# FUNDAMENTOS DE LÓGICA DIGITAL CON DISEÑO VHDL

**SEGUNDA EDICIÓN**


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

**Traducción**
**Lorena Peralta Rosales**
**Víctor Campos Olguín**
*Traductores profesionales*

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

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




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

capítulo

# 2

# Introducción a los circuitos lógicos

## Objetivos del capítulo

En este capítulo se estudian los temas siguientes:

- Las funciones y los circuitos lógicos
- El álgebra booleana para manejar las funciones lógicas
- Las compuertas lógicas y la síntesis de circuitos simples
- Las herramientas CAD y el lenguaje VHDL de descripción de hardware

Los circuitos lógicos se estudian principalmente porque se usan en las computadoras digitales. Sin embargo, también constituyen la base de muchos otros sistemas digitales en los que la realización de operaciones aritméticas con números no reviste especial interés. Por ejemplo, en múltiples aplicaciones de control las acciones están determinadas por algunas operaciones lógicas sencillas sobre la información que entra, sin necesidad de hacer muchos cálculos numéricos.

Los circuitos lógicos realizan operaciones con señales digitales y casi siempre se implementan como circuitos electrónicos donde los valores de la señal se restringen a algunos valores discretos. En los circuitos lógicos *binarios* sólo hay dos valores, 0 y 1. En los circuitos lógicos *decimales* hay 10 valores, de 0 a 9. Puesto que el valor de cada señal se representa naturalmente con un dígito, estos circuitos lógicos reciben el nombre de *circuitos digitales*. En contraste, existen los *circuitos analógicos* en los que las señales pueden adquirir una gama discontinua de valores entre un nivel mínimo y uno máximo.

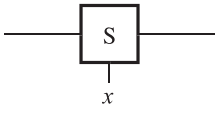
En esta obra se estudian los circuitos binarios, los cuales dominan en la tecnología digital. Esperamos brindar al lector una exposición comprensible de su funcionamiento, de cómo se representan en notación matemática y cómo se diseñan mediante modernas técnicas de diseño automatizado. Empezaremos con la definición de ciertos conceptos básicos relativos a los circuitos lógicos binarios.

## 2.1 VARIABLES Y FUNCIONES

Los circuitos binarios predominan en los sistemas digitales gracias a su simplicidad, que resulta de restringir las señales para que adopten sólo dos valores posibles. El elemento binario más sencillo es un interruptor de dos estados. Si una variable de entrada $x$ controla un interruptor, entonces se dice que éste se abre si $x = 0$ y se cierra si $x = 1$, como se ilustra en la figura 2.1*a*. Usaremos el símbolo gráfico de la figura 2.1*b* para representar este tipo de interruptores en los diagramas que siguen. Nótese que la entrada de control $x$ se muestra explícitamente en el símbolo. En el capítulo 3 se explica cómo implementar estos interruptores con transistores.

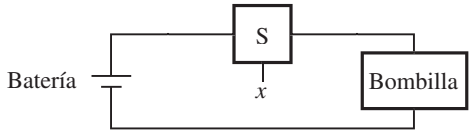
Considérese una aplicación simple de un interruptor, donde éste enciende o apaga una pequeña bombilla. Esta acción se logra con el circuito de la figura 2.2*a*. Una batería proporciona la fuente de poder. La bombilla brilla cuando pasa la corriente necesaria por su filamento, que es una resistencia eléctrica. La corriente fluye cuando el interruptor se cierra; es decir, cuando


*a*) Dos estados de un interruptor

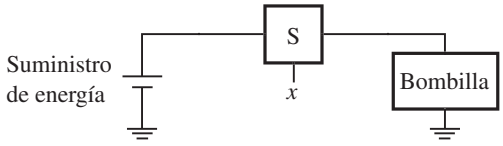



*b*) Símbolo de un interruptor

**Figura 2.1** Interruptor binario.




*a*) Conexión simple a una batería




*b*) Uso de una conexión a tierra como trayectoria de regreso

**Figura 2.2** Bombilla controlada mediante un interruptor.

$x = 1$. En este ejemplo la entrada que ocasiona el cambio en el comportamiento del circuito es el control $x$ del interruptor. La salida se define como el estado (o condición) de la luz, que se denotará con la letra $L$. Si la luz se enciende, diremos que $L = 1$; si se apaga, que $L = 0$. Con esta convención es posible describir el estado de la luz como función de la variable de entrada $x$. Puesto que $L = 1$ si $x = 1$ y $L = 0$ si $x = 0$ puede decirse que

$$L(x) = x$$

Esta sencilla *expresión lógica* describe la salida como función de la entrada. Se dice que $L(x) = x$ es una *función lógica* y que $x$ es una *variable de entrada*.

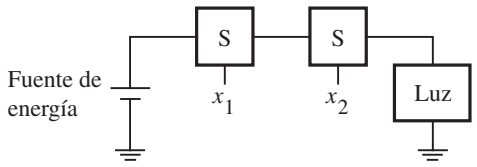
El circuito de la figura 2.2*a* se halla en una linterna ordinaria, donde el interruptor es un dispositivo mecánico sencillo. En un circuito electrónico el interruptor se implementa como un transistor y la luz puede ser un diodo emisor de luz (LED, *light-emitting diode*). Un circuito electrónico recibe la energía de una fuente de cierto voltaje, tal vez 5 voltios. Un lado de la fuente se conecta a tierra, como muestra la figura 2.2*b*. La conexión a tierra también puede usarse como la trayectoria de regreso para la corriente, a fin de cerrar el circuito, lo que se logra conectando un lado de la luz a tierra, como se indica en la figura. Desde luego la luz también puede conectarse con un cable directamente al lado aterrizado de la fuente de poder, como se advierte en la figura 2.2*a*.

Considérese ahora la posibilidad de usar dos interruptores para controlar el estado de la luz. Sean $x_1$ y $x_2$ sus entradas de control. Los interruptores pueden conectarse en serie o en paralelo, como se muestra en la figura 2.3. Si se usa conexión en serie la luz se encenderá sólo si ambos interruptores están cerrados. Si uno está abierto, la luz estará apagada. Este comportamiento puede describirse con la expresión

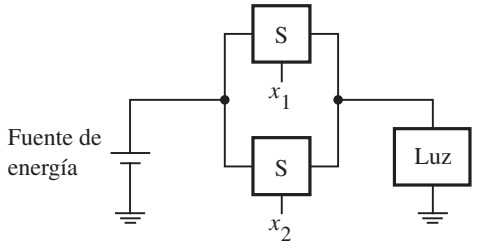
$$L(x_1, x_2) = x_1 \cdot x_2$$

donde $L = 1$ si $x_1 = 1$ y $x_2 = 1$,

$L = 0$ de otro modo.




*a*) La función lógica AND (conexión en serie)




*b*) La función lógica OR (conexión en paralelo)

**Figura 2.3** Dos funciones básicas.

El símbolo "·" es el *operador AND*, y se dice que el circuito de la figura 2.3*a* implementa la *función lógica AND*.

En la figura 2.3*b* se presenta la conexión en paralelo de dos interruptores. En este caso la luz se encenderá si cualquiera de los interruptores, $x_1$ o $x_2$, se cierra, o si ambos se cierran. La luz se apagará sólo si los dos interruptores están abiertos. Este comportamiento puede expresarse como

$$\begin{aligned} & L(x_1, x_2) = x_1 + x_2 \\ \text{donde} \quad & L = 1 \text{ si } x_1 = 1 \text{ o } x_2 = 1 \text{ o si } x_1 = x_2 = 1, \\ & L = 0 \text{ si } x_1 = x_2 = 0. \end{aligned}$$

El símbolo + es el *operador OR* y se dice que el circuito de la figura 2.3*b* implementa la *función lógica OR*.

En las expresiones anteriores para AND y OR, la salida $L(x_1, x_2)$ es una función lógica con variables de entrada $x_1$ y $x_2$. Las funciones AND y OR son dos de las funciones lógicas más importantes. Junto con algunas otras funciones simples se usan como los bloques fundamentales de la implementación de todos los circuitos lógicos. En la figura 2.4 se muestra cómo usar tres interruptores para controlar la luz de forma más compleja. Esta conexión serie-paralelo de interruptores realiza la función lógica

$$L(x_1, x_2, x_3) = (x_1 + x_2) \cdot x_3$$

La luz se enciende si $x_3 = 1$ y, al mismo tiempo, al menos una de las entradas $x_1$ o $x_2$ es igual a 1.

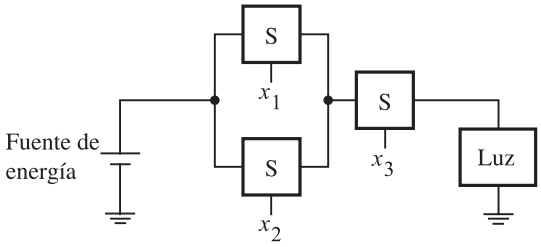



**Figura 2.4** Conexión serie-paralelo.

## 2.2 INVERSIÓN

Hasta el momento hemos supuesto que cierta acción positiva, como encender la luz, tiene lugar cuando se cierra un interruptor. Es igualmente interesante y útil considerar la posibilidad de que suceda una acción positiva cuando se abre un interruptor. Supóngase que conectamos la luz como se muestra en la figura 2.5. En este caso el interruptor se conecta en paralelo con la luz, en lugar de en serie. En consecuencia un interruptor cerrado ocasionará un cortocircuito y evitará que la corriente pase por él. Nótese que hemos incluido en este circuito un resistor adicional para garantizar que el interruptor cerrado no causará un cortocircuito en la fuente de energía. La luz se encenderá cuando el interruptor se abra. Formalmente, este comportamiento funcional se expresa como

$$L(x) = \overline{x}$$

$$\text{donde} \quad L = 1 \text{ si } x = 0,$$

$$L = 0 \text{ si } x = 1.$$

El valor de esta función es el inverso del valor de la variable de entrada. En lugar de utilizar la palabra *inverso*, es más común usar el término *complemento*. Por tanto se dice que $L(x)$ es un complemento de $x$ en este ejemplo. Otro término empleado con frecuencia para la misma operación es *operación NOT*. Diversas notaciones se usan para indicar el complemento. En la expresión precedente se coloca una barra sobre la $x$. Quizá esta notación sea la mejor desde un ángulo visual. Sin embargo, cuando se requieren complementos en las expresiones que se escriben con

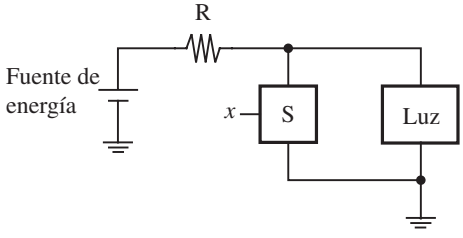



**Figura 2.5** Un circuito inversor.

el teclado de una computadora, lo que a menudo sucede cuando se emplean herramientas CAD, resulta impráctico usar barras superiores. En su lugar se coloca un apóstrofe después de la variable, o el signo de exclamación (!), o el tilde (~), o la palabra NOT frente a la variable para denotar la complementación. Por ende, las expresiones que siguen son equivalentes:

$$\overline{x} = x' = !x = \sim x = \text{NOT } x$$

La operación complemento puede aplicarse a una sola variable o a operaciones más complejas. Por ejemplo, si

$$f(x_1, x_2) = x_1 + x_2$$

entonces el complemento de $f$ es

$$\overline{f}(x_1, x_2) = \overline{x_1 + x_2}$$

Esta expresión produce el valor lógico 1 sólo cuando ni $x_1$ ni $x_2$ son iguales a 1; es decir: cuando $x_1 = x_2 = 0$. De nuevo las notaciones que siguen son equivalentes:

$$\overline{x_1 + x_2} = (x_1 + x_2)' = !(x_1 + x_2) = \sim(x_1 + x_2) = \text{NOT } (x_1 + x_2)$$

## 2.3 TABLAS DE VERDAD

Hemos presentado las tres operaciones lógicas más básicas —AND, OR y complemento— relacionándolas con circuitos sencillos construidos con interruptores. Este enfoque confiere a tales operaciones cierto "significado físico". Las mismas operaciones también pueden definirse en forma de tabla, llamada *tabla de verdad*, como se muestra en la figura 2.6. Las primeras dos columnas (a la izquierda de la línea vertical doble) proporcionan las cuatro posibles combinaciones de valores lógicos que las variables $x_1$ y $x_2$ pueden tener. La siguiente columna define la operación AND para cada combinación de valores de $x_1$ y $x_2$, y la última columna define la operación OR. Puesto que con frecuencia es necesario hacer referencia a "combinaciones de valores lógicos" aplicados a algunas variables, se adoptará un término más corto, *valoración*, para denotar tal combinación de valores lógicos.

| $x_1$ | $x_2$ | $x_1 \cdot x_2$ | $x_1 + x_2$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 |
| | | AND | OR |

**Figura 2.6** Tabla de verdad para las operaciones AND y OR.

| $x_1$ | $x_2$ | $x_3$ | $x_1 \cdot x_2 \cdot x_3$ | $x_1 + x_2 + x_3$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

**Figura 2.7** Operaciones AND y OR para tres entradas.

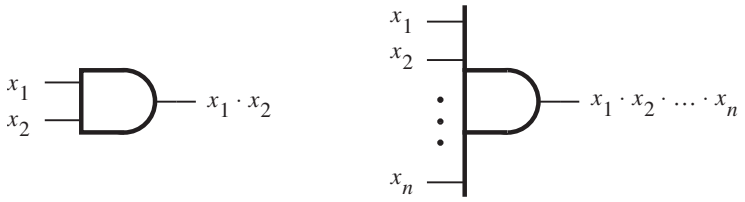
Las tablas de verdad son un auxiliar útil para describir información relacionada con funciones lógicas. En este libro se utilizan para definir funciones específicas y demostrar la validez de ciertas relaciones funcionales. Las tablas de verdad pequeñas son fáciles de manejar. Sin embargo, crecen exponencialmente en tamaño con el número de variables. Una tabla de verdad de tres variables de entrada tiene ocho filas porque hay ocho posibles valoraciones para esas variables. La figura 2.7 proporciona una tabla semejante, que define las funciones AND y OR para tres entradas. Para cuatro variables de entrada, la tabla de verdad tiene 16 filas, etc. En general, para $n$ variables de entrada, la tabla de verdad tiene $2^n$ filas.

Las operaciones AND y OR pueden extenderse a $n$ variables. Una función AND de variables $x_1, x_2, \ldots, x_n$ tiene el valor 1 sólo si todas las $n$ variables son iguales a 1. Una función OR de variables $x_1, x_2, \ldots, x_n$ tiene el valor 1 si una o más de las variables es igual a 1.

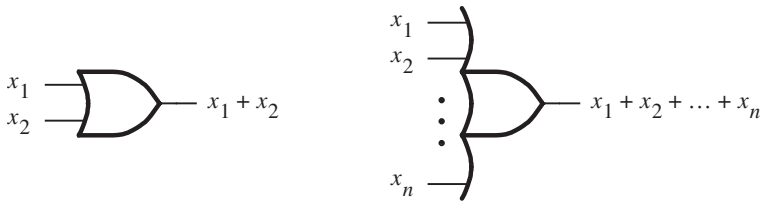
## 2.4 COMPUERTAS LÓGICAS Y CIRCUITOS

Las tres operaciones lógicas básicas expuestas en las secciones previas pueden usarse para implementar funciones lógicas de cualquier complejidad. La puesta en marcha de una función compleja puede requerir muchas de estas operaciones básicas. Cada operación lógica puede implementarse electrónicamente con transistores, lo que resulta en un elemento de circuito denominado *compuerta lógica*. Una compuerta lógica tiene una o más entradas y una salida que es función de éstas. Suele ser conveniente describir un circuito lógico trazando un diagrama del circuito, o *esquema*, que consta de símbolos gráficos que representan las compuertas lógicas. Los símbolos gráficos de las compuertas AND, OR y NOT se muestran en la figura 2.8. En el lado izquierdo se indica cómo dibujar las compuertas AND y OR cuando hay pocas entradas. En el lado derecho se muestra cómo aumentan los símbolos para dar cabida a un mayor número de entradas. En el capítulo 3 se explica la manera de construir las compuertas lógicas con transistores.

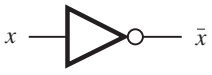
Un circuito más grande se implementa mediante una *red* de compuertas. Por ejemplo, la función lógica de la figura 2.4 puede realizarse mediante la red de la figura 2.9. La complejidad de una red tiene un efecto directo en su costo. Como siempre es deseable reducir el costo de los




*a*) Compuertas AND




*b*) Compuertas OR




*c*) Compuertas NOT

**Figura 2.8** Las compuertas básicas.

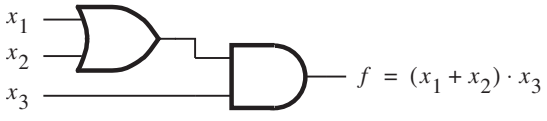



**Figura 2.9** La función de la figura 2.4.

productos fabricados, es importante encontrar formas de implementar los circuitos lógicos lo más barato posible. Páginas adelante se verá que una función lógica puede implementarse con redes diferentes, algunas de las cuales son más simples que otras; por tanto, resulta prudente buscar soluciones que representen el costo mínimo.

En el lenguaje técnico, una red de compuertas recibe el nombre de *red lógica* o, simplemente, *circuito lógico*. Utilizaremos estos términos como sinónimos.

### 2.4.1 ANÁLISIS DE UNA RED LÓGICA

Un diseñador de sistemas digitales enfrenta dos conflictos básicos. Debe ser posible determinar la función que realiza una red lógica existente. Esta tarea se conoce como proceso de *análisis*. La tarea inversa de diseñar una nueva red que desempeñe cierto comportamiento funcional se denomina proceso de *síntesis*. El proceso de análisis es más bien directo y mucho más sencillo que el de síntesis.

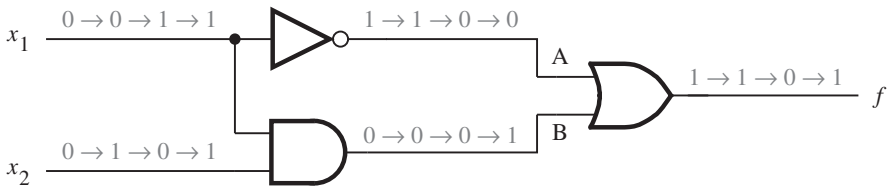
En la figura 2.10*a* se muestra una red simple formada por tres compuertas. A fin de determinar su comportamiento funcional considérese lo que ocurre si se aplican todas las señales de entrada posibles. Supóngase que se inicia $x_1 = x_2 = 0$. Esto obliga a la salida de la compuerta NOT a ser igual a 1 y a la salida de la compuerta AND a ser 0. Puesto que una de las entradas a la compuerta OR es 1, la salida de esta compuerta será 1. Por tanto, $f = 1$ si $x_1 = x_2 = 0$. Si se hace $x_1 = 0$ y $x_2 = 1$, entonces no ocurrirá cambio en el valor de *f*, pues las salidas de las compuertas NOT y AND seguirán siendo 1 y 0 respectivamente. A continuación, si se aplica $x_1 = 1$ y $x_2 = 0$, entonces la salida de la compuerta NOT cambiará a 0, mientras que la de la compuerta AND continuará siendo 0. Ambas entradas a la compuerta OR serán iguales a 0; por ende, el valor de *f* será 0. Finalmente, sea $x_1 = x_2 = 1$. Entonces la salida de la compuerta AND será 1, lo que produce que *f* sea igual a 1. La explicación verbal puede resumirse en la forma de la tabla de verdad de la figura 2.10*b*.

#### Diagramas de tiempo

El comportamiento de la red de la figura 2.10*a* se determinó al considerar los cuatro posibles valores de las entradas $x_1$ y $x_2$. Supóngase que las señales correspondientes a esas valoraciones se aplican a la red en el orden que acabamos de describir; esto es: $(x_1, x_2) = (0, 0)$, seguido de (0, 1), (1, 0) y (1, 1). Luego, los cambios en las señales en varios puntos de la red serían como se indica en gris en la figura. La misma información puede presentarse en forma gráfica, conocida como *diagrama de tiempo*, como se muestra en la figura 2.10*c*. El tiempo corre de izquierda a derecha y la valoración de cada entrada se mantiene cierto periodo fijo. En la figura se muestran las formas de onda de las entradas y salidas de la red, así como de las señales internas en los puntos *A* y *B*.

El diagrama de tiempo de la figura 2.10*c* indica que los cambios en las formas de onda de los puntos *A* y *B*, y la salida *f* tienen lugar instantáneamente cuando los valores de las entradas $x_1$ y $x_2$ cambian. Estas formas de onda idealizadas se basan en la suposición de que las compuertas lógicas responden a cambios en sus entradas en tiempo cero. Tales diagramas de tiempo son útiles para indicar el *comportamiento funcional* de los circuitos lógicos. Sin embargo, las compuertas lógicas prácticas se implementan con circuitos electrónicos que requieren cierto tiempo para cambiar sus estados. Por tanto, hay un retardo entre un cambio en los valores de entrada y el cambio correspondiente en el valor de salida de una compuerta. En capítulos próximos emplearemos diagramas de tiempo que incorporan tales retardos.

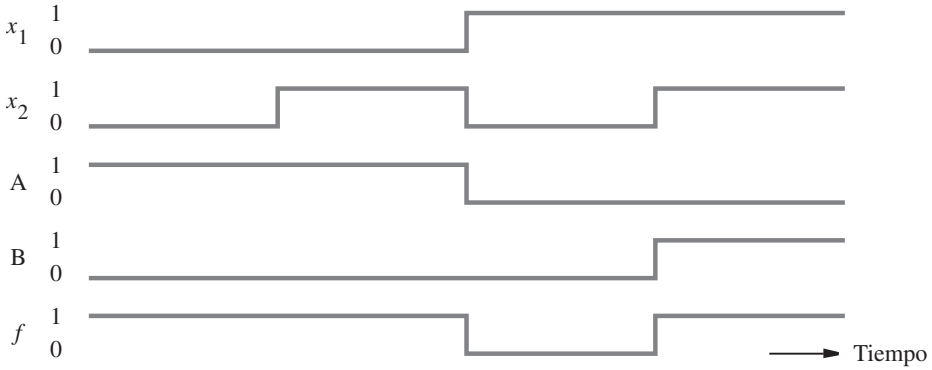
Los diagramas de tiempo sirven para muchos fines. Describen el comportamiento de un circuito lógico en una forma que es posible observar cuando se pone a prueba el circuito con instrumentos como analizadores lógicos y osciloscopios. Además a menudo se generan mediante herramientas CAD con objeto de mostrar al diseñador cómo se comporta un circuito antes de implementarlo electrónicamente en la realidad. Más adelante estudiaremos las herramientas CAD que se usarán a lo largo del libro.



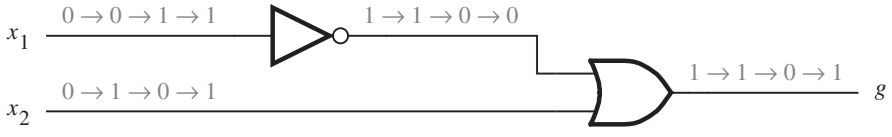

*a*) Red que implementa $f = \bar{x}_1 + x_1 \cdot x_2$

| $x_1$ | $x_2$ | $f(x_1, x_2)$ | A | B |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 |

*b*) Tabla de verdad




*c*) Diagrama de tiempo




*d*) Red que implementa $g = \bar{x}_1 + x_2$

**Figura 2.10** Ejemplo de redes lógicas.

**Redes funcionalmente equivalentes**

Considérese ahora la red de la figura 2.10*d*. Al realizar el mismo análisis se determina que la salida $g$ cambia exactamente de la misma manera en que lo hace $f$ en la parte *a*) de la figura. Por consiguiente, $g(x_1, x_2) = f(x_1, x_2)$, que indica que los dos circuitos son iguales en términos funcionales; la tabla de verdad de la figura 2.10*b* representa el comportamiento de salida de

ambos circuitos. Como los dos cumplen la misma función, es lógico utilizar la más simple, cuya implementación es menos costosa.

En general, una función lógica puede implementarse con diferentes circuitos, que quizá tengan distintos costos. Esto da lugar a una pregunta importante: ¿cómo se determina cuál es la mejor forma de implementar cierta función? Hay numerosas técnicas para sintetizar funciones lógicas. Estudiaremos los principales enfoques para ello en el capítulo 4. Por ahora cabe señalar que se precisa cierta manipulación para transformar el circuito más complejo de la figura 2.10*a* en el de la figura 2.10*d*. Puesto que $f(x_1, x_2) = \overline{x}_1 + x_1 \cdot x_2$ y $g(x_1, x_2) = \overline{x}_1 + x_2$, debe haber ciertas reglas que puedan aplicarse para demostrar la equivalencia

$$\overline{x}_1 + x_1 \cdot x_2 = \overline{x}_1 + x_2$$

Ya establecimos esta equivalencia con el análisis detallado de los dos circuitos y la elaboración de la tabla de verdad. No obstante, puede obtenerse el mismo resultado mediante manipulación algebraica de expresiones lógicas. En la siguiente sección se explicará un método matemático para tratar las funciones lógicas, el cual brinda las bases de las técnicas de diseño modernas.

## 2.5 ÁLGEBRA BOOLEANA

En 1849, George Boole publicó un esquema de la descripción algebraica de los procesos relativos al pensamiento y el razonamiento lógicos [1]. Luego ese esquema y sus posteriores refinamientos recibieron el nombre de *álgebra booleana*. Fue casi 100 años después que esta álgebra halló aplicación en la ingeniería. A fines de la década de 1930, Claude Shannon demostró que el álgebra booleana constituye un medio eficaz para describir circuitos construidos con interruptores [2]; por tanto, esta algebra sirve para describir circuitos lógicos. En este apartado veremos que el álgebra booleana constituye una poderosa herramienta para diseñar y analizar circuitos lógicos. El lector advertirá que sienta las bases de gran parte de la tecnología digital de nuestros días.

### Axiomas del álgebra booleana

Como cualquier álgebra, la booleana se basa en un conjunto de reglas derivadas a partir de un pequeño número de suposiciones fundamentales que reciben el nombre de *axiomas*. Supóngase que el álgebra booleana *B* comprende elementos que toman uno de dos valores, 0 y 1. Supóngase asimismo que los axiomas siguientes son verdaderos:

1*a*. $0 \cdot 0 = 0$
1*b*. $1 + 1 = 1$
2*a*. $1 \cdot 1 = 1$
2*b*. $0 + 0 = 0$
3*a*. $0 \cdot 1 = 1 \cdot 0 = 0$
3*b*. $1 + 0 = 0 + 1 = 1$
4*a*. Si $x = 0$, entonces $\overline{x} = 1$
4*b*. Si $x = 0$, entonces $\overline{x} = 0$

### Teoremas de una sola variable

A partir de los axiomas pueden definirse ciertas reglas para usar las variables individuales. A menudo esas reglas se denominan *teoremas*. Si $x$ es una variable en $B$, entonces se cumplen los teoremas siguientes:

5*a*. $x \cdot 0 = 0$
5*b*. $x + 1 = 1$
6*a*. $x \cdot 1 = x$
6*b*. $x + 0 = x$
7*a*. $x \cdot x = x$
7*b*. $x + x = x$
8*a*. $x \cdot \overline{x} = 0$
8*b*. $x + \overline{x} = 1$
9. $\overline{\overline{x}} = x$

Es fácil probar la validez de estos teoremas mediante inducción perfecta; es decir, mediante la sustitución de los valores $x = 0$ y $x = 1$ en las expresiones y la aplicación de los axiomas anteriores. Por ejemplo, en el teorema 5*a*, si $x = 0$ entonces el teorema afirma que $0 \cdot 0 = 0$, lo que es cierto de acuerdo con el axioma 1*a*. De manera similar, si $x = 1$ entonces el teorema 5*a* afirma que $1 \cdot 0 = 0$, que también es cierto según el axioma 3*a*. El lector debe verificar que los teoremas 5*a* a 9 pueden comprobarse de este modo.

### Dualidad

Nótese que hemos numerado por pares los axiomas y los teoremas de una sola variable. Lo hicimos para reflejar la importancia del *principio de dualidad*. Dada una expresión lógica, su *dual* se obtiene sustituyendo todos los operadores $+$ con operadores $\cdot$, y viceversa, y sustituyendo todos los 0 con 1, y viceversa. El dual de cualquier proposición verdadera (axioma o teorema) en álgebra booleana también es una proposición verdadera. Si bien en este punto de la explicación el lector no advertirá por qué la dualidad es un concepto útil, le quedará claro más adelante, cuando se muestre que la dualidad implica la existencia de al menos dos formas de expresar toda función lógica con álgebra booleana. Con frecuencia, una expresión conduce a una implementación física más simple que la otra y por tanto es preferible.

### Propiedades de dos y tres variables

Para que sea posible tratar con varias variables es útil definir algunas identidades algebraicas de dos y tres variables. Para cada una de ellas también se proporciona su versión dual. Estas identidades suelen denominarse *propiedades*. Se conocen por los nombres que se indican a continuación. Si $x$, $y$ y $z$ son las variables en $B$, entonces se cumplen las siguientes propiedades:

| | | |
|---|---|---|
| 10*a*. | $x \cdot y = y \cdot x$ | *Conmutativa* |
| 10*b*. | $x + y = y + x$ | |
| 11*a*. | $x \cdot (y \cdot z) = (x \cdot y) \cdot z$ | *Asociativa* |
| 11*b*. | $x + (y + z) = (x + y) + z$ | |
| 12*a*. | $x \cdot (y + z) = x \cdot y + x \cdot z$ | *Distributiva* |
| 12*b*. | $x + y \cdot z = (x + y) \cdot (x + z)$ | |
| 13*a*. | $x + x \cdot y = x$ | *Absorción* |

| $x$ | $y$ | $x \cdot y$ | $\overline{x \cdot y}$ | $\overline{x}$ | $\overline{y}$ | $\overline{x} + \overline{y}$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| | | LI | | LD | | |

**Figura 2.11** Prueba del teorema de DeMorgan en 15*a*.

13*b*. $x \cdot (x + y) = x$

14*a*. $x \cdot y + x \cdot \overline{y} = x$ *Combinación*

14*b*. $(x + y) \cdot (x + \overline{y}) = x$

15*a*. $\overline{x \cdot y} = \overline{x} + \overline{y}$ *Teorema de DeMorgan*

15*b*. $\overline{x + y} = \overline{x} \cdot \overline{y}$

16*a*. $x + \overline{x} \cdot y = x + y$

16*b*. $x \cdot (\overline{x} + y) = x \cdot y$

17*a*. $x \cdot y + y \cdot z + \overline{x} \cdot z = x \cdot y + \overline{x} \cdot z$ *Consenso*

17*b*. $(x + y) \cdot (y + z) \cdot (\overline{x} + z) = (x + y) \cdot (\overline{x} + z)$

De nuevo, es posible probar la validez de estas propiedades mediante inducción perfecta o por manipulación algebraica. En la figura 2.11 se indica cómo usar la inducción perfecta para demostrar el teorema de DeMorgan por medio de una tabla de verdad. La evaluación de los lados izquierdo y derecho de la identidad en 15*a* da el mismo resultado.

Hemos enumerado varios axiomas, teoremas y propiedades. No todos ellos son necesarios para definir el álgebra booleana. Por ejemplo, si suponemos que las operaciones + y · están definidas basta incluir los teoremas 5 y 8 y las propiedades 10 y 12. Éstos a veces se refieren como *postulados básicos de Huntington* [3]. El resto de las identidades puede derivarse de ellos.

Los axiomas, teoremas y propiedades anteriores proveen la información necesaria para realizar la manipulación algebraica de expresiones más complejas.

**Ejemplo 2.1**

Demostremos la validez de la ecuación lógica

$$(x_1 + x_3) \cdot (\overline{x}_1 + \overline{x}_3) = x_1 \cdot \overline{x}_3 + \overline{x}_1 \cdot x_3$$

El miembro izquierdo de la ecuación se manipula como sigue. Al aplicar la propiedad distributiva, 12*a*, se obtiene

$$\text{LI} = (x_1 + x_3) \cdot \overline{x}_1 + (x_1 + x_3) \cdot \overline{x}_3$$

Al aplicar de nuevo la propiedad distributiva se obtiene

$$\text{LI} = x_1 \cdot \overline{x}_1 + x_3 \cdot \overline{x}_1 + x_1 \cdot \overline{x}_3 + x_3 \cdot \overline{x}_3$$

Nótese que la propiedad distributiva permite aplicar la operación AND en los términos entre paréntesis en una forma análoga a la multiplicación del álgebra ordinaria. Ahora, de acuerdo con el teorema 8*a*, los términos $x_1 \cdot \overline{x}_1$ y $x_3 \cdot \overline{x}_3$ son ambos iguales a 0. Por tanto,

$$\text{LI} = 0 + x_3 \cdot \overline{x}_1 + x_1 \cdot \overline{x}_3 + 0$$

Con base en 6*b* se sigue que

$$\text{LI} = x_3 \cdot \overline{x}_1 + x_1 \cdot \overline{x}_3$$

Finalmente, al usar la propiedad conmutativa, 10*a* y 10*b*, esto se convierte en

$$\text{LI} = x_1 \cdot \overline{x}_3 + \overline{x}_1 \cdot x_3$$

que es lo mismo que el miembro derecho de la ecuación inicial.

---

**Ejemplo 2.2** Considérese la ecuación lógica

$$x_1 \cdot \overline{x}_3 + \overline{x}_2 \cdot \overline{x}_3 + x_1 \cdot x_3 + \overline{x}_2 \cdot x_3 = \overline{x}_1 \cdot \overline{x}_2 + x_1 \cdot x_2 + x_1 \cdot \overline{x}_2$$

El miembro izquierdo puede manipularse del modo siguiente

$$\begin{aligned}
\text{LI} &= x_1 \cdot \overline{x}_3 + x_1 \cdot x_3 + \overline{x}_2 \cdot \overline{x}_3 + \overline{x}_2 \cdot x_3 && \text{al aplicar } 10b \\
&= x_1 \cdot (\overline{x}_3 + x_3) + \overline{x}_2 \cdot (\overline{x}_3 + x_3) && \text{al aplicar } 12a \\
&= x_1 \cdot 1 + \overline{x}_2 \cdot 1 && \text{al aplicar } 8b \\
&= x_1 + \overline{x}_2 && \text{al aplicar } 6a
\end{aligned}$$

El miembro derecho se manipula como

$$\begin{aligned}
\text{LD} &= \overline{x}_1 \cdot \overline{x}_2 + x_1 \cdot (x_2 + \overline{x}_2) && \text{al aplicar } 12a \\
&= \overline{x}_1 \cdot \overline{x}_2 + x_1 \cdot 1 && \text{al aplicar } 8b \\
&= \overline{x}_1 \cdot \overline{x}_2 + x_1 && \text{al aplicar } 6a \\
&= x_1 + \overline{x}_1 \cdot \overline{x}_2 && \text{al aplicar } 10b \\
&= x_1 + \overline{x}_2 && \text{al aplicar } 16a
\end{aligned}$$

Al ser posible manipular ambos miembros de la ecuación inicial para llegar a expresiones idénticas se establece la validez de la ecuación. Nótese que la misma función lógica se representa mediante el miembro izquierdo o el derecho de la ecuación anterior:

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

Como resultado de la manipulación se halló una expresión mucho más simple

$$f(x_1, x_2, x_3) = x_1 + \overline{x}_2$$

que representa la misma función. Esta expresión más simple resultaría en un circuito lógico de menor costo que podría usarse para implementar la función.

---

Los ejemplos 2.1 y 2.2 ilustran el propósito de los axiomas, teoremas y propiedades como un mecanismo de manipulación algebraica. Incluso estos ejemplos simples sugieren que no es práctico tratar de esta forma con expresiones sumamente complejas. Sin embargo, dichos teoremas y propiedades ofrecen la base para automatizar la síntesis de las funciones lógicas en las herramientas CAD. Para comprender qué puede lograrse con tales herramientas el diseñador ha de estar consciente de los conceptos fundamentales.

### 2.5.1 LOS DIAGRAMAS DE VENN

Antes se sugirió que la inducción perfecta puede usarse para comprobar los teoremas y las propiedades. Este procedimiento es bastante tedioso y no muy informativo desde el punto de vista conceptual. Hay un auxiliar visual sencillo que sirve para este propósito. Se llama *diagrama de Venn* y es probable que el lector encuentre que le ofrece una explicación más intuitiva de cómo dos expresiones pueden ser equivalentes.

Tradicionalmente los diagramas de Venn se usan en matemáticas para ilustrar de modo gráfico varias operaciones y relaciones en el álgebra de conjuntos. Un conjunto $s$ es una colección de elementos que se dice son miembros de $s$. En el diagrama de Venn los elementos de un conjunto se representan mediante el contorno cerrado de una figura geométrica, digamos un cuadrado, un círculo o una elipse. Por ejemplo, en un universo $N$ de enteros de 1 a 10 el conjunto de números pares es $E = \{2, 4, 6, 8, 10\}$. Un contorno que representa a $E$ encierra los números pares. Los nones forman el complemento de $E$; por tanto, el área fuera del contorno representa $E = \{1, 3, 5, 7, 9\}$.

Como en el álgebra booleana sólo hay dos valores (elementos) en el universo, $B = \{0, 1\}$, se dice que el área dentro de un contorno que corresponde a un conjunto $s$ denota que $s = 1$, mientras que el área fuera del contorno denota que $s = 0$. En el diagrama sombrearemos el área donde $s = 1$. En la figura 2.12 se muestra el concepto del diagrama de Venn. Un cuadrado representa el universo $B$. En los incisos *a*) y *b*) de la figura se advierte la representación de las constantes 1 y 0. Un círculo representa una variable, digamos $x$, de modo que el área del círculo corresponde a $x = 1$, mientras que el área fuera del círculo corresponde a $x = 0$. Esto se ilustra en el inciso *c*). Una expresión que comprende una o más variables se describe mediante el sombreado del área donde el valor de la expresión es igual a 1. En el inciso *d*) se indica cómo representar el complemento de $x$.

Para representar dos variables, $x$ y $y$, se trazan dos círculos que se traslapan. El área de traslape representa el caso donde $x = y = 1$, es decir, el AND de $x$ y $y$, como se muestra en el inciso *e*). Puesto que esta área común se forma por las partes de $x$ y $y$ que se intersecan, la operación AND recibe formalmente el nombre de *intersección* de $x$ y $y$. En el inciso *f*) se ilustra la operación OR, donde $x + y$ representa el área total dentro de ambos círculos, o sea, donde al menos $x$ o $y$ es igual a 1. Ya que con esto se combinan las áreas de los círculos, a menudo la operación OR formalmente se llama unión de $x$ y $y$.

En el inciso *g*) se describe el término producto $x \cdot \overline{y}$, representado por la intersección del área de $x$ con la de $\overline{y}$. El inciso *h*) presenta un ejemplo de tres variables; la expresión $x \cdot y + z$ es la unión del área de $z$ con la de la intersección de $x$ y $y$.

Para ver cómo se utilizan los diagramas de Venn a fin de comprobar la equivalencia de dos expresiones demostraremos la validez de la propiedad distributiva, 12*a*, de la sección 2.5. En la figura 2.13 aparece la construcción de los miembros izquierdo y derecho de la identidad que define la propiedad

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

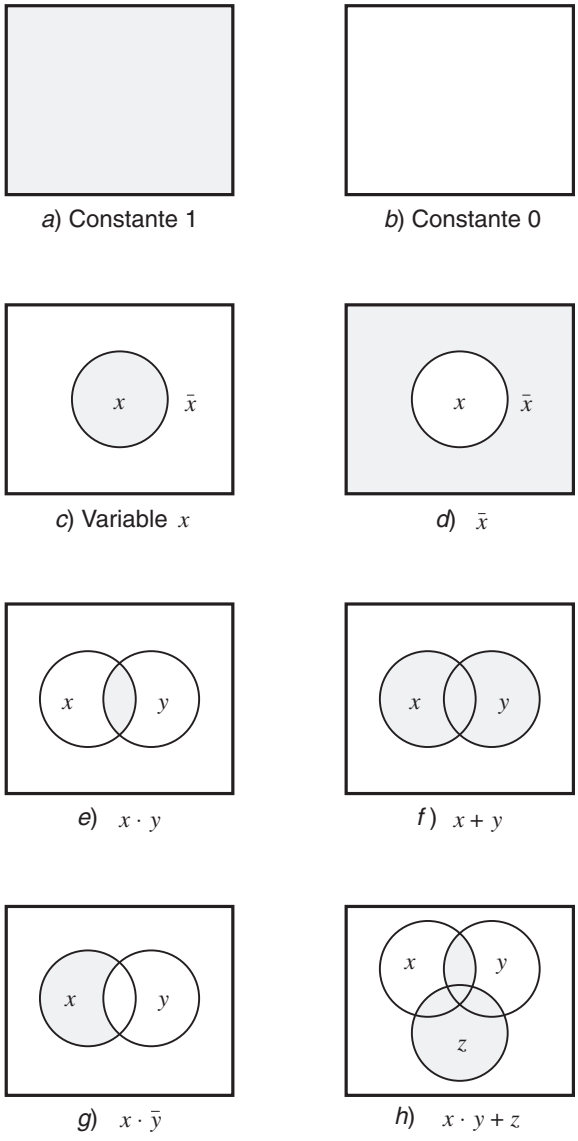



**Figura 2.12** Representación de diagramas de Venn.

El inciso *a*) muestra el área donde $x = 1$. El *b*) indica el área de $y + z$. En el *c*) se proporciona el diagrama para $x \cdot (y + z)$, la intersección de las áreas sombreadas en los incisos *a*) y *b*). El miembro derecho se construye en los incisos *d*), *e*) y *f*). Los incisos *d*) y *e*) describen los términos $x \cdot y$ y $x \cdot z$, respectivamente. La unión de las áreas sombreadas en estos dos diagramas corresponde entonces a la expresión $x \cdot y + x \cdot z$, como se observa en el inciso *f*). Como las áreas sombreadas en los incisos *c*) y *f*) son idénticas, se deduce que la propiedad distributiva es válida.

Como otro ejemplo, considérese la identidad

$$x \cdot y + \overline{x} \cdot z + y \cdot z = x \cdot y + \overline{x} \cdot z$$

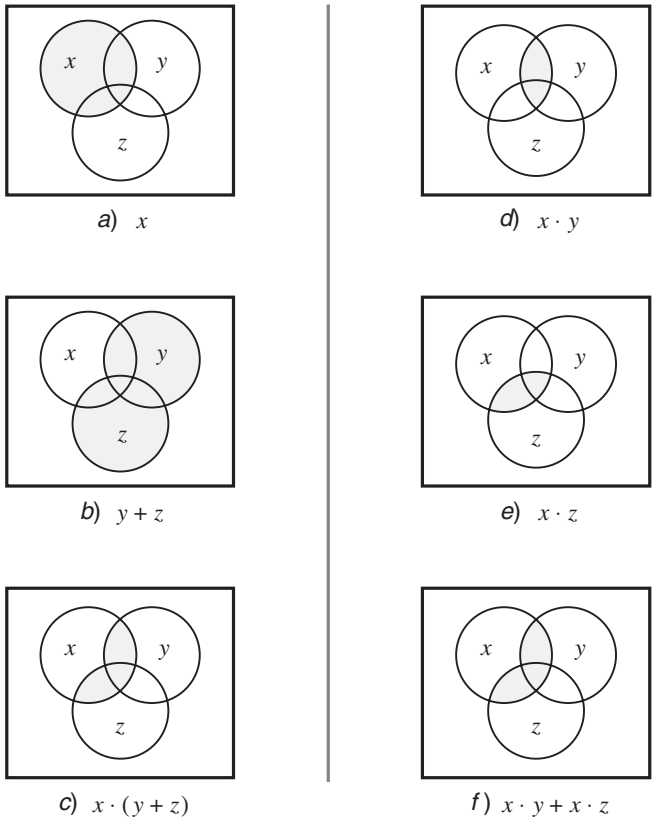



**Figura 2.13** Comprobación de la propiedad distributiva $x \cdot (y + z) = x \cdot y + x \cdot z$.

que se ilustra en la figura 2.14. Nótese que esta identidad establece que el término $y \cdot z$ está completamente cubierto por los términos $x \cdot y$ y $\overline{x} \cdot z$; en consecuencia, este término puede omitirse.

El lector debe usar los diagramas de Venn para probar algunas otras identidades. Es en particular instructivo probar la validez del teorema de DeMorgan de esta manera.

### 2.5.2 NOTACIÓN Y TERMINOLOGÍA

El álgebra booleana se basa en las operaciones AND y OR. En el texto hemos adoptado los símbolos · y + para denotarlas; se trata de los símbolos de las conocidas operaciones aritméticas de multiplicación y suma. Entre las operaciones booleanas y las aritméticas hay una similitud considerable, principal razón por la que se usan los mismos símbolos. De hecho, cuando únicamente hay dígitos solos todo se reduce a una diferencia significativa; en la aritmética ordinaria el resultado de 1 + 1 es igual a 2, mientras que el álgebra booleana es igual a 1, como lo define el teorema 7*b* de la sección 2.5.

Cuando se trabaja con circuitos digitales el símbolo + casi siempre representa la operación OR. No obstante, cuando la tarea supone el diseño de circuitos lógicos que realizan operaciones

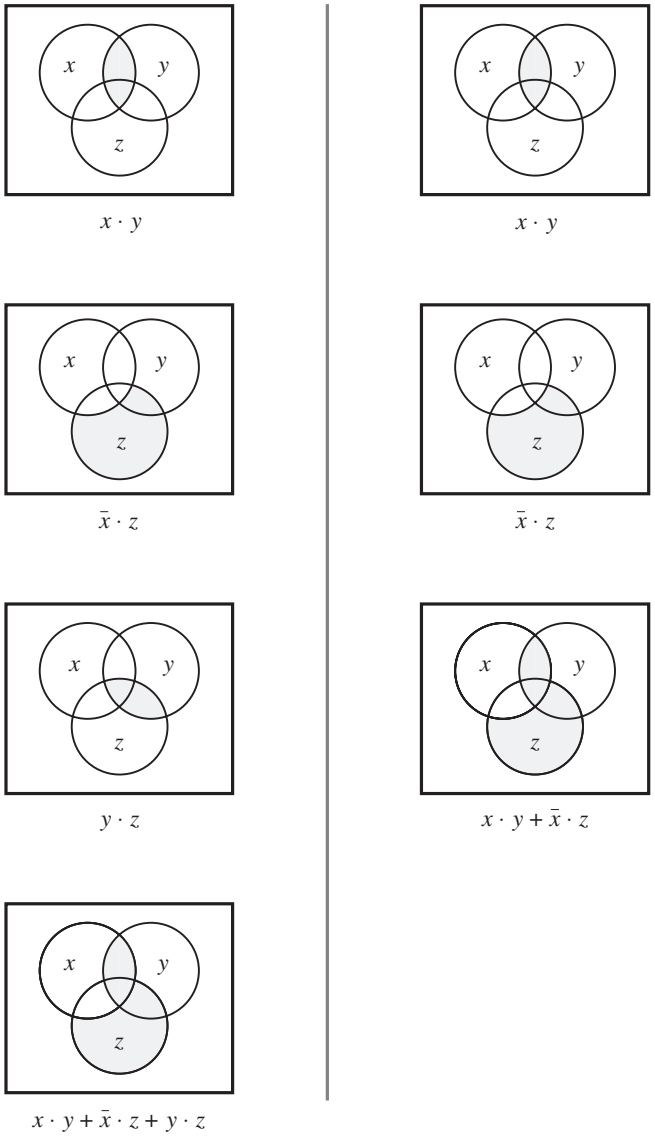



**Figura 2.14** Comprobación de $x \cdot y + \overline{x} \cdot z + y \cdot z = x \cdot y + \overline{x} \cdot z$.

aritméticas es posible que se cree cierta confusión en torno a su uso. Para evitarla existe un conjunto de símbolos diferentes para las operaciones AND y OR. Es muy común usar el símbolo $\wedge$ para denotar la operación AND y $\vee$ para la operación OR. Por ende, en lugar de $x_1 \cdot x_2$, puede escribirse $x_1 \wedge x_2$, y en vez de $x_1 + x_2$, se escribe $x_1 \vee x_2$.

Por la similitud con las operaciones de suma y multiplicación aritméticas, las operaciones OR y AND con frecuencia se denominan operaciones de *suma* y *producto lógicos*. Por tanto, $x_1 + x_2$ es la suma lógica de $x_1$ y $x_2$, y $x_1 \cdot x_2$ es el producto lógico de $x_1$ y $x_2$. En lugar de decir

"producto lógico" y "suma lógica" suele decirse simplemente "producto" y "suma". Por ende, la expresión

$$x_1 \cdot \overline{x}_2 \cdot x_3 + \overline{x}_1 \cdot x_4 + x_2 \cdot x_3 \cdot \overline{x}_4$$

es una suma de tres productos, en tanto que

$$(\overline{x}_1 + x_3) \cdot (x_1 + \overline{x}_3) \cdot (\overline{x}_2 + x_3 + x_4)$$

es un producto de tres sumas.

### 2.5.3 PRECEDENCIA DE LAS OPERACIONES

Con las tres operaciones básicas —AND, OR y NOT— es posible construir un número infinito de expresiones lógicas. Se emplean paréntesis para indicar el orden en que las operaciones deben realizarse, pero para no usarlos en exceso la precedencia de las operaciones básicas se define con otra convención. Ésta afirma que, en ausencia de paréntesis, las operaciones de una expresión lógica deben realizarse en el orden NOT, AND y luego OR. Por consiguiente, en la expresión

$$x_1 \cdot x_2 + \overline{x}_1 \cdot \overline{x}_2$$

primero hay que generar los complementos de $x_1$ y $x_2$. Después se forman los términos producto $x_1 \cdot x_2$ y $\overline{x}_1 \cdot \overline{x}_2$, seguidos por la suma de los dos términos producto. Obsérvese que sin esta convención habría que utilizar paréntesis para lograr el mismo resultado, como sigue:

$$(x_1 \cdot x_2) + ((\overline{x}_1) \cdot (\overline{x}_2))$$

Finalmente, para simplificar la presentación de las expresiones lógicas se omite el operador $\cdot$ cuando no existe ambigüedad. Por tanto, la expresión anterior puede escribirse como

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

Seguiremos este estilo a lo largo del libro.

## 2.6 LA SÍNTESIS CON COMPUERTAS AND, OR Y NOT

Con estas ideas básicas ahora podemos intentar la implementación de funciones arbitrarias mediante las compuertas AND, OR y NOT. Supóngase que queremos diseñar un circuito lógico con dos entradas, $x_1$ y $x_2$. Digamos que $x_1$ y $x_2$ representan los estados de dos interruptores, cualquiera de los cuales puede estar abierto (0) o cerrado (1). La función del circuito es revisar continuamente el estado de los interruptores $(x_1, x_2)$ y producir un valor lógico de salida 1 siempre que éstos se hallen en los estados (0, 0), (0, 1) o (1, 1). Si el estado de los interruptores es (1, 0) la salida debe ser 0. Otra manera de describir el comportamiento funcional requerido de este circuito es que la salida debe ser igual a 0 si se cierra el interruptor $x_1$ y se abre el $x_2$; de otro modo, la salida debe ser 1. El comportamiento requerido puede expresarse por medio de una tabla de verdad como la que se muestra en la figura 2.15.

Un posible procedimiento para diseñar un circuito lógico que implemente la tabla de verdad es crear un término producto que tenga un valor de 1 por cada valoración para que la función

| $x_1$ | $x_2$ | $f(x_1, x_2)$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

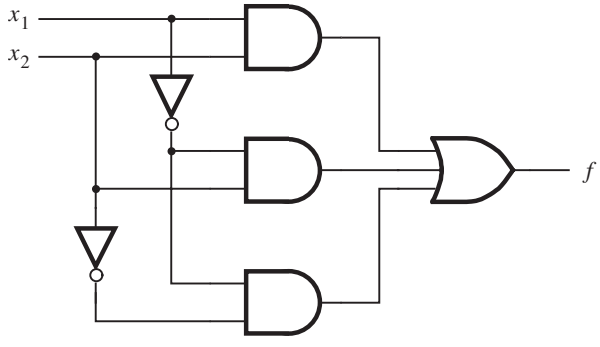
**Figura 2.15** Función que va a sintetizarse.

de salida $f$ deba ser 1. Luego podemos tomar una suma lógica de estos términos producto para realizar $f$. Empezaremos con la cuarta fila de la tabla de verdad, que corresponde a $x_1 = x_2 = 1$. El término producto que es igual a 1 para esta valoración es $x_1 \cdot x_2$, que es justo la función AND de $x_1$ y $x_2$. A continuación considérese la primera fila de la tabla, para la que $x_1 = x_2 = 0$. Para esta valoración, el valor 1 se produce por el término producto $\overline{x}_1 \cdot \overline{x}_2$. De manera similar, la segunda fila conduce al término $\overline{x}_1 \cdot x_2$. Por ende, $f$ puede realizarse como

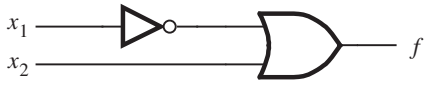
$$f(x_1, x_2) = x_1x_2 + \overline{x}_1\overline{x}_2 + \overline{x}_1x_2$$

El circuito lógico correspondiente a esta expresión se muestra en la figura 2.16*a*.

Aunque este circuito implementa $f$ correctamente, no es el más simple. Para encontrar uno más simple puede manipularse la expresión obtenida mediante los teoremas y propiedades de la sección 2.5. De acuerdo con el teorema 7*b*, es posible duplicar cualquier término de una




*a*) Suma canónica de productos




*b*) Realización de costo mínimo

**Figura 2.16** Dos implementaciones de la función de la figura 2.15.

expresión de suma lógica. Si duplicamos el tercer término producto la expresión anterior se convierte en

$$f(x_1, x_2) = x_1x_2 + \overline{x}_1\overline{x}_2 + \overline{x}_1x_2 + \overline{x}_1x_2$$

Con la propiedad conmutativa 10*b* para intercambiar los términos producto segundo y tercero se obtiene

$$f(x_1, x_2) = x_1x_2 + \overline{x}_1x_2 + \overline{x}_1\overline{x}_2 + \overline{x}_1x_2$$

Ahora la propiedad distributiva 12*a* permite escribir

$$f(x_1, x_2) = (x_1 + \overline{x}_1)x_2 + \overline{x}_1(\overline{x}_2 + x_2)$$

Al aplicar el teorema 8*b* se obtiene

$$f(x_1, x_2) = 1 \cdot x_2 + \overline{x}_1 \cdot 1$$

Por último, el teorema 6*a* conduce a

$$f(x_1, x_2) = x_2 + \overline{x}_1$$

El circuito descrito por esta expresión se presenta en la figura 2.16*b*. Resulta obvio que su costo es mucho menor que el del circuito del inciso *a*) de la figura.

Con este ejemplo sencillo se ilustran dos cosas. Primero, es posible obtener una implementación directa de una función si se usa un término producto (compuerta AND) por cada fila de la tabla de verdad para la que la función es igual a 1. Cada término producto contiene todas las variables de entrada, y se forma de tal modo que si la variable de entrada $x_i$ es igual a 1 en la fila dada, entonces se introduce $x_i$ en el término; si $x_i = 0$, entonces se introduce $\overline{x}_i$. La suma de estos términos producto realiza la función deseada. Segundo, existen muchos circuitos que pueden cumplir una función específica; algunos pueden ser más simples que otros. Mediante manipulación algebraica es posible derivar expresiones lógicas simplificadas y, por tanto, circuitos de menor costo.

El proceso mediante el cual comenzamos con una descripción del comportamiento funcional deseado y luego generamos un circuito que lo satisfaga se llama *síntesis*. En consecuencia, puede decirse que en la figura 2.16 se "sintetizaron" los circuitos a partir de la tabla de verdad de la figura 2.15. La generación de expresiones AND-OR a partir de una tabla de verdad sólo es uno de los muchos tipos de técnicas de síntesis que encontrará en esta obra.

### 2.6.1 FORMAS DE PRODUCTOS DE SUMAS Y SUMAS DE PRODUCTOS

Tras exponer el proceso de síntesis con un ejemplo muy simple, ahora lo presentaremos en términos más formales empleando la terminología de la bibliografía técnica. También mostraremos cómo aplicar el principio de dualidad, explicado en la sección 2.5, en el proceso de síntesis.

Si una función *f* se especifica en la forma de tabla de verdad, entonces es posible obtener una expresión que realice *f* considerando las filas de la tabla para las que $f = 1$, como ya se hizo, o las filas para las que $f = 0$, como veremos brevemente.

### Mintérminos

Para una función de $n$ variables, un término producto en el que cada una de las $n$ variables aparezca una vez se llama *mintérmino*. Las variables pueden aparecer en un mintérmino en forma sin complementar o en complemento. Para una fila de la tabla de verdad, el mintérmino se forma incluyendo $x_i$ si $x_i = 1$ y $\overline{x}_i$ si $x_i = 0$.

Para ilustrar este concepto considérese la tabla de verdad de la figura 2.17. Las filas están numeradas de 0 a 7, por lo que podemos hacer referencia a ellas con facilidad. (El lector familiarizado con la representación en números binarios notará que los números de fila elegidos son justo los representados por los patrones de bit de las variables $x_1$, $x_2$ y $x_3$; la representación numérica se abordará en el capítulo 5.) En la figura se muestran todos los mintérminos para la tabla de tres variables. Por ejemplo, en la primera fila las variables tienen los valores $x_1 = x_2 = x_3 = 0$, lo que conduce al mintérmino $\overline{x}_1\overline{x}_2\overline{x}_3$. En la segunda fila, $x_1 = x_2 = 0$ y $x_3 = 1$, lo cual produce el mintérmino $\overline{x}_1\overline{x}_2x_3$, etc. A fin de referir con facilidad mintérminos individuales conviene identificar cada uno de ellos mediante un índice que corresponda a los números de fila que se muestran en la figura. Emplearemos la notación $m_i$ para denotar el mintérmino para el número de fila $i$. En consecuencia, $m_0 = \overline{x}_1\overline{x}_2\overline{x}_3$, $m_1 = \overline{x}_1\overline{x}_2x_3$, etcétera.

### Forma de suma de productos

Una función $f$ puede representarse mediante una expresión que sea una suma de mintérminos, en la que a cada uno de los mintérminos se le multiplica mediante la función AND con el valor de $f$ para la valoración correspondiente de las variables de entrada. Por ejemplo, los mintérminos de dos variables son $m_0 = \overline{x}_1\overline{x}_2$, $m_1 = \overline{x}_1x_2$, $m_2 = x_1\overline{x}_2$, y $m_3 = x_1x_2$. La función de la figura 2.15 puede representarse como

$$\begin{aligned} f &= m_0 \cdot 1 + m_1 \cdot 1 + m_2 \cdot 0 + m_3 \cdot 1 \\ &= m_0 + m_1 + m_3 \\ &= \overline{x}_1\overline{x}_2 + \overline{x}_1x_2 + x_1x_2 \end{aligned}$$

que es la forma derivada en la sección anterior por medio de un enfoque intuitivo. En la expresión resultante sólo aparecen los mintérminos correspondientes a las filas para las que $f = 1$.

Cualquier función $f$ puede representarse mediante una suma de mintérminos que corresponda a las filas de la tabla de verdad para las que $f = 1$. La implementación resultante es funcional-

| Número de fila | $x_1$ | $x_2$ | $x_3$ | Mintérmino | Maxitérmino |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | $m_0 = \overline{x}_1\overline{x}_2\overline{x}_3$ | $M_0 = x_1 + x_2 + x_3$ |
| 1 | 0 | 0 | 1 | $m_1 = \overline{x}_1\overline{x}_2x_3$ | $M_1 = x_1 + x_2 + \overline{x}_3$ |
| 2 | 0 | 1 | 0 | $m_2 = \overline{x}_1x_2\overline{x}_3$ | $M_2 = x_1 + \overline{x}_2 + x_3$ |
| 3 | 0 | 1 | 1 | $m_3 = \overline{x}_1x_2x_3$ | $M_3 = x_1 + \overline{x}_2 + \overline{x}_3$ |
| 4 | 1 | 0 | 0 | $m_4 = x_1\overline{x}_2\overline{x}_3$ | $M_4 = \overline{x}_1 + x_2 + x_3$ |
| 5 | 1 | 0 | 1 | $m_5 = x_1\overline{x}_2x_3$ | $M_5 = \overline{x}_1 + x_2 + \overline{x}_3$ |
| 6 | 1 | 1 | 0 | $m_6 = x_1x_2\overline{x}_3$ | $M_6 = \overline{x}_1 + \overline{x}_2 + x_3$ |
| 7 | 1 | 1 | 1 | $m_7 = x_1x_2x_3$ | $M_7 = \overline{x}_1 + \overline{x}_2 + \overline{x}_3$ |

**Figura 2.17** Mintérminos y maxitérminos de tres variables.

mente correcta y única, pero no siempre es la implementación de *f* más barata. Se dice que una expresión lógica que consta de términos producto (AND) que se suman (con OR) es la forma de *suma de productos* (SOP, *sum-of-products*). Si cada término producto es un mintérmino, entonces la expresión es la *suma canónica de productos* para la función *f*. Como vimos en el ejemplo de la figura 2.16, el primer paso del proceso de síntesis consiste en derivar una expresión en suma canónica de productos para la función dada. Luego esa expresión puede manipularse aplicando los teoremas y las propiedades vistos en la sección 2.5, a fin de hallar una expresión funcionalmente equivalente de suma de productos que sea más barata.

Veamos otro ejemplo. Considérese la función de tres variables $f(x_1, x_2, x_3)$, especificada mediante la tabla de verdad de la figura 2.18. Para sintetizar esta función hay que incluir los mintérminos $m_1$, $m_4$, $m_5$ y $m_6$. Al copiar éstos de la figura 2.17 se desemboca en la siguiente expresión de suma canónica de productos para *f*

$$f(x_1, x_2, x_3) = \overline{x}_1\overline{x}_2x_3 + x_1\overline{x}_2\overline{x}_3 + x_1\overline{x}_2x_3 + x_1x_2\overline{x}_3$$

Esta expresión puede manipularse como sigue

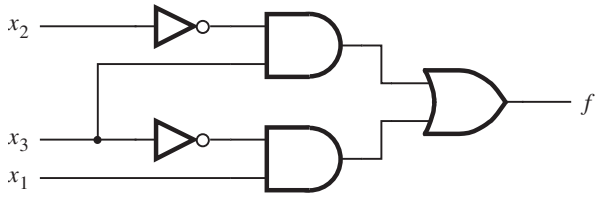
$$\begin{aligned} f &= (\overline{x}_1 + x_1)\overline{x}_2x_3 + x_1(\overline{x}_2 + x_2)\overline{x}_3 \\ &= 1 \cdot \overline{x}_2x_3 + x_1 \cdot 1 \cdot \overline{x}_3 \\ &= \overline{x}_2x_3 + x_1\overline{x}_3 \end{aligned}$$

Ésta es la expresión de suma de productos de costo mínimo para *f* y describe el circuito que se muestra en la figura 2.19*a*. Una buena indicación del costo de un circuito lógico es la cantidad total de compuertas más el número total de entradas a todas las compuertas en el circuito. Con esta medida el *costo* del circuito de la figura 2.19*a* es 13, ya que tiene cinco compuertas y ocho entradas a ellas. En comparación, el circuito implementado con base en la suma canónica de productos tendría un costo de 27; de la expresión precedente, la compuerta OR tiene cuatro entradas, cada una de las cuatro compuertas AND tiene tres entradas y cada una de las tres compuertas NOT tiene una entrada.

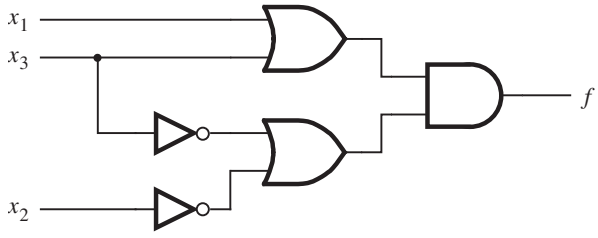
Los mintérminos, con sus subíndices correspondientes al número de fila, también pueden usarse para especificar una función de una manera más concisa.

| Número de fila | $x_1$ | $x_2$ | $x_3$ | $f(x_1\ x_2\ x_3)$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 1 | 1 | 0 |
| 4 | 1 | 0 | 0 | 1 |
| 5 | 1 | 0 | 1 | 1 |
| 6 | 1 | 1 | 0 | 1 |
| 7 | 1 | 1 | 1 | 0 |

**Figura 2.18** Función de tres variables.




*a*) Realización mínima de suma de productos




*b*) Realización mínima de productos de sumas

**Figura 2.19** Dos realizaciones de la función de la figura 2.18.

Por ejemplo, la función de la figura 2.18 puede especificarse como

$$f(x_1, x_2, x_3) = \sum(m_1, m_4, m_5, m_6)$$

o incluso de manera más sencilla como

$$f(x_1, x_2, x_3) = \sum m(1, 4, 5, 6)$$

El signo $\sum$ denota la operación suma lógica. Esta notación taquigráfica suele utilizarse en la práctica.

**Maxitérminos**

El principio de dualidad indica que si es posible sintetizar una función $f$ considerando las filas de la tabla de verdad para las que $f = 1$, entonces también debe ser posible sintetizar $f$ considerando las filas para las que $f = 0$. Este enfoque alternativo usa los complementos de los mintérminos, llamados *maxitérminos*. En la figura 2.17 se presenta una lista de todos los maxitérminos posibles para las funciones de tres variables. Haremos referencia a un maxitérmino $M_j$ mediante el mismo número de fila que su correspondiente mintérmino $m_j$, como se observa en la figura.

**Forma de producto de sumas**

Si una función $f$ se especifica mediante una tabla de verdad, entonces su complemento $\overline{f}$ puede representarse con una suma de mintérminos para los que $\overline{f} = 1$, que son las filas donde

$f = 0$. Por ejemplo, para la función de la figura 2.15

$$\begin{aligned}\overline{f}(x_1, x_2) &= m_2 \\ &= x_1\overline{x}_2\end{aligned}$$

Si esta expresión se complementa con el teorema de DeMorgan, el resultado es

$$\begin{aligned}\overline{\overline{f}} = f &= \overline{x_1\overline{x}_2} \\ &= \overline{x}_1 + x_2\end{aligned}$$

Nótese que esta expresión ya se obtuvo antes mediante la manipulación algebraica de la forma en suma canónica de productos para la función *f*. El punto clave aquí es que

$$f = \overline{m}_2 = M_2$$

donde $M_2$ es el maxitérmino para la fila 2 de la tabla de verdad.

Veamos otro ejemplo. Considérese de nuevo la función de la figura 2.18, cuyo complemento puede representarse como

$$\begin{aligned}\overline{f}(x_1, x_2, x_3) &= m_0 + m_2 + m_3 + m_7 \\ &= \overline{x}_1\overline{x}_2\overline{x}_3 + \overline{x}_1x_2\overline{x}_3 + \overline{x}_1x_2x_3 + x_1x_2x_3\end{aligned}$$

Luego, *f* se expresa como

$$\begin{aligned}f &= \overline{m_0 + m_2 + m_3 + m_7} \\ &= \overline{m}_0 \cdot \overline{m}_2 \cdot \overline{m}_3 \cdot \overline{m}_7 \\ &= M_0 \cdot M_2 \cdot M_3 \cdot M_7 \\ &= (x_1 + x_2 + x_3)(x_1 + \overline{x}_2 + x_3)(x_1 + \overline{x}_2 + \overline{x}_3)(\overline{x}_1 + \overline{x}_2 + \overline{x}_3)\end{aligned}$$

Esta expresión representa *f* como un producto de maxitérminos.

Se dice que una expresión lógica que consta de términos suma (OR) que son los factores de un producto lógico (AND) es la forma en *producto de sumas* (POS, *product-of-sums*). Si cada término suma es un maxitérmino, entonces la expresión se denomina *producto canónico de sumas* para la función dada. Cualquier función *f* puede sintetizarse encontrando su producto canónico de sumas. Ello supone tomar el maxitérmino de cada fila de la tabla de verdad para el que $f = 0$ y formar un producto de estos maxitérminos.

Volvamos al ejemplo anterior. Es posible reducir la complejidad de la expresión derivada que comprende un producto de maxitérminos. Si usamos las propiedades conmutativa, 10*b*, y asociativa, 11*b*, de la sección 2.5 podemos escribir esta expresión como

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

Luego, con la propiedad de combinación, 14*b*, la expresión se reduce a

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

El circuito correspondiente aparece en la figura 2.19*b*. Su costo es 13. Aunque resulta ser el mismo que el de la versión en suma de productos de la figura 2.19*a*, el lector no debe suponer que

el costo de un circuito derivado en la forma de suma de productos en general será igual al de un circuito correspondiente derivado en la forma de producto de sumas.

Si se utiliza la notación abreviada, una forma alternativa de especificar nuestra función de ejemplo es

$$f(x_1, x_2, x_3) = \Pi(M_0, M_2, M_3, M_7)$$

o de manera más simple

$$f(x_1, x_2, x_3) = \Pi M(0, 2, 3, 7)$$

El signo $\Pi$ denota la operación producto lógico.

En la explicación anterior se mostró cómo realizar las funciones lógicas en la forma de circuitos lógicos, que constan de circuitos de compuertas que implementan funciones básicas. Es posible realizar una función específica con circuitos de una estructura distinta, lo que casi siempre supone una diferencia en costo. Un objetivo importante para un diseñador es minimizar el costo del circuito diseñado. En el capítulo 4 estudiaremos las técnicas más relevantes para encontrar implementaciones de costo mínimo.

**Ejemplo 2.3** Considérese la función

$$f(x_1, x_2, x_3) = \sum m(2, 3, 4, 6, 7)$$

La expresión SOP canónica para la función se deriva mediante mintérminos

$$\begin{aligned} f &= m_2 + m_3 + m_4 + m_6 + m_7 \\ &= \overline{x}_1 x_2 \overline{x}_3 + \overline{x}_1 x_2 x_3 + x_1 \overline{x}_2 \overline{x}_3 + x_1 x_2 \overline{x}_3 + x_1 x_2 x_3 \end{aligned}$$

Esta expresión puede simplificarse usando las identidades de la sección 2.5 como sigue

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

**Ejemplo 2.4** Considérese de nuevo la función del ejemplo 2.3. En lugar de usar los mintérminos, podemos especificar esta función como un producto de maxitérminos para los que $f = 0$

$$f(x_1, x_2, x_3) = \Pi M(0, 1, 5)$$

Entonces, la expresión POS canónica se deriva como

$$\begin{aligned} f &= M_0 \cdot M_1 \cdot M_5 \\ &= (x_1 + x_2 + x_3)(x_1 + x_2 + \overline{x}_3)(\overline{x}_1 + x_2 + \overline{x}_3) \end{aligned}$$

Una expresión POS simplificada puede derivarse como

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

Nótese que con la propiedad distributiva, 12*b*, esta expresión conduce a

$$f = x_2 + x_1\overline{x}_3$$

que es la misma que la expresión derivada en el ejemplo 2.3.

**Ejemplo 2.5**

Supóngase que una función de cuatro variables se define mediante

$$f(x_1, x_2, x_3, x_4) = \sum m(3, 7, 9, 12, 13, 14, 15)$$

La expresión SOP canónica para esta función es

$$f = \overline{x}_1\overline{x}_2x_3x_4 + \overline{x}_1x_2x_3x_4 + x_1\overline{x}_2\overline{x}_3x_4 + x_1x_2\overline{x}_3\overline{x}_4 + x_1x_2\overline{x}_3x_4 + x_1x_2x_3\overline{x}_4 + x_1x_2x_3x_4$$

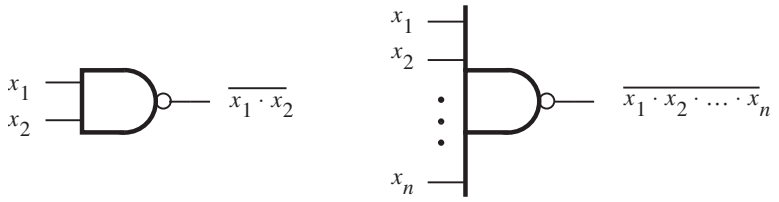
Puede obtenerse una expresión SOP más simple del modo siguiente

$$\begin{aligned} f &= \overline{x}_1(\overline{x}_2 + x_2)x_3x_4 + x_1(\overline{x}_2 + x_2)\overline{x}_3x_4 + x_1x_2\overline{x}_3(\overline{x}_4 + x_4) + x_1x_2x_3(\overline{x}_4 + x_4) \\ &= \overline{x}_1x_3x_4 + x_1\overline{x}_3x_4 + x_1x_2\overline{x}_3 + x_1x_2x_3 \\ &= \overline{x}_1x_3x_4 + x_1\overline{x}_3x_4 + x_1x_2(\overline{x}_3 + x_3) \\ &= \overline{x}_1x_3x_4 + x_1\overline{x}_3x_4 + x_1x_2 \end{aligned}$$

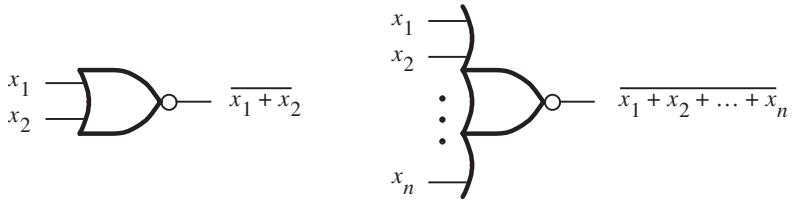
## 2.7 CIRCUITOS LÓGICOS NAND Y NOR

Ya explicamos el uso de las compuertas AND, OR y NOT en la síntesis de circuitos lógicos. Hay otras funciones lógicas básicas que sirven para el mismo propósito. Muy útiles son las funciones NAND y NOR, que se obtienen al complementar la salida generada por las operaciones AND y OR, respectivamente, y cuyo atractivo radica en que pueden implementarse con circuitos electrónicos más simples que las funciones AND y OR, como veremos en el capítulo siguiente. En la figura 2.20 se presentan los símbolos gráficos de las compuertas NAND y NOR. A fin de representar la señal de salida complementada se coloca un pequeño círculo en el lado de la salida correspondiente.

Si se realizan las compuertas NAND y NOR con circuitos más sencillos que las compuertas AND y OR, entonces cabe preguntar si pueden usarse de modo directo en la síntesis de circuitos lógicos. En la sección 2.5 expusimos el teorema de DeMorgan. En la figura 2.21 se muestra su interpretación en compuerta lógica. En el inciso *a*) de la figura se interpreta la identidad 15*a*. En ella se especifica que una función NAND de variables $x_1$ y $x_2$ equivale a complementar primero cada una de las variables y luego a aplicarles la función OR. Nótese que en el lado derecho hemos indicado las compuertas NOT simplemente como pequeños círculos, lo que denota

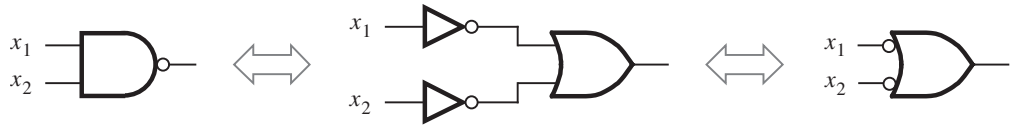



*a*) Compuertas NAND

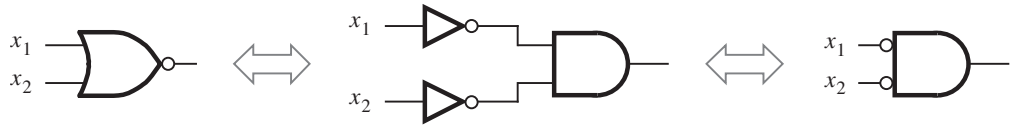



*b*) Compuertas NOR

**Figura 2.20** Compuertas NAND y NOR.




*a*) $\overline{x_1 x_2} = \bar{x}_1 + \bar{x}_2$




*b*) $\overline{x_1 + x_2} = \bar{x}_1 \bar{x}_2$

**Figura 2.21** Teorema de DeMorgan en términos de compuertas lógicas.

inversión del valor lógico en ese punto. La otra mitad del teorema de DeMorgan, la identidad 15*b*, aparece en el inciso *b*) de la figura, donde se afirma que la función NOR equivale a invertir primero las variables de entrada y luego a operarlas con la función AND.

En la sección 2.6 explicamos cómo implementar cualquier función lógica en forma de suma de productos o en forma de producto de sumas, lo que lleva a circuitos lógicos que tienen una estructura AND-OR o OR-AND respectivamente. Ahora demostraremos que tales circuitos pueden implementarse sólo con compuertas NAND o sólo con compuertas NOR.

Considérese el circuito de la figura 2.22 como representativo de los circuitos generales AND-OR. Es posible transformar este circuito en uno de compuertas NAND, como se advierte en la figura. Primero, se sustituye cada conexión entre la compuerta AND y la compuerta OR con una conexión que incluya dos inversiones de la señal: una en la salida de la compuerta AND y otra en la entrada de la compuerta OR. Esta doble inversión no tiene efecto en el comportamiento del circuito, como se afirmó formalmente en el teorema 9 de la sección 2.5. De acuerdo con la figura 2.21*a*, la compuerta OR con inversiones en sus entradas equivale a una compuerta NAND. Por tanto, el circuito puede volverse a dibujar usando únicamente compuertas NAND, como se muestra en la figura 2.22. Este ejemplo indica que es posible implementar cualquier circuito AND-OR como un circuito NAND-NAND que tenga la misma topología.

En la figura 2.23 se presenta una construcción similar para un circuito de producto de sumas, que puede transformarse en un circuito sólo con compuertas NOR. El procedimiento es exactamente el mismo que el descrito para la figura 2.22, excepto que ahora se aplica la identidad de la figura 2.21*b*. La conclusión es que cualquier circuito OR-AND puede implementarse como un circuito NOR-NOR que tenga la misma topología.

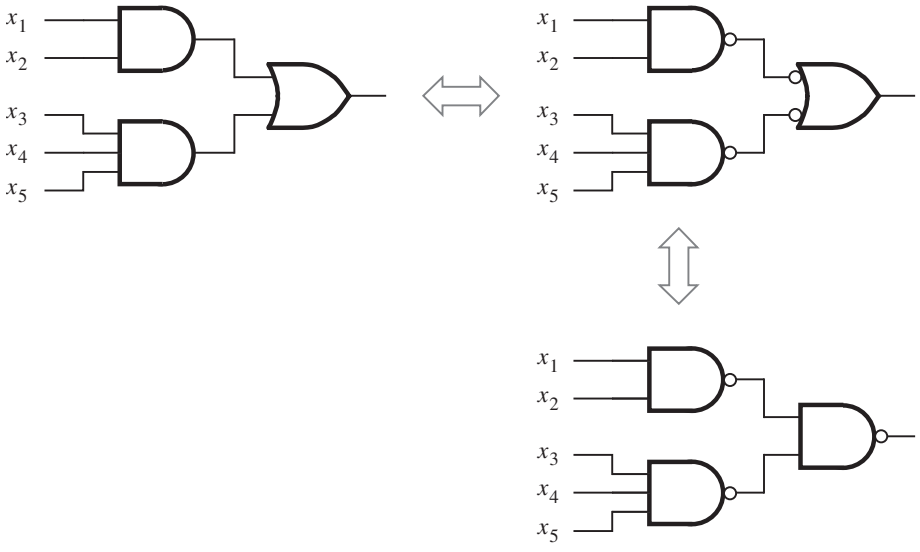



**Figura 2.22** Uso de compuertas NAND para implementar una suma de productos.

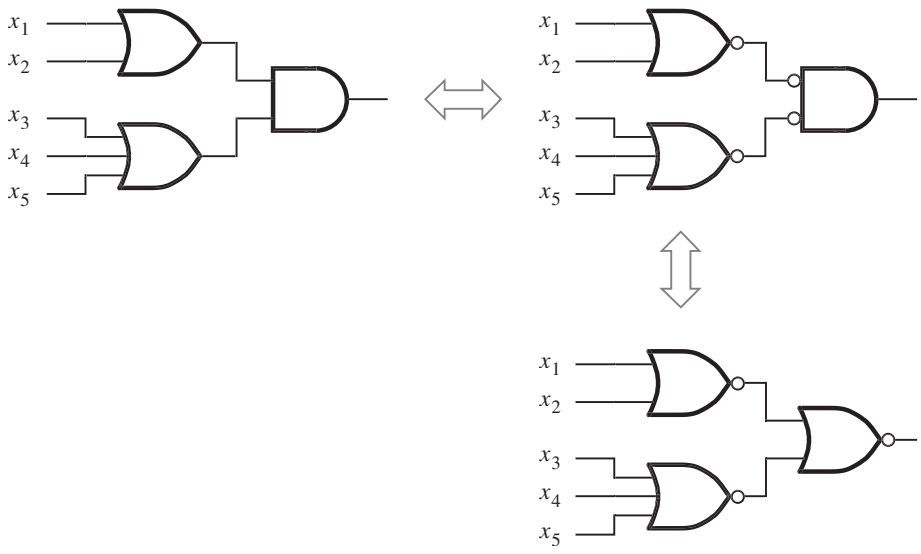



**Figura 2.23** Uso de compuerta NOR para implementar un producto de sumas.

**Ejemplo 2.6** Implemente la función

$$f(x_1, x_2, x_3) = \sum m(2, 3, 4, 6, 7)$$

empleando sólo compuertas NOR. En el ejemplo 2.4 se demostró que la función puede representarse mediante la expresión POS

$$f = (x_1 + x_2)(x_2 + \overline{x}_3)$$

En la figura 2.24*a* se observa un circuito OR-AND que corresponde a esta expresión. Con la misma estructura del circuito, la figura 2.24*b* muestra una versión en compuerta NOR. Nótese que $x_3$ se invierte mediante una compuerta NOR que tiene sus entradas unidas.

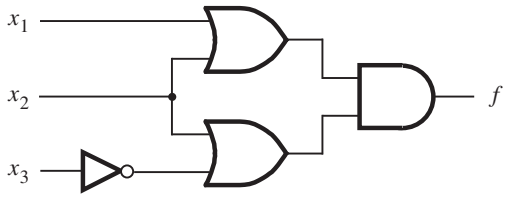
**Ejemplo 2.7** Ahora implemente la función

$$f(x_1, x_2, x_3) = \sum m(2, 3, 4, 6, 7)$$

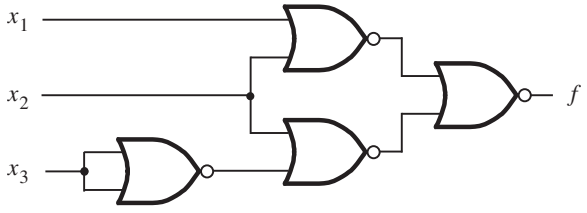
utilizando solamente compuertas NAND. En el ejemplo 2.3 derivamos la expresión SOP

$$f = x_2 + x_1\overline{x}_3$$

que se realizó por medio del circuito presentado en la figura 2.25*a*. De nuevo podemos usar la misma estructura para obtener un circuito con compuertas NAND, pero con una diferencia. La señal $x_2$ sólo pasa por una compuerta OR, en vez de hacerlo por una compuerta AND y una compuerta OR. Si nada más sustituimos la compuerta OR con una compuerta NAND, esta señal se invertirá, lo que resultará en un valor de salida equivocado. En virtud de que $x_2$ debe no invertirse

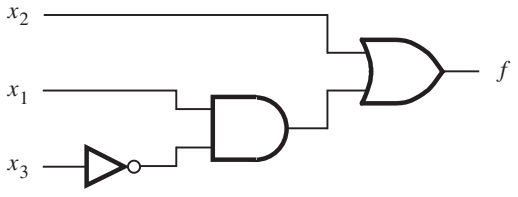



*a*) Implementación POS

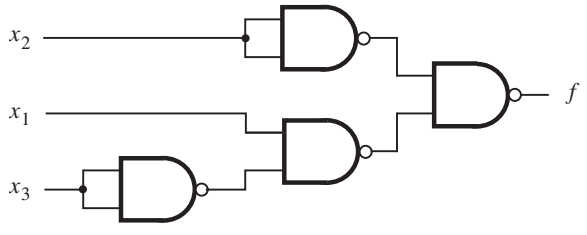



*b*) Implementación NOR

**Figura 2.24** Realización con compuertas NOR de la función del ejemplo 2.4.




*a*) Implementación SOP




*b*) Implementación NAND

**Figura 2.25** Realización con compuertas NAND de la función del ejemplo 2.3.

o invertirse dos veces, se puede pasar por dos compuertas NAND, como se muestra en la figura 2.25*b*. Obsérvese que para este circuito la salida $f$ es

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

Al aplicar el teorema de DeMorgan esta expresión se convierte en

$$f = x_2 + x_1 \overline{x}_3$$

## 2.8 EJEMPLOS DE DISEÑO

Los circuitos lógicos ofrecen una solución a un problema. Implementan funciones necesarias para llevar a cabo tareas específicas. En el marco de las computadoras, los circuitos lógicos brindan plena capacidad para la ejecución de programas y el procesamiento de datos. Tales circuitos son complejos y difíciles de diseñar. Pero sin importar su complejidad, un diseñador de circuitos lógicos siempre encara el mismo conflicto esencial. Primero ha de especificar el comportamiento deseado del circuito. Después debe sintetizarlo e implementarlo. Por último, debe probarlo a fin de verificar que cumple las especificaciones. El comportamiento deseado inicialmente se describe con palabras, que luego han de convertirse en una especificación formal. En esta sección daremos dos ejemplos sencillos de diseño.

### 2.8.1 CONTROL DE LUZ DE TRES VÍAS

Supóngase que una habitación grande tiene tres puertas y que un interruptor cerca de cada una de ellas controla la luz. Debe ser posible encenderla y apagarla mediante el cambio de estado de cualquiera de los interruptores.

Primero convirtamos este enunciado en palabras en una especificación formal por medio de una tabla de verdad. Sean $x_1$, $x_2$ y $x_3$ las variables de entrada que denotan el estado de cada interruptor. Supóngase que la luz está apagada si todos los interruptores están abiertos. Si se cierra alguno de ellos, la luz se enciende. Luego, la luz se apaga si se acciona otro. Por tanto, la luz se encenderá exactamente si un interruptor se cierra, y se apagará si dos (o ninguno) interruptores se cierran. Si la luz está apagada cuando dos interruptores están cerrados, entonces debe ser posible encenderla cerrando el tercer interruptor. Si $f(x_1, x_2, x_3)$ representa el estado de la luz, el comportamiento funcional requerido se especifica como se muestra en la tabla de verdad de la figura 2.26. La expresión en suma canónica de productos de la función especificada es

$$\begin{aligned} f &= m_1 + m_2 + m_4 + m_7 \\ &= \overline{x}_1 \overline{x}_2 x_3 + \overline{x}_1 x_2 \overline{x}_3 + x_1 \overline{x}_2 \overline{x}_3 + x_1 x_2 x_3 \end{aligned}$$

Esta expresión no puede simplificarse en una expresión de suma de productos de menor costo. En la figura 2.27*a* se presenta el circuito resultante.

| $x_1$ | $x_2$ | $x_3$ | $f$ |
|---|---|---|---|
| 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 |

**Figura 2.26** Tabla de verdad para el control de luz de tres vías.

Una realización alternativa de esta función se halla en la forma de producto de sumas, cuya expresión canónica es

$$\begin{aligned} f &= M_0 \cdot M_3 \cdot M_5 \cdot M_6 \\ &= (x_1 + x_2 + x_3)(x_1 + \overline{x}_2 + \overline{x}_3)(\overline{x}_1 + x_2 + \overline{x}_3)(\overline{x}_1 + \overline{x}_2 + x_3) \end{aligned}$$

El circuito resultante se ilustra en la figura 2.27*b*. Cuesta lo mismo que el circuito del inciso *a*) de la figura.

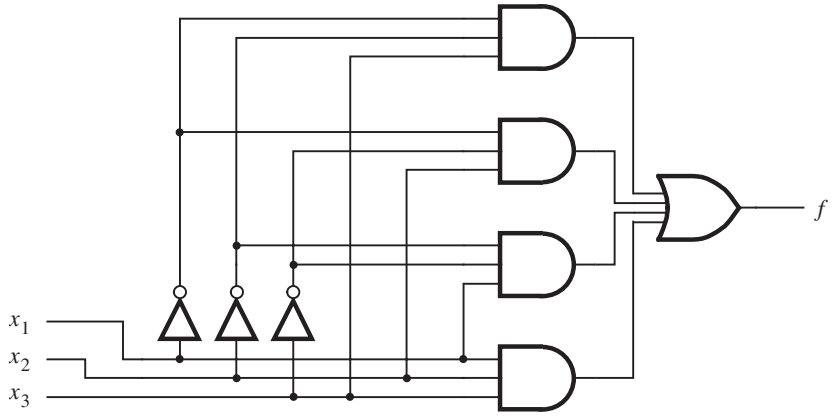
Cuando se implementa el circuito diseñado puede probarse aplicando varias valoraciones de entrada al circuito y comprobando si la salida corresponde a los valores especificados en la tabla de verdad. Un método directo consiste en comprobar que produce la salida correcta para los ocho posibles valores de entrada.

### 2.8.2 CIRCUITO MULTIPLEXOR

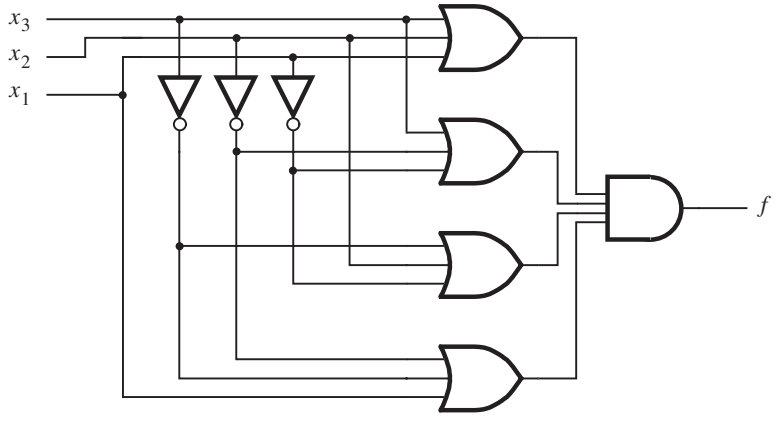
En los sistemas de cómputo a menudo es necesario elegir datos provenientes exactamente de una de varias fuentes posibles. Supóngase que existen dos fuentes de datos que proporcionan como señales de entrada $x_1$ y $x_2$. Los valores de estas señales cambian con el tiempo, quizás a intervalos regulares. Por ende, en cada una de las entradas $x_1$ y $x_2$ se aplican secuencias de 0 y 1. Queremos diseñar un circuito que produzca una salida que tenga el mismo valor que $x_1$ o $x_2$, que dependa del valor de una señal de control de selección $s$. Por tanto, el circuito debe tener tres entradas: $x_1$, $x_2$ y $s$. Digamos que la salida del circuito será la misma que el valor de entrada $x_1$ si $s = 0$, y será la misma que $x_2$ si $s = 1$.

Con base en estos requisitos podemos especificar el circuito deseado en la forma de una tabla de verdad, la de la figura 2.28*a*. A partir de ella se deriva la suma canónica de productos

$$f(s, x_1, x_2) = \overline{s}x_1\overline{x}_2 + \overline{s}x_1x_2 + s\overline{x}_1x_2 + sx_1x_2$$




*a*) Realización en suma de productos




*b*) Realización en producto de sumas

**Figura 2.27** Implementación de la función de la figura 2.26.

Si se emplea la propiedad distributiva esta expresión puede escribirse como

$$f = \overline{s}x_1(\overline{x}_2 + x_2) + s(\overline{x}_1 + x_1)x_2$$

La aplicación del teorema 8*b* produce

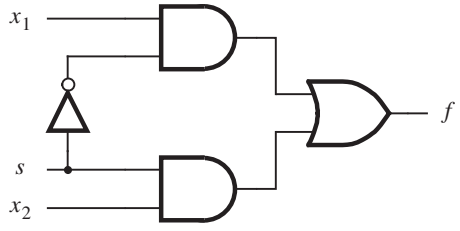
$$f = \overline{s}x_1 \cdot 1 + s \cdot 1 \cdot x_2$$

Finalmente, con el teorema 6*a* se obtiene

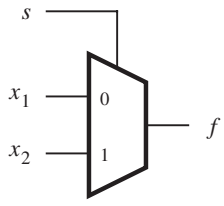
$$f = \overline{s}x_1 + sx_2$$

| $s\ x_1\ x_2$ | $f(s, x_1, x_2)$ |
|---|---|
| 0 0 0 | 0 |
| 0 0 1 | 0 |
| 0 1 0 | 1 |
| 0 1 1 | 1 |
| 1 0 0 | 0 |
| 1 0 1 | 1 |
| 1 1 0 | 0 |
| 1 1 1 | 1 |

*a*) Tabla de verdad




*b*) Circuito




*c*) Símbolo gráfico

| $s$ | $f(s, x_1, x_2)$ |
|---|---|
| 0 | $x_1$ |
| 1 | $x_2$ |

*d*) Representación más compacta de la tabla de verdad

**Figura 2.28** Implementación de un multiplexor.

En la figura 2.28*b* se muestra un circuito que implementa esta función. Los circuitos de este tipo se usan tan ampliamente que reciben un nombre especial. Un circuito que genera una salida que refleja con exactitud el estado de una de varias entradas de datos, con base en el valor de una o más entradas de control de selección, se llama *multiplexor*. Se dice que un circuito multiplexor "multiplexa" las señales de entrada en una sola salida.

En este ejemplo derivamos un multiplexor con dos entradas de datos, el cual se denomina "multiplexor 2 a 1". En la figura 2.28*c* se muestra un símbolo gráfico usado comúnmente para el multiplexor 2 a 1. La misma idea puede extenderse a circuitos más grandes. Un multiplexor 4 a 1 tiene cuatro entradas de datos y una salida. En este caso se necesitan dos entradas de control de selección para elegir una de las cuatro entradas de datos transmitidos como la señal de salida. Un multiplexor 8 a 1 precisa ocho entradas de datos y tres entradas de control de selección, etcétera.

Nótese que el enunciado "$f = x_1$ si $s = 0$, y $f = x_2$ si $s = 1$" puede presentarse en una forma más compacta de tabla de verdad, como se indica en la figura 2.28*d*. En capítulos posteriores tendremos oportunidad de usar tal representación.

Se mostró cómo es posible construir un multiplexor con compuertas AND, OR y NOT. Puede usarse la misma estructura de circuito para implementar el multiplexor con compuertas NAND, como se explica en la sección 2.7. En el capítulo 3 estudiaremos otras posibilidades para construir multiplexores, y en el 6 analizaremos de manera pormenorizada el uso de éstos.

Los diseñadores de circuitos lógicos se apoyan enormemente en las herramientas CAD. En esta obra se busca alentar al lector a conocer cuanto antes la herramienta CAD ofrecida en el libro. En este punto es útil dar una introducción a estas herramientas. En la sección siguiente se presentan algunos conceptos básicos necesarios para usarlas. Asimismo, en la sección 2.10 se expone un lenguaje especial para describir circuitos lógicos llamado VHDL, que se emplea para describir los circuitos como una entrada para las herramientas CAD, que entonces proceden a derivar una implementación adecuada.

## 2.9 INTRODUCCIÓN A LAS HERRAMIENTAS CAD

En las secciones precedentes se presentó un método básico para sintetizar circuitos lógicos. Un diseñador podría aplicarlo manualmente para circuitos pequeños. Sin embargo, los circuitos lógicos que se encuentran en sistemas complejos, como las computadoras actuales, no pueden diseñarse a mano, es preciso hacerlo con modernas herramientas CAD que implementan de forma automática las técnicas de síntesis.

Para diseñar un circuito lógico se requieren varias herramientas CAD. Casi siempre están empaquetadas en un *sistema CAD*, que por lo general incluye herramientas para las tareas siguientes: ingreso del diseño, síntesis y optimación, simulación y diseño físico. Estudiaremos algunas de estas herramientas en la presente sección y en capítulos posteriores ahondaremos en ello.

### 2.9.1 INGRESO DEL DISEÑO

El punto de partida en el proceso de diseñar un circuito lógico es la concepción de lo que se supone debe hacer éste y el planteamiento de su estructura general. Este paso lo efectúa manualmente el diseñador, pues se requiere experiencia de diseño e intuición. El resto del proceso de diseño se realiza con el auxilio de las herramientas CAD. La primera etapa de este proceso supone ingresar en el sistema CAD una descripción del circuito que se va a diseñar. Esta etapa se denomina *ingreso del diseño*. Describiremos dos métodos de ingreso de diseño: el uso de captura esquemática y la escritura de código fuente en un lenguaje de descripción de hardware.

### Captura esquemática

Un circuito lógico puede describirse dibujando las compuertas lógicas e interconectándolas con cables. La herramienta CAD para ingresar el diseño de un circuito de esta manera se llama *herramienta de captura esquemática*. La palabra *esquemática* se refiere al diagrama de un circuito en el que los elementos de éste, como las compuertas lógicas, se muestran como símbolos gráficos y las conexiones entre tales elementos se indican con líneas.

Una herramienta de captura esquemática usa las funciones gráficas de una computadora; por su parte, el ratón permite al usuario trazar un diagrama esquemático. Para facilitar la inclusión de compuertas en el esquema la herramienta ofrece un juego de símbolos gráficos que representan compuertas de varios tipos con diferentes números de entradas. Este juego de símbolos se llama *biblioteca*. Las compuertas de la biblioteca pueden importarse al esquema del usuario, y la herramienta brinda una forma gráfica de interconectarlas para crear un circuito lógico.

Es posible representar cualesquiera subcircuitos creados anteriormente como símbolos gráficos e incluirse en el esquema. En la práctica es común que el usuario de un sistema CAD cree un circuito que comprenda otros circuitos más pequeños. Este método se conoce como *diseño jerárquico* y provee una buena forma de manejar la complejidad propia de los circuitos grandes.

La herramienta de captura esquemática se describe con detalle en el apéndice B. Aunque es simple de usar, se vuelve engorrosa cuando enfrenta circuitos grandes. Un mejor método para abordar éstos es escribir código fuente mediante un lenguaje de descripción de hardware para representar el circuito.

### Lenguajes de descripción de hardware

Un *lenguaje de descripción de hardware* (HDL, *hardware description language*) es similar a un lenguaje de programación típico, salvo que el HDL sirve para describir hardware en lugar de un programa que la computadora ejecutará. Hay muchos HDL comerciales. Algunos son sujetos a un derecho de propiedad, lo que significa que los ofrece una compañía y sólo pueden usarse para implementar circuitos en la tecnología que esa compañía ofrece. En este libro no analizaremos los HDL con derechos de propiedad. En vez de ello nos centraremos en un lenguaje que apoyan prácticamente todos los comercios que ofrecen tecnología de hardware digital y oficialmente se respalda como una norma del *Instituto de Ingenieros Eléctricos* y *Electrónicos* (IEEE, *Institute of Electrical and Electronics Engineers*). El IEEE es un organismo mundial que promueve actividades técnicas para el beneficio de la sociedad. Una de ellas supone el desarrollo de normas que definan cómo usar ciertos conceptos tecnológicos de modo adecuado para un gran grupo de usuarios.

Dos HDL son normas del IEEE: VHDL (*Very High Speed Integrated Circuit Hardware Description Language*: lenguaje de descripción de hardware de circuitos integrados de muy alta velocidad) y *Verilog VHDL*. Ambos lenguajes tienen amplio uso en la industria. En esta obra emplearemos VHDL, pero la editorial [4] tiene a disposición una versión en Verilog de lo expuesto en el libro. Aunque los dos lenguajes difieren en mucho, la elección de usar uno u otro cuando estudie circuitos lógicos no reviste especial importancia porque ambos ofrecen características similares. Los conceptos ilustrados en el libro mediante VHDL pueden aplicarse directamente en Verilog.

En comparación con realizar captura esquemática, el uso de VHDL da varias ventajas. Puesto que lo apoyan la mayor parte de los organismos que ofrecen tecnología de hardware digital, VHDL brinda *portabilidad* de diseño. Un circuito especificado en VHDL puede implementarse en diferentes tipos de chips y con herramientas CAD ofrecidas por diferentes compañías, sin

necesidad de cambiar la especificación en VHDL. La portabilidad de diseño es una ventaja importante porque la tecnología de circuitos digitales cambia con rapidez. Al utilizar un lenguaje estándar el diseñador puede centrarse en la funcionalidad del circuito deseado sin preocuparse mucho por los detalles de la tecnología que a la postre usará para la implementación.

El ingreso de diseño de un circuito lógico se efectúa mediante la escritura de código VHDL. Las señales del circuito pueden representarse como variables en el código fuente, y las funciones lógicas se expresan mediante la asignación de valores a dichas variables. El código fuente de VHDL es texto llano, lo que facilita al diseñador incluir en él la documentación que explique cómo funciona el circuito. Esta característica, aunada al hecho de que VHDL se usa ampliamente, alienta a compartir y reutilizar los circuitos descritos en VHDL. Esto permite el desarrollo más rápido de productos nuevos en casos donde el código VHDL existente puede adaptarse para diseñar circuitos nuevos.

Similar al modo en que los circuitos grandes se manejan en la captura esquemática, el código VHDL puede escribirse en forma modular que facilite el diseño jerárquico. El diseño de circuitos lógicos pequeños y grandes pueden representarse eficientemente en código VHDL. Este lenguaje se usa para definir circuitos como microprocesadores con millones de transistores.

El ingreso de diseño en VHDL puede combinarse con otros métodos. Por ejemplo, es posible usar una herramienta de captura esquemática en la que un subcircuito del esquema se describa con VHDL. En la sección 2.10 se estudiará más de VHDL.

### 2.9.2 SÍNTESIS

La síntesis es el proceso por el que se genera un circuito lógico a partir de una especificación inicial que puede proporcionarse en forma de diagrama esquemático o de código escrito en un lenguaje de descripción de hardware. Con base en esa especificación las herramientas CAD de síntesis generan implementaciones eficientes de circuitos.

El proceso de traducción, o *compilación*, del código de VHDL en un circuito de compuertas lógicas forma parte de la síntesis. La salida es un conjunto de expresiones lógicas que describen las funciones lógicas necesarias para realizar el circuito.

Sin importar el tipo de ingreso de diseño que se use, las expresiones lógicas iniciales producidas por las herramientas de síntesis no tendrán una forma óptima, ya que reflejan lo que el diseñador ingresa en las herramientas CAD. Es imposible que un diseñador produzca manualmente resultados óptimos para circuitos grandes. Por ende, una de las tareas importantes de las herramientas de síntesis es manipular el diseño del usuario a fin de generar de manera automática un circuito equivalente, pero mejor.

La medida de lo que hace a un circuito mejor que otro depende tanto de las necesidades particulares de un proyecto de diseño como de la tecnología elegida para la implementación. En la sección 2.6 señalamos que un buen circuito puede ser el que tenga el menor costo. Hay otras posibles metas de optimación, motivadas por el tipo de tecnología de hardware usado para implementar el circuito. En el capítulo 3 estudiaremos las tecnologías de implementación y en el 4 volveremos al tema de las metas de optimación.

El desempeño de un circuito sintetizado puede evaluarse construyendo y probando físicamente el circuito, pero también mediante la simulación.

### 2.9.3 SIMULACIÓN FUNCIONAL

Un circuito representado en forma de expresiones lógicas se simula, entre otras cosas, para verificar que funcionará como se espera. La herramienta que cumple esta tarea recibe el nombre de *simulador funcional*. Utiliza las expresiones lógicas (conocidas como *ecuaciones*) generadas durante la síntesis y supone que se implementarán con compuertas perfectas por las que pasarán instantáneamente las señales. El simulador requiere que el usuario especifique las valoraciones de las entradas del circuito que han de aplicarse durante la simulación. Para cada una de ellas el simulador evalúa las salidas producida por las expresiones. Los resultados de la simulación suelen entregarse en forma de diagrama de tiempo que el usuario examina para verificar que el circuito opera como se requiere. La simulación funcional es un tema que se aborda con mayor hondura en el apéndice B.

### 2.9.4 DISEÑO FÍSICO

Después de la síntesis lógica el paso siguiente en el flujo de diseño consiste en determinar con exactitud cómo implementar el circuito en un chip. Este paso se denomina *diseño físico*. Como veremos en el próximo capítulo, hay varias tecnologías para implementar los circuitos lógicos. Las herramientas de diseño físico mapean un circuito especificado mediante expresiones lógicas en una realización que utiliza los recursos disponibles en el chip. Ello determina la ubicación de los elementos lógicos específicos, que no necesariamente son simples compuertas de los tipos expuestos hasta ahora. También establece las conexiones de cable que deben llevarse a cabo entre tales elementos para construir el circuito deseado.

### 2.9.5 SIMULACIÓN DE TIEMPO

Tanto las compuertas como otros elementos lógicos se implementan con circuitos electrónicos, como veremos en el capítulo 3. Un circuito electrónico no puede cumplir su función de manera instantánea. Cuando cambian los valores de las entradas al circuito se precisa cierto tiempo antes que ocurra el cambio correspondiente en la salida. Esto se llama *retardo de propagación* del circuito. El retardo de propagación consta de dos tipos de retardo. Cada elemento lógico necesita cierto lapso para generar una señal de salida válida siempre que haya cambios en los valores de sus entradas. Aparte de este retardo, existe un retardo producido por las señales que deben propagarse por los cables que conectan los diversos elementos lógicos. El efecto combinado es que los circuitos reales muestran retardos, lo que tiene un efecto significativo en su rapidez de operación.

Un *simulador de tiempo* evalúa los retardos esperados del circuito lógico diseñado. Su resultado sirve para determinar si éste satisface los requisitos de tiempo de la especificación para el diseño. Si no es así, el diseñador puede solicitar que las herramientas de diseño físico lo intenten de nuevo indicando restricciones temporales específicas que han de satisfacerse. Si esto no resulta, entonces el diseñador debe probar diferentes optimaciones en el paso de síntesis, o bien mejorar el diseño inicial presentado a las herramientas de síntesis.

### 2.9.6 CONFIGURACIÓN DE CHIP

Tras cerciorarse de que el circuito diseñado satisface todos los requisitos de la especificación, se implementa en un chip real. Este paso se llama *configuración* o *programación de chip*.

Las herramientas CAD analizadas en esta sección son las partes esenciales de un sistema CAD. En la figura 2.29 se ilustra todo el flujo de diseño expuesto. La presente es sólo una breve introducción. En el capítulo 12 se brinda una exposición completa de las herramientas CAD.

En este punto el lector debe haberse formado una idea de lo que supone el uso de herramientas CAD. Sin embargo, éstas sólo pueden apreciarse por completo cuando se usan de primera mano. Los apéndices B a D contienen tutoriales paso a paso que ilustran cómo usar el sistema CAD Quartus II incluido en el disco compacto que acompaña a esta obra. Se recomienda al lector que trabaje con el material de práctica presentado en esos apéndices. Puesto que los tutoriales emplean el lenguaje VHDL para el ingreso de diseño, en la sección siguiente se presenta una introducción a él.

## 2.10 INTRODUCCIÓN A VHDL

En el decenio de 1980, los rápidos avances en la tecnología de los circuitos integrados impulsaron el desarrollo de prácticas estándar de diseño para los circuitos digitales. VHDL se creó como parte de tal esfuerzo y se convirtió en el lenguaje estándar industrial para describir circuitos digitales, principalmente porque es un estándar oficial de la IEEE. En 1987 se adoptó la norma original para VHDL, llamada *IEEE* 1076. En 1993 se adoptó una norma revisada, la IEEE 1164.

En sus orígenes, VHDL tenía dos propósitos centrales. Primero, servía como lenguaje de documentación para describir la estructura de circuitos digitales complejos. Como estándar oficial del IEEE, ofreció una forma común de documentar los circuitos diseñados por varias personas. Segundo, VHDL aportó funciones para modelar el comportamiento de un circuito digital, lo que permitió emplearlo como entrada para programas que entonces se usaban para simular la operación del circuito.

En años recientes, aparte de usarlo para documentación y simulación, VHDL también se volvió popular para el ingreso de diseño en sistemas CAD. Las herramientas CAD se utilizan para sintetizar el código de VHDL en una implementación de hardware del circuito descrito. En este libro utilizaremos principalmente VHDL para la síntesis.

VHDL es un lenguaje complejo y refinado. Aunque aprender todas sus funciones es una tarea atemorizante, para usarlo en la síntesis basta conocer un subconjunto de ellas. Para simplificar la exposición nos centraremos en las características de VHDL que realmente se usan en los ejemplos del libro. Lo presentado debe ser suficiente para que el lector diseñe un repertorio amplio de circuitos. Quien desee aprender VHDL por completo puede remitirse a uno de los textos especializados [5-10].

VHDL se explica en varias etapas a lo largo de la obra. Nuestro enfoque general consiste en introducir funciones específicas sólo cuando sean relevantes para los temas de diseño que se aborden en la parte del texto respectiva. En el apéndice A se proporciona un resumen conciso de las funciones de VHDL expuestas en el libro. El lector encontrará conveniente remitirse a dicho

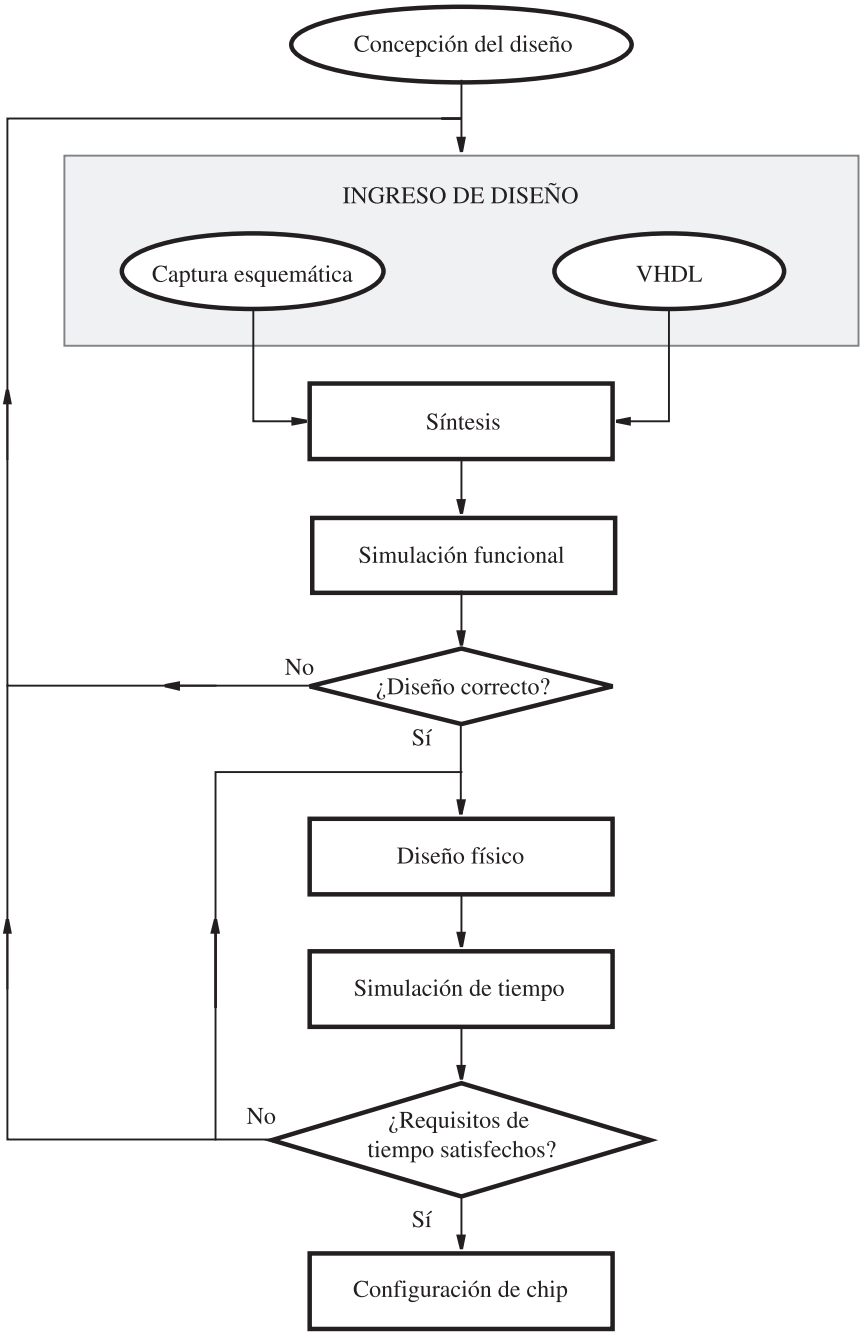



**Figura 2.29** Sistema CAD típico.

material de vez en cuando. En el resto del capítulo comentaremos los conceptos más básicos necesarios para escribir código sencillo en VHDL.

### 2.10.1 REPRESENTACIÓN DE SEÑALES DIGITALES EN VHDL

Cuando se usan herramientas CAD para sintetizar un circuito lógico el diseñador puede proporcionar la descripción inicial de varias formas, como se explicó en la sección 2.9.1. Un modo eficiente consiste en escribir esta descripción en código fuente de VHDL. El compilador de VHDL traduce ese código en un circuito lógico. Cada señal lógica del circuito se representa en el código de VHDL como un objeto de datos. Así como las variables declaradas en cualquier lenguaje de programación de alto nivel tienen tipos asociados —enteros o caracteres, por ejemplo—, los objetos de datos en VHDL pueden ser de varios tipos. La norma original de VHDL, la IEEE 1076, incluye un tipo de datos llamado *BIT*. Un objeto de este tipo es adecuado para representar señales digitales, pues sólo puede tener dos valores, 0 y 1. En este capítulo todas las señales de los ejemplos serán del tipo BIT. En la sección 4.12 estudiaremos otros tipos de datos, que se listan en el apéndice A.

### 2.10.2 CÓMO ESCRIBIR CÓDIGO SENCILLO EN VHDL

Daremos un ejemplo para ilustrar cómo escribir código fuente sencillo en VHDL. Considérese el circuito lógico de la figura 2.30. Si queremos escribir código de VHDL para representarlo, el primer paso es declarar las señales de entrada y salida. Esto se hace por medio de un constructo denominado *entidad* (*entity*). En la figura 2.31 aparece una entidad apropiada para este ejemplo.

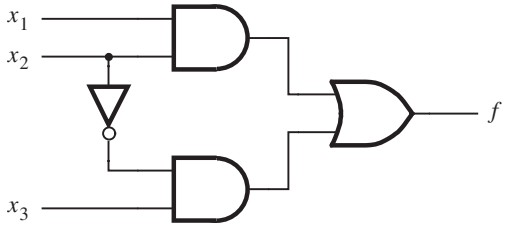



**Figura 2.30** Función lógica simple.

```
ENTITY example1 IS
    PORT ( x1, x2, x3  : IN   BIT ;
           f           : OUT  BIT ) ;
END example1 ;
```

**Figura 2.31** Declaración de entidad de VHDL para el circuito de la figura 2.30.

A una entidad debe asignársele un nombre; para este primer ejemplo se eligió el nombre *example1*. Las señales de entrada y salida de la entidad son sus puertos, identificados con la palabra clave PORT. Este nombre se deriva del lenguaje que se ocupa en electrónica en el que la palabra *port* (puerto) se refiere a una conexión de entrada o salida a un circuito electrónico. Cada puerto tiene un *modo* asociado que indica si se trata de una entrada (IN) o de una salida (OUT) del circuito. Cada puerto representa una señal, por tanto tiene un tipo asociado. La entidad *example1* tiene cuatro puertos. Los primeros tres, $x_1$, $x_2$ y $x_3$, son señales de entrada del tipo BIT. El puerto denominado *f* es una salida del tipo BIT.

En la figura 2.31 hemos usado nombres de señal simples, *x*1, *x*2, *x*3 y *f* para los puertos del circuito. Similar a la mayor parte de los lenguajes de programación, VHDL tiene reglas que fijan qué caracteres se permiten en los nombres de señal. Una regla simple es que los nombres de una señal pueden incluir cualquier letra o número, así como el carácter de subrayado '_' (guión bajo). Existen dos requisitos indispensables: un nombre de señal debe comenzar con una letra y no puede ser una palabra clave de VHDL.

Una entidad especifica las señales de entrada y salida para un circuito, pero no proporciona detalle alguno de lo que éste representa. La funcionalidad del circuito debe especificarse con un constructor de VHDL llamado *arquitectura* (*architecture*). En la figura 2.32 aparece una arquitectura para este ejemplo. Debe dársele un nombre; escogimos el de *LogicFunc*. Si bien el nombre puede ser cualquier cadena de texto, es razonable asignar uno significativo para el diseñador. En este caso el nombre elegido fue *LogicFunc* porque la arquitectura especifica la funcionalidad del diseño que usa una expresión lógica. VHDL soporta los operadores booleanos siguientes: AND, OR, NOT, NAND, NOR, XOR y XNOR. (Hasta el momento sólo hemos explicado los operadores AND, OR, NOT, NAND y NOR; el resto se presentará en el capítulo 3.) Después de la palabra clave BEGIN (comienzo), la arquitectura especifica, mediante el operador de VHDL de asignación de señal $\Leftarrow$, que hay que asignar a la salida *f* el resultado de la expresión lógica del miembro derecho del operador. Puesto que VHDL no supone precedencia alguna de los operadores lógicos, se usan paréntesis en la expresión. Cabría esperar que una instrucción de asignación como

$$f <= x1 \text{ AND } x2 \text{ OR NOT } x2 \text{ AND } x3$$

implicaría los paréntesis que siguen

$$f <= (x1 \text{ AND } x2) \text{ OR } ((\text{NOT } x2) \text{ AND } x3)$$

Pero para el código de VHDL tal suposición no es cierta. De hecho, sin los paréntesis el compilador VHDL produciría un error de tiempo de compilación para esta expresión.

La figura 2.33 muestra todo el código de VHDL para este ejemplo, con el que se ilustró el hecho de que un archivo de código fuente de VHDL tiene dos secciones principales: una entidad y una arquitectura.

```
ARCHITECTURE LogicFunc OF example1 IS
BEGIN
    f <= (x1 AND x2) OR (NOT x2 AND x3) ;
END LogicFunc ;
```

**Figura 2.32** Arquitectura (architecture) de VHDL para la entidad de la figura 2.31.

```
ENTITY example1 IS
    PORT ( x1, x2, x3  : IN   BIT ;
           f           : OUT  BIT ) ;
END example1 ;

ARCHITECTURE LogicFunc OF example1 IS
BEGIN
    f <= (x1 AND x2) OR (NOT x2 AND x3) ;
END LogicFunc ;
```

**Figura 2.33** Código completo de VHDL para el circuito de la figura 2.30.

```
ENTITY example2 IS
    PORT ( x1, x2, x3, x4  : IN   BIT ;
           f, g            : OUT  BIT ) ;
END example2 ;

ARCHITECTURE LogicFunc OF example2 IS
BEGIN
    f <= (x1 AND x3) OR (NOT x3 AND x2) ;
    g <= (NOT x3 OR x1) AND (NOT x3 OR x4) ;
END LogicFunc ;
```

**Figura 2.34** Código de VHDL para una función de cuatro entradas.

Una analogía simple para lo que representa cada sección es que la entidad equivale a un símbolo en un diagrama esquemático y la arquitectura especifica los circuitos lógicos en el interior del símbolo.

En la figura 2.34 se presenta un segundo ejemplo de código de VHDL. Este circuito tiene cuatro señales de entrada, $x1$, $x2$, $x3$ y $x4$, y dos señales de salida, $f$ y $g$. Se asigna a cada salida una expresión lógica. En la figura 2.35 se muestra un circuito lógico producido por el compilador de VHDL para este ejemplo.

Los dos ejemplos precedentes indican que una forma de asignar un valor a una señal en código de VHDL es mediante una expresión lógica. En terminología de VHDL, una expresión lógica se llama *instrucción de asignación simple*. Más adelante veremos que VHDL también soporta otros tipos distintos de instrucciones de asignación y muchas otras funciones útiles para describir circuitos mucho más complejos.

### 2.10.3 CÓMO *NO* ESCRIBIR CÓDIGO DE VHDL

Cuando está aprendiendo a usar VHDL u otros lenguajes de descripción de hardware, la tendencia del principiante es escribir código semejante al de un programa para computadora, con muchas variables y ciclos. Es difícil determinar qué circuito lógico producirán las herramientas CAD cuando sinteticen tal código. En esta obra se incluyen más de 100 ejemplos de código

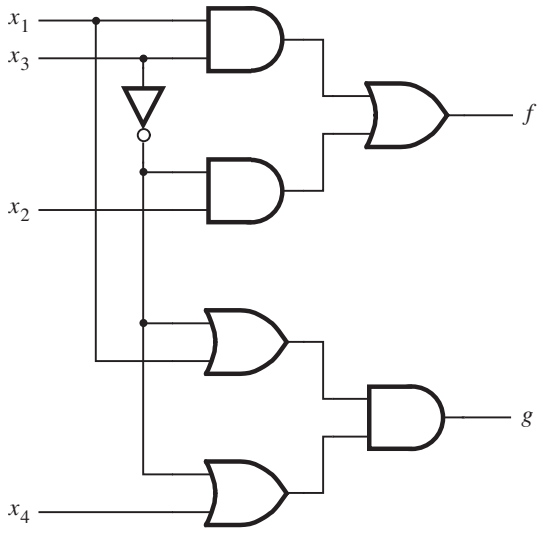



**Figura 2.35** Circuito lógico para el código de la figura 2.34.

de VHDL completo que representan numerosos circuitos lógicos de diferentes tipos. En estos ejemplos el código se relaciona fácilmente con el circuito lógico descrito. Se aconseja al lector que adopte el mismo estilo de código. Una buena pauta general es suponer que si el diseñador no puede determinar de inmediato qué circuito lógico describe el código de VHDL, entonces es probable que las herramientas CAD no sinteticen el circuito que se intenta modelar.

Una vez escrito todo el código de VHDL para un diseño específico, lo recomendable es analizar el circuito sintetizado por las herramientas CAD. Se aprende mucho acerca de VHDL, circuitos lógicos y síntesis lógica con este proceso.

## 2.11 COMENTARIOS FINALES

En este capítulo expusimos el concepto de circuitos lógicos. Demostramos que éstos pueden implementarse mediante compuertas lógicas y que es posible describirlos con un modelo matemático llamado *álgebra booleana*. Puesto que en la práctica los circuitos lógicos suelen ser grandes, es importante contar con buenas herramientas CAD que auxilien al diseñador. El CD que se entrega con el libro incluye el software Quartus II, una herramienta CAD de Altera Corporation. Presentamos algunas características básicas de esta herramienta y alentamos al lector a usar este software en cuanto le fuera posible.

La exposición ha sido muy elemental hasta ahora. En los capítulos siguientes analizaremos tanto los circuitos lógicos como las herramientas CAD de forma pormenorizada. Sin embargo, antes, en el próximo capítulo, examinaremos las tecnologías electrónicas más importantes usadas para construir circuitos lógicos. Con ello brindaremos al lector un cuadro general de las restricciones prácticas que ha de enfrentar un diseñador de circuitos lógicos.

## 2.12 EJEMPLOS DE PROBLEMAS RESUELTOS

En esta sección se presentan algunos problemas típicos que el lector puede encontrar, al tiempo que se muestra cómo resolverlos.

**Ejemplo 2.8** **Problema:** Determine si la ecuación siguiente es válida

$$\bar{x}_1\bar{x}_3 + x_2x_3 + x_1\bar{x}_2 = \bar{x}_1x_2 + x_1x_3 + \bar{x}_2\bar{x}_3$$

**Solución:** La ecuación es válida si las expresiones de los miembros izquierdo y derecho representan la misma función. A fin de realizar la comparación puede construir una tabla de verdad por cada lado y ver si ambas son iguales. Para hacerlo por medio de álgebra puede derivar una forma de suma canónica de productos por cada expresión.

En virtud de que $x + \bar{x} = 1$ (teorema 8*b*), es posible manipular el miembro izquierdo como sigue:

$$\begin{aligned} \text{LI} &= \bar{x}_1\bar{x}_3 + x_2x_3 + x_1\bar{x}_2 \\ &= \bar{x}_1(x_2 + \bar{x}_2)\bar{x}_3 + (x_1 + \bar{x}_1)x_2x_3 + x_1\bar{x}_2(x_3 + \bar{x}_3) \\ &= \bar{x}_1x_2\bar{x}_3 + \bar{x}_1\bar{x}_2\bar{x}_3 + x_1x_2x_3 + \bar{x}_1x_2x_3 + x_1\bar{x}_2x_3 + x_1\bar{x}_2\bar{x}_3 \end{aligned}$$

Estos términos producto presentan los mintérminos 2, 0, 7, 3, 5 y 4, respectivamente.

Para el miembro derecho se tiene

$$\begin{aligned} \text{LD} &= \bar{x}_1x_2 + x_1x_3 + \bar{x}_2\bar{x}_3 \\ &= \bar{x}_1x_2(x_3 + \bar{x}_3) + x_1(x_2 + \bar{x}_2)x_3 + (x_1 + \bar{x}_1)\bar{x}_2\bar{x}_3 \\ &= \bar{x}_1x_2x_3 + \bar{x}_1x_2\bar{x}_3 + x_1x_2x_3 + x_1\bar{x}_2x_3 + x_1\bar{x}_2\bar{x}_3 + \bar{x}_1\bar{x}_2\bar{x}_3 \end{aligned}$$

Estos términos producto representan los mintérminos 3, 2, 7, 5, 4 y 0, respectivamente. Como ambas expresiones especifican los mismos mintérminos, representan la misma función; por tanto, la ecuación es válida. Otra forma de representar esta función es mediante $\sum m(0, 2, 3, 4, 5, 7)$.

**Ejemplo 2.9** **Problema:** Diseñe la expresión de producto de sumas de costo mínimo para la función $f(x_1, x_2, x_3, x_4) = \sum m(0, 2, 4, 5, 6, 7, 8, 10, 12, 14, 15)$.

**Solución:** La función se define en términos de sus mintérminos. Para encontrar una expresión POS debemos comenzar con la definición en términos de maxitérminos, que es $f = \prod M(1, 3, 9, 11, 13)$. Por ende,

$$\begin{aligned} f &= M_1 \cdot M_3 \cdot M_9 \cdot M_{11} \cdot M_{13} \\ &= (x_1 + x_2 + x_3 + \bar{x}_4)(x_1 + x_2 + \bar{x}_3 + \bar{x}_4)(\bar{x}_1 + x_2 + x_3 + \bar{x}_4)(\bar{x}_1 + x_2 + \bar{x}_3 + \bar{x}_4)(\bar{x}_1 + \bar{x}_2 + x_3 + \bar{x}_4) \end{aligned}$$

El producto de los primeros dos maxitérminos puede escribirse de nuevo como

$$\begin{aligned} M_1 \cdot M_3 &= (x_1 + x_2 + \overline{x}_4 + x_3)(x_1 + x_2 + \overline{x}_4 + \overline{x}_3) && \text{por la propiedad conmutativa } 10b \\ &= x_1 + x_2 + \overline{x}_4 + x_3\overline{x}_3 && \text{por la propiedad distributiva } 12b \\ &= x_1 + x_2 + \overline{x}_4 + 0 && \text{por el teorema } 8a \\ &= x_1 + x_2 + \overline{x}_4 && \text{por el teorema } 6b \end{aligned}$$

De manera similar, $M_9 \cdot M_{11} = \overline{x}_1 + x_2 + \overline{x}_4$. Ahora podemos usar de nuevo $M_{11}$, de acuerdo con la propiedad $7a$, para derivar $M_{11} \cdot M_{13} = \overline{x}_1 + x_3 + \overline{x}_4$. En consecuencia

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

Si aplicamos de nuevo $12b$ se obtiene la respuesta final

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

---

**Ejemplo 2.10**

**Problema:** Un circuito que controla un sistema digital tiene tres entradas: $x_1$, $x_2$ y $x_3$. Debe reconocer tres condiciones:

- La condición $A$ es verdadera si $x_3$ es verdadera y $x_1$ es verdadera o $x_2$ es falsa
- La condición $B$ es verdadera si $x_1$ es verdadera y $x_2$ o $x_3$ son falsas
- La condición $C$ es verdadera si $x_2$ es verdadera y $x_1$ es verdadera o $x_3$ falsa

El circuito de control debe producir una salida de 1 si al menos dos de las condiciones *A, B* y *C* son verdaderas. Diseñe el circuito más simple que pueda usarse para este propósito.

**Solución:** Sea 1 para verdadero y 0 para falso; entonces las tres condiciones pueden expresarse del modo siguiente:

$$\begin{aligned} A = x_3(x_1 + \overline{x}_2) &= x_3x_1 + x_3\overline{x}_2 \\ B = x_1(\overline{x}_2 + \overline{x}_3) &= x_1\overline{x}_2 + x_1\overline{x}_3 \\ C = x_2(x_1 + \overline{x}_3) &= x_2x_1 + x_2\overline{x}_3 \end{aligned}$$

Luego, la salida deseada del circuito puede expresarse como $f = AB + AC + BC$. Estos términos producto se pueden determinar como:

$$\begin{aligned} AB &= (x_3x_1 + x_3\overline{x}_2)(x_1\overline{x}_2 + x_1\overline{x}_3) \\ &= x_3x_1x_1\overline{x}_2 + x_3x_1x_1\overline{x}_3 + x_3\overline{x}_2x_1\overline{x}_2 + x_3\overline{x}_2x_1\overline{x}_3 \\ &= x_3x_1\overline{x}_2 + 0 + x_3\overline{x}_2x_1 + 0 \\ &= x_1\overline{x}_2x_3 \end{aligned}$$

$$\begin{aligned} AC &= (x_3x_1 + x_3\overline{x}_2)(x_2x_1 + x_2\overline{x}_3) \\ &= x_3x_1x_2x_1 + x_3x_1x_2\overline{x}_3 + x_3\overline{x}_2x_2x_1 + x_3\overline{x}_2x_2\overline{x}_3 \\ &= x_3x_1x_2 + 0 + 0 + 0 \\ &= x_1x_2x_3 \end{aligned}$$

$$\begin{aligned} BC &= (x_1\overline{x}_2 + x_1\overline{x}_3)(x_2x_1 + x_2\overline{x}_3) \\ &= x_1\overline{x}_2x_2x_1 + x_1\overline{x}_2x_2\overline{x}_3 + x_1\overline{x}_3x_2x_1 + x_1\overline{x}_3x_2\overline{x}_3 \\ &= 0 + 0 + x_1\overline{x}_3x_2 + x_1\overline{x}_3x_2 \\ &= x_1x_2\overline{x}_3 \end{aligned}$$

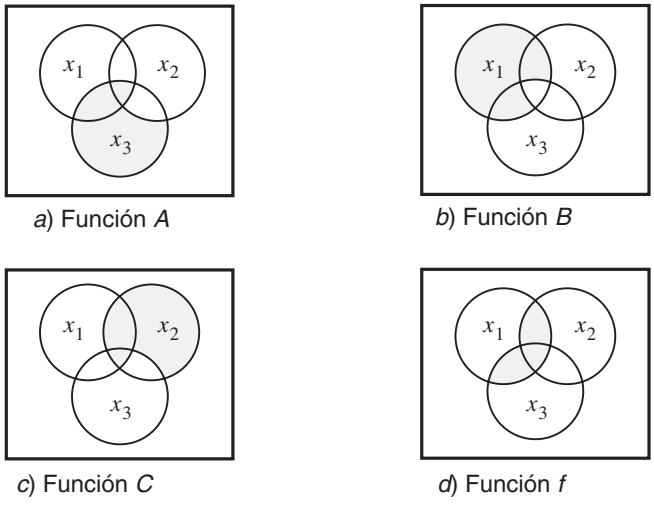
En consecuencia, *f* puede escribirse como

$$\begin{aligned} f &= x_1\overline{x}_2x_3 + x_1x_2x_3 + x_1x_2\overline{x}_3 \\ &= x_1(\overline{x}_2 + x_2)x_3 + x_1x_2(x_3 + \overline{x}_3) \\ &= x_1x_3 + x_1x_2 \\ &= x_1(x_3 + x_2) \end{aligned}$$

**Ejemplo 2.11** **Problema:** Resolver el problema del ejemplo 2.10 por medio de diagramas de Venn.

**Solución:** Los diagramas de Venn para las funciones *A*, *B* y *C* del ejemplo 2.10 se muestran en los incisos *a* a *c* de la figura 2.36. Puesto que la función *f* ha de ser verdadera cuando dos o más de *A*, *B* y *C* sean verdaderas, entonces el diagrama de Venn para *f* se forma identificando las áreas sombreadas comunes en los diagramas de Venn de *A*, *B* y *C*. Cualquiera área sombreada en dos o más de estos diagramas también está sombreada en *f*, como se muestra en la figura 2.36*d*. Este diagrama corresponde a la función

$$f = x_1x_2 + x_1x_3 = x_1(x_2 + x_3)$$




*a*) Función *A* *b*) Función *B*

*c*) Función *C* *d*) Función *f*

**Figura 2.36** Diagramas de Venn para el ejemplo 2.11.

**Ejemplo 2.12**

**Problema:** Derivar la expresión de suma de productos más simple para la función

$$f = x_2\overline{x}_3x_4 + x_1x_3x_4 + x_1\overline{x}_2x_4$$

**Solución:** La aplicación de la propiedad de consenso 17*a* a los primeros dos términos produce

$$\begin{aligned} f &= x_2\overline{x}_3x_4 + x_1x_3x_4 + x_2x_4x_1x_4 + x_1\overline{x}_2x_4 \\ &= x_2\overline{x}_3x_4 + x_1x_3x_4 + x_1x_2x_4 + x_1\overline{x}_2x_4 \end{aligned}$$

Ahora, al aplicar la propiedad de combinación 14*a* en los últimos dos términos se obtiene

$$f = x_2\overline{x}_3x_4 + x_1x_3x_4 + x_1x_4$$

Finalmente, con la propiedad de absorción 13*a* se llega a

$$f = x_2\overline{x}_3x_4 + x_1x_4$$

**Ejemplo 2.13**

**Problema:** Derivar la expresión de producto de sumas más simple para la función

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

**Solución:** La aplicación de la propiedad de consenso 17*b* a los dos primeros términos conduce a

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

Ahora, al aplicar la propiedad de combinación 14*b* a los dos últimos términos se obtiene

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

Finalmente, al aplicar la propiedad de absorción 13*b* al primero y al último término resulta

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

## PROBLEMAS

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

**2.1** Use manipulación algebraica para comprobar que $x + yz = (x + y) \cdot (x + z)$. Observe que ésta es la propiedad distributiva, como afirma la identidad 12*b* de la sección 2.5.

**2.2** Use manipulación algebraica para comprobar que $(x + y) \cdot (x + \overline{y}) = x$.

**2.3** Use manipulación algebraica para comprobar que $xy + yz + \overline{x}z = xy + \overline{x}z$. Observe que ésta es la propiedad de consenso 17*a* de la sección 2.5.

**2.4** Use diagramas de Venn para comprobar la identidad del problema 1.

**2.5** Use diagramas de Venn para comprobar el teorema de DeMorgan, según se da en las expresiones 15*a* y 15*b* de la sección 2.5.

**2.6** Use un diagrama de Venn para comprobar que

$$(x_1 + x_2 + x_3) \cdot (x_1 + x_2 + \overline{x}_3) = x_1 + x_2$$

***2.7** Determine si las expresiones siguientes son válidas, es decir, si los miembros izquierdo y derecho representan la misma función.

*a*) $\overline{x}_1x_3 + x_1x_2\overline{x}_3 + \overline{x}_1x_2 + x_1\overline{x}_2 = \overline{x}_2x_3 + x_1\overline{x}_3 + x_2\overline{x}_3 + \overline{x}_1x_2x_3$
*b*) $x_1\overline{x}_3 + x_2x_3 + \overline{x}_2\overline{x}_3 = (x_1 + \overline{x}_2 + x_3)(x_1 + x_2 + \overline{x}_3)(\overline{x}_1 + x_2 + \overline{x}_3)$
*c*) $(x_1 + x_3)(\overline{x}_1 + \overline{x}_2 + \overline{x}_3)(\overline{x}_1 + x_2) = (x_1 + x_2)(x_2 + x_3)(\overline{x}_1 + \overline{x}_3)$

**2.8** Trace un diagrama de tiempo para el circuito de la figura 2.19*a*. Muestre las formas de onda que pueden observarse en todos los cables del circuito.

**2.9** Repita el problema 2.8 para el circuito de la figura 2.19*b*.

**2.10** Use manipulación algebraica para demostrar que para las tres variables de entrada $x_1$, $x_2$ y $x_3$,

$$\sum m(1, 2, 3, 4, 5, 6, 7) = x_1 + x_2 + x_3$$

**2.11** Use manipulación algebraica para demostrar que para las tres variables de entrada $x_1$, $x_2$ y $x_3$,

$$\Pi M\,(0, 1, 2, 3, 4, 5, 6) = x_1x_2x_3$$

***2.12** Use manipulación algebraica para hallar la mínima expresión en suma de productos para la función $f = x_1x_3 + x_1\overline{x}_2 + \overline{x}_1x_2x_3 + \overline{x}_1\overline{x}_2\overline{x}_3$.

**2.13** Use manipulación algebraica para encontrar la mínima expresión en suma de productos para la función $f = x_1\overline{x}_2\overline{x}_3 + x_1x_2x_4 + x_1\overline{x}_2x_3\overline{x}_4$.

**2.14** Use manipulación algebraica para hallar la mínima expresión de producto de sumas para la función $f = (x_1 + x_3 + x_4) \cdot (x_1 + \overline{x}_2 + x_3) \cdot (x_1 + \overline{x}_2 + \overline{x}_3 + x_4)$.

***2.15** Use manipulación algebraica para encontrar la mínima expresión de producto de sumas para la función $f = (x_1 + x_2 + x_3) \cdot (x_1 + \overline{x}_2 + x_3) \cdot (\overline{x}_1 + \overline{x}_2 + x_3) \cdot (x_1 + x_2 + \overline{x}_3)$.

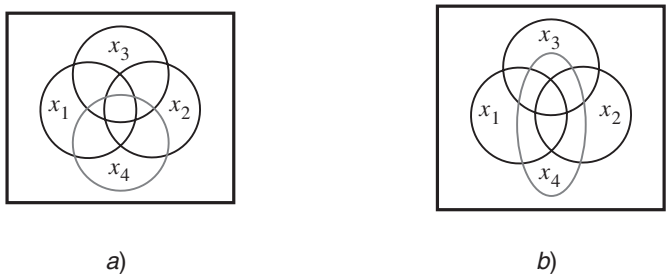
**2.16** *a*) Muestre la ubicación de todos los mintérminos en un diagrama de Venn de tres variables.

*b*) Muestre un diagrama de Venn separado por cada término producto en la función $f = x_1\overline{x}_2x_3 + x_1x_2 + \overline{x}_1x_3$. Use diagramas de Venn para hallar la mínima forma de suma de producto de *f*.

**2.17** Represente la función de la figura 2.18 en la forma de diagrama de Venn y determine su mínima forma de suma de productos.

**2.18** En la figura P2.1 se muestran dos intentos para trazar un diagrama de Venn para cuatro variables. Para los incisos *a*) y *b*) de la figura, explique por qué el diagrama no es correcto. (Sugerencia: El diagrama de Venn debe ser capaz de representar los 16 mintérminos de las cuatro variables.)

**2.19** En la figura P2.2 se observa la representación de un diagrama de Venn de cuatro variables, así como la ubicación de los mintérminos $m_0$, $m_1$ y $m_2$. Muestre la ubicación de los otros mintérminos en el diagrama. Represente la función $f = \overline{x}_1\overline{x}_2x_3\overline{x}_4 + x_1x_2x_3x_4 + \overline{x}_1x_2$ en este diagrama.




*a*) *b*)

**Figura P2.1** Dos intentos para trazar un diagrama de Venn de cuatro variables.

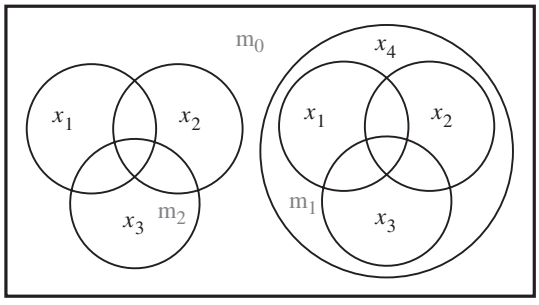



**Figura P2.2** Diagrama de Venn de cuatro variables.

***2.20** Diseñe el circuito más simple de suma de productos que implemente la función $f(x_1, x_2, x_3) = \sum m(3, 4, 6, 7)$.

**2.21** Diseñe el circuito más simple de suma de productos que implemente la función $f(x_1, x_2, x_3) = \sum m(1, 3, 4, 6, 7)$.

**2.22** Diseñe el circuito más simple de producto de sumas que implemente la función $f(x_1, x_2, x_3) = \prod M(0, 2, 5)$.

***2.23** Diseñe el circuito más simple de producto de sumas que implemente la función $f(x_1, x_2, x_3) = \prod M(0, 1, 5, 7)$.

**2.24** Derive la expresión más simple de suma de productos para la función $f(x_1, x_2, x_3, x_4) = x_1\bar{x}_3\bar{x}_4 + x_2\bar{x}_3x_4 + x_1\bar{x}_2\bar{x}_3$.

**2.25** Derive la expresión más simple de suma de productos para la función $f(x_1, x_2, x_3, x_4, x_5) = \bar{x}_1\bar{x}_3\bar{x}_5 + \bar{x}_1\bar{x}_3\bar{x}_4 + \bar{x}_1x_4x_5 + x_1\bar{x}_2\bar{x}_3x_5$. (Sugerencia: Aplique la propiedad de consenso 17*a*).

**2.26** Derive la expresión más simple de producto de sumas para la función $f(x_1, x_2, x_3, x_4) = (\bar{x}_1 + \bar{x}_3 + \bar{x}_4)(\bar{x}_2 + \bar{x}_3 + x_4)(x_1 + \bar{x}_2 + \bar{x}_3)$. (Sugerencia: Aplique la propiedad de consenso 17*b*.)

**2.27** Derive la expresión más simple de producto de sumas para la función $f(x_1, x_2, x_3, x_4, x_5) = (\bar{x}_2 + x_3 + x_5)(x_1 + \bar{x}_3 + x_5)(x_1 + x_2 + x_5)(x_1 + \bar{x}_4 + \bar{x}_5)$. (Sugerencia: Aplique la propiedad de consenso 17*b*.)

***2.28** Diseñe el circuito más simple que tenga tres entradas, $x_1$, $x_2$ y $x_3$, que produzca un valor de salida de 1 siempre que dos o más de las variables de entrada tenga el valor 1; de otro modo, la salida debe ser 0.

**2.29** Diseñe el circuito más simple que tenga tres entradas, $x_1$, $x_2$ y $x_3$, que produzca un valor de salida de 1 siempre que exactamente una o dos de las variables de entrada tenga el valor de 1; de otro modo, la salida ha de ser 0.

**2.30** Diseñe el circuito más simple que tenga cuatro entradas, $x_1$, $x_2$, $x_3$ y $x_4$, que produzca un valor de salida de 1 siempre que tres o más de las variables de entrada tengan el valor de 1; de otro modo, la salida debe ser 0.

**2.31** Para el diagrama de tiempo de la figura P2.3, sintetice la función $f(x_1, x_2, x_3)$ en la forma más simple de suma de productos.

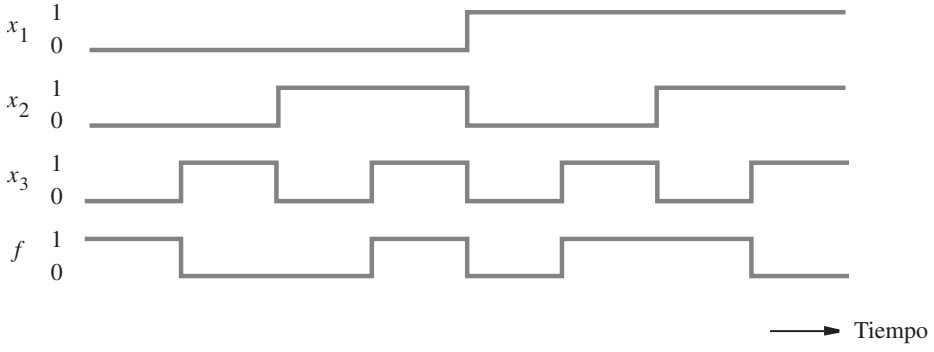



**Figura P2.3** Diagrama de tiempo que representa una función lógica.

***2.32** Para el diagrama de tiempo de la figura P2.3, sintetice la función $f(x_1, x_2, x_3)$ en la forma más simple de producto de sumas.

***2.33** Para el diagrama de tiempo de la figura P2.4, sintetice la función $f(x_1, x_2, x_3)$ en la forma más simple de suma de productos.

**2.34** Para el diagrama de tiempo de la figura P2.4, sintetice la función $f(x_1, x_2, x_3)$ en la forma más simple de producto de sumas.

**2.35** Diseñe un circuito con salida $f$ y entradas $x_1$, $x_0$, $y_1$ y $y_0$. Sea $X = x_1, x_0$ un número, donde los cuatro valores posibles de $X$ (00, 01, 10 y 11) representan los cuatro números 0, 1, 2 y 3, respectivamente. (La representación de números se explicará en el capítulo 5.) De manera similar, sea $Y = y_1y_0$ la representación de otro número con los mismos cuatro posibles valores. La salida $f$ debe ser 1 si los números representados por $X$ y $Y$ son iguales. De otro modo, $f$ debe ser 0.

*a*) Elabore la tabla de verdad para $f$.

*b*) Sintetice la expresión más simple posible de producto de sumas para $f$.

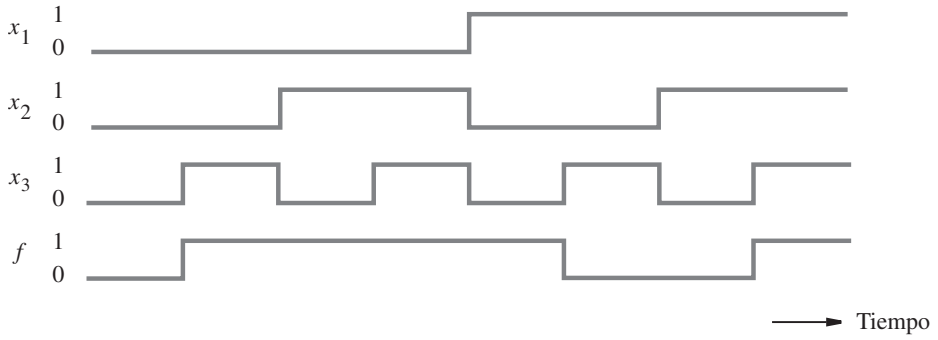



**Figura P2.4** Diagrama de tiempo que representa una función lógica.

**2.36** Repita el problema 2.35 para el caso en que $f$ debe ser 1 sólo si $X \geq Y$.

*a*) Elabore la tabla de verdad para $f$.
*b*) Muestre la expresión de suma canónica de productos para $f$.
*c*) Muestre la expresión más simple posible de suma de productos para $f$.

**2.37** Implemente la función de la figura 2.26 usando sólo compuertas NAND.

**2.38** Implemente la función de la figura 2.26 usando solamente compuertas NOR.

**2.39** Implemente el circuito de la figura 2.35 usando nada más compuertas NAND y NOR.

***2.40** Diseñe el circuito más simple que implemente la función $f(x_1, x_2, x_3) = \sum m(3, 4, 6, 7)$ usando compuertas NAND.

**2.41** Diseñe el circuito más simple que implemente la función $f(x_1, x_2, x_3) = \sum m(1, 3, 4, 6, 7)$ usando compuertas NAND.

***2.42** Repita el problema 2.40 usando ahora compuertas NOR.

**2.43** Repita el problema 2.41 usando ahora compuertas NOR.

**2.44** *a*) Use una herramienta de captura esquemática para trazar los esquemas de las funciones siguientes

$$f_1 = x_2\overline{x}_3\overline{x}_4 + \overline{x}_1x_2x_4 + \overline{x}_1x_2x_3 + x_1x_2x_3$$
$$f_2 = x_2\overline{x}_4 + \overline{x}_1x_2 + x_2x_3$$

*b*) Use simulación funcional para comprobar que $f_1 = f_2$.

**2.45** *a*) Use una herramienta de captura esquemática para trazar los esquemas de las funciones siguientes

$$f_1 = (x_1 + x_2 + \overline{x}_4) \cdot (\overline{x}_2 + x_3 + \overline{x}_4) \cdot (\overline{x}_1 + x_3 + \overline{x}_4) \cdot (\overline{x}_1 + \overline{x}_3 + \overline{x}_4)$$
$$f_2 = (x_2 + \overline{x}_4) \cdot (x_3 + \overline{x}_4) \cdot (\overline{x}_1 + \overline{x}_4)$$

*b*) Use simulación funcional para comprobar que $f_1 = f_2$.

**2.46** Escriba código de VHDL para implementar la función $f(x_1, x_2, x_3) = \sum m(0, 1, 3, 4, 5, 6)$.

**2.47** *a*) Escriba código de VHDL para describir las funciones siguientes

$$f_1 = x_1\overline{x}_3 + x_2\overline{x}_3 + \overline{x}_3\overline{x}_4 + x_1x_2 + x_1\overline{x}_4$$
$$f_2 = (x_1 + \overline{x}_3) \cdot (x_1 + x_2 + \overline{x}_4) \cdot (x_2 + \overline{x}_3 + \overline{x}_4)$$

*b*) Use simulación funcional para comprobar que $f_1 = f_2$.

**2.48** Considere las instrucciones siguientes de asignación en VHDL

```
f1 <= ((x1 AND x3) OR (NOT x1 AND NOT x3)) OR ((x2 AND x4) OR
      (NOT x2 AND NOT x4)) ;
f2 <= (x1 AND x2 AND NOT x3 AND NOT x4) OR (NOT x1 AND NOT x2 AND x3 AND x4)
      OR (x1 AND NOT x2 AND NOT x3 AND x4) OR
      (NOT x1 AND x2 AND x3 AND NOT x4) ;
```

*a*) Escriba código de VHDL completo para implementar f1 y f2.

*b*) Use simulación funcional para comprobar que $f1 = \overline{f2}$.

## BIBLIOGRAFÍA


1. G. Boole, *An Investigation of the Laws of Thought*, 1854, reimpreso por Dover Publications, Nueva York, 1954.
2. C. E. Shannon, "A Symbolic Analysis of Relay and Switching Circuits", *Transactions of AIEE* 57 (1938), pp. 713-723.
3. E. V. Huntington, "Sets of Independent Postulates for the Algebra of Logic", *Transactions of the American Mathematical Society* 5 (1904), pp. 288-309.
4. S. Brown y Z. Vranesic, *Fundamentals of Digital Logic with Verilog Design* (McGraw-Hill: Nueva York, 2003).
5. Z. Navabi, *VHDL—Analysis and Modeling of Digital Systems*, 2a. ed. (McGraw-Hill: Nueva York, 1998).
6. D. L. Perry, *VHDL*, 3a. ed. (McGraw-Hill: Nueva York, 1998).
7. J. Bhasker, *A VHDL Primer*, 3a. ed. (Prentice-Hall: Englewood Cliffs, NJ, 1998).
8. K. Skahill, *VHDL for Programmable Logic* (Addison-Wesley: Menlo Park, CA, 1996).
9. A. Dewey, *Analysis and Design of Digital Systems with VHDL* (PWS Publishing Co.: Boston, 1997).
10. D. J. Smith, *HDL Chip Design* (Doone Publications: Madison, AL, 1996).