2 Álgebra booleana y compuertas lógicas

2-1 DEFINICIONES BÁSICAS

El álgebra booleana, al igual que todos los sistemas matemáticos deductivos, se define con un conjunto de elementos, un conjunto de operadores y varios axiomas o postulados no demostrados. Un *conjunto* de elementos es cualquier colección de objetos con alguna propiedad en común. Si S es un conjunto y x y y son ciertos objetos, entonces $x \in S$ denota que x es un miembro del conjunto S, y $y \notin S$ denota que y no es un elemento de S. Un conjunto que tiene un número enumerable de elementos se especifica con llaves: $A = \{1, 2, 3, 4\}$, es decir, los elementos de A son los números 1, 2, 3 y 4. Un *operador binario* definido sobre un conjunto S de elementos es una regla que asigna a cada par de elementos de S un elemento único de S. Por ejemplo, consideremos la relación $a * b = c$. Decimos que $*$ es un operador binario si especifica una regla para encontrar c a partir del par (a, b) y si además $a, b, c \in S$. Sin embargo, $*$ no es un operador binario si $a, b \in S$ y la regla encuentra que $c \notin S$.

Los postulados de un sistema matemático constituyen los supuestos básicos a partir de los cuales es posible deducir las reglas, teoremas y propiedades del sistema. Los postulados más comunes que se utilizan para formular diversas estructuras algebraicas son:

1. *Cerradura*. Un conjunto S es cerrado respecto a un operador binario si, por cada par de elementos de S, el operador especifica una regla para obtener un elemento único de S. Por ejemplo, el conjunto de los números naturales $N = \{1, 2, 3, 4, \dots\}$ es cerrado respecto al operador binario más $(+)$ por las reglas de la suma aritmética, ya que, para cualquier $a, b \in N$, obtenemos un $c \in N$ único por la operación $a + b = c$. El conjunto de los números naturales no es cerrado respecto al operador binario menos $(-)$ por las reglas de la resta aritmética, porque $2 - 3 = -1$ y $2, 3 \in N$ pero $(-1) \notin N$.
2. *Ley asociativa*. Decimos que un operador binario $*$ sobre un conjunto S es asociativo si

$$(x * y) * z = x * (y * z) \text{ para todos } x, y, z \in S$$

3. *Ley conmutativa*. Decimos que un operador binario $*$ sobre un conjunto S es conmutativo si

$$x * y = y * x \text{ para todos } x, y \in S$$

4. *Elemento de identidad*. Decimos que un conjunto S tiene un elemento de identidad respecto a una operación binaria $*$ sobre S si existe un elemento $e \in S$ con la propiedad

$$e * x = x * e = x \text{ para todos } x \in S$$

Ejemplo: El elemento 0 es un elemento de identidad respecto a la operación $+$ sobre el conjunto de los enteros $I = \{..., -3, -2, -1, 0, 1, 2, 3, \ldots\}$, porque

$$x + 0 = 0 + x = x \text{ para cualquier } x \in I$$

El conjunto de los números naturales, N, no tiene elemento de identidad porque 0 no pertenece al conjunto.

5. *Inverso.* Decimos que un conjunto S, que tiene el elemento de identidad e respecto a un operador $*$, tiene un inverso si, para todo $x \in S$, existe un elemento $y \in S$ tal que

$$x * y = e$$

Ejemplo: En el conjunto de enteros, I, donde $e = 0$, el inverso de un elemento a es $(-a)$, ya que $a + (-a) = 0$.

6. *Ley distributiva*. Si $*$ y $\cdot$ son dos operadores binarios sobre un conjunto S, decimos que $*$ es distributivo sobre $\cdot$ si

$$x * (y \cdot z) = (x * y) \cdot (x * z)$$

Un ejemplo de estructura algebraica es un *campo*. Un campo es un conjunto de elementos, junto con dos operadores binarios, cada uno de los cuales posee las propiedades 1 a 5 y, combinados, la propiedad 6. El conjunto de los números reales, junto con los operadores binarios $+$ y $\cdot$, forman el campo de los números reales. Este campo es la base de la aritmética y el álgebra ordinaria. Los operadores y postulados significan lo siguiente:

El operador binario $+$ define la suma.

La identidad aditiva es 0.

El inverso aditivo define la resta.

El operador binario $\cdot$ define la multiplicación.

La identidad multiplicativa es 1.

El inverso multiplicativo de $a = 1/a$ define la división, es decir, $a \cdot 1/a = 1$.

La única ley distributiva válida es la de $\cdot$ sobre $+$:

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$$

2-2 DEFINICIÓN AXIOMÁTICA DEL ÁLGEBRA BOOLEANA

En 1854, George Boole introdujo un tratamiento sistemático de la lógica y desarrolló, con este fin, un sistema algebraico que ahora llamamos *álgebra booleana*. En 1938, C. E. Shannon introdujo un álgebra booleana de dos valores llamada *álgebra de conmutación* y demostró que

las propiedades de los circuitos eléctricos de conmutación biestables podían representarse con esa álgebra. Para definir formalmente el álgebra booleana, utilizaremos los postulados formulados por E. V. Huntington en 1904.

El álgebra booleana es una estructura algebraica definida por un conjunto de elementos, B, junto con dos operadores binarios, $+$ y $\cdot$, a condición de que se satisfagan los postulados siguientes (de Huntington):

1. **a)** Cerradura respecto al operador $+$.
   b) Cerradura respecto al operador $\cdot$.
2. **a)** Un elemento de identidad con respecto a $+$, designado por 0: $x + 0 = 0 + x = x$.
   b) Un elemento de identidad con respecto a $\cdot$, designado por 1: $x \cdot 1 = 1 \cdot x = x$.
3. **a)** Conmutativa respecto a $+$: $x + y = y + x$.
   b) Conmutativa respecto a $\cdot$: $x \cdot y = y \cdot x$.
4. **a)** $\cdot$ es distributivo sobre $+$: $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$.
   b) $+$ es distributivo sobre $\cdot$: $x + (y \cdot z) = (x + y) \cdot (x + z)$.
5. Para cada elemento $x \in B$, existe un elemento $x' \in B$ (llamado complemento de x) tal que a) $x + x' = 1$ y b) $x \cdot x' = 0$.
6. Existen por lo menos dos elementos $x, y \in B$ tales que $x \neq y$.

Al comparar el álgebra booleana con la aritmética y el álgebra ordinaria (el campo de los números reales), observamos las siguientes diferencias:

1. Los postulados de Huntington no incluyen la ley asociativa. Sin embargo, esta ley se cumple para el álgebra booleana y se puede derivar (para ambos operadores) de los otros postulados.
2. La ley distributiva de $+$ sobre $\cdot$, es decir, $x + (y \cdot z) = (x + y) \cdot (x + z)$, es válida para el álgebra booleana, pero no para el álgebra ordinaria.
3. El álgebra booleana no tiene inversos aditivos ni multiplicativos; por tanto, no hay operaciones de resta ni de división.
4. El postulado 5 define un operador llamado *complemento* que no existe en el álgebra ordinaria.
5. El álgebra ordinaria se ocupa de los números reales, que constituyen un conjunto infinito de elementos. El álgebra booleana se ocupa de un conjunto de elementos B que todavía no hemos definido, pero en el álgebra booleana de dos valores que definiremos a continuación (y que nos interesará para nuestro uso subsecuente de esta álgebra), B se define como un conjunto con sólo dos elementos, 0 y 1.

El álgebra booleana se parece al álgebra ordinaria en algunos sentidos. La selección de los símbolos $+$ y $\cdot$ es intencional para que quienes ya conocen el álgebra ordinaria puedan efectuar con más facilidad manipulaciones algebraicas booleanas. Aunque podemos aprovechar algunos conocimientos del álgebra ordinaria para trabajar con el álgebra booleana, el principiante debe tener cuidado de no usar las reglas del álgebra ordinaria en casos en que no son válidas.

Es importante distinguir entre los elementos del conjunto de una estructura algebraica y las variables de un sistema algebraico. Por ejemplo, los elementos del campo de los números reales son números, mientras que variables como a, b, c, etcétera, que se usan en el álgebra ordinaria, son símbolos que representan números reales. Asimismo, en el álgebra booleana, definimos los elementos del conjunto B, y las variables como x, y y z son meramente símbolos que representan a los elementos. A estas alturas, es importante entender que, para tener un álgebra booleana, es preciso mostrar:

1. los elementos del conjunto B,
2. las reglas de operación de los dos operadores binarios, y
3. que el conjunto de elementos, B, junto con los dos operadores, satisface los seis postulados de Huntington.

Podemos formular muchas álgebras booleanas, dependiendo de los elementos que escojamos para B y de las reglas de operación. De aquí en adelante, nos ocuparemos únicamente de un álgebra booleana de dos valores, es decir, una que sólo tiene dos elementos. El álgebra booleana de dos valores tiene aplicaciones en teoría de conjuntos (el álgebra de clases) y en la lógica proposicional. Lo que nos interesa aquí es la aplicación del álgebra booleana a los circuitos tipo compuerta.

Álgebra booleana de dos valores

Un álgebra booleana de dos valores se define sobre un conjunto de dos elementos, $B = \{1, 0\}$, con las reglas para los dos operadores binarios, $+$ y $\cdot$, que se muestran en las siguientes tablas de operador (la regla del operador de complemento es para verificar el postulado 5):

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

x	x'
0	1
1	0

Estas reglas son exactamente las mismas que las de las operaciones AND, OR y NOT, respectivamente, definidas en la tabla 1-8. Ahora debemos demostrar que los postulados de Huntington son válidos para el conjunto $B = \{0, 1\}$ y los dos operadores binarios que hemos definido.

1. La *cerradura* es obvia por las tablas, pues el resultado de todas las operaciones es 1 o bien 0, y $1, 0 \in B$.
2. En las tablas vemos que
   - **a)** $0 + 0 = 0$ $\qquad$ $0 + 1 = 1 + 0 = 1;$
   - **b)** $1 \cdot 1 = 1$ $\qquad$ $1 \cdot 0 = 0 \cdot 1 = 0.$

   Esto establece los dos *elementos de identidad*, 0 para $+$ y 1 para $\cdot$, definidos por el postulado 2.
3. Las leyes *conmutativas* son obvias por la simetría de las tablas de los operadores binarios.
4. **a)** Es posible demostrar que la ley *distributiva* $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$ se cumple preparando una tabla de verdad con todos los posibles valores de x, y y z. Para cada combinación, deducimos $x \cdot (y + z)$ y demostramos que su valor es igual al de $(x \cdot y) + (x \cdot z)$.

x	y	z	$y + z$	$x \cdot (y + z)$	$x \cdot y$	$x \cdot z$	$(x \cdot y) + (x \cdot z)$
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

b) Podemos demostrar que se cumple la ley *distributiva* de $+$ sobre $\cdot$ preparando una tabla de verdad similar a la anterior.

5. La tabla del complemento permite demostrar fácilmente que
a) $x + x' = 1$, ya que $0 + 0' = 0 + 1 = 1$ y $1 + 1' = 1 + 0 = 1$.
b) $x \cdot x' = 0$, ya que $0 \cdot 0' = 0 \cdot 1 = 0$ y $1 \cdot 1' = 1 \cdot 0 = 0$, lo que verifica el postulado 5.

6. El postulado 6 se satisface porque el álgebra booleana de dos valores tiene dos elementos, 1 y 0, y $1 \neq 0$.

Acabamos de establecer un álgebra booleana de dos valores que tiene un conjunto de dos elementos, 1 y 0, dos operadores binarios con reglas de operación equivalentes a las operaciones AND y OR, y un operador de complemento equivalente al operador NOT. Así pues, el álgebra booleana ha quedado definida de manera matemática formal y se ha demostrado que es equivalente a la lógica binaria que presentamos heurísticamente en la sección 1-9. La presentación heurística ayuda a entender la aplicación del álgebra booleana a los circuitos tipo compuerta. La presentación formal es necesaria para desarrollar los teoremas y propiedades del sistema algebraico. Los ingenieros también llaman "álgebra de conmutación" al álgebra booleana de dos valores que definimos en esta sección. A fin de hacer hincapié en las similitudes entre el álgebra booleana de dos valores y otros sistemas binarios, llamamos "lógica binaria" a esta álgebra en la sección 1-9. De aquí en adelante, omitiremos el calificativo "de dos valores" al hablar de álgebra booleana en las explicaciones.

2-3 TEOREMAS Y PROPIEDADES BÁSICOS DEL ÁLGEBRA BOOLEANA

Dualidad

Se han presentado los postulados de Huntington por pares y se han designado como parte a) y parte b). Es posible obtener una parte de la otra si se intercambian los operadores binarios y los elementos de identidad. Esta importante propiedad del álgebra booleana se denomina *principio de dualidad*, y establece que toda expresión algebraica que pueda deducirse de los postulados del álgebra booleana seguirá siendo válida si se intercambian los operadores y los elementos de identidad. En un álgebra booleana de dos valores, los elementos de identidad y

los elementos del conjunto, B, son los mismos: 1 y 0. El principio de dualidad tiene muchas aplicaciones. Si queremos el *dual* de una expresión algebraica, simplemente intercambiamos los operadores OR y AND y sustituimos los unos por ceros y los ceros por unos.

Teoremas básicos

En la tabla 2-1 se presentan seis teoremas del álgebra booleana y cuatro de sus postulados. La notación se simplifica omitiendo el operador binario · en los casos en que ello no causa confusión. Los teoremas y postulados que se presentan son las relaciones más básicas del álgebra booleana. Los teoremas, al igual que los postulados, se presentan por pares; cada relación es el dual de su pareja. Los postulados son axiomas básicos de la estructura algebraica y no requieren demostración. Los teoremas deben demostrarse a partir de los postulados. A continuación se presentan las demostraciones de los teoremas con una variable. A la derecha se indica el número del postulado que justifica cada paso de la demostración.

TEOREMA 1a): $x + x = x$.

$$
\begin{aligned}
x + x &= (x + x) \cdot 1 && \text{por el postulado: 2b)} \\
&= (x + x)(x + x') && \text{5a)} \\
&= x + xx' && \text{4b)} \\
&= x + 0 && \text{5b)} \\
&= x && \text{2a)}
\end{aligned}
$$

TEOREMA 1b): $x \cdot x = x$.

$$
\begin{aligned}
x \cdot x &= xx + 0 && \text{por el postulado: 2a)} \\
&= xx + xx' && \text{5b)} \\
&= x(x + x') && \text{4a)} \\
&= x \cdot 1 && \text{5a)} \\
&= x && \text{2b)}
\end{aligned}
$$

Tabla 2-1
Postulados y teoremas del álgebra booleana

Postulado 2	a)	$x + 0 = x$	b)	$x \cdot 1 = x$
Postulado 5	a)	$x + x' = 1$	b)	$x \cdot x' = 0$
Teorema 1	a)	$x + x = x$	b)	$x \cdot x = x$
Teorema 2	a)	$x + 1 = 1$	b)	$x \cdot 0 = 0$
Teorema 3, involución		$(x')' = x$		
Postulado 3, conmutatividad	a)	$x + y = y + x$	b)	$xy = yx$
Teorema 4, asociatividad	a)	$x + (y + z) = (x + y) + z$	b)	$x(yz) = (xy)z$
Postulado 4, distributividad	a)	$x(y + z) = xy + xz$	b)	$x + yz = (x + y)(x + z)$
Teorema 5, DeMorgan	a)	$(x + y)' = x'y'$	b)	$(xy)' = x' + y'$
Teorema 6, absorción	a)	$x + xy = x$	b)	$x(x + y) = x$

Observe que el teorema 1b) es el dual del teorema 1a) y que cada uno de los pasos de la demostración en la parte b) es el dual de la parte a). Cualquier teorema dual se puede deducir de forma similar, partiendo de la demostración de su par correspondiente.

TEOREMA 2a): $x + 1 = 1$.

$x + 1 = 1 \cdot (x + 1)$	por el postulado:	2b)
$= (x + x')(x + 1)$		5a)
$= x + x' \cdot 1$		4b)
$= x + x'$		2b)
$= 1$		5a)

TEOREMA 2b): $x \cdot 0 = 0$ por dualidad.

TEOREMA 3: $(x')' = x$. Del postulado 5, tenemos $x + x' = 1$ y $x \cdot x' = 0$, lo que define al complemento de x. El complemento de x' es x y también es $(x')'$. Por tanto, dado que el complemento es único, tenemos que $(x')' = x$.

Los teoremas en los que intervienen dos o tres variables se pueden demostrar algebraicamente a partir de los postulados y los teoremas que ya demostramos. Tomemos como ejemplo el teorema de absorción.

TEOREMA 6a): $x + xy = x$.

$x + xy = x \cdot 1 + xy$	por el postulado:	2b)
$= x(1 + y)$		4a)
$= x(y + 1)$		3a)
$= x \cdot 1$		2a)
$= x$		2b)

TEOREMA 6b): $x(x + y) = x$ por dualidad.

Es posible demostrar los teoremas del álgebra booleana utilizando tablas de verdad. En esas tablas, se verifica que ambos miembros de la relación den resultados idénticos para todas las posibles combinaciones de las variables que intervienen. La siguiente tabla de verdad verifica el primer teorema de absorción.

x	y	xy	$x + xy$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Las demostraciones algebraicas de la ley asociativa y del teorema de DeMorgan son largas y no se presentarán aquí. Sin embargo, es fácil mostrar su validez con tablas de verdad. Por ejemplo, he aquí la tabla de verdad del primer teorema de DeMorgan $(x + y)' = x'y'$.

x	y	$x + y$	$(x + y)'$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

x'	y'	$x'y'$
1	1	1
1	0	0
0	1	0
0	0	0

Precedencia de operadores

La precedencia de operadores para evaluar expresiones booleanas es 1) paréntesis, 2) NOT, 3) AND y 4) OR. Dicho de otro modo, la expresión encerrada en paréntesis se debe evaluar antes que todas las demás operaciones. La siguiente operación que tiene precedencia es el complemento, seguida del AND y por último el OR. Por ejemplo, consideremos la tabla de verdad para el teorema de DeMorgan. El miembro izquierdo de la expresión es $(x + y)'$. Por tanto, la expresión dentro de los paréntesis se evalúa primero y luego se complementa el resultado. El miembro derecho de la expresión es $x'y'$. Por tanto, se evalúan primero el complemento de x y el complemento de y, y luego se obtiene el AND del resultado. Cabe señalar que rige la misma precedencia en la aritmética ordinaria (excepto el complemento) si sustituimos la multiplicación y la suma por el AND y el OR, respectivamente.

2-4 FUNCIONES BOOLEANAS

El álgebra booleana es un álgebra que se ocupa de variables binarias y operaciones lógicas. Una función booleana descrita por una expresión algebraica consta de variables binarias, las constantes 0 y 1, y los símbolos lógicos de operación. Para un valor dado de las variables binarias, la función puede ser igual a 1 o bien a 0. Considere por ejemplo esta función booleana:

$$F_1 = x + y'z$$

La función F_1 es igual a 1 si x es igual a 1 o si tanto y′ como z son iguales a 1. F_1 es igual a 0 en todos los demás casos. La operación de complemento hace que si $y' = 1$, entonces $y = 0$. Por tanto, podemos decir que $F_1 = 1$ si $x = 1$ o si $y = 0$ y $z = 1$. Una función booleana expresa la relación lógica entre variables binarias. Se evalúa determinando el valor binario de la expresión para todos los posibles valores de las variables.

Podemos representar una función booleana en una tabla de verdad. Una tabla de verdad es una lista de combinaciones de unos y ceros asignados a las variables binarias y una columna que muestra el valor de la función para cada combinación binaria. El número de filas de la tabla es 2^n, donde n es el número de variables de la función. Las combinaciones binarias para la tabla de verdad se obtienen de los números binarios, contando de 0 hasta $2^n - 1$. La tabla 2-2 muestra la tabla de verdad para la función F_1. Hay ocho posibles combinaciones binarias para asignar bits a las tres variables x, y y z. La columna rotulada F_1 contiene 0 o 1 para cada una

Tabla 2-2
Tablas de verdad para F_1 y F_2

x	y	z	F_1	F_2
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	0
1	1	1	1	0

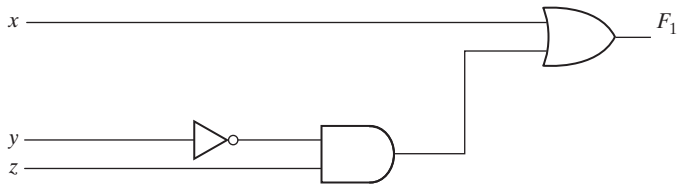


FIGURA 2-1
Implementación de $F_1 = x + y'z$ con compuertas

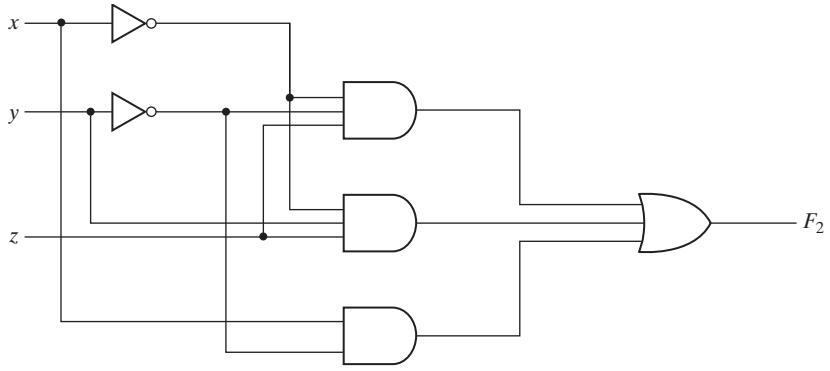
de esas combinaciones. La tabla indica que la función es igual a 1 cuando $x = 1$ o cuando $yz = 01$. En los demás casos, es igual a 0.

Una función booleana se puede transformar de una expresión algebraica a un diagrama de circuitos hecho con compuertas lógicas. En la figura 2-1 se presenta el diagrama de circuito lógico para F_1. Hay un inversor para generar el complemento de la entrada y, una compuerta AND para el término $y'z$ y una compuerta OR que combina los dos términos. En los diagramas de circuitos lógicos, las variables de la función son las entradas del circuito, y la variable binaria F_1 es la salida del circuito.

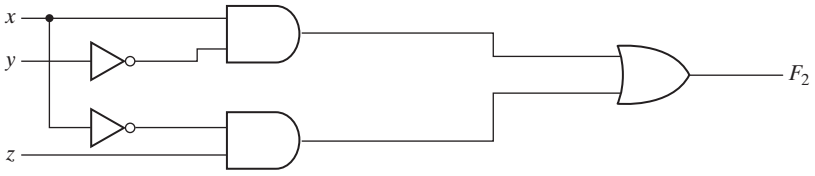
Sólo hay una forma de representar una función booleana en una tabla de verdad. En cambio, cuando la función está en forma algebraica, puede expresarse de varias maneras. La expresión específica empleada para designar la función también determinará la interconexión de compuertas en el diagrama de circuito lógico. Manipulando una expresión booleana según las reglas del álgebra booleana, a veces es posible obtener una expresión más simple para la misma función y así reducir el número de compuertas del circuito y el número de entradas de las compuertas. Consideremos, por ejemplo, esta función booleana:

$$F_2 = x'y'z + x'yz + xy'$$

La implementación de esta función con compuertas lógicas se muestra en la figura 2-2a). Las variables de entrada x y y se complementan con inversores para obtener x' y y'. Los tres términos de la expresión se implementan con tres compuertas AND. La compuerta OR forma el OR lógico de los tres términos. La tabla de verdad para F_2 se presenta en la tabla 2-2. La fun-



a) $F_2 = x'y'z + x'yz + xy'$



b) $F_2 = xy' + x'z$

FIGURA 2-2
Implementación de la función booleana F_2 con compuertas

ción es igual a 1 cuando $xyz = 001$ o 011, o cuando $xy = 10$ (sin importar el valor de z); en los demás casos es igual a 0. Esto produce cuatro unos y cuatro ceros para F_2.

Consideremos ahora la posible simplificación de la función aplicando algunas de las identidades del álgebra booleana:

$$F_2 = x'y'z + x'yz + xy' = x'z(y' + y) + xy' = x'z + xy'$$

La función se reduce a únicamente dos términos y se puede implementar con compuertas como se indica en la figura 2-2b). Es evidente que el circuito de b) es más sencillo que el de a), aunque ambos realizan la misma función. Podemos usar una tabla de verdad para comprobar que las dos expresiones son equivalentes. La expresión simplificada es igual a 1 cuando $xz = 01$ o cuando $xy = 10$. Esto produce los mismos cuatro unos en la tabla de verdad. Puesto que ambas expresiones producen la misma tabla de verdad, decimos que son equivalentes. Por tanto, los dos circuitos tienen las mismas salidas para todas las posibles combinaciones binarias de las tres variables de entrada. Ambas realizan la misma función, pero la que tiene menos compuertas y menos entradas a las compuertas es preferible porque requiere menos alambres y componentes.

Manipulación algebraica

Cuando se implementa una expresión booleana con compuertas lógicas, cada término requiere una compuerta y cada variable dentro del término implica una entrada a la compuerta. Definimos una *literal* como una sola variable dentro de un término, que podría estar complementa-

da o no. La función de la figura 2-2a) tiene tres términos y ocho literales; la de la figura 2-2b) tiene dos términos y cuatro literales. Si reducimos el número de términos, el número de literales, o ambas cosas, en una expresión booleana, podría obtenerse un circuito más sencillo. La manipulación del álgebra booleana consiste en su mayor parte en reducir una expresión con objeto de obtener un circuito más simple. Las funciones de hasta cinco variables se pueden simplificar con el método de mapa que se describirá en el capítulo siguiente. En el caso de funciones booleanas complejas, los diseñadores digitales utilizan programas de minimización computarizados. El único método manual con que se cuenta es un procedimiento de recortes y ensayos que utiliza las relaciones básicas y otras técnicas de manipulación que con el uso se vuelven familiares. Los ejemplos que siguen ilustran la manipulación algebraica del álgebra booleana.

EJEMPLO 2-1

Simplifique las funciones booleanas siguientes al número mínimo de literales.

1. $x(x' + y) = xx' + xy = 0 + xy = xy.$

2. $x + x'y = (x + x')(x + y) = 1(x + y) = x + y.$

3. $(x + y)(x + y') = x + xy + xy' + yy' = x(1 + y + y') = x.$

4.
$$\begin{aligned} xy + x'z + yz &= xy + x'z + yz(x + x') \\ &= xy + x'z + xyz + x'yz \\ &= xy(1 + z) + x'z(1 + y) \\ &= xy + x'z. \end{aligned}$$

5. $(x + y)(x' + z)(y + z) = (x + y)(x' + z)$ por dualidad de la función 4.

Las funciones 1 y 2 son una la dual de la otra y utilizan expresiones duales en pasos correspondientes. Una forma más sencilla de simplificar la función 3 es con el postulado 4b) de la tabla 2-1: $(x + y)(x + y') = x + yy' = x$. La cuarta función ilustra el hecho de que un aumento en el número de literales a veces da pie a una expresión final más simple. La función 5 no se minimiza directamente, pero puede deducirse del dual de los pasos empleados para deducir la función 4. Las funciones 4 y 5 llevan el nombre de *teorema de consenso*.

Complemento de una función

El complemento de una función F es F' y se obtiene intercambiando ceros por unos y unos por ceros en el valor de F. El complemento de una función podría deducirse algebraicamente empleando el teorema de DeMorgan. Este par de teoremas se presenta en la tabla 2-1 para dos variables. Los teoremas de DeMorgan se pueden extender a tres o más variables. La forma de tres variables del primer teorema de DeMorgan se deduce como sigue, utilizando postulados y teoremas de la tabla 2-1:

$$\begin{aligned} (A + B + C)' &= (A + x)' && \text{sea } B + C = x \\ &= A'x' && \text{por el teorema 5a) (DeMorgan)} \\ &= A'(B + C)' && \text{sustituimos } B + C = x \\ &= A'(B'C') && \text{por el teorema 5a) (DeMorgan)} \\ &= A'B'C' && \text{por el teorema 4b) (asociatividad)} \end{aligned}$$

Los teoremas de DeMorgan para cualquier número de variables tienen una forma similar al caso de dos variables y se deducen por sustituciones sucesivas como se hizo en la deducción anterior. Estos teoremas se pueden generalizar así:

$$(A + B + C + D + \ldots + F)' = A'B'C'D' \ldots F'$$
$$(ABCD \ldots F)' = A' + B' + C' + D' + \ldots + F'$$

La forma generalizada del teorema de DeMorgan dice que el complemento de una función se obtiene intercambiando operadores AND y OR y complementando cada literal.

EJEMPLO 2-2

Halle el complemento de las funciones $F_1 = x'yz' + x'y'z$ y $F_2 = x(y'z' + yz)$. Aplicando el teorema de DeMorgan tantas veces como sea necesario, se obtienen los complementos como sigue:

$$F_1' = (x'yz' + x'y'z)' = (x'yz')'(x'y'z)' = (x + y' + z)(x + y + z')$$
$$F_2' = \left[x(y'z' + yz)\right]' = x' + (y'z' + yz)' = x' + (y'z')'(yz)'$$
$$= x' + (y + z)(y' + z')$$

Un procedimiento más sencillo para deducir el complemento de una función consiste en obtener el dual de la función y complementar cada literal. Este método es consecuencia del teorema generalizado de DeMorgan. Recuerde que el dual de una función se obtiene intercambiando operadores AND y OR, y unos y ceros.

EJEMPLO 2-3

Determine el complemento de las funciones F_1 y F_2 del ejemplo 2-2 obteniendo sus duales y complementando cada literal.

1. $F_1 = x'yz' + x'y'z$.
El dual de F_1 es $(x' + y + z')(x' + y' + z)$.
Complementamos cada literal: $(x + y' + z)(x + y + z') = F_1'$.

2. $F_2 = x(y'z' + yz)$.
El dual de F_2 es $x + (y' + z')(y + z)$.
Complementamos cada literal: $x' + (y + z)(y' + z') = F_2'$.

2-5 FORMAS CANÓNICAS Y ESTÁNDAR

Minitérminos y maxitérminos

Una variable binaria podría aparecer en su forma normal (x) o en su forma complementada (x'). Considere ahora dos variables binarias x y y que se combinan con una operación AND. Puesto que cada variable podría aparecer en cualquiera de sus dos formas, hay cuatro combinaciones posibles: $x'y'$, $x'y$, xy' y xy. Cada uno de estos cuatro términos AND es un *minitérmi-*

Tabla 2-3
Minitérminos y maxitérminos para tres variables binarias

			Minitérminos		Maxitérminos	
x	*y*	*z*	Términos	Designación	Términos	Designación
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

no, o *producto estándar*. De manera similar, podemos combinar n variables para formar 2^n minitérminos. Éstos podrían obtenerse con un método similar al que se muestra en la tabla 2-3 para tres variables. Se enumeran los números binarios de 0 a $2^n - 1$ bajo las n variables. Cada minitérmino se obtiene de un término AND de las n variables, poniendo un apóstrofo a cada variable si el bit correspondiente del número binario es un 0 y sin apóstrofo si es un 1. En la tabla también se muestra un símbolo para cada minitérmino, el cual tiene la forma m_j, donde j denota el equivalente decimal del número binario del minitérmino designado.

Asimismo, n variables que forman un término OR, donde cada variable puede tener apóstrofo o no, dan pie a 2^n posibles combinaciones, llamadas *maxitérminos* o *sumas estándar*. En la tabla 2-3 se presentan los ocho maxitérminos de tres variables, junto con su designación simbólica. Podemos obtener de manera similar cualesquier 2^n maxitérminos para n variables. Cada maxitérmino se obtiene de un término OR de las n variables, donde cada variable lleva un apóstrofo si el bit correspondiente del número binario es 1. Cabe señalar que cada maxitérmino es el complemento de su minitérmino correspondiente, y viceversa.

Se puede expresar algebraicamente una función booleana a partir de una tabla de verdad dada formando un minitérmino para cada combinación de las variables que produce un 1 en la función, y formando después el OR de todos esos términos. Por ejemplo, la función f_1 de la

Tabla 2-4
Funciones de tres variables

x	*y*	*z*	Función f_1	Función f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

tabla 2-4 se obtiene expresando las combinaciones 001, 100 y 111 como $x'y'z$, $xy'z'$ y xyz, respectivamente. Puesto que cada uno de estos minitérminos hace que $f_1 = 1$, tenemos

$$f_1 = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7$$

De forma similar, es posible verificar fácilmente que

$$f_2 = x'yz + xy'z + xyz' + xyz = m_3 + m_5 + m_6 + m_7$$

Estos ejemplos ilustran una propiedad importante del álgebra booleana: cualquier función booleana se puede expresar como una suma de minitérminos (donde "suma" se refiere al OR de los términos).

Considere ahora el complemento de una función booleana. Podría leerse de la tabla de verdad formando un minitérmino para cada combinación que produce un 0 en la función, y haciendo después el OR de esos términos. El complemento de f_1 se lee así:

$$f_1' = x'y'z' + x'yz' + x'yz + xy'z + xyz'$$

Si obtenemos el complemento de f_1', obtendremos la función f_1:

$$\begin{aligned} f_1 &= (x + y + z)(x + y' + z)(x' + y + z')(x' + y' + z) \\ &= M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6 \end{aligned}$$

Asimismo, podemos leer la expresión para f_2 de la tabla:

$$\begin{aligned} f_2 &= (x + y + z)(x + y + z')(x + y' + z)(x' + y + z) \\ &= M_0 M_1 M_2 M_4 \end{aligned}$$

Estos ejemplos ilustran una segunda propiedad del álgebra booleana: cualquier función booleana se puede expresar como un producto de maxitérminos (donde "producto" se refiere a hacer el AND de los términos). El procedimiento para obtener el producto de maxitérminos directamente de la tabla de verdad es el siguiente. Se forma un maxitérmino para cada combinación de las variables que produce un 0 en la función y luego se hace el AND de todos esos maxitérminos. Se dice que las funciones booleanas expresadas como suma de minitérminos o producto de maxitérminos están en *forma canónica*.

Suma de minitérminos

Dijimos antes que, para n variables binarias, podemos obtener 2^n minitérminos distintos, y que es posible expresar cualquier función booleana como una suma de minitérminos. Los minitérminos cuya suma define a la función booleana son los que producen los unos de la función en una tabla de verdad. Puesto que la función puede dar 1 o 0 con cada minitérmino, y dado que hay 2^n minitérminos, podemos calcular que el número de funciones que es posible formar con n variables es 2^{2n}. A veces es útil expresar la función booleana en su forma de suma de minitérminos. Si no está ya en esa forma, esto se logra expandiendo primero la expresión a una suma de términos AND. Luego se examina cada término para ver si contiene todas las variables. Si falta una o más variables, se le hace AND con una expresión como $x + x'$, donde x es una de las variables faltantes. El ejemplo que sigue aclarará el procedimiento.

EJEMPLO 2-4

Exprese la función booleana $F = A + B'C$ como suma de minitérminos. La función tiene tres variables, A, B y C. En el primer término, A, faltan dos variables; por tanto:

$$A = A(B + B') = AB + AB'$$

A esta función todavía le falta una variable:

$$\begin{aligned} A &= AB(C + C') + AB'(C + C') \\ &= ABC + ABC' + AB'C + AB'C' \end{aligned}$$

Al segundo término, $B'C$, le falta una variable:

$$B'C = B'C(A + A') = AB'C + A'B'C$$

Al combinar todos los términos, tenemos

$$\begin{aligned} F &= A + B'C \\ &= ABC + ABC' + AB'C + AB'C' + A'B'C \end{aligned}$$

Sin embargo, $AB'C$ aparece dos veces y, según el teorema 1 ($x + x = x$), podemos eliminar uno de ellos. Después de reacomodar los minitérminos en orden ascendente, obtenemos por fin

$$\begin{aligned} F &= A'B'C + AB'C' + AB'C + ABC' + ABC \\ &= m_1 + m_4 + m_5 + m_6 + m_7 \end{aligned}$$

Hay ocasiones en que conviene expresar la función booleana, en su forma de suma de minitérminos, con la siguiente notación abreviada:

$$F(A, B, C) = \sum(1, 4, 5, 6, 7)$$

El símbolo de sumatoria, $\sum$, representa el OR de los términos; los números que le siguen son los minitérminos de la función. Las letras entre paréntesis después de F son una lista de las variables en el orden que se usará al convertir un minitérmino en un término AND.

Otro procedimiento para obtener los minitérminos de una función booleana consiste en deducir la tabla de verdad directamente de la expresión algebraica y luego leer los minitérminos de esa tabla. Considere la función booleana del ejemplo 2-4:

$$F = A + B'C$$

La tabla de verdad que se muestra en la tabla 2-5 se deduce directamente de la expresión algebraica enumerando las ocho combinaciones binarias bajo las variables A, B y C e insertan-

Tabla 2-5
Tabla de verdad para $F = A + B'C$

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

do unos bajo F para las combinaciones en las que $A = 1$ y $BC = 01$. Luego, se lee en la tabla de verdad que los cinco minitérminos de la función son 1, 4, 5, 6 y 7.

Producto de maxitérminos

Cada una de las 2^{2^n} funciones de n variables binarias se puede expresar también como un producto de maxitérminos. Para expresar la función booleana como producto de maxitérminos, primero debe ponerse en formato de términos OR. Esto podría hacerse con la ayuda de la ley distributiva, $x + yz = (x + y)(x + z)$. Luego, se hace el OR de cualquier variable faltante x en cada término OR con xx'. El ejemplo que sigue aclarará el procedimiento.

EJEMPLO 2-5

Exprese la función booleana $F = xy + x'z$ en forma de producto de maxitérminos. Primero, se convierte la función en términos OR empleando la ley distributiva:

$$\begin{aligned} F &= xy + x'z = (xy + x')(xy + z) \\ &= (x + x')(y + x')(x + z)(y + z) \\ &= (x' + y)(x + z)(y + z) \end{aligned}$$

La función tiene tres variables, x, y y z. A cada término OR le falta una variable; por tanto:

$$\begin{aligned} x' + y &= x' + y + zz' = (x' + y + z)(x' + y + z') \\ x + z &= x + z + yy' = (x + y + z)(x + y' + z) \\ y + z &= y + z + xx' = (x + y + z)(x' + y + z) \end{aligned}$$

Después de combinar todos los términos y eliminar los que aparecen más de una vez, se obtiene:

$$\begin{aligned} F &= (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z') \\ &= M_0 M_2 M_4 M_5 \end{aligned}$$

Una forma cómoda de expresar esta función es:

$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

El símbolo de producto, Π, denota el AND de maxitérminos; los números son los maxitérminos de la función.

■

Conversión entre formas canónicas

El complemento de una función expresado como la suma de minitérminos es igual a la suma de los minitérminos que faltan en la función original. Ello se debe a que la función original se expresa con los minitérminos que hacen que la función sea igual a 1, mientras que su complemento es 1 para aquellos minitérminos que hacen que la función original sea 0. Por ejemplo, considere la función

$$F(A, B, C) = \Sigma(1, 4, 5, 6, 7)$$

Su complemento se expresa como

$$F'(A, B, C) = \Sigma(0, 2, 3) = m_0 + m_2 + m_3$$

Ahora bien, si se determina el complemento de F' por el teorema de DeMorgan, se obtiene F en una forma distinta:

$$F = (m_0 + m_2 + m_3)' = m_0' \cdot m_2' \cdot m_3' = M_0 M_2 M_3 = \Pi(0, 2, 3)$$

Esta última conversión es consecuencia de la definición de minitérminos y maxitérminos, como se muestra en la tabla 2-3. En esa tabla es evidente que se cumple la relación

$$m_j' = M_j$$

Es decir, el maxitérmino con subíndice j es el complemento del minitérmino que lleva ese subíndice, y viceversa.

El último ejemplo ilustra la conversión entre una función expresada como suma de minitérminos y su equivalente en producto de maxitérminos. Un argumento similar demuestra que la conversión entre el producto de maxitérminos y la suma de minitérminos es similar. Ahora plantearemos un procedimiento general de conversión. Para convertir de una forma canónica a otra, intercambiamos los símbolos Σ y Π e incluimos en la lista sólo los números que faltaban en la forma original. Para hallar los términos faltantes, debemos recordar que el número total de minitérminos o maxitérminos es 2^n, donde n es el número de variables binarias en la función.

Es posible convertir una función booleana de una expresión algebraica a un producto de maxitérminos utilizando una tabla de verdad y el procedimiento de conversión canónica. Considere, por ejemplo, la expresión booleana

$$F = xy + x'z$$

Primero, obtenemos la tabla de verdad de la función, la cual se muestra en la tabla 2-6. Los unos bajo F en la tabla se determinan a partir de las combinaciones de las variables en las que $xy = 11$ o $xz = 01$. De la tabla de verdad deducimos que los minitérminos de la función son 1, 3, 6 y 7. La función expresada como suma de minitérminos es

$$F(x, y, z) = \Sigma(1, 3, 6, 7)$$

Puesto que en total hay ocho minitérminos o maxitérminos en una función de tres variables, deducimos que los términos faltantes son 0, 2, 4 y 5. La función expresada como producto de maxitérminos es

$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

Ésta es la misma respuesta que obtuvimos en el ejemplo 2-5.

Tabla 2-6
Tabla de verdad para $F = xy + x'z$

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Formas estándar

Las dos formas canónicas del álgebra booleana son formas básicas que se obtienen al leer una función de su tabla de verdad, pero casi nunca son las que tienen el número mínimo de literales, porque cada minitérmino o maxitérmino debe contener, por definición, *todas* las variables, sea complementadas o sin complementar.

Otra forma de expresar funciones booleanas es en forma *estándar*. En esta configuración, los términos que forman la función podrían contener una, dos o cualquier número de literales. Hay dos tipos de formas estándar: la suma de productos y el producto de sumas.

La *suma de productos* es una expresión booleana que contiene términos AND, llamados *términos de producto*, con una o más literales cada uno. La *suma* denota el OR de esos términos. Un ejemplo de función expresada como suma de productos es

$$F_1 = y' + xy + x'yz'$$

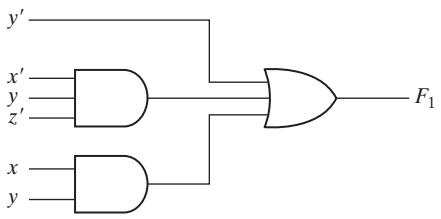
La expresión tiene tres términos de producto con una, dos y tres literales. Su suma es realmente una operación OR.

El diagrama de lógica de una expresión de suma de productos consiste en un grupo de compuertas AND seguidas de una sola compuerta OR. Este patrón de configuración se muestra en la figura 2-3a). Cada término de producto requiere una compuerta AND, salvo los términos que sólo tienen una literal. La suma lógica se forma con una compuerta OR cuyas entradas son las salidas de las compuertas AND y las literales solas. Suponemos que contamos directamente con las variables de entrada en forma de complemento, así que no se han incluido inversores en el diagrama. Esta configuración de circuito se denomina implementación de dos niveles.

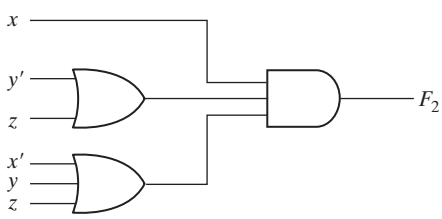
Un *producto de sumas* es una expresión booleana que contiene términos OR, llamados *términos de suma*. Cada término puede tener cualquier cantidad de literales. El *producto* denota el AND de esos términos. Un ejemplo de función expresada como producto de sumas es

$$F_2 = x(y' + z)(x' + y + z)$$

Esta expresión tiene tres términos de suma con una, dos y tres literales. El producto es una operación AND. El uso de las palabras *producto* y *suma* proviene de la similitud entre la operación AND y el producto aritmético (multiplicación), y de la similitud entre la operación OR y la suma aritmética (adición). La estructura de compuertas de la expresión de producto de sumas consiste en un grupo de compuertas OR para los términos de suma (excepto la literal

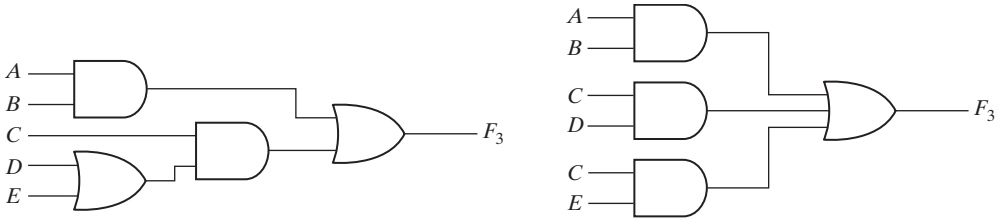


a) Suma de productos



b) Producto de sumas

FIGURA 2-3
Implementación de dos niveles



a) $AB + C(D + E)$ b) $AB + CD + CE$

FIGURA 2-4
Implementación de tres y dos niveles

sola) seguidas de una compuerta AND. Esto se observa en la figura 2-3b). Este tipo estándar de expresión produce una estructura de compuertas de dos niveles.

Las funciones booleanas se pueden expresar en forma no estándar. Por ejemplo, la función

$$F_3 = AB + C(D + E)$$

no es una suma de productos ni un producto de sumas. Su implementación se indica en la figura 2-4a), y requiere dos compuertas AND y dos compuertas OR. Este circuito tiene tres niveles de compuertas, y puede transformarse a una forma estándar utilizando la ley distributiva para eliminar los paréntesis:

$$F_3 = AB + C(D + E) = AB + CD + CE$$

La expresión de suma de productos se implementa en la figura 2-4b). En general, es preferible una implementación de dos niveles porque produce el mínimo de retardo en compuertas cuando la señal se propaga de las entradas a la salida.

2-6 OTRAS OPERACIONES LÓGICAS

Cuando colocamos los operadores binarios AND y OR entre dos variables, x y y, forman dos funciones booleanas, $x \cdot y$ y $x + y$, respectivamente. Se ha señalado ya que hay 2^{2n} funciones para n variables binarias. En el caso de dos variables, $n = 2$, y el número de posibles funciones booleanas es 16. Por tanto, las funciones AND y OR son sólo dos de un total de 16 posibles funciones que se forman con dos variables binarias. Sería interesante encontrar las otras 14 funciones e investigar sus propiedades.

En la tabla 2-7 se presentan las tablas de verdad para las 16 funciones que se forman con dos variables binarias, x y y. Cada una de las 16 columnas, F_0 a F_{15}, representa una tabla de verdad de una posible función de las dos variables x y y. Observe que las funciones se determinan a partir de las 16 combinaciones binarias que se pueden asignar a F. Las 16 funciones se expresan algebraicamente con funciones booleanas, como se indica en la primera columna de la tabla 2-8. Las expresiones booleanas que se incluyen se han simplificado al número mínimo de literales.

Aunque cada función se puede expresar en términos de los operadores booleanos AND, OR y NOT, no hay motivo para no asignar símbolos de operador especiales que expresen las otras funciones. Dichos símbolos aparecen en la segunda columna de la tabla 2-8. Sin embargo, los

3 Minimización en el nivel de compuertas

3-1 EL MÉTODO DEL MAPA

La complejidad de las compuertas de lógica digital que implementan una función booleana está relacionada directamente con la complejidad de la expresión algebraica a partir de la cual se implementa la función. Aunque la representación de una función como tabla de verdad es única, hay muchas formas de expresarla algebraicamente. Las expresiones booleanas se simplifican algebraicamente como se explicó en la sección 2-4, pero este procedimiento de minimización resulta poco práctico porque carece de reglas específicas que predigan cada paso sucesivo del proceso de manipulación. El método del mapa ofrece un procedimiento sencillo y directo para minimizar las funciones booleanas. Este método podría considerarse como una versión pictórica de la tabla de verdad. El método del mapa también se conoce como mapa de Karnaugh o mapa K.

El mapa es un diagrama hecho de cuadrados, cada uno de los cuales representa un minitérmino de la función. Puesto que cualquier función booleana se puede expresar como una suma de minitérminos, toda función booleana se reconocerá gráficamente en el mapa por el área delimitada por los cuadrados cuyos minitérminos están incluidos en la función. De hecho, el mapa presenta un diagrama visual de todas las maneras en que una función se puede expresar en forma estándar. Al reconocer diversos patrones, el usuario puede deducir expresiones algebraicas alternas para la misma función, y luego escoger la más simple.

Las expresiones simplificadas generadas por el mapa siempre están en una de las dos formas estándar: suma de productos o producto de sumas. Supondremos que la expresión algebraica más simple es la que tiene menos términos y el mínimo posible de literales en cada término. Esto produce un diagrama de circuito con el mínimo de compuertas y el mínimo de entradas a cada compuerta. Más adelante se verá que la expresión más simple no es única. Hay ocasiones en que es posible encontrar dos o más expresiones que satisfagan los criterios de minimización. En esos casos, cualquiera de las soluciones es satisfactoria.

Mapa de dos variables

En la figura 3-1a) se presenta el mapa de dos variables. Hay cuatro minitérminos para dos variables; por tanto, el mapa consiste en cuatro cuadrados, uno para cada minitérmino. En b) se ha redibujado el mapa de modo que muestre la relación entre los cuadrados y las dos variables x y y. El 0 y el 1 que se marcan en cada fila y columna indican los valores de las variables. La variable x aparece con apóstrofo en la fila 0 y sin apóstrofo en la fila 1. De forma similar, y aparece con apóstrofo en la columna 0 y sin él en la columna 1.

Si marcamos los cuadrados cuyos minitérminos pertenecen a una función dada, el mapa de dos variables se convertirá en otra forma útil de representar cualquiera de las 16 funciones booleanas de dos variables. Como ejemplo, hemos mostrado la función xy en la figura 3-2a). Puesto que xy es igual a m_3, se coloca un 1 dentro del cuadrado que pertenece a m_3. Asimismo, la función $x + y$ se representa en el mapa de la figura 3-2b) con tres cuadrados marcados con unos. Esos cuadrados se obtienen de los minitérminos de la función:

$$m_1 + m_2 + m_3 = x'y + xy' + xy = x + y$$

Los tres cuadrados también podrían haberse deducido de la intersección de la variable x en la segunda fila y la variable y en la segunda columna, que encierra el área perteneciente a x o y.

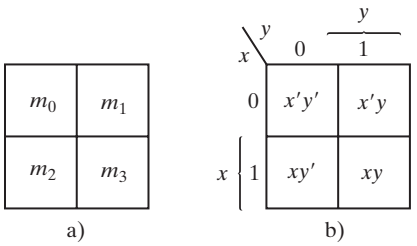


FIGURA 3-1
Mapa de dos variables

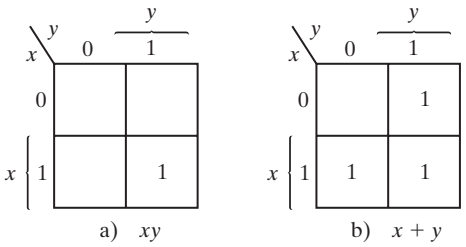


FIGURA 3-2
Representación de funciones en el mapa

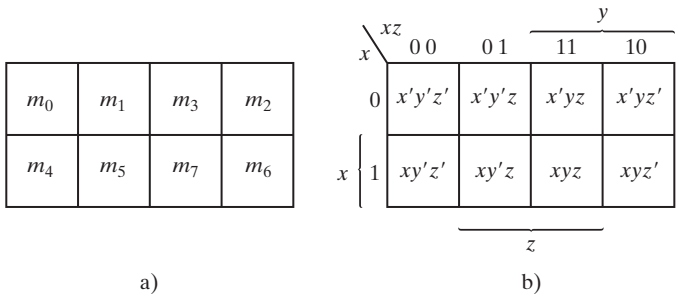


FIGURA 3-3
Mapa de tres variables

Mapa de tres variables

En la figura 3-3 se muestra un mapa de tres variables. Hay ocho minitérminos para tres variables binarias; por tanto, el mapa consta de ocho cuadrados. Advierta que los minitérminos no están acomodados en sucesión binaria, sino en una sucesión similar al código Gray (tabla 1-4). La característica de esta sucesión es que sólo un bit cambia de valor entre dos columnas adyacentes. El mapa dibujado en la parte b) se ha marcado con números en cada fila y columna que indican la relación entre los cuadrados y las tres variables. Por ejemplo, el cuadrado asignado a m_5 corresponde a la fila 1 y la columna 01. Si concatenamos estos dos números, obtendremos el número binario 101, cuyo equivalente decimal es 5. Otra forma de ver el cuadrado $m_5 = xy'z$ es considerarlo como ubicado en la fila marcada x y la columna perteneciente a $y'z$ (columna 01). Observe que hay cuatro cuadrados en los que cada variable es 1 y cuatro en los que es 0. La variable aparece sin apóstrofo en los cuatro cuadrados en los que es 1, y con apóstrofo en los que es 0. Por conveniencia, se escribe la variable con su símbolo alfabético bajo los cuatro cuadrados en los que no lleva apóstrofo.

Para entender la utilidad del mapa en la simplificación de funciones booleanas, hay que entender la propiedad básica que poseen dos cuadrados adyacentes. Cualesquier dos cuadrados adyacentes del mapa difieren en una sola variable, que tiene apóstrofo en un cuadrado y no lo tiene en el otro. Por ejemplo, m_5 y m_7 están en dos cuadrados adyacentes. La variable y tiene apóstrofo en m_5 y carece de apóstrofo en m_7, mientras que las otras dos variables son iguales en ambos cuadrados. De los postulados del álgebra booleana, se sigue que la suma de dos minitérminos en cuadrados adyacentes se simplifica a un solo término AND que consiste en sólo dos literales. Para aclarar esto, consideremos la suma de dos cuadrados adyacentes como m_5 y m_7:

$$m_5 + m_7 = xy'z + xyz = xz(y' + y) = xz$$

Aquí, los dos cuadrados difieren en la variable y, que se elimina al formarse la suma de los dos minitérminos. Así, cualesquier dos minitérminos en cuadrados adyacentes a los que se aplique un OR permitirán eliminar la variable diferente. Los ejemplos que siguen explican el procedimiento para minimizar una función booleana con un mapa.

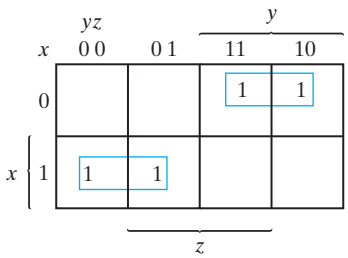


FIGURA 3-4
Mapa para el ejemplo 3-1; $F(x, y, z) = \sum(2, 3, 4, 5) = x'y + xy'$

EJEMPLO 3-1

Simplifique la función booleana

$$F(x, y, z) = \sum(2, 3, 4, 5)$$

Primero, marcamos con un 1 cada uno de los minitérminos que representan a la función. Esto se indica en la figura 3-4, donde se han marcado con 1 los cuadrados correspondientes a los minitérminos 010, 011, 100 y 101. El siguiente paso es encontrar los posibles cuadrados adyacentes, que se indican en el mapa con dos rectángulos, cada uno de los cuales encierra dos unos. El rectángulo de arriba a la derecha representa el área delimitada por $x'y$. Esto se determina observando que el área de dos cuadrados está en la fila 0, que corresponde a x', y las últimas dos columnas, que corresponden a y. De forma similar, el rectángulo inferior izquierdo representa el término de producto xy'. (La segunda fila representa a x y las dos columnas de la izquierda representan a y'.) La suma lógica de estos dos términos producto da la expresión simplificada:

$$F = x'y + xy'$$

Hay casos en los que se considera que dos cuadrados del mapa están adyacentes, aunque no se estén tocando. En la figura 3-3, m_0 es adyacente a m_2 y m_4 adyacente a m_6 porque los minitérminos difieren en una sola variable. Esto se puede verificar fácilmente con álgebra:

$$m_0 + m_2 = x'y'z' + x'yz' = x'z'(y' + y) = x'z'$$
$$m_4 + m_6 = xy'z' + xyz' = xz' + (y' + y) = xz'$$

Por tanto, deberemos modificar la definición de cuadrados adyacentes de modo que incluya este caso y otros similares. Esto se hace considerando que el mapa está dibujado en una superficie cuyos bordes derecho e izquierdo están en contacto para formar cuadrados adyacentes.

EJEMPLO 3-2

Simplifique la función booleana

$$F(x, y, z) = \sum(3, 4, 6, 7)$$

El mapa para esta función se presenta en la figura 3-5. Hay cuatro cuadrados marcados con unos, uno para cada minitérmino de la función. En la tercera columna se combinan dos cuadrados

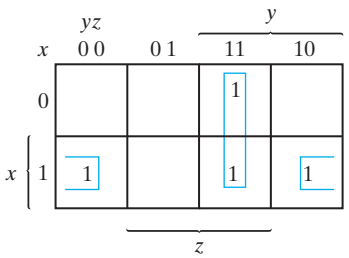


FIGURA 3-5
Mapa para el ejemplo 3-2; $F(x, y, z) = \sum(3, 4, 6, 7) = yz + xz'$

adyacentes para dar un término de dos literales, yz. Los otros dos cuadrados que incluyen unos también son adyacentes según la nueva definición, por lo que en el diagrama sus valores se encierran con medios rectángulos. Estos dos cuadrados combinados dan el término de dos literales xz'. La función simplificada es

$$F = yz + xz'$$

Considere ahora cualquier combinación de cuatro cuadrados adyacentes en el mapa de tres variables. Cualquier combinación así representa la suma lógica de cuatro minitérminos y da como resultado una expresión con una sola literal. Por ejemplo, la suma lógica de los cuatro minitérminos adyacentes 0, 2, 4 y 6 se reduce a un término de una sola literal, z':

$$\begin{aligned} m_0 + m_2 + m_4 + m_6 &= x'y'z' + x'yz' + xy'z' + xyz' \\ &= x'z'(y' + y) + xz'(y' + y) \\ &= x'z' + xz' = z'(x' + x) = z' \end{aligned}$$

El número de cuadrados adyacentes que es posible combinar siempre debe ser una potencia de 2, como 1, 2, 4 y 8. Al aumentar el número de cuadrados adyacentes que se combinan, se reduce el número de literales del término producto obtenido.

Un cuadrado representa un minitérmino, lo que da un término con tres literales.

Dos cuadrados adyacentes representan un término de dos literales.

Cuatro cuadrados adyacentes representan un término de una sola literal.

Ocho cuadrados adyacentes abarcan todo el mapa y producen una función que siempre es igual a 1.

EJEMPLO 3-3

Simplifique la función booleana

$$F(x, y, z) = \sum(0, 2, 4, 5, 6)$$

El mapa de F se muestra en la figura 3-6. Primero, combinamos los cuatro cuadrados adyacentes de la primera y la última columnas para obtener el término de una sola literal z'. El cuadrado restante, que representa el minitérmino 5, se combina con un cuadrado adyacente que ya se usó una vez. Esto no sólo es permisible, sino hasta deseable, porque los dos cuadrados adya-

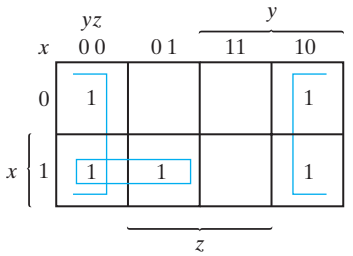


FIGURA 3-6
Mapa para el ejemplo 3-3; $F(x, y, z) = \sum(0, 2, 4, 5, 6) = z' + xy'$

centes dan el término de dos literales xy', mientras que el cuadrado solo representa el minitérmino de tres literales $xy'z$. la función simplificada es

$$F = z' + xy'$$

Si una función no está expresada como suma de minitérminos, podemos usar el mapa para obtener los minitérminos de la función y luego simplificar la función a una expresión con el mínimo de términos. Hay que asegurarse de que la expresión algebraica esté en forma de suma de productos. Cada término de producto se puede marcar en el mapa en uno, dos o más cuadrados. Luego, los minitérminos de la función se leen directamente del mapa.

EJEMPLO 3-4

Dada la función booleana

$$F = A'C + A'B + AB'C + BC$$

a) exprésela como suma de minitérminos

b) y luego halle la expresión mínima de suma de productos.

Tres términos de producto de la expresión tienen dos literales y se representan en un mapa de tres variables con dos cuadrados cada uno. Los dos cuadrados correspondientes al primer término, $A'C$, se encuentran en la figura 3-7 en la intersección de A' (primera fila) y C (dos columnas de en medio), que da los cuadrados 001 y 011. Observe que, al marcar los cuadrados

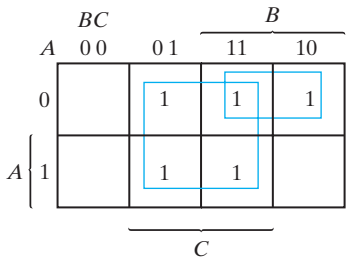


FIGURA 3-7
Mapa para el ejemplo 3-4; $A'C + A'B + AB'C + BC = C + A'B$

con 1, podríamos hallar ahí un 1 colocado por un término anterior. Esto sucede con el segundo término, $A'B$, que tiene unos en los cuadrados 011 y 010. Sin embargo, el cuadrado 011 también corresponde al primer término, $A'C$, así que simplemente dejamos el 1 que ya está ahí. Continuando de la misma manera, determinamos que el término $AB'C$ va en el cuadrado 101, que corresponde al minitérmino 5, y que el término BC tiene dos unos en los cuadrados 011 y 111. La función tiene un total de cinco minitérminos, como puede verse por los cinco unos en el mapa de la figura 3-7. Los minitérminos se leen directamente del mapa, y son 1, 2, 3, 5 y 7. La función se expresa en forma de suma de minitérminos:

$$F(A, B, C) = \sum(1, 2, 3, 5, 7)$$

La expresión de suma de productos dada originalmente tiene demasiados términos; puede simplificarse, como se muestra en el mapa, a una expresión de sólo dos términos:

$$F = C + A'B$$

3-2 MAPA DE CUATRO VARIABLES

El mapa para las funciones booleanas de cuatro variables se ilustra en la figura 3-8. En a) se presentan los 16 minitérminos y los cuadrados asignados a cada uno. En b) se ha redibujado el mapa de modo que muestre su relación con las cuatro variables. Las filas y columnas se numeran en orden según el código Gray, de modo que sólo un dígito cambie de valor entre dos filas o columnas adyacentes. El minitérmino correspondiente a cada cuadrado se obtiene de la concatenación del número de fila con el número de columna. Por ejemplo, los números de la tercera fila (11) y la segunda columna (01) dan, al concatenarse, el número binario 1101, que es el equivalente binario del 13 decimal. Así, el cuadrado de la tercera fila y la segunda columna representa al minitérmino m_{13}.

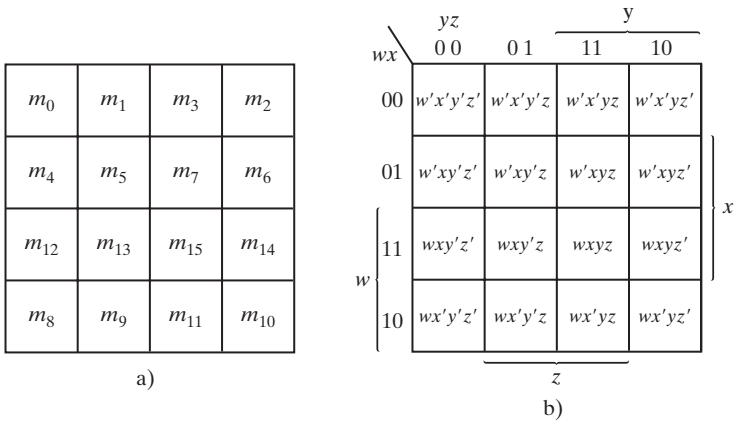


FIGURA 3-8
Mapa de cuatro variables

La minimización por mapa de funciones booleanas de cuatro variables es similar al método que se emplea para minimizar funciones de tres variables. Definimos los cuadrados adyacentes como cuadrados que están juntos. Además, consideramos que el mapa está en una superficie cuyos bordes superior e inferior, y derecho e izquierdo, están en contacto para formar cuadrados adyacentes. Por ejemplo, m_0 y m_2 forman cuadrados adyacentes, lo mismo que m_3 y m_{11}. Es fácil determinar la combinación de cuadrados adyacentes que es útil para el proceso de simplificación, por inspección del mapa de cuatro variables:

Un cuadrado representa un minitérmino, lo que da un término con cuatro literales.

Dos cuadrados adyacentes representan un término de tres literales.

Cuatro cuadrados adyacentes representan un término de dos literales.

Ocho cuadrados adyacentes representan un término de una sola literal.

Dieciséis cuadrados adyacentes representan la función igual a 1.

Ninguna otra combinación de cuadrados puede simplificar la función. Los dos ejemplos siguientes ilustran el uso del procedimiento para simplificar funciones booleanas de cuatro variables.

EJEMPLO 3-5

Simplifique la función booleana

$$F(w, x, y, z) = \sum(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$$

Puesto que la función tiene cuatro variables, hay que usar un mapa de cuatro variables. Los minitérminos indicados en la suma se han marcado con 1 en el mapa de la figura 3-9. Es posible combinar ocho cuadrados adyacentes marcados con 1 para formar el término de una literal y'. Los tres unos restantes de la derecha no pueden combinarse para dar un término simplificado. Se deberán combinar como dos o cuatro cuadrados adyacentes. Cuanto mayor sea el número

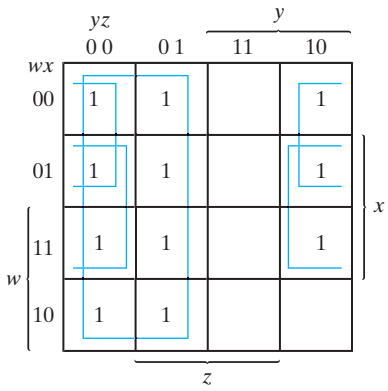


FIGURA 3-9
Mapa para el ejemplo 3-5; $F(w, x, y, z) = \sum(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$ $= y' + w'z' + xz'$

de cuadrados combinados, menos literales tendrá el término correspondiente. En este ejemplo, los dos unos de arriba a la derecha se combinan con los dos unos de arriba a la izquierda para dar el término $w'z'$. Cabe señalar que está permitido usar el mismo cuadrado más de una vez. Ahora nos queda un cuadrado marcado con 1 en la tercera fila y la cuarta columna (cuadrado 1110). En vez de tomar este cuadrado solo (lo que daría un término con cuatro literales), lo combinamos con cuadrados que ya usamos antes para formar un área de cuatro cuadrados adyacentes. Estos cuadrados ocupan la intersección de las dos filas de en medio y las dos columnas de los extremos, y dan el término xz'. La función simplificada es

$$F = y' + w'z' + xz'$$

EJEMPLO 3-6

Simplificar la función booleana

$$F = A'B'C' + B'CD' + A'BCD' + AB'C'$$

El área del mapa cubierta por esta función consiste en los cuadrados marcados con 1 en la figura 3-10. Esta función tiene cuatro variables y, en la forma en que está expresada, consta de tres términos de tres literales cada uno y un término de cuatro literales. Cada término de tres literales se representa en el mapa con dos cuadrados. Por ejemplo, $A'B'C'$ se representa en los cuadrados 0000 y 0001. La función se puede simplificar en el mapa tomando los unos de las cuatro esquinas para dar el término $B'D'$. Esto es posible porque los cuatro cuadrados están adyacentes si el mapa se dibuja en una superficie cuyos bordes superior e inferior, y derecho e izquierdo, están en contacto. Los dos unos de la izquierda en la fila superior se combinan con los dos unos de la fila inferior para dar el término $B'C'$. El uno restante se puede combinar en un área de dos cuadrados para dar el término $A'CD'$. La función simplificada es

$$F = B'D' + B'C' + A'CD'$$

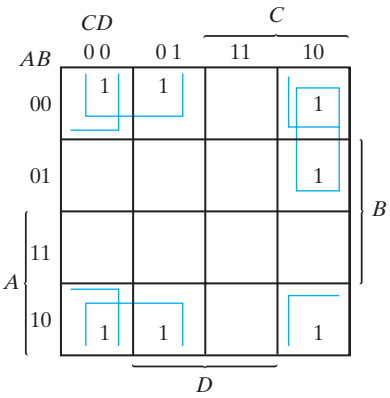


FIGURA 3-10
Mapa para el ejemplo 3-6; $A'B'C' + B'CD' + A'BCD' + AB'C' = B'D' + B'C' + A'CD'$

Implicantes primos

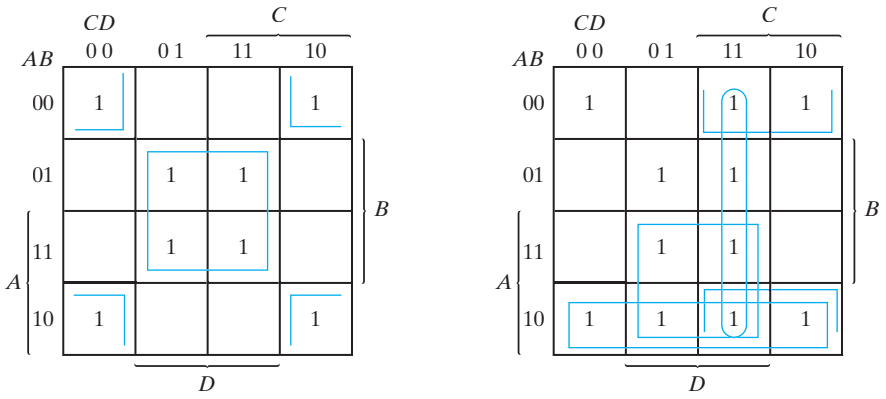
Al escoger cuadrados adyacentes en un mapa, debemos asegurarnos de cubrir todos los minitérminos de la función al combinar los cuadrados. Al mismo tiempo, es necesario minimizar el número de términos de la expresión y evitar términos redundantes cuyos minitérminos ya estén cubiertos por otros términos. Ocasionalmente, habrá dos o más expresiones que satisfacen los criterios de simplificación. El procedimiento para combinar cuadrados en el mapa podría hacerse más sistemático si entendemos el significado de los términos denominados implicante primo e implicante primo esencial. Un *implicante primo* es un término de producto que se obtiene combinando el número máximo posible de cuadrados adyacentes en el mapa. Si un minitérmino de un cuadrado está cubierto por sólo un implicante primo, decimos que ese implicante primo es *esencial*.

Podemos obtener los implicantes primos de una función a partir de un mapa combinando todos los números máximos posibles de cuadrados. Esto implica que un solo 1 en un mapa representa un implicante primo si no está adyacente a ningún otro 1. Dos unos adyacentes forman un implicante primo si no están dentro de un grupo de cuatro cuadrados adyacentes. Cuatro unos adyacentes forman un implicante primo si no están dentro de un grupo de ocho cuadrados adyacentes, y así sucesivamente. Los implicantes primos esenciales se encuentran examinando cada uno de los cuadrados marcados con 1 y tomando nota del número de implicantes primos que lo cubren. El implicante primo es esencial si es el único que cubre al minitérmino.

Considere la siguiente función booleana de cuatro variables:

$$F(A, B, C, D) = \sum(0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)$$

Los minitérminos de la función se han marcado con 1 en los mapas de la figura 3-11. La parte a) de la figura muestra dos implicantes primos esenciales. Un término es esencial porque sólo hay una forma de incluir el minitérmino m_0 en cuatro cuadrados adyacentes. Estos cuatro cuadrados definen al término $B'D'$. Asimismo, sólo hay una forma de combinar el minitérmi-



a) Implicantes primos esenciales
BD y B′D′

b) Implicantes primos CD, B′C
AD y AB′

FIGURA 3-11
Simplificación empleando implicantes primos

no m_5 con cuatro cuadrados adyacentes, y esto da el segundo término, BD. Los dos implicantes primos esenciales cubren ocho minitérminos. Ahora hay que considerar los tres minitérminos restantes, m_3, m_9 y m_{11}.

La figura 3-11b) muestra todas las formas posibles en que se cubren los tres minitérminos con implicantes primos. El minitérmino m_3 puede cubrirse con el implicante primo CD o con el $B'C$. El minitérmino m_9 puede cubrirse con AD o con AB'. El minitérmino m_{11} se cubre con cualquiera de los cuatro implicantes primos. La expresión simplificada se obtiene de la suma lógica de los dos implicantes primos esenciales y de cualesquier dos implicantes primos que cubran los minitérminos m_3, m_9 y m_{11}. Hay cuatro posibles formas de expresar la función con cuatro términos de producto de dos literales cada uno:

$$\begin{aligned} F &= BD + B'D' + CD + AD \\ &= BD + B'D' + CD + AB' \\ &= BD + B'D' + B'C + AD \\ &= BD + B'D' + B'C + AB' \end{aligned}$$

El ejemplo anterior demuestra que la identificación de los implicantes primos en el mapa ayuda a determinar las alternativas con que se cuenta para obtener una expresión simplificada.

El procedimiento para obtener del mapa la expresión simplificada requiere identificar primero todos los implicantes primos esenciales. La expresión simplificada se obtiene de la suma lógica de todos los implicantes primos esenciales y los demás implicantes primos necesarios para cubrir los minitérminos restantes que no estén cubiertos por los implicantes primos esenciales. Ocasionalmente, habrá más de una manera de combinar cuadrados, y cada combinación podría dar pie a una expresión igualmente simplificada.

3-3 MAPA DE CINCO VARIABLES

El uso de mapas para más de cuatro variables no es tan sencillo. Un mapa de cinco variables necesita 32 cuadrados, y uno de seis variables, 64 cuadrados. Cuando hay muchas variables, el número de cuadrados aumenta en forma considerable y la geometría para combinar cuadrados adyacentes se complica progresivamente.

El mapa de cinco variables se muestra en la figura 3-12. Consta de dos mapas de cuatro variables con las variables A, B, C, D y E. La variable A distingue a los dos mapas, como se indica en la parte superior del diagrama. El mapa de cuatro variables de la izquierda representa los 16 cuadrados en los que $A = 0$; el otro representa los cuadrados en los que $A = 1$. Los minitérminos 0 a 15 corresponden a $A = 0$ y los minitérminos 16 a 31 corresponden a $A = 1$. Cada mapa de cuatro variables conserva las adyacencias que definimos antes cuando se le considera aparte. Además, cada cuadrado del mapa $A = 0$ es adyacente al cuadrado correspondiente del mapa $A = 1$. Por ejemplo, el minitérmino 4 es adyacente al minitérmino 20, y el minitérmino 15, al 31. La mejor forma de visualizar esta nueva regla de adyacencia es imaginar que los dos medios mapas están uno encima del otro. Cualesquier dos cuadrados que queden uno encima del otro se considerarán adyacentes.

Siguiendo el procedimiento empleado con el mapa de cinco variables, es posible construir un mapa de seis variables con cuatro mapas de cuatro variables, para obtener los 64 cuadrados necesarios. Los mapas con seis o más variables requieren demasiados cuadrados y su uso

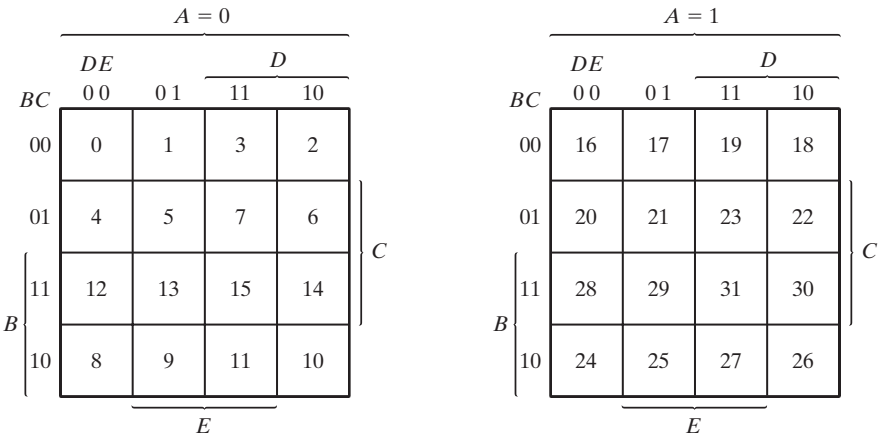


FIGURA 3-12
Mapa de cinco variables

resulta impráctico. La alternativa es utilizar programas de computadora escritos específicamente para facilitar la simplificación de funciones booleanas que tienen un gran número de variables.

Por inspección, y tomando en cuenta la nueva definición de cuadrados adyacentes, es posible demostrar que cualesquier 2^k cuadrados adyacentes, para $k = (0, 1, 2, \ldots, n)$, en un mapa de n variables, representan un área que produce un término de $n - k$ literales. Para que esta afirmación tenga sentido, n deberá ser mayor que k. Cuando $n = k$, toda el área del mapa se combina para dar la función de identidad. La tabla 3-1 muestra la relación entre el número de cuadrados adyacentes y el número de literales en el término. Por ejemplo, ocho cuadrados adyacentes combinan un área del mapa de cinco variables para dar un término de dos literales.

Tabla 3-1
Relación entre el número de cuadrados adyacentes y el número de literales en el término

	Número de cuadrados adyacentes	**Número de literales del término en un mapa de *n* variables**			
K	2^k	$n = 2$	$n = 3$	$n = 4$	$n = 5$
0	1	2	3	4	5
1	2	1	2	3	4
2	4	0	1	2	3
3	8		0	1	2
4	16			0	1
5	32				0

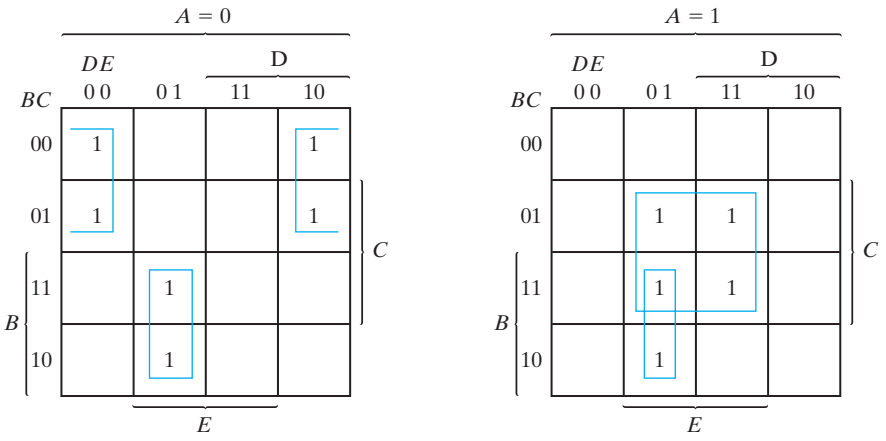


FIGURA 3-13
Mapa para el ejemplo 3-7; $F = A'B'E' + BD'E + ACE$

EJEMPLO 3-7

Simplifique la función booleana

$$F(A, B, C, D, E) = (0, 2, 4, 6, 9, 13, 21, 23, 25, 29, 31)$$

El mapa de cinco variables para esta función se muestra en la figura 3-13. Hay seis minitérminos, del 0 al 15, que pertenecen a la parte del mapa en la que $A = 0$. Los otros cinco minitérminos pertenecen a $A = 1$. Cuatro cuadrados adyacentes del mapa $A = 0$ se combinan para dar el término de tres literales $A'B'E'$. Advierta que es necesario incluir A' en el término porque todos los cuadrados están asociados a $A = 0$. Los dos cuadrados de la columna 01 y las dos últimas filas son comunes a ambas partes del mapa; por tanto, constituyen cuatro cuadrados adyacentes y dan el término de tres literales $BD'E$. La variable A no se incluye aquí porque los cuadrados adyacentes pertenecen tanto a $A = 0$ como a $A = 1$. El término ACE se obtiene de los cuatro cuadrados adyacentes que están totalmente dentro del mapa $A = 1$. La función simplificada es la suma lógica de los tres términos:

$$F = A'B'E' + BD'E + ACE$$

3-4 SIMPLIFICACIÓN DE PRODUCTO DE SUMAS

Las funciones booleanas simplificadas que dedujimos del mapa en todos los ejemplos anteriores se expresaron en la forma de suma de productos. Con una modificación menor, se obtiene la forma de producto de sumas.

El procedimiento para obtener una función minimizada en forma de producto de sumas es consecuencia de las propiedades básicas de las funciones booleanas. Los unos que se colocan en los cuadrados del mapa representan los minitérminos de la función. Los minitérminos no in-

cluidos en la función denotan el complemento de la función. Entonces, el complemento de una función está representado en el mapa por los cuadrados que no se han marcado con 1. Si marcamos los cuadrados vacíos con 0 y los combinamos en cuadrados adyacentes válidos, obtendremos una expresión simplificada del complemento de la función, es decir, de F'. El complemento de F' nos dará otra vez la función F. Por el teorema generalizado de DeMorgan, la función así obtenida estará automáticamente en forma de producto de sumas. La mejor forma de explicar esto es con un ejemplo.

EJEMPLO 3-8

Simplifique la siguiente función booleana en forma de **a)** suma de productos y **b)** producto de sumas:

$$F(A, B, C, D) = \sum(0, 1, 2, 5, 8, 9, 10)$$

Los unos marcados en el mapa de la figura 3-14 representan todos los minitérminos de la función. Los cuadrados marcados con 0 representan los minitérminos no incluidos en F y, por tanto, denotan al complemento de F. Si combinamos los cuadrados que tienen 1, obtendremos la función simplificada en forma de suma de productos:

a) $F = B'D' + B'C' + A'C'D$

Si combinamos los cuadrados marcados con 0, como se indica en el diagrama, obtendremos la función complementada simplificada:

$$F' = AB + CD + BD'$$

Al aplicar el teorema de DeMorgan (obteniendo el dual y complementando cada literal como se describe en la sección 2-4) se obtiene la función simplificada en forma de producto de sumas:

b) $F = (A' + B')(C' + D')(B' + D)$

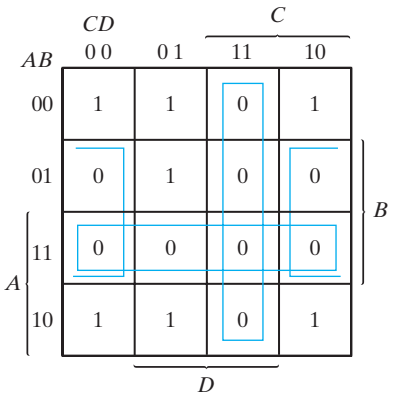
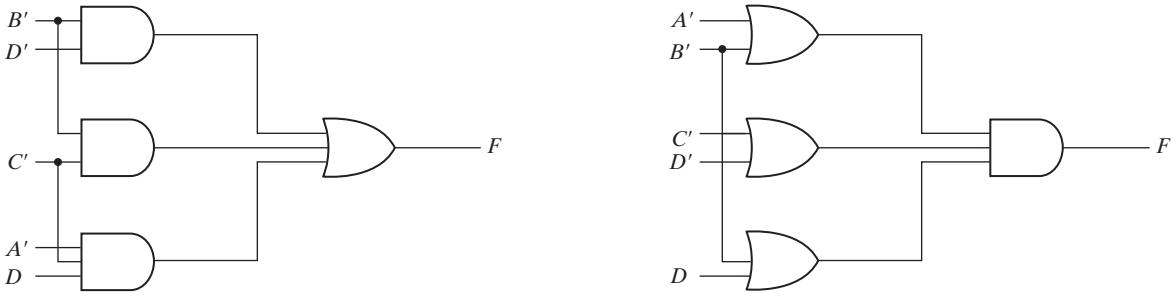


FIGURA 3-14
Mapa para el ejemplo 3-8; $F(A, B, C, D) = \sum(0, 1, 2, 5, 8, 9, 10) = B'D' + B'C' + A'C'D = (A' + B')(C' + D')(B' + D)$



a) $F = B'D' + B'C' + A'C'D$ b) $F = (A' + B')(C' + D')(B' + D)$

FIGURA 3-15
Implementación con compuertas de la función del ejemplo 3-8

La implementación de las expresiones simplificadas obtenidas en el ejemplo 3-8 se muestra en la figura 3-15. La expresión de suma de productos se implementa en a) con un grupo de compuertas AND, una para cada término AND. Las salidas de las compuertas AND se conectan a las entradas de una sola compuerta OR. En b) se ha implementado la misma función en forma de producto de sumas con un grupo de compuertas OR, una por cada término OR. Las salidas de las compuertas OR se conectan a las entradas de una sola compuerta AND. En cada caso, suponemos que contamos directamente con las variables de entrada complementadas, por lo que no se necesitan inversores. El patrón de configuración establecido en la figura 3-15 es la forma general de implementar cualquier función booleana expresada en una de las formas estándar. Si está en suma de productos, se conectan compuertas AND a una sola compuerta OR; si está en producto de sumas, se conectan compuertas OR a una sola compuerta AND. Cualquiera de las configuraciones forma dos niveles de compuertas, por lo que se afirma que la implementación de una función en forma estándar es una implementación de dos niveles.

El ejemplo 3-8 ilustró el procedimiento para obtener la simplificación de producto de sumas cuando la función se expresa originalmente en la forma canónica de suma de minitérminos. El mismo procedimiento es válido cuando la función se expresa originalmente en la forma

Tabla 3-2
Tabla de verdad de la función F

x	*y*	*z*	*F*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

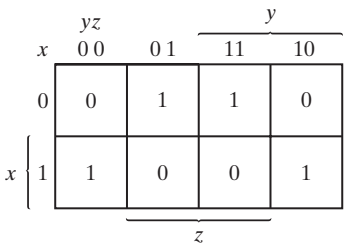


FIGURA 3-16
Mapa de la función de la tabla 3-2

canónica de producto de maxitérminos. Consideremos, por ejemplo, la tabla de verdad que define la función F en la tabla 3-2. En suma de minitérminos, esta función se expresa así:

$$F(x, y, z) = \sum(1, 3, 4, 6)$$

En producto de maxitérminos, se expresa así:

$$F(x, y, z) = \Pi\,(0, 2, 5, 7)$$

Dicho de otro modo, los unos de la función representan a los minitérminos, y los ceros, a los maxitérminos. El mapa de esta función se presenta en la figura 3-16. Podemos comenzar a simplificar esta función marcando con 1 los cuadrados correspondientes a cada minitérmino para el cual la función da 1. Los cuadrados restantes se marcan con 0. Por otra parte, si lo que se da originalmente es el producto de maxitérminos, podemos comenzar por colocar ceros en los cuadrados indicados por la función; luego marcaremos con 1 los cuadrados restantes. Una vez marcados todos los cuadrados, es posible simplificar la función en cualquiera de las formas estándar. Si queremos la suma de productos, combinaremos los unos para obtener

$$F = x'z + xz'$$

Si queremos el producto de sumas, combinaremos los ceros para obtener la función complementada simplificada

$$F' = xz + x'z'$$

lo que demuestra que la función OR exclusivo es el complemento de la función de equivalencia (sección 2-6). Al calcular el complemento de F', se obtiene la función simplificada en forma de producto de sumas:

$$F = (x' + z')(x + z)$$

Para introducir en el mapa una función expresada como producto de sumas, se calcula el complemento de la función, el cual indicará los cuadrados que deben marcarse con 0. Por ejemplo, la función

$$F = (A' + B' + C')(B + D)$$

se puede introducir en el mapa obteniendo primero su complemento,

$$F' = ABC + B'D'$$

y marcando después con 0 los cuadrados que representan a los minitérminos de F'. Los cuadrados restantes se marcarán con 1.

3-5 CONDICIONES DE INDIFERENCIA

La suma lógica de los minitérminos asociados con una función booleana especifica las condiciones en que la función da 1. La función da 0 para el resto de los minitérminos. Esto supone que todas las combinaciones de valores de las variables de la función son válidas. En la práctica, hay algunas aplicaciones en las que la función no está especificada para ciertas combinaciones de las variables. Por ejemplo, el código binario de cuatro bits para los dígitos decimales tiene seis combinaciones que no se usan y que, por tanto, se consideran no especificadas. Las funciones con salidas no especificadas para ciertas combinaciones de entradas se llaman funciones incompletamente especificadas. En casi todas las aplicaciones, es irrelevante el valor que asuma la función para los minitérminos no especificados. Por ello, se acostumbra llamar condiciones de indiferencia (*don't care*, en inglés) a los minitérminos no especificados de una función. Conviene usar estas condiciones de indiferencia en el mapa para simplificar aún más la expresión booleana.

Debe quedar claro que un minitérmino indiferente es una combinación de variables cuyo valor lógico no está especificado. No podemos marcarlo con 1 en el mapa porque ello requeriría que la función siempre dé 1 para esa combinación. Por lo mismo, si marcamos con 0 su cuadrado en el mapa, estaremos exigiendo que la función sea 0. Para distinguir la condición de indiferencia, usamos una *X* en lugar de unos y ceros. Así, una *X* en un cuadrado del mapa indica que no nos importa si se asigna el valor de 0 o de 1 a *F* para el minitérmino en cuestión.

Al escoger cuadrados adyacentes para simplificar la función, podemos suponer que los minitérminos indiferentes son 0 o 1, lo que más nos convenga. Al simplificar la función, podemos optar por incluir cada minitérmino indiferente con los unos o con los ceros, dependiendo de qué combinación produzca la expresión más simple.

EJEMPLO 3-9

Simplifique la función booleana

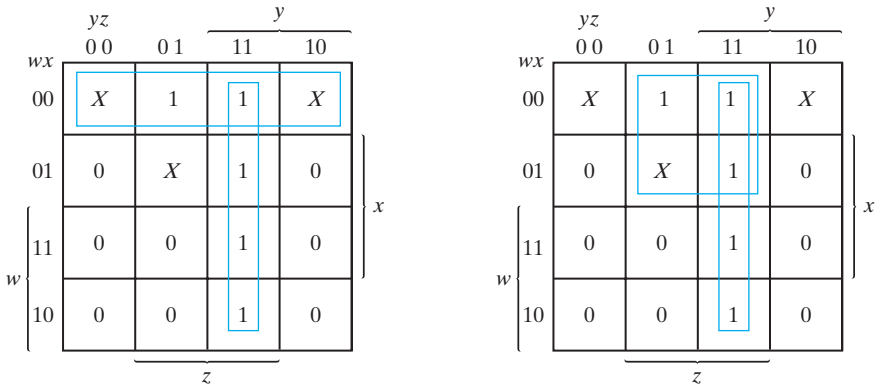
$$F(w, x, y, z) = \sum(1, 3, 7, 11, 15)$$

que tiene las condiciones de indiferencia

$$d(w, x, y, z) = \sum(0, 2, 5)$$

Los minitérminos de *F* son las combinaciones de variables que hacen que la función dé 1. Los minitérminos de *d* son los minitérminos indiferentes a los que podría asignarse 0 o bien 1. La simplificación del mapa se muestra en la figura 3-17. Los minitérminos de *F* se marcan con 1, los de *d* se marcan con *X* y los cuadrados restantes se marcan con 0. Para obtener la expresión simplificada en forma de suma de productos, deberemos incluir los cinco unos del mapa, pero podríamos incluir o no cualquiera de las *X*, dependiendo de qué tanto ello simplifique la función. El término yz cubre los cuatro minitérminos de la tercera columna. El minitérmino restante, m_1, se puede combinar con el minitérmino m_3 para dar el término de tres literales $w'x'z$. Pero si incluimos una o dos *X* adyacentes podremos combinar cuatro cuadrados adyacentes para obtener un término de dos literales. En la parte a) del diagrama se han incluido los minitérminos indiferentes 0 y 2 junto con los unos, para dar la función simplificada

$$F = yz + w'x'$$



a) $F = yz + w'x'$ b) $F = yz + w'z$

FIGURA 3-17
Ejemplo con condiciones de indiferencia

En la parte b), se ha incluido el minitérmino 5 junto con los unos, y ahora la función simplificada es

$$F = yz + w'z$$

Cualquiera de las dos expresiones anteriores satisface las condiciones planteadas para el ejemplo.

■

El ejemplo anterior ilustró cómo los minitérminos indiferentes se marcan inicialmente con *X* en el mapa y se consideran como 0 o como 1. La decisión de tomar cada uno como 0 o como 1 dependerá de cómo ello simplifique la función incompletamente especificada. Una vez tomada la decisión, la función simplificada obtenida consistirá en una suma de minitérminos que incluye los minitérminos que inicialmente no estaban especificados y que ahora se ha decidido incluir con los unos. Consideremos las dos expresiones simplificadas que se obtienen en el ejemplo 3-9:

$$F(w, x, y, z) = yz + w'x' = \sum(0, 1, 2, 3, 7, 11, 15)$$
$$F(w, x, y, z) = yz + w'z = \sum(1, 3, 5, 7, 11, 15)$$

Ambas expresiones incluyen a los minitérminos 1, 3, 7, 11 y 15, que hacen a la función *F* igual a 1. Los minitérminos indiferentes 0, 2 y 5 se tratan de diferente manera en cada expresión. La primera expresión incluye a los minitérminos 0 y 2 junto con los unos y deja al minitérmino 5 con los ceros. La segunda expresión incluye al minitérmino 5 con los unos y deja a los minitérminos 0 y 2 con los ceros. Las dos funciones son algebraicamente distintas. Ambas cubren los minitérminos especificados de la función, pero cada una cubre diferentes minitérminos indiferentes. En lo que a la función incompletamente especificada concierne, cualquiera de las dos expresiones es aceptable porque la única diferencia radica en el valor de *F* para los minitérminos indiferentes.

También es posible obtener una expresión simplificada de producto de sumas para la función de la figura 3-17. En este caso, la única forma de combinar los ceros es incluyendo los minitérminos indiferentes 0 y 2 con los ceros, para dar una función complementada simplificada:

$$F' = z' + wy'$$

Al calcular el complemento de F' obtenemos la expresión simplificada como producto de sumas:

$$F(w, x, y, z) = z(w' + y) = \sum(1, 3, 5, 7, 11, 15)$$

En este caso, hemos incluido los minitérminos 0 y 2 con los ceros, y el minitérmino 5 con los unos.

3-6 IMPLEMENTACIÓN CON NAND Y NOR

Muchos circuitos digitales se construyen con compuertas NAND y NOR en lugar de con compuertas AND y OR. Las primeras son más fáciles de fabricar con componentes electrónicos y son las compuertas básicas empleadas en todas las familias de lógica de CI digitales. En virtud del destacado papel que las compuertas NAND y NOR desempeñan en el diseño de circuitos digitales, se han desarrollado reglas y procedimientos para convertir funciones booleanas expresadas en términos de AND, OR y NOT en diagramas lógicos NAND y NOR equivalentes.

Circuitos NAND

Se dice que la compuerta NAND es una compuerta universal porque cualquier sistema digital puede implementarse con ella. Para demostrar que cualquier función booleana se puede implementar con compuertas NAND, basta con demostrar que las operaciones lógicas AND, OR y complemento se pueden obtener exclusivamente con compuertas NAND. Esto se aprecia en la figura 3-18. La operación complemento se obtiene con una compuerta NAND de una sola entrada que se comporta exactamente como un inversor. La operación AND requiere dos compuertas NAND. La primera produce la operación NAND y la segunda invierte el sentido lógico de la señal. La operación OR se logra con una compuerta NAND que lleva inversores en cada entrada.

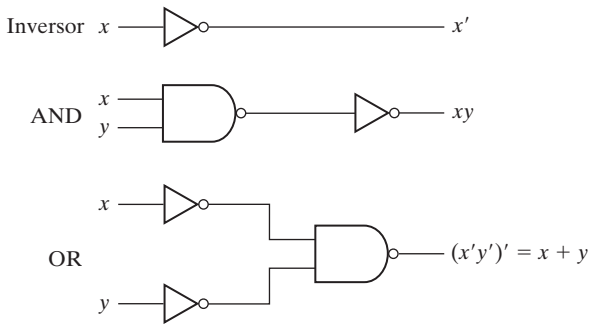
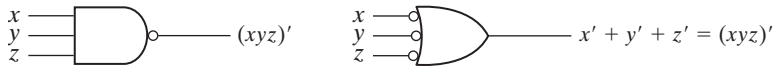


FIGURA 3-18
Operaciones lógicas con compuertas NAND



a) AND–invertir b) Invertir–OR

FIGURA 3-19
Dos símbolos gráficos para la compuerta NAND

Una forma conveniente de implementar una función booleana con compuertas NAND consiste en obtener la función booleana simplificada en términos de operadores booleanos y luego convertir la función a lógica NAND. La conversión de una expresión algebraica de AND, OR y complemento a NAND se efectúa aplicando sencillas técnicas de manipulación de circuitos que convierten los diagramas AND-OR en diagramas NAND.

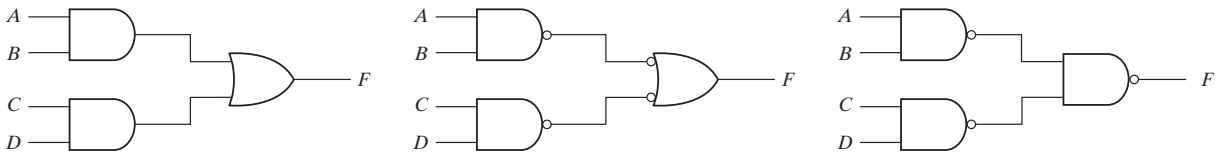
Para facilitar la conversión a lógica NAND, es conveniente definir un símbolo gráfico alterno para la compuerta. En la figura 3-19 se presentan dos símbolos gráficos equivalentes para la compuerta NAND. El símbolo AND-invertir ya se definió antes y consiste en un símbolo gráfico AND seguido de un pequeño indicador circular de negación llamado burbuja. Como alternativa, podemos representar una compuerta NAND con un símbolo gráfico OR precedido por una burbuja en cada entrada. El símbolo invertir-OR para la compuerta NAND es consecuencia del teorema de DeMorgan y de la convención de que el indicador de negación denota complementación. Los dos símbolos gráficos son útiles en el análisis y diseño de circuitos NAND. Si se usan ambos símbolos en el mismo diagrama, decimos que el circuito está en notación mixta.

Implementación de dos niveles

La implementación de funciones booleanas con compuertas NAND requiere expresar la función en forma de suma de productos. Para ver la relación entre una expresión de suma de productos y su implementación NAND equivalente, consideremos los diagramas lógicos de la figura 3-20. Los tres diagramas son equivalentes e implementan la función

$$F = AB + CD$$

En a), la función se implementa con compuertas AND y OR. En (b), las compuertas AND se han sustituido por compuertas NAND y la compuerta OR se sustituyó por una compuerta NAND representada por un símbolo gráfico invertir-OR. Recuerde que una burbuja denota



a) b) c)

FIGURA 3-20
Tres formas de implementar $F = AB + CD$

complementación y que dos burbujas en una misma línea representan doble complementación, y pueden eliminarse. Si se quitan las burbujas de las compuertas de b) se obtiene el circuito de a). Por tanto, los dos diagramas implementan la misma función y son equivalentes.

En la figura 3-20c), se ha representado la compuerta NAND final con un símbolo gráfico AND-invertir. Al dibujar diagramas lógicos NAND, los circuitos que se muestran en b) o c) son aceptables. El de b) usa notación mixta y representa una relación más directa con la expresión booleana que implementa. La implementación NAND de la figura 3-20c) se puede verificar algebraicamente. La función que implementa se convierte fácilmente a suma de productos aplicando el teorema de DeMorgan:

$$F = \big((AB)'(CD)'\big)' = AB + CD$$

EJEMPLO 3-10

Implemente la función booleana siguiente con compuertas NAND:

$$F(x, y, z) = (1, 2, 3, 4, 5, 7)$$

El primer paso es simplificar la función como suma de productos. Esto se hace con el mapa de la figura 3-21a), del cual se obtiene la función simplificada

$$F = xy' + x'y + z$$

La implementación NAND de dos niveles se presenta en la figura 3-21b) en notación mixta. Observe que la entrada z necesita una compuerta NAND de una sola entrada (inversor) para compensar la burbuja de la compuerta del segundo nivel. En la figura 3-21c) se presenta otra forma de dibujar el diagrama lógico. Aquí todas las compuertas NAND se representan con el

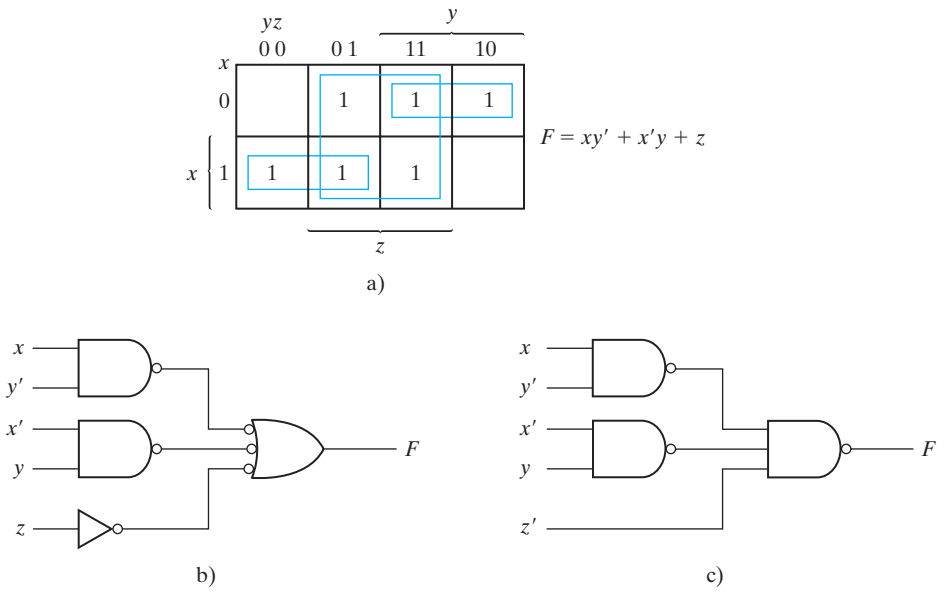


FIGURA 3-21
Solución del ejemplo 3-10

mismo símbolo gráfico. El inversor con entrada z se ha eliminado, pero la variable de entrada se ha complementado y se denota con z'.

■

El procedimiento descrito en el ejemplo anterior sugiere que es factible implementar una función booleana con dos niveles de compuertas NAND. El procedimiento para obtener el diagrama lógico a partir de una función booleana es el siguiente:

1. Simplificar la función y expresarla como suma de productos.
2. Incluir una compuerta NAND por cada término de producto que tenga por lo menos dos literales. Las entradas de cada compuerta NAND serán las literales del término. Esto constituye un grupo de compuertas de primer nivel.
3. Incluir una sola compuerta en el segundo nivel, empleando el símbolo gráfico AND-invertir o invertir-OR, cuyas entradas provienen de las salidas de las compuertas de primer nivel.
4. Los términos con una sola literal requerirán un inversor en el primer nivel, pero si esa literal solitaria está complementada, se le podrá conectar directamente a una entrada de la compuerta NAND del segundo nivel.

Circuitos NAND multinivel

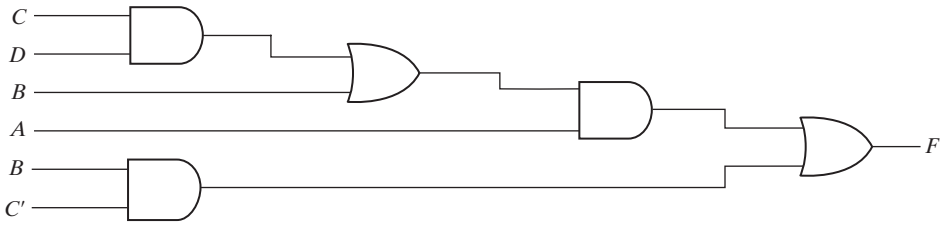
La forma estándar de expresar funciones booleanas da pie a una implementación de dos niveles. Hay ocasiones en que el diseño de sistemas digitales produce estructuras con tres o más niveles de compuertas. El procedimiento más común para diseñar circuitos multinivel es expresar la función booleana en términos de operaciones AND, OR y complemento. Entonces, la función podrá implementarse con compuertas AND y OR. Luego, si es necesario, se le puede convertir en un circuito con puras compuertas NAND. Considere, por ejemplo, la función booleana:

$$F = A(CD + B) + BC'$$

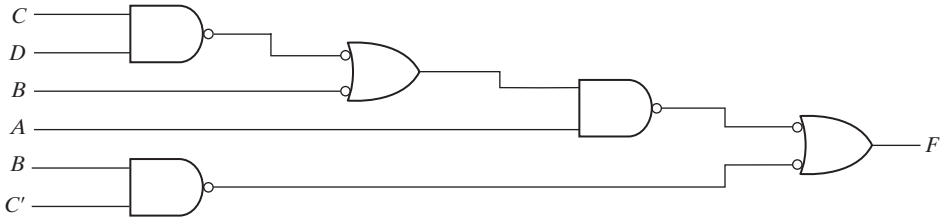
Aunque es posible quitar los paréntesis y reducir la expresión a una forma estándar de suma de productos, preferimos implementarla como circuito multinivel como ilustración. La implementación AND-OR se ilustra en la figura 3-22a). El circuito tiene cuatro niveles de compuertas. El primer nivel posee dos compuertas AND. El segundo tiene una compuerta OR seguida de una compuerta AND en el tercer nivel y una compuerta OR en el cuarto nivel. Los diagramas lógicos con un patrón de niveles alternos de compuertas AND y OR se pueden convertir fácilmente en un circuito NAND utilizando la notación mixta. Esto se muestra en la figura 3-22b). El procedimiento consiste en sustituir cada compuerta AND por un símbolo gráfico AND-invertir, y cada compuerta OR, por un símbolo gráfico invertir-OR. El circuito NAND realizará la misma lógica que el diagrama AND-OR siempre que haya dos burbujas sobre la misma línea. La burbuja asociada a la entrada B produce una complementación adicional, que debe compensarse cambiando la literal de entrada a B'.

El procedimiento general para convertir un diagrama AND-OR multinivel en un diagrama NAND con notación mixta es el siguiente:

1. Convertir todas las compuertas AND en compuertas NAND con símbolos gráficos AND-invertir.
2. Convertir todas las compuertas OR en compuertas NAND con símbolos gráficos invertir-OR.



a) Compuertas AND-OR



b) Compuertas NAND

FIGURA 3-22
Implementación de $F = A(CD + B) + BC'$

3. Revisar todas las burbujas del diagrama. Por cada burbuja que no esté compensada por otra sobre la misma línea, hay que insertar un inversor (compuerta NAND de una sola entrada) o complementar la literal de entrada.

Como ejemplo adicional, consideremos la función booleana multinivel

$$F = (AB' + A'B)(C + D')$$

La implementación AND-OR se presenta en la figura 3-23a) con tres niveles de compuertas. La conversión a NAND con notación mixta aparece en la parte b) del diagrama. Las dos burbujas adicionales asociadas a las entradas C y D' hacen que estas dos literales se complementen a C' y D. La burbuja en la compuerta NAND de salida complementa el valor de salida, por lo que necesitamos insertar una compuerta inversora en la salida para complementar otra vez la señal y obtener el valor original.

Implementación NOR

La operación NOR es el dual de la operación NAND. Por tanto, todos los procedimientos y reglas para la lógica NOR son el dual de los procedimientos y reglas correspondientes que se han desarrollado para la lógica NAND. La compuerta NOR es otra compuerta universal que sirve para implementar cualquier función booleana. La implementación de las operaciones de complemento, OR y AND con compuertas NOR se aprecia en la figura 3-24. La operación de complemento se obtiene de una compuerta NOR con una sola entrada que se comporta exac-

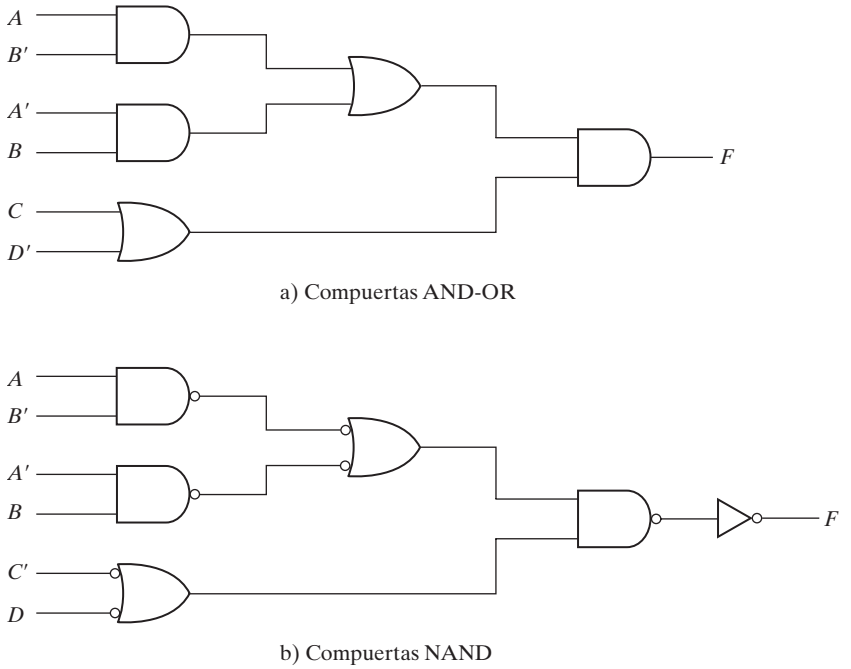


FIGURA 3-23
Implementación de $F = (AB' + A'B)(C + D')$

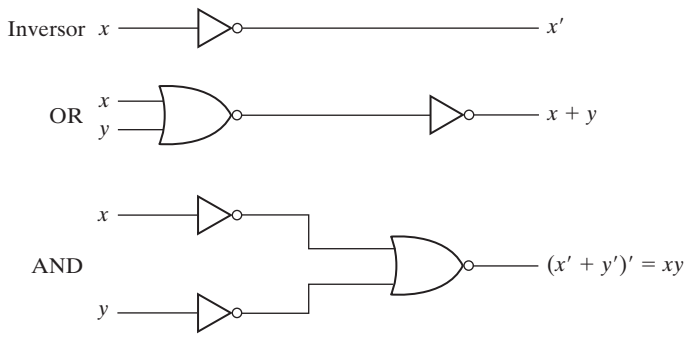
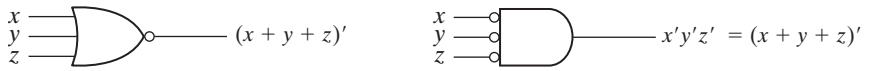


FIGURA 3-24
Operaciones lógicas con compuertas NOR

tamente como un inversor. La operación OR requiere dos compuertas NOR y la AND, una compuerta NOR con inversores en cada entrada.

Los dos símbolos gráficos de la notación mixta se muestran en la figura 3-25. El símbolo OR-invertir define la operación NOR como un OR seguido de un complemento. El símbolo invertir-AND complementa cada una de las entradas y luego realiza una operación AND. Los dos símbolos designan la misma operación NOR y son lógicamente idénticos por el teorema de DeMorgan.



a) OR-invertir b) invertir-AND

FIGURA 3-25
Dos símbolos gráficos para la compuerta NOR

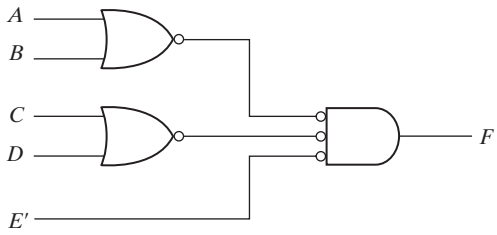


FIGURA 3-26
Implementación de $F = (A + B)(C + D)E$

Una implementación de dos niveles con compuertas NOR requiere simplificar la función en forma de producto de sumas. Recuerde que la expresión simplificada de producto de sumas se obtiene del mapa combinando los ceros y complementando. Una expresión en producto de sumas se implementa con un primer nivel de compuertas OR que produce los términos de suma, seguido de una compuerta AND de segundo nivel que da el producto. La transformación del diagrama OR-AND en un diagrama NOR se logra cambiando las compuertas OR por compuertas NOR representadas por símbolos gráficos OR-invertir, y la compuerta AND, por una compuerta NOR representada por un símbolo gráfico invertir-AND. Si la compuerta de segundo nivel tiene como entrada un término de una sola literal, ésta deberá complementarse. La figura 3-26 muestra la implementación NOR de una función expresada como producto de sumas:

$$F = (A + B)(C + D)E$$

El patrón OR-AND se detecta fácilmente quitando los pares de burbujas que estén sobre la misma línea. La variable E se complementa para compensar la tercera burbuja en la entrada de la compuerta de segundo nivel.

El procedimiento para convertir un diagrama AND-OR multinivel en un diagrama sólo NOR es similar al que se utilizó para las compuertas NAND. En el caso de NOR, hay que sustituir cada compuerta OR por un símbolo OR-invertir, y cada compuerta AND, por un símbolo invertir-AND. Toda burbuja que no esté compensada por otra burbuja en la misma línea necesitará un inversor, o se deberá complementar la literal de entrada.

La transformación del diagrama AND-OR de la figura 3-23a) en un diagrama NOR se ilustra en la figura 3-27. La función booleana para este circuito es

$$F = (AB' + A'B)(C + D')$$

El diagrama AND-OR equivalente se deduce del diagrama NOR quitando todas las burbujas. Para compensar las burbujas en cuatro entradas, es preciso complementar las literales de entrada correspondientes.

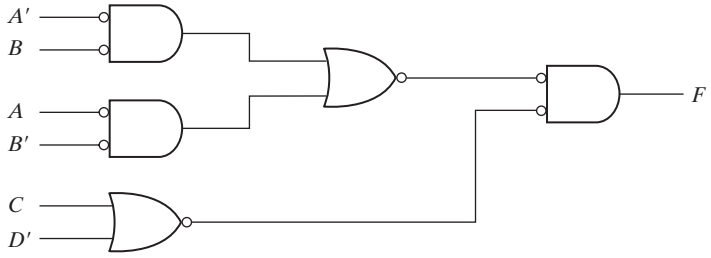


FIGURA 3-27
Implementación de $F = (AB' + A'B)(C + D')$ con compuertas NOR

3-7 OTRAS IMPLEMENTACIONES DE DOS NIVELES

Los tipos de compuertas que más comúnmente se encuentran en los circuitos integrados son las NAND y NOR. Por ello, las implementaciones de lógica NAND y NOR son las más importantes en la práctica. Algunas compuertas NAND o NOR (pero no todas) contemplan la posibilidad de una conexión con alambre entre las salidas de dos compuertas para formar una función lógica específica. Este tipo de lógica se llama *lógica alambrada* (*wired*, en inglés). Por ejemplo, si se conectan entre sí compuertas NAND TTL de colector abierto, efectúan la lógica AND alambrada. (La compuerta TTL de colector abierto se representa en la figura 10-11 del capítulo 10.) La lógica AND alambrada efectuada con dos compuertas NAND se muestra en la figura 3-28a). La compuerta AND se dibuja con líneas que pasan por el centro de la compuerta, para distinguirla de una compuerta convencional. La compuerta AND alambrada no es una compuerta física, sino un símbolo que designa la función obtenida de la conexión alambrada que se indica. La función lógica implementada por el circuito de la figura 3-28a) es

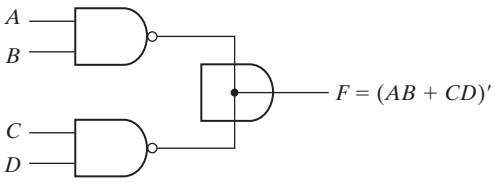
$$F = (AB)' \cdot (CD)' = (AB + CD)'$$

y se denomina función AND-OR-INVERT.

Asimismo, la salida NOR de compuertas ECL se puede vincular para desempeñar una función OR alambrada. La función lógica implementada por el circuito de la figura 3-28b) es

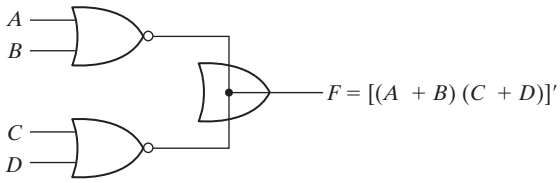
$$F = (A + B)' + (C + D)' = \big[(A + B)(C + D)\big]'$$

y se denomina función OR-AND-INVERT.



a) AND alambrado en compuertas NAND TTL de colector abierto

(AND-OR-INVERT)



b) OR alambrado en compuertas ECL

(OR-AND-INVERT)

FIGURA 3-28
Lógica alambrada

Una compuerta de lógica alambrada no produce una compuerta física de segundo nivel porque no es más que una conexión de alambres. No obstante, en nuestras explicaciones, consideraremos los circuitos de la figura 3-28 como implementaciones de dos niveles. El primer nivel consiste en compuertas NAND (o NOR) y el segundo tiene una sola compuerta AND (u OR). La conexión alambrada dentro del símbolo gráfico se omitirá en explicaciones posteriores.

Formas no degeneradas

Desde un punto de vista teórico, resulta interesante averiguar cuántas combinaciones de compuertas de dos niveles puede haber. Consideraremos cuatro tipos de compuertas: AND, OR, NAND y NOR. Si asignamos un tipo de compuerta al primer nivel y un tipo al segundo nivel, encontramos que hay 16 posibles combinaciones de formas de dos niveles. (Se puede usar el mismo tipo en el primer nivel y en el segundo, como en la implementación NAND-NAND.) Ocho de esas combinaciones se consideran formas *degeneradas* porque degeneran a una sola operación. Esto se percibe en un circuito con compuertas AND en el primer nivel y una compuerta AND en el segundo nivel. La salida del circuito no es más que la función AND de todas las variables de entrada. Las otras ocho formas *no degeneradas* producen una implementación de suma de productos o producto de sumas. Las ocho formas no degeneradas son:

AND-OR	OR-AND
NAND-NAND	NOR-NOR
NOR-OR	NAND-AND
OR-NAND	AND-NOR

La primera compuerta indicada en cada una de las formas constituye el primer nivel de la implementación. La segunda compuerta es una sola, colocada en el segundo nivel. Hemos presentado las formas en pares, de modo que las dos formas de cada línea son una el dual de la otra.

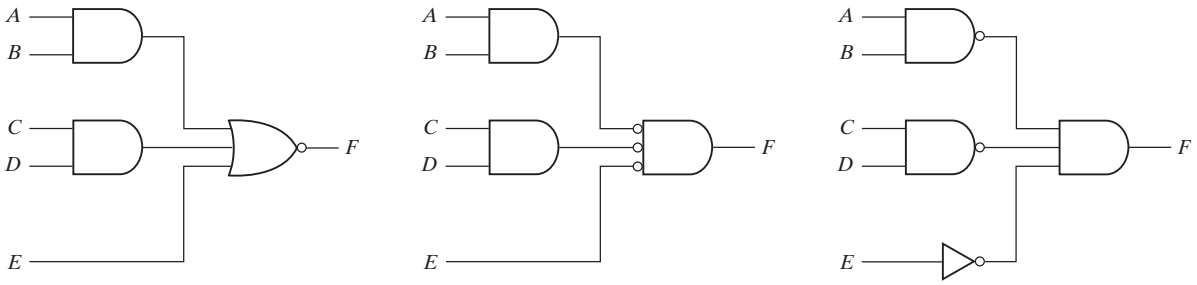
Las formas AND-OR y OR-AND son las formas básicas de dos niveles que vimos en la sección 3-4. Presentamos las formas NAND-NAND y NOR-NOR en la sección 3-6. En esta sección investigaremos las otras cuatro formas.

Implementación AND-OR-INVERT

Las dos formas NAND-AND y AND-NOR son equivalentes y podemos verlas juntas. Ambas efectúan la función AND-OR-INVERT, como se muestra en la figura 3-29. La forma AND-NOR se parece a la forma AND-OR pero con una inversión indicada por la burbuja en la salida de la compuerta NOR. Esta forma implementa la función

$$F = (AB + CD + E)'$$

Al utilizar el símbolo gráfico alterno para la compuerta NOR, se obtiene el diagrama de la figura 3-29b). Advierta que la variable sola *E no* se complementa porque el único cambio que se efectúa es en el símbolo gráfico de la compuerta NOR. Ahora pasamos la burbuja de la terminal de entrada de la compuerta de segundo nivel a las terminales de salida de las compuertas de primer nivel. Se necesita un inversor para la variable sola, como compensación de la



a) AND-NOR b) AND-NOR c) NAND-AND

FIGURA 3-29
Circuitos AND-OR-INVERT; $F = (AB + CD + E)'$

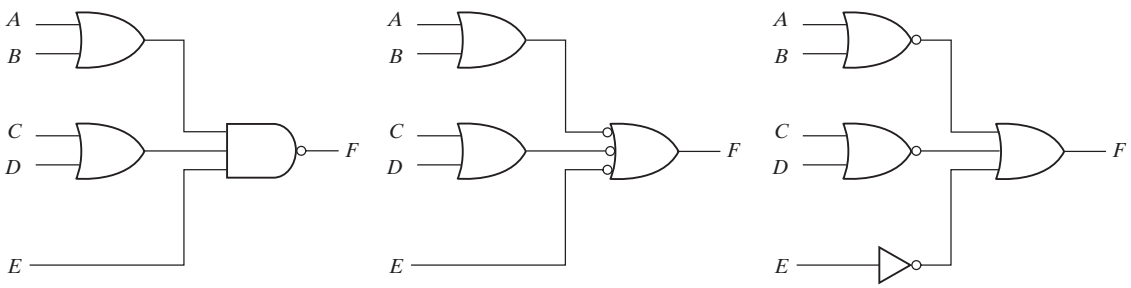
burbuja. O bien, podemos quitar el inversor si la entrada E se complementa. El circuito de la figura 3-29c) es una forma NAND-AND, y en la figura 3-28 vimos que implementa la función AND-OR-INVERT.

Una implementación AND-OR requiere una expresión en forma de suma de productos. La implementación AND-OR-INVERT es similar, excepto por la inversión. Por tanto, si simplificamos el *complemento* de la función en forma de suma de productos (combinando los ceros del mapa), podremos implementar F' con la parte AND-OR de la función. Cuando F' pase por la obligada inversión de salida (la parte INVERT), generará la salida de la función F. Más adelante se presentará un ejemplo de la implementación AND-OR-INVERT.

Implementación OR-AND-INVERT

Las formas OR-NAND y NOR-OR efectúan la función OR-AND-INVERT. Esto se observa en la figura 3-30. La forma OR-NAND se parece a la OR-AND, excepto por la inversión efectuada por la burbuja de la compuerta NAND. Esta forma implementa la función

$$F = [(A + B)(C + D)E]'$$



a) OR-NAND b) OR-NAND c) NOR-OR

FIGURA 3-30
Circuitos OR-AND-INVERT; $F = [(A + B)(C + D)E]'$

Tabla 3-3
Implementación con otras formas de dos niveles

Forma no degenerada equivalente		Implementa la función	Simplificar F' en	Para obtener como salida
a)	b)*			
AND-NOR	NAND-AND	AND-OR-INVERT	Suma de productos combinando los ceros del mapa	F
OR-NAND	NOR-OR	OR-AND-INVERT	Producto de sumas combinando los unos del mapa y complementando después	F

* La forma b) requiere un inversor en los términos de una sola literal.

Si se utiliza el símbolo gráfico alterno para la compuerta NAND, se obtiene el diagrama de la figura 3-30b). El circuito de c) se obtiene pasando las burbujas de las entradas de la compuerta de segundo nivel a las salidas de las compuertas de primer nivel. El circuito de la figura 3-30c) es una forma NOR-OR y en la figura 3-28 se demostró que implementa la función OR-AND-INVERT.

La implementación OR-AND-INVERT requiere una expresión en forma de producto de sumas. Si el complemento de la función se simplifica en producto de sumas, se podrá implementar F' con la parte OR-AND de la función. Una vez que F' pase por la parte INVERT, tendremos el complemento de F', o F, en la salida.

Resumen tabular y ejemplo

En la tabla 3-3 se resumen los procedimientos para implementar una función booleana en cualquiera de las cuatro formas de dos niveles. Debido a la parte INVERT en todos los casos, conviene usar la simplificación de la función, F' (el complemento). Al implementar F' en una de estas formas, se obtiene el complemento de la función en la forma AND-OR u OR-AND. Las cuatro formas de dos niveles invierten esta función para dar como salida el complemento de F', que es la salida normal F.

EJEMPLO 3-11

Implemente la función de la figura 3-31a) con las cuatro formas de dos niveles presentadas en la tabla 3-3.

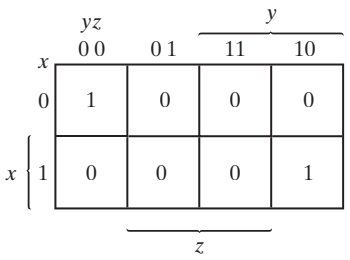
El complemento de la función se simplifica en forma de suma de productos combinando los ceros del mapa:

$$F' = x'y + xy' + z$$

La salida normal de esta función se expresa así:

$$F = (x'y + xy' + z)'$$

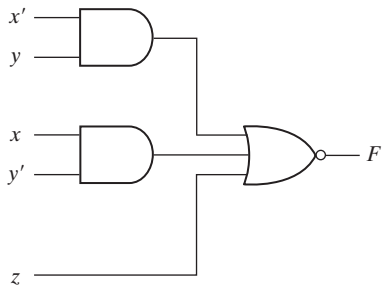
que está en la forma AND-OR-INVERT. En la figura 3-31b) se muestran las implementaciones AND-NOR y NAND-AND. Advierta que se necesita una compuerta NAND de una sola en-



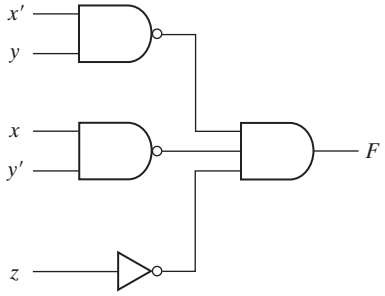
$F = x'y'z' + xyz'$

$F' = x'y + xy' + z$

a) Simplificación por mapa en forma de suma de productos

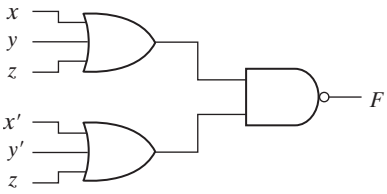


AND-NOR

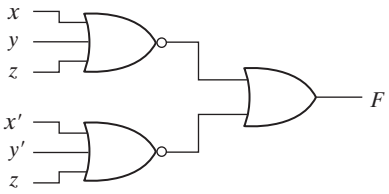


NAND-AND

b) $F = (x'y + xy' + z)'$



OR-NAND



NOR-OR

c) $F = [(x + y + z)(x' + y' + z)]'$

FIGURA 3-31
Otras implementaciones de dos niveles

trada (un inversor) en la implementación NAND-AND, pero no en el caso AND-NOR. Es posible quitar ese inversor si se aplica la variable de entrada z' en lugar de z.

Las formas OR-AND-INVERT requieren una expresión simplificada del complemento de la función en forma de producto de sumas. Para obtener esta expresión, primero hay que combinar los unos del mapa:

$$F = x'y'z' + xyz'$$

Luego se obtiene el complemento de la función

$$F' = (x + y + z)(x' + y' + z)$$

Ahora se expresa la salida normal de la función, F, en la forma

$$F = [(x + y + z)(x' + y' + z)]'$$

que está en la forma OR-AND-INVERT. A partir de esta expresión, se implementa la función en las formas OR-NAND y NOR-OR, como se indica en la figura 3-31c).

3-8 FUNCIÓN OR EXCLUSIVO

La función OR exclusivo (XOR), denotada por el símbolo $\oplus$, es una operación lógica que efectúa la operación booleana siguiente:

$$x \oplus y = xy' + x'y$$

Es igual a 1 si sólo x es igual a 1 o sólo y es igual a 1, pero no si ambas son 1. El NOR exclusivo, también llamado equivalencia, realiza la operación booleana siguiente:

$$(x \oplus y)' = xy + x'y'$$

Es igual a 1 si tanto x como y son 1 o si ambas son 0. Se puede demostrar que el NOR exclusivo es el complemento del OR exclusivo con la ayuda de una tabla de verdad o por manipulación algebraica:

$$(x \oplus y)' = (xy' + x'y)' = (x' + y)(x + y') = xy + x'y'$$

Se cumplen las identidades siguientes para la operación de OR exclusivo:

$$\begin{aligned} x \oplus 0 &= x \\ x \oplus 1 &= x' \\ x \oplus x &= 0 \\ x \oplus x' &= 1 \\ x \oplus y' &= x' \oplus y = (x \oplus y)' \end{aligned}$$

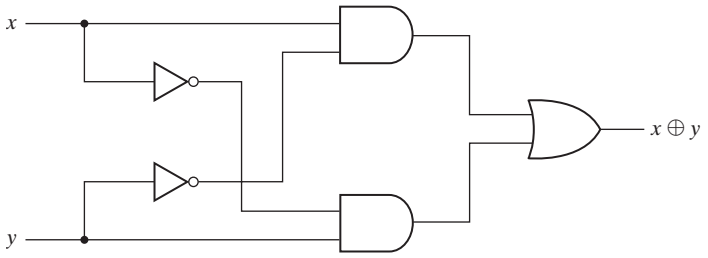
Es factible demostrar cualquiera de estas identidades con una tabla de verdad o sustituyendo la operación $\oplus$ por su expresión booleana equivalente. También puede demostrarse que la operación OR exclusivo es tanto conmutativa como asociativa; es decir,

$$A \oplus B = B \oplus A$$

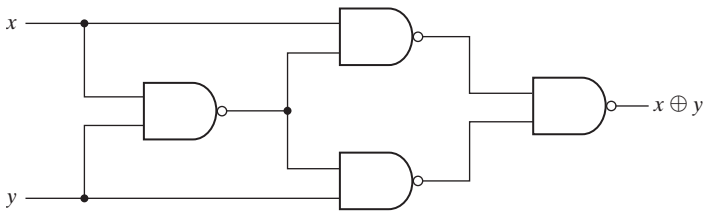
y

$$(A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C$$

Esto significa que las dos entradas de una compuerta OR exclusivo son intercambiables sin afectar a la operación. También implica que podemos evaluar una operación OR exclusivo de tres variables en cualquier orden, así que las tres o más variables se expresan sin paréntesis. Esto implicaría la posibilidad de usar compuertas OR exclusivo con tres o más entradas. Sin embargo, es difícil fabricar con hardware compuertas OR exclusivo de múltiples entradas. De hecho, incluso las funciones de dos entradas suelen construirse con otros tipos de compuertas.



a) Con compuertas AND-OR-NOT



b) Con compuertas NAND

FIGURA 3-32
Implementaciones del OR exclusivo

Construimos una función OR exclusivo de dos entradas a partir de compuertas convencionales usando dos inversores, dos compuertas AND y una compuerta OR, como se indica en la figura 3-32a). La figura 3-32b) muestra la implementación del OR exclusivo con cuatro compuertas NAND. La primera compuerta NAND efectúa la operación $(xy)' = (x' + y')$. El otro circuito NAND de dos niveles produce la suma de productos de sus entradas:

$$(x' + y')x + (x' + y')y = xy' + x'y = x \oplus y$$

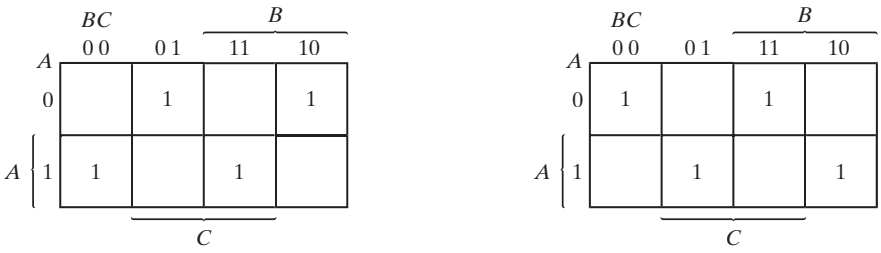
Son pocas las funciones booleanas que se pueden expresar en términos de operaciones OR exclusivo. No obstante, esta función surge con mucha frecuencia durante el diseño de sistemas digitales. Tiene especial utilidad en operaciones aritméticas y en circuitos para detectar y corregir errores.

Función impar

La operación OR exclusivo con tres o más variables se convierte en una función booleana ordinaria sustituyendo el símbolo $\oplus$ por su expresión booleana equivalente. En particular, el caso de tres variables se puede convertir en una expresión booleana así:

$$\begin{aligned} A \oplus B \oplus C &= (AB' + A'B)C' + (AB + A'B')C \\ &= AB'C' + A'BC' + ABC + A'B'C \\ &= \Sigma(1, 2, 4, 7) \end{aligned}$$

La expresión booleana indica claramente que la función OR exclusivo de tres variables es igual a 1 si sólo una variable es 1 o si las tres variables son 1. A diferencia del caso de dos variables,



a) Función impar
$F = A \oplus B \oplus C$

b) Función par
$F = (A \oplus B \oplus C)'$

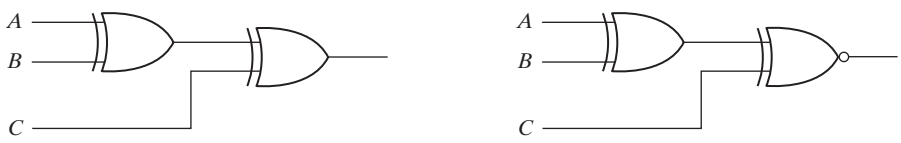
FIGURA 3-33
Mapa para una función OR exclusivo de tres variables

en el que sólo una variable debe ser igual a 1, en el caso de tres o más variables el requisito es que un número impar de variables sea igual a 1. Por consiguiente, la operación OR exclusivo de múltiples variables se define como *función impar*.

La función booleana obtenida de la operación OR exclusivo de tres variables se expresa como la suma lógica de cuatro minitérminos cuyos valores numéricos binarios son 001, 010, 100 y 111. Todos estos números binarios tienen un número impar de unos. Los otros cuatro minitérminos no incluidos en la función son 000, 011, 101 y 110, y tienen un número par de unos en su valor numérico binario. En general, una función OR exclusivo de n variables es una función impar definida como la suma lógica de los $2^n/2$ minitérminos cuyos valores numéricos binarios tienen un número impar de unos.

La definición de función impar queda más clara si se grafica en un mapa. La figura 3-33a) muestra el mapa para la función OR exclusivo de tres variables. Los cuatro minitérminos de la función están separados por una distancia unitaria. La función impar se identifica a partir de los cuatro minitérminos cuyos valores binarios tienen un número impar de unos. El complemento de una función impar es una función par. Como se aprecia en la figura 3-33b), la función par de tres variables es 1 cuando un número par de variables es igual a 1 (incluida la condición en la que ninguna de las variables es igual a 1).

La función impar de tres entradas se implementa con compuertas OR exclusivo de dos entradas, como se observa en la figura 3-34a). El complemento de una función impar se obtiene sustituyendo la compuerta de salida por una compuerta NOR exclusivo, como se indica en la figura 3-34b).



a) Función impar de tres entradas

b) Función par de tres entradas

FIGURA 3-34
Diagrama lógico de funciones impar y par

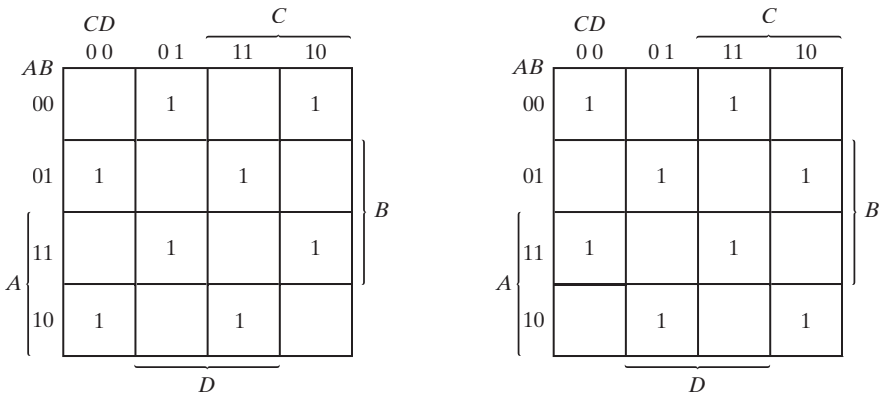


FIGURA 3-35
Mapa de una función OR exclusivo de cuatro variables

Considere ahora la operación OR exclusivo de cuatro variables. Por manipulación algebraica, se obtiene la suma de minitérminos para esta función:

$$\begin{aligned} A \oplus B \oplus C \oplus D &= (AB' + A'B) \oplus (CD' + C'D) \\ &= (AB' + A'B)(CD + C'D') + (AB + A'B')(CD' + C'D) \\ &= \sum(1, 2, 4, 7, 8, 11, 13, 14) \end{aligned}$$

Una función booleana de cuatro variables tiene 16 minitérminos. La mitad de ellos tiene valores numéricos binarios con un número impar de unos; la otra mitad tiene valores numéricos binarios con un número par de unos. Al graficar la función en el mapa, el valor numérico binario de un minitérmino se deduce de los números de fila y columna del cuadrado que representa al minitérmino. El mapa de la figura 3-35a) corresponde a la función OR exclusivo de cuatro variables. Es una función impar porque los valores binarios de todos los minitérminos tienen un número impar de unos. El complemento de una función impar es una función par. Como se observa en la figura 3-35b), la función par de cuatro variables es igual a 1 cuando un número par de variables es igual a 1.

Generación y verificación de paridad

Las funciones OR exclusivo son muy útiles en los sistemas que requieren códigos para detectar y corregir errores. Como se explicó en la sección 1-7, se utiliza un bit de paridad para detectar errores durante la transmisión de información binaria. Un bit de paridad es un bit adicional que se incluye con el mensaje binario de modo que el número total de unos sea impar o par. El mensaje, con el bit de paridad incluido, se transmite y luego se verifica en el extremo receptor para comprobar que no haya habido errores. Se detecta un error si la paridad recibida no corresponde con la transmitida. El circuito que genera el bit de paridad en el transmisor se denomina *generador de paridad*. El que comprueba la paridad en el receptor se llama *verificador de paridad*.

Tabla 3-4
Tabla de verdad de un generador de paridad par

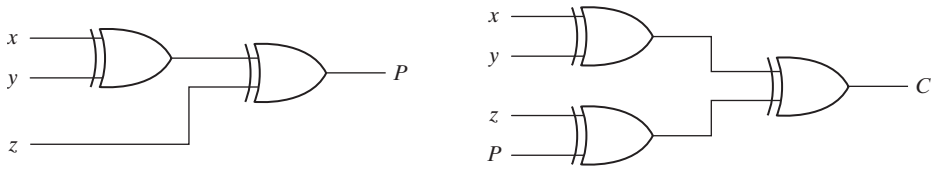
Mensaje de tres bits			Bit de paridad
x	***y***	***z***	***P***
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Como ejemplo, considere un mensaje de tres bits que se transmitirá junto con un bit de paridad par. La tabla 3-4 representa la tabla de verdad para el generador de paridad. Los tres bits —*x*, *y* y *z*— constituyen el mensaje y son las entradas del circuito. La salida es el bit de paridad, *P*. Si se usa paridad par, el bit *P* generado deberá hacer que el número total de unos (incluido *P*) sea par. Por la tabla de verdad, vemos que *P* constituye una función impar porque es igual a 1 para todos los minitérminos cuyo valor numérico binario tiene un número impar de unos. Por tanto, *P* se expresa como una función OR exclusivo de tres variables:

$$P = x \oplus y \oplus z$$

El diagrama lógico del generador de paridad se muestra en la figura 3-36a).

Los tres bits del mensaje, junto con el bit de paridad, se transmiten a su destino, donde se aplican a un circuito verificador de paridad para detectar posibles errores en la transmisión. Puesto que la información se transmitió con paridad par, los cuatro bits recibidos deberán tener un número par de unos. Si los cuatro bits recibidos tienen un número impar de unos, querrá decir que hubo un error de transmisión, pues por lo menos un bit habrá cambiado de valor durante la transmisión. La salida del verificador de paridad, denotada por *C*, será igual a 1 si hubo un error, es decir, si los cuatro bits recibidos tienen un número impar de unos. La tabla 3-5 es la tabla de verdad del verificador de paridad par. En ella se advierte que la función *C* consiste en los ocho minitérminos cuyo valor numérico binario posee un número impar de unos. Esto



a) Generador de paridad par de tres bits

b) Verificador de paridad par de cuatro bits

FIGURA 3-36
Diagrama lógico de un generador y un verificador de paridad

Tabla 3-5
Tabla de verdad de un verificador de paridad par

Cuatro bits recibidos				Verificador de errores de paridad
x	***y***	***z***	***P***	**C**
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

corresponde al mapa de la figura 3-35a), que representa una función impar. El verificador de paridad se implementa con compuertas OR exclusivo:

$$C = x \oplus y \oplus z \oplus P$$

El diagrama lógico del verificador de paridad se muestra en la figura 3-36b).

Vale la pena señalar que el generador de paridad se implementa con el circuito de la figura 3-36b) si la entrada P se conecta a 0 lógico y la salida se designa como P. Ello se debe a que $z \oplus 0 = z$, con lo que el valor de z pasa inalterado por la compuerta. La ventaja de esto es que se puede usar el mismo circuito tanto para generar como para verificar la paridad.

Por el ejemplo anterior, es obvio que los circuitos para generar y verificar paridad siempre tienen una función de salida que incluye la mitad de los minitérminos, aquellos cuyo valor numérico tiene un número impar (o par) de unos. Por consiguiente, se les puede implementar con compuertas OR exclusivo. Una función con un número par de unos es el complemento de una función impar y se implementa con compuertas OR exclusivo, salvo que la compuerta asociada con la salida debe ser un NOR exclusivo para realizar la complementación requerida.

3-9 LENGUAJE DE DESCRIPCIÓN DE HARDWARE (HDL)

Los lenguajes de descripción de hardware son lenguajes que describen el hardware de los sistemas digitales en forma textual. Se parecen a los lenguajes de programación, pero están orientados específicamente a la descripción de las estructuras y el comportamiento del hardware.