_4 + \overline{x_1 + x_2}(x_3 + x_4)$$

Entonces el circuito combinado, que se muestra en la figura 4.22, comprende tres compuertas OR, tres AND, una NOT y 13 entradas, para un total de 20.

Impacto sobre la complejidad del cableado

El espacio de los chips de circuitos integrados está ocupado por los circuitos que implementan las compuertas lógicas y por los cables necesarios para hacer las conexiones entre ellas. La

cantidad de espacio necesario para cablear es una parte sustancial del área del chip. Por tanto, resulta útil mantener la complejidad del cableado cuan baja sea posible.

En una expresión lógica cada literal corresponde a un cable en el circuito que porta la señal lógica deseada. Puesto que la factorización reduce el número de literales, proporciona un poderoso mecanismo para reducir la complejidad del cableado de un circuito lógico. En el proceso de síntesis, las herramientas CAD consideran muchos aspectos, incluso el costo del circuito, la carga de entrada y la complejidad del cableado.

4.6.2 DESCOMPOSICIÓN FUNCIONAL

En los ejemplos anteriores, que ilustran el enfoque de factorización, se usaron circuitos multinivel para manejar las limitaciones de carga de entrada. Sin embargo, tales circuitos pueden ser preferibles a sus equivalentes de dos niveles aun si la carga de entrada no es un problema. En ciertos casos, los circuitos multinivel pueden reducir el costo de implementación. Por otra parte, casi siempre implican retrasos de propagación mayores porque utilizan múltiples etapas de compuertas lógicas. Analizaremos estos temas mediante algunos ejemplos.

En términos de cableado y compuertas lógicas, la complejidad de un circuito lógico a menudo puede reducirse por medio de la *descomposición* de un circuito de dos niveles en subcircuitos, donde uno o más de ellos implementan funciones que pueden usarse en varios lugares para construir el circuito final. Para lograr este objetivo, una expresión lógica de dos niveles se sustituye con dos o más expresiones nuevas, que entonces se combinan para definir un circuito multinivel. Podemos ilustrar esta idea con un ejemplo simple.

Ejemplo 4.6 Considere la expresión de costo mínimo en suma de productos

$$f = \overline{x}_1x_2x_3 + x_1\overline{x}_2x_3 + x_1x_2x_4 + \overline{x}_1\overline{x}_2x_4$$

y suponga que las entradas x_1 a x_4 sólo están disponibles en su forma verdadera. Entonces la expresión define un circuito con cuatro compuertas AND, una OR, dos NOT y 18 entradas (cables) a todas ellas. La carga de entrada es tres para las compuertas AND y cuatro para la OR. El lector debe observar que, en este caso, hemos incluido el costo de las compuertas NOT necesarias para complementar x_1 y x_2, en vez de suponer que ambas versiones, verdadera y complementada, de todas las variables de entrada estaban disponibles, como se hizo antes.

Al factorizar x_3 de los primeros dos términos y x_4 de los dos últimos esta expresión se convierte en

$$f = (\overline{x}_1x_2 + x_1\overline{x}_2)x_3 + (x_1x_2 + \overline{x}_1\overline{x}_2)x_4$$

Ahora sea $g(x_1, x_2) = \overline{x}_1x_2 + x_1\overline{x}_2$ y observe que

$$\begin{aligned}
\overline{g} &= \overline{\overline{x}_1x_2 + x_1\overline{x}_2} \\
&= \overline{\overline{x}_1x_2} \cdot \overline{x_1\overline{x}_2} \\
&= (x_1 + \overline{x}_2)(\overline{x}_1 + x_2) \\
&= x_1\overline{x}_1 + x_1x_2 + \overline{x}_2\overline{x}_1 + \overline{x}_2x_2 \\
&= 0 + x_1x_2 + \overline{x}_1\overline{x}_2 + 0 \\
&= x_1x_2 + \overline{x}_1\overline{x}_2
\end{aligned}$$

Entonces f puede escribirse como

$$f = gx_3 + \overline{g}x_4$$

que conduce al circuito mostrado en la figura 4.23, el cual requiere una compuerta OR adicional y una compuerta NOT para invertir el valor de g. Pero sólo necesita 15 entradas. Más aún, la carga de entrada más grande se redujo a dos. El costo de este circuito es menor que el de su equivalente de dos niveles. La consecuencia es un retraso de propagación mayor porque el circuito tiene tres niveles más de lógica.

En este ejemplo, la subfunción g es una función de las variables x_1 y x_2. Se utiliza como entrada al resto del circuito que completa la realización de la función requerida f. Sea h la función de esta parte del circuito, que depende sólo de tres entradas: g, x_3 y x_4. Luego la realización descompuesta de f puede expresarse algebraicamente como

$$f(x_1, x_2, x_3, x_4) = h[g(x_1, x_2), x_3, x_4]$$

La estructura de esta descomposición puede describirse en forma de diagrama de bloques como se muestra en la figura 4.24.

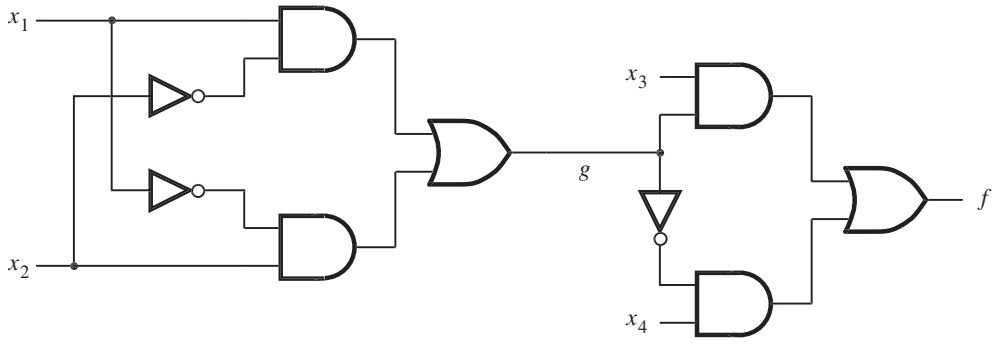


Figura 4.23 Circuito lógico del ejemplo 4.6.

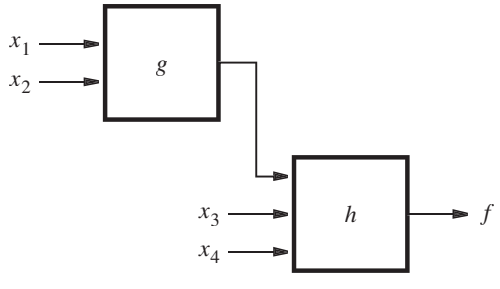
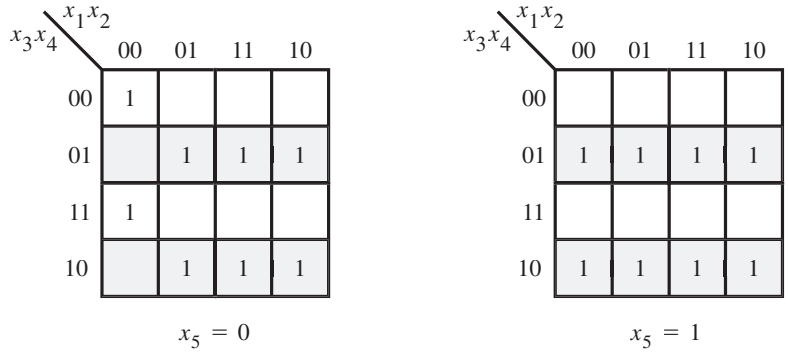


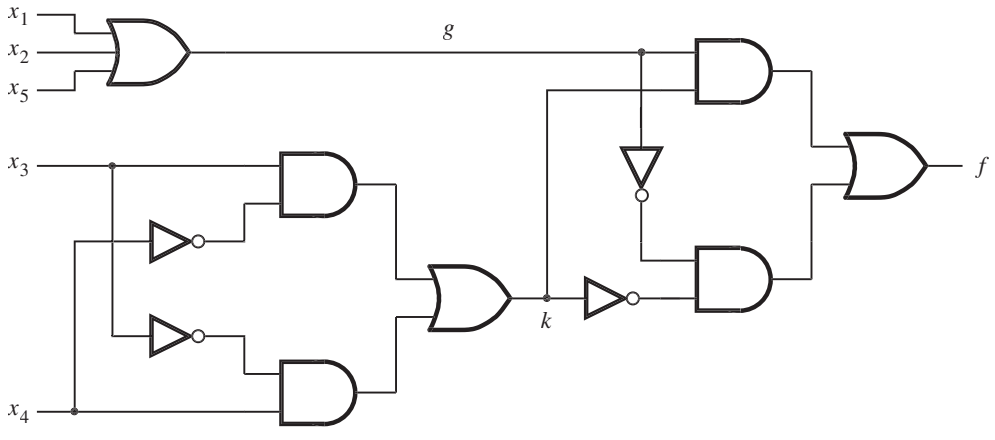
Figura 4.24 Estructura de la descomposición del ejemplo 4.6.

Aunque no es evidente a partir del primer ejemplo, la descomposición funcional puede conducir a mayores reducciones en la complejidad y el costo de los circuitos. El lector obtendrá un buen indicador de este beneficio a partir del ejemplo que sigue.

Ejemplo 4.7 En la figura 4.25*a* se define una función *f* de cinco variables en la forma de mapa de Karnaugh. Si se busca una buena descomposición para ella es necesario identificar primero las variables que se usarán como entradas a una subfunción. Una pista útil se obtiene a partir de los patrones de 1 en el mapa. Nótese que sólo existen dos patrones en las filas: la segunda y la cuarta tienen uno, resaltado en gris, mientras que la primera y la segunda tienen el otro. Una vez que se establece en qué fila se halla cada patrón, entonces el patrón mismo sólo depende de las variables que definen



a) Mapa de Karnaugh para la función *f*



b) Circuito obtenido al usar descomposición

Figura 4.25 Descomposición para el ejemplo 4.7.

las columnas en cada fila: x_1, x_2 y x_5. Sea $g(x_1, x_2, x_5)$ una subfunción que representa el patrón en las filas 2 y 4. Esta subfunción es justo

$$g = x_1 + x_2 + x_5$$

porque el patrón tiene un 1 donde cualquiera de dichas variables es igual a 1. Para especificar la ubicación de las filas donde ocurre el patrón g se utilizan las variables x_3 y x_4. Los términos $\overline{x}_3 x_4$ y $x_3 \overline{x}_4$ identifican la segunda y cuarta filas, respectivamente. Por ende, la expresión $(\overline{x}_3 x_4 + x_3 \overline{x}_4) \cdot g$ representa la parte de f definida en las filas 2 y 4.

A continuación, hay que encontrar una realización para el patrón de las filas 1 y 3, el cual tiene un 1 sólo en la celda donde $x_1 = x_2 = x_5 = 0$, que corresponde al término $\overline{x}_1 \overline{x}_2 \overline{x}_5$. Pero es posible hacer útil la observación de que este término es justo un complemento de g. La ubicación de las filas 1 y 3 se identifica por los términos $\overline{x}_3 \overline{x}_4$ y $x_3 x_4$, respectivamente. Por tanto, la expresión $(\overline{x}_3 \overline{x}_4 + x_3 x_4) \cdot \overline{g}$ representa f en las filas 1 y 3.

Se puede hacer otra útil observación. Las expresiones $(\overline{x}_3 x_4 + x_3 \overline{x}_4)$ y $(\overline{x}_3 \overline{x}_4 + x_3 x_4)$ son complementos una de la otra, como se muestra en el ejemplo 4.6. En consecuencia, si se hace $k(x_3, x_4) = \overline{x}_3 x_4 + x_3 \overline{x}_4$, la descomposición completa de f puede establecerse como

$$\begin{aligned} f(x_1, x_2, x_3, x_4, x_5) &= h[g(x_1, x_2, x_5), k(x_3, x_4)] \\ &= kg + \overline{k}\,\overline{g} \end{aligned}$$

donde

$$\begin{aligned} g &= x_1 + x_2 + x_5 \\ k &= \overline{x}_3 x_4 + x_3 \overline{x}_4 \end{aligned}$$

El circuito resultante se muestra en la figura 4.25*b*. Requiere un total de 11 compuertas y 19 entradas. La carga de entrada más grande es tres.

En contraste, una expresión de costo mínimo en suma de productos para f es

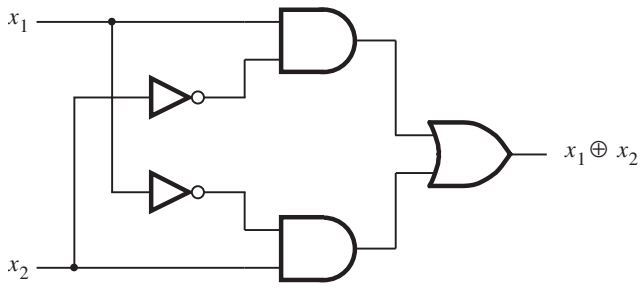
$$f = x_1 \overline{x}_3 x_4 + x_1 x_3 \overline{x}_4 + x_2 \overline{x}_3 x_4 + x_2 x_3 \overline{x}_4 + \overline{x}_3 x_4 x_5 + x_3 \overline{x}_4 x_5 + \overline{x}_1 \overline{x}_2 \overline{x}_3 \overline{x}_4 \overline{x}_5 + \overline{x}_1 \overline{x}_2 x_3 x_4 \overline{x}_5$$

El circuito correspondiente requiere un total de 14 compuertas (incluidas las cinco compuertas NOT para complementar las entradas primarias) y 41 entradas. La carga de entrada para la compuerta OR de salida es ocho. Obviamente, la descomposición funcional resulta en una implementación mucho más simple de esta función.

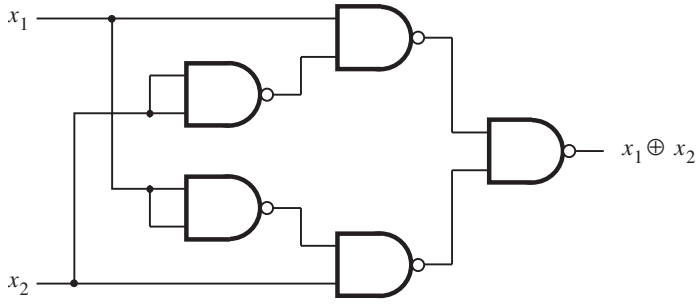
En los dos ejemplos anteriores, la descomposición es tal que una subfunción descompuesta depende de ciertas variables de entrada primarias, mientras que el resto de la implementación depende de las demás las variables. Tales descomposiciones reciben el nombre de *descomposiciones separadas* en los libros técnicos. Es posible tener una *descomposición no-separada*, donde las variables de la subfunción también se usan en la realización del resto del circuito. El ejemplo siguiente ilustra esta posibilidad.

Ejemplo 4.8

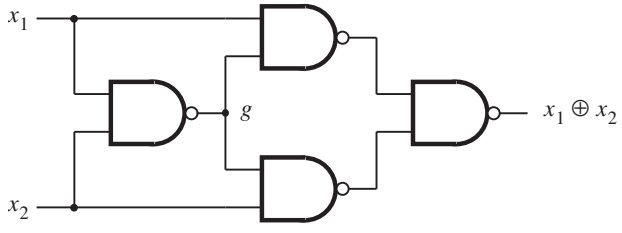
La OR exclusiva (XOR) es una función muy útil. En la sección 3.9.1 mostramos cómo puede realizarse con un circuito especial. También se puede realizar con compuertas AND y OR como



a) Implementación en suma de productos



b) Implementación con compuertas NAND



c) Implementación óptima con compuertas NAND

Figura 4.26 Implementación de XOR.

se muestra en la figura 4.26*a*. En la sección 2.7 explicamos cómo cualquier circuito AND-OR puede realizarse como un circuito NAND-NAND que tenga la misma estructura.

Ahora intentaremos explotar la descomposición funcional para encontrar una mejor implementación de XOR usando únicamente compuertas NAND. Sea el símbolo ↑ la representación de la operación NAND, de modo que $x_1 \uparrow x_2 = \overline{x_1 \cdot x_2}$. Una expresión en suma de productos para la función XOR es

$$x_1 \oplus x_2 = x_1\overline{x}_2 + \overline{x}_1 x_2$$

Con base en el análisis de la sección 2.7, esta expresión puede escribirse en términos de operaciones NAND como

$$x_1 \oplus x_2 = (x_1 \uparrow \overline{x}_2) \uparrow (\overline{x}_1 \uparrow x_2)$$

Esta expresión requiere cinco compuertas NAND, y se implementa mediante el circuito de la figura 4.26*b*. Observe que un inversor se implementa con una compuerta NAND de dos entradas unidas.

Para hallar una descomposición puede manipularse el término $(x_1 \uparrow \overline{x}_2)$ del modo siguiente:

$$(x_1 \uparrow \overline{x}_2) = \overline{(x_1\overline{x}_2)} = \overline{(x_1(\overline{x}_1 + \overline{x}_2))} = (x_1 \uparrow (\overline{x}_1 + \overline{x}_2))$$

Es posible efectuar una manipulación similar de $(\overline{x}_1 \uparrow x_2)$ para generar

$$x_1 \oplus x_2 = (x_1 \uparrow (\overline{x}_1 + \overline{x}_2)) \uparrow ((\overline{x}_1 + \overline{x}_2) \uparrow x_2)$$

El teorema de DeMorgan afirma que $\overline{x}_1 + \overline{x}_2 = x_1 \uparrow x_2$; por tanto, puede escribirse

$$x_1 \oplus x_2 = (x_1 \uparrow (x_1 \uparrow x_2)) \uparrow ((x_1 \uparrow x_2) \uparrow x_2)$$

Ahora se tiene una descomposición

$$x_1 \oplus x_2 = (x_1 \uparrow g) \uparrow (g \uparrow x_2)$$

$$g = x_1 \uparrow x_2$$

El circuito correspondiente, que sólo requiere cuatro compuertas NAND, se muestra en la figura 4.26*c*.

Aspectos prácticos

La descomposición funcional es una técnica poderosa para reducir la complejidad de los circuitos. También sirve para implementar funciones lógicas generales en circuitos que tienen restricciones inherentes. Por ejemplo, en los dispositivos lógicos programables (PLD), tema expuesto en el capítulo 3, es necesario "ajustar" el circuito lógico deseado en los bloques lógicos disponibles en dichos dispositivos. Los bloques disponibles son un blanco para subfunciones descompuestas que podrían emplearse para realizar funciones más grandes.

Un gran problema de la descomposición funcional consiste en encontrar las posibles subfunciones. Para funciones de muchas variables se debe probar un número enorme de posibilidades. Esta situación elimina los intentos por encontrar soluciones óptimas. En vez de ello se utilizan enfoques heurísticos que conducen a soluciones aceptables.

La explicación completa de la descomposición funcional y la factorización está más allá del ámbito de esta obra. El lector interesado puede consultar otras referencias [2-5]. Las modernas herramientas CAD usan mucho el concepto de descomposición.

4.6.3 CIRCUITOS NAND Y NOR MULTINIVEL

En la sección 2.7 mostramos que los circuitos de dos niveles que constan de compuertas AND y OR pueden convertirse fácilmente en circuitos que es posible realizar con compuertas NAND y NOR, usando el mismo arreglo de compuertas. En particular, un circuito AND-OR (suma de

productos) puede realizarse como un circuito NAND-NAND, mientras que un circuito OR-AND (producto de sumas) se convierte en un circuito NOR-NOR. Es posible usar el mismo enfoque de conversión para circuitos multinivel. Lo ilustramos con un ejemplo.

Ejemplo 4.9 En la figura 4.27*a* se presenta un circuito de cuatro niveles que consta de compuertas AND y OR. Primero derivemos un circuito funcionalmente equivalente que comprenda sólo compuertas NAND. Cada compuerta AND se convierte en una NAND al invertir su salida. Cada compuerta OR se convierte a una NAND al invertir sus entradas. Ésta es justo una aplicación del teorema de DeMorgan, como se ilustra en la figura 2.21*a*. En la figura 4.27*b* se muestran las inversiones necesarias en gris. Note que se aplica una inversión en ambos extremos de un cable dado. Ahora cada compuerta se convierte en una compuerta NAND. Esto explica el grueso de las inversiones agregadas al circuito original. Pero todavía hay cuatro inversiones que no forman parte de compuerta alguna; por tanto, deben implementarse por separado. Esas inversiones están en las entradas x_1, x_5, x_6 y x_7 y en la salida *f*. Pueden implementarse como dos compuertas NAND de dos entradas, donde las entradas están unidas. El circuito resultante se muestra en la figura 4.27*c*.

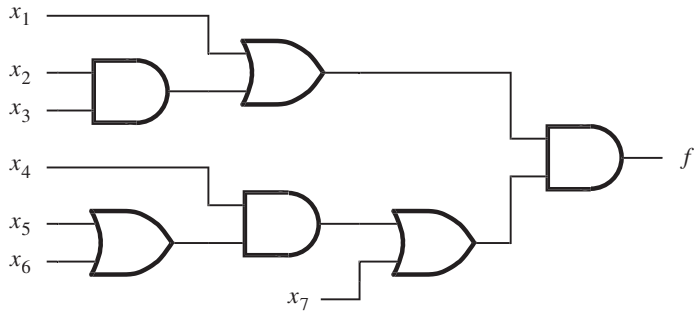
Es posible seguir un enfoque similar para convertir el circuito de la figura 4.27*a* en otro que comprenda sólo compuertas NOR. Una compuerta OR se convierte en una NOR invirtiendo su salida. Una AND se convierte en NOR si sus entradas se invierten, como se indica en la figura 2.21*b*. Con este enfoque, las inversiones necesarias para el circuito de nuestro ejemplo son las mostradas en gris en la figura 4.28*a*. Luego cada compuerta se convierte en una compuerta NOR. Las tres inversiones en las entradas x_2, x_3 y x_4 pueden realizarse como compuertas NOR de dos entradas, donde las entradas se unen. El circuito resultante se presenta en la figura 4.28*b*.

Es evidente que la topología básica de un circuito no cambia sustancialmente cuando se convierte de compuertas AND y OR en compuertas NAND o NOR. Sin embargo, puede ser necesario insertar compuertas adicionales para servir como compuertas NOT que implementen inversiones no absorbidas como parte de otras compuertas en el circuito.

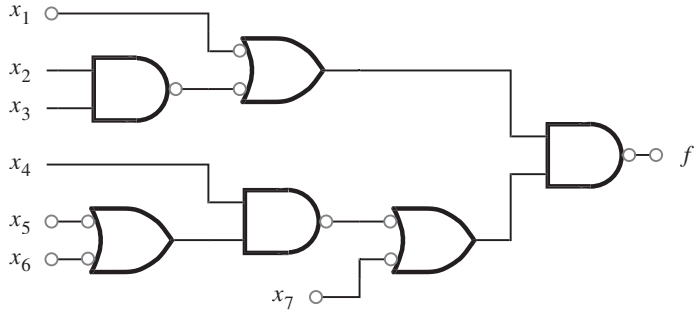
4.7 ANÁLISIS DE CIRCUITOS MULTINIVEL

En la sección precedente mostramos que puede resultar ventajoso implementar funciones lógicas mediante circuitos multinivel. También expusimos los enfoques más usados para sintetizar funciones de esta forma. En esta sección consideraremos la tarea de analizar un circuito para determinar la función que implementa.

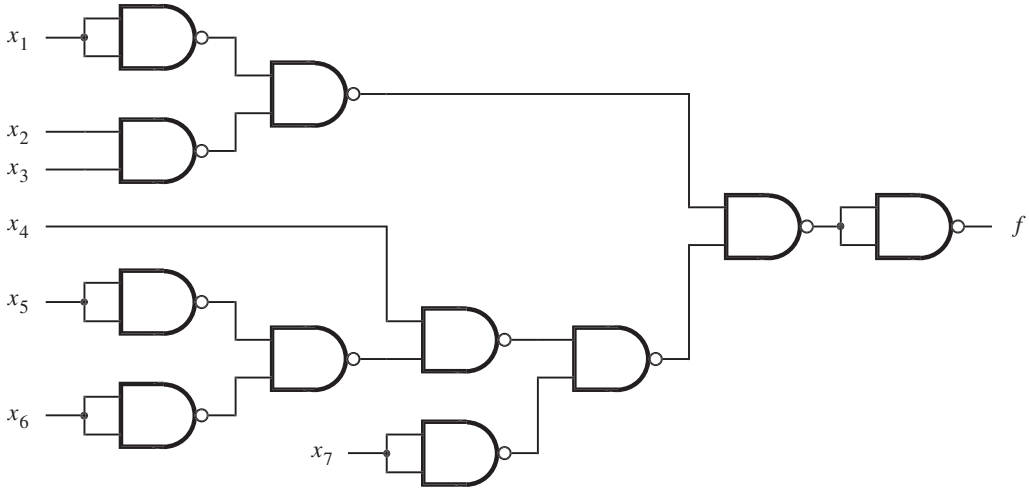
Para circuitos de dos niveles el proceso de análisis es simple. Si un circuito tiene una estructura AND-OR (NAND-NAND), entonces su función de salida puede escribirse en la forma SOP por inspección. De manera similar, es fácil derivar una expresión POS para un circuito OR-AND (NOR-NOR). La labor de análisis es más complicada para los circuitos multinivel porque es difícil escribir una expresión para la función por inspección. Hay que derivar la expresión deseada mediante el trazado del circuito y la determinación de su funcionalidad. El trazado puede realizarse primero desde el lado de entrada y luego hacia la salida, o primero en el lado de salida y después hacia atrás, a las entradas. En los puntos intermedios del circuito es preciso evaluar las subfunciones realizadas por las compuertas lógicas.



a) Circuito con compuertas AND y OR

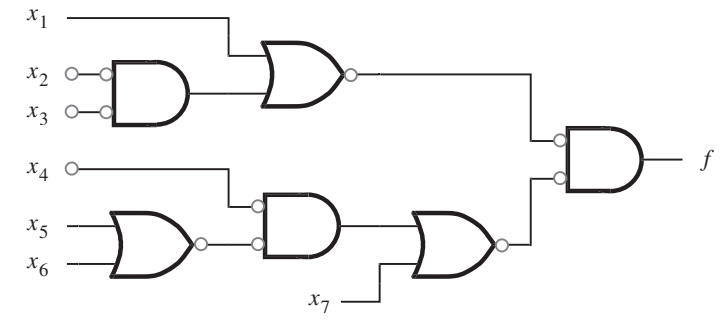


b) Inversiones necesarias para convertir a NAND

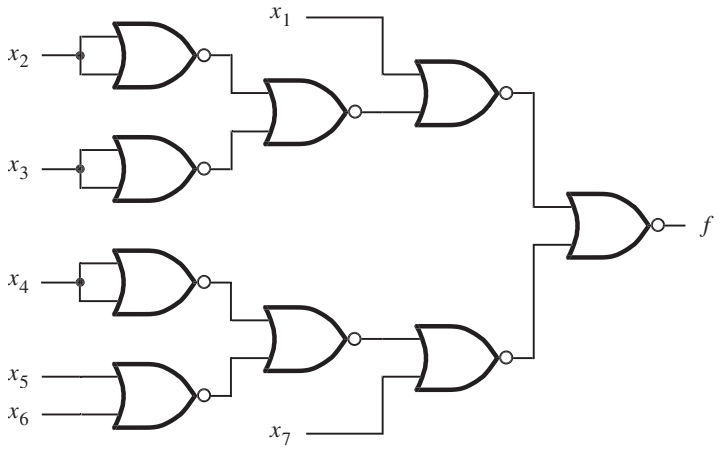


c) Circuito de compuerta NAND

Figura 4.27 Conversión a un circuito de compuerta NAND.



a) Inversiones necesarias para convertir a NOR



b) Circuito de compuerta NOR

Figura 4.28 Conversión a un circuito de compuerta NOR.

Ejemplo 4.10 En la figura 4.29 se copia el circuito de la figura 4.27*a*. Para determinar la función *f* que implementa puede considerarse la funcionalidad en puntos internos que son las salidas de varias compuertas. Tales puntos se etiquetan con P_1 a P_5 en la figura. Las funciones realizadas en ellos son

$$P_1 = x_2x_3$$
$$P_2 = x_5 + x_6$$
$$P_3 = x_1 + P_1 = x_1 + x_2x_3$$

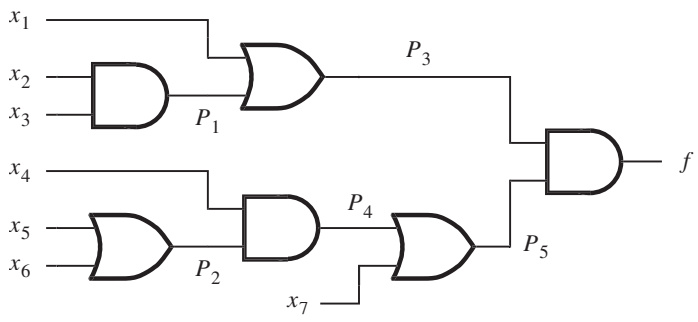


Figura 4.29 Circuito para el ejemplo 4.10.

$$P_4 = x_4P_2 = x_4(x_5 + x_6)$$
$$P_5 = P_4 + x_7 = x_4(x_5 + x_6) + x_7$$

Entonces f puede evaluarse como

$$\begin{aligned} f &= P_3P_5 \\ &= (x_1 + x_2x_3)(x_4(x_5 + x_6) + x_7) \end{aligned}$$

Al aplicar la propiedad distributiva para eliminar los paréntesis se obtiene

$$f = x_1x_4x_5 + x_1x_4x_6 + x_1x_7 + x_2x_3x_4x_5 + x_2x_3x_4x_6 + x_2x_3x_7$$

Note que la expresión representa un circuito que comprende seis compuertas AND, una OR y 25 entradas. El costo de este circuito de dos niveles es mayor que el del circuito de la figura 4.29, pero tiene menor retardo de propagación.

Ejemplo 4.11

En el ejemplo 4.7 derivamos el circuito de la figura 4.25*b*. Además de las compuertas AND y OR, el circuito tiene algunas compuertas NOT. Se reproduce en la figura 4.30, y los puntos internos se etiquetan con P_1 a P_{10}, como se muestra. Ocurren las subfunciones siguientes

$$P_1 = x_1 + x_2 + x_5$$
$$P_2 = \overline{x}_4$$
$$P_3 = \overline{x}_3$$
$$P_4 = x_3P_2$$
$$P_5 = x_4P_3$$
$$P_6 = P_4 + P_5$$
$$P_7 = \overline{P}_1$$
$$P_8 = \overline{P}_6$$

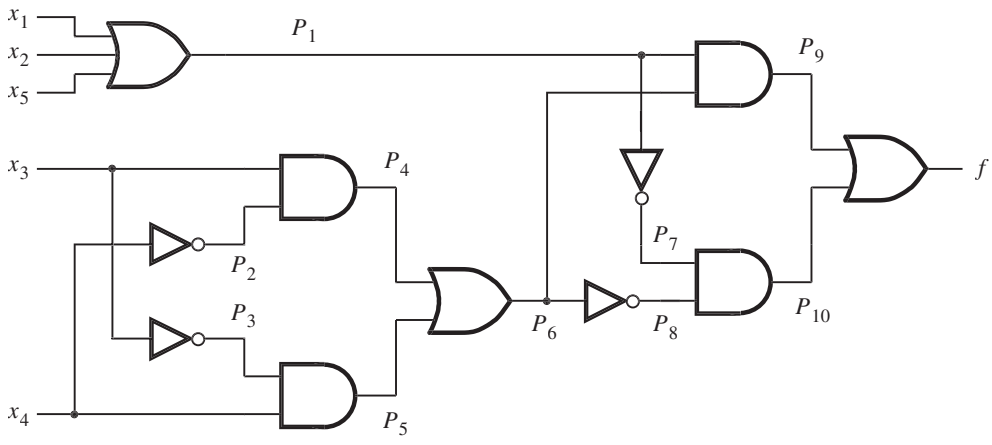


Figura 4.30 Circuito para el ejemplo 4.11.

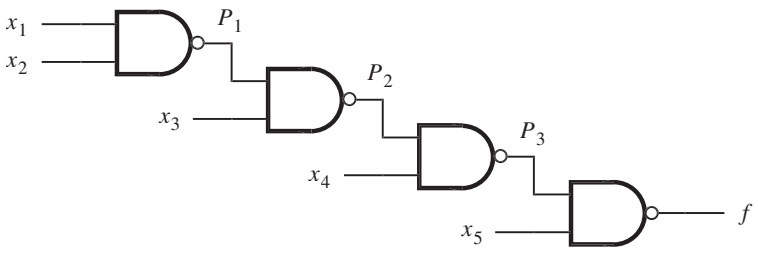
$$P_9 = P_1P_6$$
$$P_{10} = P_7P_8$$

Es posible derivar f trazando el circuito desde la salida hacia las entradas del modo siguiente

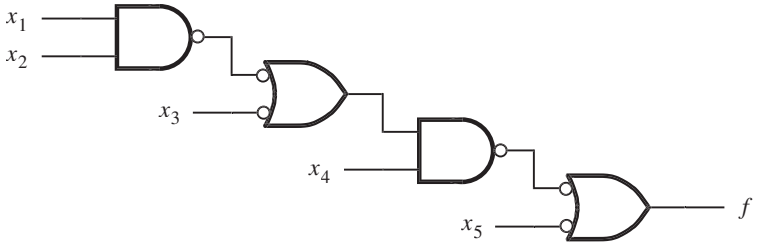
$$\begin{aligned}
f &= P_9 + P_{10} \\
&= P_1P_6 + P_7P_8 \\
&= (x_1 + x_2 + x_5)(P_4 + P_5) + \overline{P}_1\overline{P}_6 \\
&= (x_1 + x_2 + x_5)(x_3P_2 + x_4P_3) + \overline{x}_1\overline{x}_2\overline{x}_5\overline{P}_4\overline{P}_5 \\
&= (x_1 + x_2 + x_5)(x_3\overline{x}_4 + x_4\overline{x}_3) + \overline{x}_1\overline{x}_2\overline{x}_5(\overline{x}_3 + \overline{P}_2)(\overline{x}_4 + \overline{P}_3) \\
&= (x_1 + x_2 + x_5)(x_3\overline{x}_4 + \overline{x}_3x_4) + \overline{x}_1\overline{x}_2\overline{x}_5(\overline{x}_3 + x_4)(\overline{x}_4 + x_3) \\
&= x_1x_3\overline{x}_4 + x_1\overline{x}_3x_4 + x_2x_3\overline{x}_4 + x_2\overline{x}_3x_4 + x_5x_3\overline{x}_4 + x_5\overline{x}_3x_4 + \\
&\quad \overline{x}_1\overline{x}_2\overline{x}_5\overline{x}_3\overline{x}_4 + \overline{x}_1\overline{x}_2\overline{x}_5x_4x_3
\end{aligned}$$

Ésta es la misma expresión que la presentada en el ejemplo 4.7.

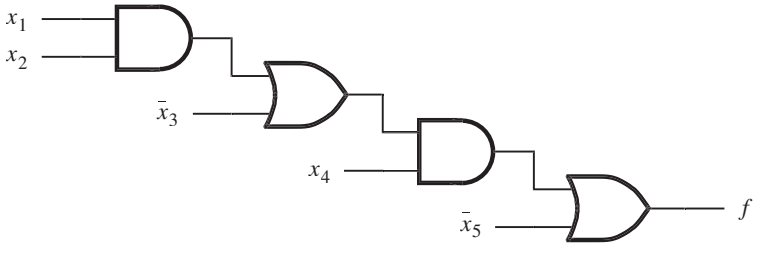
Ejemplo 4.12 Los circuitos que se basan en compuertas NAND y NOR son un poco más difíciles de analizar porque cada compuerta supone una inversión. En la figura 4.31*a* se describe un circuito de compuerta NAND simple que ilustra el efecto de las inversiones. Este circuito puede convertirse en uno con compuertas AND y OR invirtiendo el enfoque descrito en el ejemplo 4.9. Las burbujas que denotan inversiones pueden moverse, de acuerdo con el teorema de DeMorgan, como se indica en la figura 4.31*b*. Entonces el circuito puede convertirse en el presentado en el inciso



a) Circuito de compuertas NAND



b) Burbujas en movimiento para convertir a AND y OR



c) Circuito con compuertas AND y OR

Figura 4.31 Circuito para el ejemplo 4.12.

c) de ésta, el cual consta de compuertas AND y OR. Observe que, en el circuito convertido, las entradas x_3 y x_5 están complementadas. A partir de este circuito la función f se determina como

$$\begin{aligned} f &= (x_1x_2 + \overline{x}_3)x_4 + \overline{x}_5 \\ &= x_1x_2x_4 + \overline{x}_3x_4 + \overline{x}_5 \end{aligned}$$

No es necesario convertir un circuito NAND en uno con compuertas AND y OR para determinar su funcionalidad. Puede utilizarse el enfoque de los ejemplos 4.10 y 4.11 para derivar

f del modo siguiente. Sean P_1, P_2 y P_3 las etiquetas de los puntos internos, como se muestra en la figura 4.31*a*. Entonces

$$\begin{aligned}
P_1 &= \overline{x_1 x_2} \\
P_2 &= \overline{P_1 x_3} \\
P_3 &= \overline{P_2 x_4} \\
f &= \overline{P_3 x_5} = \overline{P}_3 + \overline{x}_5 \\
&= \overline{\overline{P_2 x_4}} + \overline{x}_5 = P_2 x_4 + \overline{x}_5 \\
&= \overline{P_1 x_3} x_4 + \overline{x}_5 = (\overline{P}_1 + \overline{x}_3) x_4 + \overline{x}_5 \\
&= (\overline{\overline{x_1 x_2}} + \overline{x}_3) x_4 + \overline{x}_5 \\
&= (x_1 x_2 + \overline{x}_3) x_4 + \overline{x}_5 \\
&= x_1 x_2 x_4 + \overline{x}_3 x_4 + \overline{x}_5
\end{aligned}$$

Ejemplo 4.13 El circuito de la figura 4.32 consta de compuertas NAND y NOR. Se puede analizar del modo siguiente

$$\begin{aligned}
P_1 &= \overline{x_2 x_3} \\
P_2 &= \overline{x_1 P_1} = \overline{x}_1 + \overline{P}_1 \\
P_3 &= \overline{x_3 x_4} = \overline{x}_3 + \overline{x}_4 \\
P_4 &= \overline{P_2 + P_3} \\
f &= \overline{P_4 + x_5} = \overline{P}_4 \overline{x}_5 \\
&= \overline{\overline{P_2 + P_3}} \cdot \overline{x}_5
\end{aligned}$$

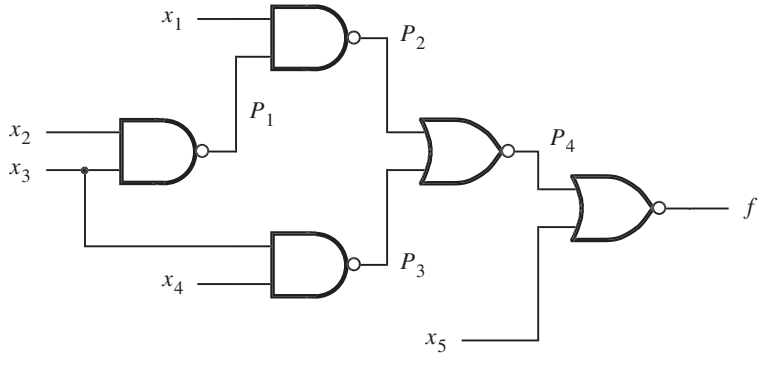


Figura 4.32 Circuito para el ejemplo 4.13.

$$
\begin{aligned}
&= (P_2 + P_3)\overline{x}_5 \\
&= (\overline{x}_1 + \overline{P}_1 + \overline{x}_3 + \overline{x}_4)\overline{x}_5 \\
&= (\overline{x}_1 + x_2x_3 + \overline{x}_3 + \overline{x}_4)\overline{x}_5 \\
&= (\overline{x}_1 + x_2 + \overline{x}_3 + \overline{x}_4)\overline{x}_5 \\
&= \overline{x}_1\overline{x}_5 + x_2\overline{x}_5 + \overline{x}_3\overline{x}_5 + \overline{x}_4\overline{x}_5
\end{aligned}
$$

Note que al derivar de la segunda a la última línea se aplicó la propiedad 16*a* de la sección 2.5 para simplificar $x_2x_3 + \overline{x}_3$ en $x_2 + \overline{x}_3$.

El análisis de circuitos es mucho más simple que la síntesis. Con un poco de práctica es posible desarrollar la habilidad de analizar fácilmente incluso circuitos muy complejos.

Hasta ahora hemos cubierto una cantidad considerable de material respecto a la síntesis y el análisis de funciones lógicas. Hemos usado el mapa de Karnaugh como vehículo para ilustrar los conceptos implícitos en la búsqueda de las implementaciones óptimas de las funciones lógicas. También demostramos que las funciones lógicas pueden realizarse en varias formas, tanto con dos niveles de lógica como con niveles múltiples. En un entorno de diseño moderno, los circuitos lógicos se sintetizan con las herramientas CAD, no a mano. Los conceptos expuestos en el capítulo son muy generales; son representativos de las estrategias aplicadas en los algoritmos CAD. Como ya dijimos, no resulta apropiado emplear el mapa de Karnaugh para representar funciones lógicas en herramientas CAD. En la sección siguiente estudiaremos una representación alternativa de las funciones lógicas, adecuada para usarse en algoritmos CAD.

4.8 REPRESENTACIÓN CÚBICA

El mapa de Karnaugh es un excelente vehículo para ilustrar conceptos, e incluso es útil para diseño manual si las funciones sólo tienen unas cuantas variables. Para manejar funciones más grandes es necesario contar con técnicas algebraicas en vez de gráficas, las cuales pueden aplicarse a funciones de cualquier número de variables.

Se han desarrollado numerosas técnicas de optimación algebraica. No las explicaremos con gran detalle, sino que intentaremos ofrecer al lector una apreciación de las tareas involucradas. Esto ayuda a comprender mejor lo que las herramientas CAD pueden hacer y qué resultados cabe esperar de ellas. Los enfoques que expondremos utilizan una representación cúbica de las funciones lógicas.

4.8.1 CUBOS E HIPERCUBOS

Hasta el momento hemos visto cuatro formas de representar las funciones lógicas: tablas de verdad, expresiones algebraicas, diagramas de Venn y mapas de Karnaugh. Otra posibilidad es mapear una función de *n* variables en un cubo de *n* dimensiones.

Cubo bidimensional

En la figura 4.33 se muestra un cubo bidimensional. Las cuatro esquinas del cubo se llaman *vértices* y corresponden a las cuatro filas de una tabla de verdad. Cada vértice se identifica mediante dos coordenadas. Se supone que la coordenada horizontal corresponde a la variable x_1 y la vertical a x_2. Por tanto, el vértice 00 es la esquina inferior izquierda, que corresponde a la fila 0 en la tabla de verdad. El vértice 01 es la esquina superior izquierda, donde $x_1 = 0$ y $x_2 = 1$, que corresponde a la fila 1 en la tabla de verdad, y así por el estilo para los otros dos vértices.

Una función se mapeará en el cubo indicando con círculos gris los vértices para los que $f = 1$. En la figura 4.33, $f = 1$ para los vértices 01, 10 y 11. La función puede expresarse como un conjunto de vértices mediante la notación $f = \{01, 10, 11\}$. En la figura, la función f también se muestra en la forma de la tabla de verdad.

Un borde une dos vértices cuyas etiquetas sólo difieren en el valor de una variable. Por tanto, si dos vértices para los que $f = 1$ se unen mediante un borde, entonces éste representa la parte de la función igual que los dos vértices individuales. Por ejemplo, $f = 1$ para los vértices 10 y 11, que se unen mediante el borde etiquetado con 1x. Es costumbre utilizar la letra x para denotar el hecho de que la variable correspondiente puede ser 0 o 1. Por consiguiente, 1x significa que $x_1 = 1$, mientras que x_2 puede ser 0 o 1. De manera similar, los vértices 01 y 11 se unen mediante el borde etiquetado con x1, lo que indica que x_1 puede ser 0 o 1, pero $x_2 = 1$. El lector no debe confundir el empleo de la letra x para este propósito, en contraste con el uso donde x_1 y x_2 se refieren a las variables.

Dos vértices que se representan mediante un solo borde es la encarnación de la propiedad combinatoria 14*a* expuesta en la sección 2.5. El borde 1x es la suma lógica de los vértices 10 y 11. En esencia define el término x_1, que es la suma de los mintérminos $x_1\overline{x}_2$ y x_1x_2. La propiedad 14*a* indica que

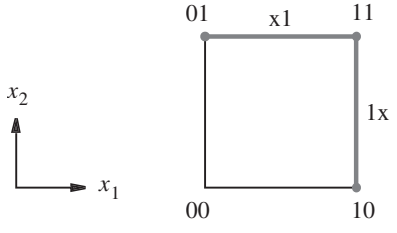
$$x_1\overline{x}_2 + x_1x_2 = x_1$$

En consecuencia, encontrar los bordes para los que $f = 1$ equivale a aplicar la propiedad combinatoria. Desde luego, esto también es semejante a hallar pares de celdas adyacentes en un mapa de Karnaugh para los que $f = 1$.

Los bordes 1x y x1 definen por completo la función de la figura 4.33; por tanto, la función puede representarse como $f = \{1x, x1\}$. Esto corresponde a la expresión lógica

$$f = x_1 + x_2$$

que también es obvia a partir de la tabla de verdad de la figura.



x_1	x_2	f
0	0	0
0	1	1
1	0	1
1	1	1

Figura 4.33 Representación de $f(x_1, x_2) = \sum m(1, 2, 3)$.

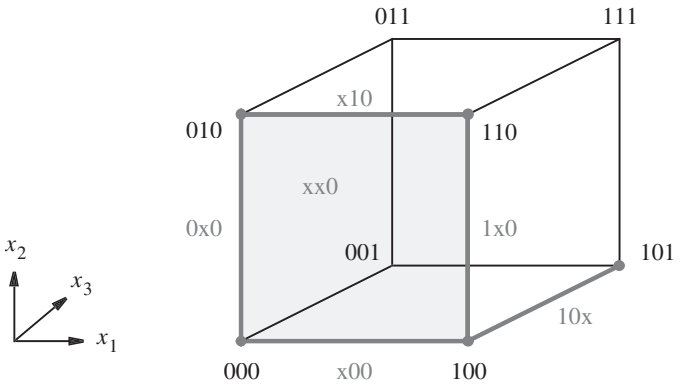


Figura 4.34 Representación de $f(x_1, x_2, x_3) = \sum m(0, 2, 4, 5, 6)$.

Cubo tridimensional

En la figura 4.34 se ilustra un cubo tridimensional. Las coordenadas x_1, x_2 y x_3 son como se muestra a la izquierda. Cada vértice se identifica mediante una combinación específica de las tres variables. La función *f* mapeada en el cubo es la de la figura 4.1, que se usó en la figura 4.5*b*. Hay cinco vértices para los que $f = 1$: 000, 010, 100, 101 y 110. Estos vértices se unen mediante los cinco bordes que se muestran en gris: x00, 0x0, x10, 1x0 y 10x. Puesto que los vértices 000, 010, 100 y 110 incluyen todas las combinaciones de x_1 y x_2 cuando x_3 es 0, pueden especificarse mediante el término xx0. Este término significa que $f = 1$ si $x_3 = 0$, independientemente de los valores de x_1 y x_2. Nótese que xx0 representa el lado frontal del cubo, que está sombreado en gris.

Con base en la explicación anterior, es evidente que la función *f* puede representarse de varias formas, algunas de las cuales son

$$\begin{aligned} f &= \{000, 010, 100, 101, 110\} \\ &= \{0x0, 1x0, 101\} \\ &= \{x00, x10, 101\} \\ &= \{x00, x10, 10x\} \\ &= \{xx0, 10x\} \end{aligned}$$

En una realización física cada uno de los términos anteriores es un término producto implementado mediante una compuerta AND. Obviamente, el circuito menos costoso se obtiene si $f = \{xx0, 10x\}$, que equivale a la expresión lógica

$$f = \overline{x}_3 + x_1\overline{x}_2$$

Ésta es la expresión que derivamos usando el mapa de Karnaugh de la figura 4.5*b*.

Cubo de cuatro dimensiones

Las imágenes gráficas de los cubos bidimensionales y tridimensionales son fáciles de trazar. Un cubo de cuatro dimensiones es más difícil. Consta de dos cubos tridimensionales con

sus esquinas conectadas. La forma más simple de visualizar un cubo de cuatro dimensiones es colocando un cubo dentro de otro, como se muestra en la figura 4.35. Hemos supuesto que las coordenadas x_1, x_2 y x_3 son las mismas que en la figura 4.34, mientras que $x_4 = 0$ define el cubo exterior y $x_4 = 1$ el interior. La figura 4.35 indica cómo se mapea la función f_3 de la figura 4.7 en el cubo de cuatro dimensiones. Para no abarrotar la figura con demasiadas etiquetas, sólo hemos etiquetado los vértices para los que $f_3 = 1$. De nuevo se resaltan en gris todos los bordes que conectan dichos vértices.

Hay dos grupos de cuatro vértices adyacentes para los que $f_3 = 1$, los cuales pueden representarse como planos. El grupo que comprende 0000, 0010, 1000 y 1010 se representa mediante x0x0. El grupo 0010, 0011, 0110 y 0111 se representa con 0x1x. En la figura se sombrean esos planos. La función f_3 puede representarse de varias formas, por ejemplo

$$\begin{aligned} f_3 &= \{0000, 0010, 0011, 0110, 0111, 1000, 1010, 1111\} \\ &= \{00x0, 10x0, 0x10, 0x11, x111\} \\ &= \{x0x0, 0x1x, x111\} \end{aligned}$$

Puesto que cada x indica que es posible ignorar la variable correspondiente, ya que puede ser 0 o 1, el circuito más simple se obtiene si $f = \{x0x0, 0x1x, x111\}$, que equivale a

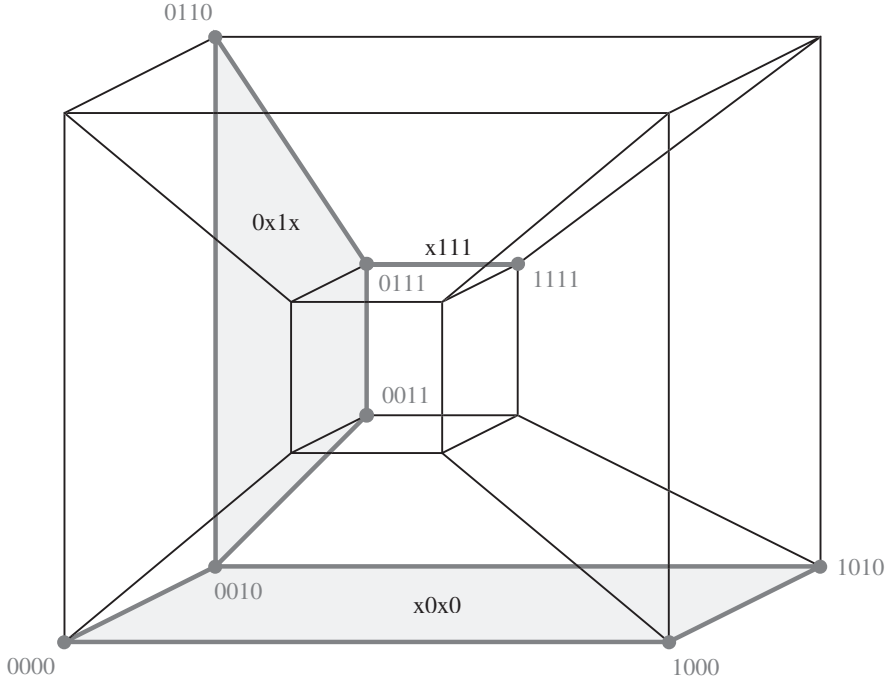


Figura 4.35 Representación de la función f_3 de la figura 4.7.

la expresión

$$f_3 = \overline{x}_2\overline{x}_4 + \overline{x}_1x_3 + x_2x_3x_4$$

La misma expresión se derivó en la figura 4.7.

Cubo de *n* dimensiones

Una función con *n* variables puede mapearse en un cubo de *n* dimensiones. Aunque resulta impráctico trazar imágenes de cubos que tengan más de cuatro variables, no es difícil extender las ideas presentadas líneas arriba a un caso general de *n* variables. Puesto que la interpretación visual no es posible y que normalmente la palabra *cubo* se usa sólo para una estructura tridimensional, muchas personas emplean la palabra *hipercubo* para referirse a estructuras con más de tres dimensiones. En este texto seguiremos empleando la palabra *cubo*.

Es conveniente referirse a un cubo cuyo *tamaño* refleje su número de vértices. Los vértices tienen el tamaño más pequeño. Cada variable tiene un valor de 0 o 1 en un vértice. Un cubo que tenga una x en la posición de una variable es más grande porque consta de dos vértices. Por ejemplo, el cubo 1x01 consta de los vértices 1001 y 1101. Un cubo que tenga dos x consta de cuatro vértices, y así sucesivamente. Un cubo que tenga *k* x consta de 2^k vértices.

Un cubo de *n* dimensiones tiene 2^n vértices. Dos vértices son adyacentes si sólo difieren en el valor de una coordenada. Puesto que hay *n* coordenadas (ejes en el cubo de *n* dimensiones), cada vértice es adyacente a otros *n* vértices. El cubo de *n* dimensiones contiene cubos de menor dimensionalidad, los cuales son vértices. Puesto que su dimensión es cero, podemos llamarlos *cubos 0*. Los bordes son cubos de dimensión 1; por tanto, los denominaremos *cubos 1*. Un lado de un cubo tridimensional es un *cubo 2*. Un cubo tridimensional entero es un *cubo 3*, etc. En general, nos referiremos a un conjunto de 2^k vértices adyacentes como un *cubo k*.

A partir de los ejemplos en las figuras 4.34 y 4.35 es claro que los *cubos k* más grandes que existen para una función son equivalentes a sus implicantes primos. A continuación veremos las técnicas de minimización que usan la representación cúbica de funciones.

4.9 UN MÉTODO TABULAR PARA MINIMIZACIÓN

La representación cúbica de las funciones lógicas es adecuada para la implementación de algoritmos de minimización que pueden programarse y ejecutarse bien en las computadoras. Las modernas herramientas CAD incluyen tales algoritmos. Si bien éstas pueden usarse eficazmente sin el conocimiento detallado de cómo se implementan sus algoritmos de minimización, tal vez parezca interesante para el lector obtener ciertos conocimientos de cómo se logra esto. En la presente sección describiremos un método tabular hasta cierto punto simple que ilustra los conceptos principales e indica algunos de los problemas que surgen.

En la década de 1950, Willard Quine [6] y Edward McCluskey [7] propusieron un enfoque tabular para la minimización. Se popularizó con el nombre de *método Quine-McCluskey*. Aunque no es lo suficientemente eficaz como para usarse en las herramientas CAD modernas, es simple e ilustra los aspectos clave. Lo presentaremos por medio de la notación cúbica expuesta en la sección 4.8.

4.9.1 GENERACIÓN DE IMPLICANTES PRIMOS

Como dijimos en la sección 4.8, los implicantes primos de una función lógica f son los cubos k más grandes posibles para los que $f = 1$. Para funciones especificadas de manera incompleta, que incluyen un conjunto de vértices no-importa, los implicantes primos son los cubos k más grandes para los que $f = 1$ o f no se especifica.

Supóngase que la especificación inicial de f está dada como mintérminos para los que $f = 1$. Además, especifiquemos los no-importa como mintérminos. Esto nos permite crear una lista de vértices para los que $f = 1$ o es una condición no-importa. Comparamos luego estos vértices por parejas para ver si pueden combinarse en cubos más grandes. Entonces podemos intentar combinar los cubos nuevos en otros todavía más grandes y continuar el proceso hasta hallar los implicantes primos.

La base del método es la propiedad combinatoria del álgebra booleana

$$x_i x_j + x_i \overline{x}_j = x_i$$

que usamos en la sección 4.8 para desarrollar la representación cúbica. Si se tienen dos cubos idénticos en todas las variables (coordenadas) excepto en una para la que un cubo tiene el valor 0 y el otro tiene 1, entonces dichos cubos pueden combinarse en un cubo más grande. Por ejemplo, considérese $f(x_1, \ldots, x_4) = \{1000, 1001, 1010, 1011\}$. Los cubos 1000 y 1001 sólo difieren en la variable x_4; pueden combinarse en un cubo nuevo 100x. De manera similar, 1010 y 1011 pueden combinarse en 101x. Luego 100x y 101x pueden combinarse en un cubo más grande, 10xx, que significa que es posible expresar la función simplemente como $f = x_1 \overline{x}_2$.

En la figura 4.36 se muestra cómo generar los implicantes primos para la función, f, de la figura 4.11. La función se define como

$$f(x_1, \ldots, x_4) = \sum m(0, 4, 8, 10, 11, 12, 13, 15)$$

No hay condiciones no-importa. Puesto que los cubos más grandes sólo pueden generarse a partir de mintérminos que difieren sólo en una variable, es posible reducir el número de comparaciones por pares colocando los mintérminos en grupos tales que los cubos de cada uno de ellos tengan

Lista 1

0	0 0 0 0	✓
4	0 1 0 0	✓
8	1 0 0 0	✓
10	1 0 1 0	✓
12	1 1 0 0	✓
11	1 0 1 1	✓
13	1 1 0 1	✓
15	1 1 1 1	✓

Lista 2

0,4	0 x 0 0	✓
0,8	x 0 0 0	✓
8,10	1 0 x 0	
4,12	x 1 0 0	✓
8,12	1 x 0 0	✓
10,11	1 0 1 x	
12,13	1 1 0 x	
11,15	1 x 1 1	
13,15	1 1 x 1	

Lista 3

0,4,8,12	x x 0 0

Figura 4.36 Generación de implicantes primos para la función de la figura 4.11.

el mismo número de 1, y ordenando los grupos por el número de unos. Por tanto, será preciso comparar cada cubo de un grupo con todos los cubos del grupo inmediato anterior. En la figura 4.36 los mintérminos se ordenan de esta forma en la lista 1. (Nótese que también se indicaron los equivalentes decimales de los mintérminos para facilitar la explicación.) Los mintérminos, que también se llaman *cubos 0* como explicamos en la sección 4.8, pueden combinarse en los cubos 1 que se muestran en la lista 2. Para hacer las entradas fácilmente comprensibles indicamos los mintérminos que se combinan para formar cada cubo 1. A continuación se revisa si los cubos 0 se incluyen en los cubos 1 y se escribe una marca junto a cada cubo incluido. Ahora se generan cubos 2 a partir de los cubos 1 de la lista 2. El único cubo 2 que se genera es xx00, que se coloca en la lista 3. De nuevo, las marcas se colocan contra los cubos 1 que se incluyen en el cubo 2. Puesto que existe sólo un cubo 2, no puede haber cubos 3 para esta función. Los cubos sin marca en cada lista son los implicantes primos de *f*. Por tanto, el conjunto *P* de implicantes primos es

$$\begin{aligned} P &= \{10x0,\ 101x,\ 110x,\ 1x11,\ 11x1,\ xx00\} \\ &= \{p_1, p_2, p_3, p_4, p_5, p_6\} \end{aligned}$$

4.9.2 DETERMINACIÓN DE UNA COBERTURA MÍNIMA

Tras generar el conjunto de todos los implicantes primos es preciso elegir un subconjunto de costo mínimo que abarque todos los mintérminos para los que $f = 1$. Como simple medida, supondremos que el costo es directamente proporcional al número de entradas a todas las compuertas, lo que significa al número de literales en los implicantes primos elegidos para implementar la función.

Para determinar una cobertura de costo mínimo se construye una *tabla de cobertura de implicantes primos* en la que haya una fila por cada implicante primo y una columna por cada mintérmino que deba cubrirse. Luego se colocan marcas para indicar los mintérminos cubiertos por cada implicante primo. En la figura 4.37*a* se muestra la tabla para los implicantes primos deducidos en la figura 4.36. Si hay una sola marca en alguna columna de la tabla de cobertura, entonces el implicante primo que cubre el mintérmino de esa columna es *esencial* y ha de incluirse en la cobertura final. Tal es el caso de p_6, que es el único implicante primo que cubre los mintérminos 0 y 4. El paso siguiente es eliminar la(s) fila(s) correspondiente(s) a los implicantes primos esenciales y la(s) columna(s) cubierta(s) por ellos. Por consiguiente, se eliminan p_6 y las columnas 0, 4, 8 y 12, lo que nos lleva a la tabla de la figura 4.37*b*.

Ahora podemos aplicar el concepto de *dominancia de fila* para reducir la tabla de cobertura. Obsérvese que p_1 sólo cubre el mintérmino 10, mientras que p_2 cubre tanto 10 como 11. Se dice que p_2 *domina* a p_1. Como el costo de p_2 es el mismo que el de p_1, es prudente elegir p_2 en vez de p_1, de modo que se eliminará p_1 de la tabla. De manera similar, p_5 domina a p_3; en consecuencia, se eliminará ésta de la tabla. Por tanto, se obtiene la tabla de la figura 4.37*c*, la cual indica que hay que elegir p_2 para cubrir el mintérmino 10 y p_5 para cubrir el mintérmino 13, que también se ocupa de cubrir los mintérminos 11 y 15. De este modo, la cobertura final es

$$\begin{aligned} C &= \{p_2, p_5, p_6\} \\ &= \{101x,\ 11x1,\ xx00\} \end{aligned}$$

Implicante primo	Mintérmino 0	4	8	10	11	12	13	15
$p_1 = 1\ 0\ x\ 0$			✓	✓				
$p_2 = 1\ 0\ 1\ x$				✓	✓			
$p_3 = 1\ 1\ 0\ x$						✓	✓	
$p_4 = 1\ x\ 1\ 1$					✓			✓
$p_5 = 1\ 1\ x\ 1$							✓	✓
$p_6 = x\ x\ 0\ 0$	✓	✓	✓			✓		

a) Tabla inicial de cobertura de implicantes primos

Implicante primo	Mintérmino 10	11	13	15
p_1	✓			
p_2	✓	✓		
p_3			✓	
p_4		✓		✓
p_5			✓	✓

b) Después de eliminar los implicantes primos esenciales

Implicante primo	Mintérmino 10	11	13	15
p_2	✓	✓		
p_4		✓		✓
p_5			✓	✓

c) Después de eliminar las filas dominantes

Figura 4.37 Selección de una cobertura para la función de la figura 4.11.

lo que significa que la implementación de costo mínimo de la función es

$$f = x_1\overline{x}_2x_3 + x_1x_2x_4 + \overline{x}_3\overline{x}_4$$

Ésta es la misma expresión que la derivada en la sección 4.2.2.

En este ejemplo aplicamos el concepto de dominancia de fila para reducir la tabla de cobertura. Las filas dominadas se eliminaron porque cubren menos mintérminos y el costo de sus

implicantes primos es el mismo que el de los implicantes primos de las filas dominantes. Sin embargo, no debe eliminarse una fila dominada si el costo de su implicante primo es menor que el del implicante primo de la fila que domina. En el problema 4.25 se presenta un ejemplo de esta situación.

En el ejemplo siguiente se ilustra el uso del método tabular con condiciones no-importa.

Ejemplo 4.14

Los mintérminos no-importa se incluyen en la lista inicial de la misma forma que los mintérminos para los que $f = 1$. Considere la función

$$f(x_1, \ldots, x_4) = \sum m(0, 2, 5, 6, 7, 8, 9, 13) + D(1, 12, 15)$$

Se alienta al lector a derivar un mapa de Karnaugh para esta función como un auxiliar para visualizar la derivación que sigue. En la figura 4.38 se muestra la generación de implicantes primos, lo que produce el resultado

$$\begin{aligned} P &= \{00x0, 0x10, 011x, x00x, xx01, 1x0x, x1x1\} \\ &= \{p_1, p_2, p_3, p_4, p_5, p_6, p_7\} \end{aligned}$$

La tabla inicial de cobertura de implicantes primos se muestra en la figura 4.39*a*. Los mintérminos no-importa no se incluyen en la tabla porque no tienen que cubrirse. No hay implicantes primos esenciales. Al examinar esta tabla se advierte que la columna 8 tiene marcas en las mismas filas que la columna 9. Más aún, la columna 9 tiene una marca más en la fila p_5. Por tanto, la columna 9 domina a la 8. Esto se designa con el concepto de *dominancia de columna*. Cuando una columna domina a otra puede eliminarse la columna dominante, que en este caso es la

Lista 1

0	0 0 0 0	✓
1	0 0 0 1	✓
2	0 0 1 0	✓
8	1 0 0 0	✓
5	0 1 0 1	✓
6	0 1 1 0	✓
9	1 0 0 1	✓
12	1 1 0 0	✓
7	0 1 1 1	✓
13	1 1 0 1	✓
15	1 1 1 1	✓

Lista 2

0,1	0 0 0 x	✓
0,2	0 0 x 0	
0,8	x 0 0 0	✓
1,5	0 x 0 1	✓
2,6	0 x 1 0	
1,9	x 0 0 1	✓
8,9	1 0 0 x	✓
8,12	1 x 0 0	✓
5,7	0 1 x 1	✓
6,7	0 1 1 x	
5,13	x 1 0 1	✓
9,13	1 x 0 1	✓
12,13	1 1 0 x	✓
7,15	x 1 1 1	✓
13,15	1 1 x 1	✓

Lista 3

0,1,8,9	x 0 0 x
1,5,9,13	x x 0 1
8,9,12,13	1 x 0 x
5,7,13,15	x 1 x 1

Figura 4.38 Generación de implicantes primos para la función del ejemplo 4.14.

Implicante primo	Mintérmino 0	2	5	6	7	8	9	13
$p_1 = 0\ 0\ x\ 0$	✓	✓						
$p_2 = 0\ x\ 1\ 0$		✓		✓				
$p_3 = 0\ 1\ 1\ x$				✓	✓			
$p_4 = x\ 0\ 0\ x$	✓					✓	✓	
$p_5 = x\ x\ 0\ 1$			✓				✓	✓
$p_6 = 1\ x\ 0\ x$						✓	✓	✓
$p_7 = x\ 1\ x\ 1$			✓		✓			✓

a) Tabla inicial de cobertura de implicantes primos

Implicante primo	Mintérmino 0	2	5	6	7	8
$p_1 = 0\ 0\ x\ 0$	✓	✓				
$p_2 = 0\ x\ 1\ 0$		✓		✓		
$p_3 = 0\ 1\ 1\ x$				✓	✓	
$p_4 = x\ 0\ 0\ x$	✓					✓
$p_5 = x\ x\ 0\ 1$			✓			
$p_6 = 1\ x\ 0\ x$						✓
$p_7 = x\ 1\ x\ 1$			✓		✓	

b) Después de eliminar las columnas 9 y 13

Implicante primo	Mintérmino 0	2	5	6	7	8
p_1	✓	✓				
p_2		✓		✓		
p_3				✓	✓	
p_4	✓					✓
p_7			✓		✓	

c) Después de eliminar las filas p_5 y p_6

Implicante primo	Mintérmino 2	6
p_1	✓	
p_2	✓	✓
p_3		✓

d) Después de incluir p_4 y p_7 en la cobertura

Figura 4.39 Selección de una cobertura para la función del ejemplo 4.14.

columna 9. Note que esto contrasta con las filas donde se eliminan las filas dominadas (en lugar de las dominantes). La razón es que cuando se elige un implicante primo para cubrir el mintérmino que corresponde a la columna dominada, este implicante primo también cubrirá el mintérmino correspondiente a la columna dominante. En el ejemplo, elegir p_4 o p_6 cubre los mintérminos 8 y 9. De manera similar, la columna 13 domina a la 5 y, por tanto, la 13 puede borrarse.

Después de eliminar las columnas 9 y 13 se obtiene la tabla reducida mostrada en la figura 4.39*b*. En ella la fila p_4 domina a la p_6 y la p_7 a la p_5. Esto significa que p_5 y p_6 pueden eliminarse, lo que resulta en la tabla de la figura 4.39*c*. Ahora, p_4 y p_7 son esenciales para cubrir los mintérminos 8 y 5, respectivamente. Por ende, se obtiene la tabla de la figura 4.39*d*, a partir de la cual es obvio que p_2 cubre los restantes mintérminos 2 y 6. Note que la fila p_2 domina las filas p_1 y p_3.

La cobertura final es

$$\begin{aligned} C &= \{p_2, p_4, p_7\} \\ &= \{0x10,\ x00x,\ x1x1\} \end{aligned}$$

y la función se implementa como

$$f = \overline{x}_1 x_3 \overline{x}_4 + \overline{x}_2 \overline{x}_3 + x_2 x_4$$

En las figuras 4.37 y 4.39 aplicamos el concepto de dominancia de fila y de columna para reducir la tabla de cobertura, pero esto no siempre es posible, como se ilustra en el ejemplo que sigue.

Ejemplo 4.15

Considere la función

$$f(x_1, \ldots, x_4) = \sum m(0, 3, 10, 15) + D(1, 2, 7, 8, 11, 14)$$

Los implicantes primos para esta función son

$$\begin{aligned} P &= \{00xx,\ x0x0,\ x01x,\ xx11,\ 1x1x\} \\ &= \{p_1, p_2, p_3, p_4, p_5\} \end{aligned}$$

La tabla inicial de cobertura de implicantes primos se muestra en la figura 4.40*a*. No hay implicantes primos esenciales. Además, no existen filas o columnas dominantes. Más aún, todos los implicantes primos tienen el mismo costo porque cada uno de ellos se implementa con dos literales. Por tanto, la tabla no ofrece pista alguna que sirva para seleccionar una cobertura de costo mínimo.

Un buen enfoque práctico consiste en usar el concepto de *ramificación*, que se expuso en la sección 4.2.2. Se puede escoger cualquier implicante primo, digamos p_3, y primero elegir incluirlo en la cobertura final. Luego el resto de la cobertura puede determinarse en la forma usual y calcular su costo. A continuación se intenta otra posibilidad excluyendo p_3 de la cobertura final y determinando el costo resultante. Se comparan los costos y se elige la opción menos costosa.

En la figura 4.40*b* se proporciona la tabla de cobertura que queda si p_3 se incluye en la cobertura final. La tabla no incluye los mintérminos 3 y 10 porque están cubiertos por p_3. Además,

Implicante primo	Mintérmino 0	3	10	15
$p_1 = 0\ 0\ x\ x$	✓	✓		
$p_2 = x\ 0\ x\ 0$	✓		✓	
$p_3 = x\ 0\ 1\ x$		✓	✓	
$p_4 = x\ x\ 1\ 1$		✓		✓
$p_5 = 1\ x\ 1\ x$			✓	✓

a) Tabla inicial de cobertura de implicantes primos

Implicante primo	Mintérmino 0	15
p_1	✓	
p_2	✓	
p_4		✓
p_5		✓

b) Después de incluir p_2 en la cobertura

Implicante primo	Mintérmino 0	3	10	15
p_1	✓	✓		
p_2	✓		✓	
p_4		✓		✓
p_5			✓	✓

c) Después de excluir p_2 de la cobertura

Figura 4.40 Selección de una cobertura para la función del ejemplo 4.15.

indica que una cobertura completa debe incluir p_1 o p_2 para cubrir el mintérmino 0, y p_4 o p_5 para cubrir el mintérmino 15. Por tanto, una cobertura completa puede ser

$$C = \{p_1, p_3, p_4\}$$

La alternativa de excluir p_3 conduce a la tabla de cobertura de la figura 4.40*c*. Ahí se observa que una cobertura de costo mínimo sólo requiere dos implicantes primos. Una posibilidad es elegir

p_1 y p_5; la otra es elegir p_2 y p_4. En consecuencia, una cobertura de costo mínimo es justo

$$\begin{aligned} C_{mín} &= \{p_1, p_5\} \\ &= \{00xx,\ 1x1x\} \end{aligned}$$

La función se realiza como

$$f = \overline{x}_1\overline{x}_2 + x_1x_3$$

4.9.3 RESUMEN DEL MÉTODO TABULAR

El método tabular se resume del modo siguiente:

1. A partir de una lista de cubos que representen los mintérminos donde $f = 1$ o una condición no-importa, se generan los implicantes primos mediante comparaciones por pares sucesivos de los cubos.
2. Se deriva una tabla de cobertura que indique los mintérminos donde $f = 1$ que están cubiertos por cada implicante primo.
3. Se incluyen los implicantes primos esenciales (si hay alguno) en la cobertura final y la tabla se reduce mediante la eliminación tanto de dichos implicantes primos como de los mintérminos cubiertos.
4. Se aplica el concepto de dominancia de fila y de columna para reducir aún más la tabla de cobertura. Una fila dominada se elimina sólo si el costo de su implicante primo es mayor o igual que el costo del implicante primo de la fila dominante.
5. Se repiten los pasos 3 y 4 hasta que la tabla de cobertura esté vacía o ya no sea posible reducirla más.
6. Si la tabla de cobertura reducida no está vacía, entonces se usa el enfoque de ramificación para determinar los implicantes primos restantes que han de incluirse en una cobertura de costo mínimo.

El método tabular ilustra cómo se utiliza una técnica algebraica para generar los implicantes primos. También muestra un enfoque simple para encarar el problema de la cobertura: encontrar una cobertura de costo mínimo. El método tiene ciertas limitaciones prácticas. Por citar una, las funciones rara vez se definen en la forma de mintérminos. En general se proporcionan en la forma de expresiones algebraicas o como conjuntos de cubos. La necesidad de comenzar el proceso de minimización con una lista de mintérminos implica que las expresiones o conjuntos deben expandirse de esta forma. Esta lista puede ser muy larga. Conforme se generan cubos más grandes, habrá numerosas comparaciones que hacer y los cálculos serán lentos. El uso de la tabla de cobertura para seleccionar el conjunto óptimo de implicantes primos también es computacionalmente intenso cuando se trata con funciones grandes.

Se han desarrollado muchas técnicas algebraicas cuya intención es reducir el tiempo que demoran en generarse las coberturas óptimas. Aunque el grueso de tales técnicas está más allá del ámbito de este libro, en la sección siguiente abordaremos brevemente un posible enfoque. Un lector cuya intención sea usar las herramientas CAD pero que no esté interesado en los detalles de la minimización automatizada, puede obviar esa sección sin perder la continuidad.

4.10 UNA TÉCNICA CÚBICA DE MINIMIZACIÓN

Supóngase que la especificación inicial de una función *f* está dada en términos de implicantes que no necesariamente son mintérminos o implicantes primos. Entonces es conveniente definir una operación que generará otros implicantes que no están dados de modo explícito en la especificación inicial, pero que a la postre conducirán a los implicantes primos de *f*. Una de tales posibilidades se conoce como la operación *producto* $*$, que se pronuncia como operación "producto estrella". En el texto simplemente nos referiremos a ella como *operación* $*$.

Operación $*$

La operación $*$ brinda una forma simple de derivar un cubo nuevo combinando dos cubos que sólo difieren en el valor de una variable. Sean $A = A_1A_2 \cdots A_n$ y $B = B_1B_2 \cdots B_n$ dos cubos que son implicantes de una función de n variables. Por tanto, cada coordenada A_i y B_i se especifica como poseedora del valor 0, 1 o x. La operación $*$ tiene dos pasos. Primero, la operación $*$ se evalúa para cada par A_i y B_i, en coordenadas $i = 1, 2,..., n$, de acuerdo con la tabla de la figura 4.41. Entonces, con base en los resultados obtenidos tras usar la tabla, se aplica un conjunto de reglas para determinar el resultado global de la operación $*$. En la tabla de la figura 4.41 se define la operación $*$ coordenada, $A_i * B_i$, la cual especifica el resultado de $A_i * B_i$ para cada posible combinación de valores de A_i y B_i. Este resultado es la intersección (es decir, la parte común) de A y B en esta coordenada. Nótese que cuando A_i y B_i tienen los valores opuestos (0 y 1, o viceversa), el resultado de la operación $*$ coordenada se indica mediante el símbolo ø. Se dice que la intersección de A_i y B_i está vacía. Al usar la tabla, la operación $*$ completa para A y B se define como sigue:

$$C = A * B, \text{ tal que}$$

1. $C =$ ø y $A_i * B_i =$ ø para más de una i.
2. De otro modo, $C_i = A_i * B_i$ cuando $A_i * B_i \neq$ ø, y $C_i =$ x para la coordenada donde $A_i * B_i =$ ø.

Por ejemplo, sea $A = \{0x0\}$ y $B = \{111\}$. Entonces $A_1 * B_1 = 0 * 1 =$ ø , $A_2 * B_2 =$ x $* 1 =$ 1 y $A_3 * B_3 = 0 * 1 =$ ø. Puesto que el resultado es ø en dos coordenadas, de la condición 1 se sigue que $A * B =$ ø. En otras palabras, estos dos cubos no pueden combinarse en otro, porque difieren en dos coordenadas.

Como otro ejemplo, considérese $A = \{11x\}$ y $B = \{10x\}$. En este caso, $A_1 * B_1 = 1 * 1 = 1$, $A_2 * B_2 = 1 * 0 =$ ø y $A_3 * B_3 =$ x $*$ x $=$ x. De acuerdo con la condición 2 anterior, $C_1 = 1$,

A_i \ B_i	0	1	x
0	0	ø	0
1	ø	1	1
x	0	1	x

$A_i * B_i$

Figura 4.41 Operación $*$ coordenada.

C_2 = x y C_3 = x, lo que produce $C = A * B = \{1xx\}$. A partir de dos cubos 1 que difieren sólo en una coordenada se crea un cubo 2 más grande.

El resultado de la operación $*$ puede ser un cubo más pequeño que los dos cubos implicados en la operación. Considérese $A = \{1x1\}$ y $B = \{11x\}$. Entonces $C = A * B = \{111\}$. Cabe señalar que C se incluye tanto en A como en B, lo que significa que este cubo no será útil en la búsqueda de implicantes primos. Por tanto, debe descartarse mediante el algoritmo de minimización.

Como ejemplo final, considérese $A = \{x10\}$ y $B = \{0x1\}$. Entonces $C = A * B = \{01x\}$. Los tres cubos tienen el mismo tamaño, pero C no se incluye en A ni en B. Por ende, debe considerarse en la búsqueda de implicantes primos. El lector puede encontrar útil trazar un mapa de Karnaugh para ver la manera en que el cubo C se relaciona con los cubos A y B.

Uso de la operación $*$ para hallar implicantes primos

La esencia de la operación $*$ es encontrar cubos nuevos a partir de pares de cubos existentes. En particular, es de interés encontrar los que no estén incluidos en los cubos existentes. Un procedimiento para encontrar los implicantes primos puede organizarse como sigue.

Supóngase que una función f se especifica mediante un conjunto de implicantes que están representados como cubos. Denotemos este conjunto como la cobertura C^k de f. Sean c^i y c^j cualesquiera dos cubos en C^k. Entonces apliquemos la operación $*$ a todos los pares de cubos en C^k; sea G^{k+1} el conjunto de cubos recién generados. En consecuencia

$$G^{k+1} = c^i * c^j \text{ para todo } c^i, c^j \in C^k$$

Ahora se puede formar una cobertura nueva para f empleando los cubos en C^k y G^{k+1}. Algunos de ellos pueden ser redundantes porque están incluidos en otros cubos; deben eliminarse. Sea la nueva cobertura

$$C^{k+1} = C^k \cup G^{k+1} - \text{cubos redundantes}$$

donde $\cup$ denota la unión lógica de dos conjuntos y el signo menos $(-)$ denota la eliminación de elementos de un conjunto. Si $C^{k+1} \neq C^k$, entonces se genera una nueva cobertura C^{k+2} con el mismo proceso. Si $C^{k+1} = C^k$, entonces los cubos en la cobertura son los implicantes primos de f. Para una función de n variables es preciso repetir el paso cuando mucho n veces.

Los cubos redundantes que han de eliminarse se identifican mediante la comparación por pares de cubos. El cubo $A = A_1A_2 \cdots A_n$ debe borrarse si está incluido en algún cubo $B = B_1B_2 \cdots B_n$, que es el caso si $A_i = B_i$ o B_i = x para toda coordenada i.

Ejemplo 4.16

Considere la función $f = (x_1x_2x_3)$ de la figura 4.9. Suponga que f inicialmente se especifica como un conjunto de vértices que corresponden a los mintérminos $m_0m_1m_2m_3$ y m_7. Por tanto, sea $C^0 = \{000, 001, 010, 011, 111\}$ la cobertura inicial. Con la operación $*$ para generar un conjunto nuevo de cubos se obtiene $G^1 = \{00x, 0x0, 0x1, 01x, x11\}$. Entonces $C^1 = C^0 \cup G^1$ – cubos redundantes. Observe que cada cubo en C^0 está incluido en uno de los cubos en G^1; por ende, todos los cubos en C^0 son redundantes. Así, $C^1 = G^1$.

El paso siguiente es aplicar la operación $*$ a los cubos en C^1, lo que produce $G^2 = \{000, 001, 0xx, 0x1, 010, 01x, 011\}$. Note que todos estos cubos están incluidos en el cubo 0xx; por

tanto, todos excepto 0xx son redundantes. Ahora es fácil ver que

$$\begin{aligned} C^2 &= C^1 \cup G^2 - \text{términos redundantes} \\ &= \{\text{x11, 0xx}\} \end{aligned}$$

puesto que todos los cubos de C^1, salvo x11, son redundantes porque están cubiertos por 0xx.

Al aplicar la operación $*$ a C^2 se obtiene $G^3 = \{011\}$ y

$$\begin{aligned} C^3 &= C^2 \cup G^3 - \text{términos redundantes} \\ &= \{\text{x11, 0xx}\} \end{aligned}$$

Como $C^3 = C^2$, la conclusión es que los implicantes primos de f son los cubos {x11, 0xx}, que representan los términos producto x_2x_3 y $\overline{x}_1$. Se trata del mismo conjunto de implicantes primos derivado por medio del mapa de Karnaugh de la figura 4.9.

Observe que la derivación de los implicantes primos en este ejemplo es similar al método tabular explicado en la sección 4.9 porque el punto de partida fue una función, f, dada como un conjunto de mintérminos.

Ejemplo 4.17 Como otro ejemplo, considere la función de cuatro variables de la figura 4.10. Suponga que esta función inicialmente se especifica como la cobertura $C^0 = \{0101, 1101, 1110, 011x, x01x\}$. Entonces las aplicaciones sucesivas de la operación $*$ y la eliminación de los términos redundantes produce

$$\begin{aligned} C^1 &= \{\text{x01x, x101, 01x1, x110, 1x10, 0x1x}\} \\ C^2 &= \{\text{x01x, x101, 01x1, 0x1x, xx10}\} \\ C^3 &= C^2 \end{aligned}$$

Por consiguiente, los implicantes primos son $\overline{x}_2x_3$, $x_2\overline{x}_3x_4$, $\overline{x}_1x_2x_4$, $\overline{x}_1x_3$ y $x_3\overline{x}_4$.

4.10.1 DETERMINACIÓN DE LOS IMPLICANTES PRIMOS ESENCIALES

A partir de una cobertura que conste de todos los implicantes primos es necesario extraer una cobertura mínima. Como hicimos en la sección 4.2.2, todos los implicantes primos *esenciales* deben incluirse en la cobertura mínima. Para encontrarlos es útil definir una operación que determine una parte de un cubo (implicante) que *no* esté cubierta por otro cubo. Una de tales operaciones se llama *operación #* (pronúnciese "operación sharp"), que se define como sigue.

Operación #

De nuevo, sean $A = A_1\,A_2 \cdots A_n$ y $B = B_1\,B_2 \cdots B_n$ dos cubos (implicantes) de una función de n variables. La operación sharp $A\#B$ deja como resultado "la parte de A que no está cubierta por B". Similar a la operación $*$, la operación # tiene dos pasos: $A_i\#B_i$ se evalúa para cada coordenada i y luego se aplica un conjunto de reglas para determinar el resultado global. La opera-

A_i \ B_i	0	1	x
0	ε	ø	ε
1	ø	ε	ε
x	1	0	ε

$A_i \# B_i$

Figura 4.42 Operación # coordenada.

ción sharp para cada coordenada se define en la figura 4.42. Después de realizar esta operación para todos los pares (A_i, B_i), la operación # completa se define como sigue:

$$C = A\#B, \text{ tal que}$$

1. $C = A$ si $A_i\#B_i = \text{ø}$ para algún i.
2. $C = \text{ø}$ si $A_i\#B_i = \varepsilon$ para todo i.
3. De otro modo, $C = \bigcup_i (A_1, A_2, \ldots, \overline{B}_i, \ldots, A_n)$, donde la unión es para todo i para el que $A_i = \text{x}$ y $B_i \neq \text{x}$.

La primera condición corresponde al caso donde los cubos A y B no se intersecan, es decir, A y B difieren en el valor de al menos una variable, lo que significa que ninguna parte de A está cubierta por B. Por ejemplo, sea $A = 0\text{x}1$ y $B = 11\text{x}$. Los productos # coordenados son $A_1\#B_1 = \text{ø}$, $A_2\#B_2 = 0$ y $A_3\#B_3 = \varepsilon$. Entonces, a partir de la regla 1 se sigue que $0\text{x}1 \# 11\text{x} = 0\text{x}1$. La segunda condición refleja el caso donde A está completamente cubierta por B. Por ejemplo, $0\text{x}1 \# 0\text{xx} = \text{ø}$. La tercera condición es para el caso donde sólo una parte de A está cubierta por B. En éste, la operación # genera uno o más cubos. Específicamente, genera un cubo por cada coordenada i que sea x en A_i, pero no lo sea en B_i. Cada cubo generado es idéntico a A, excepto que A_i se sustituye por $\overline{B}_i$. Por ejemplo, $0\text{xx} \# 01\text{x} = 00\text{x}$ y $0\text{xx} \# 010 = \{00\text{x}, 0\text{x}1\}$.

Ahora mostraremos cómo usar la operación # para hallar los implicantes primos esenciales. Sea P el conjunto de todos los implicantes primos de una función dada f. Sea p^i que denota un implicante primo en el conjunto P y DC que denota los vértices no-importa para f. (En esta sección utilizaremos superíndices para referirnos a los diferentes implicantes primos porque hemos empleado subíndices para referirnos a las posiciones coordenadas en los cubos.) Entonces p^i es un implicante primo esencial si y sólo si

$$p^i \# (P - p^i) \# DC \neq \text{ø}$$

Esto significa que p^i es esencial si existe al menos un vértice para el que $f = 1$ que esté cubierto por p^i, pero no por cualquier otro implicante primo. La operación # también se realiza con el conjunto de cubos no-importa porque no es esencial cubrir los vértices en p^i que corresponden a condiciones no-importa. El significado de $p^i \# (P - p^i)$ es que la operación # se aplica sucesivamente a cada implicante primo en P. Por ejemplo, considérese $P = \{p^1, p^2, p^3, p^4\}$ y $DC = \{d^1, d^2\}$. Para comprobar que p^3 es esencial se evalúa

$$((((p^3 \# p^1) \# p^2) \# p^4) \# d^1) \# d^2$$

Si el resultado de esta expresión no es ø, entonces p^3 es esencial.

Ejemplo 4.18 En el ejemplo 4.16 determinamos que los cubos x11 y 0xx son los implicantes primos de la función *f* de la figura 4.9. Podemos descubrir si cada uno de dichos implicantes primos es esencial del modo siguiente

$$x11 \# 0xx = 111 \neq \emptyset$$
$$0xx \# x11 = \{00x, 0x0\} \neq \emptyset$$

El cubo x11 es esencial porque es el único implicante primo que cubre el vértice 111, para el que $f = 1$. El implicante primo 0xx es esencial porque es el único que cubre los vértices 000, 001 y 010. Esto puede observarse en el mapa de Karnaugh de la figura 4.9.

Ejemplo 4.19 En el ejemplo 4.17 encontramos que los implicantes primos de la función de la figura 4.10 son $P = \{x01x, x101, 01x1, 0x1x, xx10\}$. Puesto que esta función tiene no-importa, se calcula

$$x01x \# (P - x01x) = 1011 \neq \emptyset$$

Esto se obtuvo con los pasos siguientes: x01x # x101 = x01x, entonces x01x # 01x1 = x01x, entonces x01x # 0x1x = 101x y finalmente 101x # xx10 = 1011. De manera similar se obtiene

$$x101 \# (P - x101) = 1101 \neq \emptyset$$
$$01x1 \# (P - 01x1) = \emptyset$$
$$0x1x \# (P - 0x1x) = \emptyset$$
$$xx10 \# (P - xx10) = 1110 \neq \emptyset$$

Por tanto, los implicantes primos esenciales son x01x, x101 y xx10, pues son los únicos que cubren los vértices 1011, 1101 y 1110, respectivamente. Esto es obvio a partir del mapa de Karnaugh de la figura 4.10.

Cuando se revisa si un cubo A es esencial, la operación # con uno de los cubos en $P - A$ puede generar varios cubos. Si es así, entonces cada uno de ellos ha de revisarse con la operación # con todos los cubos restantes en $P - A$.

4.10.2 PROCEDIMIENTO COMPLETO PARA HALLAR UNA COBERTURA MÍNIMA

Luego de exponer las operaciones $*$ y #, ahora delinearemos un procedimiento completo para determinar la cobertura mínima para cualquier función de n variables. Supóngase que la función f se especifica en términos de vértices para los que $f = 1$; con frecuencia tales vértices se denominan el *conjunto ON* de la función. Además, supóngase que las condiciones no-importa se especifican como un *conjunto DC*. Entonces la cobertura inicial para f es la unión de los conjuntos ON y DC.

Los implicantes primos de f pueden generarse con la operación $*$, como explicamos en la sección 4.10. Luego puede aplicarse la operación # para hallar los implicantes primos esenciales como describimos en la sección 4.10.1. Si tales implicantes cubren todo el conjunto ON, entonces forman la cobertura de costo mínimo para f. De otro modo, es necesario incluir otros implicantes primos hasta que todos los vértices en el conjunto ON queden cubiertos.

Un implicante primo no esencial p^i debe borrarse si existe un implicante primo menos costoso p^j que cubra todos los vértices del conjunto ON cubiertos por p^i. Si los restantes implicantes primos no esenciales tienen el mismo costo, entonces un posible enfoque heurístico consiste en seleccionar arbitrariamente uno de ellos, incluirlo en la cobertura y determinar el resto de ella. Luego se genera otra cobertura excluyendo este implicante primo y se elige la implementación de la cobertura de costo más bajo. Ya utilizamos este enfoque, que a menudo se conoce como *ramificación heurística*, en las secciones 4.2.2 y 4.9.2.

La explicación anterior puede resumirse en la forma del procedimiento de minimización presentado enseguida:

1. Sea $C^0 = ON \cup DC$ la cobertura inicial de la función f y sus condiciones no-importa.
2. Se encuentran todos los implicantes primos de C^0 con la operación $*$; sea P este conjunto de implicantes primos.
3. Se determinan los implicantes primos esenciales mediante la operación #. Un implicante primo p^i es esencial si $p^i \# (P - p^i) \# DC \neq \emptyset$. Si los implicantes primos esenciales cubren todos los vértices del conjunto ON, entonces forman la cobertura de costo mínimo.
4. Se borra cualquier p^i no esencial que sea más costoso (es decir, un cubo más pequeño) que algún otro implicante primo p^j si $p^i \# DC \# p^j = \emptyset$.
5. Se eligen los implicantes primos de menor costo para cubrir los vértices restantes del conjunto ON. Se aplica la ramificación heurística en los implicantes primos de igual costo y se conserva la cobertura con el costo más bajo.

Ejemplo 4.20

Para ilustrar el procedimiento de minimización emplearemos la función

$$f(x_1, x_2, x_3, x_4, x_5) = \sum m(0, 1, 4, 8, 13, 15, 20, 21, 23, 26, 31) + D(5, 10, 24, 28)$$

Para ayudar al lector a seguir la explicación, esta función también se muestra en la forma de un mapa de Karnaugh en la figura 4.43.

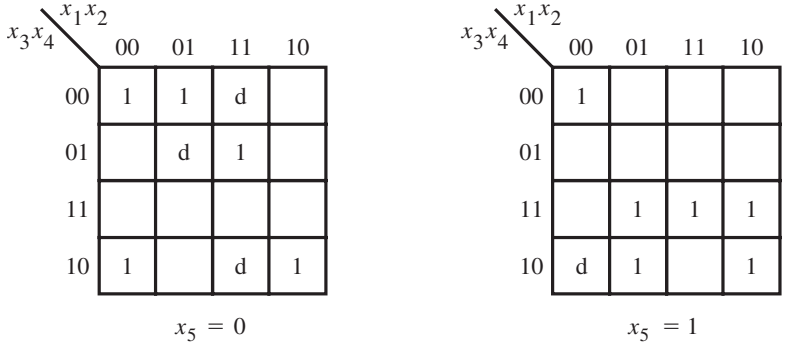


Figura 4.43 La función del ejemplo 4.20.

En lugar de que *f* se especifique en términos de mintérminos, suponga que *f* está dada como la siguiente expresión en SOP

$$f = \overline{x}_1\overline{x}_3\overline{x}_4\overline{x}_5 + x_1x_2\overline{x}_3x_4\overline{x}_5 + \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4x_5 + \overline{x}_1x_2x_3x_5 + x_1\overline{x}_2x_3x_5 + x_1x_3x_4x_5 + \overline{x}_2x_3\overline{x}_4\overline{x}_5$$

Además, supondremos que los no-importa se especifican con la expresión

$$\text{DC} = x_1x_2\overline{x}_4\overline{x}_5 + \overline{x}_1x_2\overline{x}_3x_4\overline{x}_5 + \overline{x}_1\overline{x}_2x_3\overline{x}_4x_5$$

Por tanto, el conjunto ON expresado como cubo es

$$\text{ON} = \{0\text{x}000,\ 11010,\ 00001,\ 011\text{x}1,\ 101\text{x}1,\ 1\text{x}111,\ \text{x}0100\}$$

y el conjunto de no-importa es

$$\text{DC} = \{11\text{x}00,\ 01010,\ 00101\}$$

La cobertura inicial C^0 consta de los conjuntos ON y DC:

$$C^0 = \{0\text{x}000,\ 11010,\ 00001,\ 011\text{x}1,\ 101\text{x}1,\ 1\text{x}111,\ \text{x}0100,\ 11\text{x}00,\ 01010,\ 00101\}$$

Al usar la operación $*$, las subsecuentes coberturas que se obtienen son

$$C^1 = \{0\text{x}000,\ 011\text{x}1,\ 101\text{x}1,\ 1\text{x}111,\ \text{x}0100,\ 11\text{x}00,\ 0000\text{x},\ 00\text{x}00,\ \text{x}1000,\ 010\text{x}0,\ 110\text{x}0,\ \text{x}1010,\ 00\text{x}01,\ \text{x}1111,\ 0\text{x}101,\ 1010\text{x},\ \text{x}0101,\ 1\text{x}100,\ 0010\text{x}\}$$

$$C^2 = \{0\text{x}000,\ 011\text{x}1,\ 101\text{x}1,\ 1\text{x}111,\ 11\text{x}00,\ \text{x}1111,\ 0\text{x}101,\ 1\text{x}100,\ \text{x}010\text{x},\ 00\text{x}0\text{x},\ \text{x}10\text{x}0\}$$

$$C^3 = C^2$$

En consecuencia, $P = C^2$.

Al usar la operación # se encuentra que hay dos implicantes primos esenciales: 00x0x (porque es el único que cubre el vértice 00001) y x10x0 (porque es el único que abarca el vértice 11010). Los mintérminos de *f* cubiertos por estos dos implicantes primos son $m(0, 1, 4, 8, 26)$.

A continuación, se encuentra que 1x100 puede eliminarse porque el único vértice del conjunto ON que cubre es 10100 (m_{20}), que también está cubierto por x010x y el costo de este implicante primo es más bajo. Observe que tras remover 1x100, el implicante primo x010x se convierte en esencial porque ninguno de los otros cubre el vértice 10100. En consecuencia, x010x ha de incluirse en la cobertura final. Cubre $m(20, 21)$.

Aún falta encontrar implicantes primos para cubrir $m(13, 15, 23, 31)$. Mediante el empleo de ramificación heurística, la cobertura de costo más bajo se obtiene incluyendo los implicantes primos 011x1 y 1x111. Por tanto, la cobertura final es

$$C_{mínima} = \{00\text{x}0\text{x},\ \text{x}10\text{x}0,\ \text{x}010\text{x},\ 011\text{x}1,\ 1\text{x}111\}$$

La correspondiente expresión en suma de productos es

$$f = \overline{x}_1\overline{x}_2\overline{x}_4 + x_2\overline{x}_3\overline{x}_5 + \overline{x}_2x_3\overline{x}_4 + \overline{x}_1x_2x_3x_5 + x_1x_3x_4x_5$$

Aunque este procedimiento resulta tedioso cuando se realiza a mano, no es difícil escribir un programa de computadora para implementar automáticamente el algoritmo. El lector debe comprobar la validez de la solución hallando la realización óptima del mapa de Karnaugh de la figura 4.43.

4.11 CONSIDERACIONES PRÁCTICAS

El propósito de la sección anterior fue brindar al lector cierta idea de cómo automatizar la minimización de las funciones lógicas para utilizarla en las herramientas CAD. Elegimos un esquema no muy difícil de explicar. Desde el punto de vista práctico, este esquema tiene ciertas desventajas. La principal es que el número de cubos que deben considerarse en el proceso puede ser muy grande.

Si la meta de la minimización se relaja de manera que no sea imperativo encontrar una implementación de costo mínimo, entonces es posible derivar técnicas heurísticas que produzcan buenos resultados en tiempo razonable. Una técnica de este tipo forma la base del ampliamente usado programa Espresso, que está disponible por Internet en la Universidad de California en Berkeley. Espresso es un programa de optimación en dos niveles. Tanto su entrada como su salida se especifican en el formato de cubos. En vez de usar la operación $*$ para hallar los implicantes primos, Espresso aplica una técnica de expansión de implicantes (véase, en el problema 4.30, una ilustración de la expansión de implicantes). Una explicación completa de Espresso se brinda en [19], mientras que bosquejos simplificados se encuentran en [3, 12].

La Universidad de California en Berkeley también ofrece dos programas, llamados MIS [20] y SIS [21], que sirven para diseñar circuitos multinivel. Ambos permiten que el usuario aplique varias técnicas de optimación multinivel a un circuito lógico. El usuario puede experimentar con diferentes estrategias de optimación mediante la aplicación de técnicas como la factorización y la descomposición a todo o parte de un circuito. SIS también incluye el algoritmo de Espresso para minimizar funciones en dos niveles, así como muchas otras técnicas de optimación.

En el mercado se encuentran varios sistemas CAD. Cuatro compañías cuyos productos se usan mucho son Cadence Design Systems, Mentor Graphics, Synopsys y Synplicity. La información de sus productos está disponible en la web. Cada compañía ofrece software de síntesis lógico que sirve para centrarse en varios tipos de chips, como PLD, arreglos de compuertas, celdas estándar y chips a la medida. Puesto que hay muchas posibles formas de sintetizar un circuito, como vimos en las secciones previas, cada producto comercial utiliza una estrategia de optimación lógica patentada con base en la heurística.

Para describir las herramientas CAD se han inventado términos nuevos. En particular cabe mencionar dos que se usan mucho en la industria: *síntesis lógica independiente de tecnología* y *mapeo de tecnología*. El primero se refiere a las técnicas aplicadas cuando se optima un circuito sin considerar los recursos disponibles en el chip destino. La mayor parte de las técnicas presentadas en este capítulo es de este tipo. El segundo término, mapeo de tecnología, se refiere a las técnicas empleadas para garantizar que el circuito producido por la síntesis lógica puede realizarse con los recursos lógicos disponibles en el chip destino. Un buen ejemplo de mapeo de tecnología es la transformación de un circuito en la forma de operaciones lógicas como AND y OR en otro que conste sólo de operaciones NAND. Este tipo de mapeo de tecnología se lleva a cabo cuando se dirige un circuito a un arreglo de compuertas que contiene sólo compuertas NAND. Otro ejemplo es la traducción de operaciones lógicas en tablas de consulta, el cual se realiza cuando se dirige un diseño a un FPGA.

En el capítulo 12 se explican las herramientas CAD pormenorizadamente. Se presenta un flujo de diseño típico que puede usar un diseñador para implementar un sistema digital.

4.12 EJEMPLOS DE CIRCUITOS SINTETIZADOS A PARTIR DE CÓDIGO DE VHDL

En la sección 2.10 mostramos cómo escribir programas simples en VHDL para describir funciones lógicas. En esta sección se presentan características adicionales de VHDL y se ofrecen más ejemplos de circuitos diseñados con código de ese lenguaje.

Recuérdese que una señal lógica se representa en VHDL como un objeto de datos, y cada objeto de datos tiene un tipo asociado. En los ejemplos de la sección 2.10 todos los objetos de datos tienen el tipo BIT, lo que significa que sólo pueden asumir los valores 0 y 1. Para dar más flexibilidad, VHDL proporciona otro tipo de datos llamado *STD_LOGIC*. Las señales representadas con este tipo pueden tener varios valores.

Como su nombre lo indica, STD_LOGIC busca servir como el tipo de datos estándar para la representación de señales lógicas. En la figura 4.44 se presenta un ejemplo que usa el tipo STD_LOGIC. La expresión lógica para *f* corresponde a la tabla de verdad de la figura 4.1; dicha tabla describe *f* en la forma canónica, que consiste en mintérminos. Para usar el tipo STD_LOGIC, el código de VHDL debe incluir las dos líneas dadas al principio de la figura. Tales instrucciones sirven como directivas al compilador de VHDL. Son necesarias porque el estándar original de VHDL, IEEE 1076, no incluye el tipo STD_LOGIC. La forma en que el nuevo tipo se agregó al lenguaje, en el estándar IEEE 1164, consistió en proporcionar la definición de STD_LOGIC como un conjunto de archivos que pueden incluirse con código de VHDL cuando se compile. El conjunto de archivos se llama *biblioteca* (*library*). El propósito de la primera línea de la figura 4.44 es declarar que el código usará la biblioteca IEEE.

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

ENTITY func1 IS
    PORT ( x1, x2, x3  : IN   STD_LOGIC ;
           f           : OUT  STD_LOGIC ) ;
END func1 ;

ARCHITECTURE LogicFunc OF func1 IS
BEGIN
    f <= (NOT x1 AND NOT x2 AND NOT x3) OR
         (NOT x1 AND x2 AND NOT x3) OR
         (x1 AND NOT x2 AND NOT x3) OR
         (x1 AND NOT x2 AND x3) OR
         (x1 AND x2 AND NOT x3) ;
END LogicFunc ;
```

Figura 4.44 Código de VHDL para la función de la figura 4.1.

En VHDL hay dos aspectos principales para la definición de un tipo de datos nuevo. Primero debe especificarse el conjunto de valores que un objeto de datos del tipo nuevo puede asumir. Para STD_LOGIC existen varios valores legales, pero los más importantes para describir las funciones lógicas son 0, 1, Z y –. El valor Z, que representa el estado de alta impedancia, se explicó en la sección 3.8.8. El valor lógico – representa la condición no-importa, que marcamos con *d* en la sección 4.4. El segundo requisito es que deben especificarse todos los usos legales del nuevo tipo de datos en código de VHDL. Por ejemplo, es necesario especificar que el tipo STD_LOGIC es legal para usarlo con operadores booleanos.

En la biblioteca IEEE uno de los archivos define el tipo de datos STD_LOGIC mismo y especifica algunos usos legales básicos, como para las operaciones booleanas. En la figura 4.44 la segunda línea del código indica al compilador de VHDL que use las definiciones en este archivo cuando compile el código. El archivo encapsula la definición de STD_LOGIC en lo que se conoce como *paquete*. El paquete se llama std_logic_1164. Es posible instruir al compilador de VHDL para que utilice sólo un subconjunto del paquete, pero el uso normal es especificar la palabra *all* para indicar que todo el paquete es de interés, como se hizo en la figura.

Para los ejemplos de código de VHDL presentados en este libro, casi siempre se usará únicamente el tipo STD_LOGIC. Además de simplificar el código, ocupar sólo un tipo de datos tiene otro beneficio. VHDL es un lenguaje con una enorme comprobación de tipos, lo que significa que el compilador verifica cuidadosamente todas las instrucciones de asignación de objetos de datos para asegurar que el tipo del objeto del lado izquierdo de la instrucción de asignación sea exactamente el mismo que el tipo del objeto de datos del lado derecho. Incluso si dos objetos de datos parecen compatibles desde el punto de vista intuitivo, digamos un objeto de tipo BIT y otro de tipo STD_LOGIC, el compilador no permitirá que uno se asigne al otro. Muchas herramientas de síntesis proporcionan pequeños programas de conversión para convertir —valga la redundancia— de un tipo a otro, pero este conflicto se evita usando sólo el tipo de datos STD_LOGIC en la mayoría de los casos. En el resto de la sección presentamos algunos ejemplos de código de VHDL. Mostraremos los resultados de sintetizar el código para implementarlo en dos tipos de chips, un CPLD y un FPGA.

Ejemplo 4.21

Se compiló el código de VHDL de la figura 4.44 para implementarlo en un CPLD, y las herramientas CAD produjeron la expresión

$$f = \overline{x}_3 + x_1\overline{x}_2$$

que es la forma mínima en suma de productos que se derivó con el mapa de Karnaugh de la figura 4.5*b*. En la figura 4.45 se muestra cómo implementar esta expresión en un CPLD. Los interruptores programados para estar cerrados se muestran en gris, lo mismo que las compuertas empleadas para implementar *f*. Observe que, en este caso, sólo se usan las dos compuertas AND superiores. Las tres compuertas AND de la parte inferior no tienen efecto porque cada una está conectada tanto a la versión verdadera como a la complementada de una entrada no utilizada, lo que ocasiona que la salida de la compuerta AND sea 0.

En la figura 4.46 se dan los resultados de sintetizar el código de VHDL de la figura 4.44 en un FPGA. Se supone que el compilador genera la misma forma en suma de productos que la expuesta líneas arriba. Puesto que las celdas lógicas en el chip son tablas de consulta de cuatro entradas, sólo se necesita una celda lógica para esta función. En la figura se muestra que las va-

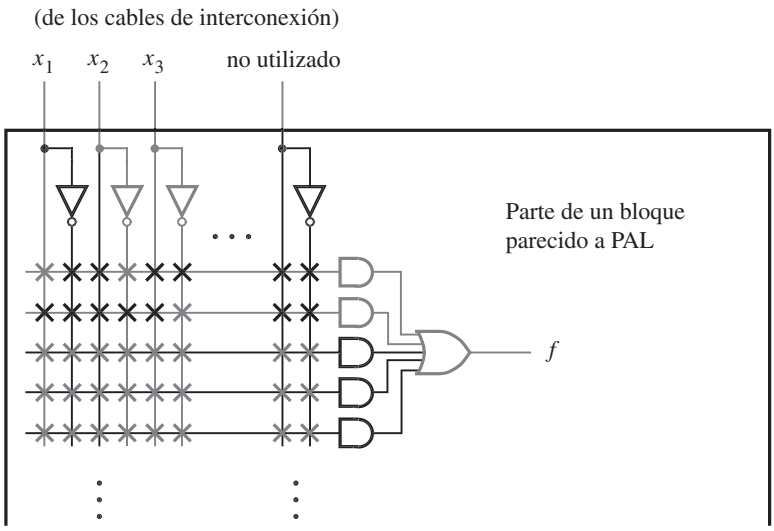
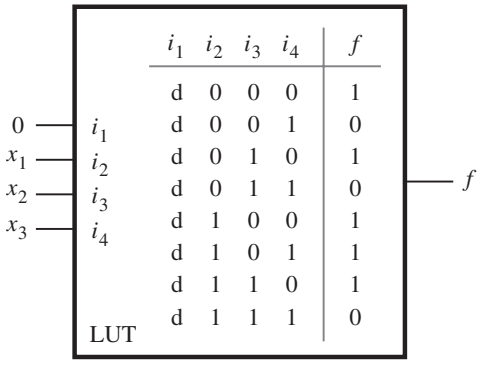


Figura 4.45 Implementación del código de VHDL de la figura 4.44.



i_1	i_2	i_3	i_4	f
d	0	0	0	1
d	0	0	1	0
d	0	1	0	1
d	0	1	1	0
d	1	0	0	1
d	1	0	1	1
d	1	1	0	1
d	1	1	1	0

Figura 4.46 El código de VHDL de la figura 4.44 implementado en una LUT.

riables x_1, x_2 y x_3 están conectadas a las entradas de la LUT llamadas i_2, i_3 e i_4. La entrada i_1 no se usa porque la función sólo requiere tres entradas. La tabla de verdad de la LUT indica que la entrada no usada se trata como un no-importa. Por tanto, sólo se muestra la mitad de las filas de la tabla, pues la otra mitad es idéntica. La entrada no usada en la LUT se muestra conectada a 0 en la figura, pero también podría conectarse a 1.

Es interesante considerar los beneficios ofrecidos por las optimaciones aplicadas en la síntesis lógica. Para la implementación en el CPLD, la función se simplificó de los cinco términos

producto originales en la forma canónica a sólo dos términos producto. Sin embargo, tanto la forma optimizada como la no optimizada encajan en una macrocelda individual en el chip y, por tanto, tienen el mismo costo (la macrocelda de la figura 4.45 tiene cinco términos producto). De manera similar, para el FPGA, no importa si la función se minimiza, ya que encaja en una sola LUT. La razón es que el circuito de nuestro ejemplo es muy pequeño. Para circuitos grandes es esencial llevar a cabo la optimación. En los ejemplos 4.22 y 4.23 se ilustran funciones lógicas para las que el costo de implementación se reduce cuando se optima.

Ejemplo 4.22

El código de VHDL de la figura 4.47 corresponde a la función f_1 de la figura 4.7. Como existen seis términos producto en la forma canónica, se necesitarían dos macroceldas del tipo de la figura 4.45. Cuando se sintetice mediante herramientas CAD, la expresión resultante podría ser

$$f = \overline{x}_2 x_3 + x_1 \overline{x}_3 x_4$$

que es la misma que la expresión derivada en la figura 4.7. Puesto que la expresión optimizada sólo tiene dos términos producto, puede realizarse con sólo una macrocelda y, por ende, resulta en un costo más bajo.

Cuando f_1 se sintetiza para implementarla en un FPGA, la expresión generada puede ser la misma que para el CPLD. Como la función sólo tiene cuatro entradas, únicamente necesita una LUT.

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

ENTITY func2 IS
    PORT ( x1, x2, x3, x4  : IN     STD_LOGIC ;
           f               : OUT  STD_LOGIC ) ;
END func2 ;

ARCHITECTURE LogicFunc OF func2 IS
BEGIN
    f <= (NOT x1 AND NOT x2 AND x3 AND NOT x4) OR
         (NOT x1 AND NOT x2 AND x3 AND x4) OR
         (x1 AND NOT x2 AND NOT x3 AND x4) OR
         (x1 AND NOT x2 AND x3 AND NOT x4) OR
         (x1 AND NOT x2 AND x3 AND x4) OR
         (x1 AND x2 AND NOT x3 AND x4) ;
END LogicFunc ;
```

Figura 4.47 Código de VHDL para f_1 de la figura 4.7.

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

ENTITY func3 IS
    PORT ( x1, x2, x3, x4, x5, x6, x7  : IN     STD_LOGIC ;
           f                           : OUT  STD_LOGIC ) ;
END func3 ;

ARCHITECTURE LogicFunc OF func3 IS
BEGIN
    f <= (x1 AND x3 AND NOT x6) OR
         (x1 AND x4 AND x5 AND NOT x6) OR
         (x2 AND x3 AND x7) OR
         (x2 AND x4 AND x5 AND x7) ;
END LogicFunc ;
```

Figura 4.48 Código de VHDL para la función de la sección 4.7.

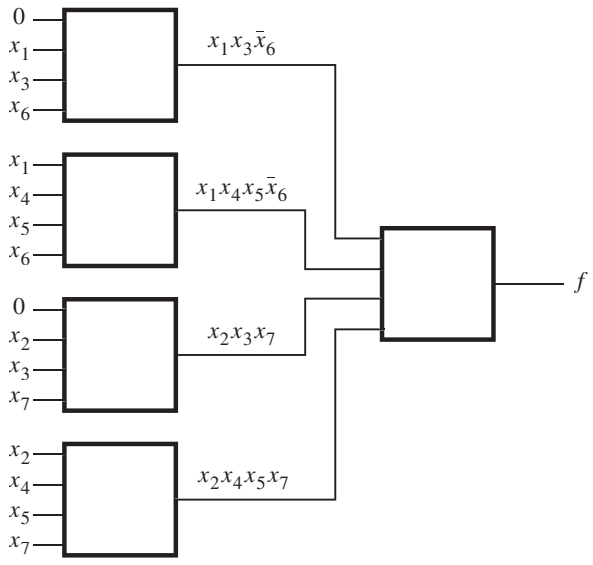
Ejemplo 4.23 En la sección 4.6 usamos una función lógica de siete variables como una motivación para la síntesis en multinivel. Esta función está dada en el código de VHDL de la figura 4.48. La expresión lógica se halla en forma mínima de suma de productos. Cuando se sintetiza para implementarla en un CPLD, las herramientas CAD no realizan optimaciones. La función requiere una macrocelda. Esta función es más interesante cuando se considera su implementación en un FPGA con LUT de cuatro entradas. Puesto que existen siete entradas, se requiere más de una LUT. Si la función se implementa directamente como se da en el código de VHDL, entonces se necesitan cinco LUT, como se muestra en la figura 4.49*a*. En lugar de mostrar la tabla de verdad programada en cada LUT, se presenta la función lógica que se implementó en la salida de la LUT. Pero si la función se sintetiza como

$$f = (x_1\overline{x}_6 + x_2x_7)(x_3 + x_4x_5)$$

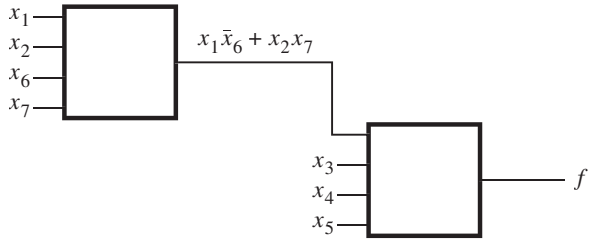
que es la expresión que derivamos mediante factorización en la sección 4.6, entonces f puede implementarse sólo con dos LUT, como se ilustra en la figura 4.49*b*. Una LUT produce el término $S = x_1\overline{x}_6 + x_2x_7$; la otra implementa la función de cuatro entradas $f = Sx_3 + Sx_4x_5$.

4.13 COMENTARIOS FINALES

La intención de este capítulo fue ofrecer al lector algunas ideas de diversos aspectos de la síntesis de funciones lógicas. Ahora que el lector se siente cómodo con los conceptos fundamentales, podemos examinar circuitos digitales de naturaleza más refinada. En el capítulo siguiente describimos circuitos que realizan operaciones aritméticas, que son una parte clave de las computadoras.



a) Realización en suma de productos



b) Realización factorizada

Figura 4.49 Implementación del código de VHDL de la figura 4.48.

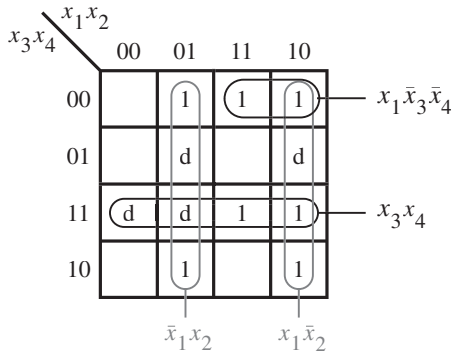
4.14 EJEMPLOS DE PROBLEMAS RESUELTOS

En esta sección presentamos algunos problemas típicos que el lector puede encontrar y mostramos cómo resolverlos.

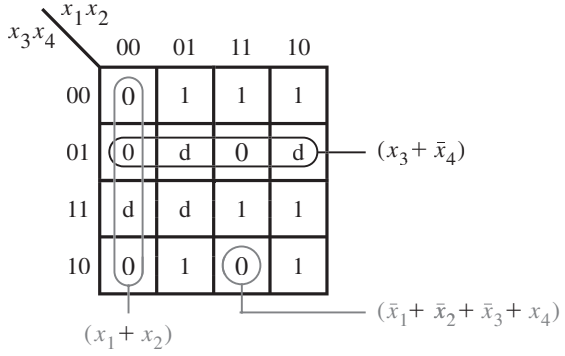
Ejemplo 4.24

Problema: Determine las expresiones en SOP y POS de costo mínimo para la función $f(x_1, x_2, x_3, x_4) = \sum m(4, 6, 8, 10, 11, 12, 15) + D(3, 5, 7, 9)$.

Solución: La función puede representarse en la forma de mapa de Karnaugh como se muestra en la figura 4.50*a*. Observe que la ubicación de los mintérminos en el mapa es como se indica en la figura 4.6.



a) Determinación de la expresión en SOP



b) Determinación de la expresión en POS

Figura 4.50 Mapas de Karnaugh para el ejemplo 4.24.

Para encontrar la expresión en SOP de costo mínimo es preciso hallar los implicantes primos que cubran todos los 1 del mapa. Los no-importa pueden usarse según se desee. El mintérmino m_6 sólo queda cubierto por el implicante primo $\overline{x}_1x_2$; por tanto, este implicante es esencial y debe incluirse en la expresión final. De manera similar, los implicantes primos $x_1\overline{x}_2$ y x_3x_4 son esenciales porque son los únicos que cubren m_{10} y m_{15}, respectivamente. Estos tres implicantes primos cubren todos los mintérminos para los que $f = 1$, excepto m_{12}. Este mintérmino puede cubrir de dos formas, al elegir $x_1\overline{x}_3\overline{x}_4$ o $x_2\overline{x}_3\overline{x}_4$. Como ambos implicantes primos tienen el mismo costo, puede elegirse cualquiera de ellos. Si es el primero, la expresión en SOP deseada es

$$f = \overline{x}_1x_2 + x_1\overline{x}_2 + x_3x_4 + x_1\overline{x}_3\overline{x}_4$$

Estos implicantes primos se encierran en círculos en el mapa.

La expresión en POS deseada puede encontrarse como se indica en la figura 4.50*b*. En este caso, hay que hallar los términos suma que cubren todos los 0 en la función. Nótese que escribimos 0 explícitamente en el mapa para destacar este hecho. El término $(x_1 + x_2)$ es esencial para cubrir los 0 en los cuadrados 0 y 2, que corresponden a los maxitérminos M_0 y M_2. Los términos $(x_3 + \overline{x}_4)$ y $(\overline{x}_1 + \overline{x}_2 + \overline{x}_3 + x_4)$ deben usarse para cubrir los 0 en los cuadrados 13 y 14, respectivamente. Como estos tres términos suma cubren todos los 0 del mapa, la expresión en POS es

$$f = (x_1 + x_2)(x_3 + \overline{x}_4)(\overline{x}_1 + \overline{x}_2 + \overline{x}_3 + x_4)$$

Los términos suma elegidos se encierran en círculos en el mapa.

Observe el uso de los no-importa en este ejemplo. Para obtener una expresión en SOP de costo mínimo hemos supuesto que los cuatro no-importa tienen el valor 1. Pero la expresión en POS de costo mínimo sólo se vuelve posible si suponemos que los no-importa 3, 5 y 9 tienen el valor 0, en tanto que el no-importa 7 tiene el valor 1. Esto significa que las expresiones en SOP y POS resultantes no son idénticas en términos de las funciones que representan. Cubren de manera idéntica todas las combinaciones para las que *f* se especifica como 1 o 0, pero difieren en las combinaciones 3, 5 y 9. Desde luego, esta diferencia no tiene importancia, porque las combinaciones no-importa nunca se aplicarán como entradas a los circuitos implementados.

Ejemplo 4.25

Problema: Use mapas de Karnaugh para encontrar las expresiones en SOP y POS de costo mínimo para la función

$$f(x_1, \ldots, x_4) = \overline{x}_1\overline{x}_3\overline{x}_4 + x_3x_4 + \overline{x}_1\overline{x}_2x_4 + x_1x_2\overline{x}_3x_4$$

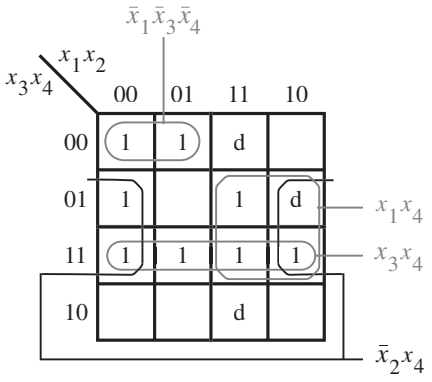
si se supone que también existen no-importa definidos como $D = \sum(9, 12, 14)$.

Solución: El mapa de Karnaugh que representa esta función se muestra en la figura 4.51*a*. El mapa se deriva colocando 1 que correspondan a cada término producto en la expresión usada para especificar *f*. El término $\overline{x}_1\overline{x}_3\overline{x}_4$ corresponde a los mintérminos 0 y 4. El término x_3x_4 representa la tercera fila del mapa, que comprende los mintérminos 3, 7, 11 y 15. El término $\overline{x}_1\overline{x}_2x_4$ especifica los mintérminos 1 y 3. El cuarto término producto representa el mintérmino 13. El mapa también incluye las tres condiciones no-importa.

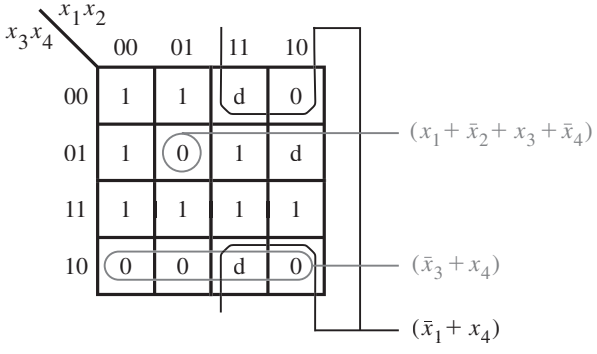
Para hallar la expresión en SOP deseada hay que encontrar el conjunto de implicantes primos menos costoso que cubra todos los 1 en el mapa. El término x_3x_4 es un implicante primo que debe incluirse porque es el único que cubre el mintérmino 7; también cubre los mintérminos 3, 11 y 15. El mintérmino 4 puede cubrirse con $\overline{x}_1\overline{x}_3\overline{x}_4$ o con $x_2\overline{x}_3\overline{x}_4$. Ambos términos tienen el mismo costo; se elegirá $\overline{x}_1\overline{x}_3\overline{x}_4$ porque también cubre el mintérmino 0. El mintérmino 1 puede cubrirse con $\overline{x}_1\overline{x}_2\overline{x}_3$ o con $\overline{x}_2x_4$; se debe elegir el último porque su costo es menor. Esto sólo deja por cubrir el mintérmino 13, lo que puede hacerse con x_1x_4 o con x_1x_2 a costos iguales. Al elegir x_1x_4 la expresión en SOP de costo mínimo es

$$f = x_3x_4 + \overline{x}_1\overline{x}_3\overline{x}_4 + \overline{x}_2x_4 + x_1x_4$$

En la figura 4.51*b* se muestra cómo encontrar la expresión en POS. El término suma $(\overline{x}_3 + x_4)$ cubre los 0 en la fila inferior. Para cubrir el 0 en el cuadrado 8 hay que incluir $(\overline{x}_1 + x_4)$. Los 0



a) Determinación de la expresión en SOP



b) Determinación de la expresión en POS

Figura 4.51 Mapas de Karnaugh para el ejemplo 4.25.

restantes, en el cuadrado 5, deben cubrirse con $(x_1 + \overline{x}_2 + x_3 + \overline{x}_4)$. Por tanto, la expresión en POS de costo mínimo es

$$f = (\overline{x}_3 + x_4)(\overline{x}_1 + x_4)(x_1 + \overline{x}_2 + x_3 + \overline{x}_4)$$

Ejemplo 4.26 **Problema:** Utilice el método tabular expuesto en la sección 4.9 para derivar una expresión en SOP de costo mínimo para la función

$$f(x_1, \ldots, x_4) = \overline{x}_1\overline{x}_3\overline{x}_4 + x_3x_4 + \overline{x}_1\overline{x}_2x_4 + x_1x_2\overline{x}_3x_4$$

si se supone que también existen no-importa definidos como $D = \sum(9, 12, 14)$.

Solución: Con el método tabular es necesario comenzar con la función definida en la forma de mintérminos. Como se encontró en la figura 4.51*a*, la función *f* también puede representarse como

$$f(x_1, \ldots, x_4) = \sum m(0, 1, 3, 4, 7, 11, 13, 15) + D(9, 12, 14)$$

Los correspondientes 11 cubos 0 se colocan en la lista 1 de la figura 4.52.

Ahora, realice una comparación por pares de todos los cubos 0 para determinar los cubos 1 que se muestran en la lista 2, los cuales se obtienen al combinar pares de cubos 0. Note que todos los cubos 0 se incluyen en los cubos 1, como se indica con las marcas en la lista 1. A continuación, realice una comparación por pares de todos los cubos 1 para obtener los cubos 2 en la lista 3. Algunos de dichos cubos 2 pueden generarse de varias formas, pero no es útil listar mas de una vez cubos 2 (por ejemplo, x0x1 en la lista 3 puede obtenerse combinando de la lista 2 los cubos 1, 3 y 9, 11, o si se usan los cubos 1, 9 y 3, 11). Note que todos los cubos 1, excepto tres, se incluyen en los cubos 2. No es posible generar algún cubo 3; por tanto, todos los términos que no se incluyen en algún otro término (los términos sin marca en la lista 2 y todos los términos en la lista 3) son los implicantes primos de *f*. El conjunto de implicantes primos es

$$\begin{aligned} P &= \{000x, 0x00, x100, x0x1, xx11, 1xx1, 11xx\} \\ &= \{p_1, p_2, p_3, p_4, p_5, p_6, p_7\} \end{aligned}$$

Para encontrar la cobertura de costo mínimo para *f*, construya la tabla de la figura 4.53*a* que muestra todos los implicantes primos y los mintérminos que deben cubrirse: aquellos para los que $f = 1$. Para indicar que un mintérmino se cubre mediante un implicante primo en particular se coloca una marca. Puesto que el mintérmino 7 se cubre sólo por p_5, este implicante primo

Lista 1

0	0 0 0 0	✓
1	0 0 0 1	✓
4	0 1 0 0	✓
3	0 0 1 1	✓
9	1 0 0 1	✓
12	1 1 0 0	✓
7	0 1 1 1	✓
11	1 0 1 1	✓
13	1 1 0 1	✓
14	1 1 1 0	✓
15	1 1 1 1	✓

Lista 2

0,1	0 0 0 x	
0,4	0 x 0 0	
1,3	0 0 x 1	✓
1,9	x 0 0 1	✓
4,12	x 1 0 0	
3,7	0 x 1 1	✓
3,11	x 0 1 1	✓
9,11	1 0 x 1	✓
9,13	1 x 0 1	✓
12,13	1 1 0 x	✓
12,14	1 1 x 0	✓
7,15	x 1 1 1	✓
11,15	1 x 1 1	✓
13,15	1 1 x 1	✓
14,15	1 1 1 x	✓

Lista 3

1,3,9,11	x 0 x 1
3,7,11,15	x x 1 1
9,11,13,15	1 x x 1
12,13,14,15	1 1 x x

Figura 4.52 Generación de implicantes primos para la función del ejemplo 4.26.

Implicante primo	Mintérmino 0	1	3	4	7	11	13	15
p_1 = 0 0 0 x	✓	✓						
p_2 = 0 x 0 0	✓			✓				
p_3 = x 1 0 0				✓				
p_4 = x 0 x 1		✓	✓			✓		
p_5 = x x 1 1			✓		✓	✓		✓
p_6 = 1 x x 1						✓	✓	✓
p_7 = 1 1 x x							✓	✓

a) Tabla inicial de cobertura de implicantes primos

Implicante primo	Mintérmino 0	1	4	13
p_1 = 0 0 0 x	✓	✓		
p_2 = 0 x 0 0	✓		✓	
p_4 = x 0 x 1		✓		
p_6 = 1 x x 1				✓

b) Después de eliminar las filas p_3, p_5, y p_7, y las columnas 3, 7, 11 y 13

Figura 4.53 Selección de una cobertura para la función del ejemplo 4.26.

debe incluirse en la cobertura final. Observe que la fila p_2 domina a la p_3, por lo que esta última puede eliminarse. De manera similar, la fila p_6 domina a la p_7. La eliminación de las filas p_5, p_3 y p_7, así como de las columnas 3, 7, 11 y 15 (que están cubiertas por p_5) conduce a la tabla reducida de la figura 4.53*b*. En ella p_2 y p_6 son esenciales. Cubren los mintérminos 0, 4 y 13. Por consiguiente, sólo falta por cubrir el mintérmino 1, lo que se hace eligiendo p_1 o p_4. Como p_4 tiene un costo más bajo, debe elegirse. En consecuencia, la cobertura final es

$$\begin{aligned} C &= \{p_2, p_4, p_5, p_6\} \\ &= \{0x00,\ x0x1,\ xx11,\ 1xx1\} \end{aligned}$$

y la función se implementa como

$$f = \overline{x}_1\overline{x}_3\overline{x}_4 + \overline{x}_2x_4 + x_3x_4 + x_1x_4$$

Ejemplo 4.27

Problema: Use la operación producto $*$ para encontrar todos los implicantes primos de la función

$$f(x_1, \ldots, x_4) = \overline{x}_1\overline{x}_3\overline{x}_4 + x_3x_4 + \overline{x}_1\overline{x}_2x_4 + x_1x_2\overline{x}_3x_4$$

si se supone que también existen no-importa definidos como $D = \sum(9, 12, 14)$.

Solución: El conjunto ON para esta función es

$$\text{ON} = \{0\text{x}00, \text{xx}11, 00\text{x}1, 1101\}$$

La cobertura inicial, que consta del conjunto ON y los no-importa, es

$$C^0 = \{0\text{x}00, \text{xx}11, 00\text{x}1, 1101, 1001, 1100, 1110\}$$

Al usar la operación $*$, las subsecuentes coberturas obtenidas son

$$C^1 = \{0\text{x}00, \text{xx}11, 00\text{x}1, 000\text{x}, \text{x}100, 11\text{x}1, 10\text{x}1, 111\text{x}, \text{x}001, 1\text{x}01, 110\text{x}, 11\text{x}0\}$$
$$C^2 = \{0\text{x}00, \text{xx}11, 000\text{x}, \text{x}100, \text{x}0\text{x}1, 1\text{xx}1, 11\text{xx}\}$$
$$C^3 = C^2$$

Por tanto, el conjunto de todos los implicantes primos es

$$P = \{\overline{x}_1\overline{x}_3\overline{x}_4, x_3x_4, \overline{x}_1\overline{x}_2\overline{x}_3, x_2\overline{x}_3\overline{x}_4, \overline{x}_2x_4, x_1x_4, x_1x_2\}$$

Ejemplo 4.28

Problema: Encuentre la implementación de costo mínimo para la función

$$f(x_1, \ldots, x_4) = \overline{x}_1\overline{x}_3\overline{x}_4 + x_3x_4 + \overline{x}_1\overline{x}_2x_4 + x_1x_2\overline{x}_3x_4$$

si se supone que también existen no-importa definidos como $D = \sum(9, 12, 14)$.

Solución: Se trata de la misma función empleada en los ejemplos 4.25 a 4.27. En ellos se encontró que la implementación en SOP de costo mínimo es

$$f = x_3x_4 + \overline{x}_1\overline{x}_3\overline{x}_4 + \overline{x}_2x_4 + x_1x_4$$

que requiere cuatro compuertas AND, una OR y 13 entradas a ellas, para un costo total de 18.

La implementación en POS de costo mínimo es

$$f = (\overline{x}_3 + x_4)(\overline{x}_1 + x_4)(x_1 + \overline{x}_2 + x_3 + \overline{x}_4)$$

que requiere tres compuertas OR, una AND y 11 entradas a ellas, para un costo total de 15.

También puede considerarse una realización multinivel para la función. Al aplicar el concepto de factorización a la expresión en SOP anterior se produce

$$f = (x_1 + \overline{x}_2 + x_3)x_4 + \overline{x}_1\overline{x}_3\overline{x}_4$$

Esta implementación requiere dos compuertas AND, dos OR y 10 entradas a ellas, para un costo total de 14. En comparación con las implementaciones en SOP y POS, ésta tiene el costo más

bajo en términos de compuertas y entradas, pero resulta en un circuito más lento porque existen tres niveles de compuertas por los que las señales deben propagarse. Desde luego, si esta función se implementa en un FPGA, entonces sólo se necesita una LUT.

Ejemplo 4.29 **Problema:** En varios FPGA comerciales los bloques lógicos son LUT de cuatro entradas. Se pueden usar dos de tales LUT, conectadas como se muestra en la figura 4.54, para implementar funciones de siete variables usando la descomposición

$$f(x_1, \ldots, x_7) = f[g(x_1, \ldots, x_4), x_5, x_6, x_7]$$

Es fácil ver que funciones tales como $f = x_1x_2x_3x_4x_5x_6x_7$ y $f = x_1+x_2+x_3+x_4+x_5+x_6+x_7$ pueden implementarse de esta forma. Demuestre que existen otras funciones de siete variables que no pueden implementarse con dos LUT de cuatro entradas.

Solución: La tabla de verdad de una función de siete variables puede ordenarse como se bosqueja en la figura 4.55. Hay $2^7 = 128$ mintérminos. Cada combinación de las variables x_1, x_2, x_3 y x_4 selecciona una de las 16 columnas de la tabla de verdad, mientras que cada combinación de x_5, x_6 y x_7 selecciona una de ocho filas. Puesto que hay que usar el circuito de la figura 4.54, la tabla de verdad para f también puede definirse en términos de la subfunción g. En este caso, g es la que selecciona una de las 16 columnas de la tabla de verdad, en vez de x_1, x_2, x_3 y x_4. Como g sólo puede tener dos valores posibles, 0 y 1, nada más se pueden tener dos columnas en la tabla de verdad. Esto es posible si únicamente existen dos patrones distintos de 1 y 0 en las 16 columnas de la figura 4.54. Por tanto, sólo un subconjunto relativamente pequeño de funciones de siete variables pueden realizarse con sólo dos LUT.

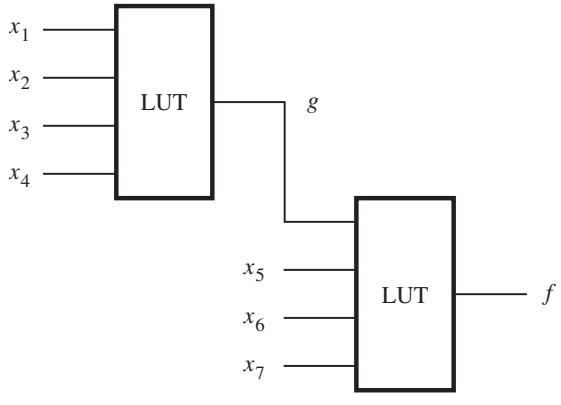


Figura 4.54 Circuito para el ejemplo 4.29.

$x_5x_6x_7$ \ $x_1x_2x_3x_4$	0000	0001	• • •	1110	1111
000	m_0	m_8		m_{112}	m_{120}
001	m_1	m_9		m_{113}	m_{121}
010	m_2	m_{10}		m_{114}	m_{122}
011	m_3	m_{11}	• • •	m_{115}	m_{123}
100	m_4	m_{12}		m_{116}	m_{124}
101	m_5	m_{13}		m_{117}	m_{125}
110	m_6	m_{14}		m_{118}	m_{126}
111	m_7	m_{15}		m_{119}	m_{127}

Figura 4.55 Posible formato para tablas de verdad de funciones de siete variables.

PROBLEMAS

Al final del libro se proporcionan las respuestas a los problemas marcados con asterisco.

***4.1** Encuentre las formas en SOP y POS de costo mínimo para la función $f(x_1, x_2, x_3) = \sum m(1, 2, 3, 5)$.

***4.2** Repita el problema 4.1 para la función $f(x_1, x_2, x_3) = \sum m(1, 4, 7) + D(2, 5)$.

4.3 Repita el problema 4.1 para la función $f(x_1, \ldots, x_4) = \prod M(0, 1, 2, 4, 5, 7, 8, 9, 10, 12, 14, 15)$.

4.4 Repita el problema 4.1 para la función $f(x_1, \ldots, x_4) = \sum m(0, 2, 8, 9, 10, 15) + D(1, 3, 6, 7)$.

***4.5** Repita el problema 4.1 para la función $f(x_1, \ldots, x_5) = \prod M(1, 4, 6, 7, 9, 12, 15, 17, 20, 21, 22, 23, 28, 31)$.

4.6 Repita el problema 4.1 para la función $f(x_1, \ldots, x_5) = \sum m(0, 1, 3, 4, 6, 8, 9, 11, 13, 14, 16, 19, 20, 21, 22, 24, 25) + D(5, 7, 12, 15, 17, 23)$.

4.7 Repita el problema 4.1 para la función $f(x_1, \ldots, x_5) = \sum m(1, 4, 6, 7, 9, 10, 12, 15, 17, 19, 20, 23, 25, 26, 27, 28, 30, 31) + D(8, 16, 21, 22)$.

4.8 Encuentre cinco funciones de tres variables para las que la forma en producto de sumas tenga menor costo que la forma en suma de productos.

***4.9** Una función lógica de cuatro variables que es igual a 1 si cualesquiera tres o las cuatro de sus variables son iguales a 1 se llama función *mayoritaria*. Diseñe un circuito en SOP de costo mínimo que implemente esta función mayoritaria.

4.10 Derive una realización en costo mínimo de la función de cuatro variables que es igual a 1 si exactamente dos o exactamente tres de sus variables son iguales a 1; de otro modo es igual a 0.

***4.11** Prueba o muestre un contraejemplo para la afirmación siguiente: si una función f tiene una única expresión en SOP de costo mínimo, entonces también tiene una única expresión en POS de costo mínimo.

***4.12** Un circuito con dos salidas tiene que implementar las funciones siguientes:

$$f(x_1, \ldots, x_4) = \sum m(0, 2, 4, 6, 7, 9) + D(10, 11)$$

$$g(x_1, \ldots, x_4) = \sum m(2, 4, 9, 10, 15) + D(0, 13, 14)$$

Diseñe el circuito de costo mínimo y compare su costo con los costos combinados de dos circuitos que implementen f y g por separado. Suponga que las variables de entrada están disponibles tanto en forma sin complementar como complementada.

4.13 Repita el problema 4.12 para las funciones siguientes

$$f(x_1, \ldots, x_5) = \sum m(1, 4, 5, 11, 27, 28) + D(10, 12, 14, 15, 20, 31)$$

$$g(x_1, \ldots, x_5) = \sum m(0, 1, 2, 4, 5, 8, 14, 15, 16, 18, 20, 24, 26, 28, 31) + D(10, 11, 12, 27)$$

***4.14** Implemente el circuito lógico de la figura 4.23 usando solamente compuertas NAND.

***4.15** Implemente el circuito lógico de la figura 4.23 usando solamente compuertas NOR.

4.16 Implemente el circuito lógico de la figura 4.25 usando solamente compuertas NAND.

4.17 Implemente el circuito lógico de la figura 4.25 usando solamente compuertas NOR.

***4.18** Considere la función $f = x_3x_5 + \bar{x}_1x_2x_4 + x_1\bar{x}_2\bar{x}_4 + x_1x_3\bar{x}_4 + \bar{x}_1x_3x_4 + \bar{x}_1x_2x_5 + x_1\bar{x}_2x_5$. Derive un circuito de costo mínimo que la implemente usando compuertas NOT, AND y OR.

4.19 Derive un circuito de costo mínimo que implemente la función $f(x_1, \ldots, x_4) = \sum m(4, 7, 8, 11) + D(12, 15)$.

4.20 Encuentre la realización más simple de la función $f(x_1, \ldots, x_4) = \sum m(0, 3, 4, 7, 9, 10, 13, 14)$, si se supone que las compuertas lógicas tienen una entrada de carga máxima de dos.

***4.21** Encuentre el circuito de costo mínimo para la función $f(x_1, \ldots, x_4) = \sum m(0, 4, 8, 13, 14, 15)$. Suponga que las variables de entrada están disponibles sólo en forma sin complementar. (*Sugerencia*: aplique descomposición funcional.)

4.22 Use descomposición funcional para encontrar la mejor implementación de la función $f(x_1, \ldots, x_5) = \sum m(1, 2, 7, 9, 10, 18, 19, 25, 31) + D(0, 15, 20, 26)$. ¿Cómo se compara su implementación con la implementación en SOP de costo más bajo? Proporcione los costos.

***4.23** Use el método tabular expuesto en la sección 4.9 para hallar una realización en SOP de costo mínimo para la función

$$f(x_1, \ldots, x_4) = \sum m(0, 2, 4, 5, 7, 8, 9, 15)$$

4.24 Repita el problema 4.23 para la función

$$f(x_1, \ldots, x_4) = \sum m(0, 4, 6, 8, 9, 15) + D(3, 7, 11, 13)$$

4.25 Repita el problema 4.23 para la función

$$f(x_1, \ldots, x_4) = \sum m(0, 3, 4, 5, 7, 9, 11) + D(8, 12, 13, 14)$$

4.26 Demuestre que son válidas las reglas parecidas a distributiva siguientes

$$(A \cdot B)\#C = (A\#C) \cdot (B\#C)$$
$$(A + B)\#C = (A\#C) + (B\#C)$$

4.27 Use la representación cúbica y el método expuesto en la sección 4.10 para hallar una realización en SOP de costo mínimo de la función $f(x_1, \ldots, x_4) = \sum m(0, 2, 4, 5, 7, 8, 9, 15)$.

4.28 Repita el problema 4.27 para la función $f(x_1, \ldots, x_5) = \overline{x}_1\overline{x}_3\overline{x}_5 + x_1x_2\overline{x}_3 + x_2x_3\overline{x}_4x_5 + x_1\overline{x}_2\overline{x}_3x_4 + x_1x_2x_3x_4\overline{x}_5 + \overline{x}_1x_2x_4\overline{x}_5 + \overline{x}_1\overline{x}_3x_4x_5$.

4.29 Utilice la representación cúbica y el método expuesto en la sección 4.10 para hallar una realización en SOP de costo mínimo de la función $f(x_1, \ldots, x_4)$, definida por el conjunto ON ON = {00x0, 100x, x010, 1111} y el conjunto de no-importa DC = {00x1, 011x}.

4.30 En la sección 4.10.1 mostramos cómo usar la operación $*$ para hallar los implicantes primos de una función *f*. Otra posibilidad consiste en encontrar los implicantes primos mediante la expansión de los implicantes en la cobertura inicial de la función. Un implicante se *expande* eliminando una literal para crear un implicante más grande (en términos del número de vértices cubiertos). Un implicante más grande sólo es válido si no incluye vértice alguno para el que $f = 0$. Los implicantes válidos más grandes que se obtienen en el proceso de expansión son los primos. En la figura P4.1 se ilustra la expansión del implicante $\overline{x}_1x_2x_3$ de la función de la figura 4.9, que también se usa en el ejemplo 4.16. A partir de la figura 4.9, note que

$$\overline{f} = x_1\overline{x}_2\overline{x}_3 + x_1\overline{x}_2x_3 + x_1x_2\overline{x}_3$$

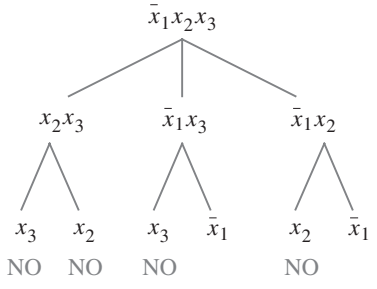


Figura P4.1 Expansión del implicante $\overline{x}_1x_2x_3$.

En la figura P4.1 la palabra NO se usa para indicar que el término expandido no es válido porque incluye uno o más vértices de $\overline{f}$. A partir de la gráfica es claro que los implicantes válidos más grandes que surgen de esta expansión son x_2x_3 y $\overline{x}_1$; se trata de los implicantes primos de *f*.

Expanda los otros cuatro implicantes dados en la cobertura inicial del ejemplo 4.14 para encontrar todos los implicantes primos de *f*. ¿Cuál es la complejidad relativa de este procedimiento en comparación con la técnica del producto $*$?

4.31 Repita el problema 4.30 para la función del ejemplo 4.17. Expanda los implicantes dados en la cobertura inicial C^0.

***4.32** Considere las expresiones lógicas

$$f = x_1\bar{x}_2\bar{x}_5 + \bar{x}_1\bar{x}_2\bar{x}_4\bar{x}_5 + x_1x_2x_4x_5 + \bar{x}_1\bar{x}_2x_3\bar{x}_4 + x_1\bar{x}_2x_3x_5 + \bar{x}_2\bar{x}_3x_4\bar{x}_5 + x_1x_2x_3x_4\bar{x}_5$$
$$g = \bar{x}_2x_3\bar{x}_4 + \bar{x}_2\bar{x}_3\bar{x}_4\bar{x}_5 + x_1x_3x_4\bar{x}_5 + x_1\bar{x}_2x_4\bar{x}_5 + x_1x_3x_4x_5 + \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_5 + x_1x_2\bar{x}_3x_4x_5$$

Pruebe o debata que $f = g$.

4.33 Considere el circuito de la figura P4.2, que implementa las funciones f y g. ¿Cuál es su costo, si se supone que las variables de entrada están disponibles tanto en verdadero como en complementado? Rediseñe el circuito para implementar las mismas funciones, pero a un costo cuan bajo sea posible. ¿Cuál es el costo de este circuito?

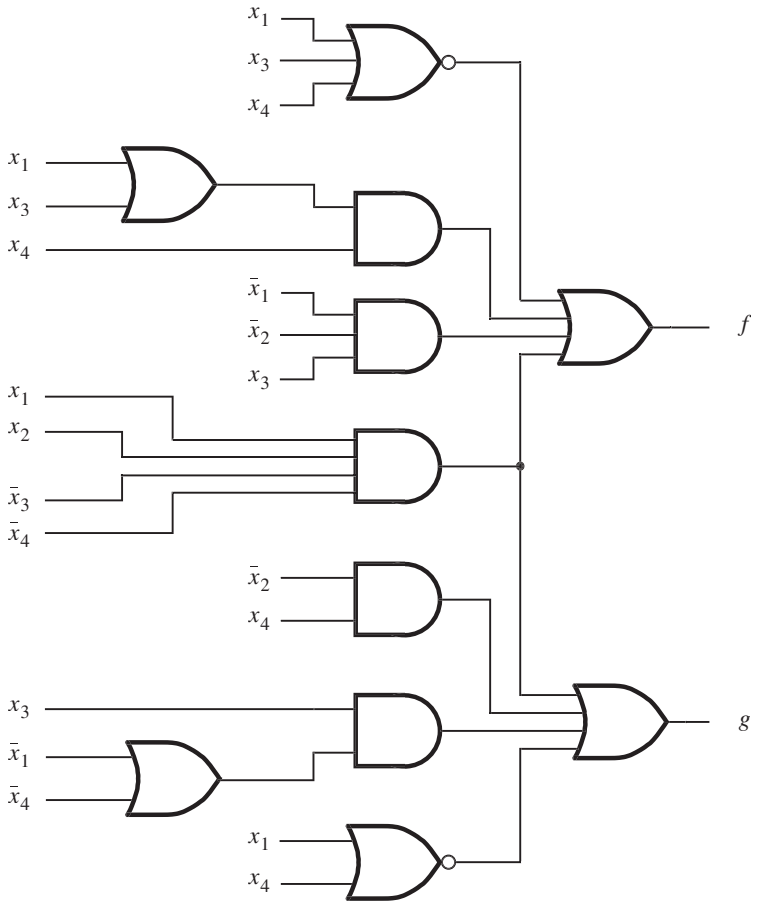


Figura P4.2 Circuito para el problema 4.33.

4.34 Repita el problema 4.33 para el circuito de la figura P4.3. En el circuito use sólo compuertas NAND.

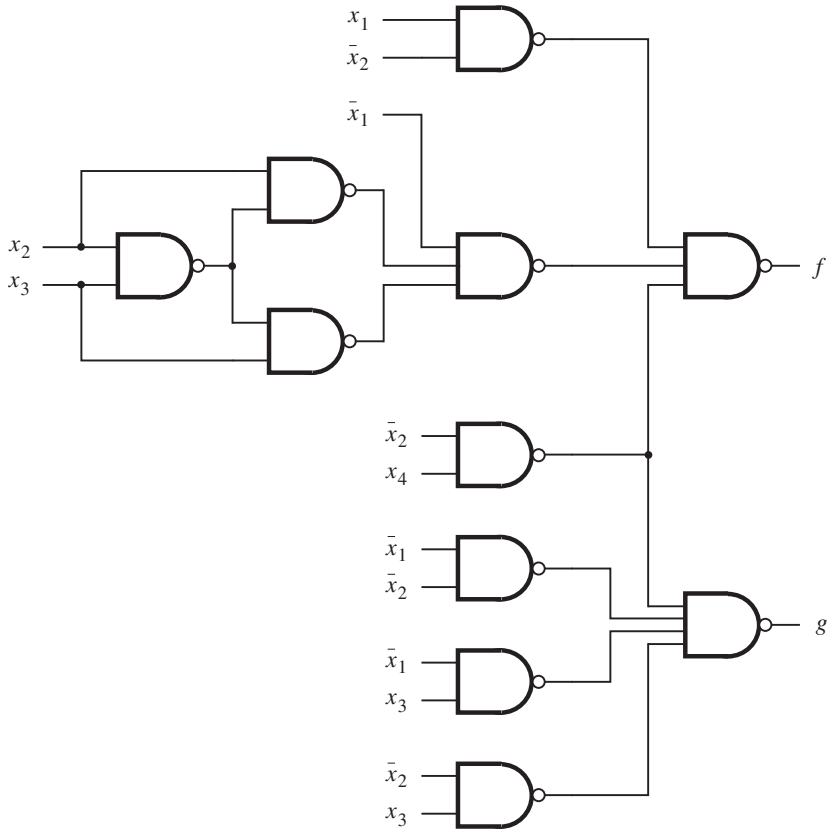


Figura P4.3 Circuito para el problema 4.34.

4.35 Escriba el código de VHDL para implementar el circuito de la figura 4.25*b*.

4.36 Escriba el código de VHDL para implementar el circuito de la figura 4.27*c*.

4.37 Escriba el código de VHDL para implementar el circuito de la figura 4.28*b*.

4.38 Escriba el código de VHDL para implementar la función $f(x_1, \ldots, x_4) = \sum m(0, 1, 2, 4, 5, 7, 8, 9, 11, 12, 14, 15)$.

4.39 Escriba el código de VHDL para implementar la función $f(x_1, \ldots, x_4) = \sum m(1, 4, 7, 14, 15) + D(0, 5, 9)$.

4.40 Escriba el código de VHDL para implementar la función $f(x_1, \ldots, x_4) = \Pi M(6, 8, 9, 12, 13)$.

4.41 Escriba el código de VHDL para implementar la función $f(x_1, \ldots, x_4) = \Pi M(3, 11, 14) + D(0, 2, 10, 12)$.

BIBLIOGRAFÍA

1. M. Karnaugh, "A Map Method for Synthesis of Combinatorial Logic Circuits", *Transactions of AIEE, Communications and Electronics* 72, parte 1, noviembre de 1953, pp. 593-599.
2. R. L. Ashenhurst, "The Decomposition of Switching Functions", Proc. of the Symposium on the Theory of Switching, 1957, *Vol. 29 of Annals of Computation Laboratory* (Harvard University: Cambridge, MA, 1959), pp. 74-116.
3. F. J. Hill y G. R. Peterson, *Computer Aided Logical Design with Emphasis on VLSI*, 4a. ed. (Wiley: Nueva York, 1993).
4. T. Sasao, *Logic Synthesis and Optimization* (Kluwer: Boston, MA, 1993).
5. S. Devadas, A. Gosh y K. Keutzer, *Logic Synthesis* (McGraw-Hill: Nueva York, 1994).
6. W. V. Quine, "The Problem of Simplifying Truth Functions", *Amer. Math. Monthly* 59 (1952), pp. 521-531.
7. E. J. McCluskey Jr., "Minimization of Boolean Functions", *Bell System Tech. Journal*, noviembre de 1956, pp. 1417-1444.
8. E. J. McCluskey, *Logic Design Principles* (Prentice-Hall: Englewood Cliffs, NJ, 1986).
9. J. F. Wakerly, *Digital Design Principles and Practices*, 3a. ed. (Prentice-Hall: Englewood Cliffs, NJ, 1999).
10. J. P. Hayes, *Introduction to Logic Design* (Addison-Wesley: Reading, MA, 1993).
11. C. H. Roth Jr., *Fundamentals of Logic Design*, 4a. ed. (West: St. Paul, MN, 1993).
12. R. H. Katz, *Contemporary Logic Design* (Benjamin/Cummings: Redwood City, CA, 1994).
13. V. P. Nelson, H. T. Nagle, B. D. Carroll y J. D. Irwin, *Digital Logic Circuit Analysis and Design* (Prentice-Hall: Englewood Cliffs, NJ, 1995).
14. J. P. Daniels, *Digital Design from Zero to One* (Wiley: Nueva York, 1996).
15. P. K. Lala, *Practical Digital Logic Design and Testing* (Prentice-Hall: Englewood Cliffs, NJ, 1996).
16. A. Dewey, *Analysis and Design of Digital Systems with VHDL* (PWS Publishing Co.: Boston, MA, 1997).
17. M. M. Mano, *Digital Design*, 3a. ed. (Prentice-Hall: Upper Saddle River, NJ, 2001).

18. D. D. Gajski, *Principles of Digital Design* (Prentice-Hall: Upper Saddle River, NJ, 1997).

19. R. K. Brayton, G. D. Hachtel, C. T. McMullen y A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis* (Kluwer: Boston, MA, 1984).

20. R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli y A. R. Wang, "MIS: A Multiple-Level Logic Synthesis Optimization System", *IEEE Transactions on Computer-Aided Design*, CAD-6, noviembre de 1987, pp. 1062-1081.

21. E. M. Sentovic, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton y A. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis", Reporte técnico UCB/ERL M92/41, Laboratorio de Investigación en Electrónica, Departamento de Ingeniería Eléctrica y Ciencias de la Computación, Universidad de California.

22. G. De Micheli, *Synthesis and Optimization of Digital Circuits* (McGraw-Hill: Nueva York, 1994).

23. N. Sherwani, *Algorithms for VLSI Physical Design Automation* (Kluwer: Boston, MA, 1995).

24. B. Preas y M. Lorenzetti, *Physical Design Automation of VLSI Systems* (Benjamin/Cummings: Redwood City, CA, 1988).