Prefacio

El *diseño digital* se ocupa del diseño de circuitos electrónicos digitales. El tema se conoce también por otros nombres, como *diseño lógico, circuitos conmutadores, lógica digital* y *sistemas digitales*. Los circuitos digitales se emplean en el diseño de sistemas, por ejemplo computadoras digitales, calculadoras electrónicas, dispositivos digitales de control, equipo de comunicación digital y muchas otras aplicaciones que requieren hardware digital electrónico. En este libro se presentan las herramientas básicas que se usan en el diseño de los circuitos digitales y se proporcionan varios métodos y procedimientos adecuados para una variedad de aplicaciones del diseño digital.

Los componentes que se utilizan para construir sistemas digitales están encapsulados en paquetes de circuitos integrados. Los circuitos de integración a pequeña escala (SSI) contienen varias compuertas o *flip-flops* en un solo paquete. Los dispositivos de integración a mediana escala (MSI) ofrecen funciones digitales específicas, y los dispositivos de integración a gran escala (LSI) proporcionan módulos completos de computadora. Es importante que el diseñador digital se familiarice con los diversos componentes digitales que se encuentran en las formas de circuitos integrados. Por esta razón, los componentes MSI y LSI de uso más frecuente se introducen en el libro junto con explicaciones de sus propiedades lógicas. El uso de circuitos integrados en el diseño de circuitos digitales se ilustra mediante ejemplos en el libro, en los problemas que se presentan al final de los capítulos y en un conjunto de 15 experimentos que se recomienda realizar en el laboratorio.

El libro consta de 11 capítulos. Los Capítulos del 1 al 5 tratan de circuitos combinacionales. Los Capítulos 6 y 7 cubren los circuitos secuenciales síncronos.

Estos siete capítulos y el Capítulo 10, los cuales abarcan los circuitos integrados digitales, se toman del libro del autor *Digital Logic and Computer Design* —Lógica digital y diseño de computadoras— (Prentice-Hall, 1979). Los Capítulos 8, 9 y 11 contienen material referente a las máquinas de estado algorítmico, circuitos secuenciales asíncronos y experimentos de laboratorio con circuitos integrados. Los once capítulos proporcionan un conjunto coherente de temas adecuados para un primer curso de diseño digital.

En el **Capítulo 1** se analizan los diversos sistemas binarios adecuados para representar información en los sistemas digitales. El **Capítulo 2** es la introducción al álgebra booleana junto con las diversas compuertas lógicas que se emplean en la construcción de circuitos digitales. En el **Capítulo 3** se cubren los métodos de mapas y tablas para simplificar los circuitos digitales y se presenta un procedimiento sistemático para el implante de las lógicas NAND y NOR. Los primeros tres capítulos dan las bases necesarias para entender el resto del libro.

En el **Capítulo 4** se compendia un procedimiento formal para el análisis y diseño de los circuitos combinacionales. El **Capítulo 5** trata con los componentes de los circuitos combinacionales MSI y LSI. Se explican las funciones de uso frecuente como sumadores, comparadores, decodificadores y multiplexores y su uso en el diseño de circuitos digitales se ilustra con ejemplos. Se introducen la memoria solo de lectura (ROM) y el arreglo lógico programable (PLA) y se demuestra su utilidad en el diseño de circuitos combinacionales complejos. En el **Capítulo 6** se compendian diversos procedimientos formales para el análisis y diseño de circuitos secuenciales síncronos con reloj. En el **Capítulo 7** se presentan varios componentes secuenciales MSI como registradores, contadores, registradores de corrimiento y la memoria de acceso aleatorio (RAM).

En el **Capítulo 8** se incluye el método de diseño digital de la máquina de estado algorítmico (ASM). El diagrama ASM es un diagrama especial de flujo adecuado para describir tanto operaciones secuenciales como las operaciones en paralelo en el hardware digital. Varios ejemplos de diseño demuestran la aplicación del diagrama ASM en el diseño del control lógico de los sistemas digitales. En el **Capítulo 9** se presentan procedimientos formales para el análisis y diseño de los circuitos secuenciales asíncronos. Se compendian métodos para mostrar cómo un circuito secuencial asíncrono puede implementarse como un circuito combinacional con retroalimentación o como un circuito con seguro SR. El **Capítulo 10** trata acerca de la electrónica de los circuitos digitales y se presentan las familias lógicas más comunes de circuito integrado digital. Se supone cierto conocimiento de electrónica básica, pero no hay un prerrequisito específico para el resto del libro.

En el **Capítulo 11** se compendian 15 experimentos que pueden realizarse en el laboratorio con hardware que comercialmente está disponible y es de bajo costo. Esos experimentos usan circuitos estándar integrados del tipo TTL. La operación de los circuitos integrados se explica con referencia a diagramas en los capítulos anteriores donde en forma original se introdujeron componentes similares. Cada experimento se presenta de manera informal y no paso a paso, de modo que se espera que el estudiante produzca los detalles del diagrama del circuito y formule un procedimiento para verificar la operación del circuito en el laboratorio.

Cada capítulo incluye un conjunto de problemas y una lista de referencias. En el Apéndice aparecen las respuestas a los problemas seleccionados para ayudar al estudiante y auxiliar al lector independiente. Para el instructor se publicó un *Manual de Soluciones*, disponible en inglés.

M. MORRIS MANO

Sistemas binarios

1

1-1 COMPUTADORAS DIGITALES Y SISTEMAS DIGITALES

Las computadoras digitales han hecho posibles muchos avances científicos, industriales y comerciales que de otra manera nunca se hubieran alcanzado. El programa espacial habría sido imposible sin monitoreo continuo por computadora en tiempo real y, muchas empresas de negocios funcionan en forma eficiente sólo con la ayuda del procesamiento automático de información. Las computadoras se usan en cálculos científicos, en el procesamiento de información comercial y de negocios, control de tránsito aéreo, vía espacial, campo educativo y muchas otras áreas. La propiedad más sorprendente de una computadora digital es su generalidad. Puede seguir una secuencia de instrucciones, denominada *programa*, que opera según la información dada. El usuario puede especificar y cambiar los programas y/o la información de acuerdo con la necesidad específica. A causa de esta flexibilidad, las computadoras digitales de propósito general pueden realizar una amplia variedad de tareas de procesamiento de información.

La computadora digital de propósito general es el ejemplo mejor conocido de un sistema digital. Otros ejemplos incluyen los intercambios de canales de comunicación telefónicos, voltmetros digitales, contadores de frecuencia, máquinas calculadoras y máquinas de teletipo. Es característico de un sistema digital la manipulación de *elementos discretos* de información. Es posible que dichos elementos sean impulsos eléctricos, los dígitos decimales, las letras de un alfabeto, operaciones aritméticas, signos de puntuación, o cualquier otro conjunto de símbolos con significado. La yuxtaposición de elementos discretos de información representa una cantidad de información. Por ejemplo, las letras *s*, *o* y *l* forman la palabra *sol*. Los dígitos 237 forman un número. Por tanto, una secuencia de elementos discretos forma un lenguaje, esto es, una disciplina que lleva información. Las primeras computadoras digitales se usaron principalmente para cálculos numéricos. En este caso, los elementos discretos que se utilizan son dígitos. Para esta aplicación surgió el término de *computadora digital*. Un nombre más apropiado pero más largo para una computadora digital sería un "sistema para procesar información discreta".

Los elementos discretos de información se representan en un sistema digital mediante cantidades físicas denominadas *señales*. Las señales eléctricas como voltajes y corrientes son las más comunes. Las señales en todos los sistemas digitales en la actualidad tienen sólo dos valores discretos y se dice que son *binarios*. El diseñador de un sistema digital está limitado al uso de señales binarias debido a la confiabilidad más baja de los circuitos electrónicos de valores múltiples. En otras palabras, puede diseñarse un circuito con diez estados, que usa un valor discreto de voltaje para cada estado, pero tendría una confiabilidad muy baja de operación. En contraste, un circuito de transistor que está, ya sea encendido o apagado, tiene dos valores de señal posibles y puede construirse para que sea en extremo confiable. Debido a esta restricción física de los componentes, y debido a que la lógica humana tiende a ser binaria, los sistemas digitales que están limitados a tomar valores discretos están restringidos aún más a tener valores binarios.

Las cantidades discretas de información emergen ya sea de la propia naturaleza del proceso o es posible cuantificarlas a propósito a partir de un proceso continuo. Por ejemplo, en forma inherente una nómina es un proceso discreto que contiene los nombres de los empleados, número de seguro social, salarios semanales, impuestos sobre el ingreso, etc. El pago de un empleado se procesa usando valores discretos de información, por ejemplo letra del alfabeto (nombre), dígitos (salario) y símbolos especiales como $. Por otra parte, un investigador científico puede observar un proceso continuo pero registra sólo cantidades específicas en una forma tabular. Por lo tanto, el científico está cuantificando su información continua. Cada número en su tabla es un elemento discreto de información.

Muchos sistemas físicos pueden describirse en forma matemática por ecuaciones diferenciales cuyas soluciones, como una función del tiempo, dan el comportamiento matemático completo del proceso. Una *computadora analógica* realiza una *simulación* directa de un sistema físico. Cada sección de la computadora es la análoga de una porción particular del proceso bajo estudio. Las variables en la computadora analógica se representan por señales continuas, por lo común voltajes eléctricos que varían con el tiempo. Las variables de señal se consideran análogas a las del proceso y se comportan de la misma forma. En consecuencia, las mediciones de voltaje analógico pueden sustituirse por las variables del proceso. El término *señal analógica* algunas veces se sustituye por *señal continua*, debido a que la "computadora analógica" ha llegado a significar una computadora que manipula variables continuas.

Para simular un proceso físico en una computadora digital, las magnitudes deben cuantificarse. Cuando las variables del proceso se presentan por señales continuas en tiempo real, estas últimas se cuantifican con un dispositivo de conversión analógica en digital. Un sistema físico cuya conducta se describe por ecuaciones matemáticas se simula en una computadora digital mediante métodos numéricos. Cuando el problema que va a procesarse es inherentemente discreto, como en aplicaciones comerciales, la computadora digital manipula las variables en su forma natural.

En la Fig. 1-1 se muestra un diagrama de bloques de la computadora digital. La unidad de memoria almacena los programas al igual que la entrada, la salida y la información intermedia. La unidad procesadora realiza las tareas aritméticas y de otros procesamientos de información como se especifica por un programa.

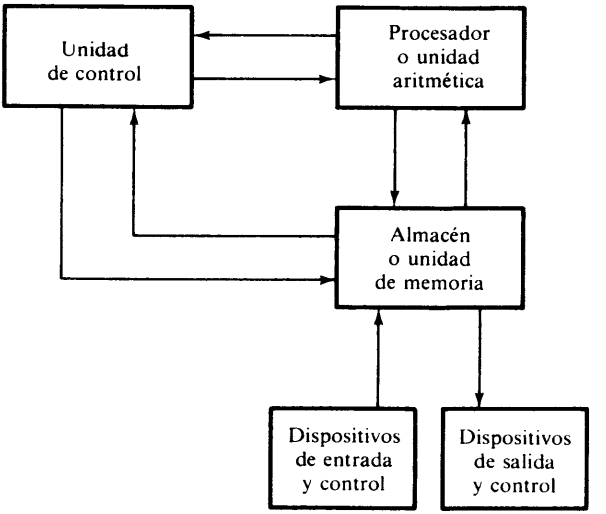


Figura 1-1 Diagrama de bloques de una computadora digital.

La unidad de control supervisa el flujo de información entre las diversas unidades. La unidad de control recupera las instrucciones, una por una, del programa que está almacenando en la memoria. Para cada instrucción, la unidad de control informa al procesador para que ejecute la operación especificada por la instrucción. Tanto el programa como la información se almacena en la memoria. La unidad de control supervisa las instrucciones del programa y el procesador manipula la información como se especifica en el programa.

El programa y la información preparada por el usuario se transfieren a la unidad de memoria mediante un dispositivo de entrada, como una lectora de tarjetas perforadas o una máquina de telescritura. Un dispositivo de salida, por ejemplo una impresora, recibe el resultado de los cálculos e imprime los resultados que se presentan al usuario. Los dispositivos de entrada y salida son sistemas digitales especiales impulsados por partes electromecánicas y controladas por circuitos digitales electrónicos.

Una calculadora electrónica es un sistema digital similar a una computadora digital, en la cual el dispositivo de entrada es un tablero de teclas y el dispositivo de salida es un exhibidor numérico. Las instrucciones entran en la calculadora mediante teclas de función, como más y menos. La información se introduce a través de las teclas numéricas. Los resultados se exhiben directamente en forma numérica. Algunas calculadoras se parecen mucho a una computadora digital si tienen capacidad de impresión y facilidades de programación. Sin embargo, una computadora digital es un dispositivo más poderoso que una calculadora. Una computadora digital debe conectarse con muchos otros dispositivos de entrada y salida, puede realizar no sólo cálculos aritméticos sino también operaciones lógicas y puede programarse para tomar decisiones basadas en condiciones internas y externas.

Una computadora digital es una interconexión de módulos digitales. Para entender la operación de cada módulo digital, es necesario tener un conocimiento

básico de los sistemas digitales y su comportamiento general. En los primeros cuatro capítulos de este libro se introducen las herramientas básicas del diseño digital, por ejemplo los números binarios y códigos, álgebra booleana y los bloques básicos con los cuales se construyen los circuitos electrónicos digitales. En los Capítulos 5 y 7 se presentan los componentes básicos que se encuentran en la unidad procesadora de una computadora digital. Las características operacionales de la unidad de memoria se explican al final del Capítulo 7. El diseño de la unidad de control se expone en el Capítulo 8 utilizando los principios básicos de los circuitos secuenciales del Capítulo 6.

Un procesador, cuando se combina con la unidad de control, forma un componente referido, una *unidad central de proceso* o CPU. Una CPU encerrada en un pequeño paquete de circuito integrado se denomina *microprocesador*. Es posible que la unidad de memoria, lo mismo que la parte que controla la interfase entre el microprocesador y los dispositivos de entrada y salida, esté encapsulada dentro del paquete del microprocesador o puede estar disponible en otros paquetes pequeños de circuitos integrados. Una CPU combinada con memoria y control de interfase para formar una computadora de tamaño pequeño se conoce como *microcomputadora*. La disponibilidad de los componentes de microcomputadoras ha revolucionado la tecnología de diseño de sistemas digitales, dando al diseñador la libertad para crear estructuras que antes eran antieconómicas. Los diversos componentes de un sistema microcomputador se construyen internamente con circuitos digitales.

Ya se ha mencionado que una computadora digital manipula elementos discretos de información y que esos elementos se representan en forma binaria. Los operandos que se utilizan para el cálculo pueden expresarse en el sistema de números binarios. Otros elementos discretos, incluyendo los dígitos decimales, se representan en códigos binarios. El procesamiento de datos se lleva a cabo mediante elementos lógicos binarios que usan señales binarias. Las cantidades se almacenan en elementos de almacenamiento binario. El propósito de este capítulo es introducir los diversos conceptos binarios como un marco de referencia para un estudio detallado en los capítulos subsecuentes.

1-2 NUMEROS BINARIOS

Un número decimal como 7392 representa una cantidad igual a 7 millares, más 3 centenas, más 9 decenas, más 2 unidades. Los millares, centenas, etc., son potencias de 10 implicadas por la posición de los coeficientes. Para ser más exactos, 7 392 debe escribirse como:

$$7 \times 10^3 + 3 \times 10^2 + 9 \times 10^1 + 2 \times 10^0$$

Sin embargo, la convención es escribir sólo los coeficientes y de sus posiciones se deducen las potencias necesarias de 10. En general, un número con un punto decimal se representa por una serie de coeficientes como sigue:

$$a_5 a_4 a_3 a_2 a_1 a_0 \cdot a_{-1} a_{-2} a_{-3}$$

Los coeficientes a_j son uno de los diez dígitos (0, 1, 2,...,9), y el valor del subíndice j da el valor del lugar y, por tanto, la potencia de 10 por la cual debe multiplicarse el coeficiente.

$$10^5a_5 + 10^4a_4 + 10^3a_3 + 10^2a_2 + 10^1a_1 + 10^0a_0 + 10^{-1}a_{-1} + 10^{-2}a_{-2} + 10^{-3}a_{-3}$$

El sistema de números decimales se dice que es de *base*, o *raíz*, 10 debido a que usa diez dígitos y los coeficientes se multiplican por potencias de 10. El sistema *binario* es un sistema diferente de números. Los coeficientes de los números del sistema binario tienen dos valores posibles: 0 y 1. Cada coeficiente a_j se multiplica por 2^j. Por ejemplo, el equivalente decimal del número binario 11010.11 es 26.75, como se muestra por la multiplicación de los coeficientes por potencias de 2:

$$1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} = 26.75$$

En general, un número expresado en un sistema base r tiene coeficientes multiplicados por las potencias de r:

$$a_n \cdot r^n + a_{n-1} \cdot r^{n-1} + \cdots + a_2 \cdot r^2 + a_1 \cdot r + a_0 + a_{-1} \cdot r^{-1} + a_{-2} \cdot r^{-2} + \cdots + a_{-m} \cdot r^{-m}$$

Los coeficientes a_j varían en valor desde 0 a $r - 1$. Para distinguir entre números de bases diferentes, se encierran entre paréntesis los coeficientes y se escribe un subíndice igual a la base usada (excepto algunas veces para números decimales, donde el contenido indica obviamente que es decimal). Un ejemplo de un número con base 5 es

$$(4021.2)_5 = 4 \times 5^3 + 0 \times 5^2 + 2 \times 5^1 + 1 \times 5^0 + 2 \times 5^{-1} = (511.4)_{10}$$

Obsérvese que los valores de los coeficientes con base 5 pueden ser sólo 0, 1, 3 y 4.

Es costumbre tomar los r dígitos necesarios para los coeficientes del sistema decimal cuando la base del número es menor que 10. Se usan las letras del alfabeto para completar los diez dígitos decimales cuando la base del número es mayor que 10. Por ejemplo, en el sistema *hexadecimal* (base 16), los primeros diez dígitos se toman del sistema decimal. Las letras A, B, C, D, E y F se utilizan para los dígitos 10, 11, 12, 13, 14 y 15, respectivamente. Un ejemplo de un número hexadecimal es:

$$(\text{B65F})_{16} = 11 \times 16^3 + 6 \times 16^2 + 5 \times 16 + 15 = (46687)_{10}$$

Los primeros 16 números en los sistemas decimal, binario, octal y hexadecimal se listan en la Tabla 1-1.

Las operaciones aritméticas con números en la base r siguen las mismas reglas que en el caso de los números decimales. Cuando se usa otra base diferente de la base familiar 10, debe tenerse cuidado de utilizar sólo los r dígitos permitidos. A continuación se muestran ejemplos de la suma, resta y multiplicación de dos números binarios:

La adición de dos números binarios se calcula según las mismas reglas que en los decimales, excepto que los dígitos de la adición en cualquier posición significativa

sumando:	101101	minuendo:	101101	multiplicando:	1011
adendo:	+ 100111	sustraendo:	− 100111	multiplicador:	× 101
suma:	1010100	diferencia:	000110		1011
					0000
					1011
				producto:	110111

pueden ser sólo 0 o 1. Cualquier "acarreo" que se obtenga en una posición significativa dada se usa por el par de dígitos en una posición significativa más alta. La sustracción es ligeramente más complicada. Las reglas siguen siendo las mismas que en el sistema decimal, excepto que "el préstamo (acarreo)" en una posición significativa dada añade 2 a un dígito minuendo. (Un "préstamo" en el sistema decimal añade 10 a un dígito minuendo.) La multiplicación es muy simple. Los dígitos multiplicadores siempre son 1 o 0. Así que los productos parciales son iguales ya sea al multiplicando o bien a 0.

1-3 CONVERSIONES DE LA BASE DE NUMEROS

Un número binario puede convertirse en decimal formando la suma de las potencias de dos de los coeficientes cuyo valor es 1. Por ejemplo:

$$(1010.011)_2 = 2^3 + 2^1 + 2^{-2} + 2^{-3} = (10.375)_{10}$$

TABLA 1-1 Números con bases diferentes

Decimal (base 10)	Binario (base 2)	Octal (base 8)	Hexadecimal (base 16)
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

El número binario tiene cuatro números 1 y el equivalente decimal se encuentra por la adición de cuatro potencias de 2. En forma similar, un número que se expresa en la base r puede convertirse en su equivalente decimal multiplicando cada coeficiente por la potencia correspondiente de r y sumando. El siguiente es un ejemplo de la conversión de octal en decimal:

$$(630.4)_8 = 6 \times 8^2 + 3 \times 8 + 4 \times 8^{-1} = (408.5)_{10}$$

La conversión de decimal en binario o cualquier otro sistema con base r es más conveniente si el número se separa en una *parte entera* y en una *parte fraccional*, y la conversión de cada parte se hace por separado. La conversión de un entero de decimal en binario se explica de manera mas adecuada con un ejemplo.

EJEMPLO 1-1: Convierta el decimal 41 en binario. Primero, 41 se divide entre 2 para dar un cociente entero de 20 y un residuo de 1/2. El cociente se divide otra vez entre 2 para dar un nuevo cociente y un residuo. Este proceso se continúa hasta que el entero cociente llega a ser 0. Los *coeficientes* del número binario deseado se obtienen por los *residuos* como sigue:

	cociente entero		*residuo*	*coeficiente*
$\frac{41}{2}$	$= 20$	$+$	$\frac{1}{2}$	$a_0 = 1$
$\frac{20}{2}$	$= 10$	$+$	0	$a_1 = 0$
$\frac{10}{2}$	$= 5$	$+$	0	$a_2 = 0$
$\frac{5}{2}$	$= 2$	$+$	$\frac{1}{2}$	$a_3 = 1$
$\frac{2}{2}$	$= 1$	$+$	0	$a_4 = 0$
$\frac{1}{2}$	$= 0$	$+$	$\frac{1}{2}$	$a_5 = 1$

respuesta: $(41)_{10} = (a_5a_4a_3a_2a_1a_0)_2 = (101001)_2$

El proceso aritmético puede manipularse en forma más conveniente como sigue:

entero	*residuo*	
41		
20	1	
10	0	
5	0	
2	1	
1	0	
0	1	101001 = *respuesta*

La conversión de enteros decimales en cualquier sistema con base r es similar al ejemplo anterior, excepto que la división se hace entre r en lugar de 2.

EJEMPLO 1-2: Convierta el decimal 153 en octal. La base requerida r es 8. Primero, 153 se divide entre 8 para dar un cociente entero de 19 y un residuo de 1. Entonces 19 se divide entre 8 para dar un cociente entero de 2 y un residuo de 3. Por último, 2 se divide entre 8 para dar un cociente de 0 y un residuo de 2. Este proceso puede manipularse en forma conveniente como sigue:

153	
19	1
2	3
0	2 $= (231)_8$

La conversión de una fracción decimal en binario se lleva a cabo por un método similar al que se utiliza para los enteros. Sin embargo, se usa la multiplicación en lugar de la división, y los enteros se acumulan en lugar de residuos. De nuevo, el método se explica de manera más adecuada con un ejemplo.

EJEMPLO 1-3: Convierta $(0.6875)_{10}$ en binario. Primero 0.6875 se multiplica por 2 para dar un entero y una fracción. La nueva fracción se multiplica por 2 para dar un entero y una nueva fracción. Este proceso se continúa hasta que la fracción llega a ser 0 o hasta que el número de dígitos tiene suficiente exactitud. Los coeficientes del número binario se obtienen mediante los enteros como sigue:

	entero		*fracción*	*coeficiente*
$0.6875 \times 2 =$	1	+	0.3750	$a_{-1} = 1$
$0.3750 \times 2 =$	0	+	0.7500	$a_{-2} = 0$
$0.7500 \times 2 =$	1	+	0.5000	$a_{-3} = 1$
$0.5000 \times 2 =$	1	+	0.0000	$a_{-4} = 1$

respuesta: $(0.6875)_{10} = (0.a_{-1}a_{-2}a_{-3}a_{-4})_2 = (0.1011)_2$

Para convertir una fracción decimal en un número expresado en base r, se usa un procedimiento similar. La multiplicación se hace por r en lugar de 2, y los coeficientes se encuentran por los enteros que pueden variar el valor desde 0 a $r - 1$ en lugar de 0 y 1.

EJEMPLO 1-4: Convierta $(0.513)_{10}$ en octal.

$$0.513 \times 8 = 4.104$$
$$0.104 \times 8 = 0.832$$
$$0.832 \times 8 = 6.656$$
$$0.656 \times 8 = 5.248$$
$$0.248 \times 8 = 1.984$$
$$0.984 \times 8 = 7.872$$

La respuesta, a siete cifras significativas, se obtiene de la parte entera de los productos:

$$(0.513)_{10} = (0.406517\ldots)_8$$

La conversión de números decimales tanto con las partes enteras como fraccionarias se hace por la conversión de la parte entera y fraccionaria por separado y, entonces, combinando las dos respuestas. Por el uso de los resultados de los Ejemplos 1-1 y 1-3; se obtiene:

$$(41.6875)_{10} = (101001.1011)_2$$

A partir de los Ejemplos 1-2 y 1-4, se obtiene:

$$(153.513)_{10} = (231.406517)_8$$

1-4 NUMEROS OCTALES Y HEXADECIMALES

La conversión de binario, octal y hexadecimal —y a la inversa— juega una parte importante en las computadoras digitales. Ya que $2^3 = 8$ y $2^4 = 16$, cada dígito octal corresponde a tres dígitos binarios y cada dígito hexadecimal corresponde a cuatro dígitos binarios. La conversión de binario en octal se lleva a cabo fácilmente por la partición del número binario en grupos de tres dígitos cada uno, principiando desde el punto binario y procediendo a la izquierda y a la derecha. Entonces, el dígito octal correspondiente se asigna a cada grupo. El siguiente ejemplo ilustra el procedimiento:

$$\begin{array}{cccccccccccc} (& \underbrace{10} & \underbrace{110} & \underbrace{001} & \underbrace{101} & \underbrace{011} & & \underbrace{111} & \underbrace{100} & \underbrace{000} & \underbrace{110} &)_2 = (26153.7406)_8 \\ & 2 & 6 & 1 & 5 & 3 & & 7 & 4 & 0 & 6 & \end{array}$$

La conversión de binario en hexadecimal es similar, excepto que el número binario se divide en grupos de cuatro dígitos.

$$\begin{array}{cccccccc} (& \underbrace{10} & \underbrace{1100} & \underbrace{0110} & \underbrace{1011} & & \underbrace{1111} & \underbrace{0010} &)_2 = (2C6B.F2)_{16} \\ & 2 & C & 6 & B & & F & 2 & \end{array}$$

El dígito correspondiente hexadecimal (u octal) para cada grupo de dígitos binarios se recuerda con facilidad después de estudiar los valores listados en la Tabla 1-1.

La conversión de octal o hexadecimal en binario se realiza por un procedimiento inverso al anterior. Cada dígito octal se convierte en su equivalente binario de tres dígitos. De manera semejante, cada dígito hexadecimal se convierte en su equivalente binario de cuatro dígitos. Esto se ilustra en los siguientes ejemplos:

$$(673.124)_8 = (\underbrace{110}_{6}\ \underbrace{111}_{7}\ \underbrace{011}_{3} \cdot \underbrace{001}_{1}\ \underbrace{010}_{2}\ \underbrace{100}_{4})_2$$

$$(306.D)_{16} = (\underbrace{0011}_{3}\ \underbrace{0000}_{0}\ \underbrace{0110}_{6} \cdot \underbrace{1101}_{D})_2$$

Los números binarios son díficiles de trabajar ya que requieren tres o cuatro veces más dígitos que su equivalente decimal. Por ejemplo, el número binario 111111111111 es equivalente al decimal 4 095. No obstante, las computadoras digitales utilizan números binarios y algunas veces es necesario que el operador humano o usuario se comunique en forma directa con la máquina mediante números binarios. Un esquema que retiene el sistema binario en la computadora, pero que reduce el número de dígitos que el humano debe considerar, emplea la relación entre el sistema de números binarios y el sistema octal o hexadecimal. Por este método, el humano piensa en términos de números octales o hexadecimales y lleva a cabo la conversión requerida por inspección, cuando es necesaria la comunicación directa con la máquina. Por tanto, el número binario 111111111111 contiene doce dígitos y se expresa en octal, 7777 (cuatro dígitos) o en hexadecimal como FFF (tres dígitos). Durante la comunicación entre personas (acerca de los números binarios en la computadora), la representación octal o hexadecimal es más deseable debido a que puede expresarse en forma compacta con un tercio o un cuarto del número de dígitos requeridos por el número binario equivalente. Cuando el humano se comunica con la máquina (a través de interruptores en la consola con luces indicadoras, o mediante programas escritos en *lenguaje de máquina*), la conversión de octal o hexadecimal en binario y viceversa se hace por inspección por el usuario humano.

1-5 COMPLEMENTOS

Los complementos se usan en las computadoras digitales para simplificar la operación de sustracción y para manipulaciones lógicas. Hay dos tipos de complementos para cada sistema base r: (1) el complemento de r y (2) el complemento de $(r-1)$. Cuando el valor de la base se sustituye, los dos tipos reciben los nombres de 2 y de 1 para complementos de números binarios o, de 10 y de 9 para complementos de números decimales.

El complemento de r

Dado un número positivo N en base r con una parte entera de n dígitos, el complemento de r de N se define como $r^n - N$ para $N \neq 0$ y 0 para $N = 0$. El siguiente ejemplo numérico ayudará a aclarar la definición:

El complemento de 10 $(52\,520)_{10}$ es $10^5 - 52\,520 = 47\,480$

El número de dígitos en el número es $n = 5$.

El complemento de 10 de $(0.3267)_{10}$ es $1 - 0.3267 = 0.6733$.

No hay parte entera, de modo que $10^n = 10^0 = 1$.

El complemento de 10 de $(25.639)_{10}$ es $10^2 - 25.639 = 74.361$.

El complemento de 2 de $(101100)_2$ es $(2^6)_{10} - (101100)_2 = (1000000 - 101100)_2 = 010100$.

El complemento de 2 de $(0.0110)_2$ es $(1 - 0.0110)_2 = 0.1010$.

Por la definición y los ejemplos, es claro que el complemento de 10 de un número decimal puede formarse dejando todos los ceros significativos sin cambio, se resta el primer dígito de cero menos significativo de 10 y, entonces, se restan todos los otros dígitos más significativos de 9. El complemento de 2 puede formarse dejando todos los ceros menos significativos y el primer dígito no cero sin cambio y, entonces, se reemplazan los 1 por 0 y los 0 por 1 en todos los otros dígitos más significativos. Un tercer método más simple para obtener el complemento de r se da después de la definición del complemento $(r - 1)$.

El complemento de r de un número existe para cualquier base r (r mayor que pero no igual a 1) y puede obtenerse por la definición que se presentó antes. Los ejemplos listados aquí usan números con $r = 10$ (decimal) y $r = 2$ (binario), debido a que estas son las dos bases de mayor interés para nosotros. El nombre del complemento se relaciona con la base del número que se usa. Por ejemplo, el complemento de $(r - 1)$ de un número en la base 11 se denomina complemento de 10, ya que $r - 1 = 10$ para $r = 11$.

El complemento de $(r - 1)$

Dado un número positivo N en base r con una parte entera de n dígitos y una parte fraccionaria de m dígitos, el complemento de $(r - 1)$ de N se define como $r^n - r^{-m} - N$. A continuación se presentan algunos ejemplos numéricos:

El complemento de 9 de $(52\,520)_{10}$ es $(10^5 - 1 - 52\,520) = 99\,999 - 52\,520 = 47\,479$.

No hay parte fraccionaria, de modo que $10^{-m} = 10^0 = 1$.

El complemento de 9 de $(0.3267)_{10}$ es $(1 - 10^{-4} - 0.3267) = 0.9999 - 0.3267 = 0.6732$.

No hay parte entera, de modo que $10^n = 10^0 = 1$.

El complemento de 9 de $(25.639)_{10}$ es $(10^2 - 10^{-3} - 25.739) = 99.999 - 25.639 = 74.360$.

El complemento de 1 de $(101100)^2$ es $(2^6 - 1) - (101100) = (111111 - 101100)_2 = 010011$.

El complemento de 1 de $(0.0110)_2$ es $(1 - 2^{-4})_{10} - (0.0110)_2 - (0.1111 - 0.0110)_2 = 0.1001$.

Por los ejemplos, puede observarse que el complemento de 9 de un número decimal se forma simplemente al restar cada dígito de 9. El complemento de 1 de un número binario es aun más simple de formar: los 1 se cambian en 0 y los 0 en 1. Ya que el complemento de $(r - 1)$ se obtiene de manera muy fácil, algunas veces es conveniente usarlo cuando se desea el complemento de r. Por las definiciones y mediante una comparación de los resultados que se obtuvieron en los ejemplos, se concluye que el complemento de r puede obtenerse del complemento $(r - 1)$ después de la adición de r^{-m} al dígito menos significativo. Por ejemplo, el complemento de 2 de 10110100 se obtiene del complemento de 1 de 01001011 por la adición de 1 para obtener 01001100.

Vale la pena mencionar que el complemento del complemento restablece el número a su valor original. El complemento r de N es $r^n - N$ y el complemento de $(r^n - N)$ es $r^n - (r^n - N) = N$; y en forma similar para el complemento de 1.

Sustracción con complementos de r

El método directo de sustracción que se enseña en las escuelas elementales usa el concepto de préstamo. En este método, se presta un 1 de una posición significativa más alta cuando el dígito minuendo es menor que el dígito sustraendo correspondiente. Esto parece ser más fácil cuando las personas realizan la resta con lápiz y papel. Cuando la resta se implanta mediante componentes digitales, se encuentra que este método es menos eficiente que el método que utiliza complementos y suma como se establece más adelante.

La sustracción de dos números positivos $(M - N)$, ambos en base r, puede hacerse como sigue:

1. Agréguese el minuendo M al complemento de r del sustraendo N.
2. Inspecciónese el resultado que se obtuvo en el paso 1 para un "acarreo final":
   (a) Si ocurre un "acarreo final", descártese.
   (b) Si no ocurre un "acarreo final", tómese el complemento r del número que se obtuvo en el paso 1 y colóquese un signo negativo enfrente.

Los siguientes ejemplos ilustran el procedimiento:

EJEMPLO 1-5: Usando el complemento de 10, reste 72 532 − 3 250.

$M = 72532$ 72532
$N = 03250$

\+

10 complemento de $N = 96\,750$ 96750

acarreo final → 1 69282

respuesta: 69282

EJEMPLO 1-6: Reste: $(3\,250 - 72\,532)_{10}$.

$M = 03250$ 03250
$N = 72532$

\+
10 complemento de $N = 27468$ 27468
no acarreo 30718

respuesta: —69 282 = — (10 complemento de 30 618).

EJEMPLO 1-7: Utilice el complemento de 2 para realizar $M - N$ con los números binarios dados.

(a) $M = 1010100$ 1010100
$N = 1000100$

\+
complemento de 2 de $N = 0111100$ 0111100
acarreo final 1 0010000

respuesta: 10 000

(b) $M = 1000100$ 1000100
$N = 1010100$

\+
2 complemento de $N = 0101100$ 0101100
no acarreo 1110000

respuesta: — 10 000 = — (2 complemento de 1110000)

La prueba del procedimiento es: La adición de M al complemento r de N da $(M + r^n - N)$. Para números que tienen una parte entera de n dígitos, r^n es igual a 1 en la posición $(n + 1)$ava (lo que se ha denominado "acarreo final"). Ya que se supone que tanto M como N son positivos, entonces:

$$(a) \quad (M + r^n - N) \geq r^n \quad \text{si } M \geq N, \quad \text{o}$$
$$(b) \quad (M + r^n - N) < r^n \quad \text{si } M < N$$

En el caso (a) la respuesta es positiva e igual a $M - N$, la cual se obtiene en forma directa descartando y se lleva cuenta final r^n. En el caso (b), la respuesta es negativa e igual a $-(N - M)$. Este caso se detecta con la ausencia de un acarreo final. La respuesta se obtiene tomando un segundo complemento y añadiendo un signo negativo:

$$-[r^n - (M + r^n - N)] = -(N - M).$$

Sustracción con el complemento de $(r - 1)$

El procedimiento para la resta con el complemento de $(r - 1)$ es exactamente el mismo que se usó con el complemento de r, excepto por una variación, llamada "acarreo final desplazado", como se muestra adelante. La resta de $M - N$, con ambos números positivos en la base r, puede calcularse en la siguiente forma:

1. Agréguese el minuendo M al complemento de $(r - 1)$ del sustraendo N.
2. Inspecciónese el resultado que se obtuvo en el paso 1 para un acarreo final.
   (a) Si ocurre un acarreo final, agréguese 1 al dígito menos significativo (acarreo final desplazado).
   (b) Si no ocurre un acarreo final, tómese el complemento de $(r - 1)$ del número obtenido en el paso 1 y colóquese al frente un signo negativo.

La prueba de este procedimiento es muy semejante a la dada para el caso de complemento de r y se deja como ejercicio. Los ejemplos siguientes ilustran el procedimiento.

EJEMPLO 1-8: Repita los Ejemplos 1-5 y 1-6 usando complementos de 9.

(a)

$M = 72532$
$N = 03250$
9 complemento de $N = 96749$

```
        72532
     +  96749
     ---------
  1 /   69281
acarreo final                +
y se pasa a la derecha  →   1
        -------
        69282
```

respuesta: 69282

(b)

$M = 03250$
$N = 72532$
9 complemento de $N = 27467$

```
           03250
     +     27467
     -----------
no acarreo  30717
```

respuesta: $-69282 = -$ (9 complemento de 30717)

EJEMPLO 1-9: Repita el Ejemplo 1-7 utilizando complementos de 1.

(a) $M = 1010100$
$N = 1000100$
1 complemento de $N = 0111011$

acarreo final
y se pasa a la derecha

```
 1010100
+0111011
 1 0001111
+       1
 0010000
```

respuesta: 10000

(b) $M = 1000100$
$N = 1010100$
1 complemento de $N = 0101011$

```
 1000100
+0101011
 1101111
```

no acarreo

respuesta: $-10000 = -$ (1 complemento de 1101111)

Comparación entre los complementos de 1 y 2

Una comparación entre los complementos de 1 y 2 revela las ventajas y desventajas de cada uno. El complemento de 1 tiene la ventaja de ser más fácil de implantar por componentes digitales, ya que lo único que debe hacerse es cambiar los 0 en números 1 y los 1 en números 0. El implante del complemento de 2 puede obtenerse en dos formas: (1) por la adición de 1 al dígito menos significativo del complemento de 1 y (2), dejando todos los 0 precedentes en las posiciones menos significativas y el primer 1 sin cambio, y sólo entonces cambiar todos los 1 en 0 y todos los 0 en 1. Durante la resta de dos números por complementos, el complemento de 2 es ventajoso ya que sólo se requiere una operación aritmética de adición. El complemento de 1 necesita dos adiciones aritméticas cuando ocurre un acarreo final desplazado. El complemento de 1 tiene la desventaja adicional de poseer dos ceros aritméticos: uno con todos los 0 y uno con todos los 1. Para ilustrar este hecho considérese la resta de dos números binarios iguales $1100 - 1100 = 0$.

Utilizando el complemento de 1:

```
  1100
+ 0011
+ 1111
```

Se complementa de nuevo para obtener —0000.

Usando el complemento de 2:

$$\begin{array}{rr} & 1100 \\ + & \\ & 0100 \\ + & \overline{0000} \end{array}$$

Mientras que el complemento de 2 tiene sólo un cero aritmético, el complemento de 1 cero puede ser positivo o negativo, lo cual puede complicar las cosas.

Los complementos, muy útiles para las manipulaciones aritméticas en computadoras digitales, se exponen con más detalle en los Capítulos 8 y 9. Sin embargo, el complemento de 1 también es útil en manipulaciones lógicas (como se mostrará después), ya que el cambio de 1 a 0 y viceversa es equivalente a una operación lógica de inversión. El complemento de 2 se utiliza sólo en conjunción con aplicaciones aritméticas. En consecuencia, es conveniente aceptar la siguiente convención: Cuando la palabra *complemento*, sin mención del tipo, se usa en conjunción con una aplicación no aritmética, se supone que el tipo es complemento de 1.

1-6 CODIGOS BINARIOS

Los sistemas electrónicos digitales utilizan señales que tienen dos valores distintos y elementos de circuito que tienen dos estados estables. Hay una analogía directa entre las señales binarias, los elementos de circuito binario y dígito binario. Un número binario de n dígitos, por ejemplo, puede representarse por n elementos de números binarios, cada uno con una señal de salida equivalente a 0 o a 1. Los sistemas digitales representan y manipulan no sólo números binarios, sino también otros muchos elementos discretos de información. Cualquier elemento discreto de información distinto entre un grupo de cantidades puede representarse por un código binario. Por ejemplo, el *rojo* es un color definido del espectro. La letra *A* es una letra distinta del alfabeto.

Un *bit*, por definición, es un dígito binario. Cuando se usa junto con un código binario, es mejor considerarlo como si denotara una cantidad binaria igual a 0 o 1. Para representar un grupo de 2^n elementos distintos en un código binario, se requiere un mínimo de n bits. Esto se debe a que es posible ordenar n bits en 2^n formas distintas. Por ejemplo, un grupo de cuatro cantidades diferentes puede representarse mediante un código de 2 bits, con cada cantidad asignada a una de las siguientes combinaciones de bit: 00, 01, 10, 11. Un grupo de ocho elementos requiere un código de tres bits, con cada elemento asignado a uno y sólo uno de los siguientes: 000, 001, 010, 011, 100, 101, 110, 111. En los ejemplos se muestra que las distintas combinaciones de bits para un código de n bits pueden encontrarse al contar en binario desde 0 a $(2^n - 1)$. Algunas combinaciones de bit quedan sin asignarse cuando el número de elementos del grupo que va a codificarse no es un múltiplo de la potencia de 2. Los diez dígitos decimales 0, 1, 2,..., 9 son un ejemplo de tal clase de grupo. Un código binario que se distingue entre diez elementos debe contener cuando menos cuatro bits; tres bits pueden distinguir un máximo de ocho elementos. Cuatro bits pueden formar 16 combinaciones distintas, pero como sólo se codifican diez dígitos, las seis combinaciones restantes quedan sin asignar y no se utilizan.

Aunque el número *mínimo* de bits necesario para codificar 2^n cantidades distintas es n, no hay número *máximo* de bits que puedan usarse para un código binario. Por ejemplo, los diez dígitos decimales pueden codificarse con diez bits y, cada dígito decimal se asigna a una combinación de bits de nueve números 0 y un 1. En este código binario particular, al dígito 6 se le asigna la combinación de bits 0001000000.

Códigos decimales

Los códigos binarios para dígitos decimales requieren un mínimo de cuatro bits. Se obtienen numerosos códigos diferentes al ordenar cuatro o más bits en diez distintas combinaciones. Unas cuantas posibilidades se muestran en la Tabla 1-2.

El BCD (de las iniciales en inglés para decimal codificado binario) es una asignación directa del equivalente binario. Es posible asignar pesos a los bits binarios de acuerdo con sus posiciones. Los pesos en el código BCD son 8, 4, 2, 1. Por ejemplo, la asignación de bits 0110 puede interpretarse por los pesos que representan el dígito decimal 6, porque $0 \times 8 + 1 \times 4 + 1 \times 2 + 0 \times 1 = 6$. También es factible asignar pesos negativos a un código decimal, como se muestra por el código 8, 4, -2, -1. En este caso la combinación de bits 0110 se interpreta como el dígito decimal 2, cuando se obtiene por $0 \times 8 + 1 \times 4 + 1 \times (-2) + 0 \times (-1) = 2$. Los otros dos códigos pesados que se muestran en la tabla son 2421 y el 5043210. Un código decimal que se ha usado en algunas computadoras antiguas es el código exceso-3. Este es un código sin peso; sus asignaciones se obtienen por el valor correspondiente del BCD después de la adición de 3.

Los números se representan en las computadoras digitales ya sea en binario o en decimal a través de un código binario. Cuando se especifican los datos, al usuario le gusta dar información en forma decimal. Los números decimales de entrada se almacenan internamente en la computadora mediante un código decimal. Cada dígito decimal requiere cuando menos el almacenamiento de cuatro elementos binarios. Los números decimales se convierten en binarios cuando las operaciones aritméticas se hacen de manera interna con números representados en binario. También es posible

TABLA 1-2 Códigos binarios para los dígitos decimales

Dígito decimal	(BCD) 8421	Exceso-3	84-2-1	2421	(Biquinario) 5043210
0	0000	0011	0000	0000	0100001
1	0001	0100	0111	0001	0100010
2	0010	0101	0110	0010	0100100
3	0011	0110	0101	0011	0101000
4	0100	0111	0100	0100	0110000
5	0101	1000	1011	1011	1000001
6	0110	1001	1010	1100	1000010
7	0111	1010	1001	1101	1000100
8	1000	1011	1000	1110	1001000
9	1001	1100	1111	1111	1010000

llevar a cabo en forma directa las operaciones aritméticas en decimal, con todos los números que quedan en una forma codificada a través del proceso. Por ejemplo, el número decimal 395, cuando se convierte en binario es igual a 110001011 y consta de nueve dígitos binarios. El mismo número, cuando se representa internamente en el código BCD, ocupa cuatro bits para cada dígito decimal, para un total de 12 bits: 001110010101. Los primeros cuatro bits representan un 3, los siguientes cuatro un 9 y los últimos cuatro un 5.

Es muy importante entender la diferencia entre *conversión* de un número decimal en binario y la *codificación* binaria de un número decimal. En cada caso el resultado final es una serie de bits. Los bits que se obtienen por conversión son dígitos binarios. Los bits que se obtienen por la codificación son combinaciones de 1 y 0 arreglados de acuerdo con las reglas del código que se emplee. Por consiguiente, es de extrema importancia darse cuenta de que una serie de números 1 y 0 en un sistema digital algunas veces representa un número binario y en otras ocasiones representa alguna otra cantidad discreta de información como se especifique por un código binario dado. Por ejemplo, el código BCD se ha escogido para que sea tanto un código como una conversión binaria discreta, en tanto los números decimales sean enteros desde 0 a 9. Para números mayores de 9, la conversión y la codificación son por completo diferentes. Este concepto es tan importante que vale la pena repetirlo con otro ejemplo. La conversión binaria del decimal 13 es 1101; la verificación del decimal 13 con el código BCD es 00010011.

De los cinco códigos binarios que se listan en la Tabla 1-2, el BCD parece ser el de uso más natural y por supuesto es el que se encuentra por lo común. Los otros códigos de cuatro bits listados tienen una característica en común que no se encuentra en el BCD. El exceso-3 el 2, 4, 2, 1, y el 8, 4, −2, −1, son códigos autocomplementarios, es decir, el complemento a 9 del número decimal se obtiene fácilmente cambiando los números 1 a 0 y los 0 a 1. Por ejemplo, el decimal 395 se representa en el código 2, 4, 2, 1, por 001111111011. El complemento a 9 de 604, se representa por 110000000100, lo cual se obtiene por el reemplazo de los 1 por 0 y los 0 por 1. Esta propiedad es útil cuando se hacen operaciones aritméticas en forma interna con números decimales (en un código binario) y la resta se calcula mediante el complemento a 9.

El código biquinario que se muestra en la Tabla 1-2 es un ejemplo de un código de siete bits con propiedades de detección de error. Cada dígito decimal consta de cinco números 0 y dos 1 colocados en las columnas pesadas correspondientes. Es posible comprender la propiedad de detección de error de este código al percatarse de que el sistema digital representa el binario 1 por una señal distinta y el binario 0 por una segunda diferente. Durante la transmisión de señales de una localidad a otra es factible que ocurra un error. Uno o más bits pueden cambiar de valor. Un circuito en el lado receptor puede detectar la presencia de más (o menos) dos números 1 y, si la combinación recibida de bits no coincide con esta combinación permitida, se detecta error.

Códigos de detección de error

La información binaria, se trate de señales de pulso modulado o bien, entrada o salida digital a computadora, puede transmitirse a través de alguna forma de medio de

comunicación, como alambres u ondas de radio. Cualquier ruido externo que se introduce en un medio de comunicación física cambia los valores de bits de 0 a 1 y viceversa. Puede utilizarse un código de detección de errores para detectar los errores cuando se realiza la transmisión. No es posible corregir el error detectado pero se indica su presencia. El procedimiento usual es observar la frecuencia de errores. Si el error ocurre sólo de manera esporádica, al azar y sin efecto pronunciado en la información global transmitida, entonces no se hace nada o el mensaje erróneo en particular se transmite otra vez. Si ocurren errores con frecuencia y se distorciona el significado de la información recibida, se verifica el sistema por mal funcionamiento.

Un bit de *paridad* es un bit adicional incluido con un mensaje para hacer que el número total de los 1 sea impar o par. Un mensaje de cuatro bits y un bit de paridad, P, se muestran en la Tabla 1-3. En (a), P se escoge de modo que la suma de todos los números 1 sea impar (un total de cinco bits). En (b), P se escoge de modo que la suma de todos los 1 sea par. Durante la transferencia de información de una localidad a otra, el bit de paridad se manipula como sigue: Al final del envío, el mensaje (en este caso los primeros cuatro bits) se aplica a un circuito "generador de paridad", donde el bit P requerido se genera. El mensaje, incluyendo el bit de paridad, se transfiere a su destino. En el extremo receptor, todos los bits que llegan (en este caso cinco) se aplican a un circuito "de verificación de paridad" para verificar la apropiada paridad adoptada. Se detecta un error si la paridad verificada no corresponde a la adoptada. El método de paridad detecta la presencia de uno, tres o cualquier combinación impar de errores. Una combinación par de errores no se detecta. En la Sección 4-9 se encontrará un análisis adicional de la generación de paridad y verificación.

El código reflejado

Los sistemas digitales pueden diseñarse para procesar datos sólo en una forma discreta. Muchos sistemas físicos suministran salida de información continua. Esta información puede convertirse en forma digital o discreta antes de que se aplique a un sistema digital. La información continua o analógica se convierte en forma digital mediante un convertidor de analógico a digital. Algunas veces es conveniente usar el código reflejado, que se muestra en la Tabla 1-4, para representar la información digital convertida en una información analógica. La ventaja del código reflejado sobre los números binarios puros es que un número en el código reflejado cambia sólo por un bit conforme proceda de un número al siguiente. Una aplicación típica del código reflejado ocurre cuando la información analógica se representa por el cambio continuo de una posición de eje. El eje está dividido en segmentos y, a cada segmento se le asigna un número. Si se hace que los segmentos adyacentes correspondan a los números adyacentes de código reflejado, la ambigüedad se reduce cuando la detección se percibe en la línea que separa dos segmentos cualesquiera. El código reflejado que se muestra en la Tabla 1-4 es sólo uno de muchos códigos posibles de esta clase. Para obtener un código reflejado diferente, puede comenzarse con cualquier combinación de bits y obtener después la siguiente combinación de bits cambiando sólo un bit de 0 a 1, o de 1 a 0, en cualquier forma que se haga al azar, en tanto que dos números no tengan asignaciones idénticas de código. El código reflejado también se conoce como código *Gray*.

TABLA 1-3 Generación de bits de paridad

(a) Mensaje	P (impar)	(b) Mensaje	P (par)
0000	1	0000	0
0001	0	0001	1
0010	0	0010	1
0011	1	0011	0
0100	0	0100	1
0101	1	0101	0
0110	1	0110	0
0111	0	0111	1
1000	0	1000	1
1001	1	1001	0
1010	1	1010	0
1011	0	1011	1
1100	1	1100	0
1101	0	1101	1
1110	0	1110	1
1111	1	1111	0

TABLA 1-4 Código reflejado de 4 bits

Código reflejado	Equivalencia decimal
0000	0
0001	1
0011	2
0010	3
0110	4
0111	5
0101	6
0100	7
1100	8
1101	9
1111	10
1110	11
1010	12
1011	13
1001	14
1000	15

Códigos alfanuméricos

Muchas de las aplicaciones de las computadoras digitales requieren la manipulación de datos que constan no sólo de números, sino también de letras. Por ejemplo, una compañía de seguros con millones de tenedores de póliza debe usar una computadora digital para procesar sus archivos. Para representar el nombre del tenedor de póliza en forma binaria, es necesario tener un código binario para el alfabeto. Además, el mismo código binario debe representar números decimales y algunos otros caracteres especiales. Un código alfanumérico (algunas veces abreviado *alfamérico*) es un código binario de un grupo de elementos que constan de diez dígitos decimales, las 26 letras del alfabeto y cierto número de símbolos especiales como $. El número total de elementos en un grupo alfanumérico es mayor de 36. Por lo tanto, debe codificarse con un mínimo de seis bits ($2^6 = 64$, pero $2^5 = 32$ no es suficiente).

Un arreglo posible de un código alfanumérico de seis bits se muestra en la Tabla 1-5 bajo el nombre de "código interno". Con unas pocas variaciones, se utiliza en muchas computadoras para representar en forma interna los caracteres alfanuméricos. La necesidad de representar más de 64 caracteres (las letras minúsculas y algunos caracteres especiales de control para la transmisión de información digital) dieron lugar a los códigos alfanuméricos de siete y ocho bits. Uno de dichos códigos se conoce como el ASCII (American Standard Code for Information Interchange); otro se conoce como EBCDIC (Extended BCD Interchange Code). El código ASCII que se lista en la Tabla 1-5 consta de siete bits pero es, en la práctica, un código de ocho bits debido a que de manera invariable se agrega un octavo bit por paridad. Cuando se transfiere información discreta a través de tarjetas perforadas, los caracteres alfanuméricos usan un código binario de doce bits. Una tarjeta perforada consta de 80 columnas y 12 renglones. En cada columna se representa un carácter alfanumérico mediante agujeros perforados en los renglones apropiados. Un agujero se detecta como un 1 y la ausencia de perforación como 0. Los 12 renglones están marcados, principiando desde la parte superior, como la perforación 12, 11, 0, 1, 2,..., 9. Los primeros tres se denominan perforación de *zona* y los últimos nueve se conocen como perforación *numérica*. El código de tarjeta de 12 bits que se muestra en la Tabla 1-5 lista los renglones cuando se perfora un agujero (dando los números 1). Se supone que los renglones restantes no listados son 0. El código de tarjeta de 12 bits es ineficiente con respecto al número de bits que se utilizan. La mayoría de las computadoras traducen el código de entrada a un código externo de 6 bits. Como ejemplo, la representación en el código interno de nombre de "John Doe" es:

100001	100110	011000	100101	110000	010100	100110	010101
J	O	H	N	espacio	D	O	E

1-7 ALMACENAMIENTO BINARIO Y REGISTROS

Los elementos discretos de información en una computadora digital deben tener una existencia física en ciertos medios de almacenamiento de información. Además,

TABLA 1-5 Códigos de caracteres alfanuméricos

Carácter	Código interno 6-Bit	Código ASCII 7-Bit	Código EBCDIC 8-Bit	Código de tarjeta 12-Bit
A	010 001	100 0001	1100 0001	12,1
B	010 010	100 0010	1100 0010	12,2
C	010 011	100 0011	1100 0011	12,3
D	010 100	100 0100	1100 0100	12,4
E	010 101	100 0101	1100 0101	12,5
F	010 110	100 0110	1100 0110	12,6
G	010 111	100 0111	1100 0111	12,7
H	011 000	100 1000	1100 1000	12,8
I	011 001	100 1001	1100 1001	12,9
J	100 001	100 1010	1101 0001	11,1
K	100 010	100 1011	1101 0010	11,2
L	100 011	100 1100	1101 0011	11,3
M	100 100	100 1101	1101 0100	11,4
N	100 101	100 1110	1101 0101	11,5
O	100 110	100 1111	1101 0110	11,6
P	100 111	101 0000	1101 0111	11,7
Q	101 000	101 0001	1101 1000	11,8
R	101 001	101 0010	1101 1001	11,9
S	110 010	101 0011	1110 0010	0,2
T	110 011	101 0100	1110 0011	0,3
U	110 100	101 0101	1110 0100	0,4
V	110 101	101 0110	1110 0101	0,5
W	110 110	101 0111	1110 0110	0,6
X	110 111	101 1000	1110 0111	0,7
Y	111 000	101 1001	1110 1000	0,8
Z	111 001	101 1010	1110 1001	0,9
0	000 000	011 0000	1111 0000	0
1	000 001	011 0001	1111 0001	1
2	000 010	011 0010	1111 0010	2
3	000 011	011 0011	1111 0011	3
4	000 100	011 0100	1111 0100	4
5	000 101	011 0101	1111 0101	5
6	000 110	011 0110	1111 0110	6
7	000 111	011 0111	1111 0111	7
8	001 000	011 1000	1111 1000	8
9	001 001	011 1001	1111 1001	9
espacio	110 000	010 0000	0100 0000	sin perforación
.	011 011	010 1110	0100 1011	12,8,3
(	111 100	010 1000	0100 1101	12,8,5
+	010 000	010 1011	0100 1110	12,8,6
$	101 011	010 0100	0101 1011	11,8,3
*	101 100	010 1010	0101 1100	11,8,4
)	011 100	010 1001	0101 1101	11,8,5
−	100 000	010 1101	0110 0000	11
/	110 001	010 1111	0110 0001	0,1
,	111 011	010 1100	0110 1011	0,8,3
=	001 011	011 1101	0111 1110	8,6

cuando los elementos discretos de información se representan en forma binaria, el medio de almacenamiento de la información debe contener elementos binarios de almacenamiento, para almacenar bits individuales. Una *celda binaria* es un dispositivo que posee dos estados estables y es capaz de almacenar un bit de información. La entrada a la celda recibe señales de excitación que la fijan a uno de los dos estados. La salida de la celda es una cantidad física que distingue entre los dos estados. La información almacenada en una celda es 1 cuando está en un estado estable y 0 cuando está en el otro estado estable. Los ejemplos de celdas binarias son circuitos flip-flop electrónicos, núcleos de ferrita usados en memorias y posiciones perforadas con un agujero o no perforadas en una tarjeta.

Registros

Un *registro* es un grupo de celdas binarias. Ya que una celda almacena un bit de información, se deduce que un registro con n celdas puede almacenar cualquier cantidad discreta de información que contenga n bits. El *estado* de un registro es un número múltiplo n de 1 y 0, con cada bit designando el estado en una celda en el registro. El *contenido* de un registro es una función de la interpretación dada a la información que está almacenada en él. Considérese, por ejemplo, el siguiente registro de 16 celdas:

1	1	0	0	0	0	1	1	1	1	0	0	1	0	0	1
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

En forma física, puede pensarse que el registro está compuesto de 16 celdas binarias, con cada celda almacenando ya sea un 1 o un 0. Supóngase que la configuración de bits almacenada en el registro es como se muestra. El estado de registro es el número múltiplo 16, 1100001111001001. Claramente, un registro con n celdas puede estar en uno de 2^n estados posibles. Ahora bien, si se supone que el contenido del registro representa un entero binario, entonces es obvio que el registro puede almacenar cualquier número binario desde 0 a $2^{16} - 1$. Para el ejemplo particular que se expuso, el contenido del registro es el equivalente binario del número decimal 50 121. Si se supone que el registro almacena caracteres alfanuméricos en un código de ocho bits, el contenido del registro es cualquiera de dos caracteres significativos (combinaciones de bits no asignadas no representan información significativa). En el código EBCDIC, el ejemplo anterior representa los dos caracteres C (los ocho bits de la izquierda) e I (los ocho bits de la derecha). Por otra parte, si se interpreta que el contenido del registro es cuatro dígitos decimales representados por un código de cuatro bits, el contenido del registro es un número decimal de cuatro dígitos. En el código exceso-3, el ejemplo anterior es el número decimal 9 096. El contenido de registro no tiene significado en el BCD, ya que la combinación de bits 1100 no está asignada a cualquier dígito decimal. Por este ejemplo, es claro que un registro puede almacenar uno o más elementos discretos de información y que la misma configuración de bits puede interpretarse en forma diferente para distintos tipos de elementos de información. Es importante que el usuario almacene información con significado en los registros y que la computadora esté programada para procesar esta información de acuerdo con el *tipo* de información almacenada.

Transferencia de registro

Una computadora digital se caracteriza por sus registros. La unidad de memoria (Fig. 1-1) simplemente es una colección de miles de registros para almacenar información digital. La unidad de proceso está compuesta de diversos registros que almacenan operandos bajo los cuales se realizan operaciones. La unidad de control utiliza registros para mantener cuenta de diversas secuencias de computación y, cada dispositivo de entrada o salida debe tener cuando menos un registro para almacenar la información transferida al dispositivo o desde éste. Una operación de *transferencia interregistro*, una operación básica en sistemas digitales, consiste en la transferencia de información almacenada en un registro a otro. En la Fig. 1-2 se ilustra la transferencia de información entre registros y se demuestra en forma gráfica la transferencia de información binaria de un teclado de teletipo a un registro en la unidad de memoria. Se supone que la unidad de teletipo de entrada tiene un teclado, un circuito de control y registros de entrada. Cada vez que se oprime una tecla, el control introduce al registro de entrada un caracter de código alfanumérico de ocho bits. Se supone que el código usado es el código ASCII con un octavo bit de paridad impar. La información del registro de entrada se transfiere en las ocho celdas menos significativas de un registro

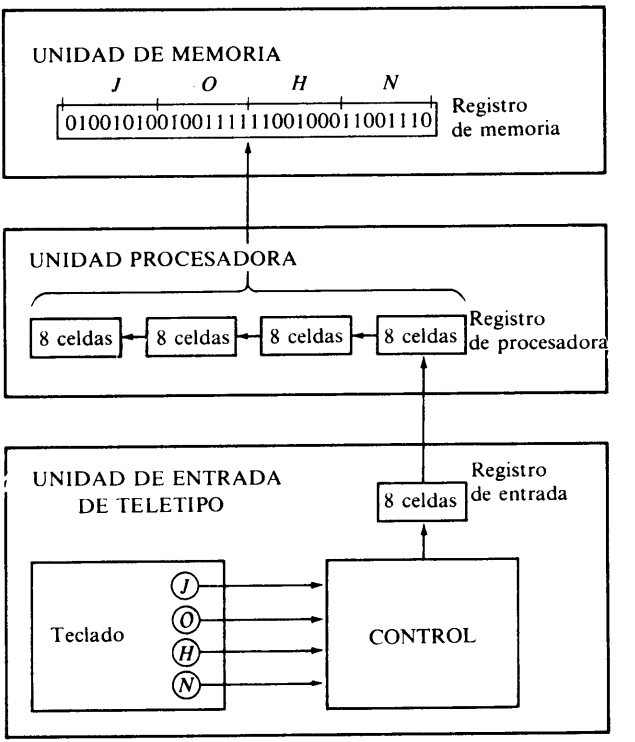


Figura 1-2 Transferencia de información con registros

de procesador. Después de cada transferencia, el registro de entrada se limpia para posibilitar que el control inserte un nuevo código de ocho bits cuando se oprimen otra vez las teclas. Cada caracter de ocho bits transferidos al registro del procesador está precedido por un corrimiento del caracter anterior a las siguientes ocho celdas a su izquierda. Cuando se completa una transferencia de cuatro caracteres, el registro del procesador está lleno y su contenido se transfiere a un registro de memoria. El contenido almacenado en el registro de memoria que se muestra en la Fig. 1-2 proviene de la transferencia de los caracteres JOHN después de que se han oprimido las cuatro teclas apropiadas.

Para procesar cantidades discretas de información en la forma binaria, una computadora debe estar provista con (1) dispositivos que retengan los datos que van a procesar y (2) elementos de circuito que manipulen los bits individuales de información. El dispositivo de uso más común para retener los datos es un registro. La manipulación de variables binarias se hace mediante circuitos lógicos digitales. En la Fig. 1-3 se ilustra el proceso de sumar dos números binarios de 10 bits. La unidad de memoria, que en forma normal consta de miles de registros, se muestra en el diagrama

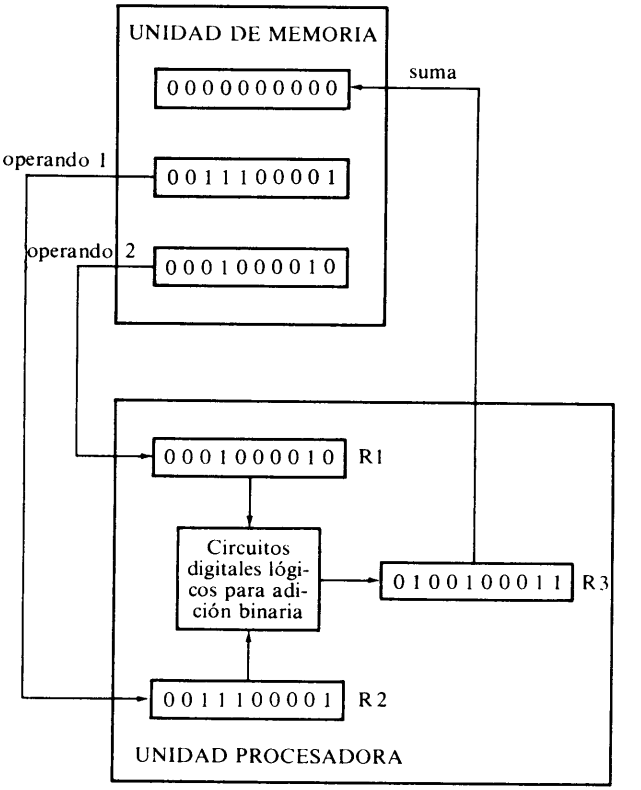


Figura 1-3 Ejemplo de procesamiento de información binaria.

con sólo tres de sus registros. La parte de la unidad de proceso que se presenta consta de tres registros, R1, R2 y R3, junto con los circuitos digitales que manipulan los bits de R1 y R2 y transfieren a R3 un número binario igual a su suma aritmética. Los registros de memoria almacenan información y son incapaces de procesar los dos operandos. Sin embargo, la información almacenada en memoria puede transferirse a los registros del procesador. Los resultados obtenidos en los registros del procesador pueden transferirse devolviéndolos a un registro de memoria para almacenamiento, hasta que se necesiten otra vez. El diagrama muestra los contenidos de los dos operandos transferidos de los dos registros de memoria a R1 y R2. Los circuitos lógicos digitales producen la suma, que se transfiere al registro R3. Los contenidos de R3 pueden transferirse ahora devolviéndolos a uno de los registros de memoria.

Los últimos dos ejemplos demuestran las capacidades de flujo de información de un sistema digital en una forma muy simple. Los registros del sistema son los elementos básicos para almacenar y retener la información binaria. Los circuitos lógicos digitales procesan la información. Los circuitos lógicos digitales y sus capacidades de manipulación se introducen en la sección siguiente. Los registros y la memoria se presentan en el Capítulo 7.

1-8 LOGICA BINARIA

La lógica binaria trata con variables que toman dos valores discretos y con operaciones que tienen significado lógico. Los dos valores que toman las variables pueden designarse con nombres diferentes (esto es, *verdadero* y *falso*, *si* y *no*, etc.), pero para este propósito no es conveniente pensar en términos de bits y asignarles los valores de 1 y 0. La lógica binaria se usa para describir, en forma matemática, la manipulación y el proceso de la información binaria. Es en particular adecuada para el análisis y diseño de sistemas digitales. Por ejemplo, los circuitos lógicos digitales en la Fig. 1-3 que llevan a cabo la aritmética binaria, son circuitos cuyo comportamiento se expresa en la forma más conveniente mediante variables binarias y operaciones lógicas. La lógica binaria se introduce en esta sección y es equivalente a un álgebra llamada booleana. La presentación formal de un álgebra booleana de dos valores se cubre con más detalle en el Capítulo 2. El propósito de esta sección es introducir el álgebra booleana en una forma heurística y relacionarla con los circuitos lógicos digitales y las señales binarias.

Definición de la lógica binaria

La lógica binaria consta de variables binarias y operaciones lógicas. Las variables se denotan con letras del alfabeto como A, B, C, x, y, z, etc., y cada variable tiene dos y sólo dos valores posibles distintos: 1 y 0. Hay tres operaciones lógicas básicas: AND, OR y NOT.

1. AND (Y): esta operación se representa mediante un punto o por la ausencia de operador. Por ejemplo, $x \cdot y = z$ o $xy = z$ se lee "x Y y es igual a z". La operación lógica AND se interpreta con el significado $z = 1$ si y sólo si $x = 1$ y

$y = 1$; en cualquier otro caso $z = 0$. (Recuérdese que x, y y z son variables binarias y pueden ser iguales a 1 o 0 y a nada más.)

2. OR (O): Esta operación se representa mediante un signo de suma. Por ejemplo, $x + y = z$ se lee "x O y es igual a z", lo cual significa que $z = 1$ si $x = 1$ *o* si $y = 1$ *o* si tanto $x = 1$ como $y = 1$. Si tanto $x = 0$ como $y = 0$, entonces $z = 0$.

3. NOT (NO): Esta operación está representada por una sola comilla (algunas veces por una barra). Por ejemplo, $x' = z$ (o $\bar{x} = z$) se lee "x no es igual a z", significa que z es lo que x no es. En otras palabras, si $x = 1$, entonces $z = 0$; pero si $x = 0$, entonces $z = 1$.

La lógica binaria es semejante a la aritmética binaria y, las operaciones AND y OR tienen ciertas similitudes con la multiplicación y la suma, respectivamente. De hecho, los símbolos que se utilizan para AND y OR son los mismos que se usan para la multiplicación y la suma. Sin embargo, la lógica binaria no debe confundirse con la aritmética binaria. Debe tomarse en cuenta que una variable aritmética denota un número que puede constar de muchos dígitos. Una variable lógica es siempre ya sea 1 o 0. Por ejemplo, en la aritmética binaria se tiene $1 + 1 = 10$ (se lee "uno más uno es igual a 2"), en tanto que en la lógica binaria se tiene $1 + 1 = 1$ (se lee: "uno OR uno es igual a uno").

Para cada combinación de los valores de x y y, hay un valor de z especificado por la definición de la operación lógica. Estas definiciones pueden listarse en forma compacta usando las *tablas de verdad.* Una tabla de verdad es una tabla de todas las combinaciones posibles de las variables, que muestra la relación entre los valores que pueden tomar las variables y el resultado de la operación. Por ejemplo, las tablas de verdad para las operaciones AND y OR con las variables x y y se obtienen haciendo la lista de todos los valores posibles que pueden tener las variables cuando se combinan en pares. El resultado de la operación para cada combinación se lista entonces en una columna separada. Las tablas de verdad para AND, OR y NOT se listan en la Tabla 1-6. Estas tablas demuestran en forma clara las definiciones de las operaciones.

Circuitos con interruptores y señales binarias

El uso de las variables binarias y la aplicación de la lógica binaria se demuestra por los circuitos con interruptores simples en la Fig. 1-4. Permítase que los interruptores manuales A y B representan dos variables binarias cuyos valores son iguales a 0 cuando el interruptor está abierto y 1 cuando el interruptor está cerrado. En forma semejante, permítase que la lámpara L represente una tercera variable binaria igual a 1 cuando la luz esté encendida y 0 cuando esté apagada. Para los interruptores en serie, la luz se enciende y A y B están cerrados. Para los interruptores en paralelo, la luz se enciende si A o B está cerrado. Es obvio que los dos circuitos pueden expresarse mediante la lógica binaria con las operaciones Y y O, respectivamente:

$L = A \cdot B$ para el circuito en la Fig. 1-4(a)

$L = A + B$ para el circuito en la Fig. 1-4(b)

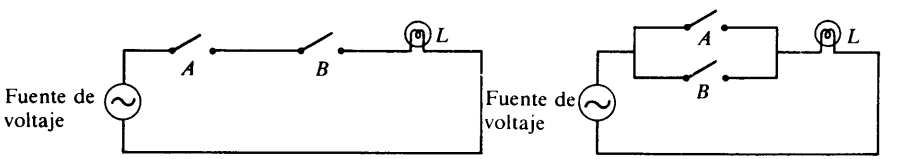
TABLA 1-6 Tablas de verdad de operaciones lógicas

AND			OR			NOT	
x	y	$x \cdot y$	x	y	$x + y$	x	x'
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

Algunas veces los circuitos digitales electrónicos se denominan *circuitos interruptores* (o circuitos conmutadores) debido a que se comportan como un interruptor, con el elemento activo, por ejemplo un transistor que conduce (interruptor cerrado) o bien no conduce (interruptor abierto). En lugar de cambiar manualmente el interruptor, un circuito interruptor electrónico usa señales binarias para controlar los estados de conducción o no conducción del elemento activo. Las señales eléctricas como voltajes o corriente existen a través de un sistema digital ya sea en uno de dos valores reconocibles (excepto durante la transición). Los circuitos operados por voltaje, por ejemplo, responden a dos niveles separados de voltaje que representan una variable binaria igual a lógica 1 o lógica 0. Por ejemplo, un sistema digital particular puede definir la lógica 1 como una señal con un valor nominal de 3 volts y, lógica 0 como una señal con un valor nominal de 0 volt. Como se muestra en la Fig. 1-5, cada nivel de voltaje tiene una desviación aceptable de la nominal. La región intermedia entre las regiones permitidas se cruza sólo durante las transiciones de estado. Las terminales de entrada de los circuitos digitales aceptan señales binarias con las tolerancias permitidas y responden a la terminal de salida con señales binarias que caen dentro de las tolerancias especificadas.

Compuertas lógicas

Los circuitos digitales electrónicos también se denominan *circuitos lógicos* ya que, con la entrada apropiada, establecen trayectorias lógicas de manipulación. Cualquier información que se desee para computación o control puede operarse por el paso de señales binarias a través de diversas combinaciones de circuitos lógicos, cada señal



(a) Interruptores en serie — lógica AND (b) Interruptores en paralelo — lógica OR

Figura 1-4 Circuitos interruptores que demuestran la lógica binaria.

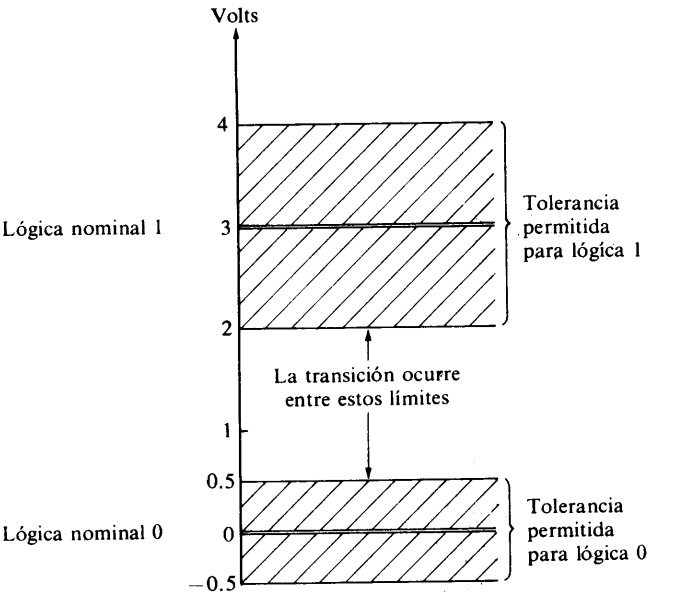
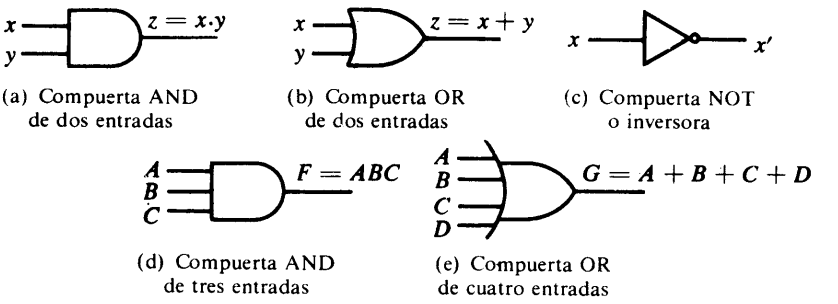


Figura 1-5 Ejemplo de señales binarias.

representa una variable y lleva un bit de información. Los circuitos lógicos que realizan las operaciones lógicas de AND, OR y NOT se muestran con sus símbolos en la Fig. 1-6. Estos circuitos, llamados *compuertas*, son bloques de hardware que producen una señal de salida lógica 1 o lógica 0 y se satisfacen los requisitos de la entrada lógica. Obsérvese que se han utilizado cuatro nombres diferentes para el mismo tipo de circuitos: circuitos digitales, circuitos interruptores, circuitos lógicos y compuertas. Todos los cuatro nombres tienen uso amplio, pero aquí se hará referencia a los circuitos como compuertas AND, OR y NOT. Algunas veces la compuerta NOT se denomina *circuito inversor* ya que invierte una señal binaria.

Las señales de entrada x y y en las dos compuertas de entrada en la Fig. 1-6 pueden existir en uno de cuatro estados posibles: 00, 10, 11 o 01. Estas señales de entrada se muestran en la Fig. 1-7, junto con las señales de salida para las compuertas AND y OR. Los diagramas de tiempo en la Fig. 1-7 ilustran la respuesta de cada circuito a cada una de las cuatro combinaciones binarias de entrada posibles. La razón del nombre "inversor" para la compuerta NOT es aparente por la comparación de la señal x (entrada del inversor) y la de x' (salida del inversor).

Las compuertas AND y OR pueden tener más de dos entradas. Una compuerta AND con tres entradas y una compuerta OR con cuatro entradas se muestran en la Fig. 1-6. La compuerta de tres entradas AND responde con una salida lógica 1 si todas las tres señales de entrada son de lógica 1. La salida produce una señal de lógica 0 si cualquier entrada es lógica 0. Las cuatro entradas en la compuerta OR responden con una lógica 1 cuando cualquier entrada es lógica 1. Su salida llega a ser lógica 0 si todas las señales de entrada son lógica 0.



(a) Compuerta AND de dos entradas

(b) Compuerta OR de dos entradas

(c) Compuerta NOT o inversora

(d) Compuerta AND de tres entradas

(e) Compuerta OR de cuatro entradas

Figura 1-6 Símbolos para los circuitos digitales lógicos.

El sistema matemático de la lógica binaria es mejor conocido como álgebra booleana. Esta álgebra se usa en forma conveniente para describir la operación de redes complejas de circuitos digitales. Los diseñadores de sistemas digitales utilizan el álgebra booleana para transformar los diagramas de circuitos en expresiones algebraicas y viceversa. Los Capítulos 2 y 3 se dedican al estudio del álgebra booleana, sus propiedades y capacidades de manipulación. En el capítulo 4 se muestra cómo puede usarse el álgebra booleana para expresar en forma matemática las interconexiones entre redes de compuertas.

1-9 CIRCUITOS INTEGRADOS

Los circuitos digitales en forma invariable se construyen con circuitos integrados. Un circuito integrado (abreviado IC) es un cristal semiconductor pequeño de silicio, llamado *pastilla*, que contiene componentes eléctricos como transistores, diodos, resistores y capacitores. Los diversos componentes están interconectados dentro de la pastilla para formar un circuito electrónico. La pastilla se monta en un paquete de metal o plástico y se soldan conexiones a las clavijas externas para formar el IC. Los circuitos integrados difieren de otros circuitos electrónicos compuestos de componentes desprendibles en que los componentes individuales de un IC no pueden separarse o

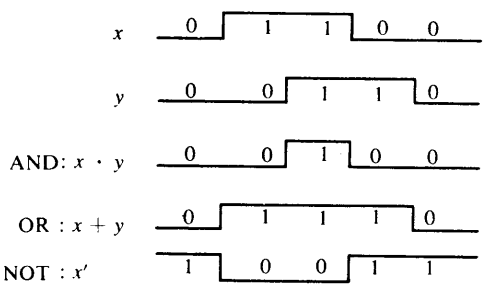


Figura 1-7 Señales de entrada-salida para las compuertas (a), (b) y (c) en la Fig. 106.

desconectarse y el circuito en el interior del paquete es accesible sólo a través de las clavijas externas.

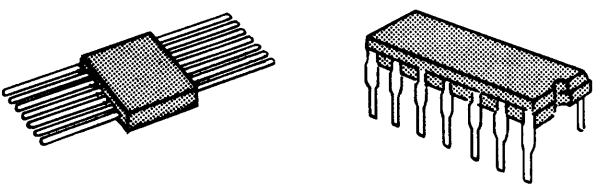
Los circuitos integrados se obtienen en dos tipos de paquetes: el paquete *plano* y el paquete *dual en línea* (DIP) como se muestra en la Fig. 1-8. El paquete dual en línea es el tipo de mayor uso debido a su precio bajo y fácil instalación en tableros para conexionar circuitos. La envolvente del paquete IC se hace de plástico o cerámica. La mayoría de los paquetes tienen tamaño estándar y el número de clavijas varía desde 8 a 64. Cada IC tiene una denominación numérica impresa en la superficie del paquete para su identificación. Cada vendedor publica un libro o catálogo con información que proporciona los datos necesarios que conciernen a los diversos productos.

El tamaño de los paquetes IC es muy pequeño. Por ejemplo, cuatro compuertas AND están encerradas dentro de un paquete dual en línea de 14 clavijas con dimensiones de $20 \times 8 \times 3$ milímetros. Un microprocesador entero se encuentra dentro de un paquete dual en línea de 40 clavijas con dimensiones de $50 \times 15 \times 4$ milímetros.

Aparte de una reducción sustancial en tamaño, los IC ofrecen otras ventajas y beneficios en comparación con los circuitos electrónicos hechos de componentes discretos. El costo de los IC es muy bajo, lo que los hace económicos para su utilización. Su consumo reducido de potencia hace que el sistema digital tenga una operación más económica. Tienen una alta confiabilidad contra fallas, de modo que el sistema digital necesita menos reparaciones. La velocidad de operación es más alta, lo cual los hace adecuados para operaciones de alta velocidad. El uso de los IC reduce el número de conexiones de alambrado externas, debido a que muchas de las conexiones están en el interior del paquete. Debido a todas estas ventajas, los sistemas digitales siempre se construyen con circuitos integrados.

Los circuitos integrados se clasifican en dos categorías generales, *lineales* y *digitales*. Los IC lineales operan con señales continuas para proporcionar funciones electrónicas como amplificadores y comparadores de voltaje. Los circuitos integrados digitales operan con señales binarias y están hechos de compuertas digitales interconectadas. Aquí el interés se centra sólo en los circuitos integrados digitales.

Conforme ha mejorado la tecnología de los IC, el número de compuertas que pueden colocarse dentro de una sola pastilla de silicio ha aumentado en forma considerable. La diferenciación entre los IC que tienen unas cuantas compuertas internas y los que tienen decenas o cientos de compuertas, se hace por una referencia acostumbrada de que un paquete es un dispositivo de pequeña, mediana o gran escala



Paquete plano Paquete dual en línea

Figura 1-8 Paquetes de circuitos integrados.

de integración. Varias compuertas lógicas en un solo paquete hacen un dispositivo con integración a pequeña escala (SSI). Para calificar como un dispositivo de integración a media escala (MSI), el IC debe realizar una función lógica completa y tener una complejidad de 10 a 100 compuertas. Un dispositivo de integración a gran escala (LSI) lleva a cabo una función lógica con más de 100 compuertas. También hay dispositivos de integración a muy alta escala (VLSI) que contienen miles de compuertas en una sola pastilla.

Muchos de los diagramas de circuitos digitales que se consideran en este libro se muestran en detalle hasta las compuertas individuales y sus conexiones. Dichos diagramas son útiles para demostrar la construcción lógica de una función particular. Sin embargo, debe tenerse en cuenta que, en la práctica, la función puede obtenerse por un dispositivo MSI o LSI, y el usuario tiene acceso a las entradas y salidas externas pero no a las entradas y salidas de las compuertas intermedias. Por ejemplo, un diseñador que desea incorporar un registro en su sistema es más probable que escoja una función de esta clase de un circuito MSI disponible, en lugar de diseñarlo con circuitos digitales individuales como puede mostrarse en un diagrama.

BIBLIOGRAFIA

1. Richard, R. K., *Arithmetic Operations in Digital Computers*. New York: Van Nostrand Co., 1955.
2. Flores, I., *The Logic of Computer Arithmetic*. Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1963.
3. Chu, Y., *Digital Computer Design Fundamentals*. New York: McGraw-Hill Book Co., 1962, Caps. 1 y 2.
4. Kostopoulos, G. K., *Digital Engineering*. New York: John Wiley & Sons, Inc., 1975, Cap. 1.
5. Rhyne, V. T., *Fundamentals of Digital Systems Design*. Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1973, Cap. 1.

PROBLEMAS

1-1. Escriba los primeros 20 dígitos decimales en base 3.

1-2. Sume y multiplique los siguientes números en la base dada sin convertirlos en decimales.
(a) $(1230)_4$ y $(23)_4$
(b) $(135.4)_6$ y $(43.2)_6$
(c) $(367)_8$ y $(715)_8$
(d) $(296)_{12}$ y $(57)_{12}$

1-3. Convierta el número decimal 250.5 en base 3, base 4, base 7, base 8 y base 16.

1-4. Convierta los siguientes números decimales en binarios: 12.0625, 10^4, 673.23 y 1998.

1-5. Convierta los siguientes números binarios en decimales:
10.10001, 101110.0101, 1110101.110, 1101101.111.

1-6. Convierta los siguientes números de la base dada en las bases indicadas:
(a) decimal 225.225 en binario, octal y hexadecimal

(b) binario 11010111.110 en decimal, octal y hexadecimal
(c) octal 623.77 en decimal, binario y hexadecimal
(d) hexadecimal 2AC5.D en decimal, octal y binario

1-7. Convierta los siguientes números en decimales:
(a) $(1001001.011)_2$
(b) $(12121)_3$
(c) $(1032.2)_4$
(d) $(4310)_5$
(e) $(0.342)_6$
(f) $(50)_7$
(g) $(8.3)_9$
(h) $(198)_{12}$

1-8. Obtenga los complementos de 1 y de 2 de los siguientes números binarios: 1010101, 0111000, 0000001, 10000, 00000.

1-9. Obtenga los complementos de 9 y de 10 de los siguientes números decimales: 13579, 09900, 90090, 10000, 00000.

1-10. Encuentre el complemento de 10 de $(935)_{11}$.

1-11. Lleve a cabo la resta con los siguientes números decimales usando (1) el complemento de 10 y (2) el complemento de 9. Verifique la respuesta por resta directa.
(a) 5250 − 321
(b) 3570 − 2100
(c) 753 − 864
(d) 20 − 1000

1-12. Haga la resta con los siguientes números binarios usando (1) el complemento de 2 y (2) el complemento de 1. Verifique la respuesta por resta directa.
(a) 11010 − 1101
(b) 11010 − 10000
(c) 10010 − 10011
(d) 100 − 110000

1-13. Demuestre el procedimiento establecido en la Sección 1-5 para la sustracción de dos números con complemento de $(r - 1)$.

1-14. Para los códigos pesados (a) 3, 3, 2, 1 y (b) 4, 4, 3, −2 para dígitos decimales determine todas las tablas posibles, de modo que el complemento de 9 de cada dígito decimal se obtenga por el cambio de 1 a 0 y de 0 a 1.

1-15. Represente el número decimal 8620 (a) en BCD, (b) en el código exceso-3, (c) en el código 2, 4, 2, 1 y (d), como un número binario.

1-16. Un código binario usa diez bits para representar cada uno de los diez dígitos decimales. Cada dígito está asignado a un código de nueve números 0 y un 1. El código para el dígito 6, por ejemplo, es 0001000000. Determine el código binario para los dígitos decimales restantes.

1-17. Obtenga el código pesado binario para los dígitos en base 12 usando pesos de 5421.

1-18. Determine el bit de paridad-impar generado cuando el mensaje consta de diez dígitos decimales en el código 8, 4, −2, −1.

1-19. Determine otras dos combinaciones para un código reflejado diferente al que se muestra en la Tabla 1-4.

1-20. Obtenga un código binario para representar todos los dígitos base 6 de modo que el complemento de 5 se obtenga por el reemplazo de 1 por 0 y 0 por 1 en los bits del código.

1-21. Asigne un código binario en cierta manera ordenada de los 52 naipes de la baraja. Utilice el número mínimo de bits.

1-22. Escriba su primer nombre, la inicial del segundo y su apellido paterno en un código de ocho bits hecho con los siete bits ASCII de la Tabla 1-5 y un bit de paridad par en la

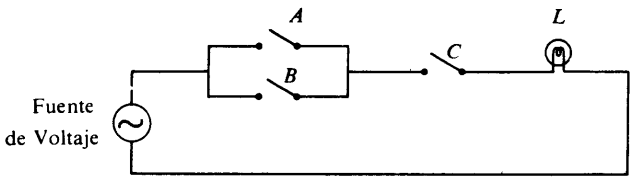
posición más significativa. Incluya espacios en blanco entre nombres y un punto después de la inicial del segundo nombre.

1-23. Muestre la configuración de bits de un registro de 24 celdas cuando su contenido representa (a) el número $(295)_{10}$ en binario, (b) el número decimal 295 en BCD y (c), los caracteres XY5 en EBCDIC.

1-24. El estado de un registro de 12 celdas es 010110010111. ¿Cuál es su contenido si representa (a) tres dígitos decimales en BCD, (b) tres dígitos decimales en el código exceso-3, (c) tres dígitos decimales en el código 2, 4, 2, 1 y (d), dos caracteres en el código interno de la Tabla 1-5?

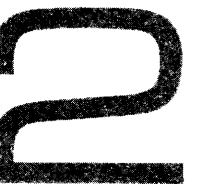
1-25. Muestre el contenido de todos los registros en la Fig. 1-3 si los dos números binarios que se agregan tienen el equivalente decimal de 257 y 1050. (Suponga que hay registros con 11 celdas.)

1-26. Exprese el siguiente circuito de interruptores en notación lógica binaria.



1-27. Muestre las señales (mediante un diagrama similar al de la Fig. 1-7) de las salidas F y G en la Fig. 1-6. Utilice señales binarias arbitrarias para las entradas A, B, C y D.

Algebra booleana y compuertas lógicas



2-1 DEFINICIONES BASICAS

El álgebra booleana, como cualquier otro sistema matemático deductivo, puede definirse con un conjunto de elementos, un conjunto de operadores y un número de axiomas no probados o postulados. Un *conjunto* de elementos es cualquier colección de objetos que tienen una propiedad común. Si S es un conjunto y, x y y son ciertos objetos, entonces $x \in S$ denota que x es un miembro del conjunto S y, $y \notin S$ denota que y no es un elemento de S. Un conjunto con un número denumerable de elementos se especifica por llaves: $A = \{1, 2, 3, 4\}$, esto es, los elementos del conjunto A son los números 1, 2, 3 y 4. Un *operador binario* definido en un conjunto S de elementos es una regla que asigna a cada par de elementos de S un elemento único de S. Como ejemplo, considérese la relación $a*b = c$. Se dice que $*$ es un operador binario y especifica una regla para encontrar c mediante el par (a, b) y también si $a, b, c \in S$. Sin embargo, $*$ no es un operador binario si $a, b \in S$, si la regla encuentra que $c \notin S$.

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

1. *Cierre.* Un conjunto S está cerrado con respecto a un operador binario si, para cada par de elementos de S, el operador binario especifica una regla para obtener un elemento único de S. Por ejemplo, el conjunto de los números naturales $N = \{1, 2, 3, 4, \ldots\}$ está cerrado con respecto al operador binario más (+) por las reglas de la adición aritmética, ya que para cualquier $a, b \in N$ se obtiene una única $c \in N$ por la operación $a + b = c$. El conjunto de los números naturales no está cerrado con respecto al operador binario menos (—) por las reglas de la resta aritmética debido a que $2 - 3 = -1$ y $2, 3 \in N$, ya que $(-1) \notin N$.
2. *Ley asociativa.* Un operador binario $*$ en un conjunto S se dice que es asociativo siempre que

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

3. *Ley conmutativa.* Un operador binario $*$ en un conjunto S se dice que es conmutativo siempre que:

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

4. *Elemento identidad.* Un conjunto S se dice que tiene un elemento identidad respecto a una operación binaria $*$ en S si existe un elemento $e \in S$ con la propiedad:

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

Ejemplo: El elemento 0 es un elemento identidad con respecto a la operación + en el conjunto de enteros $I = \{ ..., -3, -2, -1, 0, 1, 2, 3, ... \}$ ya que:

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

El conjunto de los números naturales N no tiene elemento identidad ya que 0 está excluido del conjunto.

5. *Inversa.* Un conjunto S que tiene el elemento identidad e con respecto a un operador binario $*$ se dice que tiene una inversa siempre que, para cada $x \in S$, existe un elemento $y \in S$ tal que:

$$x*y = e$$

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

6. *Ley distributiva.* Si $*$ y $\cdot$ son dos operadores binarios en un conjunto S, $*$ se dice que es distributivo sobre $\cdot$ siempre que:

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

Un ejemplo de una estructura algebraica es un *campo.* Un *campo* es un conjunto de elementos, junto con dos operadores binarios, cada uno teniendo las propiedades 1 a 5 y ambos operadores combinados para dar la propiedad 6. El conjunto de los números reales junto con los operadores binarios + y $\cdot$ forman el campo de los números reales. El campo de los números reales es la base de la aritmética y del álgebra ordinaria. Los operadores y los postulados tienen los siguientes significados:

El operador binario + define la adición.

La identidad aditiva es 0.

La inversa aditiva define la sustracción.

El operador binario · define la multiplicación.

La identidad multiplicativa es 1.

La inversa multiplicativa de $a = 1/a$ define la división, esto es, $a \quad 1/a = 1$.

La única ley distributiva aplicable es la de · sobre +:

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

2-2 DEFINICION AXIOMATICA DEL ALGEBRA BOOLEANA

En 1854 George Boole (1) introdujo un tratamiento sistemático de la lógica y desarrolló para este propósito un sistema algebraico que ahora se conoce como álgebra booleana. En 1938 C.E. Shannon (2) introdujo un álgebra booleana de dos valores denominada *álgebra de interruptores*, en la cual demostró que las propiedades de los circuitos eléctricos y estables con interruptores pueden representarse con esta álgebra. Para la definición formal del álgebra booleana, se emplean los postulados formulados por E.V. Huntington (3) en 1904. Estos postulados o axiomas no son únicos para definir el álgebra booleana. Se han usado otros conjuntos de postulados.* El álgebra booleana es una estructura algebraica definida en un conjunto de elementos B junto con dos operadores binarios + y · siempre que se satisfagan los siguientes postulados (Huntington):

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

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

1. Los postulados de Huntington no incluyen la ley asociativa. No obstante, esta ley es válida para el álgebra booleana y puede derivarse (para ambos operadores) mediante los otros postulados.

*Véase, por ejemplo, Birk off y Bartee (4), Capítulo 5.

2. La ley distributiva de + sobre ·, esto es, $x + (y \cdot z) = (x + y) \cdot (x + z)$, es válida para el álgebra booleana, pero no para el álgebra ordinaria.
3. El álgebra booleana no tiene inversas aditiva o multiplicativa; por lo tanto, no hay operaciones de sustracción o división.
4. El postulado 5 define un operador llamado *complemento* que no se encuentra en el álgebra ordinaria.
5. El álgebra ordinaria trata con números reales, los cuales constituyen un conjunto infinito de elementos. El álgebra booleana trata con el conjunto todavía no definido de elementos B, pero en el álgebra booleana de dos valores que se define más adelante (y de interés en el uso subsecuente de esta álgebra), B se define como un conjunto con sólo dos elementos, 0 y 1.

El álgebra booleana se parece en algunos aspectos al álgebra ordinaria. La elección de los símbolos + y · es intencional para facilitar las manipulaciones algebraicas booleanas por las personas que ya están familiarizadas con el álgebra ordinaria. Aunque puede utilizarse cierto conocimiento del álgebra ordinaria para tratar con el álgebra booleana, el principiante debe tener cuidado de no sustituir las reglas del álgebra ordinaria cuando no son aplicables.

Es importante distinguir entre los elementos del conjunto de una estructura algebraica y las variables de un sistema algebraico. Por ejemplo, los elementos de campos de los números reales son números, en tanto que variables como a, b, c, etc., que se usan en el álgebra ordinaria, son símbolos que representan números reales. En forma semejante, en el álgebra booleana se definen los elementos del conjunto B y variables como x, y, z son simplemente símbolos que representan los elementos. En este punto, es importante tener en cuenta que con objeto de tener un álgebra booleana. deben mostrarse:

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

Pueden formularse muchas álgebras booleanas, dependiendo de la elección de los elementos de B y las reglas de operación. * En el trabajo subsecuente, se tratará sólo con el álgebra booleana de dos valores, esto es, una con sólo dos elementos. El álgebra booleana de dos valores tiene aplicaciones en la teoría de conjuntos (el álgebra de clases) y en la lógica proposicional. El interés aquí es la aplicación del álgebra booleana a los circuitos tipo compuerta.

*Véase, por ejemplo, Hohn (6), Whitesitt (7) o Birkhoff y Bartee (4).

Algebra booleana de dos valores

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

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

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

x	x'
0	1
1	0

Estas reglas son exactamente las mismas que las operaciones AND, OR y NOT, respectivamente, definidas en la Tabla 1-6. Ahora debe mostrarse que los postulados de Huntington son válidos para el conjunto $B = \{0, 1\}$ y los dos operadores binarios que se definieron antes.

1. *Cierre* es obvio por las tablas, ya que el resultado de cada operación es, ya sea 1 o 0 y 1, $0 \in B$.

2. A partir de las tablas puede verse que:

   (a) $0 + 0 = 0 \qquad 0 + 1 = 1 + 0 = 1$
   (b) $1 \cdot 1 = 1 \qquad 1 \cdot 0 = 0 \cdot 1 = 0$

   establece que los dos *elementos identidad* son 0 para + y 1 para · como se define por el postulado 2.

3. Las leyes *conmutativas* son obvias por la simetría de las tablas del operador binario.

4. (a) La ley *distributiva* $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$ puede mostrarse que es verdadera por las tablas del operador, al formar una tabla de verdad de todos los valores posibles de x, y y z. Para cada combinación, se deriva $x \cdot (y + z)$ y se muestra que el valor es el mismo que $(x \cdot y) + (x \cdot z)$.

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

(b) La ley *distributiva* de + sobre · puede mostrarse que es válida mediante una tabla de verdad semejante a la anterior.

5. Mediante la tabla de complemento es fácil mostrar que:
   (a) $x + x' = 1$, ya que $0 + 0' = 0 + 1 = 1$ y $1 + 1' = 1 + 0 = 1$.
   (b) $x \cdot x' = 0$, ya que $0 \cdot 0' = 0 \cdot 1 = 0$ y $1 \cdot 1' = 1 \cdot 0 = 0$ lo cual verifica el postulado 5.
6. El postulado 6 se satisface ya que el álgebra booleana de dos valores tiene dos elementos distintos 1 y 0 con $1 \neq 0$.

Acaba de establecerse un álgebra booleana de dos valores que tiene un conjunto de dos elementos, 1 y 0, dos operadores binarios con reglas de operación equivalentes a las operaciones AND y OR y un operador complemento equivalente al operador NOT. En consecuencia, el álgebra booleana se ha definido de una manera matemáticamente formal y se ha mostrado que es equivalente a la lógica binaria que se presentó en forma heurística en la Sección 1-8. La presentación heurística es de ayuda para entender la aplicación del álgebra booleana a los circuitos tipo compuerta. La presentación formal es necesaria para desarrollar los teoremas y las propiedades del sistema algebraico. El álgebra booleana de dos valores que se define en esta sección también se conoce como "álgebra de interruptores" (o de conmutación) entre los ingenieros. Para dar énfasis a las similitudes entre el álgebra booleana de dos valores y otros sistemas binarios, esta álgebra se denominó "lógica binaria" en la Sección 1-8. De aquí en adelante, se eliminará el calificativo "dos valores" del álgebra booleana en las exposiciones subsecuentes.

2-3 TEOREMAS BASICOS Y PROPIEDADES DEL ALGEBRA BOOLEANA

Dualidad

Los postulados de Huntington se listaron en pares y se designaron en la parte (a) y la (b). Una parte puede obtenerse de la otra si los operadores binarios y los elementos identidad se intercambian. Esta propiedad importante del álgebra booleana se denomina *principio de dualidad.* Establece que cada expresión algebraica deducida de los postulados del álgebra booleana permanece válida si los operadores y los elementos identidad se intercambian. En una álgebra booleana de dos valores, los elementos identidad y los elementos del conjunto B son los mismos: 1 y 0. El principio de dualidad tiene muchas aplicaciones. Si se desea el dual de una expresión algebraica, simplemente se intercambian los operadores OR y AND y se reemplazan los 1 por 0 y los 0 por 1.

Teoremas básicos

En la Tabla 2-1 se listan seis teoremas del álgebra booleana y cuatro de sus postulados. La notación se simplifica omitiendo el · siempre que esto no provoque confusiones. Los teoremas y postulados que se listan son las relaciones más básicas en el álgebra

TABLA 2-1 Postulados y teoremas del álgebra booleana

Postulado 2	(a) $x + 0 = x$	(b) $x \cdot 1 = x$
Postulado 5	(a) $x + x' = 1$	(b) $x \cdot x' = 0$
Teorema 1	(a) $x + x = x$	(b) $x \cdot x = x$
Teorema 2	(a) $x + 1 = 1$	(b) $x \cdot 0 = 0$
Teorema 3, involución	$(x')' = x$	
Postulado 3, conmutativo	(a) $x + y = y + x$	(b) $xy = yx$
Teorema 4, asociativo	(a) $x + (y + z) = (x + y) + z$	(b) $x(yz) = (xy)z$
Postulado 4, distributivo	(a) $x(y + z) = xy + xz$	(b) $x + yz = (x + y)(x + z)$
Teorema 5, de De Morgan	(a) $(x + y)' = x'y'$	(b) $(xy)' = x' + y'$
Teorema 6, absorción	(a) $x + xy = x$	(b) $x(x + y) = x$

booleana. Se aconseja al lector que se familiarice con ellos tan pronto como le sea posible. Los teoremas, al igual que los postulados, se listan en pares; cada relación es el dual de su pareja. Los postulados son axiomas básicos de la estructura algebraica y no necesitan prueba. Los teoremas deben probarse mediante los postulados. Las pruebas de los teoremas con una variables se presentan más adelante. A la derecha se lista el número del postulado que justifica cada paso de la prueba.

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

$x + x = (x + x) \cdot 1$	por el postulado:	2(b)
$= (x + x)(x + x')$		5(a)
$= x + xx'$		4(b)
$= x + 0$		5(b)
$= x$		2(a)

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

$x \cdot x = xx + 0$	por el postulado:	2(a)
$= xx + xx'$		5(b)
$= x(x + x')$		4(a)
$= x \cdot 1$		5(a)
$= x$		2(b)

Obsérvese que el teorema 1(b) es el dual del teorema 1(a) y que cada paso de la prueba en la parte (b) es el dual de la parte (a). Cualquier teorema dual puede derivarse en forma similar de la prueba de su pareja correspondiente.

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

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

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

TEOREMA 3: $(x')' = x$. Por el postulado 5, se tiene $x + x' = 1$ y $x \cdot x' = 0$, lo cual define el complemento de x. El complemento de x' es x y también es $(x')'$. Por tanto, ya que el complemento es único, se tiene que $(x')' = x$.

Los teoremas que implican dos o tres variables pueden probarse en forma algebraica por los postulados y teoremas que ya se han probado. Por ejemplo, tómese el teorema de absorción.

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

$$
\begin{aligned}
x + xy &= x \cdot 1 + xy && \text{por el postulado 2(b)} \\
&= x(1 + y) && \text{por el postulado 4(a)} \\
&= x(y + 1) && \text{por el postulado 3(a)} \\
&= x \cdot 1 && \text{por el postulado 2(a)} \\
&= x && \text{por el postulado 2(b)}
\end{aligned}
$$

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

Puede demostrarse que los teoremas del álgebra booleana son válidos mediante las tablas de verdad. En estas tablas, ambos lados de la relación se verifican para que den resultados idénticos en todas las combinaciones posibles de las variables implicadas. La siguiente tabla de verdad verifica el primer teorema de absorción.

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

Las pruebas algebraicas de la ley asociativa y del teorema de De Morgan son largas y no se mostrarán aquí. Sin embargo, su validez se ilustra fácilmente con tablas de verdad. Por ejemplo, la tabla de verdad para el primer teorema de De Morgan $(x + y)' = x'y'$ se muestra a continuación.

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

Precedencia de los operadores

La precedencia de los operadores para evaluar las expresiones booleanas es (1) paréntesis, (2) NOT, (3) AND y (4) OR. En otras palabras, la expresión entre paréntesis debe evaluarse antes que las otras operaciones. La siguiente operación que toma precedencia es el complemento, entonces sigue AND y, por último, OR. Como ejemplo, considérese la tabla de verdad para el teorema de De Morgan. El lado izquierdo de la expresión es $(x + y)'$. Por consiguiente, la expresión entre paréntesis se evalúa primero y entonces se toma el complemento del resultado. El lado derecho de la expresión es $x'y'$. Así que, el complemento de x y el complemento de y se evalúan primero y el resultado se opera por AND. Obsérvese que en aritmética ordinaria es válida la misma precedencia (excepto para el complemento) cuando la multiplicación y la suma se reemplazan por AND y OR, respectivamente.

Diagrama de Venn

Una ilustración de ayuda que es posible utilizar para visualizar las relaciones entre las variables de una expresión booleana es el *diagrama de Venn*. Este diagrama consta de un rectángulo, como el que se muestra en la Fig. 2-1, dentro del cual se dibujan círculos traslapados, uno para cada variable. Cada círculo se etiqueta por una variable. Se designan todos los puntos dentro de un círculo como pertenecientes a la variable etiquetada y todos los puntos fuera del círculo como no pertenecientes a la variable. Tómese, por ejemplo, el círculo etiquetado x. Si se considera el interior del círculo, se dice que $x = 1$; si se considera el exterior, se dice que $x = 0$. Ahora, con dos círculos traslapados, hay cuatro áreas distintas dentro del rectángulo: el área que no pertenece ya sea a x o y $(x'y')$. El área dentro del círculo y pero fuera de x $(x'y)$, el área en el interior del círculo x pero fuera de y (xy') y el área dentro de ambos círculos (xy).

Los diagramas de Venn pueden usarse para ilustrar los postulados del álgebra booleana o para mostrar la validez de los teoremas. En la Fig. 2-2, por ejemplo, se ilustra que el área que pertenece a xy está en el interior del círculo x y, por lo tanto, $x + xy = x$. En la Fig. 2-3 se muestra la ley distributiva $x(y + z) = xy + xz$. En este diagrama se tienen tres círculos traslapados, uno para cada una de las variables x, y y z. Es posible distinguir ocho áreas distintas en un diagrama de Venn de tres variables. Para este ejemplo particular, la ley distributiva se demuestra observando que el área intersecada por el círculo x, con el área que encierra y o z, es la misma área que pertenece a xy o xz.

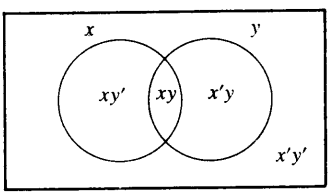


Figura 2-1 Diagrama de Venn para dos variables.

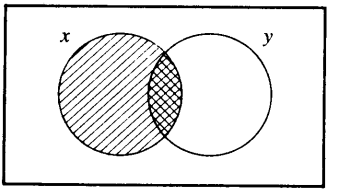
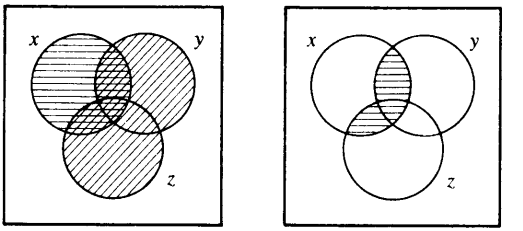


Figura 2-2 Ilustración en el diagrama de Venn de $x = xy + x$.



$x(y+z)$ $xy+xz$

Figura 2-3 Ilustración en el diagrama de Venn de la ley distributiva.

2-4 FUNCIONES BOOLEANAS

Una variable binaria puede tomar el valor de 0 o 1. Una función booleana es una expresión formada por variables binarias, los dos operadores binarios OR y AND, operador unitario NOT, paréntesis y signo de igual. Para un valor dado de variables, la función puede ser 0 o bien 1. Considérese, por ejemplo, la función booleana:

$$F_1 = xyz'$$

La función F_1 es igual a 1 si $x = 1$ y $y = 1$ y $z' = 1$; de otra manera, $F_1 = 0$. Este es un ejemplo de una función booleana representada como una función algebraica. Una función booleana también puede representarse en una tabla de verdad. Para representar una función en una tabla de verdad, se necesita una lista de las 2^n combinaciones de 1 y 0 de las n variables binarias y, una columna que muestre las combinaciones para las cuales la función es igual a 1 o 0. Como se muestra en la Tabla 2-2, hay ocho combinaciones distintas posibles para asignar bits a tres variables. La columna etiquetada F_1 contiene un 0 o bien un 1, para cada una de estas combinaciones. En la tabla se muestra que la función F_1 es igual a 1 sólo cuando $x = 1$, $y = 1$ y $z = 0$. De otra manera, es igual a 0. (Obsérvese que el enunciado $z' = 1$ es equivalente a decir que $z = 0$.) Considérese ahora la función:

$$F_2 = x + y'z$$

$F_2 = 1$ si $x = 1$ o si $y = 0$, mientras $z = 1$. En la Tabla 2-2, $x = 1$ en los últimos cuatro renglones y $yz = 01$ en los renglones 001 y 101. La última combinación se aplica también para $x = 1$. Por tanto, hay cinco combinaciones que hacen $F_2 = 1$. Como un tercer ejemplo, considérese la función:

$$F_3 = x'y'z + x'yz + xy'$$

Esto se muestra en la Tabla 2-2 con cuatro números 1 y cuatro números 0. F_4 es la misma que F_3 y se considera a continuación.

Cualquier función booleana puede representarse en una tabla de verdad. El número de renglones en la tabla es 2^n, donde n es el número de variables binarias en la función. Las combinaciones de 1 y 0 para cada renglón se obtienen fácilmente mediante los números binarios contando desde 0 a $2^n - 1$. Para cada renglón de la tabla, hay un valor para la función igual ya sea a 1 o 0. Surge ahora la pregunta, ¿es única una expresión algebraica de una función booleana dada? En otras palabras, ¿es posible encontrar dos expresiones algebraicas que especifiquen la misma función? La respuesta a esta pregunta es afirmativa. De hecho, la manipulación del álgebra booleana se aplica principalmente al problema de encontrar expresiones más simples para la misma función. Considérese, por ejemplo, la función:

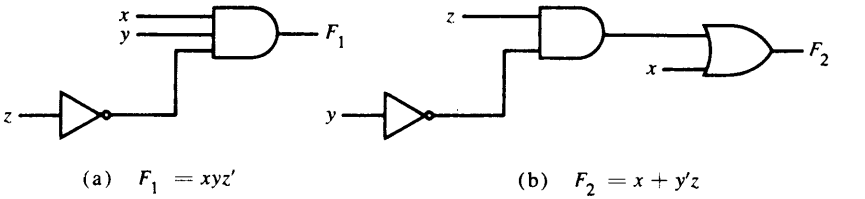
$$F_4 = xy' + x'z$$

Mediante la Tabla 2-2, se encuentra que F_4 es la misma que F_3, ya que ambas tienen 1 idénticos y 0 idénticos para cada combinación de valores de las tres variables binarias. En general, se dice que dos funciones de n variables binarias son iguales si tienen el mismo valor para todas las 2^n combinaciones posibles de las n variables.

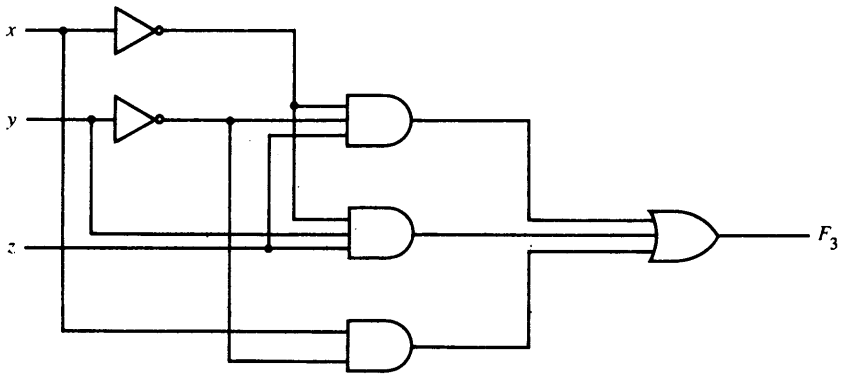
Una función booleana puede transformarse de una expresión algebraica en un diagrama lógico compuesto de compuertas AND, OR y NOT. El implante de las cuatro funciones que se introdujo en la exposición anterior se muestra en la Fig. 2-4.

TABLA 2-2 Tablas de verdad para $F_1 = xyz'$, $F_2 = x + y'z$, $F_3 = x'y'z + x'yz + xy'$, y $F_4 = xy' + x'z$

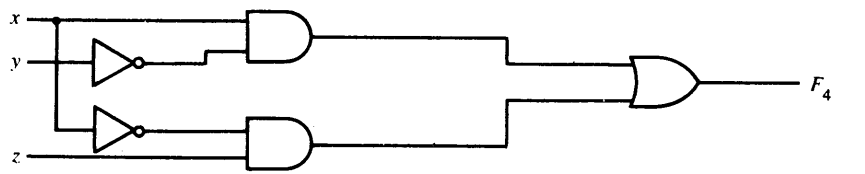
x	y	z	F_1	F_2	F_3	F_4
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	0



(a) $F_1 = xyz'$ (b) $F_2 = x + y'z$



(c) $F_3 = x'y'z + x'yz + xy'$



(d) $F_4 = xy' + x'z$

Figura 2-4 Implementación de funciones booleanas con compuertas.

El diagrama lógico incluye un circuito inversor para cada variable presente en su forma de complemento. (El inversor es innecesario si está disponible el complemento de la variable.) Hay una compuerta Y para cada término en la expresión y, se usa una compuerta O para combinar dos o más términos. Para los diagramas, es obvio que el implante de F_4 requiere menos compuertas y menos entradas que F_3. Ya que F_4 y F_3 son funciones booleanas iguales, es más económico implantar la forma F_4 que la F_3. Para encontrar circuitos más simples, debe conocerse cómo manipular las funciones booleanas para obtener expresiones iguales y más simples. Lo que constituye la mejor forma de una función booleana depende de la aplicación particular. En esta sección, se toma en consideración el criterio de minimización de equipo.

Manipulación algebraica

Una *literal* es una variable prima o no prima. Cuando una función booleana se implanta con compuertas lógicas, cada literal en la función denota una entrada a una compuerta, y cada término se implanta con una compuerta. La minimización del número de literales y el número de términos resulta en un circuito con menos equipo. No siempre es posible minimizar ambos en forma simultánea; por lo común, debe disponerse de más criterios. Por el momento, se reduce el criterio de minimización a la minimización de literales. Se expondrán otros criterios en el Capítulo 5. El número de literales en una función booleana puede minimizarse por manipulaciones algebraicas. Desafortunadamente, no hay reglas específicas que seguir que garanticen la respuesta final. El único método disponible es un procedimiento de corte y ensayo empleando los postulados, teoremas básicos y cualquier otro método de manipulación que llegue a ser familiar con el uso. Los siguientes ejemplos ilustran este procedimiento.

EJEMPLO 2-1: Simplifique la siguiente función booleana a un número mínimo de literales.

1. $x + x'y = (x + x')(x + y) = 1 \cdot (x + y) = x + y$
2. $x(x' + y) = xx' + xy = 0 + xy = xy$
3. $x'y'z + x'yz + xy' = x'z(y' + y) + xy' = x'z + xy'$
4. $$\begin{aligned} xy + x'z + yz &= xy + x'z + yz(x + x') \\ &= xy + x'z + xyz + x'yz \\ &= xy(1 + z) + x'z(1 + y) \\ &= xy + x'z \end{aligned}$$
5. $(x + y)(x' + z)(y + z) = (x + y)(x' + z)$ por la dualidad de la función 4.

Las funciones 1 y 2 son duales una de otra y utilizan expresiones duales en los pasos correspondientes. La función 3 muestra la igualdad de las funciones F_3 y F_4, expuestas con anterioridad. La cuarta ilustra el hecho de que un incremento en el número de literales algunas veces conduce a una expresión final más simple. La función 5 no se minimiza en forma directa, pero puede derivarse del dual de los pasos usados para derivar la función 4.

Complemento de una función

El complemento de una función F es F' y se obtiene por el intercambio de números 0 a números 1 y de números 1 a números 0 en el valor de F. El complemento de una función puede derivarse en forma algebraica mediante el teorema de De Morgan. Este par de teoremas se lista en la Tabla 2-1 para dos variables. Los teoremas de De Morgan pueden ampliarse a tres o más variables. La forma de tres variables del primer

teorema de De Morgan se deriva a continuación. Los postulados y los teoremas son los que se listan en la Tabla 2-1.

$$
\begin{aligned}
(A + B + C)' &= (A + X)' && \text{sea } B + C = X \\
&= A'X' && \text{por el teorema 5(a) (De Morgan)} \\
&= A' \cdot (B + C)' && \text{se sustituye } B + C = X \\
&= A' \cdot (B'C') && \text{por el teorema 5(a) (De Morgan)} \\
&= A'B'C' && \text{por el teorema 4(b) (asociativo)}
\end{aligned}
$$

Los teoremas de De Morgan para cualquier número de variables son semejantes en forma al caso de dos variables y pueden derivarse por sustituciones sucesivas en forma similar al método usado en la derivación anterior. Estos teoremas pueden generalizarse como sigue:

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

La forma generalizada del teorema de De Morgan enuncia que el complemento de una función se obtienen por el intercambio de los operadores AND y OR y complementando cada literal.

EJEMPLO 2-2: Encuentre el complemento de las funciones $F_1 = x'yz' + x'y'z$ y $F_2 = x(y'z' + yz)$. Se aplica el teorema de De Morgan cuantas veces sea necesario y se obtienen los complementos como sigue:

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

Un procedimiento más simple para derivar el complemento de una función es tomar la dual de la función y complementar cada literal. Este método se sigue del teorema generalizado de De Morgan. Recuérdese que la dual de una función se obtiene por el intercambio de los operadores AND y OR y los 1 y los 0.

EJEMPLO 2-3: Obtenga el complemento de las funciones F_1 y F_2 del Ejemplo 2-2, tomando sus duales y complementando cada literal.

1. $F_1 = x'yz' + x'y'z$.
   La dual de F_1 es $(x' + y + z')\,(x' + y' + z)$.
   Complemento de cada literal: $(x + y' + z)\,(x + y + z') = F'_1$.
2. $F_2 = x(y'z' + yz)$.
   La dual de F_2 es $x + (y' + z')\,(y + z)$.
   Complemento de cada literal: $x' + (y + z)(y' + z') = F'_2$.

2-5 FORMAS CANONICA Y ESTANDAR

Mintérminos y maxtérminos

Una variable binaria puede aparecer ya sea en forma normal (x) o en su forma complementaria (x'). Ahora considérense dos variables binarias x y y combinadas con un operador AND. Ya que cada variable puede aparecer en cualquier forma, hay cuatro combinaciones posibles: $x'y'$, $x'y$, xy' y xy. Cada uno de esos cuatro términos AND representa una de las áreas diferentes en el diagrama de Venn en la Fig. 2-1 y se denomina un *mintérmino* o un *producto estándar*. En forma semejante, pueden combinarse n variables para formar 2^n mintérminos. Los 2^n mintérminos diferentes pueden determinarse por un método similar al que se muestra en la Tabla 2-3 para tres variables. Los números binarios desde 0 a $2^n - 1$ se listan bajo las n variables. Cada mintérmino se obtiene de un término AND de las n variables, con cada variable vuelta prima si el bit correspondiente del número binario es un 0 y no prima si es un 1. En la tabla también se muestra un símbolo para cada mintérmino y está en la forma m_j, donde j indica el equivalente decimal del número binario del mintérmino denotado.

De manera semejante, n variables forman un término OR, con cada variable vuelta prima o no prima, proporcionando 2^n combinaciones posibles, denominadas *maxtérminos* o *sumas estándar*. Los ocho maxtérminos para tres variables, junto con su denotación simbólica, se listan en la Tabla 2-3. Cualesquiera 2^n maxtérminos para n variables pueden determinarse en forma similar. Cada maxtérmino se obtiene de un término OR de las n variables, con cada variable no prima si el bit correspondiente es 0 y prima si es un 1.* Obsérvese que cada maxtérmino es el complemento de su mintérmino correspondiente y viceversa.

*En algunos libros se define maxtérmino como un término OR de n variables, con cada variable sin prima si el bites un 1 y con prima si es un 0. La definición que se ha adoptado en este libro es preferible, ya que lleva a conversiones más simples entre funciones del tipo maxtérmino y mintérmino.

TABLA 2-3 Mintérminos y maxtérminos para tres variables binarias.

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

Una función booleana puede expresarse en forma algebraica mediante una tabla de verdad dada, formando un mintérmino para cada combinación de variables que produce un 1 en la función y, tomando entonces los OR de todos esos términos. Por ejemplo, la función F_1 en la Tabla 2-4 se determina al expresar las combinaciones 001, 100 y 111 como $x'y'z$, $xy'z'$ y xyz, respectivamente. Ya que cada uno de estos mintérminos resulta en $F_1 = 1$, se debe tener:

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

De manera semejante, puede verificarse con facilidad que:

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

Estos ejemplos demuestran una propiedad importante del álgebra booleana: Cualquier función booleana puede expresarse como una suma de mintérminos (por "suma" se entiende la aplicación del operador OR en los términos).

Ahora considérese el complemento de una función booleana. A partir de la tabla de verdad puede leerse al formar un mintérmino para cada combinación que produce un 0 en la función y aplicando el operador OR a esos términos. El complemento de f_1 se lee como:

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

Si se toma el complemento de f'_1, se obtiene la función f_1:

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

En forma similar, es posible leer la expresión para f_2 de la tabla:

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

TABLA 2-4 Función de tres variables

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

Estos ejemplos demuestran una segunda propiedad importante del álgebra booleana: Cualquier función booleana puede expresarse como un producto de maxtérminos (por "producto" se entiende que se aplica el operador AND a los términos). El procedimiento para obtener el producto de los maxtérminos en forma directa de la tabla de verdad es como sigue. Fórmese un maxtérmino para cada combinación de las variables que produce un 0 en la función, y entonces fórmese AND de todos los maxtérminos. Las funciones booleanas expresadas como una suma de mintérminos o producto de maxtérminos se dice que están en *forma canónica.*

Suma de mintérminos

Con anterioridad se enunció que para n variables binarias, pueden obtenerse 2^n mintérminos diferentes y, que cualquier función booleana puede expresarse como una suma de mintérminos. Los mintérminos cuya suma define la función booleana son los que dan los 1 de la función en una tabla de verdad. Ya que la función puede ser 1 o bien 0 para cada mintérmino, y puesto que hay 2^n mintérminos, pueden calcularse las funciones posibles que es factible formarse con n variables para hacer 2^{2^n}. Algunas veces es conveniente expresar la función booleana en la forma de su suma de mintérminos. Si no puede hacerse en esta forma, entonces puede realizarse primero por la expansión de la expresión en una suma de términos AND. Después cada término se inspecciona para ver si contiene todas las variables. Si se han perdido una o más variables, se aplica el operador AND con una expresión como $x + x'$, en donde x es una de las variables perdidas. El siguiente ejemplo aclara este procedimiento.

EJEMPLO 2-4: Exprese la función booleana $F = A + B'C$ en una suma de mintérminos. La función tiene tres variables A, B y C. El primer término A pierde dos variables; por tanto:

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

Todavía se pierde una variable:

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

El segundo término $B'C$ pierde una variable:

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

Combinando todos los términos, se tiene:

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

Pero $AB'C$ aparece dos veces y, de acuerdo con el teorema 1 ($x + x = x$). es posible quitar uno de ellos. Reordenando los mintérminos de manera ascendente, por último se obtiene:

$$F = A'B'C + AB'C' + AB'C + ABC' + ABC$$
$$= m_1 + m_4 + m_5 + m_6 + m_7$$

Algunas veces es conveniente expresar la función booleana, cuando está en su suma de mintérminos, en la siguiente notación abreviada:

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

El símbolo de suma Σ representa el operador OR que opera en los términos; los números siguientes son los mintérminos de la función. Las letras entre paréntesis que siguen a F forman una lista de las variables en el orden tomado cuando el mintérmino se convierte en un término AND.

Producto de los maxtérminos

Cada una de las funciones 2^{2^n} de n variables binarias también puede expresarse como un producto de maxtérminos. Para expresar la función booleana como un producto de maxtérminos, primero debe llevarse a una forma de términos OR. Es posible hacer esto por el uso de la ley distributiva $x + yz = (x + y)(x + z)$. Entonces, cualquier variable perdida x en cada término 0 se opera a OR con xx', Este procedimiento se aclara en el siguiente ejemplo.

EJEMPLO 2-5: Exprese la función booleana $F = xy + x'z$ en un producto de forma maxtérmino. Primero convierta la función en términos OR usando la ley distributiva:

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

La función tiene tres variables: x, y y z. Cada término OR pierde una variable; por tanto:

$$x' + y = x' + y + zz' = (x' + y + z)(x' + y + z')$$
$$x + z = x + z + yy' = (x + y + z)(x + y' + z)$$
$$y + z = y + z + xx' = (x + y + z)(x' + y + z)$$

Se combinan todos los términos y se eliminan los que aparecen más de una vez y, por último, se obtiene:

$$F = (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z')$$
$$= M_0M_2M_4M_5$$

Una forma conveniente de expresar esta función es como sigue:

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

El símbolo de producto, Π, denota la operación AND de maxtérminos; los números son los maxtérminos de la función.

Conversión entre formas canónicas

El complemento de una función expresada como suma de mintérminos es igual a la suma de los mintérminos perdidos de la función original. Esto se debe a que la función original está expresada por los mintérminos que hacen la función igual a 1, mientras que su complemento es un 1 para los términos en los que la función es un 0. Como ejemplo, considérese la función:

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

Esta tiene un complemento que puede expresarse como:

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

Ahora bien, si se toma el complemento de F' por el teorema de De Morgan, se obtiene F en una forma diferente:

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

La última conversión se sigue de la definición de mintérminos y maxtérminos como se muestra en la Tabla 2-3. Por la tabla, es claro que la siguiente relación es válida:

$$m_j' = M_j$$

Esto es, el maxtérmino con subíndice j es un complemento del mintérmino con el mismo subíndice j y viceversa.

El último ejemplo demuestra la conversión entre una función expresada en sumas de mintérminos y su equivalente en producto de maxtérminos. Un argumento similar mostrará que la conversión entre el producto de maxtérminos y la suma de mintérminos es similar. Ahora se enuncia un procedimiento general de conversión. Para convertir de una forma canónica a otra, se intercambian los símbolos Π y Σ y se listan los números perdidos de la forma original. Como otro ejemplo, la función:

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

se expresa en la forma de producto de maxtérminos. Su conversión en suma de mintérminos es:

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

Obsérvese que, con objeto de encontrar los términos perdidos, debe tomarse en cuenta que el número total de mintérminos o maxtérminos es 2^n, donde n es el número de variables binarias en la función.

Formas estándar

Las dos formas canónicas del álgebra booleana son formas básicas que se obtienen al leer una función de la tabla de verdad. Estas formas muy rara vez son las que tienen el menor número de literales, debido a que cada mintérmino y maxtérmino debe contener, por definición, todas las variables ya sea complementadas o sin complementar.

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

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

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

La expresión tiene tres términos producto de una, dos y tres literales cada uno, respectivamente. Su suma es, en efecto, una operación OR.

Un *producto de sumas* es una expresión booleana que contiene términos OR, llamados *términos suma.* Cada término puede tener cualquier número de literales. El *producto* denota la operación AND de esos términos. Un ejemplo de una función expresada en producto de sumas es:

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

Esta expresión tiene tres términos suma de una, dos y cuatro literales cada uno. El producto es una operación AND. El uso de las palabras *producto* y *suma* surge de la similitud de la operación AND con el producto aritmético (multiplicación) y la semejanza de la operación OR con la suma aritmética (adición).

Una función booleana puede expresarse en una forma no estándar. Por ejemplo, la función:

$$F_3 = (AB + CD)(A'B' + C'D')$$

no es una suma de productos ni un producto de suma. Puede cambiarse a una forma estándar usando la ley distributiva para eliminar los paréntesis:

$$F_3 = A'B'CD + ABC'D'$$

2-6 OTRAS OPERACIONES LOGICAS

Cuando los operadores binarios AND y OR se colocan entre dos variables x y y, forman dos funciones booleanas $x \cdot y$ y $x + y$, respectivamente. Se enunció previamente que hay 2^{2^n} funciones para n variables binarias. Para dos variables, $n = 2$ y el

número de funciones booleanas posibles es 16. Por tanto, las funciones AND y OR son sólo dos de un total de 16 funciones posibles formadas con dos variables binarias. Sería instructivo encontrar las otras 14 funciones e investigar sus propiedades.

Las tablas de verdad para las 16 funciones formadas con dos variables binarias x y y se listan en la Tabla 2-5. En esta tabla, cada una de las 16 columnas, de F_0 a F_{15}, representa una tabla de verdad de una función posible para las dos variables dadas x y y. Obsérvese que la función está determinada por las 16 combinaciones binarias que pueden asignarse a F. Algunas de las funciones se muestran con un símbolo de operador. Por ejemplo, F_1 representa la tabla de verdad para AND y F_7 representa la tabla de verdad para OR. Los símbolos de los operadores para esas funciones son $(\cdot)$ y $(+)$, respectivamente.

Las 16 funciones que se listan en forma de tabla de verdad pueden expresarse de manera algebraica mediante expresiones booleanas. Esto se muestra en la primera columna de la Tabla 2-6. Las expresiones booleanas que se listan se simplifican a su número mínimo de literales.

Aun cuando cada función puede expresarse en términos de las operaciones booleanas AND, OR y NOT, no hay razón para que no puedan asignarse símbolos especiales de operador para expresar las otras funciones. Tales símbolos de operador se listan en la segunda columna de la Tabla 2-6. Sin embargo, todos los nuevos símbolos que se muestran, excepto para el símbolo del operador OR-excluyente $\oplus$, no son de uso común por los diseñadores digitales.

Cada una de las funciones de la Tabla 2-6 se lista con un nombre que la acompaña y un comentario que explica la función en cierta forma. Las 16 funciones listadas pueden subdividirse en tres categorías:

1. Dos funciones que producen un constante 0 o 1.
2. Cuatro funciones con operaciones unitarias de complemento y transferencia.
3. Diez funciones con operadores binarios que definen ocho operaciones diferentes AND, OR, NAND, NOR, O-excluyente, equivalencia, inhibición e implicación.

Cualquier función puede ser igual a una constante, pero una función binaria puede ser igual sólo a 1 o 0. La función complemento produce el complemento de cada

TABLA 2-5 Tablas de verdad para las 16 funciones de dos variables binarias

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Símbolo Operador			$\cdot$	/		/		$\oplus$	+	$\downarrow$	$\odot$	′	$\subset$	′	$\supset$	$\uparrow$	

TABLA 2-6 Expresiones booleanas para las 16 funciones de dos variables

Funciones booleanas	Símbolo del operador	Nombre	Comentarios
$F_0 = 0$		Nulo	Constante binaria 0
$F_1 = xy$	$x \cdot y$	AND	x y y
$F_2 = xy'$	x/y	Inhibición	x pero no y
$F_3 = x$		Transferencia	x
$F_4 = x'y$	y/x	Inhibición	y pero no x
$F_5 = y$		Transferencia	y
$F_6 = xy' + x'y$	$x \oplus y$	Excluyente-OR	x o y pero no ambas
$F_7 = x + y$	$x + y$	OR	x o y
$F_8 = (x + y)'$	$x \downarrow y$	NOR	NOT-OR
$F_9 = xy + x'y'$	$x \odot y$	Equivalencia*	x igual a y
$F_{10} = y'$	y'	Complemento	No y
$F_{11} = x + y'$	$x \subset y$	Implicación	Si y, entonces x
$F_{12} = x'$	x'	Complemento	No x
$F_{13} = x' + y$	$x \supset y$	Implicación	Si x, entonces y
$F_{14} = (xy)'$	$x \uparrow y$	NAND	NOT-AND
$F_{15} = 1$		Identidad	Constante binaria 1

*La *equivalencia* también se conoce como *igualdad, coincidencia* y excluyente NOR.

una de las variables binarias. Una función que es igual a una variable de entrada recibe el nombre de *transferencia,* ya que la variable x o y se transfiere a través de la compuerta que forma la función sin cambiar su valor. De los diez operadores binarios, cuatro (que corresponden a las funciones de inhibición e implicación) los utilizan especialistas en lógica, pero rara vez se usan en la lógica de computadora. Se han mencionado operadores AND y OR junto con el álgebra booleana. Las otras cuatro funciones se emplean en forma extensa en el diseño de sistemas digitales.

La función NOR es el complemento de la función OR y su nombre es la abreviatura de *no*-OR. En forma similar, NAND es el complemento de AND y es una abreviatura de *no*-AND. Excluyente OR se abrevia XOR o EOR es similar a OR, pero excluye la combinación *tanto de x* como de *y* cuando son iguales a 1. La equivalencia es una función que es 1 cuando dos variables binarias son iguales, esto es, cuando ambas son 0 o ambas son 1. La excluyente OR y las funciones de equivalencia son los complementos una de otra. Esto puede verificarse con facilidad por la inspección de la Tabla 2-5. La tabla de verdad para la excluyente OR es F_6 y para la equivalencia es F_9 y, estas dos funciones son los complementos una de la otra. Por esta razón, la función de equivalencia con frecuencia se denomina excluyente NOR, es decir, excluyente-OR-NOT.

El álgebra booleana, como se define en la Sección 2-2, tiene dos operadores binarios, que se han denominado AND y OR y un operador unario, NOT (complemento). Por las definiciones, se deduce cierto número de propiedades de estos operadores y ahora se han definido otros operadores binarios en términos de ellos. No hay

nada excepcional en este procedimiento. Se puede empezar también con el operador NOR (↓) , por ejemplo, y definir después AND, OR y NOT en términos de él. Sin embargo, hay buenas razones para introducir el álgebra booleana en el modo que se ha hecho. Los conceptos de "y", "o" y "no" son familiares y las personas los utilizan para expresar ideas lógicas en la vida cotidiana. No obstante, los postulados de Huntington reflejan la naturaleza dual del álgebra, con énfasis en la simetría de + y · de uno con respecto a otro.

2-7 COMPUERTAS LOGICAS DIGITALES

Ya que las funciones booleanas se expresan en términos de operaciones AND, OR y NOT, es fácil implantar una función booleana con estos tipos de compuertas. La posibilidad de construir compuertas para otras operaciones lógicas es de interés práctico. Los factores que hay que pesar cuando se considera la construcción de otros tipos de compuertas lógicas son (1) la factibilidad y economía de producir la compuerta con componentes físicos, (2) la posibilidad de extender la compuerta a más de dos entradas, (3) las propiedades básicas del operador binario como conmutabilidad y asociatividad y (4), la habilidad de la compuerta para implantar compuertas booleanas solas o junto con otras compuertas.

De las 16 funciones que se definen en la Tabla 2-6, dos son iguales a una constante y otras cuatro se repiten dos veces. Solo que dan diez funciones que considerar como candidatos para compuertas lógicas. Dos, inhibición y complicación, no son conmutativas o asociativas y, por tanto, no es práctico usarlas como compuertas lógicas estándar. Las otras ocho: complemento, transferencia, AND, OR, NAND, NOR, excluyente-OR, y equivalencia, se utilizan como compuertas estándar en el diseño digital.

Los símbolos gráficos y las tablas de verdad de las ocho compuertas se muestran en la Fig. 2-5. Cada compuerta tiene una o dos variables binarias de entrada designadas por x y y y una variable binaria de salida designada por F. Los circuitos AND, OR e inversor se definen en la Fig. 1-6. El circuito inversor invierte el sentido lógico de una variable binaria. Produce la función NOR o complemento. El pequeño círculo en la salida del símbolo gráfico de un inversor designa el complemento lógico. El símbolo de triángulo por sí mismo denota un circuito buffer. Un buffer produce la función de *transferencia* pero no produce alguna operación lógica particular, ya que el valor binario de la salida es igual al valor binario de la entrada. El circuito se usa simplemente para amplificación de potencia de la señal y es equivalente a dos inversores conectados en cascada.

La función NAND es el complemento de la función AND, como se indica por un símbolo gráfico, que consta de un símbolo gráfico AND seguido de un círculo pequeño. La función NOR es el complemento de la función OR y usa un símbolo gráfico OR seguido de un círculo pequeño. Las compuertas NAND y NOR se utilizan en forma extensa como compuertas lógicas estándar y de hecho se emplean más que las compuertas AND y OR. Esto se debe a que las compuertas NAND y NOR se construyen fácilmente con circuitos de transistores y a que las funciones booleanas pueden implementarse con sencillez con dichas compuertas.

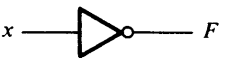
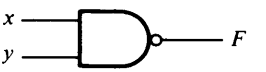
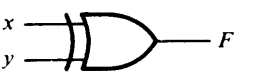
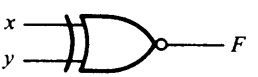
Nombre	Símbolo gráfico	Función algebraica	Tabla de verdad
AND		$F = xy$	$x\ y \mid F$ $0\ 0 \mid 0$ $0\ 1 \mid 0$ $1\ 0 \mid 0$ $1\ 1 \mid 1$
OR		$F = x + y$	$x\ y \mid F$ $0\ 0 \mid 0$ $0\ 1 \mid 1$ $1\ 0 \mid 1$ $1\ 1 \mid 1$
Inversor		$F = x'$	$x \mid F$ $0 \mid 1$ $1 \mid 0$
Buffer		$F = x$	$x \mid F$ $0 \mid 0$ $1 \mid 1$
NAND		$F = (xy)'$	$x\ y \mid F$ $0\ 0 \mid 1$ $0\ 1 \mid 1$ $1\ 0 \mid 1$ $1\ 1 \mid 0$
NOR		$F = (x + y)'$	$x\ y \mid F$ $0\ 0 \mid 1$ $0\ 1 \mid 0$ $1\ 0 \mid 0$ $1\ 1 \mid 0$
Excluyente-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	$x\ y \mid F$ $0\ 0 \mid 0$ $0\ 1 \mid 1$ $1\ 0 \mid 1$ $1\ 1 \mid 0$
Excluyente-NOR o equivalente		$F = xy + x'y'$ $= x \odot y$	$x\ y \mid F$ $0\ 0 \mid 1$ $0\ 1 \mid 0$ $1\ 0 \mid 0$ $1\ 1 \mid 1$

Figura 2-5 Compuertas lógicas digitales.

La compuerta excluyente-OR tiene un símbolo gráfico similar al de la compuerta OR excepto por la línea adicional curva en el lado de entrada. La equivalencia, o compuerta excluyente NOR es el complemento de la excluyente OR, como se indica por el círculo pequeño en el lado de salida del símbolo gráfico.

Extensión a entradas múltiples

Las compuertas que se muestran en la Fig. 2-5, excepto por el inversor y el buffer, pueden extenderse para tener más de dos entradas. Una compuerta puede extenderse para tener entradas múltiples si la operación binaria que representa es conmutativa y asociativa. Las operaciones AND y OR, definidas en el álgebra booleana, poseen esas dos propiedades. Para la función OR se tiene:

$$x + y = y + x \qquad \text{conmutativa}$$

y

$$(x + y) + z = x + (y + z) = x + y + z \qquad \text{asociativa}$$

lo cual indica que las entradas de compuerta pueden intercambiarse y que la función O puede extenderse a tres o más variables.

Las funciones NAND y NOR son conmutativas y sus compuertas pueden extenderse para tener más de dos entradas, siempre que se modifique ligeramente la definición de la operación. La dificultad es que los operadores NAND y NOR no son asociativos, esto es, $(x \downarrow y) \downarrow z \neq x \downarrow (y \downarrow z)$, como se muestra en la Fig. 2-6 y a continuación:

$$(x \downarrow y) \downarrow z = [(x + y)' + z]' = (x + y)z' = xz' + yz'$$
$$x \downarrow (y \downarrow z) = [x + (y + z)']' = x'(y + z) = x'y + x'z$$

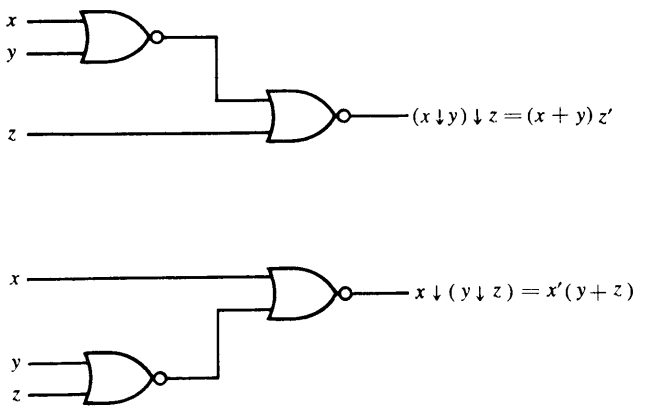


Figura 2-6 Demostración de la no asociabilidad del operador NOR; $(x \downarrow y) \downarrow z \neq x(y \downarrow z)$.

Para superar esta dificultad, se define la compuerta múltiple NOR (o NAND) como una compuerta complementada OR (o AND). Así, por definición, se tiene:

$$x\downarrow y\downarrow z = (x + y + z)'$$
$$x\uparrow y\uparrow z = (xyz)'$$

Los símbolos gráficos para las compuertas de tres entradas se muestran en la Fig. 2-7. Al indicar por escrito operaciones NOR y NAND en cascada, deben utilizarse los paréntesis correctos para indicar la secuencia apropiada de las compuertas. Para demostrar esto, considérese el circuito de la Fig. 2-7(c). La función booleana para el circuito debe escribirse como:

$$F = [(ABC)'(DE)']' = ABC + DE$$

La segunda expresión se obtiene a partir del teorema de De Morgan. Muestra también que una expresión en suma de productos puede inplantarse con compuertas NAND. En las Secciones 3-6, 4-7 y 4-8 puede encontrarse la exposición detallada de las compuertas NAND y NOR.

Las compuertas excluyente OR y de equivalencia son conmutativas y asociativas y pueden extenderse a más de dos entradas. Sin embargo, las compuertas excluyente OR de entradas múltiples no son usuales desde el punto de vista del hardware. De hecho, incluso una función de dos entradas por lo común se construye con otros tipos de compuertas. Además, la definición de esas funciones debe modificarse cuando se extienden a más de dos variables. La excluyente-OR es una función *impar*, esto es, es igual a 1 si las variables de entrada tienen un número impar de 1. La función de equivalencia es una función *par*, es decir, es igual a 1 si las variables de entrada tienen un número par de 0. La construcción de una función excluyente OR de tres entradas se muestra en la Fig. 2-8. En forma normal se implementa con compuertas de dos entradas en cascada, como se muestra en (a). De manera gráfica, puede representarse con una sola compuerta de tres entradas como se muestra en (b). La tabla de verdad en

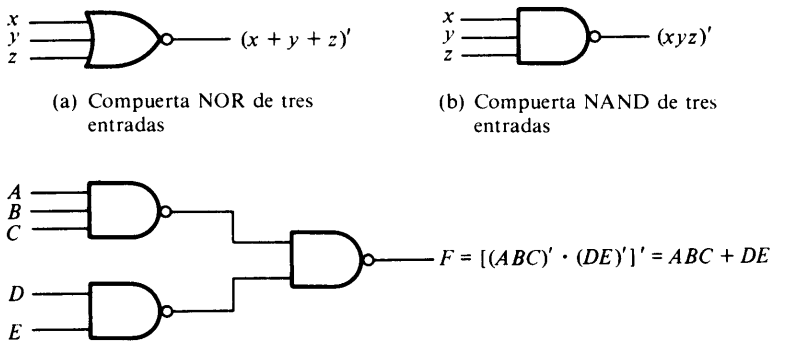
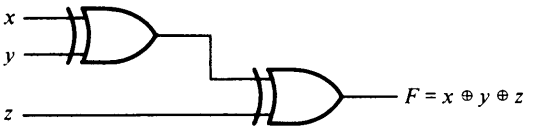
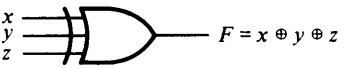


Figura 2-7 Compuertas de entradas múltiples NOR y NAND puestas en cascada.



(a) Uso de compuertas con dos entradas



(b) Una compuerta de tres entradas

x	y	z	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

(c) Tabla de verdad

Figura 2-8 Compuerta excluyente OR de tres entradas.

(c) indica con claridad que la salida F es igual a 1 si sólo una entrada es igual a 1 o si todas las tres entradas son iguales a 1, esto es, cuando el número total de 1 en las variables de entrada es *impar*. En la Sección 4-9 puede encontrarse un análisis adicional de la excluyente OR y la equivalencia.

2-8 FAMILIAS LOGICAS DIGITALES IC

El IC se introdujo en la Sección 1-9, donde se estableció que los circuitos digitales se construyen en forma invariable con IC. En las secciones previas se han expuesto diversas compuertas lógicas digitales, ahora ya están dadas las condiciones para presentar las compuertas IC y exponer sus propiedades generales.

Las compuertas digitales IC se clasifican no sólo por su operación lógica, sino también por la familia de circuitos lógicos a las cuales pertenecen. Cada familia lógica tiene su propio circuito electrónico básico con el cual se desarrollan circuitos y funciones digitales más complejos. El circuito básico de cada familia es una compuerta NAND o bien una compuerta NOR. Los componentes electrónicos que se emplean en la construcción del circuito básico por lo general se utilizan para nombrar la familia lógica. En el comercio se han introducido muchas familias lógicas diferentes de IC digitales. Las que han alcanzado un amplio uso popular se listan a continuación.

TTL	Lógica de transistor-transistor
ECL	Lógica de emisor acoplado
MOS	Semiconductor de óxido metálico
CMOS	Semiconductor complementario de óxido metálico
I^2L	Lógica de inyección integrada

La lógica TTL tiene una lista extensa de funciones digitales y hoy día es la familia lógica más popular. La lógica ECL se utiliza en sistemas que requieren

operaciones de alta velocidad. Las MOS e I^2L se usan en circuitos que requieren alta densidad de componentes y la CMOS se emplea en sistemas que necesitan bajo consumo de potencia.

El análisis del circuito electrónico en cada familia básica se presenta en el Capítulo 10. El lector familiarizado con la electrónica básica puede consultar el Capítulo 10 para así conocer estos circuitos electrónicos. Aquí la exposición se limita a las propiedades generales de las diversas compuertas IC disponibles en forma comercial.

Debido a la alta densidad con la cual pueden fabricarse los transistores en MOS e I^2L, estas dos familias son las que más se utilizan para las funciones LSI. Las otras tres familias, TTL, ECL, y CMOS, tienen dispositivos LSI y también un gran número de dispositivos MSI y SSI. Los dispositivos SSI son los que incluyen un pequeño número de compuertas o flip-flops (presentados en la Sección 6-2) en un paquete IC. El límite del número de circuitos en los dispositivos SSI es el número de clavijas en el paquete. Por ejemplo, un paquete de 14 clavijas puede acomodar sólo cuatro compuertas de dos entradas, debido a que cada compuerta requiere tres clavijas externas, dos para cada una de las entradas y una para la salida, con un total de 12 clavijas. Las dos clavijas restantes se necesitan para suministrar potencia a los circuitos.

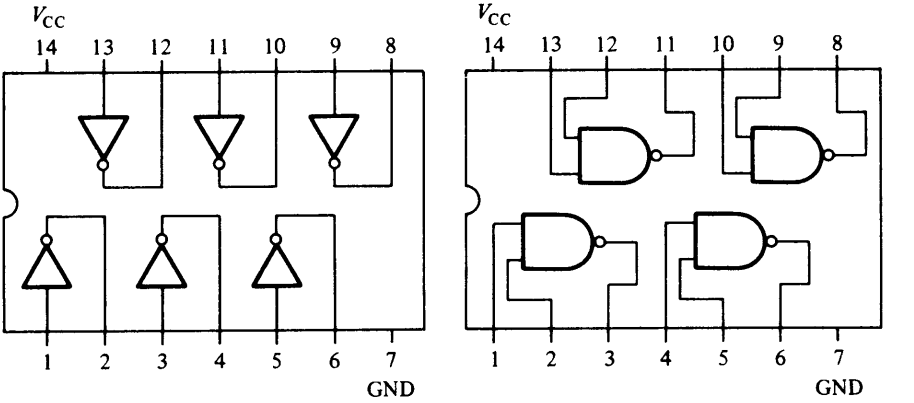
Algunos circuitos típicos SSI se muestran en la Fig. 2-9. Cada IC se encapsula un paquete de 14 o 16 clavijas. Las clavijas se numeran a lo largo de los dos lados del paquete y especifican las conexiones que pueden hacerse. Las compuertas dibujadas dentro de los IC son sólo para información y no pueden verse debido a que el paquete IC real aparece como se muestra en la Fig. 1-8.

Los IC de la familia TTL por lo común se distinguen por designaciones numéricas como las series 5400 y 7400. La primera tiene amplios márgenes de temperatura de operación, adecuados para uso militar y, la segunda tiene márgenes más reducidos de temperatura, adecuados para uso industrial. La designación numérica de la serie 7400 significa que los paquetes IC están numerados como 7400, 7401, 7402, etc. Algunos proveedores ponen a la disposición IC de la familia TTL con denominaciones numéricas diferentes, como las series 9000 u 8000.

En la Fig. 2-9(a) se muestran dos circuitos TTL SSI. La serie 7404 proporciona seis (hex) inversores en un paquete. La serie 7400 proporciona cuatro (cuádruple) puertas NAND de dos entradas. Las terminales marcadas V_{CC} y GND son las clavijas de suministro de potencia que requieren un voltaje de 5 volts para la operación apropiada.

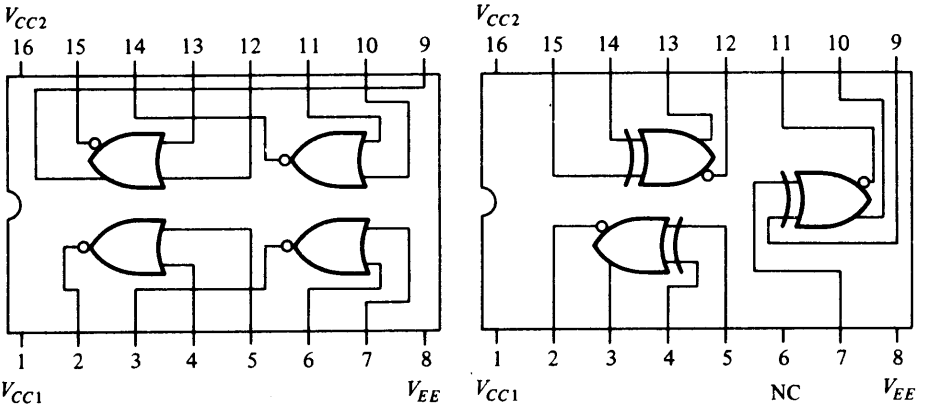
El tipo más común de ECL se designa como la serie 10 000. En la Fig. 2-9(b) se muestran dos circuitos ECL. La serie 10102 proporciona compuertas NOR de dos entradas. Obsérvese que una compuerta ECL puede tener dos salidas, una para la función NOR y otra para la función 0 (clavija 9 del 10102 IC). El 10107 IC proporciona tres compuertas excluyentes OR. Aquí hay de nuevo dos salidas para cada compuerta; la otra salida de la función excluyente NOR o de equivalencia. Las compuertas ECL tienen tres terminales para suministro de potencia. V_{CC1} y V_{CC2} por lo común se conectan a tierra y V_{EE} a un suministro de -5.2 volt.

Los circuitos CMOS de la serie 4000 se muestran en la Fig. 2-9(c). Sólo pueden acomodarse en el 4002 dos compuertas NOR de cuatro entradas, debido a la limitación de clavijas. El tipo 4059 proporciona seis compuertas buffer. Ambos ICs tienen



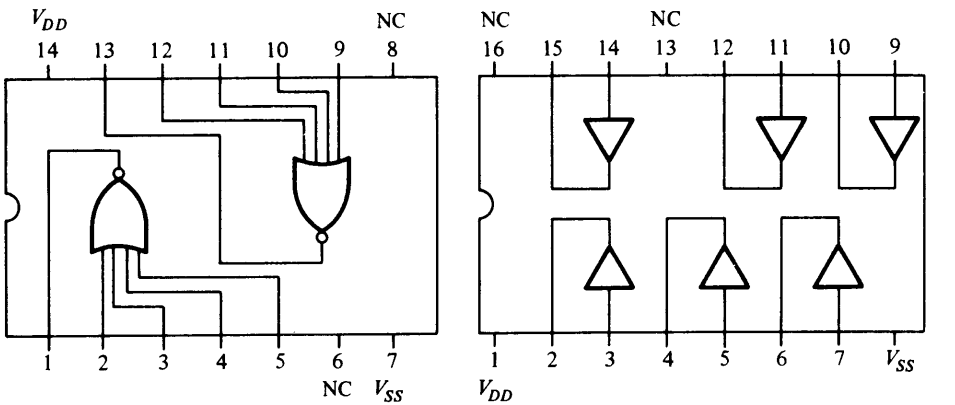
Inversores HEX-7408 7400-Cuádruple con compuertas NAND de 2 entradas

(a) Compuertas TTL.



10102-Cuádruple con compuertas NOR de 2 entradas 10107-Triple con compuertas excluyente OR/NOR

(b) Compuertas ECL



4002-Dual con compuertas NOR de 4 entradas 4050-Buffer Hex.

(c) Compuertas CMOS.

Figura 2-9 Algunas compuertas típicas en circuitos integrados.

dos terminales sin uso marcadas NC (no conexión). La terminal marcada V_{DD} requiere un voltaje en el suministro de potencia de 3 a 15 volts, en tanto V_{SS} por lo común se conecta a tierra.

Lógicas positiva y negativa

La señal binaria en la entrada o salida de cualquier compuerta puede tener uno de dos valores, excepto durante la transición. Un valor de señal representa la lógica 1 y el otro, la lógica 0. Ya que se asignan dos valores de señal a dos valores lógicos, existen dos diferentes asignaciones de señales a lógica. Debido al principio de dualidad de álgebra booleana, un intercambio en la asignación del valor de señal resulta en el implante de una función dual.

Considérense los dos valores de una señal binaria tal como se muestra en la Fig. 2-10. un valor debe ser más alto que el otro, ya que los dos valores deben ser diferentes con objeto de distinguir entre ellos. Se designa el nivel alto por *H* y el nivel bajo por *L*. Hay dos elecciones para la asignación del valor de lógica. La elección del nivel alto *H* para que represente la lógica 1, como se muestra en la Fig. 2-10(a), define un sistema de *lógica positiva.* La elección del nivel bajo *L* para representar la lógica 1, como se muestra en la Fig. 12-10(b), define un sistema de *lógica negativa.* Los términos *positiva* y *negativa* algunas veces pueden ser engañosos, ya que ambas señales de valor pueden ser positivas o negativas. No es la polaridad de la señal la que determina el tipo de lógica, sino más bien la asignación de valores lógicos de acuerdo con las amplitudes relativas de las señales.

Las hojas de datos de los circuitos integrados definen las funciones digitales no en términos de la lógica 1 o lógica 0, sino más bien en términos de los niveles *H* y *L*. Se deja al usuario decidir la asignación de una lógica positiva o negativa. Los voltajes de alto nivel y bajo nivel para las tres familias lógicas digitales IC se listan en la Tabla 2-7. En cada familia, hay unos márgenes de valores de voltaje que el circuito reconocerá como nivel alto o bajo. El valor típico es el que más se encuentra por lo común. En la tabla también se listan los requisitos del suministro de voltaje para cada familia como una referencia.

La familia TTL tiene valores típicos de $H = 3.5$ volts y $L = 0.2$ volts. La familia ECL tiene dos valores negativos, con $H = -0.8$ volts y $L = -1.8$ volts. Obsérvese que aunque ambos niveles son negativos, el más elevado eso —0.8. Las compuertas CMOS pueden usar un voltaje de suministro V_{DD} en cualquier parte entre 3 y 15 volts; en forma típica, utilizan ya sea 5 o 10 volts. Los valores de señal en las CMOS son una función

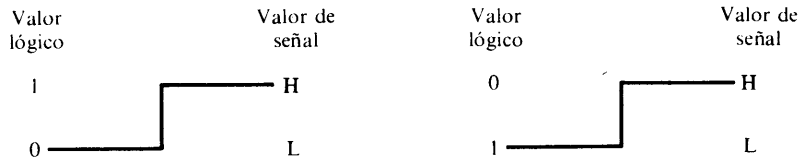


Figura 2-10 Asignación de amplitud de señal y tipo de lógica.

TABLA 2-7 Niveles H y L en las familias lógicas IC

Tipo de familia IC	Voltaje de suministro (V)	Alto nivel de voltaje (V)		Bajo nivel de voltaje (V)	
		Márgenes	Típico	Márgenes	Típico
TTL	$V_{CC} = 5$	2.4–5	3.5	0–0.4	0.2
ECL	$V_{EE} = -5.2$	−0.95–−0.7	−0.8	−1.9–−1.6	−1.8
CMOS	$V_{DD} = 3$–10	V_{DD}	V_{DD}	0–0.5	0
Lógica positiva:			lógica 1		lógica 0
Lógica negativa:			lógica 0		lógica 1

del voltaje de suministro con $H = V_{DD}$ y $L = 0$ volts. Las asignaciones de polaridad para lógica positiva y negativa también se indican en la tabla.

A la luz de esta exposición, es necesario justificar los símbolos lógicos usados para los IC que se listan en la Fig. 2-9. Tómese, por ejemplo, una de las compuertas del IC 7400. Esta compuerta se muestra en forma de diagrama de bloques en la Fig. 2-11(b). La tabla de verdad del fabricante para esta compuerta dada en una hoja de datos se muestra en la Fig. 2-11(a). En esta tabla se especifica el comportamiento físico de la compuerta, con H de 3.5 volts en forma típica y L de 0.2 volts. Esta compuerta física puede funcionar ya sea como compuerta NAND o NOR, dependiendo de la asignación de polaridad.

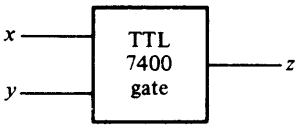
En la tabla de verdad de la Fig. 2-11(c) se supone la asignación de lógica positiva con $H = 1$ y $L = 0$. Al verificar esta tabla de verdad en la Fig. 2-5, se reconoce como una compuerta NAND. El símbolo gráfico para una compuerta NAND de lógica positiva se muestra en la Fig. 2-11(b) y es similar a la que se adoptó con anterioridad.

Ahora considérese la asignación de lógica negativa a esta compuerta física con $L = 1$ y $H = 0$. El resultado es la tabla de verdad que se muestra en la Fig. 2-11(e). Puede reconocerse que esta tabla representa la función NOR aun cuando sus entradas están listadas hacia atrás. El símbolo gráfico para una compuerta NOR de lógica negativa se muestra en la Fig. 2-11(f). El pequeño triángulo en los alambres de entrada y salida designa un *indicador de polaridad.* La presencia de este indicador de polaridad a lo largo de una terminal indica que se asigna una lógica negativa a la terminal. Por tanto, la misma compuerta física puede funcionar ya sea como una NAND de lógica positiva o como una NOR de lógica negativa. La que está dibujada en el diagrama depende por completo de la asignación de polaridad que desee emplear el diseñador.

De manera semejante, es posible mostrar que una NOR de lógica positiva es la misma compuerta física que una NAND de lógica negativa. La misma relación es válida entre las compuertas AND y OR o entre las compuertas excluyente-OR y equivalencia. En cualquier caso, si se supone lógica negativa en cualquier terminal de entrada o salida, es necesario incluir el símbolo del triángulo indicador de polaridad junto a la terminal. Algunos diseñadores digitales utilizan esta convención para facilitar el diseño de circuitos digitales cuando se usan exclusivamente compuertas NAND o NOR. En este libro no se emplea esta simbología, pero se recurre a otros

x	y	z
L	L	H
L	H	H
H	L	H
H	H	L

(a) Tabla de verdad de términos de H y L.



(b) Diagrama de bloque de compuerta.

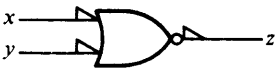
x	y	z
0	0	1
0	1	1
1	0	1
1	1	0

(c) Tabla de verdad para lógica positiva: $H = 1, L = 0$.

(d) Símbolo gráfico para la compuerta NAND de lógica positiva.

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

(e) Tabla de verdad para lógica negativa: $L = 1, H = 0$.



(f) Símbolo gráfico para la compuerta NOR de lógica negativa.

Figura 2-11 Demostración de las lógicas positiva y negativa.

métodos para diseñar con las compuertas NAND y NOR. Obsérvese que los IC que se presentan en la Fig. 2-9 se muestran con sus símbolos gráficos de lógica positiva. Podrían haberse ilustrado con sus símbolos de lógica negativa si así se hubiera deseado.

La conversión de lógica positiva en lógica negativa y viceversa es en esencia una operación que cambia los 1 en 0 y los 0 en 1, tanto en las entradas como en las salidas de una computadora. Ya que esta operación produce el dual de una función, el cambio de todas las terminales de una polaridad a la otra resulta en tomar la dual de la función. El resultado de esta conversión es que todas las operaciones AND se convierten en operaciones OR (o símbolos gráficos) y viceversa. Además, no debe olvidarse incluir el indicador de polaridad en los símbolos gráficos cuando se supone lógica negativa.

El pequeño triángulo que representa un indicador de polaridad y el pequeño círculo que representa una complementación tienen efectos similares pero diferente significados. Por tanto, puede reemplazarse uno por otro, pero la interpretación es diferente. Un círculo seguido por un triángulo, como en la Fig. 2-11(f), representa una complementación seguida por un indicador de polaridad de lógica negativa. Los dos se cancelan uno a otro y ambos pueden eliminarse. Pero si se eliminan ambos, entonces las entradas y salidas de la compuerta representarán polaridades diferentes.

Características especiales

Las características de las familias IC de lógica digital por lo común se comparan por el análisis de circuito de la compuerta, básica en cada familia. Los parámetros más importantes que se evalúan y comparan son la salida en abanico (multiplicidad de conexiones en la salida), disipación de potencia, retardo de propagación y margen de ruido. Se explicarán primero las propiedades de este parámetro y después se utilizarán para comparar las familias IC lógicas.

El *abanico de salida* especifica el número de cargas estándar que pueden impulsar la salida de una compuerta sin menoscabar su operación normal. Una carga estándar por lo común se define como la cantidad de corriente necesaria por una entrada de otra compuerta en la misma familia IC. Algunas veces el término *cargado* se usa en lugar de abanico de salida. Este término se deriva del hecho de que la salida de una compuerta puede suministrar una cantidad limitada de corriente, arriba de la cual cesa su operación apropiada y se dice que está sobrecargada. La salida de una compuerta por lo general se conecta a las entradas de otras compuertas similares. Cada entrada consume una cierta cantidad de potencia de la entrada de la compuerta, de modo que cada conexión adicional se agrega a la carga de la compuerta. Las "reglas de carga" por lo común se listan para una familia de circuitos digitales estándar. Estas reglas especifican la máxima cantidad de carga permitida para cada salida de cada circuito. El exceder la carga máxima especificada puede causar un mal funcionamiento debido a que el circuito no puede suministrar la potencia demandada de él. El abanico de salida es el número máximo de entradas (a otros circuitos) que pueden conectarse a la salida de una compuerta y se expresa por un número.

Las capacidades del abanico de salida de una compuerta pueden considerarse cuando se simplifican las funciones booleanas. Debe tenerse cuidado de no desarrollar expresiones que resulten en una compuerta sobrecargada. Los amplificadores no inversores o buffer algunas veces se emplean para proporcionar capacidades adicionales de impulsión para cargas pesadas.

La *disipación de potencia* es la potencia suministrada requerida para operar la compuerta. Este parámetro se expresa en miliwatts (mW) y representa la potencia real disipada en la compuerta. El número que representa este parámetro no incluye la potencia suministrada por otra compuerta; más bien, representa la potencia suministrada a la compuerta por el suministro de potencia. Un IC con cuatro compuertas requerirá, de su suministro de potencia, cuatro veces la potencia disipada por cada compuerta. En un sistema dado, puede haber muchos IC y, la potencia requerida por cada IC debe considerarse. La disipación total de potencia en un sistema es la suma total de la potencia disipada en todos los IC.

El *retardo de propagación* es el retardo de tiempo de transición promedio para que una señal se propague desde la entrada a la salida cuando la señal binaria cambia en valor. Las señales a través de una compuerta toman cierta cantidad de tiempo para propagarse desde las entradas a la salida. Este intervalo de tiempo se define como el retardo de propagación de la compuerta. El retardo de propagación se expresa en nanosegundos (ns) y, un ns es igual a 10^{-9} de un segundo.

Las señales que viajan de las entradas de un circuito digital a sus salidas pasan a través de una serie de compuertas. La suma de los retardos de propagación a través de las compuertas es el retardo total de propagación del circuito. Cuando la velocidad de operación es importante, cada compuerta debe tener un pequeño retardo de propagación y el circuito digital debe tener un número mínimo de compuertas en serie entre las entradas y las salidas.

En la mayoría de los circuitos digitales las señales de entrada se aplican en forma simultánea a más de una compuerta. Todas las compuertas que reciben sus entradas exclusivamente desde las entradas externas, constituyen el primer nivel lógico del circuito. Las compuertas que reciben cuando menos una entrada de una salida de una compuerta del primer nivel lógico se considera que están en el segundo nivel lógico, y en forma semejante, para el tercer nivel y los más altos. El retardo total de propagación del circuito es igual al retardo de propagación de una compuerta multiplicado por el número de niveles lógicos en el circuito. Luego, una reducción en el número de niveles lógicos produce una reducción del retardo de señal y en circuitos más rápidos. La reducción del retardo de propagación en los circuitos puede ser más importante que la reducción en el número total de compuertas si la velocidad de operación es un factor principal.

El *margen de ruido* es el máximo voltaje de ruido añadido a la señal de entrada de un circuito digital que no causa un cambio indeseable en la salida del circuito. Hay dos tipos de ruido que considerar: el ruido CC es causado por una deriva en los niveles de voltaje de una señal. El ruido CA es un pulso aleatorio que puede crearse por otras señales de interrupción. Por eso, el ruido es un término que se utiliza para denominar una señal indeseable que está superpuesta sobre la señal normal de operación. La capacidad de los circuitos para operar en forma confiable en un ambiente de ruido es importante en muchas aplicaciones. El margen de ruido se expresa en volts (V) y representa la señal de ruido máximo que puede tolerarse por la compuerta.

Características de las familias lógicas IC

El circuito básico de la familia lógica TTL es la compuerta NAND. Hay muchas versiones de la TTL y tres de ellas se listan en la Tabla 2-8. En esta tabla se dan las

TABLA 2-8 Características típicas de las familias lógicas IC

Familia lógica IC	Abanico de salida	Disipación de potencia (mW)	Retardo de propagación (ns)	Margen de ruido (V)
Estándar TTL	10	10	10	0.4
Schottky TTL	10	22	3	0.4
Baja potencia Schottky TTL	20	2	10	0.4
ECL	25	25	2	0.2
CMOS	50	0.1	25	3

características generales de las familias lógicas IC. Los valores que se listan son representativos en una base de comparación. Para cualquier familia o versión, los valores pueden tener cierta variación.

La compuerta estándar TTL fue la primera versión de la familia TTL. Conforme progresó la tecnología, se agregaron mejoras adicionales. La TTL Schottky es una última mejora que reduce el retardo de propagación, pero resulta en un aumento de la disipación de potencia. La versión TTL Schottky de baja potencia sacrifica cierta velocidad para reducir la disipación de potencia. Tiene el mismo retardo de propagación que la TTL estándar, pero la disipación de potencia se reduce en forma considerable. El abanico de salida de la TTL estándar es 10, pero la versión Schottky de baja potencia tiene un abanico de salida de 20. Bajo ciertas condiciones las otras versiones también pueden tener un abanico de salida de 20. El margen de ruido es mejor que 0.4 V, con un valor típico de 1 V.

El circuito básico de la familia ECL es la compuerta NOR. La ventaja especial de las compuertas ECL es su bajo retardo de propagación. Algunas versiones ECL pueden tener un retardo de propagación tan bajo como 0.5 ns. La disipación de potencia en las compuertas ECL es comparativamente alta y el margen de ruido bajo. Estos dos parámetros imponen una desventaja cuando se elige la ECL sobre las otras familias lógicas. Sin embargo, debido a su bajo retardo de propagación, la ECL ofrece la velocidad más alta entre todas las familias y es la elección final para sistemas muy rápidos.

El circuito más bajo de la CMOS es el inversor por el cual ambas compuertas NAND y NOR pueden construirse. La ventaja especial del CMOS es su disipación de potencia en extremo baja. Bajo condiciones estáticas, la disipación de potencia de la compuerta CMOS es despreciable, con promedios de cerca de 10 nW. Cuando la señal de la compuerta cambia de estado, hay una disipación dinámica de potencia que es proporcional a la frecuencia a la cual se ejerce el circuito. El número que se lista en la tabla es un valor típico de la disipación dinámica de potencia en las compuertas CMOS.

Una desventaja principal de la compuerta CMOS es su alto retardo de propagación. Esto significa que no es práctica para utilizarse en sistemas que requieren operaciones a alta velocidad. Los parámetros característicos de la compuerta CMOS dependen del voltaje de suministro de potencia V_{DD} que se use. La disipación de potencia aumenta conforme aumenta el voltaje de suministro. El retardo de propagación disminuye con el incremento en el voltaje de suministro, y el margen de ruido se estima que es alrededor del 40% del valor del voltaje de suministro.

BIBLIOGRAFIA

1. Boole, G., *An Investigation of the Laws of Thought.* New York: Dover Pub., 1954.
2. Shannon, C. E., "A Symbolic Analysis of Relay and Switching Circuits." *Trans. of the AIEE,* Vol. 57 (1938), 713-23.
3. Huntington, E. V., "Sets of Independent Postulates for the Algebra of Logic." *Trans. Am. Math. Soc.,* Vol. 5 (1904), 288-309.

4. Birkhoff, G., and T. C. Bartee, *Modern Applied Algebra.* New York: McGraw-Hill Book Co., 1970.
5. Birkhoff, G., and S. Maclane, *A Survey of Modern Algebra,* 3a. ed. New York: The Macmillan Co., 1965.
6. Hohn, F. E., *Applied Boolean Algebra,* 2a. ed. New York: The Macmillan Co., 1966.
7. Whitesitt, J. E., *Boolean Algebra and its Applications.* Reading, Mass.: Addison-Wesley Pub. Co., 1961.
8. *The TTL Data Book for Design Engineers.* Dallas, Texas: Texas Instruments Inc., 1976.
9. *MECL Integrated Circuits Data Book.* Phoenix, Ariz.: Motorola Semiconductor Products, Inc., 1972.
10. *RCA Solid State Data Book Series: COS/MOS Digital Integrated Circuits.* Somerville, N. J.: RCA Solid State Div., 1974.

PROBLEMAS

2-1. ¿Cuál de las seis leyes básicas (cierre, asociativa, conmutativa, identidad, inversa y distributiva) se satisface por el par de operadores binarios que se listan a continuación?

+	0	1	2
0	0	0	0
1	0	1	1
2	0	1	2

·	0	1	2
0	0	1	2
1	1	1	2
2	2	2	2

2-2. Muestre que el conjunto de tres elementos $\{0, 1, 2\}$ y los dos operadores binarios + y · como se define en la tabla anterior no es un álgebra booleana. Establezca cuál de los postulados de Huntington no se satisface.

2-3. Demuestre mediante tablas de verdad la validez de los siguientes teoremas del álgebra booleana.
(a) Las leyes asociativas.
(b) Los teoremas de De Morgan para tres variables.
(c) La ley distributiva + sobre ·

2-4. Repita el problema 2.3 utilizando diagramas de Venn.

2-5. Simplifique las siguientes funciones booleanas a un número mínimo de literales.

(a) $xy + xy'$
(b) $(x + y)(x + y')$
(c) $xyz + x'y + xyz'$
(d) $zx + zx'y$
(e) $(A + B)'(A' + B')'$
(f) $y(wz' + wz) + xy$

2-6. Reduzca las siguientes expresiones booleanas al número requerido de literales.

(a) $ABC + A'B'C + A'BC + ABC' + A'B'C'$ a cinco literales
(b) $BC + AC' + AB + BCD$ a cuatro literales
(c) $[(CD)' + A]' + A + CD + AB$ a tres literales
(d) $(A + C + D)(A + C + D')(A + C' + D)(A + B')$ a cuatro literales

• 2-7. Encuentre el complemento de las siguientes funciones booleanas y redúzcalas a un número mínimo de literales.

(a) $(BC' + A'D)(AB' + CD')$

(b) $B'D + A'BC' + ACD + A'BC$

(c) $[(AB)'A][(AB)'B]$

(d) $AB' + C'D'$

• 2-8. Dadas dos funciones booleanas F_1 y F_2:

(a) Muestre que la función booleana $E = F_1 + F_2$, obtenida al aplicar el operador OR a las dos funciones, contiene la suma de todos los mintérminos en F_1 y F_2.

(b) Muestre que la función booleana $G = F_1F_2$, obtenida al aplicar el operador AND a las dos funciones, contiene los mintérminos comunes tanto a F_1 como F_2.

2-9. Obtenga la tabla de verdad de la función:

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

2-10. Implemente las funciones booleanas simplificadas del problema 2-6 con compuertas lógicas.

2-11. Dada la función booleana:

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

(a) Impleméntela con las compuertas AND, OR y NOT

(b) Impleméntela *sólo* con las compuertas OR y NOT

(c) Impleméntela *sólo* con las compuertas AND y NOT

2-12. Simplifique las funciones T_1 y T_2 a un número mínimo de literales.

A	B	C	T_1	T_2
0	0	0	1	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	0	1

• 2-13. Exprese las siguientes funciones de una suma de mintérminos y un producto de maxtérminos.

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

(b) $F(w, x, y, z) = y'z + wxy' + wxz' + w'x'z$

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

(d) $F(A, B, C) = (A' + B)(B' + C)$

(e) $F(x, y, z) = 1$

(f) $F(x, y, z) = (xy + z)(y + xz)$

2-14. Convierta las siguientes funciones en la otra forma canónica.

(a) $F(x, y, z) = \Sigma(1, 3, 7)$
(b) $F(A, B, C, D) = \Sigma(0, 2, 6, 11, 13, 14)$
(c) $F(x, y, z) = \Pi(0, 3, 6, 7)$
(d) $F(A, B, C, D) = \Pi(0, 1, 2, 3, 4, 6, 12)$

2-15. ¿Cuál es la diferencia entre la forma canónica y la estándar? ¿Cuál forma es preferible cuando se implementa con compuertas una función booleana? ¿Cuál forma se obtiene cuando se lee una función de una tabla de verdad?

2-16. La suma de todos los mintérminos de una función booleana de n variables es 1.
(a) Pruebe la enunciación anterior para $n = 3$.
(b) Sugiera un procedimiento para una prueba general.

2-17. El producto de todos los maxtérminos de una función booleana de n variables es 0.
(a) Pruebe la enunciación anterior para $n = 3$.
(b) Sugiera un procedimiento para una prueba general. ¿Puede usarse el principio de dualidad después de probar (b) del problema 2-16?

2-18. Muestre que la dual de la excluyente OR es igual a su complemento.

2-19. Por la sustitución de la función booleana equivalente a las operaciones binarias como se definen en la Tabla 2-6, muestre que:
(a) Los operadores de inhibición e implicación no son ni conmutativos ni asociativos.
(b) Los operadores excluyente OR y de equivalencia son conmutativos y asociativos.
(c) El operador NAND no es asociativo.
(d) Los operadores NOR y NAND no son distributivos.

• 2-20. Una compuerta de *mayoría* es un circuito digital cuya salida es igual a 1 si la mayoría de las entradas son 1. En otra forma, la salida es 0. Mediante una tabla de verdad, encuentre la función booleana implementada por una compuerta de mayoría de 3 entradas. Simplifique la función.

2-21. Verifique la tabla de verdad para la compuerta excluyente OR de 3 entradas que se lista en la Fig. 2-8(c). Liste todas las ocho combinaciones de x, y y z; evalúe $A = x \oplus y$; después evalúe $F = A \oplus z = x \oplus y \oplus z$.

2-22. La familia lógica TTL SSI existe principalmente en paquetes de 14 clavijas. Dos clavijas se reservan para suministro de potencia y las otras clavijas se utilizan para terminales de entrada y salida. Cuántas compuertas están encapsuladas en un paquete de esta clase si contiene los siguientes tipos de compuertas:
(a) Compuertas excluyente-OR de 2 entradas.
(b) Compuertas AND de 3 entradas.
(c) Compuertas NAND de 4 entradas.
(d) Compuertas NOR de 5 entradas.
(e) Compuertas NAND de 8 entradas.

2-23. Muestre que una compuerta AND de lógica positiva es una compuerta OR de lógica negativa y viceversa.

2-24. Una familia lógica IC tiene compuertas NAND con abanico de salida de 5 y compuertas buffer con abanico de salida de 10. Muestre cómo la señal de salida de una sola compuerta NAND puede aplicarse a otras 50 entradas de compuerta.

Simplificación de las funciones booleanas

3

3-1 METODO DE MAPAS

La forma completa de las compuertas lógicas digitales que implementan una función booleana está relacionada en forma directa con la complejidad de las expresiones algebraicas de las cuales se implementa la función. Aunque la representación en tabla de verdad de una función es única, cuando se expresa en forma algebraica puede aparecer en muchas formas diferentes. Las funciones booleanas pueden simplificarse por medios algebraicos como se expuso en la Sección 2-4. Sin embargo, el procedimiento de minimización es difícil debido a que carece de reglas específicas para predecir cada paso sucesivo en el proceso de manipulación. El método de mapas proporciona un procedimiento simple y directo para minimizar las funciones booleanas. Este método puede considerarse ya sea como una forma gráfica de una tabla de verdad o como una extensión del diagrama de Venn. El método de mapas, que Veitch (1) fue el primero en proponer y que modificó ligeramente Karnaugh (2), también se conoce como el "diagrama de Veitch" o "mapa de Karnaugh".

El mapa es un diagrama compuesto por cuadros. Cada cuadro representa un mintérmino. Ya que cualquier función booleana puede expresarse como una suma de mintérminos, se concluye que una función booleana se reconoce en forma gráfica en el mapa por el área encerrada en los cuadros cuyos mintérminos se incluyen en la función. De hecho, el mapa presenta un diagrama visual de todas las formas posibles en que puede expresarse una función en una manera estándar. Mediante el reconocimiento de diversos patrones, el usuario puede derivar expresiones algebraicas alternas para la misma función, de las cuales él puede seleccionar la más simple. Se supondrá que la expresión algebraica más simple es cualquiera en una suma de productos o producto de sumas que tiene un número mínimo de literales. (Esta expresión no es única necesariamente.)

3-2 MAPAS DE DOS Y TRES VARIABLES

En la Fig. 3-1 se muestra un mapa de dos variables. Hay cuatro mintérminos para dos variables; por tanto, el mapa consta de cuatro cuadros, uno para cada mintérmino. El

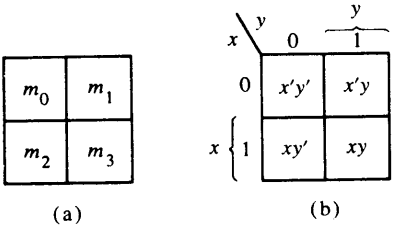


Figura 3-1 Mapa de dos variables.

mapa vuelve a dibujarse en (b) para mostrar las relaciones entre los cuadros y las dos variables. Los números 0 y 1 que se marcan para cada renglón y cada columna designan los valores de las variables x y y, respectivamente. Obsérvese que x aparece como prima en el renglón 0 y sin prima en el renglón 1. En forma similar, y aparece como prima en la columna 0 y sin prima en la columna 1.

Si se marcan los cuadros cuyos mintérminos pertenecen a una función dada, el mapa de dos variables se convierte en otra forma útil para representar cualquiera de las 16 funciones booleanas de dos variables. Como ejemplo, la función xy se muestra en la Fig. 3-2(a). Ya que xy es igual a m_3, se coloca un 1 en el interior del cuadro que pertenece a m_3. En forma semejante, la función $x + y$ se representa en el mapa de la Fig. 3-2(b) por tres cuadros marcados por 1. Estos cuadros se encuentran mediante los mintérminos de la función:

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

Los tres cuadros pudieron haberse determinado mediante la intersección de la variable x en el segundo renglón y la variable y en la segunda columna, la cual encierra el área que pertenece a x o y.

En la Fig. 3-3 se muestra un mapa de tres variables. Hay ocho mintérminos para tres variables binarias. Por lo tanto, un mapa consta de ocho cuadros. Obsérvese que los mintérminos no están arreglados en una secuencia binaria, sino en una secuencia similar al código reflejado que se lista en la Tabla 1-4. Las características de esta secuencia es que sólo un bit cambia de 1 a 0 o de 0 a 1 en la secuencia listada. El mapa que se dibuja en la parte (b) se marca con números en cada renglón y cada columna para mostrar las relaciones entre los cuadros y las tres variables. Por ejemplo, el cuadro asignado a m_5 corresponde al renglón 1 y la columna 01. Cuando estos dos

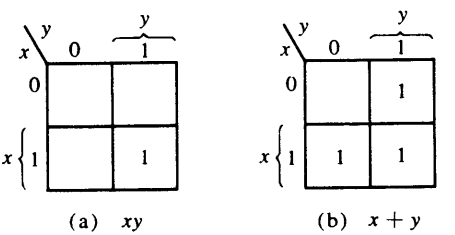


Figura 3-2 Representación de funciones en el mapa.

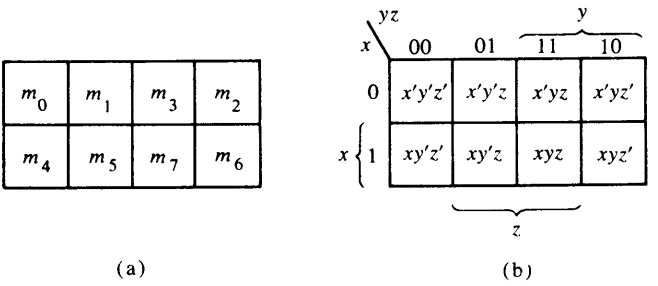


Figura 3-3 Mapa de tres variables.

números se concatenan, dan el número binario 101, cuyo equivalente decimal es 5. Otra forma de ver el cuadro $m_5 = xy'z$ es considerar que está en el renglón marcado x y la columna que pertenece a $y'z$ (columna 01). Obsérvese que hay cuatro cuadros donde cada variable es igual a 1 y cuatro donde cada 1 es igual a 0. La variable aparece sin prima en los cuatro cuadros donde es igual a 1 y con prima en los cuadros donde es igual a 0. Por motivos de comodidad, se escribe la variable con su símbolo de letra bajo los cuatro cuadros donde está sin prima.

Para entender la utilidad del mapa y simplificar funciones booleanas, debe reconocerse la propiedad básica que poseen los cuadros adyacentes. Cualesquiera dos cuadros adyacentes en el mapa difieren sólo en una variable que está con prima en un cuadro y sin prima en el otro. Por ejemplo, m_5 y m_7, caen en dos cuadros adyacentes. La variable y tiene prima en m_5 y no tiene prima en m_7, en tanto que las otras dos variables son las mismas en ambos cuadros. Mediante los postulados del álgebra booleana, se concluye que la suma de dos mintérminos en cuadros adyacentes puede simplificarse a un solo término AND que consta de sólo dos literales. Para aclarar esto, considérese la suma de dos cuadros adyacentes como m_5 y m_7:

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

Aquí los dos cuadros difieren por la variable y, la cual puede eliminarse cuando se forma la suma de los dos mintérminos. Por eso, cualesquiera dos mintérminos en cuadros adyacentes que se unen por el operador OR causarán una eliminación de la variable diferente. El siguiente ejemplo explica el procedimiento para minimizar una función booleana con un mapa.

EJEMPLO 3-1: Simplifique la función booleana:

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

Primero, se marca un 1 en cada cuadro como se necesite para representar la función como se muestra en la Fig. 3-4. Esto puede llevarse a cabo en dos formas: ya sea por la conversión de cada mintérmino en un número binario y marcando entonces un 1 en el cuadro correspondiente o por la obtención de la coincidencia de las variables en cada término.

Por ejemplo, el término $x'yz$ tiene el número binario correspondiente 011 y representa el mintérmino m_3 en el cuadro 011. La segunda forma de reconocer el cuadro es por la coincidencia de las variables x', y y z, la cual se encuentra en el mapa al observar que x' pertenece a los cuatro cuadros en el primer renglón, y pertenece a los cuatro cuadros en las dos columnas de la derecha y z pertenece a los cuatro cuadros en las dos columnas centrales. El área que pertenece a todas las tres literales es el único cuadro en el primer renglón y tercera columna. En forma similar, los otros tres cuadros pertenecientes a la función F están marcados con 1 en el mapa. Por consiguiente, la función está representada por un área que contiene cuatro cuadros, cada uno marcado con un 1, como se muestra en la Fig. 3-4. El paso siguiente es subdividir el área dada en cuadros adyacentes. Esto se indica en el mapa por dos rectángulos, cada uno encierra dos 1. El rectángulo superior de la derecha representa el área encerrada por $x'y$; el inferior de la izquierda, el área encerrada por xy'. La suma de estos dos términos da la respuesta:

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

A continuación considérense los dos cuadros etiquetados m_0 y m_2 en la Fig. 3-3(a) o $x'y'z'$ y $x'yz'$ en la Fig. 3-3(b). Estos dos mintérminos también difieren por una variable y, y su suma puede simplificarse a una expresión de dos literales:

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

En consecuencia, debe modificarse la definición de cuadros adyacentes para incluir éste y otros casos similares. Esto se hace considerando el mapa como si estuviera dibujado en una superficie donde las orillas derecha e izquierda se tocan una con otra para formar cuadros adyacentes.

EJEMPLO 3-2: Simplifique la función booleana:

$$F = x'yz + xy'z' + xyz + xyz'$$

El mapa para esta función se muestra en la Fig. 3-5. Hay cuatro cuadros marcados con 1, uno para cada mintérmino de la función. Se combinan

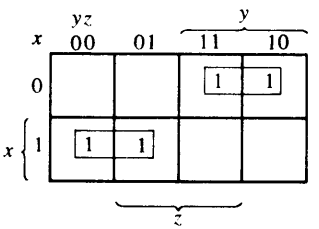


Figura 3-4 Mapa para el ejemplo 3-1; $x'yz + x'yz' + xy'z' + xy'z = x'y + xy'$.

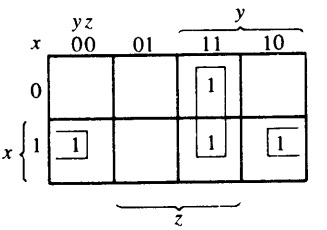


Figura 3-5 Mapa para el ejemplo 3-2; $x'yz + xy'z' + xyz + xyz' = yz + xz'$.

dos cuadros adyacentes en la tercera columna para dar un término de dos literales yz. Los dos cuadros restantes con 1 también son adyacentes por la nueva definición y se muestran en el diagrama dentro de medios rectángulos. Estos dos cuadros, cuando se combinan dan el término de dos literales xz'. La función simplificada llega a ser:

$$F = yz + xz'$$

Considérese ahora cualquier combinación de cuatro cuadros adyacentes en el mapa de tres variables. Cualquiera de dichas combinaciones representa la aplicación del operador OR a cuatro mintérminos adyacentes y resulta en una expresión de sólo una literal. Como un ejemplo, la suma de los cuatro mintérminos adyacentes m_0, m_2, m_4 y m_6 se reduce a la única literal z' como se muestra:

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

EJEMPLO 3-3: Simplifique la función booleana:

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

El mapa que simplifica esta función se muestra en la Fig. 3-6. Algunos de los términos de la función tienen menos de tres literales y se representan en el mapa por más de un cuadro. Por ejemplo, para encontrar los cuadros correspondientes a $A'C$, se forma la coincidencia de A' (primer renglón) y C (dos columnas centrales) y se obtienen los cuadros 001 y 011. Obsérvese que cuando se marcan números 1 en los cuadros, es

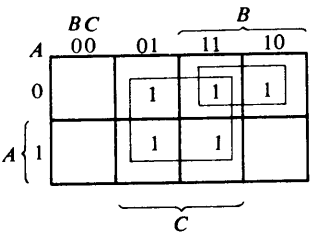


Figura 3-6 Mapa para el ejemplo 3-3; $A'C + A'B + AB'C + BC = C + A'B$.

posible encontrar un 1 ya colocado ahí por un término precedente. En este ejemplo, el segundo término $A'B$ tiene números 1 en los cuadros 011 y 010, pero el cuadro 011 es común al primer término $A'C$ y sólo un 1 está marcado en él. En este ejemplo, la función tiene cinco mintérminos, como se indica por los cinco cuadros marcados con números 1. Se simplifica por la combinación de cuatro cuadros en el centro para dar la literal C. El único cuadro restante marcado con un 1 en 010 se combina con un cuadro adyacente que ya se ha usado una vez. Esto se permite y es inclusive deseable ya que la combinación de los dos cuadros da el término $A'B$, en tanto que el único mintérmino representado por el cuadro da el término de tres variables $A'BC'$. La función simplificada es:

$$F = C + A'B$$

EJEMPLO 3-4: Simplifique la función booleana.

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

Aquí se dan los mintérminos por sus números decimales. Los cuadros correspondientes están marcados con números 1 como se muestra en la Fig. 3-7. Mediante el mapa se obtiene la función simplificada:

$$F = z' + xy'$$

3-3 MAPA DE CUATRO VARIABLES

El mapa para las funciones booleanas de cuatro variables binarias se muestra en la Fig. 3-8. En (a) se listan los 16 mintérminos y los cuadros asignados a cada uno. En (b) el mapa vuelve a dibujarse para mostrar las relaciones con las cuatro variables. Los renglones y columnas se numeran en una secuencia de código reflejado, con sólo un dígito cambiando de valor entre dos renglones adyacentes o columnas. El mintérmino que corresponde a cada cuadro puede obtenerse por la concatenación del número de renglón con el número de columna. Por ejemplo, los números del tercer renglón (11) y

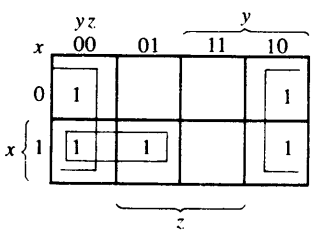


Figura 3-7 $f(x, y, z) = \Sigma(0, 2, 4, 5, 6) = z' + xy'$.

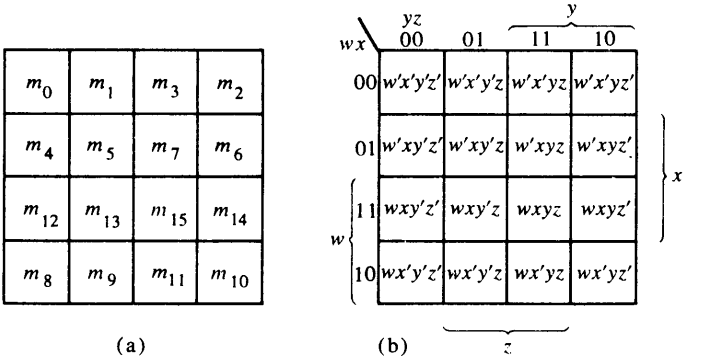


Figura 3-8 Mapa de cuatro variables.

de la segunda columna (01), cuando se concatenan, dan el número binario 1101, el equivalente binario del decimal 13. Por tanto, el cuadro en el tercer renglón y la segunda columna representa el mintérmino m_{13}.

La minimización por mapa de las funciones booleanas de cuatro variables es similar al método que se utiliza para minimizar las funciones de tres variables. Se definen cuadros adyacentes para que sean cuadros juntos entre sí. Además, se considera que el mapa cae en una superficie en las orillas superior e inferior, al igual que en las orillas derecha e izquierda, tocándose uno a otro para formar cuadros adyacentes. Por ejemplo, m_0 y m_2 forman cuadros adyacentes, como sucede con m_3 y m_{11}. La combinación de cuadros adyacentes que es útil durante el proceso de simplificación se determina con facilidad por la inspección del mapa de cuatro variables:

Un cuadro representa un mintérmino, dando un término de cuatro literales.

Dos cuadros adyacentes representan un término de tres literales.

Cuatro cuadros adyacentes representan un término de dos literales.

Ocho cuadros adyacentes representan un término de una literal.

Dieciséis cuadros adyacentes representan la función igual a 1.

Ninguna otra combinación de cuadros puede simplificar la función. Los dos ejemplos siguientes muestran el procedimiento que se usa para simplificar funciones booleanas de cuatro variables.

EJEMPLO 3-5: Simplifique la función booleana:

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

Ya que la función tiene cuatro variables, debe usarse un mapa de cuatro variables. Los mintérminos que se listan en la suma se marcan con números 1 en el mapa de la Fig. 3-9. Ocho cuadros adyacentes marcados con números 1 pueden combinarse para formar un término de una

literal y'. Los tres 1 restantes a la derecha no pueden combinarse juntos para dar un término simplificado. Deben combinarse como dos o cuatro cuadros adyacentes. Mientras mayor sea el número de cuadros combinados, menor será el número de literales en el término. En este ejemplo, los dos 1 de la parte superior a la derecha se combinan con los dos 1 de la parte superior a la izquierda para dar el término $w'z'$. Obsérvese que se permite usar el mismo cuadro más de una vez. Ahora queda un cuadro marcado con 1 en el tercer renglón y cuarta columna (cuadro 1110). En lugar de tomar este cuadro sólo (lo cual daría un término de cuatro literales), se combina con cuadros que ya se han empleado para formar una área de cuatro cuadros adyacentes. Estos cuadros comprenden los dos renglones centrales y dos columnas en los extremos, dando el término xz'. La función simplificada es:

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

EJEMPLO 3-6: Simplifique la función booleana:

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

El área en el mapa cubierta por esta función consta de cuadros marcados con 1 en la Fig. 3-10. Esta función tiene cuatro variables y, como se expresa, consta de tres términos, cada uno con tres literales, y un término de cuatro literales. Cada término de tres literales se representa en el mapa por dos cuadros. Por ejemplo, $A'B'C'$ se representa en los cuadros 0000 y 0001. La función puede simplificarse en el mapa tomando los 1 en las cuatro esquinas para dar el término $B'D'$. Esto es posible debido a que estos cuatro cuadros son adyacentes cuando el mapa se dibuja en una superficie, con las orillas superior e inferior o de izquierda a derecha tocándose una a otra. Los dos 1 del lado izquierdo del renglón

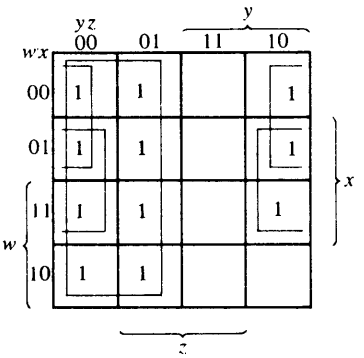


Figura 3-9 Mapa para el ejemplo 3-5; $F(w, x, y, z) = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14) = y' + w'z' + xz'$.

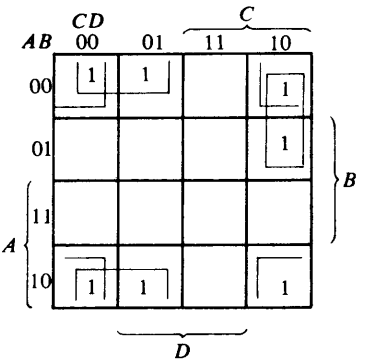


Figura 3-10 Mapa para el ejemplo 3-6; $A'B'C' + B'CD' + A'BCD' + AB'C' = B'D' + B'C' + A'CD'$.

superior se combinan con los dos 1 en el renglón inferior para dar el término $B'C'$. Los restantes 1 pueden combinarse en un área de dos cuadros para dar el término $A'CD'$. La función simplificada es:

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

3-4 MAPAS DE CINCO Y SEIS VARIABLES

Los mapas de más de cuatro variables no son de uso tan simple. El número de cuadros se vuelve en exceso grande y la geometría para combinar cuadros adyacentes se vuelve más complicada. El número de cuadros siempre es igual al número de mintérminos. Para mapas de cinco variables, se necesitan 32 cuadros; para mapas de seis variables se requieren 64 cuadros. Los mapas con siete o más variables necesitan tantos cuadros que no es práctico usarlos. Los mapas de cinco o seis variables se muestran en las Figs. 3-11 y 3-12, respectivamente. Los renglones y columnas se numeran en una secuencia de código reflejado; el mintérmino asignado a cada cuadro se lee mediante esos números. En esta forma, el cuadro en el tercer renglón (11) y la segunda columna (001), en el mapa de cinco variables, es el número 11001, el equivalente del decimal 25. Por tanto, este cuadro representa el mintérmino m_{25}. El símbolo de letra de cada variable se marca junto a los cuadros donde el valor correspondiente de bit del número de código reflejado es un 1. Por ejemplo, en el mapa de cinco variables, la variable A es un 1 en los últimos dos renglones; B es un 1 en los dos renglones centrales. Los números reflejados en las columnas muestran las variables C con un 1 en las cuatro columnas más a la derecha, la variable D con un 1 en las cuatro columnas centrales y los 1 para la variable E no son físicamente adyacentes pero se dividen en dos partes. La asignación de variables en el mapa de seis variables se determina de manera semejante.

La definición de cuadros adyacentes para los mapas en las Figs. 3-11 y 3-12 debe modificarse de nuevo para tomar en cuenta el hecho de que algunas variables están divididas en dos partes. Debe considerarse que el mapa de cinco variables consta de

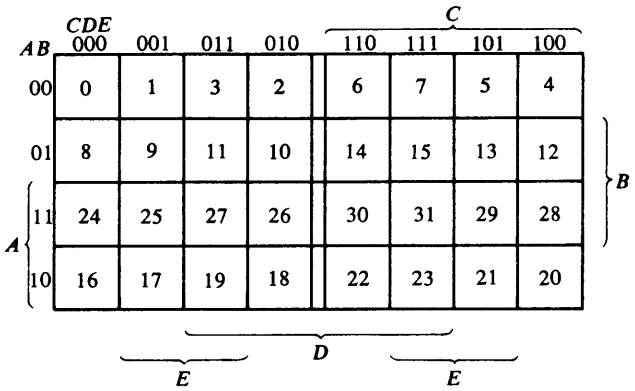


Figura 3-11 Mapa de cinco variables.

dos mapas de cuatro variables y, que el mapa de seis variables consta de cuatro mapas de cuatro variables. Cada uno de estos mapas de cuatro variables se reconoce por las líneas dobles en el centro del mapa; cada uno tiene la adyacencia previamente definida cuando se toma de manera individual. Además, la doble línea del centro debe considerarse como el centro de un libro, como si cada mitad del mapa fuera una

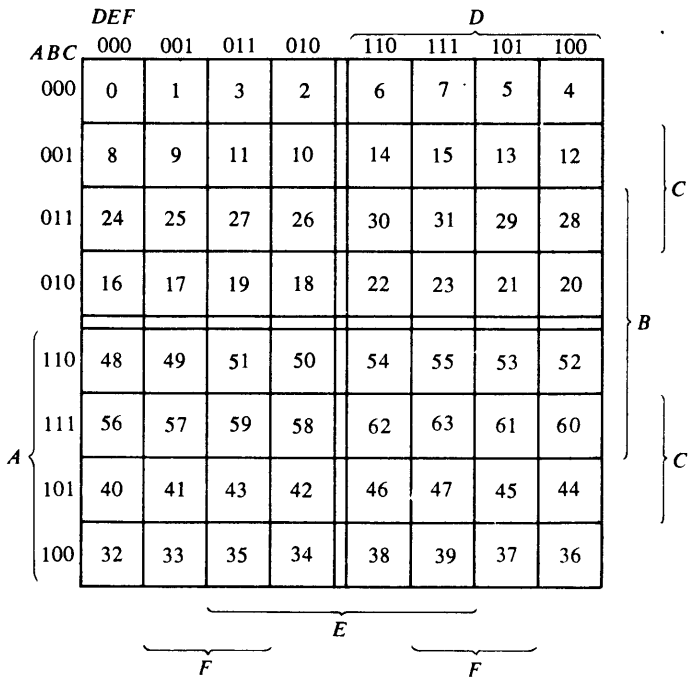


Figura 3-12 Mapa de seis variables.

página. Cuando se cierra el libro, dos cuadros adyacentes caen uno sobre otro. En otras palabras, la línea doble del centro es como un espejo con cada cuadro que es adyacente, no sólo a sus cuatro cuadros vecinos, sino también a su imagen de espejo. Por ejemplo, el mintérminos 31 en el mapa de cinco variables es adyacente a los mintérminos 30, 15, 29, 23 *y* 27. El mismo mintérmino en el mapa de seis variables es adyacente a todos esos mintérminos más el mintérmino 63.

Mediante inspección, y tomando en cuenta la nueva definición de cuadros adyacentes, es posible mostrar que cualesquiera cuadros adyacentes 2^k para $k = 0, 1, 2, \ldots, n$, en un mapa de n variables, representarán un área que da un término de $n - k$ literales. Para que la enunciación anterior tenga algún significado, n debe ser mayor que k. Cuando $n = k$, el área entera del mapa está combinada para dar la función de identidad. La Tabla 3-1 muestra la relación entre el número de cuadros adyacentes con el número de literales en el término. Por ejemplo, ocho cuadros adyacentes se combinan en un área en el mapa de cinco variables para dar un término de dos literales.

EJEMPLO 3-7: Simplifique la función booleana:

$$F(A, B, C, D, E) = \Sigma(0, 2, 4, 6, 9, 11, 13, 15, 17, 21, 25, 27, 29, 31)$$

El mapa de cinco variables de esta función se muestra en la Fig. 3-13. Cada mintérmino se convierte en su número binario equivalente y los 1 se marcan en sus cuadros correspondientes. Ahora es necesario encontrar combinaciones de cuadros adyacentes que resulten en el área más grande posible. Los cuatro cuadros en el centro de la mitad del mapa a la derecha se reflejan a través de la línea doble y se combinan con los cuatro cuadros en el centro del mapa de la mitad izquierda, para dar ocho cuadros adyacentes disponibles equivalentes al término *BE*. Los

TABLA 3-1 La relación entre el número de cuadros adyacentes y el número de literales en el término

	Número de cuadros adyacentes	Número de literales de un término en un mapa de n variables					
k	2^k	$n = 2$	$n = 3$	$n = 4$	$n = 5$	$n = 6$	$n = 7$
0	1	2	3	4	5	6	7
1	2	1	2	3	4	5	6
2	4	0	1	2	3	4	5
3	8		0	1	2	3	4
4	16			0	1	2	3
5	32				0	1	2
6	64					0	1

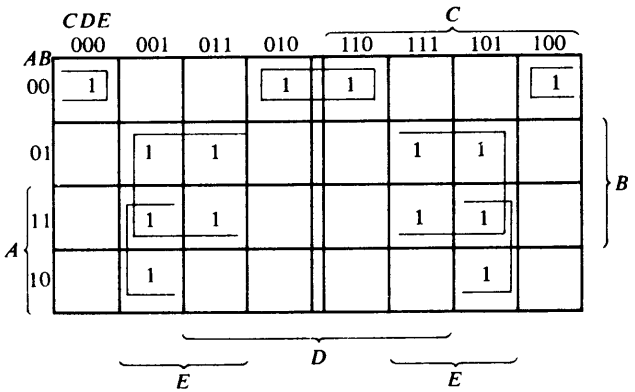


Figura 3-13 Mapa para el ejemplo 3-7; $F(A, B, C, D, E) = \Sigma(0, 2, 4, 6, 9, 11, 13, 15, 17, 21, 25, 27, 29, 31) = BE + AD' E + A' B' E'$.

dos 1 en el renglón inferior son reflejo uno de otro sobre la doble línea del centro. Por la combinación de ellos con los otros dos cuadros adyacentes, se obtiene el término $AD'E$. Los cuatro 1 en el renglón superior son todos adyacentes y pueden combinarse para dar el término $A'B'E'$. Todos los 1 están incluidos ahora. La función simplificada es:

$$F = BE + AD'E + A'B'E'$$

3-5 SIMPLIFICACION DE PRODUCTOS DE SUMA

Las funciones booleanas minimizadas derivadas del mapa en todos los ejemplos previos se expresaron en la forma de suma de productos. Con una pequeña modificación, puede obtenerse la forma de producto de sumas.

El procedimiento para obtener una función minimizada en producto de sumas es una consecuencia de las propiedades básicas de las funciones booleanas. Los 1 ubicados en los cuadros del mapa representan los mintérminos de la función. Los mintérminos que no se incluyen en la función denotan el complemento de la función. De esto puede verse que el complemento de una función está representado en el mapa por los cuadros no marcados por 1. Si se marcan los cuadros vacíos con 0 y se combinan en cuadros válidos adyacentes, se obtiene una expresión simplificada del complemento de la función, esto es, de F'. El complemento de F' resulta de nuevo en la función F. Debido al teorema generalizado de De Morgan, la función así obtenida automáticamente está en la forma de producto de sumas. La mejor manera para mostrar esto es con un ejemplo.

EJEMPLO 3-8: Simplifique la siguiente función booleana en (a) suma de productos y (b) producto de sumas.

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

Los 1 marcados en el mapa de la Fig. 3-14 representan todos los mintérminos de la función. Los cuadros marcados con 0 representan los mintérminos no incluidos en F y, en consecuencia, denotan el complemento de F. La combinación de los cuadros con 1 da la función simplificada en suma de productos:

$$\text{(a)} \qquad F = B'D' + B'C' + A'C'D$$

Si los cuadros marcados con 0 se combinan, como se muestra en el diagrama, se obtiene la función complementada en forma simple:

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

Mediante la aplicación del teorema de De Morgan (se toma la dual y se complementa cada literal como se describió en la Sección 2-4), se obtiene la función simplificada en producto de sumas:

$$\text{(b)} \quad F = (A' + B')(C' + D')(B' + D)$$

La implementación de las expresiones simplificadas obtenidas en el Ejemplo 3-8 se muestra en la Fig. 3-15. La expresión de la suma de productos se implementa en (a) con un grupo de compuertas AND, una para cada término AND. Las salidas de las compuertas AND se conectan a la entrada de una sola compuerta OR. La misma función se implementa en (b) en la forma de su producto de sumas con un grupo de compuertas OR, una para cada término OR. Las salidas de las compuertas OR se conectan con las entradas de una sola compuerta AND. En cada caso, se supone que las variables de entrada están disponibles en forma directa en su complemento, de

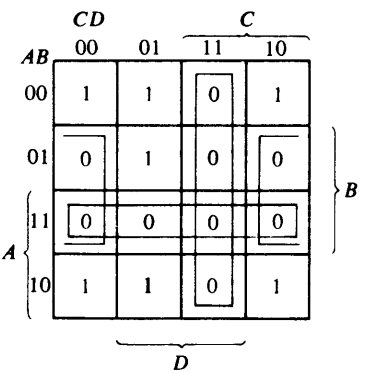
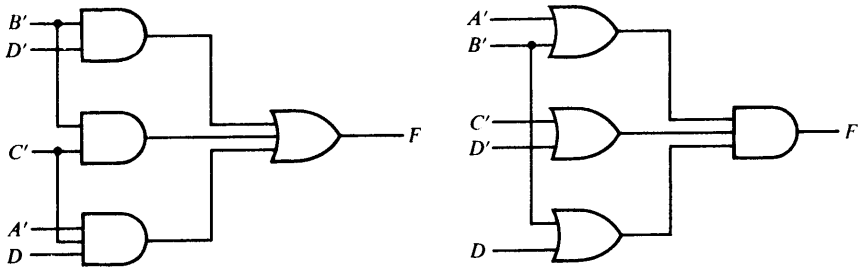


Figura 3-14 Mapa para el ejemplo 3-8; $F(A, B, C, D) = \Sigma(0, 1, 2, 5, 8, 9, 10) = B'D' + B'C' + A'C'D = (A' + B')(C' + D')(B' + D)$.



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

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

modo que no son necesarios inversores. El patrón de configuración establecido en la Fig. 3-15 es la forma general mediante la cual cualquier función booleana se implementan cuando se expresa en una de las formas estándar. Las compuertas AND se conectan a una sola compuerta OR cuando se requiere la suma de productos; las compuertas OR se conectan a una sola compuerta AND cuando se necesita un producto de sumas. Cualquier configuración forma dos niveles de compuertas. Así que, la implementación de una función en una forma estándar se dice que es una implementación de dos niveles.

En el ejemplo 3-8 se mostró el procedimiento para obtener la simplificación cuando la función original estaba expresada en la forma canónica de suma de mintérminos. El procedimiento también es válido cuando la función original está expresada en la forma canónica de producto de maxtérminos. Considérese, por ejemplo, la tabla de verdad que define la función F en la Tabla 3-2. Esta función se expresa en suma de mintérminos como:

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

En producto de maxtérminos, se expresa como:

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

Tabla 3-2 Tabla de verdad para la función F

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

En otras palabras, los 1 de la función representan los mintérminos, y los 0 representan los maxtérminos. El mapa para esta función se dibuja en la Fig. 3-16. Puede iniciarse la simplificación de esta función marcando primero los 1 para cada mintérmino en los que la función es un 1. Los cuadros restantes se marcan con 0. Por otra parte, si al inicio está dado el producto de maxtérminos, puede principiarse marcando 0 en los cuadros que se listan en la función; los cuadros restantes se marcan entonces con 1. Una vez que se han marcado los 1 y 0, la función puede simplificarse en cualquiera de las formas estándar. Para la suma de productos, se combinan los 1 para obtener:

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

Para el producto de sumas, se combinan los 0 para obtener la función complementada simplificada:

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

la cual muestra que la función excluyente OR es el complemento de la función de equivalencia (Sección 2-6). Al tomar el complemento de F', se obtiene la función simplificada en producto de sumas:

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

Para darle entrada a una función expresada en producto de sumas en el mapa, se toma el complemento de la función y, por este medio, se encuentran los cuadros que se marcan con números 0. Por ejemplo, la función:

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

puede introducirse en el mapa al tomar primero su complemento:

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

y entonces se marcan números 0 en los cuadros que representan los mintérminos de F'. Los cuadros restantes se marcan con números 1.

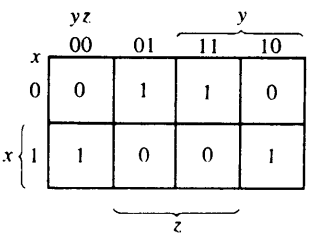


Figura 3-16 Mapa para la función de la Tabla 3-2.

3-6 IMPLEMENTACION CON NOR Y NAND

Los circuitos digitales con más frecuencia se construyen mediante compuertas NAND y NOR que con compuertas AND u OR. Las compuertas NAND y NOR son más fáciles de fabricar con componentes electrónicos y son las compuertas básicas que se utilizan en todas las familias IC de lógica digital. Debido a la preeminencia de las compuertas NAND y NOR en el diseño de circuitos digitales, se han desarrollado reglas y procedimientos para la conversión de las funciones booleanas dadas en términos de AND, OR y NOT en diagramas lógicos equivalentes NAND o NOR. El procedimiento para la implementación de dos niveles se presenta en esta sección. La implementación en niveles múltiples se expone en la Sección 4-7.

Para facilitar la conversión a las lógicas NAND y NOR, es conveniente definir otros dos símbolos gráficos para esas compuertas. En la Fig. 3-17(a) se muestran dos símbolos equivalentes para la compuerta NAND. El símbolo AND invertido se ha definido con anterioridad y consta de un símbolo gráfico AND seguido por un círculo pequeño. En lugar de esto, es posible representar una compuerta NOR con un símbolo gráfico OR precedido por círculos pequeños en todas las entradas. El símbolo OR invertido para la compuerta NAND es consecuencia del teorema de De Morgan y de la convención de que los círculos pequeños denotan la complementación.

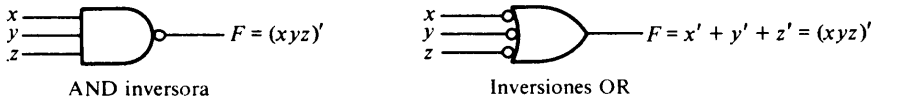
De manera semejante, hay dos símbolos gráficos para la compuerta NOR como se muestra en la Fig. 3-17(b). El símbolo OR invertido es el convencional. La compuerta AND invertida es una alternativa conveniente que utiliza el teorema de De Morgan y la convención de que los círculos pequeños en las entradas denotan la complementación.

Una compuerta de una entrada NAND o NOR se comporta como una inversora. En consecuencia, una compuerta inversora puede dibujarse en tres formas diferentes como se muestra en la Fig. 3-17(c). Los círculos pequeños en todos los símbolos inversores pueden transferirse a la terminal de entrada sin cambiar la lógica de la compuerta.

Debe puntualizarse que los símbolos alternos para las compuertas NAND y NOR pueden dibujarse con triángulos pequeños en todas las terminales de entrada en lugar de círculos. Un triángulo pequeño es un indicador de polaridad de lógica negativa (véase la Sección 2-8 y la Fig. 2-11). Con triángulos pequeños en las terminales de entrada, el símbolo gráfico denota una polaridad de lógica negativa para las entradas, pero la salida de la compuerta (que no tiene un triángulo) debe tener asignada una lógica positiva. A lo largo de este libro se prefiere utilizar la lógica positiva y emplear círculos pequeños cuando sea necesario denotar complementación.

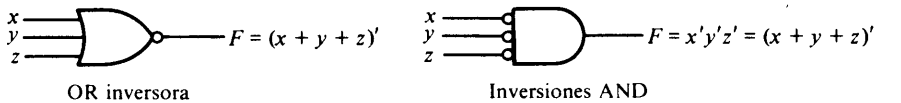
Implementación con NAND

La implementación de una función booleana con compuertas NAND requiere que la función se simplifique en la forma de suma de productos. Para ver la relación entre una expresión de suma de productos y su implementación equivalente NAND, considérense los diagramas lógicos que se dibujan en la Fig. 3-18. Todos los tres diagramas son equivalentes e implementan la función:



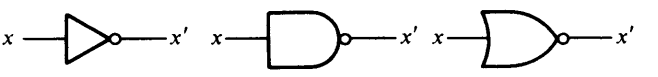
AND inversora Inversiones OR

(a) Dos símbolos gráficos para la compuerta NAND.



OR inversora Inversiones AND

(b) Dos símbolos gráficos para la compuerta NOR.



Buffer inversor AND inversora OR inversora

(c) Tres símbolos gráficos para inversores

Figura 3-17 Símbolos gráficos para las compuertas NAND y OR.

$$F = AB + CD + E$$

La función está implantada en la Fig. 3-18(a) en la forma de suma de productos con compuertas NAND y OR. En (b) las compuertas AND se reemplazan por compuertas NAND y la compuerta OR se sustituye por una compuerta NAND con un símbolo OR invertido. La variable única E se complementa y se aplica a la compuerta OR invertida del segundo nivel. Recuérdese que un círculo pequeño denota la complementación. Por tanto, dos círculos en la misma línea representan complementación doble y ambos pueden eliminarse. El complemento de E pasa a través de un círculo pequeño que complementa la variable otra vez para producir el valor normal de E. La eliminación de los círculos pequeños en las compuertas de la Fig. 3-18(b) produce el circuito en (a). Por tanto, los dos diagramas implementan la misma función y son equivalentes.

En la Fig. 3-18(c), la compuerta NAND de salida se vuelve a dibujar con el símbolo ordinario. La compuerta NAND de una entrada complementa la variable E. Es posible eliminar esta inversora y aplicar E' directamente a la entrada de la compuerta NAND en el segundo nivel. El diagrama en (c) es equivalente al que se muestra en (b), el cual a su vez es equivalente al diagrama en (a). Obsérvese la similitud entre los diagramas en (a) y (c). Las compuertas AND y OR se han cambiado a compuertas NAND, pero se ha incluido una compuerta NAND adicional con la variable única E. Cuando se dibujan diagramas lógicos NAND, el circuito que se muestra ya sea en (b) o (c) es aceptable. Sin embargo, el que se ilustra en (b) representa una relación más directa con la expresión booleana que implementa.

La implementación NAND en la Fig. 3-18(c) puede verificarse en forma algebraica. La función NAND que implanta puede convertirse con facilidad en una forma de suma de productos por la aplicación del teorema de De Morgan:

$$F = [(AB)' \cdot (CD)' \cdot E']' = AB + CD + E$$

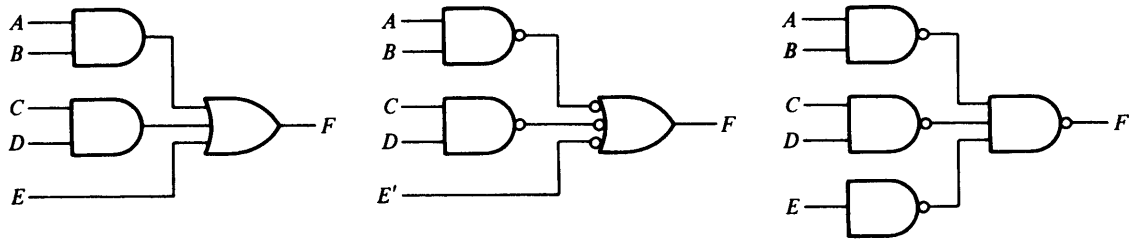


Figura 3-18 Tres formas de implementar $F = AB + CD + E$.

Mediante la transformación que se muestra en la Fig. 3-18, se concluye que una función booleana puede implementarse con dos niveles de compuertas NAND. La regla para obtener el diagrama lógico NAND mediante una función booleana es la siguiente:

1. Simplifíquese la función y exprésese en una suma de productos.
2. Dibújese una compuerta NAND para cada término producto de la función que tiene cuando menos dos literales. Las entradas a cada compuerta NAND son las literales del término. Esto constituye un grupo de compuertas de primer nivel.
3. Dibújese una sola compuerta NAND (utilizando el símbolo gráfico AND invertido o bien OR invertido) en el segundo nivel, con entradas que vienen de las salidas de las compuertas del primer nivel.
4. Un término con una sola literal requiere un inversor en el primer nivel o puede complementarse y aplicarse como una entrada a la compuerta NAND en el segundo nivel.

Antes de aplicar estas reglas a un ejemplo específico, debe mencionarse que hay una segunda forma de implementar una función booleana con compuertas NAND. Recuérdese que si se combinan los 0 en un mapa, se obtiene la expresión simplificada del *complemento* de la función en suma de productos. Entonces, el complemento de la función puede implementarse con dos niveles de compuertas NAND por el uso de las reglas que antes se establecieron. Si se desea la salida normal, es necesario insertar una compuerta NAND de una entrada o inversora para generar el valor verdadero de la variable de salida. Hay ocasiones en que es posible que el diseñador quiera generar el complemento de la función; de modo que puede ser preferible emplear este segundo método.

EJEMPLO 3-9: Impleméntese la siguiente función con compuertas NAND:

$$F(x, y, z) = \Sigma(0, 6)$$

El primer paso es simplificar la función en la forma de suma de productos. Esto se intenta con el mapa que se muestra en la Fig. 3-19(a). Hay sólo dos 1 en el mapa y no puede combinarse. La función simplificada en suma de productos para este ejemplo es:

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

La implementación NAND de dos niveles se muestra en la Fig. 3-19(b). A continuación se tratará de simplificar el complemento de la función en suma de productos. Esto se hace por la combinación de los 0 en el mapa:

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

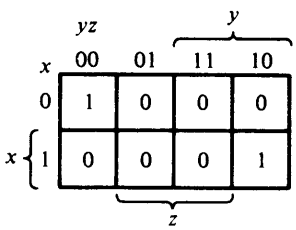
La compuerta NAND de dos niveles para generar F' se muestra en la Fig. 3-19(c). Si se requiere la salida F, es necesario agregar una compuerta NAND de una entrada para invertir la función. Esto da una implementación de tres niveles. En cada caso, se supone que están disponibles las variables de entrada tanto en forma normal como complementaria. Si están disponibles sólo en una forma, puede ser necesario insertar inversores en las entradas, lo que agregaría otro nivel a los circuitos. La compuerta NAND de una entrada asociada con la variable única z puede eliminarse siempre que la entrada se cambie a z'.

Implementación NOR

La función NOR es la dual de la función NAND. Por esta razón, todos los procedimientos y las reglas para la lógica NOR son la dual de los procedimientos y reglas correspondientes que se desarrollaron para la lógica NAND.

La implementación de una función booleana con compuertas NOR requiere que la función se simplifique en la forma de producto de sumas. Una expresión en producto de sumas especifica un grupo de compuertas OR para los términos suma, seguidos por una compuerta AND para obtener el producto. La transformación del diagrama OR-AND al NOR-NOR se indica en la Fig. 3-20. Es similar a la transformación NAND expuesta, excepto que ahora se utiliza la expresión de producto de sumas:

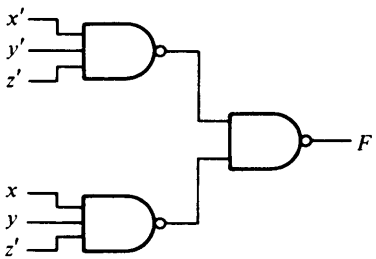
$$F = (A + B)(C + D)E$$



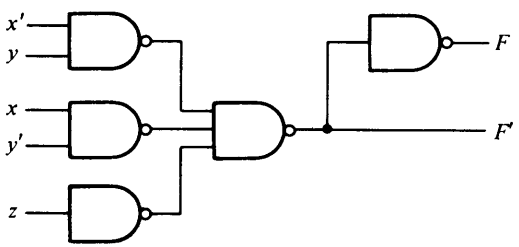
$F = x'y'z' + xyz'$

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

(a) para la simplificación en suma de productos

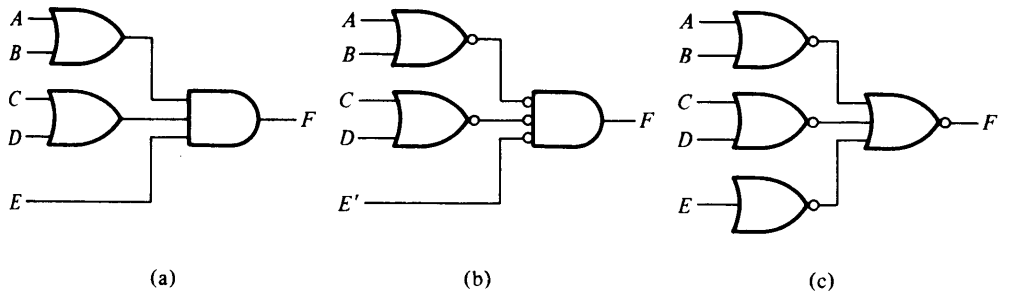


(b) $F = x'y'z' + xyz'$



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

Figura 3-19 Implementación de la función del ejemplo 3-9 con compuertas NAND.



(a) (b) (c)

Figura 3-20 Tres formas de implementar $F = (A + B)(C + D)E$.

La regla para obtener el diagrama lógico NOR de una función booleana puede derivarse de esta transformación. Es similar a la regla NAND de tres pasos, excepto que la expresión simplificada debe ser el producto de sumas y los términos para el primer nivel de compuertas NOR son los términos suma. Un término con una sola literal requiere una compuerta de una entrada NOR o inversora, o bien puede complementarse y aplicarse directamente al segundo nivel de compuerta NOR.

Un segundo modo para implementar una función con compuertas NOR sería usar la expresión para el complemento de la función en producto de sumas. Esto dará una implementación de nivel dos para F' y una implementación de nivel tres si se requiere la salida normal de F.

Para obtener el producto simplificado de sumas de un mapa, es necesario combinar los 0 en el mapa y complementar entonces la función. Para obtener la expresión simplificada de producto de sumas para el complemento de la función, es necesario combinar los 1 en el mapa y complementar entonces la función. El siguiente ejemplo demuestra el procedimiento para la implementación NOR.

EJEMPLO 3-10: Implemente la función del Ejemplo 3-9 con compuertas NOR.

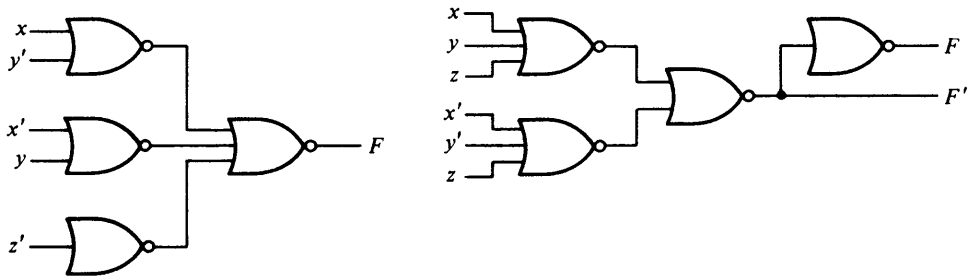
El mapa de esta función se dibuja en la Fig. 3-19(a). Primero, se combinan los 0 en el mapa para obtener dos puntos.

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

Esto es el complemento de la función en suma de productos. Se complementa F' para obtener la función simplificada en producto de sumas como se requiere para la implementación NOR:

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

La implementación de dos niveles con compuertas NOR se muestra en la Fig. 3-21(a). El término con una sola literal z' requiere una compuerta de una entrada NOR o inversora. Esta se elimina y la entrada z se aplica directamente a la entrada de la compuerta de segundo nivel NOR.



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

Figura 3-21 Implementación con compuertas NOR.

Es posible una segunda implementación mediante el complemento de la función en producto de sumas. Para este caso, se combinan primero los 1 en el mapa para obtener:

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

Esta es la expresión simplificada en suma de productos. Se complementa esta función para obtener el complemento de la función en producto de sumas como se requiere para la implementación NOR:

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

La implementación de dos niveles para F' se muestra en la Fig. 3-21(b). La salida F, puede generarse con una inversora en el tercer nivel.

En la Tabla 3-3 se resumen los procedimientos para la implementación NAND o NOR. No debe olvidarse simplificar siempre la función con objeto de reducir el número de compuertas en la implementación. Las formas estándar obtenidas por los procedimientos de simplificación de mapa se aplican de manera directa y son muy útiles cuando se trata con las lógicas NAND o NOR.

TABLA 3-3 Reglas para las implementaciones NAND y NOR

Caso	Función que se simplifica	Forma estándar a usar	Cómo derivarla	Implementada con	Número de niveles para F
(a)	F	Suma de productos	Combínense los 1 en el mapa	NAND	2
(b)	F'	Suma de productos	Combínense los 0 en el mapa	NAND	3
(c)	F	Producto de sumas	Compleméntese F' en (b)	NOR	2
(d)	F'	Producto de sumas	Compleméntese F en (a)	NOR	3

3-7 OTRAS IMPLEMENTACIONES DE NIVEL DOS

Los tipos de compuertas que con más frecuencia se encuentran en los circuitos integrados son NAND y NOR. Por esta razón, las implementaciones de las lógicas NAND y NOR son las más importantes desde el punto de vista práctico. Algunas compuertas NAND o NOR (pero no todas) permiten la posibilidad de una conexión alambrada entre las salidas de dos compuertas para proporcionar una función lógica específica. Este tipo de lógica se conoce como *lógica alambrada*. Por ejemplo, las compuertas de colector abierto TTL NAND, cuando se ligan, realizan la lógica alambrada AND. (La compuerta de colector abierto TTL se muestra en el Capítulo 10, Fig. 10-11.) La lógica alambrada AND realizada con dos compuertas NAND se muestra en la Fig. 3-22(a). La compuerta AND se dibuja con las líneas pasando a través del centro de la compuerta para distinguirla de una compuerta convencional. La compuerta alambrada AND no es una compuerta física, sino sólo un símbolo para designar la función que se obtiene por la conexión alambrada indicada. La función lógica que implanta el circuito en la Fig. 3-22(a) es:

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

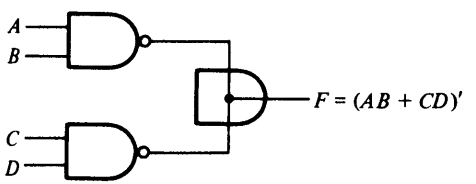
y se denomina función AND-OR-INVERTIDA.

De manera semejante, la salida NOR de las compuertas ECL pueden ligarse para realizar una función alambrada OR. La función lógica que implementa el circuito en la Fig. 3-22(b) es:

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

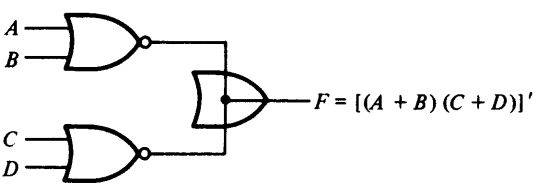
y se denomina función OR-AND-INVERTIDA.

Una compuerta lógica alambrada no produce una compuerta física de segundo nivel, ya que simplemente es una conexión alambrada. Sin embargo, para propósitos de exposición se considerarán los circuitos de la Fig. 3-22 como una implementación de nivel dos. El primer nivel consta de compuertas NAND (o NOR) y el segundo nivel tiene una sola compuerta AND (u OR). La conexión alambrada en el símbolo gráfico se omitirá en las exposiciones subsecuentes.



(a) Compuertas AND alambrada y AND TTL de colector abierto

(AND-OR-INVERSORA)



(b) Compuerta OR alambrada en compurtas ECL

(OR-AND-INVERSORA)

Figura 3-22 Lógica alambrada.

Formas no degeneradas

Desde un punto de vista teórico será instructivo encontrar cuántas combinaciones de compuertas en el nivel dos son posibles. Se consideran cuatro tipos de compuertas: AND, OR, NAND y NOR. Si se asigna un tipo de compuerta para el primer nivel y un tipo para el segundo nivel, se encuentra que hay 16 combinaciones posibles de formas de nivel dos. (El mismo tipo de compuerta puede estar en el primero y segundo niveles, como en la implementación NAND-NAND.) Ocho de estas combinaciones se dice que son formas *degeneradas* ya que degeneran en una sola operación. Esto puede verse mediante un circuito con compuertas AND en el primer nivel y una compuerta AND en el segundo nivel. La salida del circuito es simplemente la función AND de todas las variables de entrada. Las otras ocho formas no degeneradas producen una implementación en suma de productos o producto de sumas. Las ocho formas *no degeneradas* son:

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

La primera compuerta que se lista en cada una de las formas constituye un primer nivel en la implementación. La segunda compuerta que se lista es una sola compuerta colocada en el segundo nivel. Obsérvese que cualesquiera de las dos formas listadas en el mismo renglón son las duales de las otras.

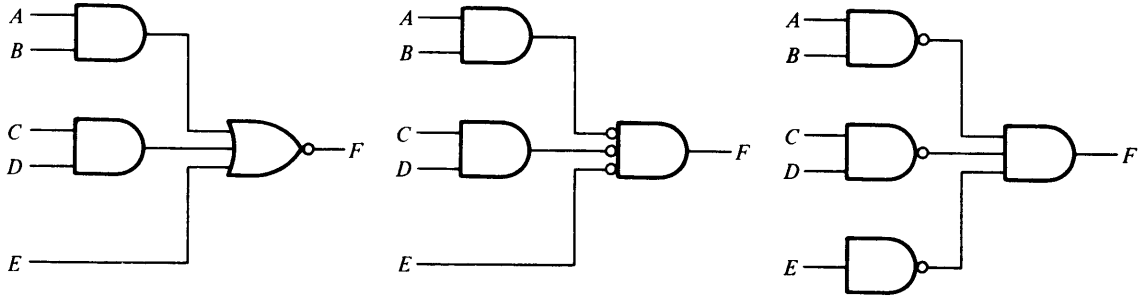
Las formas AND-OR y OR-AND son las formas básicas de nivel dos que se exponen en la Sección 3-5. Las NAND-NAND y NOR-NOR se introdujeron en la Sección 3-6. Las cuatro formas restantes se investigan en esta sección.

Implementación con AND-OR-INVERTIDA

Las dos formas NAND-AND y AND-NOR son formas equivalentes y pueden tratarse juntas. Ambas realizan la función AND-OR-INVERTIDA, como se muestra en la Fig. 3-23. La forma AND-NOR es semejante a la forma AND-OR con una inversión hecha por el círculo pequeño en la salida de la compuerta NOR. Implanta la función:

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

Por el uso del símbolo gráfico alterno para la compuerta NOR, se obtiene el diagrama de la Fig. 3-23(b). Obsérvese que la única variable E no está complementada debido a que el único cambio hecho es en el símbolo gráfico de la compuerta NOR. Ahora se mueven los círculos de la terminal de entrada de la compuerta de segundo nivel a las terminales de salida de las compuertas de primer nivel. Se necesita una inversora para la variable única para mantener el círculo. En forma alterna, la inversora puede quitarse siempre que la entrada E se complemente. El circuito de la Fig. 3-23(c) es una forma NAND-AND y se mostró en la Fig. 3-22 para implementar la función AND-OR-INVERTIDA.



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

Figura 3-23 Circuitos AND-OR-INVERSORES; $F = (AB + CD + E)'$.

Una implementación AND-OR requiere una expresión en suma de productos. La implementación AND-OR-INVERTIDA es similar excepto por la inversión. Por consiguiente, si el complemento de la función se simplifica en suma de productos (por la combinación de los 0 en el mapa), será posible implantar F' con la parte AND-OR de la función. Cuando F' pasa a través de la inversión de salida siempre presente (la parte inversora), generará la salida F de la función. Un ejemplo de la implementación AND-OR-INVERTIDA se mostrará posteriormente.

Implementación con OR-AND-INVERTIDA

Las formas OR-NAND y NOR-OR realizan la función OR-AND-INVERTIDA. Esto se muestra en la Fig. 3-24. La forma OR-NAND se asemeja a la forma OR-AND, a excepción de la inversión hecha por el círculo en la compuerta NAND. Implanta la función:

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

Por el uso del símbolo gráfico alterno para la compuerta NAND, se obtiene el diagrama de la Fig. 3-24(b). El circuito en (c) se obtiene moviendo los círculos pequeños de las entradas de la compuerta de segundo nivel a las salidas de las compuertas de primer nivel. El circuito de la Fig. 3-24(c) es una forma NOR-OR y se mostró en la Fig. 3-22 para implementar la función OR-AND-INVERTIDA.

La implementación OR-AND-INVERTIDA requiere una expresión en producto de sumas. Si el complemento de la función se simplifica en producto de sumas, puede implementarse F' con la parte OR-AND de la función. Cuando F' pasa a través de la parte INVERSORA, se obtiene el complemento de F', o F, en la salida.

Resumen tabular y ejemplo

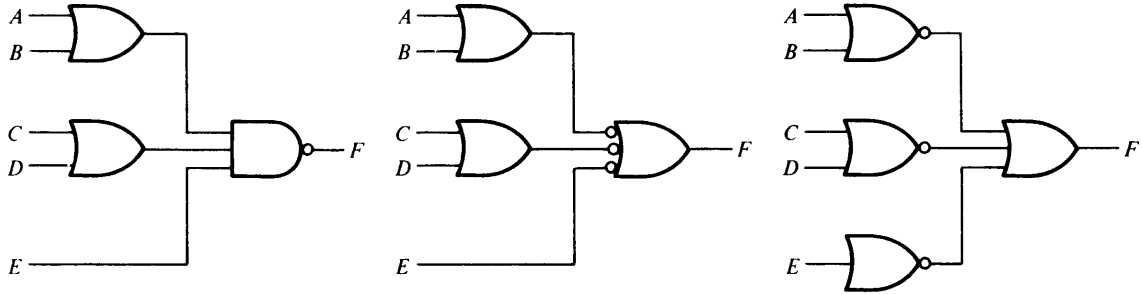
En la Tabla 3-4 se resumen los procedimientos para implementarse una función booleana en cualquiera de las cuatro formas de nivel dos. Debido a la parte INVERSORA en cada caso, es conveniente utilizar la simplificación de F' (el complemento) de la función. Cuando se implanta F' en una de esas formas, se obtiene el complemento de la función en la forma AND-OR o OR-AND. Las cuatro formas de nivel dos invierten esta función, dando una salida que es el complemento de F'. Esta es la salida normal F.

EJEMPLO 3-11: Implante la función de la Fig. 3-19(a) con las cuatro formas de nivel dos que se listan en la Tabla 3-4. El complemento de la función se simplifica en suma de productos por la combinación de los 0 en el mapa:

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

La salida normal para esta función puede expresarse como

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



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

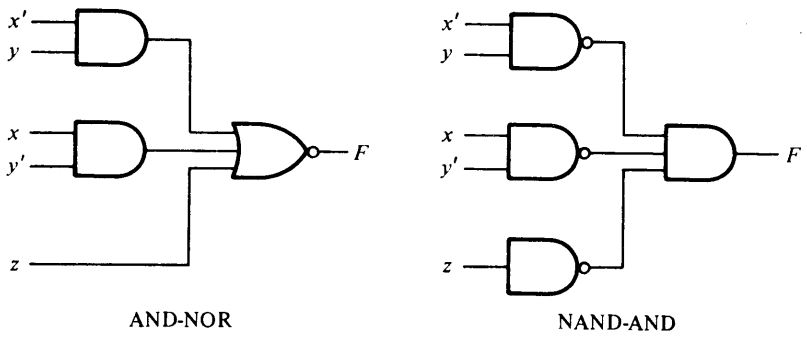
Figura 3-24 Circuitos OR-AND-INVERSORES; $F = [(A + B)(C + D)E]'$.

TABLA 3-4 Implementación con otras formas de dos niveles

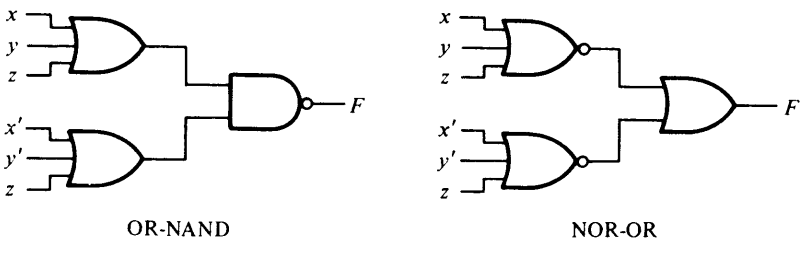
Forma equivalente no degenerada		Implementación de función	Simplificar F' en	Para obtener una salida de
(a)	(b)*			
AND-NOR	NAND-AND	OR-AND-INVERSORA	Suma de productos por la combinación de números 0 en el mapa	F
OR-NAND	NOR-OR	AND-OR-INVERSORA	Producto de sumas por la combinación de números 1 en el mapa, complementar entonces	F

* La forma (b) requiere una compuerta NAND de una entrada o NOR (inversora) para un solo término literal.

la cual está en la forma AND-OR-INVERTIDA. Las implementaciones AND-NOR y NAND-AND se muestran en la Fig. 3-25(a). Obsérvese que se necesita una compuerta de una entrada NAND o inversora en la imple-



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



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

Figura 3-25 Otras implementaciones de dos niveles.

mentación de la NAND-AND, pero no en el caso AND-NOR. La inversora puede eliminarse si se aplica la variable de entrada z' en lugar de z. Las formas OR-AND-INVERTIDA requieren una expresión simplificada del complemento de la función en producto de sumas. Para obtener esta expresión, primero deben combinarse los 1 en el mapa:

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

Entonces se toma el complemento de la función:

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

Ahora la salida normal F puede expresarse en la forma:

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

la cual está en la forma OR-AND-INVERTIDA. Mediante esta expresión puede implantarse la función en las formas OR-NAND y NOR-OR como se muestra en la Fig. 3-25(b).

3-8 CONDICIONES NO IMPORTA

Los 1 y los 0 en el mapa significan la combinación de variables que hacen la función igual a 1 o 0, respectivamente. Las combinaciones por lo común se obtienen de una tabla de verdad donde se listan las condiciones bajo las cuales la función es un 1. La función se supone que es igual a 0 bajo todas las otras condiciones. Esta suposición no siempre es verdadera ya que hay aplicaciones donde ciertas combinaciones de variables de entrada nunca ocurren. Un código decimal de cuatro bits, por ejemplo, tiene seis combinaciones que no se usan. Cualquier circuito digital que utiliza este código opera bajo la suposición de que estas combinaciones no usadas nunca ocurrirán mientras el sistema trabaje apropiadamente. Como resultado, no importa cuál es la salida de la función para esas combinaciones de las variables, ya que está garantizado que nunca ocurrirán. Estas condiciones no importa pueden usarse en un mapa para proporcionar simplificación adicional de la función.

Debe tomarse en cuenta que una combinación no importa no puede marcarse con un 1 en el mapa, debido a que requeriría que la función siempre fuera un 1 para tal combinación de entrada. En forma similar, colocar un 0 en el cuadro requiere que la función sea 0. Para distinguir las condiciones no importa de números 1 y 0, se usará una X.

Cuando se escogen cuadros adyacentes para simplificar la función en el mapa, puede suponerse que las X son ya sea 0 o 1, lo que dé la expresión más simple. Además, en absoluto es necesario usar X si no contribuye a cubrir un área más grande. En cada caso, la elección depende sólo de la simplificación que pueda lograrse.

EJEMPLO 3-12: Simplifique la función booleana:

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

y las condiciones de no importa:

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

Los mintérminos de F son las combinaciones de variables que hacen que la función sea igual a 1. Los mintérminos de d son las combinaciones no importa que se sabe nunca ocurren. La minimización se muestra en la Fig. 3-26. Los mintérminos de F se marcan con números 1, los de d se marcan con X y los cuadros restantes se llenan con números 0. En (a), los 1 y X, se combinan en cualquier forma conveniente de modo que incluyan el número máximo de cuadros adyacentes. No es necesario incluir todas o cualesquiera de las X, sino sólo las que son útiles para simplificar un término. Una combinación que da una función mínima encierra una X y deja dos fuera. Esto resulta en una función simplificada de suma de productos:

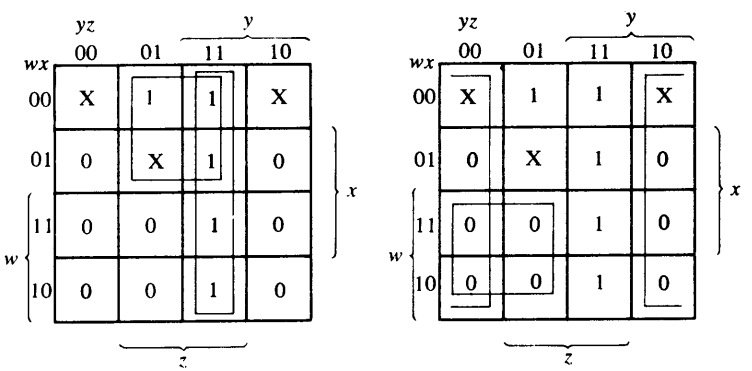
$$F = w'z + yz$$

En (b), los 0 se combinan con cualesquiera X convenientes para simplificar el complemento de la función. Los mejores resultados se obtienen y se encierran las dos X como se muestra. La función complemento se simplifica a:

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

Complementando de nuevo, se obtiene una función simplificada de producto de sumas:

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



(a) Combinación de 1 y X $F = w'z + yz$ (b) Combinación de 0 y X $F = z(w' + y)$

Figura 3-26 Ejemplo con condiciones no importa.

Las dos expresiones que se obtuvieron en el Ejemplo 3-12 dan dos funciones que puede mostrarse que son algebraicamente iguales. Este no es siempre el caso cuando están implicadas condiciones no importa. De hecho, si se usa una X como un 1 cuando se combinan los 1 y otra vez como un 0 cuando se combinan los 0, las dos funciones resultantes no darán respuestas algebraicamente iguales. La selección de las condiciones no importa como un 1 en el primer caso y como un 0 en el segundo resultan en expresiones de mintérminos diferentes y, por lo tanto, funciones diferentes. Esto puede verse en el Ejemplo 3-12. En la solución de este ejemplo, la X que se elije para ser 1 no se escogió para ser un 0. Ahora, si en la Fig. 3-26(a) se elige el término $w'x'$ en lugar de $w'z'$, se sigue obteniendo una función minimizada:

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

Pero no es igual algebraicamente a la obtenida en producto de sumas debido a que las mismas X se utilizaron como números 1 en la primera minimización y como números 0 en la segunda.

Este ejemplo también demuestra que una expresión con el número mínimo de literales no es necesariamente única. Algunas veces el diseñador se enfrenta con una elección entre dos términos con un número igual de literales, y cualquier elección resulta en una expresión minimizada.

3-9 METODO DE TABULACION

El método de mapa de simplificación es conveniente en tanto que el número de variables no exceda cinco o seis. Conforme aumenta el número de variables, el número excesivo de cuadros evita una selección razonable de cuadros adyacentes. La desventaja obvia del mapa es que en esencia es un procedimiento de ensayo y error, que depende de la habilidad del usuario humano para reconocer ciertos patrones. Para funciones de seis o más variables, es difícil tener la seguridad de que se ha hecho la mejor selección.

El método de tabulación supera esta dificultad. Es un procedimiento específico de paso a paso que está garantizado para producir una expresión simplificada en forma estándar para una función. Puede aplicarse a problemas con muchas variables y tiene la ventaja de ser adecuado para computación por máquina. Sin embargo, es bastante tedioso para el uso humano y propenso a errores debido a su proceso rutinario y monótono. El método de tabulación lo formuló por vez primera Quine (3) y lo mejoró posteriormente McCluskey (4). También se conoce como método de Quine-McCluskey.

El método tabular de simplificación consta de dos partes. La primera es encontrar por una búsqueda exhaustiva todos los términos que son candidatos para su inclusión en la función simplificada. Estos términos se denominan implicantes primos. La segunda operación es escoger entre los implicantes primos los que dan una expresión con el menor número de literales.

3-10 DETERMINACION DE LOS IMPLICANTES PRIMOS*

El punto de inicio del método de tabulación es la lista de los mintérminos que especifican la función. La primera operación tabular es encontrar los implicantes primos usando un proceso de comparación. Este proceso compara cada mintérmino con cada uno de los otros mintérminos. Si dos mintérminos difieren sólo en una variable, esta variable se elimina y se encuentra un término con una literal menos. Este proceso se repite para cada mintérmino hasta que se completa la búsqueda exhaustiva. El ciclo del proceso de comparación se repite para los nuevos términos que acaban de encontrarse. Los ciclos tercero y posteriores se continúan hasta que un paso único a través de un ciclo no rinde más eliminación de literales. Los términos restantes y todos los términos que no comparan durante el proceso comprenden los implicantes primos. Este método de tabulación se ilustra en el siguiente ejemplo.

EJEMPLO 3-13: Simplifique la siguiente función booleana utilizando el método de tabulación:

$$F = \Sigma(0, 1, 2, 8, 10, 11, 14, 15)$$

Paso 1: Se hace la representación binaria de grupo de los mintérminos de acuerdo con el número de 1's contenidos, como se muestra en la Tabla 3-5, columna (a). Esto se hace agrupando los mintérminos en cinco secciones separadas por líneas horizontales. La primera sección contiene el número sin números 1 en ella. La segunda sección contiene los números que tienen sólo un 1. La tercera, cuarta y quinta secciones contienen los números binarios con dos, tres y cuatro números 1, respectivamente. Los equivalentes decimales de los mintérminos también se llevan para identificación.

Paso 2: Cualesquiera dos mintérminos que difieran uno del otro sólo por una variable pueden combinarse, y la variable que no compara se elimina. Dos números mintérminos caen en esta categoría si ambos tienen el mismo valor de bit en todas las posiciones excepto una. Los mintérminos en una sección se comparan con los de la siguiente hacia abajo solamente, debido a que dos términos que difieren por más de un bit no pueden comparar. El mintérmino en la primera sección se compara con cada uno de los tres mintérminos en la segunda sección. Si dos números cualesquiera son los mismos en cada posición excepto una, se coloca una marca a la derecha de ambos mintérminos para mostrar que se han utilizado. El término resultante, junto con los equivalentes decimales, se lista en la columna (b) de la tabla. La variable eliminada durante la comparación se indica con un guión en su posición original.

* Esta sección y la siguiente pueden omitirse sin pérdida de continuidad.

TABLA 3-5 Determinación de implicantes primos para el Ejemplo 3-13

(a)		(b)		(c)	
	w x y z		*w x y z*		*w x y z*
0	0 0 0 0 √	0, 1	0 0 0 –	0, 2, 8, 10	– 0 – 0
		0, 2	0 0 – 0 √	0, 8, 2, 10	– 0 – 0
1	0 0 0 1 √	0, 8	– 0 0 0 √	10, 11, 14, 15	1 – 1 –
2	0 0 1 0 √			10, 14, 11, 15	1 – 1 –
8	1 0 0 0 √	2, 10	– 0 1 0 √		
		8, 10	1 0 – 0 √		
10	1 0 1 0 √				
		10, 11	1 0 1 – √		
11	1 0 1 1 √	10, 14	1 – 1 0 √		
14	1 1 1 0 √				
		11, 15	1 – 1 1 √		
15	1 1 1 1 √	14, 15	1 1 1 – √		

En este caso m_0 (0000) combina con m_1 (0001) para formar (000 —). Esta combinación es equivalente a la operación algebraica:

$$m_0 + m_1 = w'x'y'z' + w'x'y'z = w'x'y'$$

El mintérmino m_0 también combina con m_2 para formar (00-0) y con m_8 para formar (—000). El resultado de esta comparación se coloca en la primera sección de la columna (b). Los mintérminos de las secciones dos y tres de la columna (a) se comparan a continuación para producir los términos que se listan en la segunda sección de la columna (b). Todas las otras secciones de (a) se comparan en forma similar y se forman las secciones subsecuentes en (b). Este proceso de comparación exhaustivo resulta en las cuatro secciones de (b).

Paso 3: Los términos de la columna (b) tienen sólo tres variables. Un 1 bajo la variable indica que no tiene prima, un 0 significa que tiene prima, y un guión significa que la variable no se incluye en el término. El proceso de búsqueda y comparación se repite para los términos en la columna (b) para formar los términos de dos variables de la columna (c). De nuevo, los términos en cada sección necesitan compararse sólo si tienen guiones en la misma posición. Observe que el término (000-) no compara con cualquier otro término. Por tanto, no tiene marca de verificación a la derecha. Los equivalentes decimales se escriben en el lado izquierdo de cada entrada para propósitos de identificación. El proceso de comparación debe llevarse a cabo otra vez en la columna (c)

y en las columnas subsecuentes, en tanto se encuentre una comparación apropiada. En este ejemplo, la operación se detiene en la tercera columna.

Paso 4: Los términos que no están marcados en la tabla forman los implicantes primos. En este ejemplo, se tiene el término $w'x'y'$ (000-) en la columna (b), y los términos $x'z'$ (-0-0) y wy (1-1-) en la columna (c). Observe que cada término en la columna (c) aparece dos veces en la tabla, y en tanto el término forma un implicante primo, es inncesario usar el mìsmo término dos veces. La suma de los implicantes primos da una expresión simplificada de la función. Esto se debe a que cada término marcado en la tabla lo ha tomado en cuenta una entrada en un término más simple en una columna subsecuente. Por tanto, las entradas no marcadas (implicantes primos) son los términos que se dejan para formular la función. Para este ejemplo, la suma de los implicantes primos de la función minimizada en suma de productos:

$$F = w'x'y' + x'z' + wy$$

Vale la pena comparar estas respuestas con la que se obtuvo por el método de mapa. En la Fig. 3-27 se muestra el mapa de simplificación de esta funión. Las combinaciones de cuadros adyacentes dan los tres implicantes primos de la función. La suma de estos tres términos es la expresión simplificada en suma de productos.

Es importante puntualizar que el Ejemplo 3-13 se escogió con el propósito de dar la función simplificada a partir de la suma de los implicantes primos. En la mayoría de otros casos, la suma de implicantes primos no necesariamente forma la expresión con el número mínimo de términos. Esto se demuestra en el Ejemplo 3-14.

La manipulación tediosa que debe seguirse cuando se utiliza el método de tabulación se reduce si se compara con la que se hace con números decimales en lugar de binarios. Se mostrará un método en el que se usa la sustracción de números decimales en lugar de comparar e igualar los números binarios. Obsérvese que cada 1 en un número binario representa el coeficiente multiplicado por una potencia de 2.

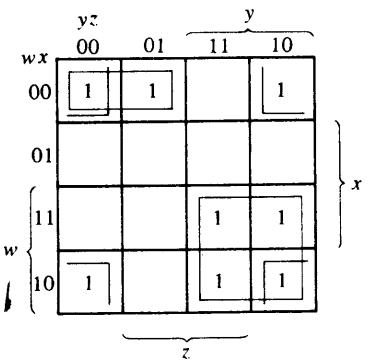


Figura 3-27 Mapa para la función del Ejemplo 3-13; $F = w'x'y' + x'z' + wy$.

Cuando dos mintérminos son los mismos en cada posición excepto uno, el mintérmino con el 1 adicional debe ser mayor que el número de otros mintérminos por una potencia de 2. Por tanto, dos mintérminos pueden combinarse si el número del primer mintérmino difiere en una potencia de 2 respecto a un segundo número más grande en la siguiente sección abajo en la tabla. Se ilustrará este procedimiento repitiendo el Ejemplo 3-13.

Como se muestra en la Tabla 3-6, en la columna (a), los mintérminos se ordenan en secciones como antes, excepto que ahora sólo se listan los equivalentes decimales de los mintérminos. El proceso de comparar mintérminos es como sigue: Inspecciónense cada dos números decimales en secciones adyacentes de la tabla. Si el número en la sección abajo es *mayor* que el número que en la sección arriba por una potencia de 2 (esto es, 1, 2, 4, 8, 16, etc.), verifíquense ambos números para mostrar que se han usado y se escribe abajo en la columna (b). El par de números transferidos a la columna (b) incluye un tercer número entre paréntesis que designa la potencia de 2 por la cual difieren los números. El número entre paréntesis indica la posición del guión en la notación binaria. El resultado de todas las comparaciones de la columna (a) se muestra en la columna (b).

La comparación entre secciones adyacentes en la columna (b) se lleva a cabo de manera semejante, excepto que sólo se comparan los términos con el mismo número entre paréntesis. El par de números en una sección debe diferir por una potencia de 2 del par de números en la sección siguiente. Y los números en la sección siguiente abajo deben ser *mayores* para que tenga lugar la combinación. En la columna (c), se escriben todos los cuatro números decimales con los dos números entre paréntesis designando la posición de los guiones. Una comparación de las Tablas 3-5 y 3-6 puede ser de ayuda para entender las derivaciones en la Tabla 3-6.

Los implicantes primos son los términos no señalados en la tabla. Estos son los mismos de antes, excepto que están dados en notación decimal. Para convertir de la notación decimal a la binaria, conviértanse todos los números decimales en el término en binarios y entonces insértese un guión en las posiciones designadas por los números entre paréntesis. Por eso, 0, 1 (1) se convierte en binario como 0000, 0001; un guión en la primera posición de cualquier número da como resultado (000-). En forma similar, 0, 2, 8, 10 (2, 8), se convierten a la notación binaria de 0000, 0010, 1000 y 1010, y se inserta un guión en las posiciones 2 y 8, para producir (-0-0).

EJEMPLO 3-14: Determine los implicantes primos de la función:

$$F(w, x, y, z) = \Sigma(1, 4, 6, 7, 8, 9, 10, 11, 15)$$

Los números mintérminos se agrupan en secciones como se muestra en la Tabla 3-7, columna (a). El equivalente binario del mintérmino se incluye con el propósito de contar el número de 1. Los números binarios en la primera sección tienen sólo un 1; en la segunda sección, dos 1; etc. Los números mintérminos se comparan por el método decimal y se encuentra un par si el número en la sección de abajo es mayor que el que está en la sección superior. Si el número en la sección inferior es menor

TABLA 3-6 Determinación de implicantes primos del Ejemplo 3-13 en notación decimal

(a)		(b)		(c)	
0	√	0, 1 (1)		0, 2, 8, 10	(2, 8)
		0, 2 (2)	√	0, 2, 8, 10	(2, 8)
1	√	0, 8 (8)	√		
2	√			10, 11, 14, 15	(1, 4)
8	√	2, 10 (8)	√	10, 11, 14, 15	(1, 4)
		8, 10 (2)	√		
10	√				
		10, 11 (1)	√		
11	√	10, 14 (4)	√		
14	√				
		11, 15 (4)	√		
15	√	14, 15 (1)	√		

que el de arriba, no se registra un par aun si los dos números difieren por una potencia de 2. La búsqueda exhaustiva en la columna (a) resulta en los términos de la columna (b), con todos los mintérminos en la columna (a) marcados. Hay sólo dos pares de términos en la columna (b). Cada uno da el mismo término de dos literales registrado en la columna (c). Los implicantes primos constan de todos los términos no marcados en la tabla. La conversión de la notación decimal en la binaria se muestra en la parte inferior de la tabla. Los implicantes primos se encuentra que son $x'y'z$, $w'xz'$, $w'xy$, xyz, wyz y wx'.

La suma de los implicantes primos da una expresión algebraica válida para la función. Sin embargo, esta expresión no necesariamente es la que tiene el número mínimo de términos. Esto puede demostrarse inspeccionando el mapa para la función del Ejemplo 3-14. Como se muestra en la Fig. 3-28, la función minimizada se reconoce que es:

$$F = x'y'z + w'xz' + xyz + wx'$$

la cual consta de la suma de cuatro de los seis implicantes primos que se derivaron en el Ejemplo 3-14. El procedimiento tabular para seleccionar los implicantes primos que da la función minimizada es tema de la siguiente sección.

3-11 SELECCION DE IMPLICANTES PRIMOS

La selección de implicantes primos que forman la función minimizada se realiza mediante una tabla de implicantes primos. En esta tabla, cada implicante primo se

TABLA 3-7 Determinación de implicantes primos para el Ejemplo 3-14

(a)			(b)			(c)
0001	1	√	1, 9	(8)		8, 9, 10, 11 (1, 2)
0100	4	√	4, 6	(2)		8, 9, 10, 11 (1, 2)
1000	8	√	8, 9	(1)	√	
			8, 10	(2)	√	
0110	6	√				
1001	9	√	6, 7	(1)		
1010	10	√	9, 11	(2)	√	
			10, 11	(1)	√	
0111	7	√				
1011	11	√	7, 15	(8)		
			11, 15	(4)		
1111	15	√				

Implicantes primos

Decimal	Binario *w x y z*	Término
1, 9 (8)	− 0 0 1	*x'y'z*
4, 6 (2)	0 1 − 0	*w'xz'*
6, 7 (1)	0 1 1 −	*w'xy*
7, 15 (8)	− 1 1 1	*xyz*
11, 15 (4)	1 − 1 1	*wyz*
8, 9, 10, 11 (1, 2)	1 0 − −	*wx'*

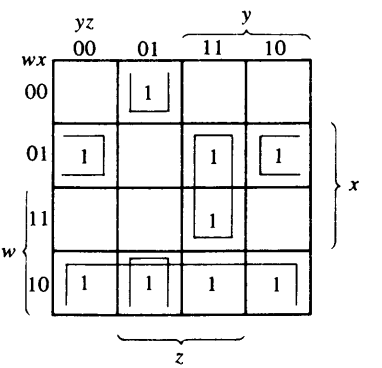


Figura 3-28 Mapa para la función del Ejemplo 3-14; $F = x'y'z + w'xz' + xyz + wx'$.

representa en un renglón y cada mintérmino en una columna. Se marcan cruces en cada renglón para mostrar la composición de mintérminos que hacen los implicantes primos. Un conjunto mínimo de implicantes primos se elije entonces para que cubra todos los mintérminos en la función. Este procedimiento se ilustra en el Ejemplo 3-15.

EJEMPLO 3-15: Minimice la función del Ejemplo 3-14. La tabla de implicantes primos para este ejemplo se muestra en la Tabla 3-8. Hay seis renglones, uno para cada implicante primo (derivado en el Ejemplo 3-14), y nueve columnas, cada una representando un mintérmino de la función. Se marcan cruces en cada renglón para indicar los mintérminos contenidos en el implicante primo de este renglón. Por ejemplo, las dos cruces en el primer renglón indican que los mintérminos 1 y 9 están contenidos en el implicante primo $x'y'z$. Es aconsejable incluir la equivalente decimal del implicante primo en cada renglón, ya que en forma conveniente da los mintérminos contenidos en él. Después de marcar todas las cruces, proceda a seleccionar un número mínimo de implicantes primos.

La tabla completada de implicantes primos se inspecciona para buscar las columnas que contienen sólo una cruz. En este ejemplo, hay cuatro mintérminos cuyas columnas dan una sola cruz: 1, 4, 8 y 10. El mintérmino 1 está cubierto por el implicante primo $x'y'z$, esto es, la selección del implicante primo $x'y'z$ garantiza que el mintérmino 1 esté incluido en la función. De manera semejante, el mintérmino 4 está cubierto por el implicante primo $w'xz'$ y los mintérminos 8 y 10 por el wx'. Los implicantes primos que cubren mintérminos con una sola cruz en su columna se denominan *implicantes primos esenciales*. Para permitir que la expresión simplificada final contenga todos los mintérminos, no se tiene otra alternativa que incluir los implicantes primos esenciales. En la tabla se coloca una marca de verificación junto a los implicantes primos esenciales para indicar que se han seleccionado.

A continuación se verifica cada columna cuyo mintérmino está cubierto por los implicantes primos esenciales que se seleccionaron. Por ejemplo, el implicante primo seleccionado $x'y'z$ cubre los mintérmi-

TABLA 3-8 Tabla de implicantes primos para el Ejemplo 3-15

		1	4	6	7	8	9	10	11	15
$\surd x'y'z$	1, 9	X					X			
$\surd w'xz'$	4, 6		X	X						
$w'xy$	6, 7			X	X					
xyz	7, 15				X					X
wyz	11, 15								X	X
$\surd wx'$	8, 9, 10, 11					X	X	X	X	
		$\surd$	$\surd$	$\surd$		$\surd$	$\surd$	$\surd$	$\surd$	

nos 1 y 9. Se inserta una marca de verificación en la parte inferior de las columnas. En forma similar el implicante primo $w'xz'$ cubre los mintérminos 4 y 6, y wx' cubre los mintérminos 8, 9, 10 y 11. La inspección de la tabla de primo-implicandos muestra que la selección de los implicantes primos esenciales cubre todos los mintérminos de la función, excepto 7 y 15. Estos dos mintérminos deben incluirse por la selección de uno o más primo-implicandos. En este ejemplo, es claro que el implicante primo xyz cubre ambos mintérminos y, por tanto, es el que se selecciona. Asi se ha encontrado el conjunto mínimo de implicante primo cuya suma da la función minimizada requerida:

$$F = x'y'z + w'xz' + wx' + xyz$$

Todas las expresiones simplificadas que se derivan de los ejemplos anteriores están en la forma de suma de productos. El método de tabulación puede adaptarse para dar una expresión simplificada en producto de sumas. Como en el método de mapa, se principia con el complemento de la función tomando los 0 como la lista inicial de mintérminos. Esta lista contiene los mintérminos que no se incluyen en la función original, los cuales son numéricamente iguales a los maxtérminos de la función. El proceso de tabulación se lleva a cabo con los 0 de la función y termina con una expresión simplificada en suma de productos del complemento de la función. Tomando de nuevo el complemento, se obtiene la expresión simplificada en producto de sumas.

Una función con condiciones no importa puede simplificarse por el método de tabulación después de una ligera modificación. Los términos no importa se incluyen en la lista de mintérminos cuando se determinan los implicantes primos. Esto permite la derivación de implicantes primos con el número más bajo de literales. Los términos no importa no se incluyen en la lista de mintérminos cuando la tabla de implicantes primos se establece, porque los términos no importa no tienen que cubrirse por los primo-implicandos seleccionados.

3-12 COMENTARIOS CONCLUYENTES

En este capítulo se introdujeron dos métodos de simplificación de función booleana. El criterio que se tomó para la simplificación fue la minimización del número de literales en las expresiones en suma de productos o producto de sumas. Tanto los métodos de mapa como de tabulación tienen capacidades restringidas, ya que son útiles para simplificar solamente funciones booleanas que se expresen en las formas estándar. Aunque es una desventaja de los métodos, no es muy crítica. La mayoría de las aplicaciones requieren las formas estándar sobre cualquier otra forma. Mediante la Fig. 3-15 se vio que el implante en compuerta de las expresiones en formas estándar consta de no más de dos niveles de compuertas. Las expresiones que no están en la forma estándar se implementan con más de dos niveles. Humphrey (5) muestra una extensión del método de mapa que produce expresiones simplificadas de nivel múltiple.

Debe reconocerse que la secuencia de código reflejado elegida para los mapas no es única. Es posible dibujar un mapa y asignar una secuencia de código reflejado binario a los renglones y columnas diferentes de la secuencia que aquí se emplea. En tanto que la secuencia binaria elegida produzca un cambio sólo de un bit entre cuadros adyacentes, producirá un mapa válido y útil.

En la Fig. 3-29 se muestran dos versiones alternas de mapas de tres variables que con frecuencia se encuentran en la literatura referente a la lógica digital. Los números mintérminos se escriben en cada cuadro para referencia. En (a), la asignación de variables a los renglones y columnas es diferente a la que se utiliza en este libro. En (b), el mapa se ha girado a una posición vertical. La asignación del número mintérmino en todos los mapas permanece en el orden *xyz*. Por ejemplo, el cuadro para el mintérmino 6 se encuentra por la asignación de las variables ordenadas del número binario $xyz = 110$. El cuadro de este mintérmino se encuentra en (a) la columna marcada $xy = 11$ y el renglón con $z = 0$. El cuadro correspondiente en (b) pertenece a la columna marcada con $x = 1$ y el renglón con $yz = 10$. El procedimiento de simplificación con estos mapas es exactamente el mismo que se describe en este capítulo, excepto, por supuesto, la variación en la asignación en mintérmino y variable.

En la Fig. 3-30 se muestran otras dos versiones del mapa de cuatro variables. El mapa en (a) tiene gran preferencia y con mucha frecuencia se utiliza en la literatura. De nuevo, aquí la diferencia es ligera y se manifiesta por un simple intercambio de asignación de variables de renglones a columnas y viceversa. El mapa en (b) es el diagrama original de Veitch (1) el cual modificó Karnaugh (2) a la forma que se muestra en (a). De nuevo, los procedimientos de simplificación no cambian cuando estos mapas se usan en lugar del que se emplea en este libro. También hay variaciones de los mapas de cinco y seis variables. En cualquier caso, cualquier mapa de diferente apariencia respecto al que se utiliza en este libro, o que reciba un nombre distinto, se reconoce en forma simple como una variación de la asignación de mintérminos a los cuadros en el mapa.

Como es evidente a partir de los Ejemplos 3-13 y 3-14, el método de tabulación tiene la desventaja de la ocurrencia inevitable de errores que se dan al tratar de comparar números en listas largas. El método de mapa parece ser preferible pero, para

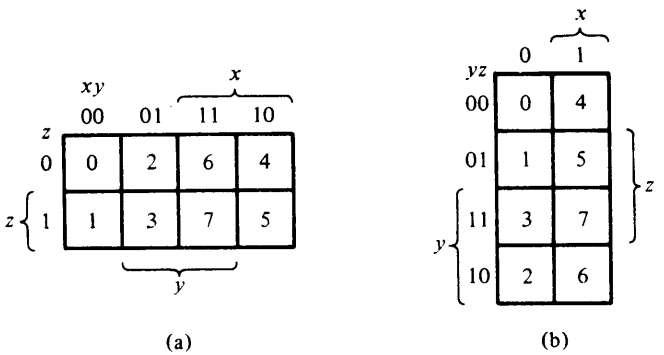


Figura 3-29 Variaciones del mapa de tres variables.

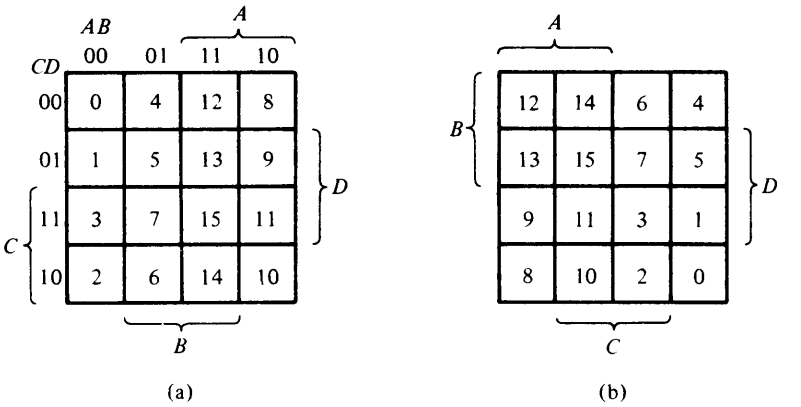


Figura 3-30 Variaciones del mapa de cuatro variables.

más de cinco variables, no puede tenerse la certeza de que se ha encontrado la mejor expresión simplificada. La ventaja real del método de tabulación se basa en el hecho de que consta de procedimientos específicos paso a paso que garantizan una respuesta. Además, este procedimiento formal es adecuado para mecanización por computadora.

En la Sección 3-9 se estableció que el método de tabulación siempre principia con la lista de mintérminos de la función. Si la función no está en esta forma, debe convertirse. En la mayoría de las aplicaciones, la función que va a simplificarse proviene de una tabla de verdad y ya se dispone de la lista de mintérminos. De otra manera, la conversión en mintérminos agrega considerable trabajo de manipulación al problema. Sin embargo, existe una extensión del método de tabulación para encontrar los implicantes primos de expresiones arbitrarias de suma de productos. Véase, por ejemplo, McCluskey (7).

En este capítulo, se han considerado las simplificaciones de funciones con muchas variables de entrada y una sola variable de salida. Sin embargo, algunos circuitos digitales tienen más de una salida. Tales circuitos se describen por un conjunto de funciones booleanas, una para cada variable de salida. Un circuito con salidas múltiples algunas veces puede tener términos comunes entre las diversas funciones que pueden utilizarse para formar compuertas comunes durante la implementación. Esto resulta en una simplificación adicional que no se toma en consideración cuando cada variable se simplifica por separado. Existe una extensión del método de tabulación para circuitos de salidas múltiples (6, 7). Sin embargo, este método es demasiado especializado y muy tedioso para manipulación humana. Es de importancia práctica sólo si el usuario dispone de un programa de computadora basado en este método.

BIBLIOGRAFIA

1. Veitch, E. W., "A Chart Method for Simplifying Truth Functions." *Proc. of the ACM* (*Mayo* 1952), 127-33.

2. Karnaugh, M., "A Map Method for Synthesis of Combinational Logic Circuits." *Trans. AIEE, Comm. and Electronics*, Vol. 72, Parte I (Noviembre 1953), 593-99.

3. Quine, W.V., "The Problem of Simplifying Truth Functions." *Am. Math. Monthly*, Vol. 59, No. 8 (Octubre 1952), 521-31.

4. McCluskey, E.J., Jr., "Minimization of Boolean Functions." *Bell System Tech. J.*, Vol. 35, No. 6 (Noviembre 1956), 1417-44.

5. Humphrey, W. S., Jr., *Switching Circuits with Computer Applications*. New York: McGraw-Hill Book Co., 1958, Cap. 4.

6. Hill, F. J., and G.R. Peterson, *Introduction to Switching Theory and Logical Design*, 3a. ed. New York: John Wiley & Sons, Inc., 1981.

7. McCluskey, E. J., Jr., *Introduction to the Theory of Switching Circuits*. New York: McGraw-Hill Book Co., 1965, Cap. 4.

8. Kohavi, Z., *Switching and Finite Automata Theory*. New York: McGraw-Hill Book Co., 1970.

9. Nagle, H.T. Jr., B.D. Carrol, and J.D. Irwin, *An Introduction to Computer Logic*. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1975.

PROBLEMAS

3-1. Obtenga las expresiones simplificadas en suma de productos para las siguientes funciones booleanas:

(a) $F(x, y, z) = \Sigma(2, 3, 6, 7)$

(b) $F(A, B, C, D) = \Sigma(7, 13, 14, 15)$

(c) $F(A, B, C, D) = \Sigma(4, 6, 7, 15)$

(d) $F(w, x, y, z) = \Sigma(2, 3, 12, 13, 14, 15)$

3-2. Obtenga las expresiones simplificadas en suma de productos para las siguientes funciones booleanas:

(a) $xy + x'y'z' + x'yz'$

(b) $A'B + BC' + B'C'$

(c) $a'b' + bc + a'bc'$

(d) $xy'z + xyz' + x'yz + xyz$

3-3. Obtenga las expresiones simplificadas en suma de productos para las siguientes expresiones booleanas:

(a) $D(A' + B) + B'(C + AD)$

(b) $ABD + A'C'D' + A'B + A'CD' + AB'D'$

(c) $k'lm' + k'm'n + klm'n' + lmn'$

(d) $A'B'C'D' + AC'D' + B'CD' + A'BCD + BC'D$

(e) $x'z + w'xy' + w(x'y + xy')$

3-4. Obtenga las expresiones simplificadas en suma de productos para las siguientes funciones booleanas:

(a) $F(A, B, C, D, E) = \Sigma(0, 1, 4, 5, 16, 17, 21, 25, 29)$

(b) $BDE + B'C'D + CDE + A'B'CE + A'B'C + B'C'D'E'$

(c) $A'B'CE' + A'B'C'D' + B'D'E' + B'CD' + CDE' + BDE'$

3-5. Dada la siguiente tabla de verdad:

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

(a) Exprese F_1 y F_2 en producto de maxtérminos.
(b) Obtenga las funciones simplificadas en suma de productos.
(c) Obtenga las funciones simplificadas en producto de sumas.

3-6. Obtenga las expresiones simplificadas en producto de sumas:

(a) $F(x, y, z) = \Pi(0, 1, 4, 5)$
(b) $F(A, B, C, D) = \Pi(0, 1, 2, 3, 4, 10, 11)$
(c) $F(w, x, y, z) = \Pi(1, 3, 5, 7, 13, 15)$

3-7. Obtenga las expresiones simplificadas en (1) suma de productos y (2) producto de sumas:

(a) $x'z' + y'z' + yz' + xyz$
(b) $(A + B' + D)(A' + B + D)(C + D)(C' + D')$
(c) $(A' + B' + D')(A + B' + C')(A' + B + D')(B + C' + D')$
(d) $(A' + B' + D)(A' + D')(A + B + D')(A + B' + C + D)$
(e) $w'yz' + vw'z' + vw'x + v'wz + v'w'y'z'$

3-8. Dibuje la implementación de compuertas para las funciones booleanas que se obtuvieron en el problema 3-7 utilizando compuertas AND y OR.

3-9. Simplifique cada una de las siguientes funciones e implántelas con compuertas NAND. Dé dos alternativas.

(a) $F_1 = AC' + ACE + ACE' + A'CD' + A'D'E'$
(b) $F_2 = (B' + D')(A' + C' + D)(A + B' + C' + D)(A' + B + C' + D')$

3-10. Repita el problema 3-9 para implementaciones NOR.

3-11. Implemente las siguientes funciones con compuertas NAND. Suponga que están disponibles entradas tanto normal como complementaria.
(a) $BD + BCD + AB'C'D' + A'B'CD'$ con no más de seis compuertas y cada una con tres entradas.
(b) $(AB + A'B')(CD' + C'D)$ con compuertas de dos entradas.

3-12. Implemente las siguientes funciones con compuertas NOR. Suponga que están disponibles entradas tanto normal como complementaria.

(a) $AB' + C'D' + A'CD' + DC'(AB + A'B') + DB(AC' + A'C)$
(b) $AB'CD' + A'BCD' + AB'C'D + A'BC'D$

3-13. Liste las ocho formas degeneradas de dos niveles y muestre que se reducen a una sola operación. Explique cómo las formas degeneradas de dos niveles pueden usarse para extender el abanico de entrada de compuertas.

3-14. Implemente las funciones del problema 3-9 con las siguientes formas de dos niveles: NOR-OR, NAND-AND, OR-NAND y AND-NOR.

3-15. Simplifique la función booleana F en suma de productos usando las condiciones no importa d:

(a) $F = y' + x'z'$
$d = yz + xy$

(b) $F = B'C'D' + BCD' + ABC'D$
$d = B'CD' + A'BC'D'$

3-16. Simplifique la función booleana F utilizando las condiciones no importa d, en (1) suma de productos y (2) producto de sumas:

(a) $F = A'B'D' + A'CD + A'BC$
$d = A'BC'D + ACD + AB'D'$

(b) $F = w'(x'y + x'y' + xyz) + x'z'(y + w)$
$d = w'x(y'z + yz') + wyz$

(c) $F = ACE + A'CD'E' + A'C'DE$
$d = DE' + A'D'E + AD'E'$

(d) $F = B'DE' + A'BE + B'C'E' + A'BC'D'$
$d = BDE' + CD'E'$

3-17. Implemente las siguientes funciones usando las condiciones no importa. Suponga que están disponibles las entradas tanto normal como complementaria.

(a) $F = A'B'C' + AB'D + A'B'CD'$ con no más de dos compuertas NOR.
$d = ABC + AB'D'$

(b) $F = (A + D)(A' + B)(A' + C')$ con no más de tres compuertas NAND.

(c) $F = B'D + B'C + ABCD$ con compuertas NAND.
$d = A'BD + AB'C'D'$

3-18. Implemente la siguiente función ya sea con compuertas NAND o NOR. Use sólo cuatro compuertas. Sólo están disponibles las entradas normales.

$$F = w'xz + w'yz + x'yz' + wxy'z$$
$$d = wyz$$

3-19. La siguiente expresión booleana:

$$BE + B'DE'$$

es una versión simplificada de la expresión:

$$A'BE + BCDE + BC'D'E + A'B'DE' + B'C'DE'$$

¿Aquí hay condiciones no importa? Si es así, ¿cuáles son?

3-20. Dé tres formas posibles de expresar la función:

$$F = A'B'D' + AB'CD' + A'BD + ABC'D$$

con ocho o menos literales.

3-21. Con el uso de mapas, encuentre la forma más simple en suma de productos de la función $F = fg$, en donde f y g están dadas por:

$$f = wxy' + y'z + w'yz' + x'yz'$$
$$g = (w + x + y' + z')(x' + y' + z)(w' + y + z')$$

Sugerencia: Vea el problema 2-8(b).

3-22. Simplifique la función booleana del problema 3-2(a) utilizando el mapa que se define en la Fig. 3-29(a). Repita esto con el mapa en la Fig. 3-30(b).

3-23. Simplifique la función booleana en el problema 3-3(a) usando el mapa que se define en la Fig. 3-30(a). Repita esto con el mapa en la Fig. 3-30(b).

3-24. Simplifique las funciones booleanas siguientes mediante el método de tabulación.

(a) $F(A, B, C, D, E, F, G) = \Sigma(20, 28, 52, 60)$

(b) $F(A, B, C, D, E, F, G) = \Sigma(20, 28, 38, 39, 52, 60, 102, 103, 127)$

(c) $F(A, B, C, D, E, F) = \Sigma(6, 9, 13, 18, 19, 25, 27, 29, 41, 45, 57, 61)$

3-25. Repita el problema 3-6 utilizando el método de tabulación.

3-26. Repita el problema 3-16(c) y (d) usando el método de tabulación.

Lógica combinacional

4

4-1 INTRODUCCION

Los circuitos lógicos para sistemas digitales pueden ser combinacionales o secuenciales. Un circuito combinacional consta de compuertas lógicas cuyas salidas en cualquier momento están determinadas en forma directa por la combinación presente de las entradas sin tomar en cuenta las entradas previas. Un circuito combinacional realiza una operación específica de procesamiento de información, especificada por completo en forma lógica por un conjunto de funciones booleanas. Los circuitos secuenciales emplean elementos de memoria (celdas binarias) además de las compuertas lógicas. Sus salidas son una función de las entradas y el estado de los elementos de memoria. El estado de los elementos de memoria, a su vez, es una función de las entradas previas. Como consecuencia, las salidas de un circuito secuencial dependen no sólo de las entradas presentes, sino también de las entradas del pasado y, el comportamiento del circuito debe especificarse en una secuencia de tiempo de entradas y de estados internos. En el Capítulo 6 se exponen los circuitos secuenciales.

En el Capítulo 1 se aprendió a reconocer los números binarios y los códigos binarios que representan cantidades discretas de información. Estas variables binarias se representan por voltajes eléctricos o alguna otra señal. Las señales pueden manipularse en las compuertas lógicas digitales para realizar las funciones requeridas. En el Capítulo 2 se introdujo el álgebra booleana como una forma para expresar de manera algebraica las funciones lógicas. En el Capítulo 3 se aprendió cómo simplificar las funciones booleanas para lograr la implementación económica de compuertas. El objetivo de este capítulo es usar el conocimiento adquirido en los capítulos previos y formular varios procedimientos sistemáticos de diseño y análisis de los circuitos combinacionales. La solución de algunos ejemplos típicos proporcionará un catálogo útil de funciones elementales importantes para el entendimiento de las computadoras y sistemas digitales.

Un circuito combinacional consta de variables de entrada, compuertas lógicas y variables de salida. Las compuertas lógicas aceptan las señales de las entradas y generan señales a las salidas. Este proceso transforma la información binaria de los datos dados de entrada en los datos requeridos de salida. En forma obvia, tanto los datos de entrada y

salida se representan por señales binarias, esto es, existen en dos valores posibles, uno representa la lógica 1 y el otro la lógica 0. En la Fig. 4-1, se muestra un diagrama de bloques de un circuito. Las n variables binarias de entrada provienen de una fuente externa; las m variables de salida van a un destino externo. En muchas aplicaciones, la fuente y/o destino son registros de almacenamiento (Sección 1-7) localizados ya sea en la proximidad del circuito combinacional o en un dispositivo externo remoto. Por definición, un registro externo no influencia el comportamiento del circuito combinacional ya que, si lo hace, el sistema total se vuelve un circuito secuencial.

Para las n variables de entrada, hay 2^n combinaciones posibles de los valores binarios de entrada. Para cada combinación posible de entrada, hay una y sólo una combinación posible de salida. Un circuito combinacional puede describirse por n funciones booleanas, una para cada variable de salida. Cada función de salida se expresa en términos de las n variables de entrada.

Cada variable de entrada a un circuito combinacional puede tener uno o dos alambres. Cuando está disponible sólo un alambre, puede representar la variable, ya sea en la forma normal (sin prima) o en la forma complementaria (con prima). Ya que una variable en una expresión booleana puede aparecer con prima y/o sin prima, es necesario proporcionar un inversor para cada literal que no está disponible en el alambre de entrada. Por otra parte, una variable de entrada puede aparecer en dos alambres, suministrando las formas tanto normal como complementaria a la entrada del circuito. En este caso, no es necesario incluir inversores para las entradas. El tipo de celdas binarias utilizadas en la mayoría de los sistemas digitales son circuitos flip-flop (Capítulo 6), que tienen salidas para los valores tanto normal como complementario de la variable binaria almacenada. En el trabajo subsecuente, se supondrá que cada variable de entrada aparece en dos alambres, suministrando en forma simultánea valores normal al igual que complementario. Debe tenerse en cuenta que un circuito inversor siempre puede suministrar el complemento de la variable si sólo está disponible un alambre.

4-2 PROCEDIMIENTO DE DISEÑO

El diseño de los circuitos combinacionales surge del planteamiento verbal del problema y termina en un diagrama de circuito lógico, o un conjunto de funciones booleaneas del cual puede obtenerse con facilidad el diagrama lógico. El procedimiento sigue estos pasos:

1. Se enuncia el problema.

Figura 4-1 Diagrama de bloques de un circuito combinacional.

2. Se determina el número de las variables de entrada disponibles y de las variables de salida requeridas.
3. Se asignan símbolos de letra a las variables de entrada y salida.
4. Se deriva la tabla de verdad que define las relaciones requeridas entre las entradas y las salidas.
5. Se obtiene la función booleana simplificada para cada salida.
6. Se dibuja el diagrama lógico.

Una tabla de verdad para un circuito combinacional consta de columnas de entrada y columnas de salida. Los 1 y 0 en las columnas de entrada se obtienen de las 2^n combinaciones binarias disponibles para las n variables de entrada. Los valores binarios para las salidas se determinan del examen del problema enunciado. Una salida puede ser igual ya sea a 0 o 1 para cada combinación válida de entrada. Sin embargo, las especificaciones pueden indicar que algunas combinaciones de entrada no ocurrirán. Estas combinaciones se vuelven condiciones no importa.

Las funciones de salida que se especifican en la tabla de verdad dan la definición exacta del circuito combinacional. Es importante que las especificaciones verbales se interpreten correctamente en una tabla de verdad. Algunas veces el diseñador debe usar su intuición y experiencia para llegar a la interpretación correcta. Las especificaciones verbales rara vez son muy completas y exactas. Cualquier interpretación equivocada que resulte en una tabla de verdad incorrecta producirá un circuito combinacional que no cubriría los requisitos enunciados.

Las funciones booleanas de salida de la tabla de verdad se simplifican por cualquier método disponible, como manipulación algebraica, el método de mapa, o el procedimiento de tabulación. Por lo común, habrá una variedad de expresiones simplificadas a elegir. No obstante, en cualquier aplicación particular ciertas restricciones, limitaciones y criterios servirán como guía en el proceso de escoger una expresión algebraica particular. Un método práctico de diseño sería tener que considerar tales restricciones como (1) número mínimo de compuertas, (2) número mínimo de entradas a una compuerta, (3) tiempo mínimo de propagación de la señal a través del circuito, (4) número mínimo de interconexiones y (5) limitaciones de las capacidades de impulsión de cada compuerta. Ya que todos estos criterios no pueden satisfacerse en forma simultánea, y ya que la importancia de cada restricción se dicta por la aplicación particular, es difícil hacer un enunciado general de lo que constituye una simplificación aceptable. En la mayoría de los casos, la simplificación principia por satisfacer un objetivo elemental, como producir una función booleana simplificada en una forma estándar y proceder de ese punto a cumplir cualesquiera otros criterios de comportamiento.

En la práctica, los diseñadores tienden a ir de la función booleana a una lista de alambrado que muestra las interconexiones entre varias compuertas lógicas estándar. En este caso, el diseño no va más allá de la función booleana simplificada de salida requerida. Sin embargo, un diagrama lógico es de ayuda para visualizar la implementación de compuertas de las expresiones.

4-3 SUMADORES

Las computadoras digitales realizan una variedad de tareas de procesamiento de información. Entre las funciones básicas encontradas están las diversas operaciones aritméticas. Sin duda, la operación aritmética más básica es la adición de dos dígitos binarios. Esta adición simple consta de cuatro operaciones elementales posibles, a saber, $0 + 0 = 0$, $0 + 1 = 1$, $1 + 0 = 1$ y $1 + 1 = 10$. Las primeras tres operaciones producen una suma cuya longitud es un dígito, pero cuando tanto los bits sumando como adendo son iguales a 1, la suma binaria consta de dos dígitos. El bit significativo más alto de este resultado se denomina acarreo. Cuando los números sumando y adendo contienen más dígitos significativos, la cuenta que se lleva obtenida por la adición de dos bits se añade al siguiente par de orden más alto de bits significativos. Un circuito combinacional que lleva a cabo la adición de dos bits se denomina *medio sumador*. Uno que lleva a cabo la adición de tres bits (dos bits significativos y una cuenta que se lleva previa) es un sumador completo. El nombre del primero proviene del hecho de que dos medios sumadores se emplean para implementar un adicionador completo. Los dos circuitos adicionadores son los primeros circuitos combinacionales que van a diseñarse.

Medio sumador

De la explicación verbal del medio sumador, se encuentra que este circuito necesita dos entradas binarias y dos salidas binarias. Las variables de entrada designan los bits sumando y adendo; las variables de salida producen la suma y el acarreo. Es necesario especificar dos variables de salida debido a que el resultado puede constar de dos dígitos binarios. Se asignan en forma arbitraria los símbolos x y y a las dos entradas y S (de suma) y C (para el acarrero) a las salidas.

Ahora que se han establecido el número y nombres de las variables de entrada y salida, ya puede formularse una tabla de verdad para identificar en forma exacta la función del medio sumador. Esta tabla de verdad se muestra a continuación:

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

El acarreo de salida es 0 a menos que ambas entradas sean 1. La salida S representa el bit menos significativo de la suma.

La función booleana simplificada de las dos salidas puede obtenerse de manera directa mediante la tabla de verdad. Las expresiones simplificadas en suma de productos son:

$$S = x'y + xy'$$
$$C = xy$$

El diagrama lógico para esta implementación se muestra en la Fig. 4-2(a), lo mismo que otras cuatro implementaciones para un medio sumador. Todos logran el mismo resultado en lo que respecta al comportamiento de entrada-salida. Ilustran la flexibilidad de la que dispone el diseñador cuando implementa incluso una función lógica combinacional simple como ésta.

Como se mencionó antes, la Fig. 4-2(a) es la implementación del medio sumador en suma de productos. En la Fig. 4-2(b) se muestra la implementación en producto de sumas:

$$S = (x + y)(x' + y')$$
$$C = xy$$

Para obtener la implementación de la Fig. 4-2(c), se observa que S es la OR excluyente de x y y. El complemento de S es la equivalencia de x y y (Sección 2-6):

$$S' = xy + x'y'$$

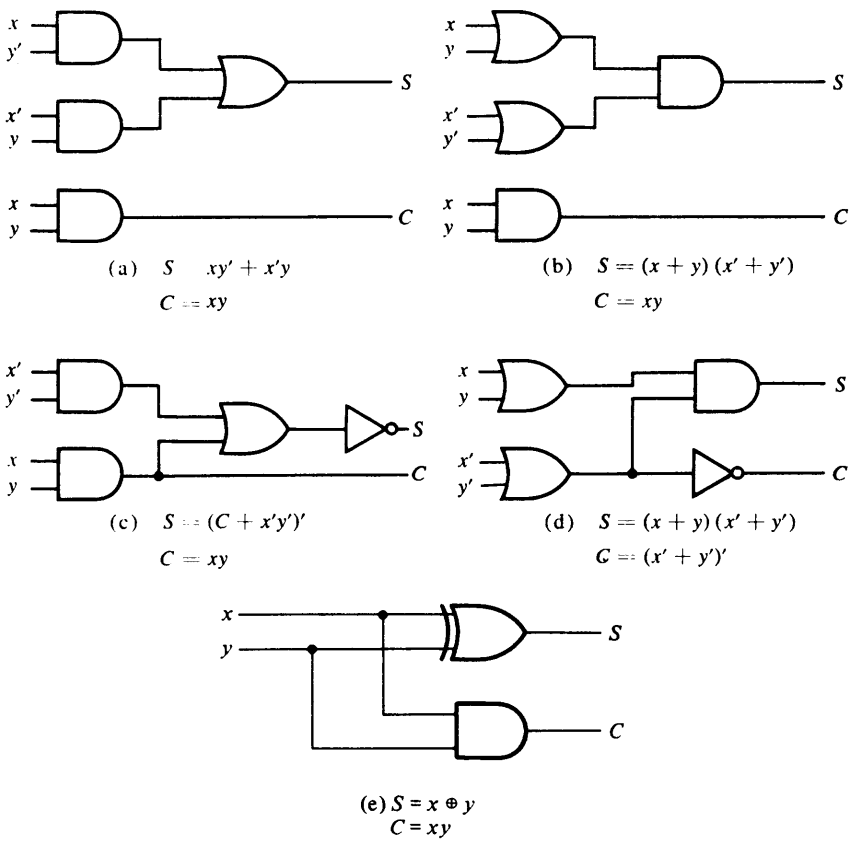


Figura 4-2 Varias implementaciones de un medio adicionador.

pero $C = xy$ y, por lo tanto, tenemos:

$$S = (C + x'y')'$$

En la Fig. 4-2(d) se utiliza la implementación de producto de sumas con C derivada como sigue:

$$C = xy = (x' + y')'$$

el medio sumador puede implementarse con una compuerta OR excluyente y AND, como se muestra en la Fig. 4-2(c). Esta fórmula se usa posteriormente para mostrar que son necesarios dos circuitos medio sumadores para construir un circuito sumador completo.

Sumador completo

Un sumador completo es un circuito combinacional que formar la suma aritmética de tres bits de entrada. Consta de tres entradas y dos salidas. Dos de las variables de entrada, que se indican por x y y, representan los dos bits significativos que van a añadirse. La tercera entrada, z, representa la cuenta que se lleva de la posición previa significativa más baja. Son necesarias dos salidas debido a que la suma aritmética de tres dígitos binarios varía en valor desde 0 a 3 y el 2 o 3 binarios requieren dos dígitos. Las dos salidas se denotan por los símbolos S para suma y C para la cuenta que se lleva. La variable binaria S da el valor del bit menos significativo de la suma. La variable binaria C da la cuenta que se lleva de salida. La tabla de verdad del sumador completo es como sigue:

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Los ocho renglones bajo las variables de entrada denotan todas las combinaciones posibles de 1 y 0 que pueden tener esas variables. Los 1 y 0 de las variables de salida se determinan de la suma aritmética de los bits de entrada. Cuando todos los bits de entrada son 0, la salida es 0. La salida S es igual a 1 sólo cuando una entrada es igual a 1, o cuando todas las tres entradas son iguales a 1. La salida C tiene una cuenta que se lleva de 1 si dos o tres entradas son iguales a 1.

Los bits de entrada y salida del circuito combinacional tienen diferentes interpretaciones en las diversas etapas del problema. En forma física, las señales binarias de

los alambres de entrada se consideran dígitos binarios agregados de manera aritmética para dar una suma de dos dígitos a los alambres de salida. Por otra parte, los mismos valores binarios se consideran variables de funciones booleanas cuando se expresan en la tabla de verdad o cuando el circuito se implementa con compuertas lógicas. Es importante darse cuenta de que se dan dos interpretaciones diferentes a los valores de los bits que se encuentran en este circuito.

La relación lógica de entrada-salida del circuito sumador completo puede expresarse en dos funciones booleanas, una para cada variable de salida. Cada función booleana de salida requiere un mapa único para su simplificación. Cada mapa debe tener ocho cuadros, ya que cada salida es una función de tres variables de entrada. Los mapas en la Fig. 4-3 se utilizan para simplificar las dos funciones de salida. Los 1 en los cuadros de los mapas de S y C se determinan en forma directa mediante la tabla de verdad. Los cuadros con 1 para la salida S no se combinan en cuadros adyacentes para dar una expresión simplificada en suma de productos. La salida C puede simplificarse a una expresión de seis literales. El diagrama lógico para el sumador completo implementado en suma de productos se muestra en la Fig. 4-4. En esta implementación se usan las expresiones booleanas siguientes:

$$S = x'y'z + x'yz' + xy'z' + xyz$$
$$C = xy + xz + yz$$

Pueden desarrollarse otras configuraciones para un adicionador completo. La implementación en producto de sumas requiere el mismo número de compuerta como en la Fig. 4-4, con el número de compuertas AND y OR intercambiado. Un sumador completo puede implementarse con dos medios sumadores y una compuerta OR, como se muestra en la Fig. 4-5. La salida S del segundo medio sumador es la OR excluyente de z y la salida del primer medio sumador, dando:

$$\begin{aligned} S &= z \oplus (x \oplus y) \\ &= z'(xy' + x'y) + z(xy' + x'y)' \\ &= z'(xy' + x'y) + z(xy + x'y') \\ &= xy'z' + x'yz' + xyz + x'y'z \end{aligned}$$

y el acarreo de salida es:

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

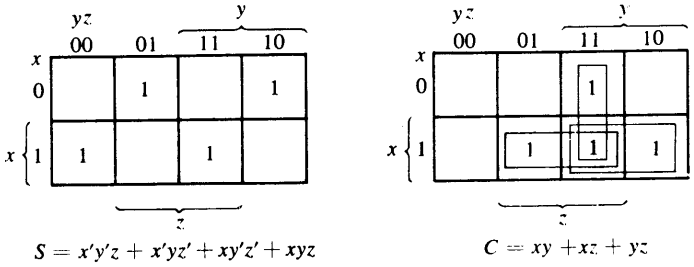


Figura 4-3 Mapas para un sumador completo.

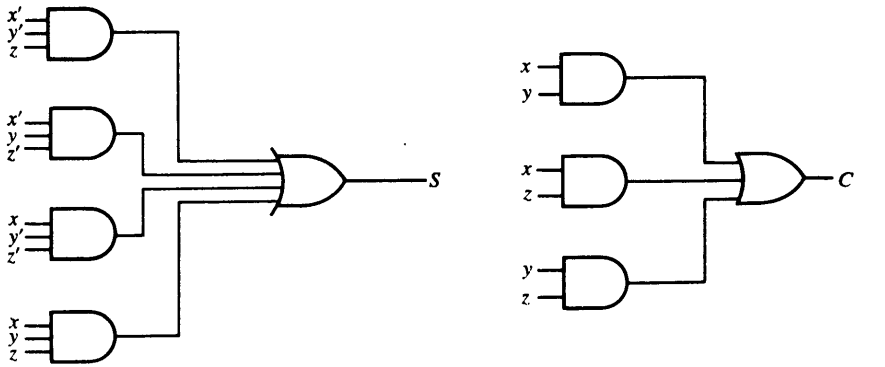


Figura 4-4 Implementación de un sumador completo en suma de productos.

4-4 RESTADORES

La sustracción de dos números binarios puede llevarse a cabo tomando el complemento del sustraendo y agregándolo al minuendo (Sección 1-5). Por este método, la operación de sustracción llega a ser una operación de división que requiere sumadores completos para su implementación en máquina. Es posible implementar la sustracción con circuitos lógicos en una forma directa, como se hace con lápiz y papel. Por este método cada bit sustraendo del número se sustrae de su bit minuendo correspondiente significativo para formar un bit de diferencia. Si el bit minuendo es menor que el bit sustraendo, se toma un 1 de la siguiente posición significativa. El hecho de que se ha tomado un 1 debe llevarse al siguiente par más alto de bit mediante una señal binaria que llega de fuera (salida) de una etapa dada y va a (entrada) la siguiente etapa más alta. En forma precisa así como hay medio sumadores y sumadores completos, hay medio restadores y restadores completos.

Medio restador

Un medio restador es un circuito combinacional que sustrae dos bits y produce su diferencia. También tiene una salida para especificar si se ha tomado un 1. Se designa el bit minuendo por x y el bit sustraendo mediante y. Para llevar a cabo $x - y$, tienen

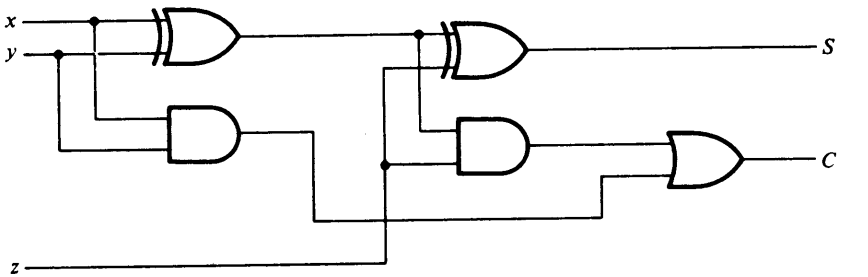


Figura 4-5 Implementación de un sumador completo con dos medio adicionadores y una compuerta OR.

que verificarse las magnitudes relativas de x y y. Si $x \geq y$, se tienen tres posibilidades; $0 - 0 = 0$, $1 - 0 = 1$ y, $1 - 1 = 0$. El resultado se denomina *bit de diferencia*. Si $x < y$, tenemos $0 - 1$ y es necesario tomar un 1 de la siguiente etapa más alta. El 1 que se toma de la siguiente etapa más alta añade 2 al bit minuendo, de la misma forma que en el sistema decimal lo que se toma añade 10 a un dígito minuendo. Con el minuendo igual a 2, la diferencia llega a ser $2 - 1 = 1$. El medio restador requiere dos salidas. Una salida genera la diferencia y se denotará por el símbolo D. La segunda salida, denotada B para lo que se toma, genera la señal binaria que informa a la siguiente etapa que se ha tomado un 1. La tabla de verdad para las relaciones de entrada-salida de un medio restador ahora puede derivarse como sigue:

x	y	B	D
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

La salida que toma B es un 0 en tanto que $x \geq y$. Es un 1 para $x = 0$ y $y = 1$. La salida D es el resultado de la operación aritmética $2B + x - y$.

Las funciones booleanas para las dos salidas del medio restador se derivan de manera directa de la tabla de verdad:

$$D = x'y + xy'$$
$$B = x'y$$

Es interesante observar que la lógica para D es exactamente la misma que la lógica para la salida S en el medio sumador.

Restador completo

Un restador completo es un circuito combinacional que lleva a cabo una sustracción entre dos bits, tomando en cuenta que un 1 se ha tomado por una etapa significativa más baja. Este circuito tiene tres entradas y dos salidas. Las tres entradas, x, y y z, denotan al minuendo, sustraendo y a la toma previa, respectivamente. Las dos salidas, D y B, representan la diferencia y la salida tomada, respectivamente. La tabla de verdad para el circuito es como sigue:

x	y	z	B	D
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Los ocho renglones bajo las variables de entrada designan todas las combinaciones posibles de 1 y 0 que pueden tomar las variables binarias. Los 1 y 0 para las variables de salida están determinados por la sustracción de $x - y - z$. Las combinaciones que tienen salida de toma $z = 0$ se reducen a las mismas cuatro condiciones del medio sumador. Para $x = 0$, $y = 0$ y $z = 1$, tiene que tomarse un 1 de la siguiente etapa, lo cual hace $B = 1$ y añade 2 a x. Ya que $2 - 0 - 1 = 1$, $D = 1$. Para $x = 0$ y $yz = 11$, necesita tomarse otra vez, haciendo $B = 1$ y $x = 2$. Ya que $2 - 1 - 1 = 0$, $D = 0$. Para $x = 1$ y $yz = 01$, se tiene $x - y - z = 0$, lo cual hace $B = 0$ y $D = 0$. Por último, para $x = 1, y = 1, z = 1$, tiene que tomarse 1, haciendo $B = 1$ y $x = 3$ y, $3 - 1 - 1 = 1$, haciendo $D = 1$.

La función booleana simplificada para las dos salidas del restador completo se derivan en los mapas de la Fig. 4-6. Las funciones simplificadas de salida en suma de productos son:

$$D = x'y'z + x'yz' + xy'z' + xyz$$
$$B = x'y + x'z + yz$$

De nuevo se observa que la función lógica para la salida D en el restador completo es exactamente la misma que para la salida S del sumador completo. Además, la salida B se asemeja a la función para C en el sumador completo, excepto que la variable de entrada x está complementada. Debido a estas similitudes, es posible convertir un sumador completo en un restador completo, complementando tan sólo la entrada x antes de su aplicación a las compuertas que forman la salida de acarreo.

4-5 CONVERSION DE CODIGO

La disponibilidad de una gran variedad de códigos para los mismos elementos discretos de información origina el uso de códigos diferentes por sistemas digitales diferentes. Algunas veces es necesario usar la salida de un sistema como la entrada a otro. Debe insertarse un circuito de conversión entre los dos sistemas si cada uno utiliza códigos diferentes para la misma información. Así que, un convertidor de código es un circuito que hace dos sistemas compatibles aun cuando cada uno use un código binario diferente.

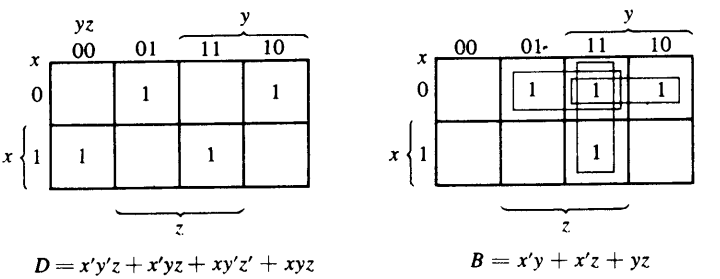


Figura 4-6 Mapas para un restador completo.

Para convertir un código binario *A* en el código binario *B*, las líneas de entrada deben suministrar la combinación bit de elementos como los especifica el código *A* y las líneas de salida, deben generar la combinación bit correspondiente del código *B*. Un circuito combinacional lleva a cabo esta transformación mediante compuertas lógicas. El procedimiento de diseño de los convertidores de código se ilustrará mediante un ejemplo específico de conversión del código BCD en el código exceso-3.

Las combinaciones bit para los códigos BCD y exceso-3 se listan en la Tabla 1-2 (Sección 1-6). Ya que cada código usa cuatro bits para representar un dígito decimal, debe haber cuatro variables de entrada y cuatro variables de salida. Permítase designar las cuatro variables de entrada con los símbolos *A, B, C* y *D* y las cuatro variables de salida por *w, x, y* y *z*. La tabla de verdad que relaciona las variables de entrada y salida se muestra en la Tabla 4-1. Las combinaciones bit de las entradas y sus correspondientes salidas se obtienen de manera directa de la Tabla 1-2. Se observa que cuatro variables binarias pueden tener 16 combinaciones bit, sólo 10 de las cuales se listan en la tabla de verdad. Las seis combinaciones bit que no se listan para las variables de *entrada* son combinaciones no importa. Ya que nunca ocurrirán, se tiene la libertad de asignar las variables de salida ya sea con un 1 o un 0, el que dé un circuito más simple.

Los mapas en la Fig. 4-7 están dibujados para obtener una función booleana simplificada para cada salida. Cada uno de los cuatro mapas en la Fig. 4-7 representa una de las cuatro salidas de este circuito como una función de las cuatro variables de entrada. Los 1 que se marcan en el interior de los cuadros se obtienen de los mintérminos que hacen la salida igual a 1. Los 1 se obtienen de la tabla de verdad pasando sobre las columnas de salida una a la vez. Por ejemplo, la columna bajo la salida *z* tiene cinco números 1; por tanto, el mapa para *z* puede tener cinco 1, cada uno en un cuadro correspondiente al mintérmino que hace que *z* sea igual a 1. Las seis combinaciones no importa se marcan con letras *X*. Una forma posible de simplificar las funciones en suma de productos se lista bajo el mapa de cada variable.

TABLA 4-1 Tabla de verdad para el ejemplo de conversión de código

Entrada BCD				Salida código-exceso-3			
A	*B*	*C*	*D*	*w*	*x*	*y*	*z*
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

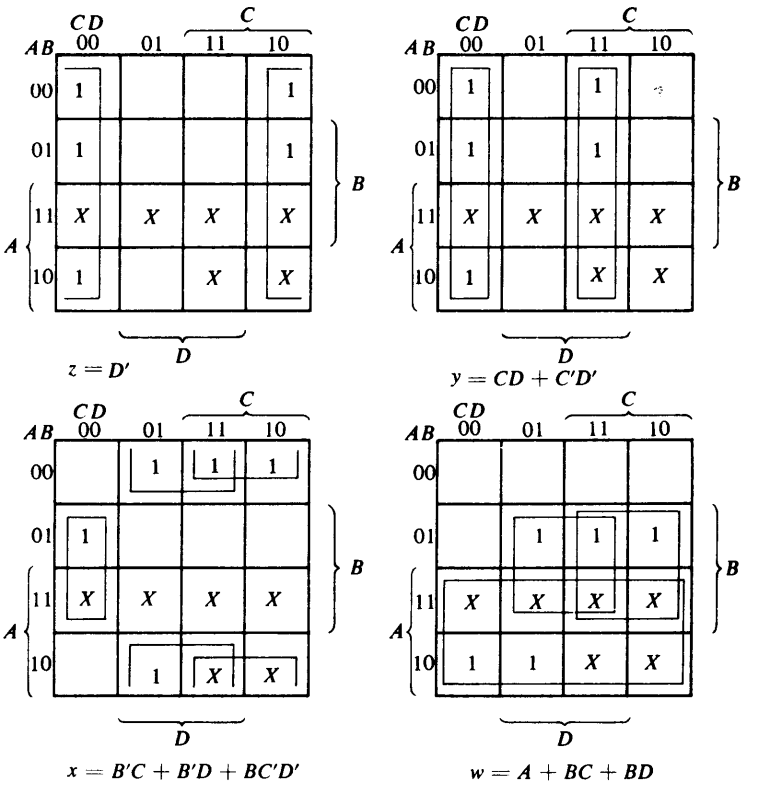


Figura 4-7 Mapas para un convertidor de código BCD-a-exceso-3.

Puede obtenerse de manera directa un diagrama lógico de dos niveles mediante las expresiones booleanas derivadas por los mapas. Hay otras posibilidades diferentes para un diagrama lógico que implemente este circuito. Las expresiones que se obtienen en la Fig. 4-7 pueden manipularse en forma algebraica con el objeto de usar compuertas comunes para dos o más salidas. Esta manipulación, que se muestra a continuación, ilustra la flexibilidad que se obtiene con sistemas de salidas múltiples cuando se implementan con tres o más niveles de compuertas.

$$z = D'$$
$$y = CD + C'D' = CD + (C + D)'$$
$$x = B'C + B'D + BC'D' = B'(C + D) + BC'D'$$
$$= B'(C + D) + B(C + D)'$$
$$w = A + BC + BD = A + B(C + D)$$

El diagrama lógico que implementa las expresiones anteriores se muestra en la Fig. 4-8. En él puede verse que la compuerta OR cuya salida es $C + D$ se ha utilizado para implementar parcialmente cada una de las tres salidas.

Sin contar los invertidores de entradas, la implementación en suma de productos requiere siete compuertas AND y tres compuertas OR. La implementación de la Fig. 4-8 requiere cuatro compuertas AND, cuatro OR y un inversor. Si sólo están disponibles entradas normales, la primera implementación requerirá invertidores para las variables *B, C* y *D,* en tanto que la segunda implementación necesita invertidores para las variables *B* y *D.*

4-6 PROCEDIMIENTO DE ANALISIS

El diseño de un circuito combinacional se inicia con las especificaciones verbales de una función requerida y culmina con un conjunto de funciones booleanas de salida o un diagrama lógico. El *análisis* de un circuito combinacional es en cierta forma el proceso inverso. Principia con un diagrama lógico dado y termina con un conjunto de funciones booleanas, una tabla de verdad o una explicación verbal de la operación del circuito. Si el diagrama lógico que va a analizarse se acompaña con una función nombre o una explicación de lo que se supone que realiza, entonces el problema del análisis se reduce a una verificación de la función enunciada.

El primer paso en el análisis es tener la seguridad de que el circuito dado es combinacional y no secuencial. El diagrama de un circuito combinacional tiene compuertas lógicas sin trayectorias de retroalimentación o elementos de memoria. Una trayectoria de retroalimentación es una conexión de la salida de una compuerta a

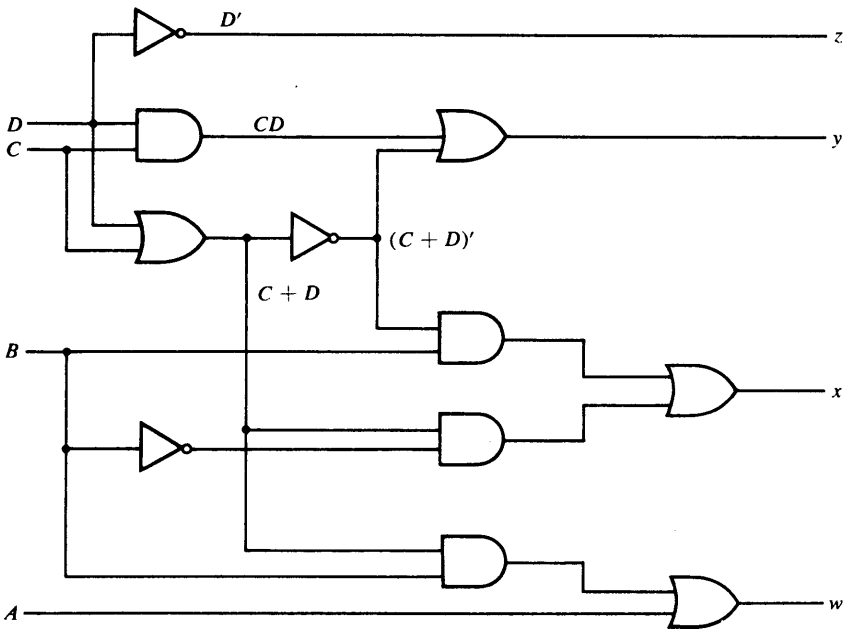


Figura 4-8 Diagrama lógico para un convertidor de código BCD-a-exceso-3.

la entrada de una segunda compuerta que forma parte de la entrada a la primera compuerta. Las trayectorias de retroalimentación o elementos de memoria en un circuito digital definen un circuito secuencial y deben analizarse de acuerdo con los procedimientos delineados en el Capítulo 6.

Una vez que se ha verificado que el diagrama lógico es un circuito combinacional, puede procederse a obtener las funciones booleanas de salida y/o la tabla de verdad. Si el circuito está acompañado por una explicación verbal de su función, entonces las funciones booleanas o la tabla de verdad son suficientes para la verificación. Si la función del circuito está bajo investigación, entonces es necesario interpretar la operación del circuito mediante la tabla de verdad derivada. El éxito de tal investigación se favorece si se tiene experiencia previa y familiaridad con una amplia variedad de circuitos digitales. La habilidad para correlacionar una tabla de verdad con una tarea de procesamiento de información es un arte que se adquiere con la experiencia.

Para obtener las funciones booleanas de salida de un diagrama lógico, se procede como sigue:

1. Se etiquetan con símbolos arbitrarios todas las salidas de compuerta que son una función de las variables de entrada. Se obtienen las funciones booleanas para cada compuerta.
2. Se etiquetan con otros símbolos arbitrarios las compuertas que son una función de las variables de entrada y/o compuertas previamente etiquetadas. Se encuentran las funciones booleanas para esas compuertas.
3. Se repite el proceso delineado en el paso 2 hasta que se han obtenido las salidas del circuito.
4. Por sustitución repetida de las funciones previamente definidas, se obtienen las funciones booleanas de salida en términos sólo de las variables de entrada.

El análisis del circuito combinacional en la Fig. 4-9 ilustra el procedimiento propuesto. Se observa que el circuito tiene tres entradas binarias, A, B y C, y dos salidas binarias, F_1 y F_2. Las salidas de las diversas compuertas se etiquetan con símbolos intermedios. Las salidas de las compuertas que son una función de las variables de entrada sólo son F_2, T_1 y T_2. Las funciones booleanas para estas tres salidas son:

$$F_2 = AB + AC + BC$$
$$T_1 = A + B + C$$
$$T_2 = ABC$$

A continuación se consideran las salidas de compuertas que son una función de los símbolos ya definidos:

$$T_3 = F_2' T_1$$
$$F_1 = T_3 + T_2$$

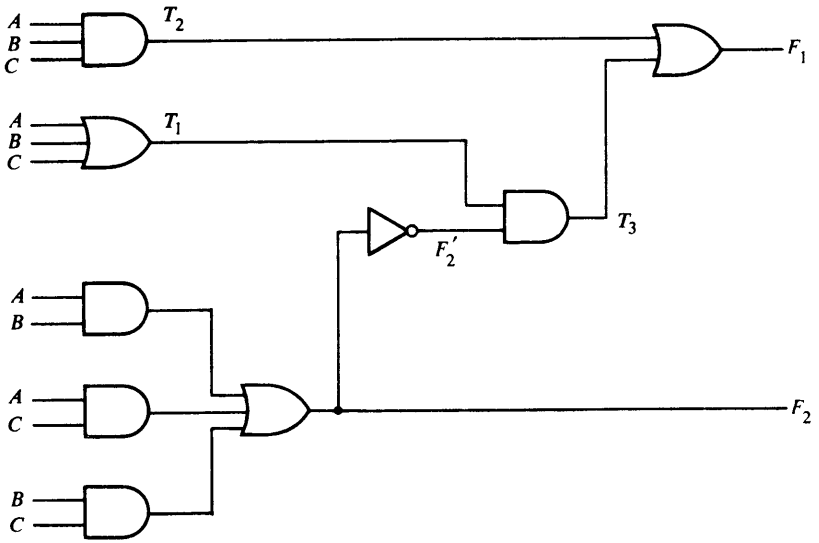


Figura 4-9 Diagrama lógico para el ejemplo de análisis.

La función booleana de salida F_2 expresada con anterioridad ya está dada como una función sólo de las entradas. Para obtener F_1 como una función de A, B y C se forma una serie de sustituciones como sigue:

$$\begin{aligned} F_1 &= T_3 + T_2 = F_2'T_1 + ABC = (AB + AC + BC)'(A + B + C) + ABC \\ &= (A' + B')(A' + C')(B' + C')(A + B + C) + ABC \\ &= (A' + B'C')(AB' + AC' + BC' + B'C) + ABC \\ &= A'BC' + A'B'C + AB'C' + ABC \end{aligned}$$

Si se desea proseguir la investigación y determinar la tarea de transformación de la información realizada por este circuito, puede derivarse la tabla de verdad en forma directa de las funciones booleanas y tratar de reconocer una operación familiar. Por este ejemplo, se observa que el circuito es un sumador completo, con F_1 como la salida de suma y F_2 la salida de la cuenta que se lleva. A, B y C son las tres entradas agregadas en forma aritmética.

La derivación de la tabla de verdad para el circuito es un proceso directo una vez que se conocen las funciones booleanas de salida. Para obtener la tabla de verdad de manera directa del diagrama lógico sin pasar a través de las derivaciones de las funciones booleanas, se procede como sigue:

1. Determínese el número de las variables de entrada al circuito. Para n entradas, fórmense las 2^n combinaciones posibles de entrada de 1 y 0 mediante el listado de los números binarios de 0 a $2^n - 1$.
2. Etiquétense las salidas de las compuertas seleccionadas con símbolos arbitrarios.

3. Obténgase la tabla de verdad para las salidas de las compuertas que son una función sólo de las variables de entrada.
4. Procédase a obtener la tabla de verdad para las salidas de las compuertas que son una función de los valores previamente definidos hasta que las columnas para todas las salidas estén determinadas.

Este proceso puede usarse utilizando el circuito en la Fig. 4-9. En la Tabla 4-2, se forman las ocho combinaciones posibles de las tres variables de entrada. La tabla de verdad para F_2 se determina de manera directa de los valores de A, B y C, con $F = 1$ para cualquier combinación que tenga dos de las tres entradas iguales a 1. La tabla de verdad para F'_2 es el complemento de F_2. Las tablas de verdad para T_1 y T_2 son las funciones OR y AND de las tres variables de entrada, respectivamente. Los valores para T_3 se derivan mediante T_1 y F'_2: T_3 es igual a 1 cuando tanto T_1 y F'_2 son iguales a 1 y a 0 de otra manera. Por último, F_1 es igual a 1 para las combinaciones en las cuales T_2 o T_3, o ambas, son iguales a 1. La inspección de las combinaciones en la tabla de verdad para A, B, C, F_1 y F_2 de la Tabla 4-2 muestra que es idéntica a la tabla de verdad del sumador completo que se presenta en la Sección 4-3 para x, y, z, S y C, respectivamente.

Considérese ahora un circuito combinacional que tiene combinaciones no importa de entrada. Cuando se diseña un circuito como éste, las combinaciones no importa se marcan con X en el mapa y se les asigna una salida de 1 o bien 0, lo que sea más conveniente para la simplificación de la función booleana de salida. Cuando se analiza, un circuito con combinaciones no importa la situación es por completo diferente. Aun cuando se supone que las combinaciones no importa de entrada nunca ocurrirán, el hecho es que si cualquiera de esas combinaciones se aplica a las entradas (intencionalmente o por error), estará presente una salida binaria. El valor de la salida dependerá de la elección para las X que se toma durante el diseño. Parte del análisis de tal circuito puede implicar la determinación de los valores de salida para las combinaciones no importa de entrada. Como ejemplo, considérese el convertidor de códigos BCD-a-exceso-3 diseñado en la Sección 4-5. Las salidas obtenidas cuando las seis combinaciones que no se utilizan del código BCD se aplican a las entradas son:

Entradas *BCD* sin uso				Salidas			
A	*B*	*C*	*D*	*w*	*x*	*y*	*z*
1	0	1	0	1	1	0	1
1	0	1	1	1	1	1	0
1	1	0	0	1	1	1	1
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	1
1	1	1	1	1	0	1	0

Estas salidas pueden derivarse mediante el método de análisis de la tabla como se delinea en esta sección. En este caso particular, las salidas pueden obtenerse de manera directa de los mapas en la Fig. 4-7. Por la inspección de los mapas, se determina cuándo las X en los cuadros del mintérmino correspondiente para cada salida se han

TABLA 4-2 Tabla de verdad para el diagrama lógico de la Fig. 4-9

A	B	C	F_2	F_2'	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

incluido con los 1 o los 0. Por ejemplo, el cuadro para el mintérmino m_{10} (1010) se ha incluido con los 1 para las salidas w, x y z, pero no para y. Por lo tanto, las salidas para m_{10} son $wxyz = 1101$, como se lista en la tabla anterior. También se observa que las primeras tres salidas de la tabla no tienen significado en el código exceso-3, y que las últimas tres salidas corresponden al decimal 5, 6, y 7, respectivamente. Esta coincidencia es por entero una función de la elección para las X tomadas durante el diseño.

4-7 CURCUITOS NAND DE NIVEL MULTIPLE

Los circuitos combinacionales se construyen más a menudo con compuertas NAND o NOR, más bien que con compuertas AND y OR. Las compuertas NAND y NOR son más comunes desde el punto de vista del hardware, ya que están disponibles en la forma de circuitos integrados. Debido a la preeminencia de las compuertas NAND y NOR en el diseño de los circuitos combinacionales, es importante tener la capacidad de reconocer las relaciones que existen entre los circuitos construidos con compuertas AND-OR y sus diagramas equivalentes NAND o NOR.

La implementación de diagramas lógicos en dos niveles NAND y NOR se presentó en la Sección 3-6. Aquí se considera el caso más general de circuitos de niveles múltiples. El procedimiento para obtener circuitos NAND se presenta en esta sección, y para obtener los circuitos NOR en la siguiente sección.

Compuerta universal

La compuerta NAND se dice que es una compuerta universal porque cualquier sistema digital puede implementarse con ella. Los circuitos combinacionales al igual que los secuenciales pueden construirse con esta compuerta, debido a que el circuito flip-flop (el elemento de memoria de uso más frecuente en los circuitos secuenciales) puede construirse mediante dos compuertas NAND conectadas de la parte posterior de una a la de otra, como se muestra en la Sección 6-2.

Para mostrar que cualquier función booleana puede implementarse con compuertas NAND sólo se necesita mostrar que las operaciones lógicas AND, OR y NOT

pueden implementarse con compuertas NAND. La implementación de las operaciones AND, OR y NOT con compuertas NAND se muestra en la Fig. 4-10. La operación NOT se obtiene mediante una compuerta NAND de una entrada, que es en realidad otro símbolo para un circuito inversor. La operación AND requiere dos compuertas NAND. La primera produce el inversor AND y la segunda actúa como un inversor para producir la salida normal. La operación OR se lleva a cabo a través de una compuerta NAND con inversores adicionales en cada salida.

Una forma conveniente de implementar un circuito combinacional con compuertas NAND es obtener las funciones booleanas simplificadas en términos de AND, OR y NOT y convertir las funciones en la lógica NAND. La conversión de la expresión algebraica para operaciones AND, OR y NOT en operaciones NAND, por lo común es bastante complicada debido a que implica un gran número de aplicaciones del teorema de De Morgan. Esta dificultad se evita por el uso de simples manipulaciones de circuito y simples reglas, como se delinea abajo.

Implementación de una función booleana
Método de diagrama de bloques

La implementación de funciones booleanas con compuertas NAND puede obtenerse mediante una técnica de manipulación de un simple diagrama de bloques. El método requiere que se dibujen otros dos diagramas lógicos antes para obtener el diagrama lógico NAND. No obstante, el procedimiento es muy sencillo y directo.

1. Mediante la expresión algebraica, dibújese el diagrama lógico con compuertas AND, OR y NOT. Se supone que están disponibles entradas tanto normal como complementaria.
2. Dibújese un segundo diagrama lógico con la lógica NAND equivalente, como se da en la Fig. 4-10, para sustituir cada compuerta AND, OR y NOT.

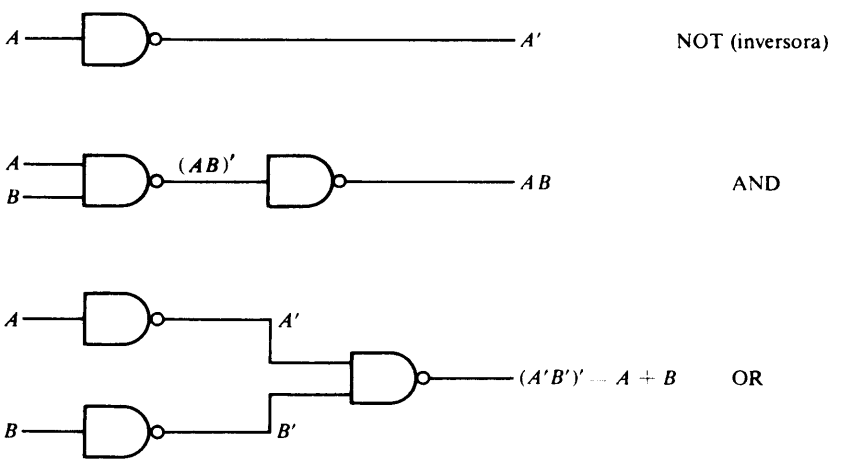


Figura 4-10 Implementación de compuertas NOT, AND u OR con compuertas.

3. Elimínense del diagrama cualesquiera dos inversores en cascada, ya que la inversión doble no realiza una función lógica. Elimínense los inversores conectados a entradas únicas alternas y compleméntense las variables de entrada correspondientes. El nuevo diagrama lógico que se obtiene es la implementación requerida en compuerta NAND.

Este procedimiento se ilustra en la Fig. 4-11 para la función:

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

La implementación AND-OR de esta función se muestra en el diagrama lógico de la Fig. 4-11(a). Para cada compuerta AND, se sustituye una compuerta AND seguida por un inversor; para cada compuerta OR, se sustituyen inversores de entrada seguidos por una compuerta NAND. Esta sustitución es consecuencia directa de las equivalencias lógicas de la Fig. 4-10 y se muestra en el diagrama en la Fig. 4-11(b). Este diagrama tiene siete inversores y cinco compuertas de dos entradas NAND listadas con números dentro del símbolo de compuerta. Los pares de inversores conectados en cascada (de cada casilla AND a cada casilla OR) se eliminan, ya que forman una inversión doble. El inversor conectado a la entrada B se elimina y la variable de entrada se designa por B'. El resultado es el diagrama lógico NAND que se muestra en la Fig. 4-11(c), con el número dentro de cada símbolo que identifica la compuerta en la Fig. 4-11(b).

En este ejemplo se demuestra que el número requerido de compuertas NAND para implementar la función booleana es igual al número de compuertas AND-OR, siempre que estén disponibles las entradas tanto normal como de complemento. Si sólo están disponibles las entradas normales, deben usarse inversores para generar cualesquiera entradas complementarias requeridas.

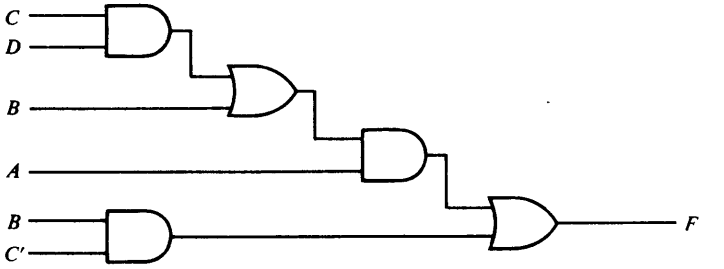
Un segundo ejemplo de implementación NAND se muestra en la Fig. 4-12. La función booleana que se implantará es:

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

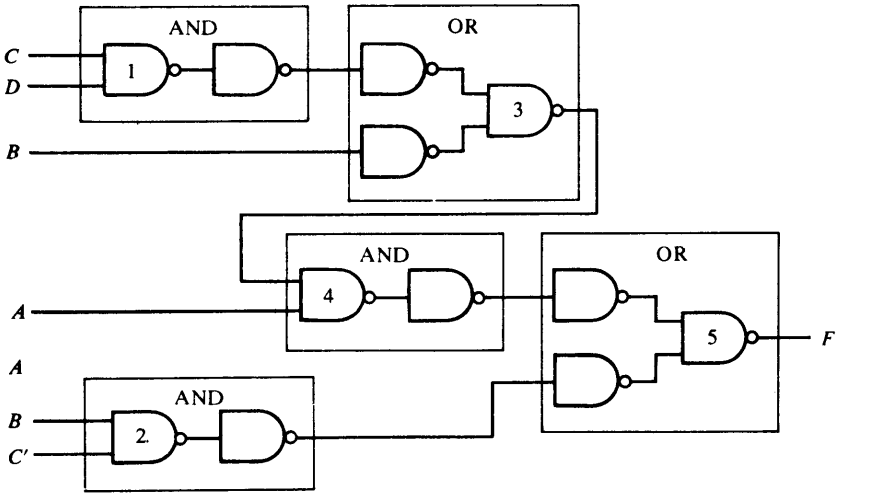
La implementación AND-OR se muestra en la Fig. 4-12(a) y su sustitución en lógica NAND, en la Fig. 4-12(b). Puede eliminarse un par de inversores en cascada. Las tres entradas externas E, A y B', las cuales van directamente a los inversores, se complementan y se eliminan los inversores correspondientes. La implementación final en compuerta NAND se muestra en la Fig. 4-12(c).

El número de compuertas NAND para el segundo ejemplo es igual al número de compuertas AND-OR más un inversor adicional en la salida (compuerta NAND 5). En general, el número de compuertas NAND requeridas para implementar una función es igual al número de compuertas AND-OR, excepto por un inversor ocasional. Esto es cierto siempre que estén disponibles las entradas tanto normal como complementaria, debido a que la conversión obliga a que se complementen ciertas variables de entrada.

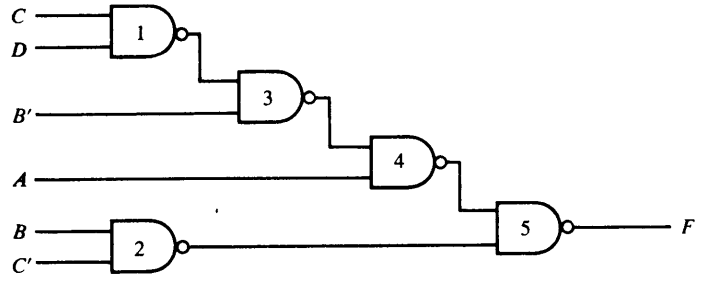
El método de diagrama de bloques es un poco cansado, ya que requiere el dibujo de dos diagramas lógicos para obtener la respuesta de un tercero. Con cierta experien-



(a) implementación AND/OR

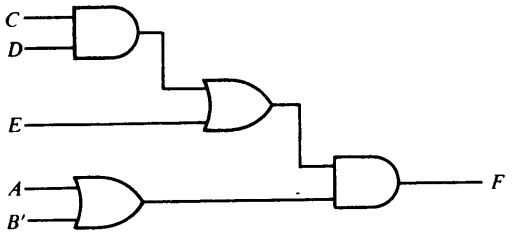


(b) Sustitución con funciones equivalentes NAND de las de la Fig. 5-8

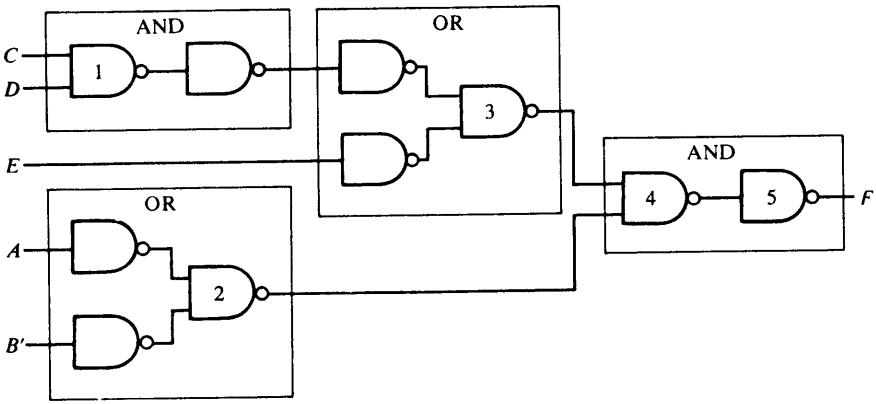


(c) Implementación NAND

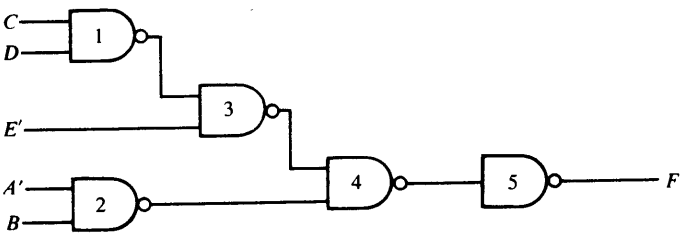
Figura 4-11 Implementación de $F = A(B + CD) + BC'$ con compuertas NAND.



(a) implementación AND/OR



(b) Sustitución con funciones equivalentes NAND



(c) Implementación NAND

Figura 4-12 Implementación de *(A + B') (CD + E)* con compuertas NAND.

cia, es posible reducir la cantidad de trabajo anticipando los pares de inversores en cascada y los inversores en las entradas. A partir del procedimiento que acaba de delinearse, no es difícil derivar reglas generales para la implementación de funciones booleanas con compuertas NAND en forma directa de una expresión algebraica.

Procedimiento de análisis

En el procedimiento anterior se consideró el problema de derivar un diagrama lógico NAND de una función booleana dada. El procedimiento inverso es el análisis del

problema que principia con un diagrama lógico NAND dado y termina con una expresión booleana o una tabla de verdad. El análisis de diagramas lógicos NAND sigue los mismos procedimientos que se presentaron en la Sección 4-6 para el análisis de circuitos combinacionales. La única diferencia es que la lógica NAND requiere una aplicación repetida del teorema de De Morgan. Ahora se demostrará la derivación de la función booleana mediante un diagrama lógico. Entonces se mostrará la derivación de la tabla de verdad de manera directa a partir del diagrama lógico NAND. Por último, se presenta un método para convertir un diagrama lógico AND-OR mediante la manipulación en diagrama de bloques.

Derivación de la función booleana por manipulación algebraica

El procedimiento para derivar la función booleana de un diagrama lógico se delineó en la Sección 4-6. Este procedimiento se demuestra para el diagrama lógico NAND que se ilustra en la Fig. 4-13 y es el mismo que se encuentra en la Fig. 4-11(c). Primero, todas las salidas de compuerta se etiquetan con símbolos arbitrarios. Segundo, se derivan las funciones booleanas para las salidas de compuerta que reciben solamente entradas externas:

$$T_1 = (CD)' = C' + D'$$
$$T_2 = (BC')' = B' + C$$

La segunda forma se sigue en forma directa del teorema de De Morgan y a veces puede ser de uso más conveniente. Tercero, las funciones booleanas de compuertas que tienen entrada de funciones derivadas previamente se determinan en orden consecutivo hasta que la salida se expresa en términos de las variables de entrada:

$$\begin{aligned} T_3 &= (B'T_1)' = (B'C' + B'D')' \\ &= (B + C)(B + D) = B + CD \\ T_4 &= (AT_3)' = [A(B + CD)]' \\ F &= (T_2T_4)' = \{(BC')'[A(B + CD)]'\}' \\ &= BC' + A(B + CD) \end{aligned}$$

Derivación de la tabla de verdad

El procedimiento para obtener en forma directa la tabla de verdad a partir de un diagrama lógico también se delinea en la Sección 4-6. Este procedimiento se demuestra para el diagrama lógico NAND que se ilustra en la Fig. 4-13. Primero, las cuatro variables de entrada, junto con sus 16 combinaciones de número 1 y 0, se listan como se muestra en la Tabla 4-3. Segundo, las salidas de todas las compuertas se etiquetan con símbolos arbitrarios como en la Fig. 4-13. Tercero, se obtiene la tabla de verdad para las salidas de las compuertas que son una función sólo de las variables de entrada.

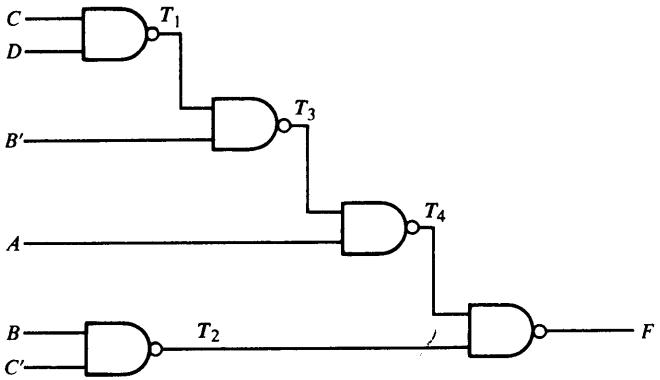
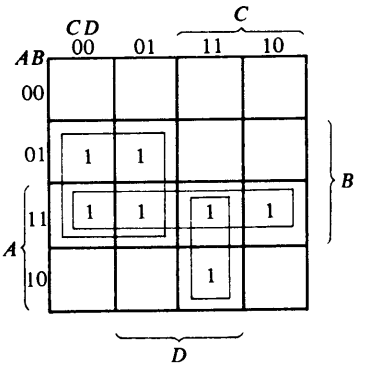


Figura 4-13 Ejemplo de análisis.

Estas son T_1 y T_2. $T_1 = (CD)'$; de modo que se marcan 0 en los renglones donde tanto C como D son iguales a 1 y se llenan los demás renglones de T_1 con números 1. También, $T_2 = (BC')'$; de modo que se marcan números 0 en los renglones donde $B = 1$ y $C = 0$, y se llenan los demás renglones de T_2 con números 1. Se procede entonces a obtener la tabla de verdad para las salidas de las compuertas que son una función de las salidas que se definieron con anterioridad hasta que la columna para la salida F queda determinada. Ahora es posible obtener una expresión algebraica para la salida mediante la tabla de verdad derivada. El mapa que se muestra en la Fig. 4-14 se obtiene de manera directa de la Tabla 4-3 y tiene números 1 en los cuadros de los mintérminos

TABLA 4-3 Tabla de verdad del circuito de la Figura 4-13

A	B	C	D	T_1	T_2	T_3	T_4	F
0	0	0	0	1	1	0	1	0
0	0	0	1	1	1	0	1	0
0	0	1	0	1	1	0	1	0
0	0	1	1	0	1	1	1	0
0	1	0	0	1	0	1	1	1
0	1	0	1	1	0	1	1	1
0	1	1	0	1	1	1	1	0
0	1	1	1	0	1	1	1	0
1	0	0	0	1	1	0	1	0
1	0	0	1	1	1	0	1	0
1	0	1	0	1	1	0	1	0
1	0	1	1	0	1	1	0	1
1	1	0	0	1	0	1	0	1
1	1	0	1	1	0	1	0	1
1	1	1	0	1	1	1	0	1
1	1	1	1	0	1	1	0	1



$$F = AB + BC' + ACD$$

Figura 4-14 Derivación de F a partir de la Tabla 4-3.

para los cuales F es igual a 1. La expresión simplificada que se obtiene mediante el mapa es:

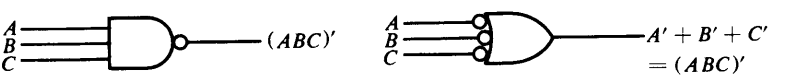
$$F = AB + ACD + BC' = A(B + CD) + BC'$$

Esta es igual a la expresión que se muestra en la Fig. 4-11, por tanto se verifica la respuesta correcta.

Transformación del diagrama de bloque

Algunas veces es conveniente convertir un diagrama lógico NAND en un diagrama lógico AND-OR equivalente para facilitar el procedimiento de análisis. Al hacer esto, la función booleana puede derivarse con más facilidad sin emplear el teorema de De Morgan. La conversión de diagramas lógicos se lleva a cabo a través de un proceso inverso del que se utiliza para la implementación. En la Sección 3-6, se mostraron dos símbolos gráficos alternos para la compuerta NAND. Estos símbolos se repiten en la Fig. 4-15 por motivos de comodidad. Por el uso juicioso de ambos símbolos, es posible convertir un diagrama NAND en una forma AND-OR equivalente.

La conversión de un diagrama lógico NAND en un diagrama AND-OR se lleva a cabo a través de un cambio en símbolos desde AND-invertida a OR- invertida en niveles alternos de compuertas. El primer nivel que va a cambiarse a un símbolo OR-invertido debe ser el último nivel. Estos cambios producen pares de círculos a lo

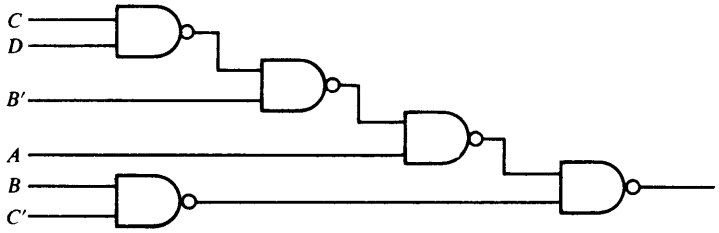


(a) AND-inversora (b) OR-inversora

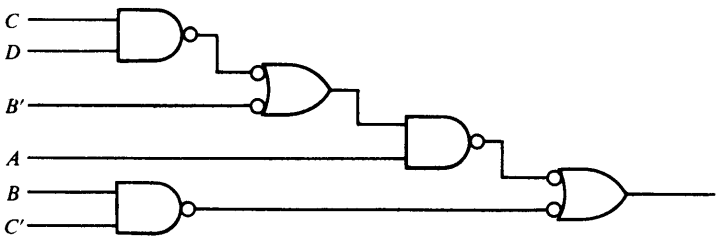
Figura 4-15 Dos símbolos para compuerta NAND.

largo de la misma línea, y esos pueden eliminarse ya que representan complementación doble. Además, una compuerta de una entrada AND u OR puede eliminarse ya que no realiza una función lógica. Una compuerta AND u OR de una entrada con un círculo en la entrada o salida se cambia a un circuito inversor.

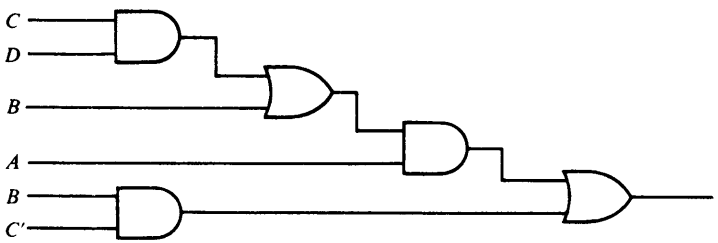
El procedimiento se demuestra en la Fig. 4-16. El diagrama lógico NAND en la Fig. 4-16(a) se convertirá en un diagrama AND-OR. El símbolo de la compuerta en el último nivel se cambia a un OR-invertido. Al buscar niveles alternos, se encuentra una compuerta más que requiere un cambio de símbolo como se muestra en la Fig. 4-16(b). Cualesquiera dos círculos a lo largo de la misma línea se eliminan. Los círculos que van a entradas externas también se eliminan, siempre que esté complementada la variable correspondiente de entrada. El diagrama lógico AND-OR requerido se muestra en la Fig. 4-16(c).



(a) Diagrama lógico NAND



(b) Sustitución con símbolos OR inversora en niveles alternos



(c) Diagrama lógico AND-OR

Figura 4-16 Conversión del diagrama lógico NAND en AND-OR.

4-8 CIRCUITOS NOR DE NIVELES MULTIPLES

La función NOR es la dual de la función NAND. Por esta razón, todos los procedimientos para la lógica NOR forman un dual de los procedimientos y reglas correspondientes desarrollados para la lógica NAND. En esta sección se enumeran diversos métodos para la implementación y análisis de la lógica NOR por el seguimiento de la misma lista de tópicos usados para la lógica NAND. Sin embargo, se incluyen explicaciones menos detalladas para evitar la repetición excesiva del material en la Sección 4-7.

Compuerta universal

La compuerta NOR es universal debido a que cualquier función booleana puede implementarse con ella, incluyendo un circuito flip-flop como se muestra en la Sección 6-2. La conversión de AND, OR y NOT en NOR se ilustra en la Fig. 4-17. La operación NOT se obtiene de una compuerta NOR de una entrada, todavía otro símbolo de un circuito inversor. La operación OR requiere dos compuertas NOR. La primera produce la OR-invertida y la segunda actúa como un inversor para obtener la salida normal. La operación AND se lleva a cabo a través de una compuerta NOR con inversores adicionales en cada entrada.

Implementación de una función booleana
Método de diagrama de bloques

El procedimiento de diagrama de bloques para implementar funciones booleanas con compuertas NOR es similar al procedimiento delineado en la sección anterior para las compuertas NAND.

1. Se dibuja el diagrama lógico AND-OR a partir de la expresión algebraica dada. Se supone que están disponibles las entradas tanto normal como complementaria.

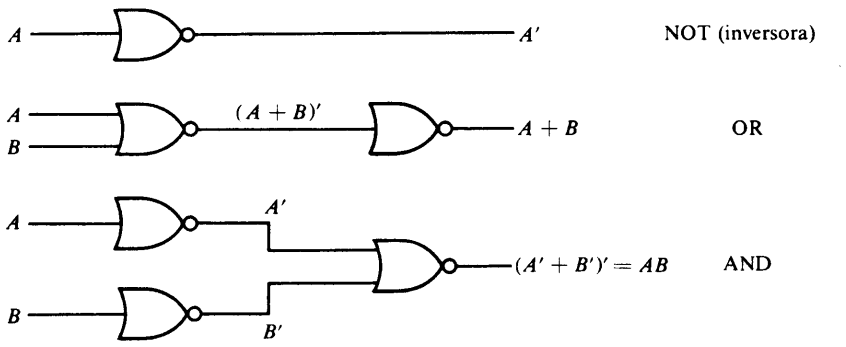
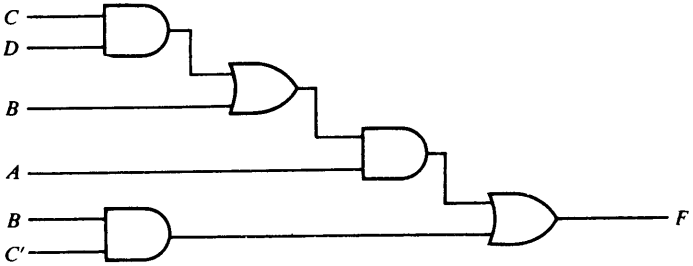
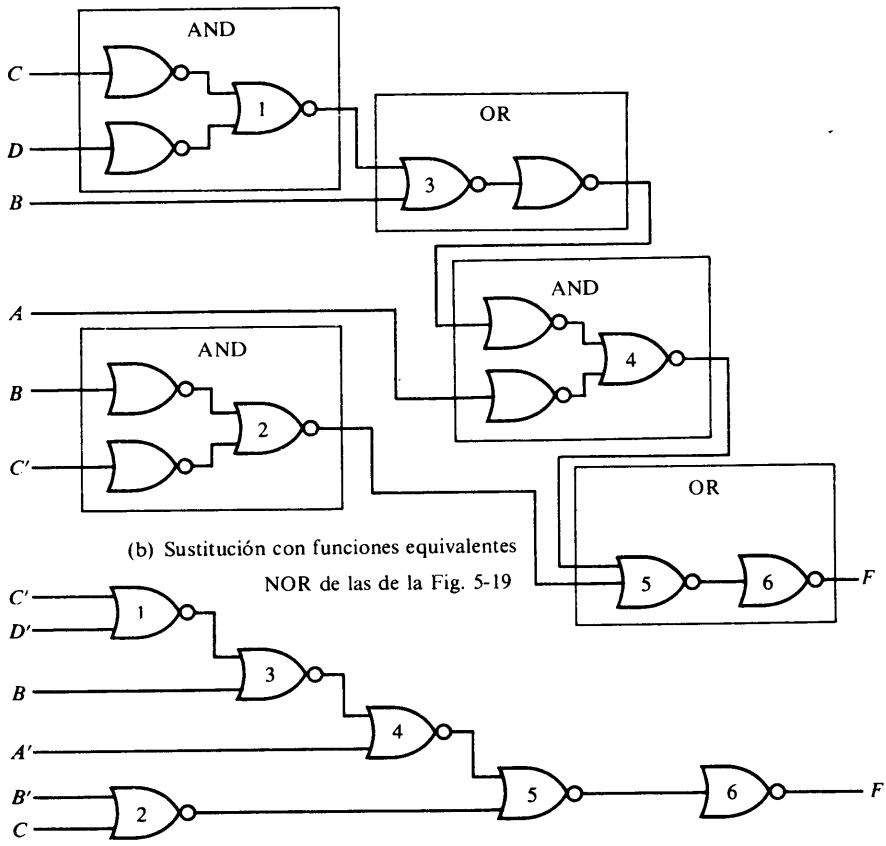


Figura 4-17 Implementación de compuertas NOT, OR y AND por compuertas NOR.



(a) Implementación AND/OR



(b) Sustitución con funciones equivalentes NOR de las de la Fig. 5-19

(c) Implementación NOR

Figura 4-18 Implementación de $F = A(B + CD) + BC'$ con compuertas NOR.

2. Se dibuja un segundo diagrama lógico con la lógica NOR equivalente, como se muestra en la Fig. 4-17, que sustituye a cada compuerta AND, OR y NOT.

3. Se eliminan del diagrama pares de inversores en cascada. Se eliminan inversores conectados a entradas externas únicas y se complementa la variable de entrada correspondiente.

El procedimiento se ilustra en la Fig. 4-18 para la función:

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

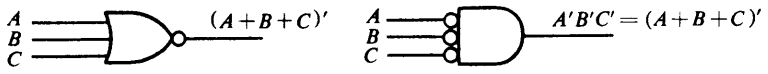
La implementación AND-OR de la función se muestra en el diagrama lógico en la Fig. 4-18(a). Para cada compuerta OR, se sustituye una compuerta NOR seguida por un inversor. Para cada compuerta AND, se sustituyen inversores de entrada seguidos por una compuerta NOR. El par de inversores en cascada de la casilla OR y de la casilla AND se eliminan. Los cuatro inversores conectados a entradas externas se eliminan y las variables de entrada se complementan. El resultado es el diagrama lógico NOR que se muestra en la Fig. 4-18(c). El número de compuertas NOR en este ejemplo es igual al número de compuertas AND-OR más un inversor adicional en la salida (compuerta NOR 6). En general, el número requerido de compuertas NOR para implementar una función booleana es igual al número de compuertas AND-OR, excepto por un inversor ocasional. Esto es cierto siempre que estén disponibles las entradas tanto normal como complementaria, debido a que la conversión obliga a que ciertas variables de entrada estén complementadas.

Procedimiento de análisis

El análisis de los diagramas lógicos NOR sigue los mismos procedimientos que se presentaron en la Sección 4-6 para el análisis de circuitos combinacionales. Para derivar la función booleana de un diagrama lógico, se marcan las salidas de las diversas compuertas con símbolos arbitrarios. Por sustituciones repetitivas, se obtiene la variable de salida como función de las variables de entrada. Para obtener la tabla de verdad de un diagrama lógico sin derivar primero la función booleana, se forma una tabla donde se listan las n variables de entrada con 2^n renglones de 1 y 0. Se deriva la tabla de verdad de las diversas salidas de compuerta NOR en sucesión, hasta que se obtiene la salida en la tabla de verdad. La función de salida de una compuerta NOR típica es de la forma $T + (A + B' + C)'$; de modo que la tabla de verdad para T está marcada con un 0 para las combinaciones donde $A = 1$ o $B = 0$ o $C = 1$. El resto de los renglones se llena con números 1.

Transformación del diagrama de bloques

Para convertir un diagrama lógico NOR en su diagrama lógico AND-OR equivalente, se usan los dos símbolos para las compuertas NOR que se muestran en la Fig. 4-19. El OR-invertido es el símbolo normal para una compuerta NOR y el AND-invertido es una alternativa conveniente en la que se utiliza el teorema de De Morgan y la convención de los círculos pequeños en las entradas denota complementación.



(a) OR-inversora (b) AND-inversora

Figura 4-19 Dos símbolos para compuerta NOR.

La conversión de un diagrama lógico NOR en un diagrama AND-OR se lleva a cabo a través de un cambio de símbolos de OR-invertido en AND-invertido principiando desde el último nivel y de niveles alternos. Los pares de círculos pequeños a lo largo de la misma línea se eliminan. Se elimina una compuerta de una entrada AND u OR, pero si tiene un pequeño círculo en la entrada o salida, se convierte en una inversora.

Este procedimiento se demuestra en la Fig. 4-20, donde el diagrama lógico NOR en (a) se convierte en un diagrama AND-OR. El símbolo de la compuerta en el último nivel (5) se cambia a un AND-invertido. Al buscar niveles alternos, se encuentra una compuerta en el nivel 3 y dos en el nivel 1. Estas tres compuertas pasan por un cambio de símbolo como se muestra en (b). Cualesquiera dos círculos a lo largo de la misma línea se eliminan. Los círculos que van a entradas externas también se eliminan, siempre que la variable de entrada correspondiente se complemente. La compuerta en el nivel 5 se vuelve una compuerta AND de una entrada y se elimina. El diagrama lógico AND-OR requerido se muestra en la Fig. 4-20(c).

4-9 OR-EXCLUYENTE Y FUNCIONES DE EQUIVALENCIA

Las operaciones binarias OR-excluyente y de equivalencia, denotadas por 0 y 0, respectivamente, realizan las siguientes funciones booleanas:

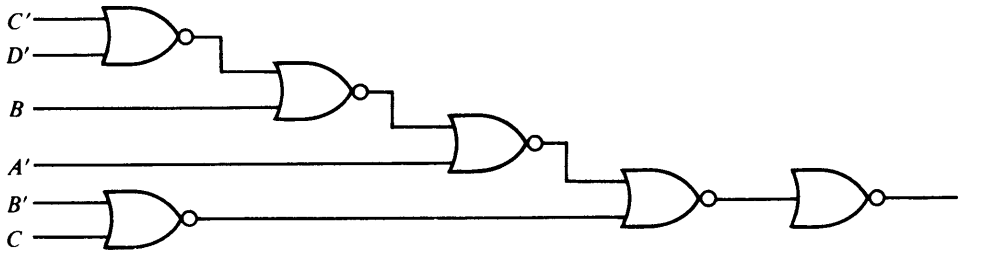
$$x \oplus y = xy' + x'y$$
$$x \odot y = xy + x'y'$$

Las dos operaciones son los complementos una de otra. Cada una es conmutativa y asociativa. Debido a estas dos propiedades, una función de tres o más variables puede expresarse sin paréntesis como se indica:

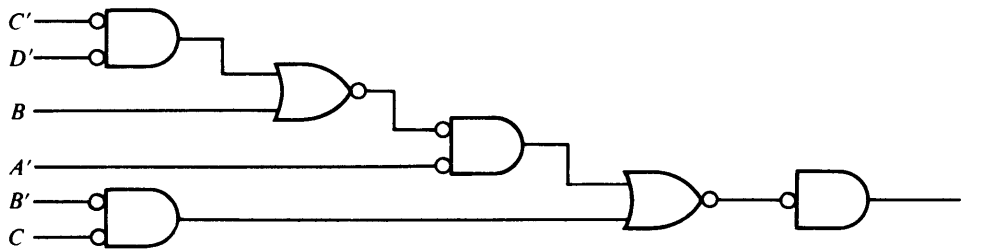
$$(A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C$$

Esto puede implicar la posibilidad de utilizar compuertas OR-excluyente (o equivalencia) con tres o más salidas. Sin embargo, las compuertas OR-excluyente de entradas múltiples son antieconómicas desde un punto de vista de hardware. De hecho, incluso una función de dos entradas por lo común se construye con otros tipos de compuertas. Por ejemplo, en la Fig. 4-21(a) se muestra la implementación de una función OR-excluyente de dos entradas con compuertas AND, OR y NOT. En la Fig. 4-21 (b) se muestra con compuertas NAND.

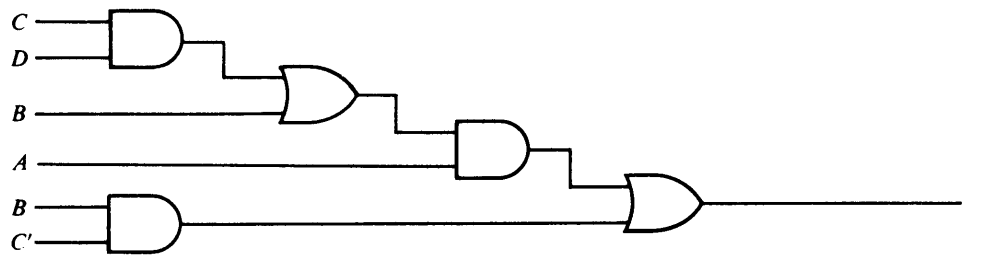
Sólo un número limitado de funciones booleanas puede expresarse exclusivamente en términos de operaciones OR-excluyente o de equivalencia. No obstante, estas funciones surgen con bastante frecuencia durante el diseño de sistemas digitales. Las



(a) Diagrama lógico NOR



(b) Sustitución con símbolos AND inversora en niveles alternos



(c) Diagrama lógico AND-OR

Figura 4-20 Conversión del diagrama lógico NOR en AND-OR.

dos funciones son de utilidad particular en operaciones aritméticas y en detección y corrección de errores.

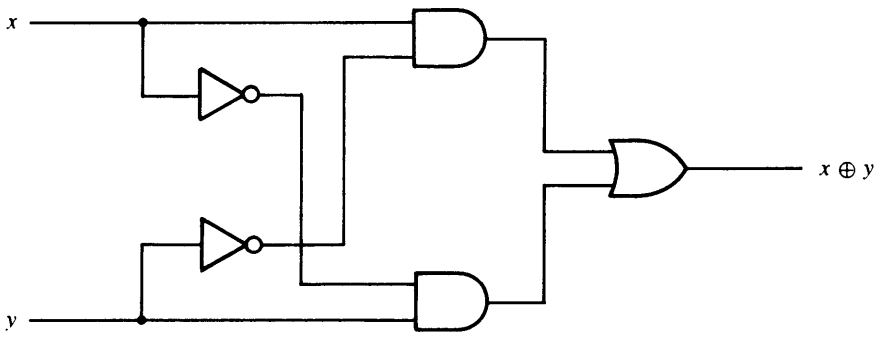
Una expresión OR-excluyente de n variables es igual a la función booleana con $2^n/2$ mintérminos cuyos números binarios equivalentes tienen un número impar de 1. Esto se demuestra en el mapa de la Fig. 4-22(a) para el caso de cuatro variables. Hay 16 mintérminos para cuatro variables. La mitad de los mintérminos tiene un valor numérico con número impar de 1; la otra mitad tiene un valor numérico con un número par de 1. El valor numérico de un mintérmino está determinado por los números de renglón y columna del cuadro que representa al mintérmino. El mapa de la Fig. 4-22(a) tiene 1 en los cuadros cuyos números de mintérmino tienen un número impar de 1. La función puede expresarse en términos de operaciones OR excluyentes de las cuatro variables. Esto se justifica por la siguiente manipulación algebraica:

$$A \oplus B \oplus C \oplus D = (AB' + A'B) \oplus (CD' + C'D)$$
$$= (AB' + A'B)(CD + C'D') + (AB + A'B')(CD' + C'D)$$
$$= \Sigma(1, 2, 4, 7, 8, 11, 13, 14)$$

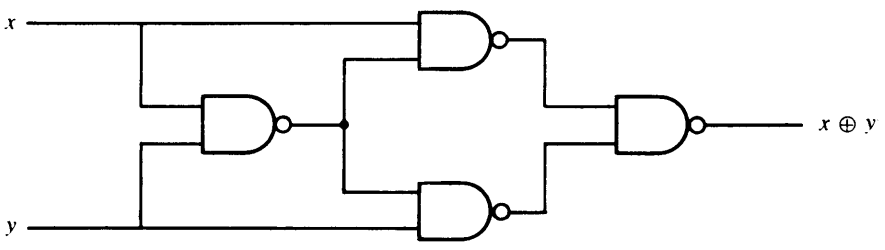
Una expresión de n variables de equivalencia es igual a la función booleana con $2^n/2$ mintérminos, cuyos números binarios equivalentes tienen un número par de 0. Esto se demuestra en el mapa en la Fig. 4-22(b) para el caso de cuatro variables. Los cuadros con 1 representan los ocho mintérminos con número par de 0, y la función puede expresarse en términos de las operaciones de equivalencia en las cuatro variables.

Cuando el número de variables en una función es impar, los mintérminos con un número par de 0 son los mismos que los mintérminos con un número impar de 1. Esto se demuestra en el mapa de tres variables de la Fig. 4-23(a). En consecuencia, una expresión OR-excluyente es igual a una expresión de equivalencia cuando ambas tienen el mismo número impar de variables. Sin embargo, forman los complementos una de otra cuando el número de variables es par, como se muestra en los dos mapas en la Fig. 4-22(a) y (b).

Cuando los mintérminos de una función con un número impar de variables tienen un número par de 1 (o en forma equivalente, un número impar de 0), la función



(a) con compuertas AND-OR-NOT



(b) con compuertas NAND

Figura 4-21 Implementación OR-excluyente.

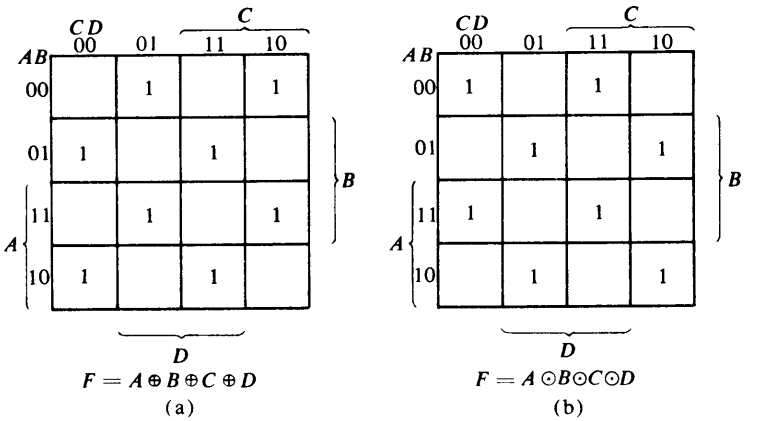


Figura 4-22 Mapa para (a) una función OR-excluyente y (b) una función equivalente, ambas de cuatro variables.

puede expresarse como el complemento ya sea de una expresión OR-excluyente o una expresión equivalente. Por ejemplo, la función de tres variables que se muestra en el mapa de la Fig. 4-23(b) puede expresarse de la siguiente forma:

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

o

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

La salida S de un sumador completo y la salida D de un restador completo (Sección 4-3) puede implementarse con funciones OR-excluyente debido a que cada función consta de cuatro mintérminos con valores numéricos que tienen un número impar de 1. La función OR-excluyente se usa en forma extensa en la implementación de operaciones aritméticas digitales, debido a que estas últimas por lo común se implantan a través de procedimientos que requieren una operación repetitiva de adición o sustracción.

Las funciones OR-excluyente y de equivalencia son muy útiles en sistemas que requieren códigos de detección de errores y corrección de errores. Como se expuso en

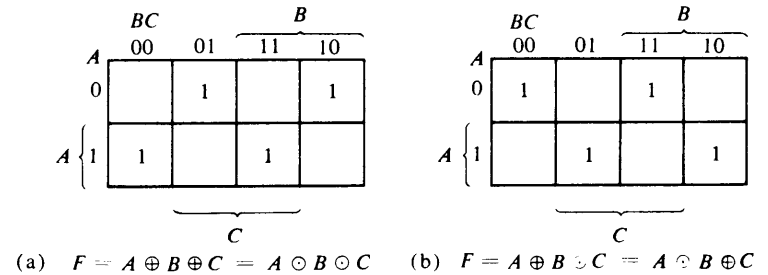


Figura 4-23 Mapa para funciones de tres variables.

la Sección 1-6, un bit de paridad es un esquema para detectar errores durante la transmisión de información binaria. Un bit de paridad es un bit adicional incluido con un mensaje binario para hacer que el número de los 1 sea impar o bien par. El mensaje, que incluye el bit de paridad, se transmite y entonces se verifica en la terminal receptora para buscar errores. Un error se detecta si la paridad verificada no corresponde con la transmitida. El circuito que genera el bit de paridad en el transmisor se conoce como *generador de paridad.* El circuito que verifica la pridad en el receptor se denomina *verificador de paridad.*

Como ejemplo, considérese un mensaje de tres bits que se transmite con un bit de impar-paridad. En la Tabla 4-4 se muestra la tabla de verdad para el generador de paridad. Los tres bits x, y y z constituyen el mensaje y son las entradas al circuito. El bit de paridad P es la salida. Para paridad impar, el bit P se genera de modo que el número total de 1 sea impar (incluyendo P). Mediante la tabla de verdad, se ve que $P = 1$, cuando el número de 1 en x, y y z es par. Esto corresponde al mapa de la Fig. 4-23(b); de modo que la función para P puede expresarse como sigue:

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

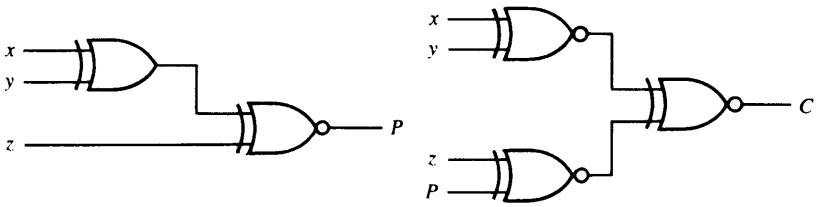
El diagrama lógico para el generador de paridad se muestra en la Fig. 4-24(a). Consta de una compuerta OR-excluyente de dos entradas y una compuerta de equivalencia de dos entradas. Las dos compuertas pueden intercambiarse y producir todavía la misma función, ya que P también es igual a:

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

El mensaje de tres bits y el bit de paridad se transmiten a su destino, donde se aplican a un circuito verificador de paridad. Un error ocurre durante la transmisión si la paridad de los cuatro bits recibidos es par, ya que la información binaria transmitida fue originalmente impar. La salida C del verificador de paridad debe ser un 1 cuando ocurre un error, esto es, cuando el número de 1 en las cuatro entradas es par. La tabla 4-5 es la tabla de verdad para el circuito verificador de impar-paridad. Mediante el cual

TABLA 4-4 Generación de paridad-impar

Mensaje de 3 bit			Bit de paridad generado
x	y	z	P
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0



(a) Generador paridad impar de 3 bit (b) Verificador paridad impar de 4 bit

Figura 4-24 Diagramas lógicos para generación y verificación de paridad.

se ve que la función para C consta de ocho mintérminos con valores numéricos que tienen un número par de 0. Esto corresponde al mapa de la Fig. 4-22(b); de modo que la función puede expresarse con operadores de equivalencia como sigue:

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

El diagrama lógico para el verificador de paridad se muestra en la Fig. 4-24(b) y consta de tres compuertas de equivalencia de dos entradas.

Es de interés observar que el generador de paridad puede implementarse con el circuito de la Fig. 4-24(b), si la entrada P se mantiene en forma permanente a lógica 0 y la salida se marca como P, en donde la ventaja es que el mismo circuito puede usarse para generación de paridad al igual que para verificación de paridad.

Del ejemplo anterior es obvio, que los circuitos de generación de paridad y verificación de paridad siempre tienen una función de salida, que incluye la mitad de

TABLA 4-5 Verificación de paridad-impar

Cuatro bits recibidos				Verificación de error de paridad
x	y	z	P	
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

los mintérminos cuyos valores numéricos tienen ya sea un número par o bien impar de 1. Como consecuencia, deben implementarse con compuertas de equivalencia y/o OR excluyente.

BIBLIOGRAFIA

1. Rhyne, V. T., *Fundamentals of Digital Systems Design.* Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1973.
2. Peatman, J. P., *The Design of Digital Systems.* New York: McGraw-Hill Book Co., 1972.
3. Nagle, H. T. Jr., B. D. Carrol, and J. D. Irwin, *An Introduction to Computer Logic.* Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1975.
4. Hill, F. J., y G. R. Peterson, *Introduction to Switching Theory and Logical Design,* 3a. ed. New York: John Wiley & Sons, Inc., 1981.
5. Maley, G. A., y J. Earle, *The Logic Design of Transistor Digital Computers.* Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1963.
6. Friedman, A. D., y P. R. Menon, *Theory and Design of Switching Circuits.* Woodland Hills, Calif.: Computer Science Press, Inc., 1975.

PROBLEMAS

4-1. Un circuito combinacional tiene cuatro entradas y una salida. La salida es igual a 1 cuando (1) todas las entradas son iguales a 1 o (2) ninguna de las entradas es igual a 1 o (3) un número impar de entradas son iguales a 1.
(a) Obtenga la tabla de verdad.
(b) Encuentre la función simplificada de salida en suma de productos.
(c) Encuentre la función simplificada de salida en producto de sumas.
(d) Dibuje los dos diagramas lógicos.

4-2. Diseñe un circuito combinacional que acepte un número de tres bits y genere una salida de número binario igual al cuadrado del número de entrada.

4-3. Es necesario multiplicar dos números binarios, cada uno de dos bits de largo, con objeto de formar su producto en binario. Permita que los dos números se representen por a_1, a_0 y b_1, b_0, donde el subíndice 0 denota el bit menos significativo.
(a) Determine el número de líneas de salida requerido.
(b) Encuentre las expresiones booleanas simplificadas para cada salida.

4-4. Repita el problema 4-3 para formar la suma (en lugar del producto) de los dos números binarios.

4-5. Diseñe un circuito combinacional con cuatro líneas de entrada que representen un dígito decimal en BCD y cuatro líneas de salida que generen el complemento a 9 del dígito de entrada.

4-6. Diseñe un circuito combinacional cuya entrada es un número de cuatro bits y cuya salida es el complemento a 2 del número de entrada.

4-7. Diseñe un circuito combinacional que multiplique por 5 un dígito decimal de entrada representado en BCD. La salida también está en BCD. Muestre que las salidas pueden obtenerse por las líneas de entrada sin utilizar ninguna compuerta lógica.

4-8. Diseñe un circuito combinacional que detecte un error en la representación de un dígito decimal en BCD. En otras palabras, obtenga un diagrama lógico cuya salida sea lógica 1 cuando las salidas contienen una combinación no usada en el código.

4-9. Implementar un restador completo con dos medios restadores y una compuerta OR.

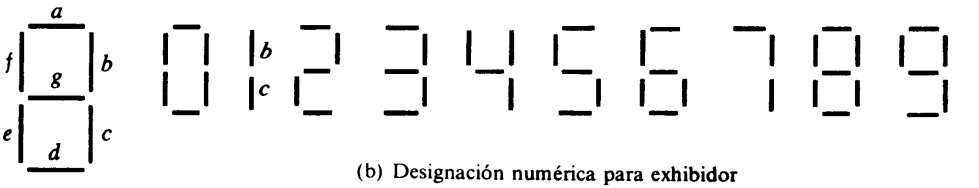
4-10. Muestre cómo puede convertirse un sumador completo en un restador completo con adición de un circuito inversor.

4-11. Diseñe un circuito combinacional que convierta un dígito decimal del código 8, 4, —2, —1 en el código BCD.

4-12. Diseñe un circuito combinacional que convierta un dígito decimal del código 2, 4, 2, 1 en el código 8, 4, —2, —1.

4-13. Obtenga el diagrama lógico que convierte un número binario de cuatro dígitos en un número decimal en BCD. Observe que son necesarios dos dígitos decimales ya que los números binarios están comprendidos entre 0 y 15.

4-14. Un decodificador de BCD-a-siete-segmentos es un circuito combinacional que acepta un dígito decimal en BCD y genera las salidas apropiadas para la selección de segmentos en



(a) Designación de segmentos

(b) Designación numérica para exhibidor

Figura P4-14.

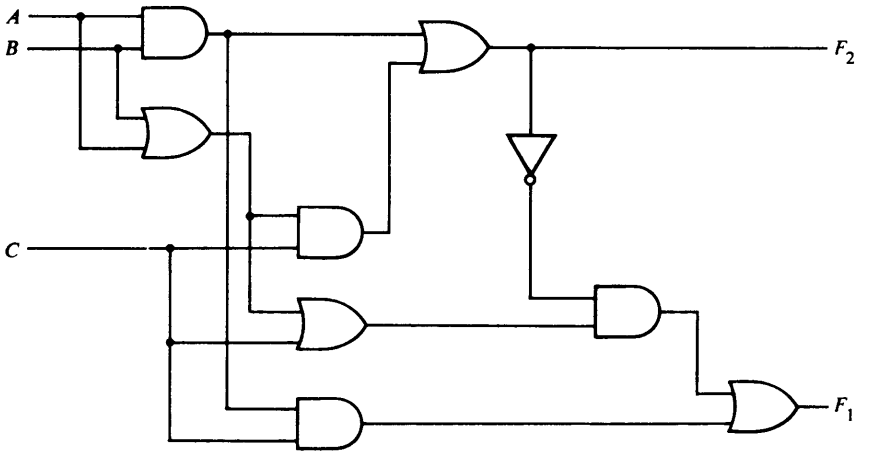


Figura P4-15.

un exhibidor indicador usado para mostrar el dígito decimal. Las siete salidas del decodificador *(a, b, c, d, e, f, g)* seleccionan los segmentos correspondientes al exhibidor como se muestra en la Fig. P4-14(a). La designación numérica escogida para representar el dígito decimal se muestra en la Fig. P4-14(b). Diseñe el circuito decodificador de BCD-a-siete-segmentos.

4-15. Analice los circuitos combinacionales de dos salidas que se muestran en la Fig. P4-15. Obtenga las funciones booleanas para las dos salidas y explique la operación del circuito.

4-16. Derive la tabla de verdad del circuito que se muestra en la Fig. P4-15.

4-17. Utilice el método de diagrama de bloques para convertir el diagrama lógico de la Fig. 4-8 en una implementación NAND.

4-18. Repita el problema 4-17 para la implementación NOR.

4-19. Obtenga el diagrama lógico NAND de un adicionador completo mediante las funciones booleanas:

$$C = xy + xz + yz$$

$$S = C'(x + y + z) + xyz$$

4-20. Determine la función booleana para la salida F del circuito que se muestra en la Fig. P4-20. Obtenga un circuito equivalente con menos compuertas NOR.

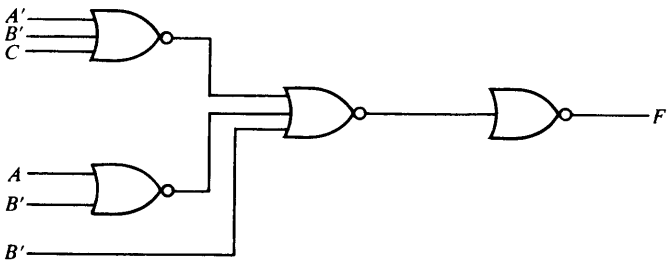


Figura P4-20.

4-21. Determine las funciones booleanas de salida de los circuitos en la Fig. P4-21.

4-22. Obtenga la tabla de verdad para los circuitos en la Fig. P4-21.

4-23. Obtenga el diagrama lógico AND-OR equivalente de la Fig. P4-21(a).

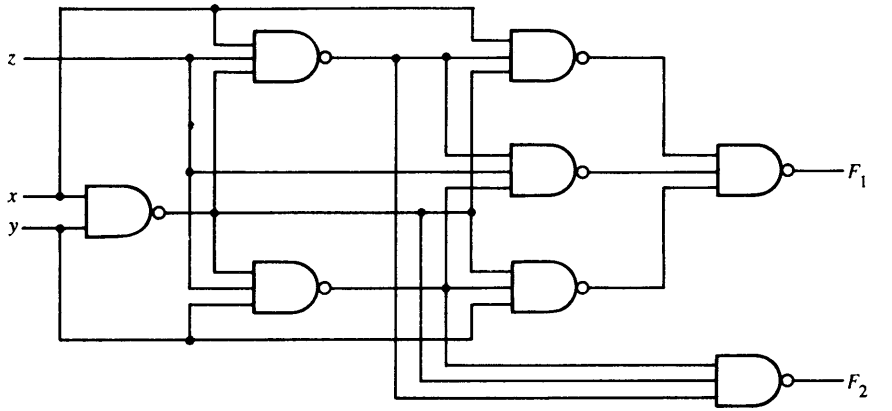
4-24. Obtenga el diagrama lógico AND-OR equivalente en la Fig. P4-21(b).

4-25. Obtenga el diagrama lógico de una función de equivalencia de dos entradas utilizando (a) compuertas AND, OR y NOT; (b) compuertas NOR; y (c) compuertas NAND.

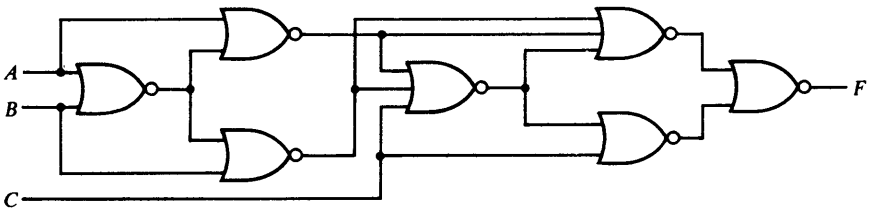
4-26. Muestre que el circuito en la Fig. 4-2(b) es un OR-excluyente.

4-27. Muestre que $A \odot B \odot C \odot D = \Sigma\,(0, 3, 5, 6, 9, 10, 12, 15)$.

4-28. Diseñe un circuito combinacional que convierta un número de cuatro bits en código reflejado (Tabla 1-4) en un número binario de cuatro bits. Implemente el circuito con compuertas OR-excluyentes.



(a)



(b)

Figura P4-21.

4-29. Diseñe un circuito combinacional para verificar paridad par de cuatro bits. Se requiere una salida de lógica 1 cuando los cuatro bits no constituyen una paridad par.

4-30. Implemente las cuatro funciones booleanas listadas usando tres circuitos medio adicionadores (Fig. 4-2e).

$D = A \oplus B \oplus C$

$E = A'BC + AB'C$

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

$G = ABC$

4-31. Implemente la función booleana:

$$F = AB'CD' + A'BCD' + AB'C'D + A'BC'D$$

con compuertas OR-excluyente y AND.

Lógica combinacional con MSI y LSI

5

5-1 INTRODUCCION

El objetivo de la simplificación de función booleana es obtener una expresión algebraica que, cuando se implemente, proporcione un circuito de bajo costo. Sin embargo, pueden definirse los criterios que determinan un sistema o circuito de bajo costo si va a evaluarse el éxito de la simplificación lograda. El procedimiento de diseño para los circuitos combinacionales que se presentó en la Sección 4-2 minimiza el número de compuertas requeridas para implementar una función dada. El procedimiento clásico supone que, dados dos circuitos que realizan la misma función, el que requiere menos compuertas es preferible, ya que costará menos. Esto no necesariamente es cierto cuando se utilizan circuitos integrados.

Ya que se incluyen varias compuertas lógicas en un solo paquete IC, se vuelve económico usar el mayor número de compuertas que estén listas en el paquete que se utilice, incluso si al hacerlo se aumenta el número total de compuertas. Además, algunas de las interconexiones entre compuertas entre muchos circuitos IC están en el interior de la pastilla y es más económico usar tantas conexiones internas como sea posible con objeto de minimizar el número de alambres entre las clavijas externas. Con los circuitos integrados, no es la cuenta de compuertas la que determina el costo, sino el número y tipo de circuito IC empleados del número de interconexiones externas necesarias para implementar la función dada.

Hay numerosas ocasiones en las que el método clásico de la Sección 4-2 no producirá el mejor circuito combinacional para implementar una función dada. Además, la tabla de verdad y el procedimiento de simplificación en este método llegan a ser demasiado engorrosos si el número de variables de entrada es grande en exceso. El circuito final que se obtiene dicta que se implemente con una conexión aleatoria de compuertas SSI, lo cual puede requerir un número relativamente grande de circuitos IC y alambres de interconexión. En muchos casos la aplicación de un procedimiento de diseño alterno puede producir un circuito combinacional para una función dada que sea mejor en mucho al que se obtiene al seguir el método clásico de diseño. La posibilidad de un procedimiento alterno de diseño depende del problema particular y del ingenio del diseñador. El método clásico constituye un procedimiento general que,

si se sigue, garantiza producir un resultado. Sin embargo, antes de aplicar el método clásico, siempre es prudente investigar la posibilidad de un método alterno que puede ser más eficiente para el problema particular a la mano.

La primera pregunta que hay que responder antes de embarcarse en un diseño detallado de un circuito combinacional es verificar si la función está ya disponible en un paquete IC. Están disponibles en forma comercial numerosos dispositivos MSI. Estos dispositivos realizan funciones digitales específicas empleadas en forma común en el diseño de sistemas digitales de computadora. Si no puede encontrarse un dispositivo MSI que produzca exactamente la función necesaria, un diseñador con recursos puede ser capaz de formular un método para incorporar un dispositivo MSI en su circuito. La selección de componentes MSI con preferencia a las compuertas SSI es en extremo importante, ya que en forma invariable causa una reducción considerable de paquetes IC y alambres de interconexión.

En la primera mitad de este capítulo se presentan ejemplos de circuitos combinacionales diseñados por otros métodos diferentes del procedimiento clásico. Todos los ejemplos demuestran la construcción interna de funciones MSI existentes. Por tanto, se presentan nuevas herramientas de diseño y al mismo tiempo se familiariza al lector con las funciones MSI existentes. La familiaridad con las funciones MSI disponibles en muy importante, no sólo en el diseño de circuitos combinacionales, sino también en el diseño de sistemas digitales de computadoras más complicados.

En forma ocasional se encuentran circuitos MSI y LSI que pueden aplicarse en forma directa al diseño e implementación de cualquier circuito combinacional. En la segunda mitad del capítulo se introducen cuatro técnicas de diseño lógico combinacional mediante MSI y LSI. Estas técnicas hacen uso de las propiedades generales de decodificadores, multiplexores, memorias sólo de lectura (ROM) y arreglos lógicos programables (PLA). Estos cuatro componentes IC tienen un gran número de aplicaciones. Su uso en la implementación de circuitos combinacionales como aquí se describe es sólo una de muchas otras aplicaciones.

5-2 SUMADOR BINARIO PARALELO

El sumador completo que se introdujo en la Sección 4-3 forma la suma de dos bits y un previo acarreo. Dos números binarios de n bits cada uno pueden sumarse mediante este circuito. Para demostrarlo con un ejemplo específico, considérense dos números binarios, $A = 1011$ y $B = 0011$, cuya suma es $S = 1110$. Cuando se agrega un par de bits a través de un sumador completo, el circuito produce un acarreo para usarse con el par de bits una posición significativa más alta. Esto se muestra en la siguiente tabla:

Subíndice i	4 3 2 1		*Sumador completo de la Fig. 4-5*
Acarreo de entrada	0 1 1 0	C_i	z
Consumando	1 0 1 1	A_i	x
Sumando	0 0 1 1	B_i	y
Suma	1 1 1 0	S_i	S
Acarreo de salida	0 0 1 1	C_{i+1}	C

Los bits se suman con sumadores completos, principiando desde la posición menos significativa (subíndice 1), para formar el bit de suma y el bit de acarreo. Las entradas y salidas del circuito sumador completo en la Fig. 4-5 también se indican arriba. El acarreo de entrada, C_1 en la posición menos significativa debe ser 0. El valor de C_{i+1} en una posición significativa dada, es el acarreo de salida del sumador completo. Este valor se transfiere al acarreo de entrada del sumador completo que suma los bits en una posición significativa más alta a la izquierda. Por lo tanto, los bits de suma se generan principiando desde la posición más a la derecha y están disponibles tan pronto se genera el previo bit correspondiente de acarreo.

La suma de dos números binarios de *n*-bit, *A* y *B*, puede generarse en dos formas: ya sea en serie o en paralelo. El método de adición en serie usa sólo un circuito sumador completo y un dispositivo de almacenamiento para mantener la salida de acarreo generada. El par de bits en *A* y *B* se transfieren en serie, uno a la vez, a través del sumador completo único para producir una cadena de bits de salida para la suma. El acarreo de salida almacenado, de un par de bits, se utiliza como un acarreo de entrada, para el siguiente par de bits. En el método paralelo se emplean *n* circuitos adicionadores completos, y todos los bits de *A* y *B* se aplican en forma simultánea. El acarreo de salida de un sumador completo se conecta al acarreo de entrada del sumador completo una posición a la izquierda. Tan pronto se generan los acarreos, la suma correcta de bits emerge de las salidas de suma de todos los sumadores completos.

Un sumador binario paralelo es una función digital que produce la suma aritmética de dos números binarios en paralelo. Consta de sumadores completos conectados en cascada, con el acarreo de salida de un sumador completo conectada al acarreo de entrada, del siguiente sumador completo.

En la Fig. 5-1 se muestra la interconexión de cuatro circuitos del sumador completo (FA) para proporcionar un sumador binario paralelo de 4-bit. Los bits sumandos de *A* y los bits adendos de *B* se designan por números de subíndice de derecha a izquierda, donde el subíndice 1 denota el bit de bajo orden. Los acarreos se conectan en una cadena a través de los sumadores completos. El acarreo de entrada al sumador es C_1 y el acarreo de salida es C_5. Las salidas *S* generan los bits requeridos de suma. Cuando el circuito sumador completo de 4-bit se encapsula dentro de un

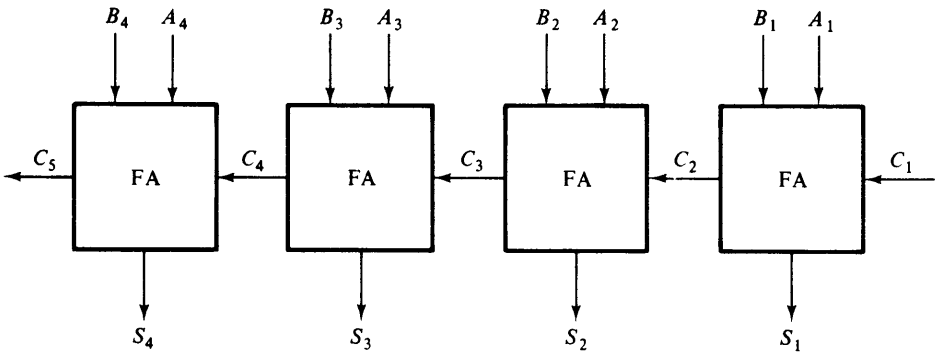


Figura 5-1 Sumadores completos de 4-bit.

paquete IC, tiene cuatro terminales para los bits sumando, cuatro terminales para los bits adendo, cuatro terminales para los bits suma y dos terminales para los acarreos de entrada y salida.*

Un sumador paralelo de n-bit requiere n sumadores completos. Puede construirse para 4-bit, 2-bit y 1-bit de circuitos IC sumadores completos poniendo en cascada varios paquetes. El acarreo de salida, de un paquete debe conectarse con el acarreo de entrada del paquete siguiente con los bits siguientes de orden más alto.

Los sumadores completos de 4-bit son un ejemplo típico de una función NSI. Pueden utilizarse en muchas aplicaciones que implican operaciones aritméticas. Obsérvese que el diseño de este circuito por el método clásico requeriría una tabla de verdad con $2^9 = 512$ entradas, ya que hay nueve entradas al circuito. Por el uso de un método iterativo de poner en cascada una función ya conocida, se tiene capacidad de obtener una implementación simple y bien organizada.

La aplicación de esta función MSI al diseño de un circuito combinacional se demuestra en el siguiente ejemplo.

EJEMPLO 5-1: Diseñe un convertidor de código BCD-a-exceso-3.

Este circuito se diseñó en la Sección 4-5 por el método clásico. El circuito obtenido mediante este diseño se muestra en la Fig. 4-8 y requiere 11 compuertas. Cuando se implementa con compuertas SSI requiere 3 paquetes IC y 14 conexiones internas de alambre (sin incluir las conexiones de entrada y salida). La inspección de la tabla de verdad revela de manera inmediata que el código equivalente exceso-3 puede obtenerse mediante el código BCD por la adición del binario 0011. Esta adición puede implementarse con facilidad mediante un circuito MSI de sumadores completos de 4-bit, como se muestra en la Fig. 5-2. El dígito BCD se aplica a las entradas A. Las entradas B se establecen a un 0011 constante. Esto se hace por la aplicación de lógica 1 a B_1 y B_2 y lógica 0 a B_3, B_4 y C_1. La lógica 1 y la lógica 0 son señales físicas cuyos valores dependen de la familia lógica IC que se usa. Para circuitos TTL, la lógica 1 es equivalente a 3.5 volts y la lógica 0 es equivalente a tierra. Las salidas S del circuito dan el código equivalente exceso-3 del dígito de entrada BCD. Esta implementación requiere un paquete IC y cinco conexiones de alambre, sin incluir el alambrado de entrada y salida.

Propagación de acarreo

La adición de dos números binarios en paralelo implica que todos los bits del sumando y del adendo estén disponibles para computarse al mismo tiempo. Como en cualquier circuito combinacional, la señal debe propagarse a través de las compuertas antes de que esté disponible la salida de suma correcta en las terminales de salida. El tiempo total de propagación es igual al retardo de propagación de una compuerta típica

*Un ejemplo de un sumador completo de 4-bit es el IC tipo TTL 74283.

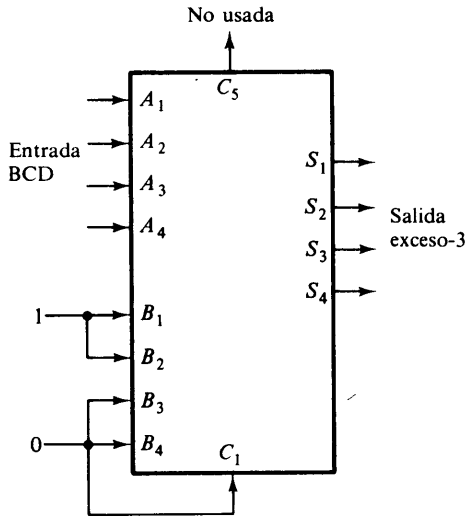


Figura 5-2 Convertidor de código BCD-a-exceso-3.

multiplicado por el número de niveles de compuertas en el circuito. El tiempo más largo de retardo de propagación en un sumador paralelo es el tiempo que toma el acarreo para propagarse a través de los sumadores completos. Ya que cada bit de la salida de suma depende del valor del acarreo de entrada, el valor de S_i en cualquier etapa dada del sumador estará en el estado estacionario de su valor final sólo después de que el acarreo de entrada a esa etapa se ha propagado. Considérese la salida S_4 en la Fig. 5-1. Las entradas A_4 y B_4 alcanzan un valor estacionario tan pronto como unas señales de entrada se aplican al sumador. Pero el acarreo de entrada C_4, no se asienta a su estado estacionario final hasta que esté disponible C_3 en su valor con estado estacionario. En forma similar, C_3 tiene que esperar a C_2 y así sucesivamente hasta C_1. Así que, sólo después de que el acarreo se propaga a través de todas las etapas, la última salida S_4 y el acarreo C_5 se establezcan a su valor final en estado estacionario.

El número de niveles de compuerta para la propagación del acarreo, puede encontrarse mediante el circuito del sumador completo. Este circuito se derivó en la Fig. 4-5 y vuelve a mostrarse en la Fig. 5-3 por motivos de comodidad. Las variables de

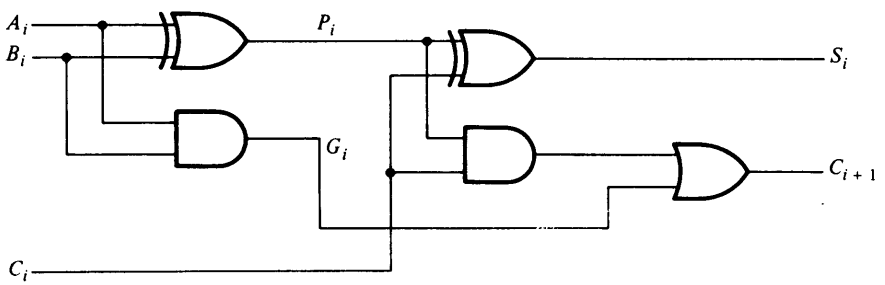


Figura 5-3 Circuito sumador completo.

entrada y salida usan el subíndice i para denotar una etapa típica en el sumador paralelo. Las señales en P_i y G_i se establecen en su valor de estado estacionario después de la propagación a través de sus respectivas compuertas. Estas dos señales son comunes a todos los sumadores completos y dependen sólo del sumando de entrada y de los bits de adendo. La señal del acarreo de entrada, C_i, al acarreo de salida, C_{i+1}, se propaga a través de una compuerta AND y una compuerta OR, lo cual constituye dos niveles de compuerta. Si hay tres o cuatro sumadores completos en el sumador paralelo, el acarreo de salida C_5, tendrá $2 \times 4 = 8$ niveles de compuerta desde C_1 hasta C_5. El tiempo total de propagación en el sumador sería el tiempo de propagación en un medio sumador más ocho niveles de compuerta. Para un sumador paralelo de n-bit, hay $2n$ niveles de compuerta para que un acarreo se propague.

El tiempo de propagación del acarreo, es un factor limitante en la velocidad con la cual se agregan dos números en paralelo. Aun cuando un sumador paralelo, o cualquier circuito combinacional, siempre tendrá algún valor en sus terminales de salida, las salidas no serán correctas a menos que se dé a las señales suficiente tiempo para propagarse a través de las compuertas conectadas desde las entradas hasta las salidas. Ya que todas las otras operaciones aritméticas se implementan por adiciones sucesivas, el tiempo consumido durante el proceso de adición es muy crítico. Una solución obvia para reducir el tiempo de retardo de propagación del acarreo, es emplear compuertas más rápidas con retardos reducidos. Pero los circuitos físicos tienen un límite de su capacidad. Otra solución es aumentar la complejidad del equipo en forma tal que se reduzca el tiempo de retardo del acarreo. Hay varias técnicas para reducir el tiempo de propagación del acarreo, en un sumador paralelo. La técnica de más amplia utilización emplea el principio de acarreo por anticipado y se describe a continuación.

Considérese el circuito del sumador completo que se muestra en la Fig. 5-3. Si se definen dos nuevas variables binarias:

$$P_i = A_i \oplus B_i$$
$$G_i = A_i B_i$$

la suma y acarreo de salida pueden expresarse como:

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_i C_i$$

G_i se denomina acarreo generado y produce un acarreo de salida cuando tanto A_i como B_i son uno, haciendo caso omiso del acarreo de entrada P_i se denomina acarreo propagado, ya que es el término asociado con la propagación acarreo de C_i a C_{i+1}.

Ahora se escribe la función booleana para el acarreo de salida, en cada etapa y se sustituye para cada C_i su valor mediante las ecuaciones previas:

$$C_2 = G_1 + P_1C_1$$
$$C_3 = G_2 + P_2C_2 = G_2 + P_2(G_1 + P_1C_1) = G_2 + P_2G_1 + P_2P_1C_1$$
$$C_4 = G_3 + P_3C_3 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1C_1$$

Ya que la función booleana para cada acarreo de salida se expresa en suma de productos, cada función puede implementarse con un nivel de compuertas AND seguido por una compuerta OR (por un NAND de nivel dos). Las tres funciones booleanas para C_2, C_3 y C_4 se implementan en el generador de acarreo por anticipado, que se muestra en la Fig. 5-4. Obsérvese que C_4 no tiene que esperar para que se propaguen C_3 y C_2; de hecho, C_4 de propaga al mismo tiempo que C_2 y C_3.*

La construcción de un sumador paralelo de 4-bit con un esquema de acarreo por anticipado, se muestra en la Fig. 5-5. Cada salida de suma requiere dos compuertas OR-excluyentes. La salida de la primer compuerta OR-excluyente genera la variable P_i, y la compuerta AND genera la variable G_i. Todas las P y G se generan en dos niveles de

*Un generador típico de acarreo por anticipado es el IC tipo 74182. Se implementa con compuertas AND-OR-INVERSORAS. tiene dos salidas, G y P, para generar $C_5 = G + PC_1$.

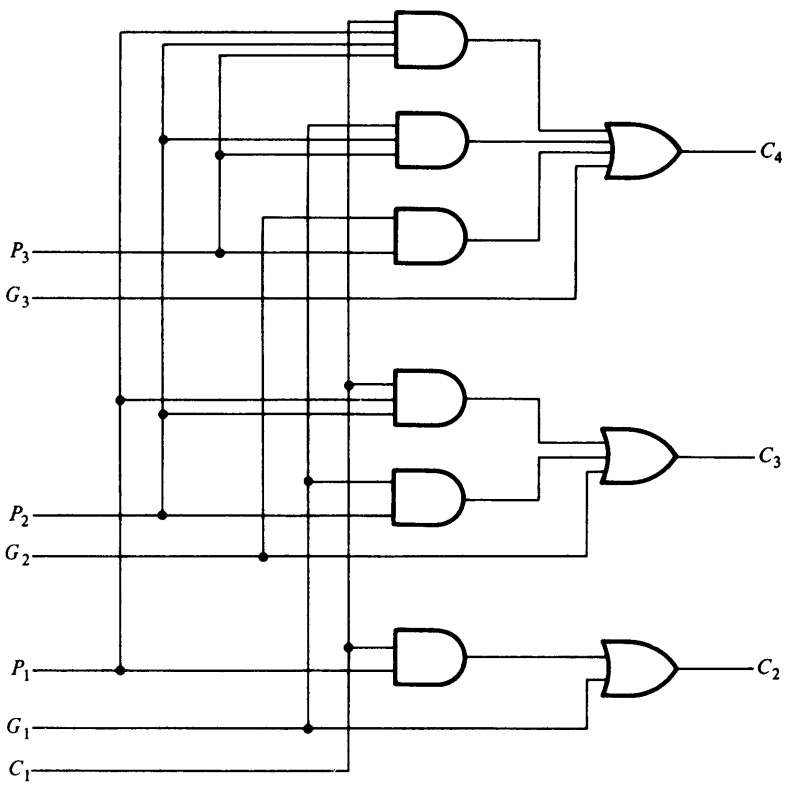


Figura 5-4 Diagrama lógico de un generador de acarreo anticipado.

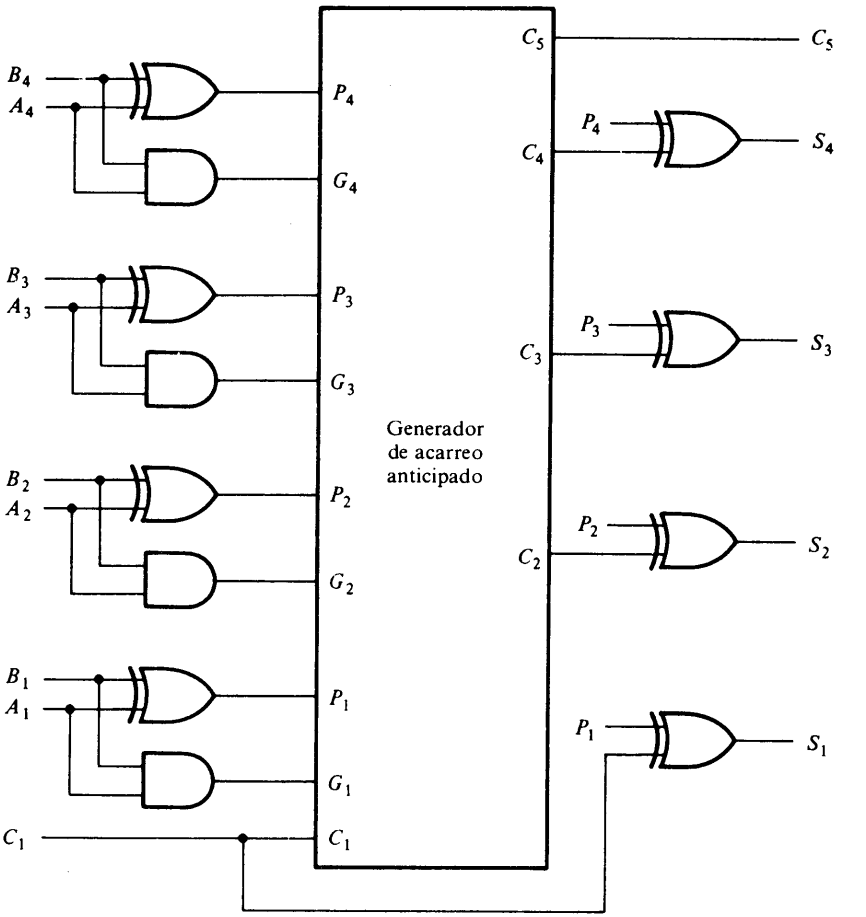


Figura 5-5 Sumadores completos de 4-bit con acarreo anticipado.

compuerta. Los acarreos se propagan a través del generador de acarreo por anticipado (similar al de la Fig. 5-4) y se aplican como entradas a la segunda compuerta OR-excluyente. Después de que las señales P y G se establecen en sus valores de estado estacionario, todos los acarreos de salida se generan después de un retardo de dos niveles de compuertas. Así que, las salidas S_2 a la S_4 tienen tiempos iguales de propagación. El circuito de dos niveles para el acarreo de salida C_5 no se muestra en la Fig. 5-4. Este circuito puede derivarse con facilidad por el método de ecuación-sustitución como se hizo antes (véase el problema 5-4).

5-3 SUMADOR DECIMAL

Las computadoras o calculadoras que realizan operaciones aritméticas directamente en el sistema de números decimales representan números decimales en forma de

código binario. Un sumador para una de dichas computadoras debe emplear circuitos aritméticos que acepten números decimales codificados y presenten resultados en el código aceptado. Para adición binaria, fue suficiente considerar un par de bits significativos a la vez, junto con un acarreo previo. Un sumador decimal requiere un mínimo de nueve entradas y cinco salidas, ya que se necesitan cuatro bits para codificar cada dígito decimal y el circuito debe tener un acarreo de entrada y un acarreo de salida. Por supuesto, hay una amplia variedad de circuitos sumadores decimales posibles, dependiendo del código que se utilice para representar los dígitos decimales.

El diseño de un circuito combinacional de nueve entradas y cinco salidas por el método clásico requiere una tabla de verdad con $2^9 = 512$ entradas. Muchas de las combinaciones de entrada son condiciones no importa, ya que cada entrada en código binario tiene seis combinaciones que son inválidas. Las funciones booleanas simplificadas para el circuito pueden obtenerse por un método tabular generado por computadora, y el resultado probablemente sería una conexión de compuertas que forman un patrón irregular. Un procedimiento alterno es sumar los números con circuitos sumadores completos, tomando en consideración el hecho de que seis combinaciones en cada entrada de 4-bit no se utilizan. La salida debe modificarse de modo que se generen sólo las combinaciones binarias que son combinaciones válidas del código decimal.

Sumador BCD

Considérese la adición aritmética de dos dígitos decimales en BCD, junto con un posible acarreo de una etapa anterior. Ya que cada dígito de entrada no excede 9, la suma de salida no puede ser mayor de $9 + 9 + 1 = 19$, donde el 1 en la suma es un acarreo de entrada. Supóngase que se aplican dos dígitos BCD a un sumador binario de 4-bit. El sumador formará la suma en binario y produce un resultado que puede variar de 0 a 19. Estos números binarios se listan en la Tabla 5-1 y se etiquetan con los símbolos K, Z_8, Z_4, Z_2 y Z_1. K es el acarreo y los subíndices bajo la letra Z representan los pesos de 8, 4, 2 y 1 que pueden asignarse a los cuatro bits en el código BCD. En la primera columna en la tabla se listan las sumas binarias como aparecen en las salidas de un sumador binario de 4-bit. La salida de suma de dos dígitos decimales debe representarse en BCD y debe aparecer en la forma que se lista en la segunda columna de la tabla. El problema es encontrar una regla simple por la cual pueda convertirse el número binario en la primera columna en la representación del dígito correcto BCD del número en la segunda columna.

Al examinar el contenido de la tabla, es evidente que cuando la suma binaria es igual o menor que 1001, el número correspondiente BCD es idéntico y, por tanto, no se necesita conversión. Cuando la suma binaria es mayor de 1001, se obtiene una representación BCD que no es válida. La adición del binario 6 (0110) a la suma binaria la convierte en la representación BCD correcta y también produce un acarreo de salida cuando se requiere.

El circuito lógico que detecta la corrección necesaria puede derivarse mediante las entradas de la tabla. Es obvio que se requiere una conexión cuando la suma binaria

TABLA 5-1 Derivación de un sumador BCD

Suma binaria					Suma BCD					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

tiene un acarreo de salida $K = 1$. Las otras seis combinaciones de 1010 a 1111 que necesitan una corrección tienen un 1 en la posición Z_8. Para distinguirlas de los binarios 1000 y 1001 que también tienen un 1 en la posición Z_8, se especifica además que Z_4 o bien Z_2 deben tener un 1. La condición para una corrección y un acarreo de salida pueden expresarse por la función booleana:

$$C = K + Z_8Z_4 + Z_8Z_2$$

cuando $C = 1$, es necesario agregar 0110 a la suma binaria y proporcionar un acarreo de salida para la siguiente etapa.

Un sumador BCD es un circuito que suma dos dígitos BCD en paralelo y produce una suma dígita también en BCD. Un sumador BCD debe incluir la lógica de corrección en su construcción interna. Para sumar 0110 a la suma binaria, se utiliza un segundo sumador binario de 4-bit como se muestra en la Fig. 5-6. Los dos dígitos decimales, junto con el acarreo de entrada, se suman primero en el sumador binario de 4-bit superior para producir la suma binaria. Cuando el acarreo de salida es igual a cero, no se agrega cosa alguna a la suma binaria. Cuando es igual a 1, se añade el binario 0110 a la suma binaria a través del sumador binario de 4-bit de abajo. El acarreo de salida generado por el sumador binario de abajo puede ignorarse, puesto que suministra información ya disponible en la salida terminal de acarreo.

El sumador BCD puede construirse con tres paquetes IC. Cada uno de los sumadores de 4-bits es una función MSI y las tres compuertas para la corrección lógica necesitan un paquete SSI. Sin embargo, el sumador BCD está disponible en un circuito MSI*. Para lograr retardos de propagación más cortos, un sumador MSI BCD incluye los circuitos necesarios para los acarreos por anticipado. El circuito sumador para la corrección no requiere los cuatro sumadores completos y este circuito puede optimizarse dentro del paquete IC.

Un sumador decimal paralelo que suma n dígitos decimales necesita n etapas sumadoras BCD. El acarreo de salida para una etapa, debe conectarse con el acarreo de entrada de la siguiente etapa de orden más alto.

5-4 COMPARADOR DE MAGNITUD

La comparación de dos números es una operación que determina si un número es mayor que, menor que o igual a otro número. Un comparador de magnitud es un

*El IC tipo TTL 82S83 es un sumador, BCD.

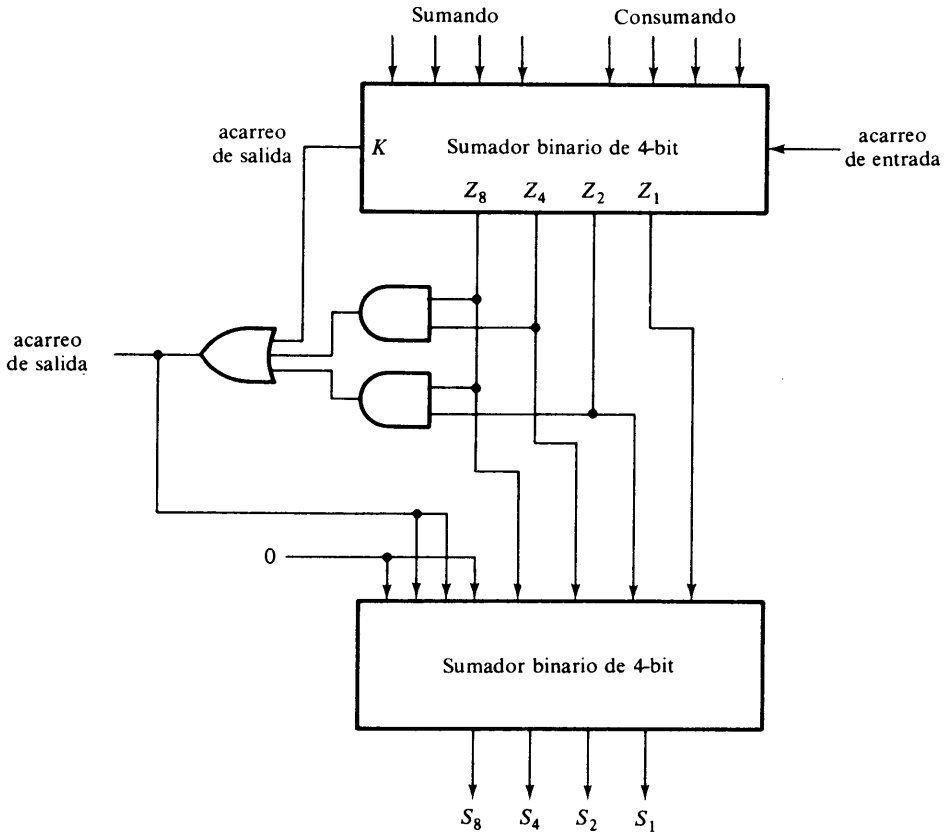


Figura 5-6 Diagrama de bloques de un sumador BCD.

circuito combinacional que compara dos números, A y B y determina sus magnitudes relativas. La salida de la comparación se especifica por tres variables binarias que indican si $A > B$, $A = B$ o $A < B$.

El circuito para comparar dos números de n-bit tiene 2^{2n} entradas en la tabla de verdad y llega a ser demasiado engorroso aun con $n = 3$. Por otra parte, como puede sospecharse, un circuito comparador posee cierto grado de regularidad. Las funciones digitales que poseen una regularidad inherente bien definida por lo común pueden diseñarse mediante un procedimiento algorítmico, si se encuentra que existe uno. Un algoritmo es un procedimiento que especifica un conjunto finito de pasos mediante los cuales, si se siguen, dan la solución a un problema. Aquí se ilustra este método derivando un algoritmo para el diseño de un comparador de magnitud de 4-bit.

El algoritmo es una aplicación directa del procedimiento que una persona utiliza para comparar las magnitudes relativas de dos números. Considérense dos números, A y B, con cuatro dígitos cada uno. Se escriben los coeficientes de números con significación decreciente como sigue:

$$A = A_3A_2A_1A_0$$
$$B = B_3B_2B_1B_0$$

en donde cada letra con subíndice representa uno de los dígitos en el número. Los dos números son iguales si todos los pares de dígitos significativos son iguales, esto es, si $A_3 = B_3$ y $A_2 = B_2$ y $A_1 = B_1$ y $A_0 = B_0$. Cuando los números son binarios, los dígitos son ya sea 1 o 0 y la relación de igualdad de cada par de bits puede expresarse en forma lógica con una función de equivalencia:

$$x_i = A_iB_i + A'_iB'_i \qquad i = 0, 1, 2, 3$$

en donde $x_i = 1$ sólo si el par de bits en la posición i son iguales, esto es, si ambos son 1 o ambos son 0.

La igualdad de dos números, A y B, se exhibe en un circuito combinacional por una salida de variable binaria que se designa con el símbolo $(A = B)$. Esta variable binaria es igual a 1 si los números de entrada, A y B, son iguales, y es igual a 0 de otra manera. Para que exista la condición de igualdad, todas las variables x_i deben ser iguales a 1. Esto dicta una operación AND de todas las variables:

$$(A = B) = x_3x_2x_1x_0$$

la variable *binaria* $(A = B)$ es igual a 1 sólo si todos los pares de dígitos de los dos números son iguales.

Para determinar si A es mayor o menor que B, se inspeccionan las magnitudes relativas de pares de dígitos significativos principiando desde la posición más significativa. Si los dos dígitos son iguales, el par de dígitos de la siguiente posición significativa más baja se comparan. Esta comparación continúa hasta que se alcanza un par de dígitos desiguales. Si el dígito correspondiente de A es 1 y el de B es 0, se concluye que $A > B$. Si el correspondiente dígito de A es 0 y el de B es 1, se tiene que $A < B$. La comparación secuencial puede expresarse en forma lógica por las siguientes dos funciones booleanas:

$$(A > B) = A_3B'_3 + x_3A_2B'_2 + x_3x_2A_1B'_1 + x_3x_2x_1A_0B'_0$$
$$(A < B) = A'_3B_3 + x_3A'_2B_2 + x_3x_2A'_1B_1 + x_3x_2x_1A'_0B_0$$

los símbolos $(A > B)$ y $(A < B)$ son variables *binarias* de salida que son iguales a 1 cuando $A > B$ o $A < B$, respectivamente.

La implementación de compuertas de las tres variables de salida que acaban de derivarse es más simple de lo que parece, ya que implica cierta cantidad de repetición. Las salidas "desiguales" pueden usar las mismas compuertas que se necesitan para generar la salida "igual". El diagrama lógico del comparador de magnitud de 4-bit se muestra en la Fig. 5-7*. Las cuatro salidas x se generan con circuitos de equivalencia (NOR-excluyente) y se aplican a una compuerta AND para dar la variable binaria de salida $(A = B)$. Las otras dos salidas usan las variables x para generar las funciones booleanas que se listaron antes. Esta es una implementación de nivel múltiple y, como se ve claramente, tiene un patrón regular. El procedimiento para obtener circuitos comparadores de magnitud para números binarios con más de cuatro bits debe ser obvio mediante este ejemplo. El mismo circuito puede utilizarse para comparar las magnitudes relativas de dos dígitos BCD.

5-5 DECODIFICADORES

Las cantidades discretas de información se representan en sistemas digitales con códigos binarios. Un código binario de n bits es capaz de representar hasta 2^n elementos distintos de información codificada. Un *decodificador* es un circuito combinacional que convierte información binaria de n líneas de entrada a un máximo de 2^n líneas únicas de salida. Si la información decodificada de n-bit tiene combinaciones no usadas o no importa, las salida del decodificador tendrá menos de 2^n salidas.

Los decodificadores que aquí se presentan se llaman decodificadores n-a-m líneas, donde $m \leq 2^n$. Su propósito es generar los 2^n (o menos) mintérminos de las n variables de entrada. El nombre *decodificador* también se utiliza junto con algunos convertidores de código como un decodificador BCD-a-siete-segmentos (véase el Problema 4-14).

Como un ejemplo, considérese el circuito decodificador de 3-a-8 líneas en la Fig. 5-8. Las tres entradas se decodifican en ocho salidas, cada salida representando uno de los mintérminos de las tres variables de entrada. Los tres inversores proporcionan el complemento de las entradas, y cada una de las ocho compuertas AND genera uno de los mintérminos. Una aplicación particular de este decodificador sería una conversión de binario-en-octal. Las variables de entrada pueden representar un número binario, y las salidas representan entonces los ocho dígitos en el sistema numérico octal. Sin embargo, un decodificador de 3-a-8 líneas puede usarse para decodificar cualquier código 3-bit para proporcionar ocho salidas, una para cada elemento del código.

*El tipo TTL 7485 es un comparador de magnitudes de 4-bit. Tiene tres entradas más para conectar comparadores en cascada (véase el Problema 5-4).

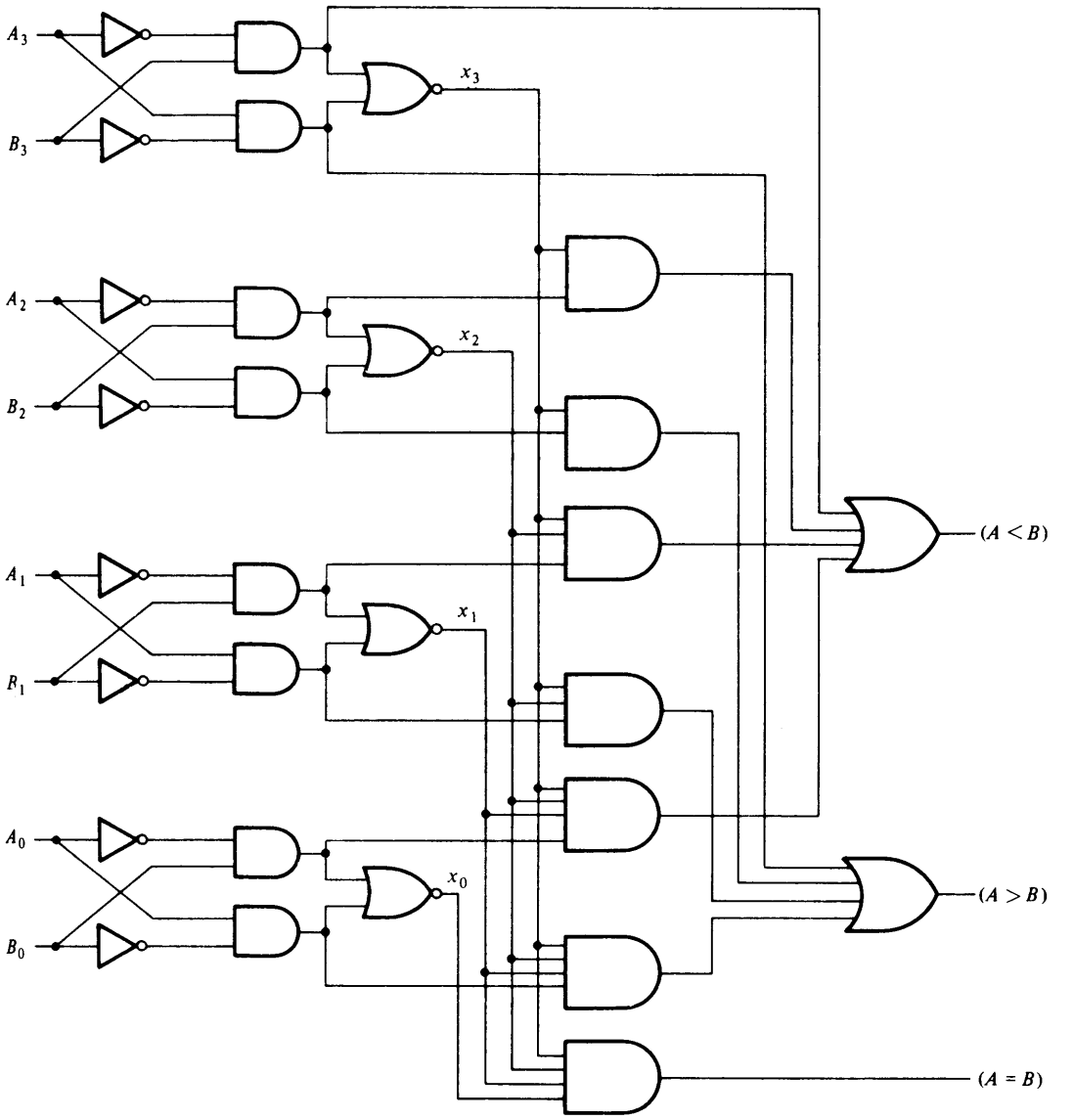


Figura 5-7 Comparador de magnitud de 4-bit.

La operación del decodificador puede aclararse aún más mediante sus relaciones de entrada-salida, que se listan en la Tabla 5-2. Obsérvese que las variables de salida son mutuamente excluyentes debido a que sólo una salida puede ser igual a 1 en cualquier momento. La línea de salida cuyo valor es igual a 1 representa el mintérmino equivalente del número binario presente disponible en las líneas de entrada.*

*El IC tipo 74138 es un decodificador 3-a-8 línea. Está construido con compuertas NAND. Las salidas son los complementos de los valores que se muestran en la Tabla 5-2.

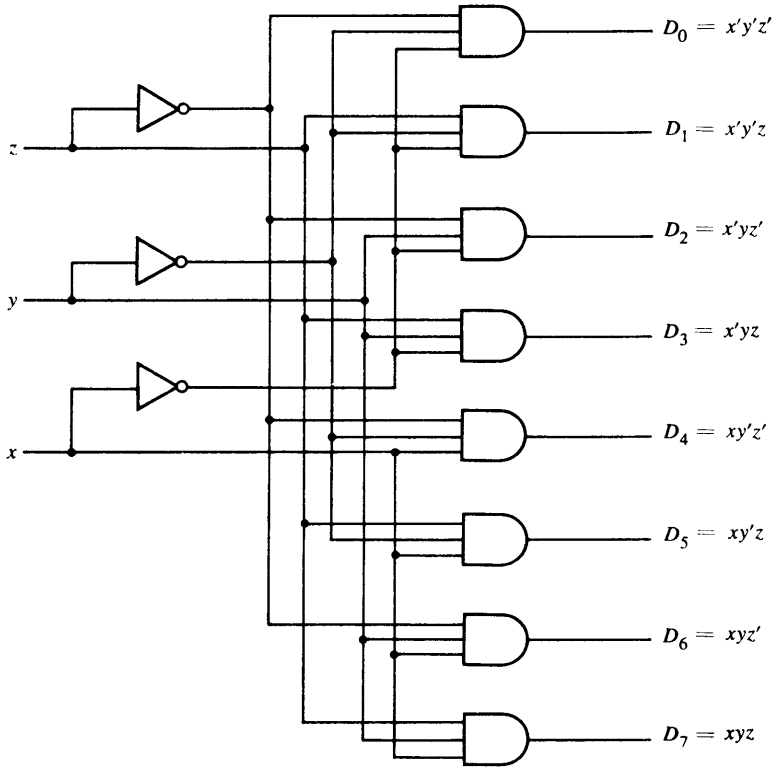


Figura 5-8 Un decodificador 3-a-8 línea.

TABLA 5-2 Tabla de verdad de un decodificador de 3-a-8 línea

Entradas			Salidas							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

EJEMPLO 5-2: Diseñe un decodificador BCD-a-decimal.

Los elementos de información en este caso son los diez dígitos decimales representados por el código BCD. El código mismo tiene cuatro bits. Por tanto, el decodificador debe tener cuatro entradas para aceptar el dígito codificado y diez salidas, una para cada dígito decimal. Esto dará una 4-línea a un decodificador BCD-a-decimal de 10-líneas.

En realidad no es necesario diseñar dicho decodificador porque puede encontrarse en la forma IC y como una función MSI. De cualquier manera, se diseñará por dos razones. Primero, permite obtener mejor comprensión de lo que se espera en esa función MSI. Segundo, es un buen ejemplo para demostrar las consecuencias prácticas de las condiciones no importa.

Ya que el circuito tiene diez salidas, sería necesario dibujar diez mapas para simplificar cada una de las funciones de salida. Hay seis condiciones no importa aquí y, debe tomarse en consideración cuando se simplifica cada una de las funciones de salida. En lugar de dibujar diez mapas, se dibuja sólo un mapa y se escribe cada una de las variables de salida, D_0 a D_9, en el interior del cuadro de su mintérmino correspondiente como se muestra en la Fig. 5-9. Seis combinaciones de entrada no ocurrirán nunca, de modo que se marcan sus cuadros de mintérminos correspondientes con X.

Es responsabilidad del diseñador decidir cómo tratar las condiciones no importa. Se supone que se ha decidido usarlas en tal modo para simplificar las funciones al número mínimo de literales. D_0 y D_1 no pueden combinarse con cualquiera de los mintérminos no importa. D_2 puede combinarse con el mintérmino no importa m_{10} para dar:

$$D_2 = x'yz'$$

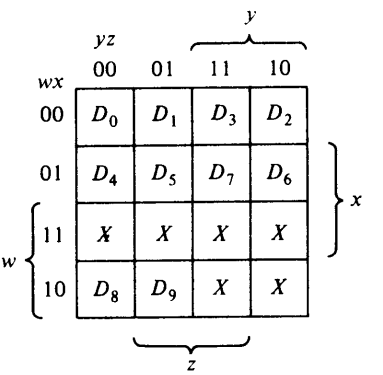


Figura 5-9 Mapa para simplificar un decodificador BCD-a-decimal.

El cuadro con D_9 puede combinarse con otros tres cuadros no importa para dar:

$$D_9 = wz$$

Utilizando los términos no importa para las otras salidas, se obtiene el circuito que se muestra en la Fig. 5-10. Por eso, los términos no importa causan una reducción en el número de entradas en la mayoría de las compuertas AND.

Un diseñador cuidadoso debe investigar el efecto de la minimización anterior. Aun cuando es cierto que bajo las condiciones normales de operación las seis combinaciones inválidas no ocurrirán nunca, ¿qué pasa si hay un mal funcionamiento y entonces ocurren? Un análisis del circuito en la Fig. 5-10 muestra que las seis combinaciones inválidas de entrada producirán salidas como se lista en la Tabla 5-3. El lector puede ver la tabla y decidir cuándo éste es un diseño bueno o malo.

Otra decisión razonable de diseño sería hacer todas las salidas iguales a 0 cuando ocurre una combinación inválida de entrada.* Esto requeriría diez compuertas AND

*El IC tipo 7442 es un decodificador BCD-a-decimal. Las salidas seleccionadas están en el estado 0, y todas las combinaciones inválidas dan una salida de todos los 1.

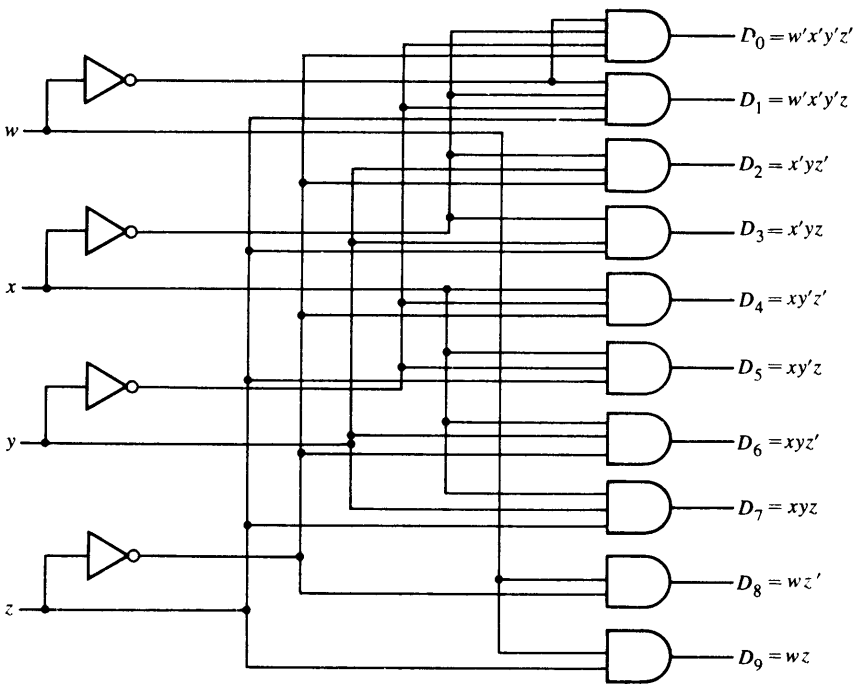


Figura 5-10 Decodificador BCD-a-decimal.

TABLA 5-3 Tabla de verdad parcial para el circuito de la Fig. 5-10

Entradas				Salidas									
w	x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9
1	0	1	0	0	0	1	0	0	0	0	0	1	0
1	0	1	1	0	0	0	1	0	0	0	0	0	1
1	1	0	0	0	0	0	0	1	0	0	0	1	0
1	1	0	1	0	0	0	0	0	1	0	0	0	1
1	1	1	0	0	0	0	0	0	0	1	0	1	0
1	1	1	1	0	0	0	0	0	0	0	1	0	1

de 4 entradas. Pueden considerarse otras posibilidades. En cualquier caso, no deben tratarse las condiciones no importa en forma indiscriminada, sino que debe intentarse investigar sus efectos una vez que el circuito esté en operación.

Implementación de lógica combinacional

Un decodificador proporciona los 2^n mintérminos de n variables de entrada. Ya que cualquier función booleana puede expresarse en la forma canónica de suma de mintérminos, puede utilizarse un decodificador para generar los mintérminos y una compuerta externa OR para formar la suma. De esta manera, cualquier circuito combinacional con n entradas y m salidas puede implementarse con un decodificador de n-a-2^n y m compuertas OR.

El procedimiento para implementar un circuito combinacional mediante un decofificador y compuertas OR requiere que las funciones booleanas para el circuito se expresen en sumas de mintérminos. Esta forma puede obtenerse con facilidad mediante la tabla de verdad o por la expansión de las funciones en su suma de mintérminos (véase la Sección 2-5). Se elige entonces un decodificador que genere todos los mintérminos de las n variables de entrada. Las entradas a cada compuerta OR se seleccionan de las salidas de decodificador de acuerdo con la lista de mintérminos en cada función.

EJEMPLO 5-3: Implemente un circuito adicionador completo con un decodificador y dos compuertas OR.

Mediante la tabla de verdad del sumador completo (sección 4-3), obtenga las funciones para este circuito combinacional en suma de mintérminos:

$$S(x, y, z) = \Sigma(1, 2, 4, 7)$$
$$C(x, y, z,) = \Sigma(3, 5, 6, 7)$$

Ya que hay tres entradas y un total de ocho mintérminos, necesita un decodificador de 3-a-8 líneas. La implementación se muestra en la Fig. 5-11. El decodificador genera los ocho mintérminos para x, y, z. La compuerta OR para la salida S forma la suma de mintérminos 1, 2, 4 y 7.

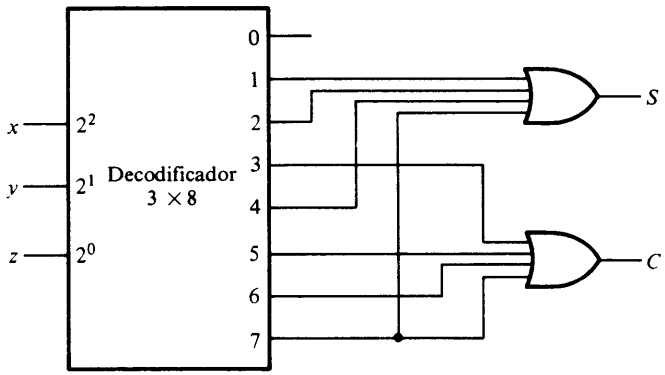


Figura 5-11 Implementación de un sumador completo con un decodificador.

La compuerta OR para la salida C forma la suma de mintérminos 3, 5, 6 y 7.

Una función con una lista larga de mintérminos requiere una compuerta OR con un gran número de entradas. Una función F que tenga una lista de k mintérminos puede expresarse en su forma complementaria F' con $2^n - k$ mintérminos. Si el número de mintérminos en una función es mayor de $2^n/2$, entonces F' puede expresarse con menos mintérminos que los requeridos para F. En tal caso, es ventajoso usar una compuerta NOR para la suma de mintérminos de F'. La salida de la compuerta NOR generará la salida normal F.

El método de decodificador puede utilizarse para implementar cualquier circuito combinacional. Sin embargo, su implementación debe compararse con todas las demás implementaciones posibles para determinar la mejor solución. En algunos casos este método puede proporcionar la mejor implementación, en especial si el circuito combinacional tiene muchas salidas y si cada función de salida (o su complemento) se expresa con un número pequeño de mintérminos.

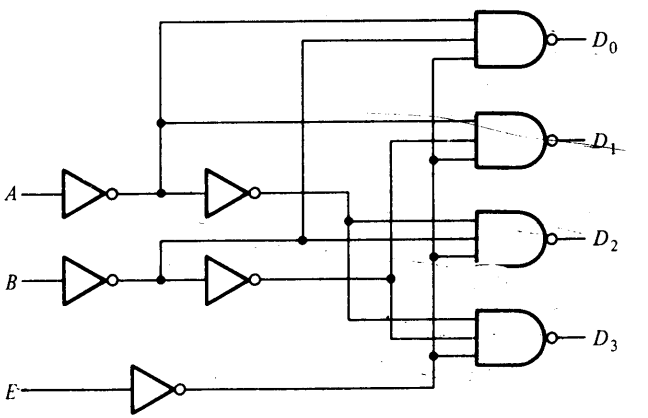
Demultiplexores

Algunos decodificadores IC se construyen con compuertas NAND. Ya que una compuerta NAND produce la operación AND con una salida invertida, se vuelve más económico generar los mintérminos del decodificador en su forma complementaria. La mayoría, si no todos, los decodificadores IC incluyen una o más entradas de capacitación para controlar la operación del circuito. Un decodificador de 2-a-4 líneas con una entrada de habilitación construida con compuertas NAND se muestra en la Fig. 5-2. Todas las salidas son iguales a 1 si la entrada de capacitación E es 1, sin importar los valores de entrada A y B. Cuando la entrada de habilitación es 0, el circuito opera como un decodificador con salidas complementadas. En la tabla de verdad se listan esas condiciones. Las X bajo A y B son condiciones despreocupadas. La operación normal del decodificador ocurre sólo con $E = 0$, y las salidas se seleccionan cuando están en el estado 0.

El diagrama de bloques del decodificador se muestra en la Fig. 5-3(a). El círculo pequeño en la entrada E indica que el decodificador está habilitado cuando $E = 0$. Los círculos pequeños en las salidas indican que todas las salidas están complementadas.

Un decodificador con una entrada de habilitación puede funcionar como un demultiplexor. Un demultiplexor es un circuito que recibe información en una sola línea y transmite esta información en una de 2^n líneas de salida posibles. La selección de una línea específica de salida está controlada por los valores bit de n líneas de selección. El decodificador en la Fig. 5-12 puede funcionar como un demultiplexor si la línea E se toma como una línea de entrada de datos y las líneas A y B se toman como las líneas de selección. Esto se muestra en la Fig. 5-13(b). La sola variable de entrada E tiene una trayectoria a todas las cuatro salidas, pero la información de entrada se dirige a una sola de las líneas de salida, como se especifica por el valor binario de las dos líneas de selección A y B. Esto puede verificarse mediante la tabla de verdad de este circuito, que se muestra en la Fig. 5-12(b). Por ejemplo, si las líneas de selección $AB = 10$, la salida D_2 será la misma que el valor de entrada E, en tanto que las otras salidas se mantienen en 1. Debido a que las operaciones de decodificador y demultiplexor se obtienen mediante el mismo circuito, un decodificador con una entrada de capacitación se refiere como un *decodificador/demultiplexor*. La entrada de capacitación es la que hace que el circuito se convierta en demultiplexor; el decodificador por sí mismo puede usar las compuertas AND, NAND o NOR.

Los circuitos decodificadores/demultiplexores pueden conectarse juntos para formar un circuito decodificador más grande. En la Fig. 5-14 se muestran dos decodificadores 3×8 con entradas de capacitación conectadas para formar un decodificador de 4×16. Cuando $w = 0$, el decodificador superior está capacitado y el otro está incapacitado. Las salidas del decodificador inferior son todas 0, y las ocho salidas superiores generan mintérminos 0000 a 0111. Cuando $w = 1$, las condiciones de capacitación están invertidas; las salidas del decodificador inferior generan mintér-



(a) Diagrama lógico

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

(b) Tabla de verdad

Figura 5-12 Un decodificador 2-a-4 línea con habilitación de entrada (E).

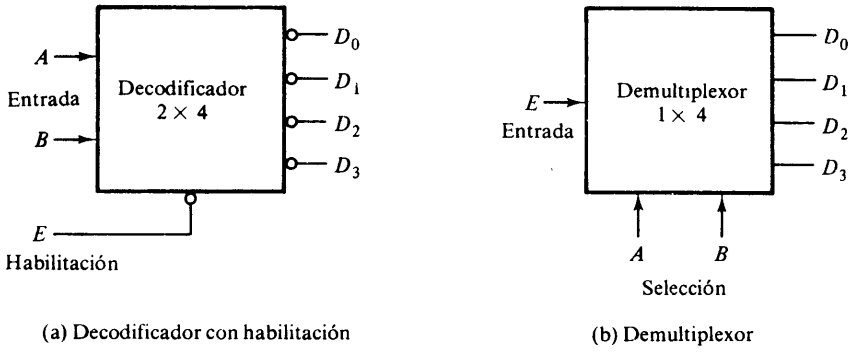


Figura 5-13 Diagramas de bloque para el circuito de la Fig. 5-12.

minos 1000 a 1111, en tanto las salidas del decodificador superior son todas 0. Este ejemplo demuestra la utilidad de las entradas de capacitación en los IC. En general, las líneas de capacitación son una característica conveniente para conectar dos o más paquetes IC para el propósito de hacer que crezca la función digital hasta una función similar con más entradas y salidas.

Codificadores

Un *codificador* es una función digital que produce una operación inversa de la de un decodificador. Un codificador tiene 2^n (o menos) líneas de entrada y n líneas de salida. Las líneas de salida generan el código binario para las 2^n variables de entrada. Un ejemplo de un codificador se muestra en la Fig. 5-15. El codificador de octal a binario consta de ocho entradas, una para cada uno de los ocho dígitos, y tres salidas que generan el número binario correspondiente. Se construye con compuertas OR cuyas

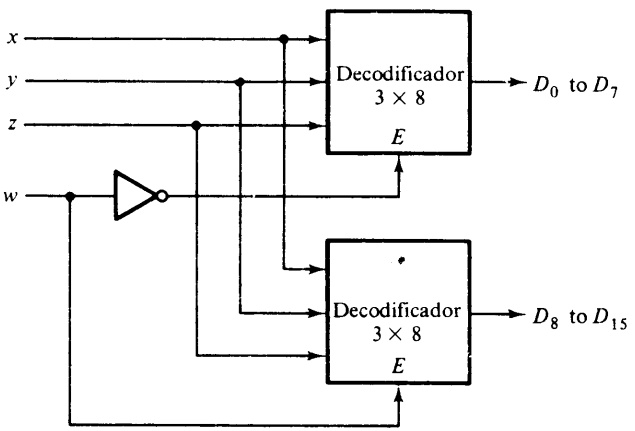


Figura 5-14 Un decodificador 4 × 16 construido con dos decodificadores 3 × 8.

salidas pueden determinarse mediante la tabla de verdad que se da en la Tabla 5-4. El bit de salida z de bajo orden es 1 si el dígito octal de entrada es impar. La salida y es 1 para los dígitos octales 2, 3, 6 o 7. La salida x es un 1 para los dígitos octales 4, 5, 6 o 7. Obsérvese que D_0 no está conectado a cualquier compuerta OR; las salidas binarias deben ser todas 0 en este caso. Una salida por completo 0 también se obtiene cuando todas las entradas son 0. Esta discrepancia puede resolverse proporcionando una salida más para indicar el hecho de que todas las entradas no son 0.

El codificador en la Fig. 5-15 supone que sólo una línea de entrada puede ser igual a 1 en cualquier momento; de otra manera el circuito carece de significado. Obsérvese que el circuito tiene ocho entradas y puede tener $2^8 = 256$ combinaciones posibles de entrada. Sólo ocho de esas combinaciones tienen algún significado. Las otras combinaciones de entrada son condiciones no importa.

Los codificadores de este tipo (Fig. 5-15) no están disponibles en paquetes IC, ya que pueden construirse fácilmente con compuertas OR. El tipo de codificador disponible en la forma IC se denomina *codificador de prioridad**. Estos codificadores establecen una prioridad de entrada para asegurar que sólo se codifique la línea de entrada de más alta prioridad. Así que, en la Tabla 5-4, si se da prioridad a una entrada con un número de subíndice más alto sobre uno con un número de subíndice más bajo, entonces si tanto D_2 y D_5 son en forma simultánea de lógica 1, la salida será 101 debido a que D_5 tiene una prioridad más alta sobre D_2. Por supuesto, la tabla de verdad de un codificador de prioridad es diferente a la que se muestra en la Tabla 5-4 (véase el Problema 5-21).

5-6 MULTIPLEXORES

La multiplexión significa transmitir un gran número de unidades de información sobre un número más pequeño de canales o líneas. Un *multiplexor digital* es un circuito combinacional que selecciona información binaria de una de muchas líneas de entrada

*Por ejemplo, el IC tipo 74148.

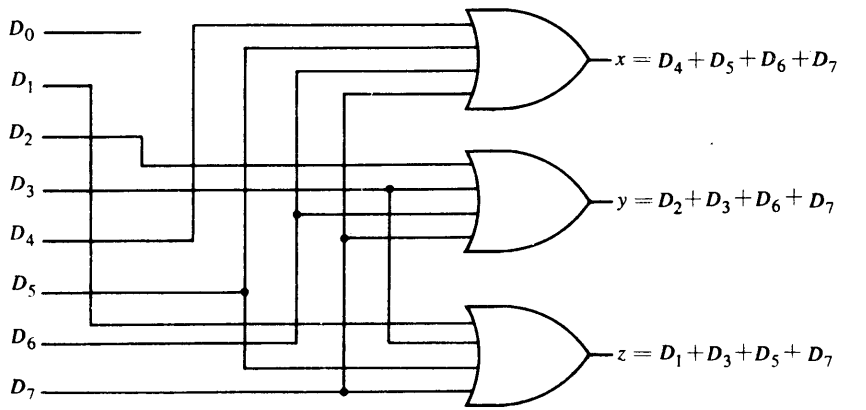


Figura 5-15 Codificador octal-a-binario.

TABLA 5-4 Tabla de verdad para un codificador de octal-a-binario

Entradas								Salidas		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

y la dirige a una sola línea de salida. La selección de una línea particular de entrada está controlada por un conjunto de líneas de selección. En forma normal, hay 2^n líneas de entrada y n líneas de selección cuyas combinaciones bit determinan cuál entrada se selecciona.

Un multiplexor de 4-líneas a 1-línea se muestra en la Fig. 5-16. Cada una de las cuatro líneas de entrada, I_0 a I_3, se aplica a una entrada de una compuerta AND. Las líneas de selección s_1 y s_0 se decodifican para seleccionar una compuerta AND particular. La tabla de función en la figura lista la trayectoria de entrada-a-salida para cada posible combinación de bit de las líneas de selección. Cuando esta función MSI se utiliza en el diseño de un sistema digital, se representa en la forma de diagrama de bloques como se muestra en la Fig. 5-16(c). Para demostrar la operación del circuito, se considera el caso cuando $s_1s_0 = 10$. La compuerta AND asociada con la entrada I_2 tiene dos de sus entradas iguales a 1 y la tercera entrada se conecta a I_2. Las otras tres compuertas AND tienen cuando menos una entrada igual a 0, lo cual hace que su salida sea igual a 0. La salida de la compuerta OR ahora es igual al valor de I_2, proporcionando de esta manera una trayectoria desde la entrada seleccionada a la salida. Un multiplexor también se conoce como *selector de datos,* ya que selecciona una de muchas entradas y dirige la información binaria a la línea de salida.

Las compuertas AND e inversores en el multiplexor se asemejan a un circuito decodificador y, por supuesto, decodifican la selección de líneas de entrada. En general, un multiplexor de 2^n-a-1 líneas se construye mediante un decodificador n-a-2^n agregándolo a 2^n líneas de entrada, una a cada compuerta AND. Las salidas de las compuertas AND se aplican a una sola compuerta OR para proporcionar la salida de 1-línea. El tamaño de un multiplexor se especifica por el número 2^n de sus líneas de entrada y la única línea de salida. Entonces se implica que también contiene n líneas de selección. Un multiplexor con frecuencia se abrevia como MUX.

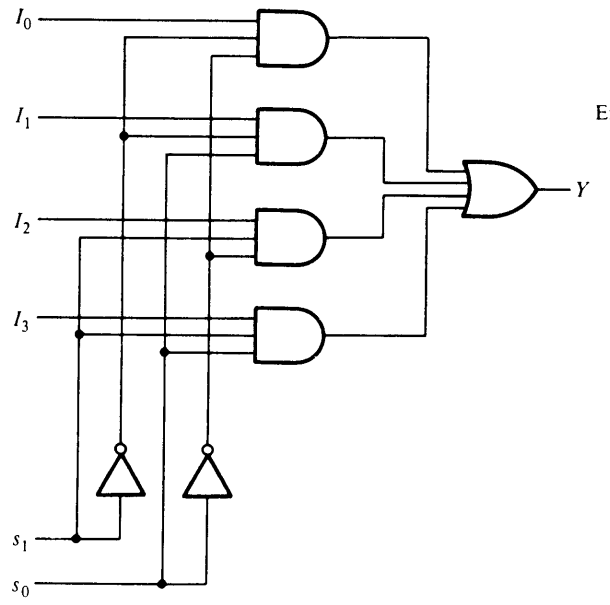
Como en los decodificadores, los multiplexores IC pueden tener una entrada *de habilitación* para controlar la operación de la unidad. Cuando la entrada de habilitación se encuentra en un estado binario dado, las salidas están inhabilitadas y cuando está en el otro estado (el estado de habilitación), el circuito funciona como un

multiplexor normal. La entrada de habilitación (algunas veces denominada estroboscopio) puede utilizarse para expander dos o más multiplexores IC a un multiplexor digital con un número más grande de salidas.

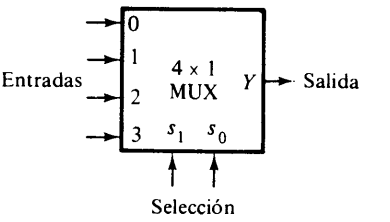
En algunos casos, dos o más multiplexores se encapsulan en un paquete IC. Las entradas de selección y de habilitación en una unidad múltiple de IC pueden ser comunes a todos los multiplexores. Como una ilustración, se muestra en la Fig. 5-17* un multiplexor IC cuádruple 2-líneas a 1-línea. Tiene cuatro multiplexores, cada uno capaz de seleccionar una de las dos líneas de entrada. La salida I_1 puede seleccionarse para que sea igual a A_1 o bien B_1. En forma similar, la salida I_2 puede tener el valor de A_2 o B_2 y así sucesivamente. Una línea de selección de entrada, S, es suficiente para seleccionar una de dos líneas en todos los cuatro multiplexores. La entrada de control E habilita los multiplexores en el estado 0 y los inhabilita en el estado 1. Aun cuando el circuito contiene cuatro multiplexores, puede considerarse como un circuito que selecciona una en un par de cuatro líneas de entrada. Como se muestra en la tabla de función la unidad se selecciona cuando $E = 0$. Entonces, si $S = 0$, las cuatro entradas A tienen una trayectoria a las salidas. Por otra parte, si $S = 1$, las cuatro entradas B se seleccionan. Las salidas tendrán todas 0 cuando $E = 1$, sin importar el valor de S.

El multiplexor es una función MSI muy útil y tiene multitud de aplicaciones. Se usa para conectar dos o más fuentes a un solo destino entre unidades computadoras, y

*Este multiplexor es similar al IC tipo 74157.



(a) Diagrama lógico



(c) Diagrama de bloque

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b) Tabla de función

Figura 5-16 Un multiplexor de 4-a-1 línea.

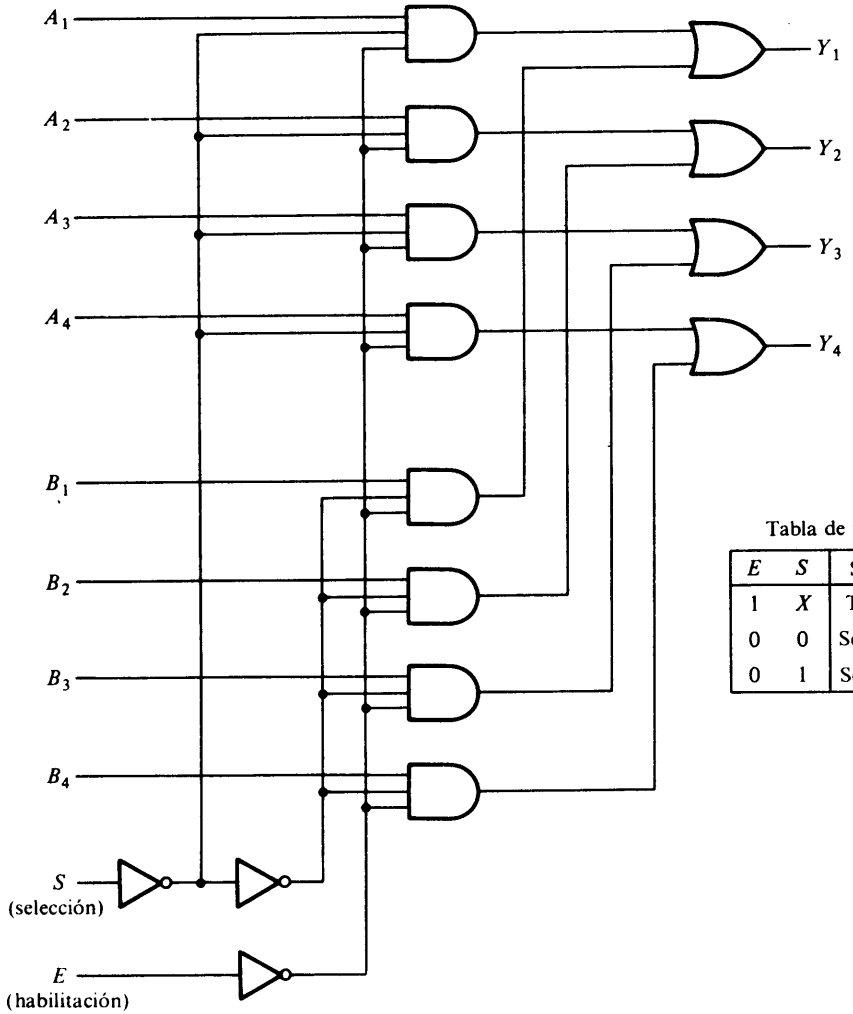


Tabla de función

E	S	Salida Y
1	X	Todos 0
0	0	Selección A
0	1	Selección B

Figura 5-17 Multiplexores cuádruples de 2-a-1 línea.

es útil para construir un sistema de bus común. Este y otros usos del multiplexor se exponen en los últimos capítulos junto con sus aplicaciones particulares. Aquí se demuestran las propiedades generales de este dispositivo y se muestra que puede utilizarse para implementar cualquier función booleana.

Implementación de una función booleana

Se mostró en la sección anterior que un decodificador puede usarse para implementar una función booleana empleando una compuerta OR externa. Una rápida referencia al multiplexor en la Fig. 5-16 revela que en esencia es un decodificador con la compuerta OR con la que ya se cuenta. Los mintérminos de salida del decodificador

que se seleccionarán pueden controlarse con las líneas de entrada. Los mintérminos que van a incluirse con la función que se está implementando se escogen haciendo que sus líneas de entrada correspondientes sean iguales a 1, los mintérminos que no se incluyen en la función se inhabilitan haciendo sus líneas de entrada iguales a 0. Esto proporciona un método para implementar cualquier función booleana de n variables con un multiplexor 2^n-a-1. Sin embargo, es posible hacer esto de mejor forma.

Si se tiene una función booleana de $n + 1$ variables, se toman n de esas variables y se conectan a las líneas de selección de un multiplexor. La única variable remanente de la función se utiliza para las entradas del multiplexor. Si A es esta única variable, las entradas del multiplexor se eligen de manera que sean A o bien A', o 1 o bien 0. Por el uso juicioso de estos cuatro valores para las entradas y por la conexión de otras variables a las líneas de selección, puede implementarse cualquier función booleana con un multiplexor. En esta forma es posible generar cualquier función de $n + 1$ variables con un multiplexor de 2^n-a-1.

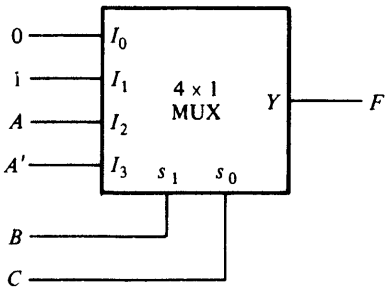
Para demostrar este procedimiento con un ejemplo concreto, considérese la función de tres variables:

$$F(A, B, C) = \Sigma(1, 3, 5, 6)$$

La función puede implementarse con un multiplexor de 4-a-1 como se muestra en la Fig. 5-18. Dos de las variables, B y C, se aplican a las líneas de selección en ese orden, esto es, B se conecta a s_1 y C a s_0. Las entradas del multiplexor son 0, 1, A y A'. Cuando $BC = 00$, la salida es $F = 0$, ya que $I_0 = 0$. En consecuencia, tanto el mintérmino $m_0 = A'B'C'$ como el $m_4 = AB'C'$ producen una salida 0, ya que la salida es 0 cuando $BC = 00$ sin importar el valor de A. Cuando $BC = 01$, la salida es $F = 1$, ya que $I_1 = 1$. Entonces, tanto el mintérmino $m_1 = A'B'C$ como el $m_5 = AB'C$ producen una salida 1, ya que la salida es 1 cuando $BC = 01$ sin importar el valor de A. Cuando $BC = 10$, la entrada I_2 se selecciona. Ya que A está conectada a esta entrada, la salida será igual a 1 sólo para el mintérmino $m_6 = ABC'$, pero no para el mintérmino $m_2 = A'BC'$, porque cuando $A' = 1$, entonces $A = 0$, y ya que $I_2 = 0$, se tiene $F = 0$. Por último, cuando $BC = 11$, la entrada I_3 se selecciona. Ya que A' está conectada a esta entrada, la salida será igual a 1 sólo para el mintérmino $m_3 = A'BC$, pero no para $m_7 = ABC$. Esta información se resume en la Fig. 5-18(b), la cual es la tabla de verdad de la función que se desea implementar.

La exposición anterior muestra por análisis que el multiplexor implementa la función requerida. Ahora se presenta un procedimiento general para implementar cualquier función booleana de n variables con un multiplexor de 2^{n-1}-a-1.

Primero, se expresa la función en su forma de suma de mintérminos. Se supone que la secuencia ordenada de las variables elegidas para los mintérminos es $ABCD$ ..., en donde A es la variable más a la izquierda en la secuencia ordenada de las n variables y BCD... son las $n-1$ remanentes. Se conectan las $n-1$ variables a las líneas de selección del multiplexor, con B conectada a la línea de selección de orden más alto, C a la siguiente línea de selección más baja y así sucesivamente descendiendo hasta la última variable, la cual está conectada con la línea de selección de orden más bajo s_0. Considérese ahora la variable única A. Ya que esta variable está en la posición de orden más alto en la secuencia de variables, se complementará en mintérminos 0 a



(a) Implementación de multiplexores

Mintérmino	A	B	C	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

(b) Tabla de verdad

	I_0	I_1	I_2	I_3
A'	0	①	2	③
A	4	⑤	⑥	7
	0	1	A	A'

(c) Tabla de implementación

Figura 5-18 Implementación de $F(A, B, C) = (1, 3, 5, 6)$ con un multiplexor.

$(2^n/2) - 1$ los cuales comprenden la primera mitad en la lista de mintérminos. La segunda mitad de los mintérminos tendrá su variable A sin complementar. Para una función de tres variables, A, B, C, se tienen ocho mintérminos. La variable A se complementa en mintérminos 0 a 3 y está sin complementar en los mintérminos de 4 a 7.

Se listan las entradas del multiplexor y bajo ellas se listan todos los mintérminos en dos renglones. En el primer renglón se listan todos los mintérminos donde A está complementada, en el segundo renglón todos los mintérminos con A sin complementar, como se muestra en la Fig. 5-18(c). Se encierran dentro de un círculo todos los mintérminos de la función y se inspecciona por separado cada columna.

Se listan las entradas del multiplexor y bajo ellas se listan todos los mintérminos en dos renglones. En el primer renglón se listan todos los mintérminos donde A está complementada, en el segundo renglón todos los mintérminos con A sin complementar, como se muestra en la Fig. 5-18(c). Se encierran dentro de un círculo todos los mintérminos de la función y se inspecciona por separado cada columna.

Si los dos mintérminos en una columna no están dentro de un círculo, aplíquese 0 a la entrada correspondiente del multiplexor.

Si los dos mintérminos están dentro de un círculo, se aplica 1 a la entrada correspondiente del multiplexor.

Si el mintérmino inferior está dentro de un círculo y el superior no lo está, se aplica A a la entrada del multiplexor correspondiente.

Si el mintérmino superior está dentro de un círculo y el inferior no lo está, se aplica A' a la entrada del multiplexor correspondiente.

Este procedimiento resulta de las condiciones establecidas durante el análisis previo.

En la Fig. 5-18(c) se muestra la tabla de implementación para la función booleana:

$$F(A, B, C) = \Sigma(1, 3, 5, 6)$$

mediante la cual se obtienen las conexiones del multiplexor en la Fig. 5-18(a). Obsérvese que B debe conectarse a s_1 y C a s_0.

No es necesario escoger la variable de la extrema izquierda en la secuencia ordenada de una lista de variables para las entradas de multiplexor. De hecho, puede elegirse cualquiera de las variables para las entradas del multiplexor, siempre que se multiplique la tabla de implementación del multiplexor. Supóngase que se desea implementar la misma función con un multiplexor, pero utilizando las variables A y B para las líneas de selección s_1 y s_0 y la variable C para las entradas del multiplexor. La variable C se complementa en los mintérminos numerados con pares y sin complementos en los mintérminos numerados impares, ya que es la última variable en la secuencia de las variables listadas. El arreglo de los dos renglones de mintérminos en este caso debe ser como se muestra en la Fig. 5-19(a). Por los mintérminos encerrados dentro de círculos de la función y utilizando las reglas establecidas con anterioridad, se obtiene la conexión del multiplexor para implementar la función como en la Fig. 5-19(b).

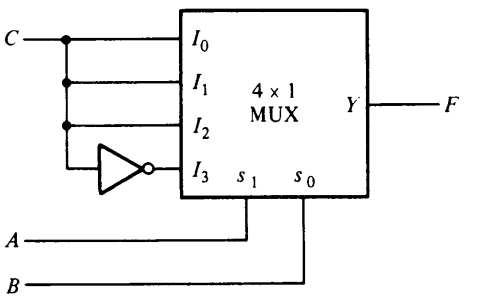
En una forma similar, es posible usar cualquier variable única de la función para utilizarse en las entradas del multiplexor. Pueden formularse diversas combinaciones para implementar una función booleana con multiplexores. Todas las variables de entrada, excepto una, se aplican a las líneas de selección. La única variable remanente, o su complemento, o 0 o bien 1, se aplica entonces a las entradas del multiplexor.

EJEMPLO 5-4: Implemente la siguiente función con un multiplexor:

$$F(A, B, C, D) = \Sigma(0, 1, 3, 4, 8, 9, 15)$$

	I_0	I_1	I_2	I_3
C'	0	2	4	⑥
C	①	③	⑤	7
	C	C	C	C'

(a) Tabla de implementación



(b) Conexión del multiplexor

Figura 5-19 Implementación alterna para $F(A, B, C) = \Sigma(1, 3, 5, 6)$.

Esta es una función de cuatro variables y, así es que, necesita un multiplexor con tres líneas de selección de ocho entradas. Elija aplicar las variables *B, C* y *D* a las líneas de selección. La tabla de implementación queda entonces como se muestra en la Fig. 5-20. La primera mitad de los mintérminos está asociada con *A'* y la segunda mitad con *A*. Debido a que están encerrados dentro de un círculo los mintérminos de la función y aplicando las reglas para encontrar los valores de las entradas del multiplexor, el lector obtiene la implementación que se muestra.

Ahora se compara el método de multiplexor con el método decodificador para implementar circuitos combinacionales. El método decodificador requiere una compuerta OR para cada función de salida, pero sólo se necesita un decodificador para generar todos los mintérminos. El método multiplexor usa unidades de tamaño más pequeño, pero requiere un multiplexor para cada función de salida. Puede parecer razonable suponer que con circuitos combinacionales con un número pequeño de salidas deberían implementarse con multiplexores. Los circuitos combinacionales con muchas funciones de salida probablemente utilizarían menos IC con el método decodificador.

Aunque pueden usarse multiplexores y decodificadores en la implementación de circuitos combinacionales, debe tenerse en cuenta que los decodificadores tienen más uso en la decodificación de información binaria y los multiplexores tienen más uso para formar una trayectoria seleccionada entre fuentes múltiples y un solo destino. Deben considerarse cuando se diseñan circuitos combinacionales especiales que no están disponibles en otra forma que como funciones MSI. Para circuitos combinacionales grandes con múltiples entradas y salidas, hay un componente IC más adecuado y se presenta en la siguiente sección.

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
A'	⓪	①	2	③	④	5	6	7
A	⑧	⑨	10	11	12	13	14	⑮
	1	1	0	A'	A'	0	0	A

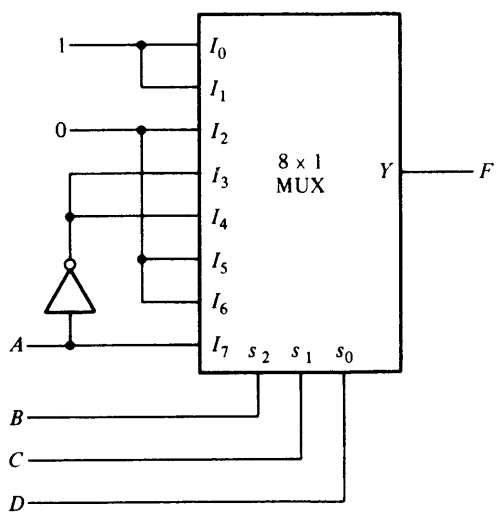


Figura 5-20 Implementación de $F(A, B, C, D) = \Sigma(0, 1, 3, 4, 8, 9, 15)$.

5-7 MEMORIA DE SOLO LECTURA (ROM)

En la Sección 5-5 se vio que un decodificador genera los 2^n mintérminos de las n variables de entrada. Por la inserción de compuertas OR para la suma de los mintérminos de las funciones booleanas, se tuvo capacidad de generar cualquier circuito combinacional deseado. Una memoria de sólo lectura (ROM) es un dispositivo que incluye tanto el decodificador como las compuertas OR dentro de un solo paquete IC. Las conexiones entre las salidas del decodificador y las entradas de las compuertas OR pueden especificarse para cada configuración particular por la "programación" de la ROM. Con mucha frecuencia se usa la ROM para implementar un circuito combinacional complejo en un paquete IC y, en ese caso, elimina todos los alambres de interconexión.

Una ROM en forma esencial es un dispositivo de memoria (o almacén) en el cual se almacena un conjunto fijo de información binaria. La información binaria primero debe especificarla el usuario y entonces se inserta en la unidad para formar el patrón requerido de interconexión. Las ROM se obtienen con eslabones internos especiales que pueden fusionarse o romperse. La interconexión deseada para una aplicación particular requiere que se fusionen ciertos eslabones para formar las trayectorias del circuito necesario. Una vez que se establece un patrón para una ROM, permanece fijo aun cuando se enciende la potencia y se apaga otra vez.

En la Fig. 5-21 se muestra un diagrama de bloque de una ROM. Consta de n líneas de entrada y m líneas de salida. Cada combinación de bits de las variables de entrada se denomina *dirección.* Cada combinación de bits que sale de las líneas de salida se conoce como *palabra.* El número de bits por palabra es igual al número de líneas de salida m. Una dirección es en esencia un número binario que denota uno de los mintérminos de las n variables. El número de direcciones distintas posibles con n variables de entrada es 2^n. Una palabra de salida puede seleccionarse por una dirección única, ya que hay 2^n direcciones distintas en una ROM, hay 2^n palabras distintas que se dice que se almacenan en la unidad. La palabra disponible en las líneas de salida en cualquier momento dado depende del valor de dirección aplicado a las líneas de entrada. Una ROM se caracteriza por el número de palabras 2^n y el número de bits por palabra m. Esta terminología se usa por la similitud entre la memoria de sólo lectura y la memoria de lectura escritura, que se presenta en la Sección 7-7.

Considérese una ROM de 32×8. La unidad consta de 32 palabras de 8 bits cada una. Esto significa que hay ocho líneas de salida y que hay 32 palabras distintas almacenadas en la unidad, cada una de las cuales puede aplicarse a las líneas de salida. La palabra particular seleccionada que al presente está disponible en las líneas de salida está determinada por las cinco líneas de entrada. Sólo hay cinco salidas en una ROM de 32×8, ya que $2^5 = 32$, y con cinco variables pueden especificarse 32 direcciones o mintérminos. Para cada selección de entrada, hay una palabra única seleccionada. De este modo, si la dirección de entrada es 00000, se selecciona la palabra número 0 y aparece en las líneas de salida. Si la dirección de entrada es 11111, se elige la palabra número 31 y se aplica a las líneas de salida. Mientras tanto, hay otras 30 direcciones que pueden seleccionar las otras 30 palabras.

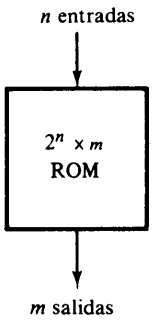


Figura 5-21 Diagrama de bloques de una ROM.

El número de palabras con dirección en una ROM está determinado por el hecho de que n líneas de entrada se necesitan para especificar 2^n palabras. Una ROM algunas veces se especifica por el número total de bits que contiene, el cual es $2^n \times m$. Por ejemplo, una ROM de 2048-bit puede organizarse en 512 palabras de 4 bits cada una. Esto significa que la unidad tiene 4 líneas de salida y 9 líneas de entrada para especificar $2^9 = 512$ palabras. El número total de bits almacenados en la unidad es $512 \times 4 = 2048$.

En su interior, la ROM es un circuito combinacional con compuertas AND conectadas como un decodificador y un número de compuertas OR igual al número de salidas de la unidad. En la Fig. 5-22 se muestra la construcción lógica interna de una

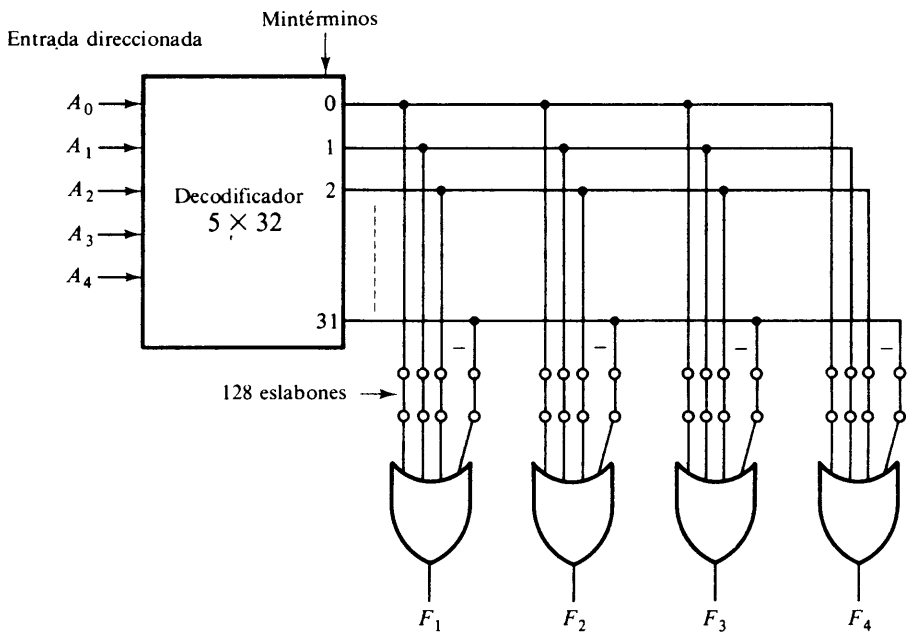


Figura 5-22 Construcción lógica de una ROM de 32 × 4.

ROM de 32 × 4. Las cinco variables de entrada están decodificadas en 32 líneas mediante 32 compuertas AND y 5 inversores. Cada salida del decodificador representa uno de los mintérminos en una función de cinco variables. Cada una de las 32 direcciones selecciona una y sólo una salida del decodificador. La dirección es un número de 5-bit aplicado a las entradas, y el mintérmino seleccionado de salida del decodificador es el marcado con el número decimal equivalente. Las 32 salidas del decodificador están conectadas a través de eslabones a cada compuerta OR. Sólo cuatro de estos eslabones se muestran en el diagrama, pero en la realidad cada compuerta OR tiene 32 entradas y cada entrada va a través de un eslabón que puede romperse según se desee.

La ROM es una implementación de dos niveles en la forma de suma de mintérminos. No tiene que ser una de tipo AND-OR, pero puede ser cualquier otra implementación posible de mintérminos en dos niveles. El segundo nivel por lo común es una conexión de lógica alambrada (véase la Sección 3-7) para facilitar la fusión de eslabones.

Las ROM tienen muchas aplicaciones importantes en el diseño de sistemas de computadoras digitales. Su uso para circuitos combinacionales complejos es sólo una de estas aplicaciones. En otras secciones de este libro se presentan otros usos de la ROM junto con sus aplicaciones particulares.

Implementación de lógica combinacional

Mediante el diagrama lógico de la ROM, es claro que cada salida proporciona la suma de todos los mintérminos de las n variables de entrada. Recuérdese que cualquier función booleana puede expresarse en la forma de suma de mintérminos. Por la ruptura de los eslabones de los mintérminos que no se incluyen en la función, cada salida ROM puede hacerse que represente la función booleana de una de las variables de entrada en el circuito combinacional. Para un circuito combinacional de n-entrada, m-salida, se necesita una ROM $2^n \times m$. La apertura de los eslabones se conoce como *programación* de la ROM. El diseñador necesita especificar tan solo una tabla de programa de la ROM que da la información para las trayectorias requeridas en la ROM. La programación real es un procedimiento de hardware que sigue las especificaciones que se listan en la tabla del programa.

Se clarifica el proceso con un ejemplo específico. La tabla de verdad en la Fig. 5-23(a) especifica un circuito combinacional con dos entradas y dos salidas. Las funciones Booleanas pueden expresarse en forma de mintérminos:

$$F_1(A_1, A_0) = \Sigma(1, 2, 3)$$
$$F_2(A_1, A_0) = \Sigma(0, 2)$$

Cuando se implementa un circuito combinacional mediante una ROM, las funciones deben expresarse en suma de mintérminos, o mucho mejor en una tabla de verdad. Si las funciones de salida se simplifican, se encuentra que el circuito necesita sólo una compuerta OR y un invertidor. En forma obvia, este es un circuito combinacional demasiado simple para implementarse con una ROM. La ventaja de una ROM se hace

patente en los circuitos combinacionales complejos. Este ejemplo demuestra en forma simple el procedimiento y no debe considerarse en una situación práctica.

La ROM que implementa al circuito combinacional debe tener dos entradas y dos salidas; de modo que su tamaño debe ser 4×2. En la Fig. 5-23(b) se muestra la construcción interna de dicha ROM. Ahora es necesario determinar cuál de los ocho eslabones disponibles debe romperse y cuáles deben dejarse colocados. Esto puede hacerse en forma fácil mediante las funciones de salida que se listan en la tabla de verdad. Los mintérminos que especifican una salida de 0 no deben tener una trayectoria a la salida a través de la compuerta OR. Siendo así, para este caso particular, la tabla de verdad muestra tres números 0, y sus correspondientes eslabones a las compuertas OR deben eliminarse. Es obvio que debe suponerse aquí que una entrada abierta a una compuerta OR se comporta como una entrada 0.

Algunas unidades ROM están disponibles con un inversor después de cada una de las compuertas OR y, en consecuencia, se especifican como que tienen en forma inicial todas sus salidas 0. El procedimiento de programación en dicha ROM requiere que se abran los eslabones en las trayectorias a los mintérminos (o direcciones) que especifican una salida de 1 en la tabla de verdad. La salida de la compuerta OR entonces generará el complemento de la función, pero el inversor colocado después de la compuerta OR complementa la función una vez más para proporcionar la salida normal. Esto se muestra en la ROM de la Fig. 5-23(c).

El ejemplo anterior demuestra el procedimiento general para cualquier circuito combinacional con una ROM. Mediante el número de entradas y salidas en el circuito combinacional, se determina primero al tamaño de la ROM necesario. Entonces debe obtenerse la tabla de verdad para programar la ROM; no se requiere otra manipulación o simplificación. Los 0 (o 1) en las funciones de salida de la tabla de verdad especifican de manera directa los eslabones que deben eliminarse para proporcionar el circuito combinacional requerido en la forma de suma de mintérminos.

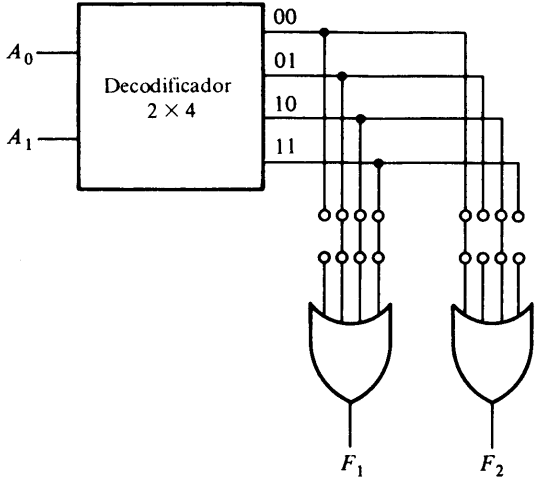
En la práctica, cuando se diseña un circuito mediante una ROM, no es necesario mostrar las conexiones internas en la compuerta de los eslabones dentro de la unidad como se hizo en la Fig. 5-23. Esto se mostró aquí tan solo para propósitos de demostración. Todo lo que el diseñador tiene que hacer es especificar la ROM particular (o su número de designación) y proporcionar la tabla de verdad de la ROM como en la Fig. 5-23(a). La tabla de verdad da toda la información para la programación de la ROM. No es necesario ningún diagrama lógico interno para acompañar la tabla de verdad.

EJEMPLO 5-5: Diseñe un circuito combinacional usando una ROM. El circuito acepta un número binario de 3-bit y genera un número binario de salida igual al cuadrado del número de entrada.

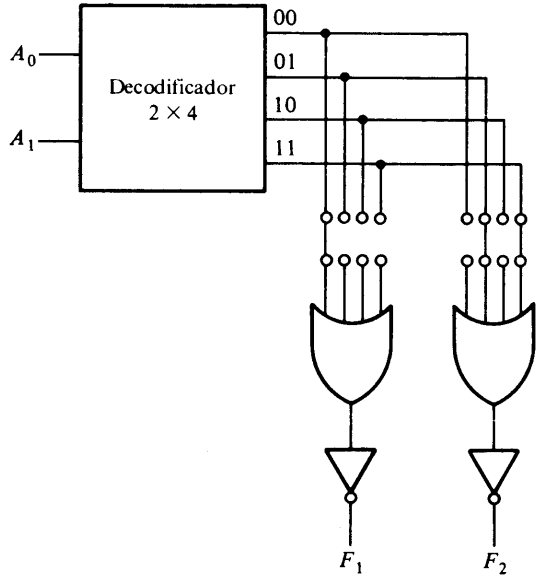
El primer paso es derivar la tabla de verdad para el circuito combinacional. En la mayoría de los casos esto es todo lo que se necesita. En algunos casos puede ajustarse una tabla de verdad más pequeña para la ROM por el uso de ciertas propiedades en la tabla de verdad del circuito combinacional. La Tabla 5-5 es la tabla de verdad para el circuito combinacional. Se requieren tres entradas y seis salidas

A_1	A_0	F_1	F_2
0	0	0	1
0	1	1	0
1	0	1	1
1	1	1	0

(a) Tabla de verdad



(b) ROM con compuertas AND-OR



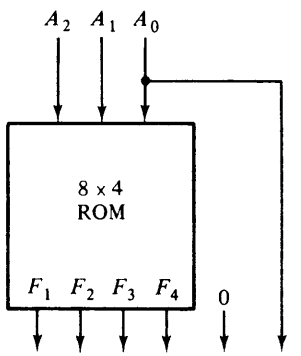
(c) ROM con compuertas AND-OR-INVERSORA

Figura 5-23 Implementación de un circuito combinacional con una ROM de 4 × 2.

TABLA 5-5 Tabla de verdad para el circuito del Ejemplo 5-5

Entradas			Salidas						Decimal
A_2	A_1	A_0	B_5	B_4	B_3	B_2	B_1	B_0	
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0	4
0	1	1	0	0	1	0	0	1	9
1	0	0	0	1	0	0	0	0	16
1	0	1	0	1	1	0	0	1	25
1	1	0	1	0	0	1	0	0	36
1	1	1	1	1	0	0	0	1	49

para acomodar todos los números posibles. Se observa que la salida B_0 siempre es igual a la entrada A_0; de modo que no se necesita generar B_0 con una ROM ya que es igual a una variable de entrada. Además, la salida B_1 siempre es 0, de modo que su salida siempre se conoce. En realidad sólo es necesario generar cuatro salidas con la ROM; las otras dos se obtienen con facilidad. El tamaño mínimo de la ROM necesaria debe tener tres entradas y cuatro salidas. Tres entradas especifican ocho palabras, de modo que el tamaño de la ROM debe ser 8 x 4. El implante ROM se muestra en la Fig. 5-24. Las tres entradas especifican ocho palabras de cuatro bits cada una. Las otras dos salidas del circuito combinacional son iguales a 0 y A_0. La tabla de verdad en la Fig. 5-24 especifica toda la información necesaria para programar la ROM, y el diagrama de bloques muestra las conexiones requeridas.



(a) Diagrama de bloques

A_2	A_1	A_0	F_1	F_2	F_3	F_4
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	1
0	1	1	0	0	1	0
1	0	0	0	1	0	0
1	0	1	0	1	1	0
1	1	0	1	0	0	1
1	1	1	1	1	0	0

(b) Tabla de verdad de la ROM

Figura 5-24 Implementación de la ROM del Ejemplo 5-5.

Tipos de ROM

Las trayectorias requeridas en una ROM pueden programarse en dos formas diferentes. La primera se llama *programación enmascarada* y la hace el fabricante durante el último proceso de producción de la unidad. El procedimiento para fabricar una ROM requiere que el cliente llene la tabla de verdad que él desea satisfaga la ROM. La tabla de verdad puede presentarse en una forma especial que proporciona el fabricante. Con más frecuencia, se presenta en cinta de papel o tarjetas perforadas en el formato especificado en la hoja de datos de la ROM particular. El fabricante hace la máscara correspondiente para las trayectorias para producir los 1 y 0 de acuerdo con la tabla de verdad del cliente. Este procedimiento es costoso debido a que el vendedor carga honorarios especiales al cliente por la máscara especial de pedido para una ROM. Por esta razón, la programación en máscara es económica sólo si van a fabricarse grandes cantidades de la misma configuración de ROM.

Para pequeñas cantidades, es más económico utilizar un segundo tipo de ROM llamado *memoria programable de solo lectura,* o PROM. Cuando se ordenan, las unidades PROM contienen todos los 0 (o todos los 1) en cada bit de las palabras almacenadas. Los eslabones en la PROM se rompen por la aplicación de pulsos de corriente a través de las terminales de entrada. Un eslabón roto define un estado binario y un eslabón sin romper representa el otro estado. Esto le permite al usuario programar la unidad en su propio laboratorio para lograr la relación deseada entre las direcciones de entrada y las palabras almacenadas. Están disponibles comercialmente unidades especiales denominadas *programadores PROM* para facilitar este procedimiento. En cualquier caso, todos los procedimientos para programar las ROM son procedimientos de *hardware* aun cuando se utilice la palabra *programación.*

El procedimiento de hardware para programar las ROM o PROM es irreversible y, una vez programados los patrones fijos, son permanentes y no pueden alterarse. Ya que se ha establecido un patrón de bit, la unidad debe descartarse si el patrón de bits va a cambiarse. Un tercer tipo de unidad disponible se conoce como *PROM borrable* o EPROM. Las EPROM pueden reestructurarse a su valor inicial (todos 0 o todos 1) aun cuando se hayan cambiado previamente. Cuando una EPROM se coloca bajo una luz ultravioleta especial por un periodo dado de tiempo, la radiación de onda corta descarga las compuertas internas que sirven como contactos. Después del borrado, la ROM regresa a su estado inicial y puede reprogramarse. Ciertas ROM pueden borrarse con señales eléctricas en lugar de luz ultravioleta, y éstas algunas veces se llaman *ROM alterables eléctricamente* o EAROM.

La función de una ROM puede interpretarse en dos formas diferentes. La primera interpretación es de una unidad que implementa cualquier circuito combinacional. Desde este punto de vista, cada terminal de salida se considera en forma separada como la salida de una función booleana expresada en suma de mintérminos. La segunda interpretación considera la ROM como una unidad de almacenamiento que tiene un patrón fijo de cadenas de bits denominadas *palabras.* Desde este punto de vista, las entradas especifican una *dirección* para una palabra particular almacenada la cual se aplica entonces a las salidas. Por ejemplo, la ROM en la Fig. 5-24 tiene tres líneas de dirección que especifican ocho palabras almacenadas como dadas por la

tabla de verdad. Cada palabra tiene una longitud de cuatro bits. Por esta razón la unidad recibe el nombre de *memoria de solo lectura.* La palabra *memoria* se utiliza de manera común para designar una unidad de almacenamiento. La palabra *lectura* se utiliza por lo normal para expresar que el contenido de una palabra especificada por una dirección en una unidad de almacenamiento se coloca en las terminales de salida. Por tanto, una ROM es una unidad de memoria con un patrón fijo de palabras que pueden leerse por la aplicación de una dirección dada. El patrón bit en la ROM es permanente y no puede cambiarse durante la operación normal.

Las ROM se usan mucho para implementar circuitos combinacionales complejos en forma directa desde sus tablas de verdad. Son útiles para convertir de un código binario a otro (como ASCII a EBCDIC y viceversa), para funciones aritméticas, por ejemplo multiplicadores, para exhibición de caracteres en un tubo de rayos catódicos, y en muchas otras aplicaciones que requieren un gran número de entradas y salidas. También se emplean en el diseño de unidades de control de sistemas digitales. Como tales, se usan para almacenar patrones de bit fijos que representan la secuencia de variables de control necesarias para capacitar las diversas operaciones en el sistema. Una unidad de control que emplea una ROM para almacenar información de control binaria se conoce como *unidad de control microprogramada.*

5-8 ARREGLO LOGICO PROGRAMABLE (PLA)

Ocasionalmente es posible que un circuito combinacional tenga condiciones no importa. Cuando se implementa con una ROM, una condición no importa se vuelve una entrada de dirección que nunca ocurrirá. Las palabras en las direcciones no importa no necesitan programarse y pueden dejarse en su estado original (todas 0 o todas 1). El resultado es que no se usan todos los patrones de bit disponibles en la ROM, lo cual puede considerarse un desperdicio de equipo disponible.

Considérese, por ejemplo, un circuito combinacional que convierte un código de tarjeta de 12-bit en un código alfanumérico interno de 6-bit, como se lista en la Tabla 1-5. El código de tarjeta de entrada consta de 12 líneas designadas por 0, 1, 2, ..., 9, 11, 12. El tamaño de la ROM para implementar el convertidor de código debe ser 4096 × 6, ya que hay 12 entradas y 6 salidas. Sólo hay 47 entradas válidas para el código de tarjeta; todas las otras combinaciones de entrada son condiciones no importa. Así que, sólo se usan 47 palabras de 4096 disponibles. Las 4049 palabras restantes de la ROM no se usan y, por lo tanto, se desperdician.

Para casos donde el número de las condiciones no importa es excesivo, es más económico usar un segundo tipo de componente LSI llamado *arreglo lógico programable* o PLA. Un PLA es similar en concepto a una ROM; sin embargo, el PLA no proporciona la plena decodificación de las variables y no genera todos los mintérminos como en la ROM. En el PLA, el decodificador se reemplaza por un grupo de compuertas AND, cada una de las cuales puede programarse para generar un término producto de las variables de entrada. Las compuertas AND y OR dentro del PLA están fabricadas inicialmente con eslabones entre ellas. Las funciones booleanas específicas se implementan en la forma de suma de productos por la apertura de los eslabones apropiados y dejando las conexiones deseadas.

En la Fig. 5-25 se muestra un diagrama de bloques del PLA. Consta de n entradas, m salidas, k términos producto y m suma de términos. Los términos productos constituyen un grupo de k compuertas AND y los términos suma constituyen un grupo de m compuertas OR. Los eslabones se insertan entre todas las n entradas y sus valores de complemento a cada una de las compuertas AND. Se proporcionan también eslabones entre las salidas de las compuertas AND y las entradas de las compuertas OR. Otro conjunto de eslabones en los inversores de salida permiten que se genere la función de salida ya sea en la forma AND-OR o en la forma AND-OR-inversora. Con el eslabón inversor en su lugar, el inversor se deriva, dando un AND-OR. Con el eslabón roto, el inversor se vuelve parte del circuito y se implanta la función en la forma AND-OR-inversora.

El tamaño de PLA se especifica por el número de entradas. El número de términos producto y el número de salidas (el número de los términos suma, es igual al número de salida). Un PLA típico tiene 16 entradas, 48 términos producto y 8 salidas.* El número de eslabones programado es $2^n \times k + k \times m + m$, en tanto que el de una ROM es $2^n \times m$.

En la Fig. 5-26 se muestra la construcción interna de un PLA específico. Tiene tres entradas, tres términos producto y dos salidas. Dicho PLA es demasiado pequeño para tener disponibilidad comercial; se presenta aquí sólo con fines de demostración. Cada entrada y su complemento se conectan a través de eslabones a las entradas de todas la compuertas AND. Las salidas de las compuertas AND se conectan a través de eslabones a cada entrada de las compuertas OR. Se proporcionan dos eslabones más con los inversores de salida. Mediante la rotura de eslabones seleccionados y la colocación de otros en su lugar, es posible implantar funciones booleanas en su forma de suma de productos.

Como con una ROM, el PLA puede ser programable por máscara o programable en campo. Con un PLA programable en máscara, el cliente debe someter una tabla de programa PLA al fabricante. Esta tabla la usa el vendedor para producir un PLA hecho sobre pedido que tenga las trayectorias internas requeridas entre entradas y salidas. Un segundo tipo de PLA disponible se conoce como *arreglo lógico programable en campo* o FPLA. El FPLA puede programarlo el usuario mediante ciertos procedimientos recomendados. Están disponibles unidades comerciales de hardware programable para usarse junto con ciertos FPLA.

*El IC tipo TTL 82S100.

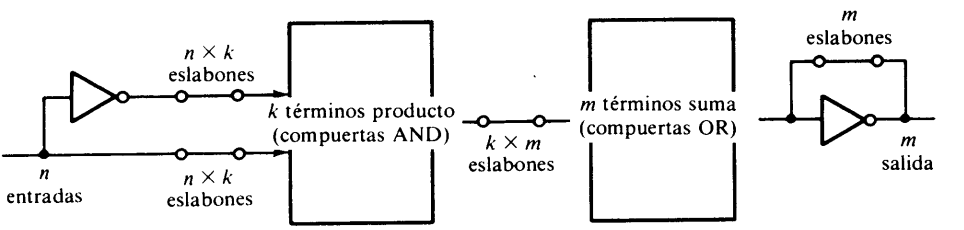


Figura 5-25 Diagrama de bloques del arreglo PLA.

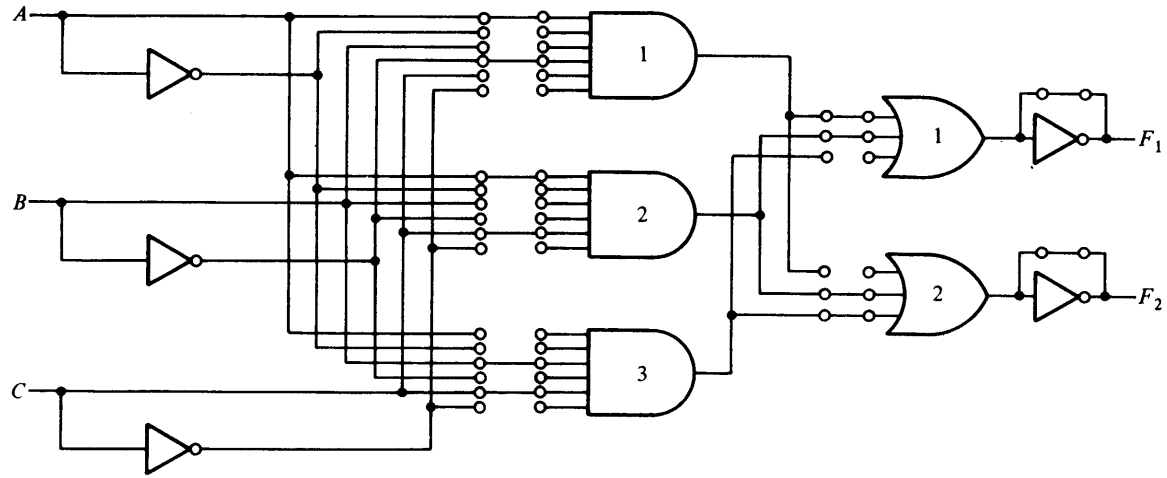


Figura 5-26 Arreglo PLA con 3 entradas, 3 términos producto y 2 salidas; implementa el circuito combinacional especificado en la Fig. 5-27.

Tabla de programa PLA

El uso de un PLA debe considerarse para circuitos combinacionales que tengan un gran número de entradas y salidas. Es superior a una ROM para circuitos que tienen un gran número de condiciones no importa. El ejemplo que se presenta a continuación demuestra cómo se programa un PLA. Cuando el lector vea el ejemplo debe tener en consideración que un circuito tan simple no requiere un PLA porque puede implementarse en forma más económica con compuertas SSI.

Considérese la tabla de verdad del circuito combinacional, que se muestra en la Fig. 5-27(a). Aunque una ROM implementa un circuito combinacional en la forma de suma de mintérminos, un PLA implementa las funciones en su forma de suma de productos. Cada producto término en la expresión requiere una compuerta AND. Ya que el número de compuertas AND en un PLA es finito, es necesario simplificar la función a un número mínimo de términos producto con objeto de minimizar el número de compuertas AND que se utilicen. Las funciones simplificadas en suma de productos se obtienen mediante los mapas en la Fig. 5-27(b):

$$F_1 = AB' + AC$$

$$F_2 = AC + BC$$

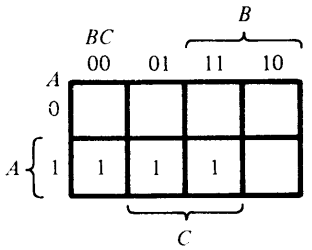
Hay tres distintos términos producto en este circuito combinacional: AB', AC y BC. El circuito tiene tres entradas y dos salidas; de modo que el PLA de la Fig. 5-26 puede usarse para implementar este circuito combinacional.

La programación del PLA significa que se especifican las trayectorias en su patrón AND-OR-NOT. Una tabla típica de programa PLA se muestra en la Fig. 5-27(c). Consta de tres columnas. La primera columna lista los términos producto numéricamente. En la segunda columna se especifican las trayectorias requeridas entre las entradas y las compuertas AND. La tercera columna especifica las trayectorias entre las compuertas AND y las compuertas OR. Bajo cada variable de entrada, se escribe una T (de la inicial en inglés de verdadero) si la salida inversora va a derivarse, y C (de la inicial de complemento) si la función va a complementarse con la salida inversora. Los términos booleanos que se listan a la izquierda no son parte de la tabla; se incluyen sólo como referencia.

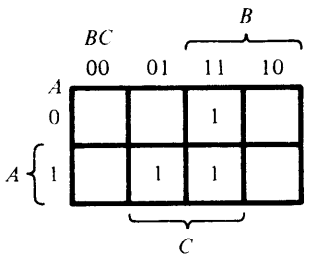
Para cada término producto, las entradas se marcan con 1, 0, o - (guión). Si una variable en el producto término aparece en su forma normal (sin prima), la variable correspondiente de entrada se marca con un 1. Si aparece complementada (con prima), la variable de entrada correspondiente se marca con un 0. Si la variable está ausente en el término producto, se marca con un guión. Cada término producto se asocia con una compuerta AND. Las trayectorias entre las entradas y las compuertas AND se especifican bajo la columna con encabezado *entradas*. Un 1 en la columna de entrada especifica una trayectoria de la entrada correspondiente a la entrada de la compuerta AND que forma el término producto. Un 0 en la columna de entrada especifica una trayectoria desde la entrada complementada correspondiente a la entrada de la compuerta AND. Un guión especifica que no hay conexión. Los

A	B	C	F_1	F_2
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	1	1
1	1	0	0	0
1	1	1	1	1

(a) Tabla de verdad



$F_1 = AB' + AC$



$F_2 = AC + BC$

(b) Simplificación de mapa

	Término producto	Entradas A	B	C	Salidas F_1	F_2	
AB'	1	1	0	–	1	–	
AC	2	1	–	1	1	1	
BC	3	–	1	1	–	1	
					T	T	T/C

(c) Tabla de programa del PLA

Figura 5-27 Pasos requeridos en la implementación del PLA.

eslabones apropiados están rotos, y los que se dejan en su lugar forman las trayectorias deseadas, como se muestra en la Fig. 5-26. Se supone que las terminales abiertas en la compuerta AND se comportan como una entrada 1.

Las trayectorias entre las compuertas AND y OR se especifican bajo la columna con encabezado de *salidas*. Las variables de salida se marcan con 1 para todos los términos producto que formulan la función. En el ejemplo de la Fig. 5-27 se tiene:

$$F_1 = AB' + AC$$

de modo que F_1 está marcada con 1 para los términos producto 1 y 2 con un guión para el término producto 3. Cada término producto que tiene 1 en la columna de salidas requiere una trayectoria desde la compuerta correspondiente AND a la compuerta de salida OR. Los que están marcados con un guión especifican que no hay conexión. Por último, una salida T (verdad) determina que el eslabón a través de la salida inversora permanezca en su lugar, y una C (complemento) especifica que el eslabón correspon-

diente está roto. Las trayectorias internas del PLA para este circuito se muestran en la Fig. 5-26. Se supone que una terminal abierta en una compuerta OR se comporta como un 0, y que un circuito corto a través de la salida inversora no daña el circuito.

Cuando se diseña un sistema digital con un PLA no es necesario mostrar las conexiones internas de la unidad como se hizo en la Fig. 5-26. Todo lo que se necesita es una tabla de programas PLA mediante la cual puede programarse el PLA para suministrar las trayectorias apropiadas.

Cuando se implementa un circuito combinacional con PLA, debe emprenderse una investigación cuidadosa con objeto de reducir el número total de términos producto distintos, ya que un PLA dado puede tener un número finito de términos AND. Esto puede hacerse por la simplificación de cada función a un número mínimo de términos. El número de literales en un término no es importante ya que se tienen disponibles todas las variables de entrada. Tanto el valor verdadero como el complemento de la función deben simplificarse para ver cuál puede expresarse con menos términos producto y cuál proporciona términos producto que son comunes a otras funciones.

EJEMPLO 5-6: Un circuito combinacional se define por las funciones:

$$F_1(A, B, C) = \Sigma(3, 5, 6, 7)$$
$$F_2(A, B, C) = \Sigma(0, 2, 4, 7)$$

Implemente el circuito con un PLA que tenga tres entradas, cuatro términos producto y dos salidas.

Las dos funciones están simplificadas en los mapas de la Fig. 5-28. Tanto los valores verdaderos como los complementos de las funciones se simplifican. La combinación que da el número mínimo de términos producto son:

$$F_1 = (B'C' + A'C' + A'B')'$$
$$F_2 = B'C' + A'C' + ABC$$

Esto da sólo cuatro términos producto distintos: $B'C'$, $A'C'$, $A'B'$, y ABC. La tabla programa del PLA para esta combinación se muestra en la Fig. 5-28. Observe que la salida F_1 es la salida normal (o verdadera) aun cuando está marcada bajo ella C. Esto se debe a que F'_1 se generó antes que la salida inversora. La inversora complementa la función para producir F_1 en la salida.

El circuito combinacional para este ejemplo es demasiado pequeño para implementación práctica con un PLA. Aquí simplemente se presenta con propósitos de demostración. Un PLA comercial típico tendría más de 10 entradas y cerca de 50 términos producto. La simplificación de funciones booleanas con tantas variables se lleva a cabo mediante un método de tabulación o bien otro método de simplificación

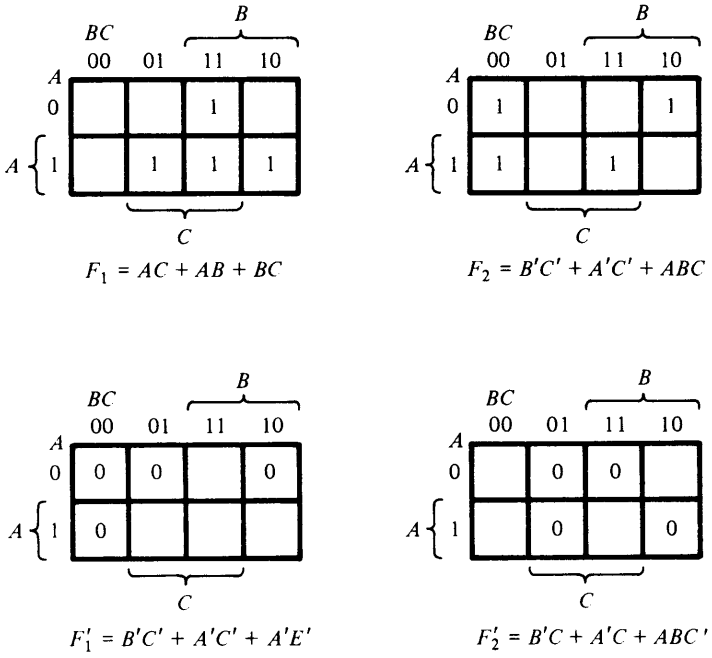


Tabla de programa del PLA

	Término producto	Entradas A	B	C	Salidas F_1	F_2	
$B'C'$	1	–	0	0	1	1	
$A'C'$	2	0	–	0	1	1	
$A'B'$	3	0	0	–	1	–	
ABC	4	1	1	1	–	1	
					C	T	T/C

Figura 5-28 Solución del Ejemplo 5-6.

ayudado por computadora. Aquí es donde un programa de computadora puede ayudar en el diseño de sistemas digitales complejos. El programa de computadora debe simplificar cada función del circuito combinacional y su complemento hacia un número mínimo de términos. El programa selecciona entonces un número mínimo de términos distintos que cubren todas las funciones en sus formas verdadera o complementaria.

5-9 COMENTARIOS CONCLUYENTES

En este capítulo se presentó una variedad de métodos de diseño para circuitos combinacionales. También se presentó y explicó un número de circuitos MSI y LSI que pueden utilizarse cuando se diseña sistemas digitales más complicados. Se dio

énfasis en las funciones de lógica combinacional MSI y LSI. Las funciones de lógicas secuencial MSI se exponen en el Capítulo 7. Estas funciones digitales MSI y LSI son los bloques básicos de construcción mediante los cuales se construyen los sistemas digitales y la computadoras digitales.

Las funciones MSI que aquí se presentan y otras disponibles en el comercio se describen en los libros y catálogos de datos. Los libros de datos de IC contienen descripciones exactas de muchos MSI y otros circuitos integrados. Algunos de estos libros de datos se listan en la siguiente bibliografía.

Los circuitos MSI y LSI pueden usarse en una variedad de aplicaciones. Algunas de estas aplicaciones se expusieron en este capítulo, algunas se incluyen en los problemas y otras se presentarán en los capítulos siguientes junto con sus aplicaciones particulares. Los diseñadores con inventiva pueden encontrar muchas otras aplicaciones que se ajusten a sus necesidades particulares. Los fabricantes de circuitos integrados publican numerosas *notas de aplicación* para sugerir utilizaciones posibles de sus productos. Puede obtenerse una lista de las notas de aplicaciones disponibles escribiendo directamente a los fabricantes o investigando con sus representantes locales.

BIBLIOGRAFIA

1. Mano, M. M., *Computer System Architecture,* 2a. ed. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1982.
2. Morris, R. L., and J. R. Miller, eds., *Designing with TTL Integrated Circuits.* New York: McGraw-Hill Book Co., 1971.
3. Blakeslee, T. R., *Digital Design with Standard MSI and LSI.* New York: John Wiley & Sons, 1975.
4. Barna A., and D. I. Porat, *Integrated Circuits in Digital Electronics.* New York: John Wiley & Sons, 1973.
5. Lee, S. C., *Digital Circuits and Logic Design,* Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1976.
6. Semiconductor Manufacturers Data Books (Consúltese la última edición):
   (a) *The TTL Data Book for Design Engineers.* Dallas, Texas: Texas Instruments, Inc.
   (b) *The Fairchild Semiconductor TTL Data Book.* Mountain View, Calif.: Fairchild Semiconductor.
   (c) *Digital Integrated Circuits.* Santa Clara, Calif.: National Semiconductor Corp.
   (d) *Signetics Digital, Linear, MOS.* Sunnyvale, Calif.: Signetics.
   (e) *MECL Integrated Circuits Data Book.* Phoenix, Ariz.: Motorola Semiconductor Products, Inc.
   (f) *RCA Solid State Data Book Series.* Somerville, N. J.: RCA Solid State Div.

PROBLEMAS

5-1. Diseñe un convertidor de código exceso-3- a-BCD usando un circuito MSI de sumadores completos de 4-bit.

5-2. Usando cuatro circuitos MSI, construya un sumador paralelo binario para sumar dos números binarios de 16-bit. Etiquete todos los acarreos entre los circuitos MSI.

• 5-3. Utilizando cuatro compuertas OR-excluyente y un circuito MSI de sumadores completos de 4-bit, construya un sumador restador paralelo de 4-bit. Use una variable V de selección de entrada de modo que cuando $V = 0$, el circuito sume y cuando $V = 1$, el circuito reste. (*Sugerencia:* Utilice resta de complemento a 2.)

5-4. Derive la ecuación de dos niveles para el acarreo C_5 mostrando el generador de acarreo por anticipado en la Fig. 5-5.

5-5. (a) Usando el procedimiento de implementación AND-OR-INVERSORA que se describe en la Sección 3-7, mostrando que el acarreo de salida en un circuito sumador completo puede expresarse como:

$$C_{i+1} = G_i + P_iC_i = (G_i'P_i' + G_i'C_i')'$$

(b) El IC tipo 74182 es un circuito MSI generador de acarreo por anticipado, que genera los acarreos con compuertas AND-OR-INVERSORA. El circuito MSI supone que las terminales de entrada tienen los complementos de las G_i las P y de C_1. Derive las funciones booleanas para los acarreos por anticipado C_2, C_3 y C_4 en este IC. (*Sugerencia:* Use el método de sustitución de ecuaciones para derivar acarreos en términos de C'_1.)

5-6. (a) Redefina la propagación de acarreo y el acarreo generado como sigue:

$$P_i = A_i + B_i$$
$$G_i = A_iB_i$$

Muestre que el acarreo de salida y la salida de suma de un sumador completo llega a ser:

$$C_{i+1} = (C_i'G_i' + P_i')' = G_i + P_iC_i$$
$$S_i = (P_iG_i') \oplus C_i$$

(b) El diagrama lógico de la primera etapa de un sumador en paralelo de 4-bit cuando se implementa en el IC tipo 74283 se muestra en la Fig. P5-6. Identifique las terminales P_i' y G_i' como se define en (a) y muestra que el circuito implementa un circuito sumador completo.

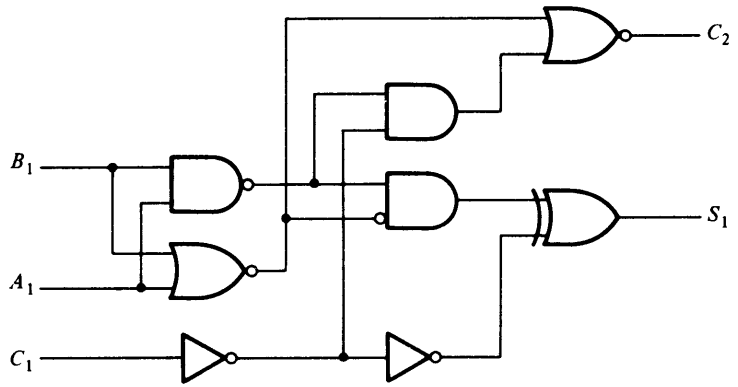


Figura P5-6 Primera etapa de un sumador paralelo.

(c) Obtenga los acarreos de salida C_3 y C_4 como función de P'_1, P'_2, P'_3, G'_1, G'_2, G'_3 y C'_1 en la forma AND-OR-INVERSORA, y dibuje el circuito de anticipación de dos niveles para este IC. [*Sugerencia:* Utilice el método de sustitución de ecuaciones como se hizo en el texto cuando se derivó la Fig. 5-4, pero use la función AND-OR-INVERSORA dada en (a) para C_{i+1}.]

5-7. (a) Suponga que la compuerta OR excluyente tiene un retardo de propagación de 20 ns y que las compuertas AND u OR tienen un retardo de propagación de 10 ns. ¿Cuál es el tiempo retardo total de propagación en el sumador de 4-bit de la Fig. 5-5?
(b) Suponga que C_5 se propaga en la caja de la Fig. 5-7 al mismo tiempo que los otros acarreos (véase el Problema 5-4). ¿Cuál será el tiempo de retardo de propagación del sumador de 16-bit del problema 5-2?

5-8. Diseñe un multiplicador binario que multiplique un número de 4-bit $B = b_3b_2b_1b_0$ por un número de 3-bit $A = a_2a_1a_0$ para formar el producto $C = c_6c_5c_4c_3c_2c_1c_0$. Esto puede hacerse con 12 compuertas y dos sumadores paralelos de 4-bit. Las compuertas AND se usan para formar los productos de pares de bits. Por ejemplo, el producto de a_0 y b_0 puede generarse por la reunión de a_0 con b_0 en la lógica AND. Los productos parciales formados por las compuertas AND se suman con los sumadores paralelos.

5-9. ¿Cuántas entradas no importa hay en un sumador BCD?

5-10. Diseñe un circuito combinacional que genere el complemento a 9 de un dígito BCD.

5-11. Diseñe una unidad aritmética decimal con dos variables de selección, V_1 y V_0, y dos dígitos BCD, A y B. La unidad debe tener cuatro operaciones aritméticas que dependen de los valores de las variables de selección como se muestra a continuación.

V_1	V_0	Función salida	
0	0	$A + 9$	complemento de B
0	1	$A + B$	
1	0	$A + 10$	complemento de B
1	1	$A + 1$	(añada 1 a A)

Use funciones MSI en el diseño y la complementación a 9 del problema 5-10.

5-12. Es necesario diseñar un sumador decimal para dos dígitos representados en el código exceso-3 (Tabla 1-2). Muestre que la corrección después de agregar los dos dígitos con un sumador binario de 4-bit es como sigue:
(a) El acarreo de salida es igual al acarreo de salida del sumador binario.
(b) Si el acarreo de salida = 1, agregue 0011.
(c) Si el acarreo de salida = 0, agregue 1101.
Construya el sumador con dos sumadores binarios de 4-bit y un inversor.

5-13. Diseñe un circuito que compare dos números de 4-bit, A y B, para revisar si son iguales. El circuito tiene una salida x, de modo que $x = 1$ si $A = B$, y $x = 0$ si $A \neq B$.

5-14. El IC 74L85 es un comparador de magnitud de 4-bit similar al de la Fig. 5-7, excepto que tiene tres entradas más y circuitos internos que realizan la lógica equivalente como se muestra en la Fig. P5-14. Con esos IC, pueden compararse números de longitud más grande al conectarlos con separadores en cascada. Las salidas $A < B$, $A > B$ y $A = B$ de una etapa, manipulando los bits menos significativos están conectadas a las entradas correspondientes $A <$ B, $A > B$ y $A = B$ de la siguiente etapa, manipulando bits más signi..ativos.

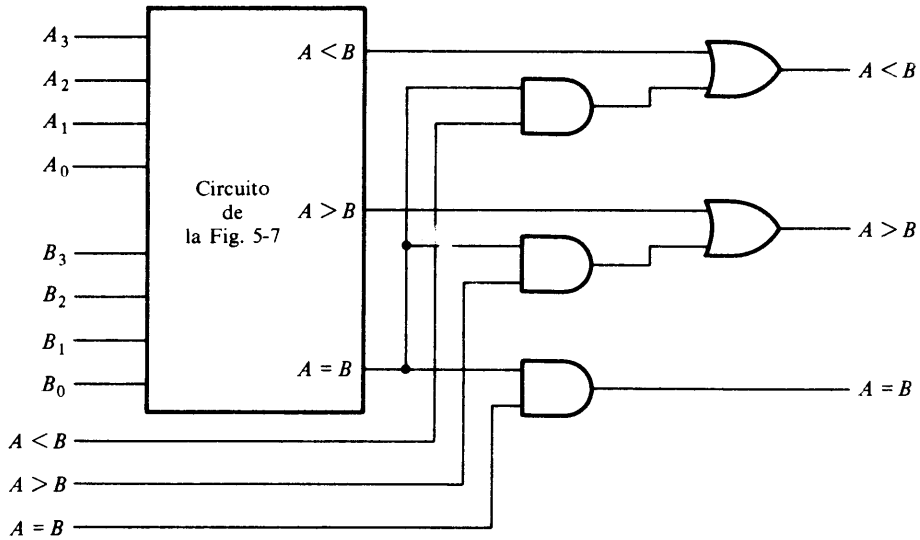


Figura P5-14 Circuito lógico equivalente del IC tipo 74L85.

La etapa que manipula los bits menos significativos puede ser un circuito como se muestra en la Fig. 5-7. Si se usa el IC 74L85, debe aplicarse un 1 a la entrada $A = B$ y un 0 a las entradas $A < B$ y $A > B$ en el IC que manipula los cuatro bits menos significativos. Utilice un circuito como en la Fig. 5-7 y un IC 74L85, y obtenga un circuito para comparar dos números de 8-bit. Justifique la operación del circuito.

5-15. Modifique el decodificador de BCD a decimal de la Fig. 5-10 para dar una salida por completo en 0 cuando ocurre cualquier combinacación inválida de entrada.

5-16. Diseñe un convertidor de código BCD-a-exceso-3 con un decodificador BCD-a-decimal y cuatro compuertas OR.

5-17. Un circuito combinacional se define por las tres funciones siguientes:

$$F_1 = x'y' + xyz'$$
$$F_2 = x' + y$$
$$F_3 = xy + x'y'$$

Diseñe el circuito con un decodificador y compuertas externas.

5-18. Un circuito combinacional se define por las dos funciones siguientes:

$$F_1(x, y) = \Sigma(0, 3)$$
$$F_2(x, y) = \Sigma(1, 2, 3)$$

Implemente el circuito combinacional mediante el decodificador que se muestra en la Fig. 5-12 y compuertas externas NAND.

5-19. Construya un decodificador 5×32 con cuatro decodificadores/demultiplexores 3×8 y un decodificador 2×4. Use una construcción en diagrama de bloques como Fig. 5-14.

5-20. Dibuje el diagrama lógico de un decodificador/demultiplexor 2-línea a 4-línea usando sólo compuertas NOR.

5-21. Especifique la tabla de verdad de un codificador de prioridad octal-a-binario. Proporcione una salida para indicar si cuando menos una de las entradas es un 1. La tabla puede listarse con nueve renglones y algunas de las entradas tendrán valores no importa.

5-22. Diseñe un codificador de prioridad de 4-líneas a 2-línea. Incluya una salida E para indicar que cuando menos una entrada es un 1.

5-23. Implemente la función booleana del Ejemplo 5-4 con un multiplexor de 8×1 con A, B y D conectadas a líneas de selección s_2, s_1 y s_0, respectivamente.

5-24. Implemente el circuito combinacional especificado en el problema 5-17 con multiplexores dual 4-línea a 1-línea, y compuerta OR e inversora.

5-25. Obtenga un multiplexor de 8×1 con multiplexores dual 4-línea a 1-línea teniendo entradas de habilitación separadas pero líneas comunes de selección. Use una construcción de diagrama de bloques.

5-26. Implemente un circuito sumador completo con multiplexores.

5-27. La ROM de 32×6 junto con la 2^0 línea como se muestra en la Fig. P5-27 convierte un número binario de 6-bit en su correspondiente número BCD de 2-dígitos. Por ejemplo, el binario 100001 convierte a BCD 011 0011 (decimal 33). Especifique la tabla de verdad para la ROM.

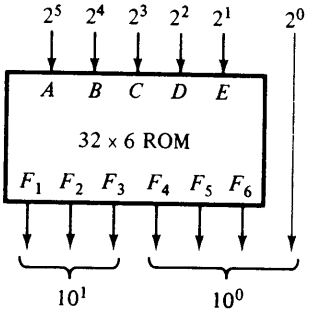


Figura P5-27 Convertidor de binario en decimal.

5-28. Pruebe que una ROM de 32×8 puede usarse para implementar circuito que genere el cuadrado binario de un número de entrada de 5-bit con $B_0 = A_0$ y $B_1 = 0$ como en la Fig. 5-24(a). Dibuje un diagrama de bloques del circuito y liste las primeras cuatro entradas y las últimas cuatro entradas de la tabla de verdad de la ROM.

5-29. ¿Qué tamaño debe tener la ROM para implementar:
(a) Un sumador/restador BCD con un control de entrada para seleccionar entre la adición y la resta.
(b) Un multiplicador binario que multiplique dos números de 4-bit.
(c) Multiplexores dual 4-línea a 1-línea con entradas de selección comunes.

5-30. Cada salida inversora en el PLA de la Fig. 5-26 se reemplaza por una compuerta OR-excluyente. Cada compuerta OR-excluyente tiene dos entradas. Una entrada está conectada a la salida de la compuerta OR, y la otra entrada está conectada a través de

eslabones a una señal equivalente ya sea a 0 o 1. Muestre cómo seleccionar la salida verdad/complemento en esta configuración.

5-31. Derive la tabla programa de PLA para un circuito combinacional que eleve al cuadrado un número de 3-bit. Minimice el número de términos producto. (Véase la Fig. 5-24 para la implementación de la ROM equivalente.)

5-32. Liste la tabla programa PLA para el convertidor de código BCD-a-exceso-3 que se define en la Sección 4-5.

Lógica secuencial síncrona

6

6-1 INTRODUCCION

Los circuitos digitales que hasta ahora se han considerado han sido combinacionales, esto es, las salidas en cualquier momento dependen por completo de las entradas presentes en ese tiempo. Aunque cualquier sistema digital es susceptible de tener circuitos combinacionales, la mayoría de los sistemas que se encuentran en la práctica también incluyen elementos de memoria, los cuales requieren que el sistema se describa en términos de *lógica secuencial.*

Un diagrama de bloques de un circuito secuencial se muestra en la Fig. 6-1. Consta de un circuito combinacional al que se conectan elementos de memoria para formar una trayectoria de retroalimentación. Los elementos de memoria son dispositivos capaces de almacenar dentro de ellos información binaria. La información binaria almacenada en los elementos de memoria en cualquier momento dado define el *estado* del circuito secuencial. El circuito secuencial recibe información binaria de entradas externas. Estas entradas, junto con el estado presente de los elementos de memoria, determinan el valor binario en las terminales de salida. También determinan las condiciones para cambiar el estado en los elementos de memoria. El diagrama de bloque demuestra que las salidas externas en un circuito secuencial son funciones no sólo de las entradas externas sino también del estado presente de los elementos de memoria. El siguiente estado de los elementos de memoria también es una función de las entradas externas y del estado presente. Por tanto, un circuito secuencial está especificado por una secuencia de tiempo de entradas, salidas y estados internos.

Hay dos tipos principales de circuitos secuenciales. Su clasificación depende del temporizado de sus señales. Un circuito secuencial *síncrono* es un sistema cuyo comportamiento puede definirse por el conocimiento de sus señales en instantes discretos de tiempo. El comportamiento de un circuito secuencial *asíncrono* depende del orden en el cual cambian sus señales de entrada y puede afectarse en cualquier instante de tiempo. Los elementos de memoria que por lo común se utilizan en los circuitos secuenciales asíncronos son dispositivos de retardo de tiempo. La capacidad de memoria de un dispositivo de retardo de tiempo se debe al hecho de que toma un tiempo finito para que la señal se propague a través del dispositivo. En la práctica, el

retardo de propagación interno en las compuertas lógicas es de suficiente duración para producir el retardo necesario, de modo que pueden ser innecesarias unidades físicas de retardo de tiempo. En los sistemas asíncronos de tipo de compuerta, los elementos de memoria en la Fig. 6-1 constan de compuertas lógicas cuyos retardos de propagación constituyen la memoria requerida. Por consiguiente, un circuito secuencial asíncrono puede considerarse como un circuito combinacional con retroalimentación. Debido a la retroalimentación entre compuertas lógicas, un circuito secuencial asíncrono a veces puede llegar a ser inestable. El problema de la inestabilidad le impone muchas dificultades al diseñador. Los circuitos secuenciales asíncronos se presentan en el Capítulo 9.

Un sistema lógico secuencial asíncrono, por definición, debe emplear señales que afecten los elementos de memoria sólo en instantes discretos de tiempo. Una forma de lograr este objetivo es usar pulsos de duración limitada a través del sistema, de modo que una amplitud de pulso represente la lógica 1 y otra amplitud del pulso (o la ausencia de un pulso) represente la lógica 0. La dificultad con un sistema de pulsos es que cualesquiera dos pulsos que lleguen de fuentes independientes separadas a las entradas de la misma compuerta exhibirán retardos impredecibles, que separarán los pulsos ligeramente y resultarán en operación poco confiable.

Los sistemas lógicos secuenciales síncronos usan amplitudes fijas, como niveles de voltaje para las señales binarias. La sincronización se logra por un dispositivo temporizador llamado *reloj maestro generador*, el cual genera un tren periódico de pulsos de reloj. Los pulsos de reloj se distribuyen a través del sistema en tal forma que los elementos de memoria están afectados sólo por la llegada del pulso de sincronización. En la práctica los pulsos de reloj se aplican a compuertas AND junto con las señales que especifican el cambio requerido en los elementos de memoria. Las salidas de la compuerta AND pueden transmitir señales sólo a los instantes que coinciden con la llegada de los pulsos de reloj. Los circuitos secuenciales síncronos que usan pulsos de reloj en las entradas de los elementos de memoria se denominan *circuitos secuenciales de reloj*. Los circuitos secuenciales de reloj son el tipo que se encuentra con más frecuencia. No manifiestan problemas de inestabilidad y su temporizado se desglosa fácilmente en pasos discretos independientes, cada uno de los cuales se considera por separado. Los circuitos secuenciales que se exponen en este capítulo son exclusivamente del tipo de reloj.

Los elementos de memoria que se usan en los circuitos secuenciales de reloj se llaman *flip-flops*. Estos circuitos son celdas binarias capaces de almacenar un bit de información. Un circuito flip-flop tiene dos salidas, una para el valor normal y otra

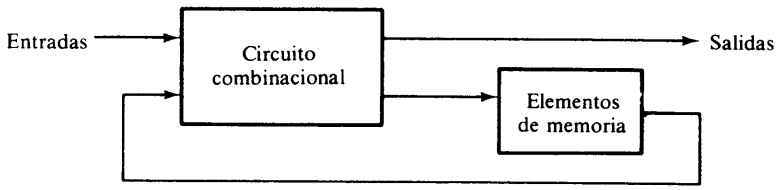


Figura 1-1 Diagrama de bloques de un circuito secuencial.

para el valor complementario del bit almacenado en él. La información binaria puede entrar a un flip-flop en una gran variedad de formas, hecho que da lugar a diferentes tipos de flip-flops. En la siguiente sección se examinarán los diversos tipos de flip-flops y se definirán sus propiedades lógicas.

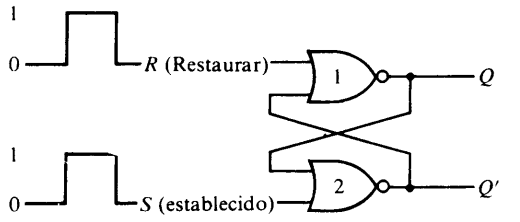
6-2 FLIP-FLOPS

Un circuito flip-flop puede mantener un estado binario en forma indefinida (en tanto se suministre potencia al circuito) hasta que recibe la dirección de una señal de entrada para cambiar estado. La diferencia principal entre los diversos tipos de flip-flops está en el número de entradas que poseen y en la manera en la cual las entradas afectan el estado binario. Los tipos más comunes de flip-flop se exponen a continuación.

Circuito básico flip-flop

En las secciones 4-7 y 4-8 se mencionó que un circuito flip-flop puede construirse mediante dos compuertas NAND o dos compuertas NOR. Estas construcciones se muestran en los diagramas lógicos de las Figs. 6-2 y 6-3. Cada circuito forma un flip-flop básico bajo el cual pueden construirse otros tipos más complicados. La conexión y acoplamiento cruzado mediante la salida de una compuerta a la entrada de otra constituye una trayectoria de retroalimentación. Por esta razón, los circuitos se clasifican como circuitos secuenciales asíncronos. Cada flip-flop tiene dos salidas, Q y Q', y dos entradas, *ajustar* (*set*) y *restaurar* (*reset*). Este tipo de flip-flop algunas veces se denomina flip-flop *RS directamente acoplado* o *seguro* (*latch*) *SR*. La R y S son las iniciales de los dos nombres de la entrada (set y reset en inglés).

Para analizar la operación del circuito en la Fig. 6-2, debe recordarse que la salida de una compuerta NOR es 0 si cualquier entrada es 1, y que la salida es 1 sólo cuando todas las entradas son 0. Como punto de inicio, se supone que la entrada ajuste (set) es 1, y la entrada restaurar (reset) es 0. Ya que la compuerta 2 tiene una entrada de 1, su salida Q' debe ser 0, la cual pone ambas entradas de la compuerta 1 en 0, de modo que la salida Q es 1. Cuando la entrada ajuste se regresa a 0, la salida permanece igual, debido a que la salida Q permanece en 1, dejando una entrada de la compuerta 2 en 1.

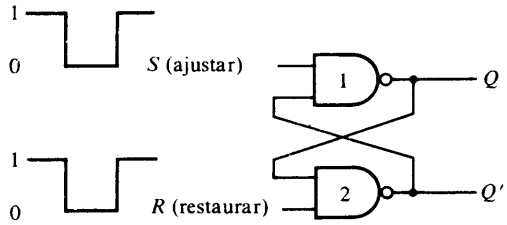


(a) Diagrama lógico

S	R	Q	Q'	
1	0	1	0	
0	0	1	0	(después que $S = 1$, $R = 0$)
0	1	0	1	
0	0	0	1	(después que $S = 0$, $R = 1$).
1	1	0	0	

(b) Tabla de verdad

Figura 6-2 Circuito flip-flop básico con compuertas NOR.



(a) Diagrama lógico

S	R	Q	Q'	
1	0	0	1	
1	1	0	1	(después que $S = 1$, $R = 0$)
0	1	1	0	
1	1	1	0	(después que $S = 0$, $R = 1$)
0	0	1	1	

(b) Tabla de verdad

Figura 6-3 Circuito flip-flop básico con compuertas NAND.

Esto causa que la salida Q' permanezca en 0, lo cual deja ambas entradas de compuerta 1 en 0, de modo que la salida Q está en 1. En la misma forma es posible mostrar que un 1 en la entrada de restaurar cambia la salida Q a 0 y Q' a 1. Cuando la entrada de restaurar vuelve a 0, las salidas no cambian.

Cuando se aplica un 1 a ambas entradas de ajuste (set) y restaurar (reset), tanto la salida Q como la Q' van a 0. Esta condición viola el hecho de que las salidas Q y Q' son los complementos una de otra. En la operación normal esta condición debe evitarse al tener la seguridad de que los 1 no son aplicables en forma simultánea a ambas entradas.

Un flip-flop tiene dos estados útiles, cuando $Q = 1$ y $Q' = 0$, está en el *estado ajuste* (o estado 1). Cuando $Q = 0$ y $Q' = 1$, está en el *estado despejado* (o estado 0). Las salidas Q y Q' son complementarias una de otra y se refieren como las salidas normal y complementaria, respectivamente. El estado binario del flip-flop se toma para que sea el valor de la salida normal.

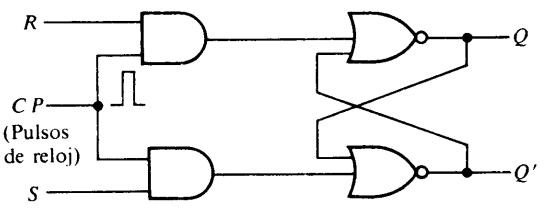
Bajo operación normal, ambas entradas permanecen en 0 a menos que tenga que cambiarse el estado del flip-flop. La aplicación de un 1 momentáneo a la entrada de ajuste provoca que el flip-flop pase al estado ajuste. La entrada ajuste debe volver a 0 antes de que un 1 se aplique a la entrada de restaurar. Un 1 momentáneo aplicado a la entrada de restaurar causa que el flip-flop vaya al estado despejado. Cuando ambas entradas son inicialmente 0, un 1 aplicado a la entrada de puesto mientras el flip-flop está en el estado ajuste o un 1 aplicado a la entrada de restaurar mietras el flip-flop está en el estado despejado deja las salidas sin cambio. Cuando se aplica un 1 a ambas entradas de ajuste y restaurar, ambas salidas pasan a 0. Este estado es indefinido y por lo común se evita. Si ambas entradas ahora van a 0, el estado del flip-flop es indeterminado y depende de cuál entrada permanezca en 1 más tiempo antes de la transición a 0.

El circuito flip-flop NAND básico en la Fig. 6-3 opera con ambas entradas normalmente en 1, a menos que el estado del flip-flop tenga que cambiarse. La aplicación de un 0 momentáneo a la entrada de ajuste causa que la salida Q vaya a 1 y Q' a 0, poniendo por tanto el flip-flop en el estado de ajuste. Después de que la entrada de ajuste regresa a 1, un 0 momentáneo en la entrada de restaurar provoca una transición al estado despejado. Cuando ambas entradas van a 0, ambas salidas irán a 1, una condición que se evita en la operación normal del flip-flop.

Flip-flop *RS* con reloj

El flip-flop básico, tal como está, es un circuito secuencial asíncrono. Por la adición de compuertas a las entradas del circuito básico, puede hacerse que el flip-flop responda a niveles de entrada durante la ocurrencia de un pulso de reloj. El flip-flop *RS* con reloj que se muestra en la Fig. 6-4(a) consta de un flip-flop básico NOR y dos compuertas AND. Las salidas de las dos compuertas AND permanecen en 0 en tanto que el pulso de reloj (abreviado *CP*, de las iniciales en inglés de *clock pulse*) sea 0, sin importar los valores de entrada *S* y *R*. Cuando el pulso de reloj va a 1, se permite que la información de las entradas *S* y *R* alcancen al flip-flop básico. El estado de ajuste se alcanza con $S = 1$, $R = 0$ y $CP = 1$. Para cambiar al estado despejado, las entradas deben ser $S = 0$, $R = 1$ y $CP = 1$. Tanto con $S = 1$ y $R = 1$, la ocurrencia de un pulso de reloj provoca que ambas salidas momentáneamente a 0. Cuando se elimina el pulso, el estado del flip-flop es indeterminado, esto es, puede resultar cualquier estado, dependiendo de si la entrada de ajuste o la de restaurar del circuito flip-flop básico permanezca en 1 durante un tiempo más prolongado antes de la transición a 0 al fin del pulso.

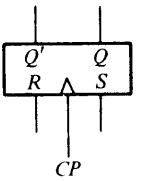
El símbolo gráfico para el flip-flop *RS* con reloj se muestra en la Fig. 6-4(b). Tiene tres entradas: *S*, *R* y *CP*. La entrada *CP* no está indicada dentro de la caja, debido a que se reconoce por el triángulo pequeño marcado. El triángulo es un símbolo para un *indicador dinámico* y denota el hecho de que el flip-flop responde a



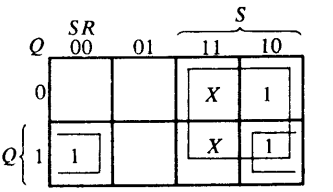
(a) Diagrama lógico

Q	S	R	$Q(t+1)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	indeterminado
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	indeterminado

(c) tabla característica



(b) símbolo gráfico



$Q(t+1) = S + R'Q$

$SR = 0$

(d) ecuación característica

Figura 6-4 Flip-flop *RS* con pulsos de reloj.

una *transición* de reloj en una señal de bajo nivel (binario 0) a un alto nivel (binario 1). Las salidas del flip-flop están marcadas con Q y Q' *dentro* de la caja. Puede asignarse al flip-flop una variable con nombre diferente aunque Q esté escrita dentro de la caja. En ese caso, la letra que se elige para la variable del flip-flop se marca *fuera* de la caja junto a la línea de salida. El estado del flip-flop está determinado por el valor de su salida normal Q. Si se desea obtener el complemento de la salida normal, no es necesario insertar un invertidor, ya que el valor complementado está disponible directamente mediante la salida Q'.

La tabla característica para el flip-flop se muestra en la Fig. 6-4(c). En esta tabla se resume la operación del flip-flop en una forma tabular. Q es el estado binario del flip-flop en un momento dado (referido como *estado presente*), las columnas S y R dan los valores posibles de las entradas y $Q(t+1)$ es el estado del flip-flop después de la ocurrencia de un pulso de reloj (referida como *estado siguiente*).

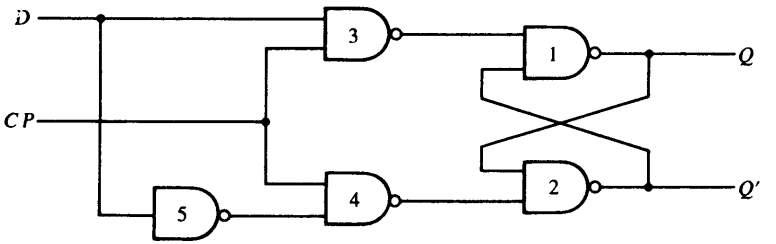
La ecuación característica del flip-flop se deriva en el mapa en la Fig. 6-4(d). Esta ecuación especifica el valor del estado siguiente como una función del estado presente y las entradas. La ecuación característica es una expresión algebraica para la información binaria de la tabla característica. Los dos estados indeterminados están marcados con X en el mapa, ya que pueden resultar en 1 o bien en 0. Sin embargo, la relación $SR = 0$ debe incluirse como parte de la ecuación característica para especificar que tanto S como R no pueden ser iguales a 1 en forma simultánea.

Flip-flop *D*

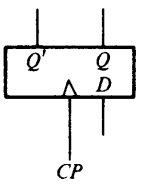
El flip-flop D que se muestra en la Fig. 6-5 es una modificación del flip-flop RS con reloj. Las compuertas NAND 1 y 2 forman un flip-flop básico y las compuertas 3 y 4 modifican para formar un flip-flop RS con reloj. La entrada D va en forma directa a la entrada S, y su complemento, a través de la compuerta 5, se aplica a la entrada R. En tanto que el pulso de reloj en la entrada esté en 0, las compuertas 3 y 4 tienen un 1 en sus salidas, sin importar el valor de las otras entradas. Esto se apega al requisito de que las dos entradas de un flip-flop básico NAND (Fig. 6-3) permanezcan inicialmente en el nivel 1. La salida D se muestrea durante la ocurrencia de un pulso de reloj. Si es 1, la salida de la compuerta 3 pasa a 0, cambiando el flip-flop al estado de ajuste (a menos que ya esté puesto), si es 0, la salida de la compuerta 4 va a 0, cambiando el flip-flop al estado despejado.

El flip-flop D recibe su denominación debido a su capacidad de transferir "datos" en el flip-flop. En forma básica es un flip-flop RS con un inversor en la salida R. El inversor agregado reduce el número de entradas de dos a una. Este tipo de flip-flop algunas veces se denomina un *seguro-D con compuertas*. La entrada CP con frecuencia recibe la designación variable G (de la inicial en inglés de compuerta, es decir *gate*) para indicar que esta entrada habilita el seguro con compuertas para hacer posible la entrada de información dentro del flip-flop.

El símbolo para un flip-flop D temporizado se muestra en la Fig. 6-5(b). La tabla característica se lista en la parte (c) y se deriva la ecuación característica en la parte (d). La ecuación característica muestra que el estado siguiente del flip-flop es el mismo de la entrada D y es independiente del valor del estado presente.



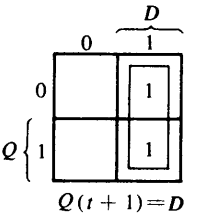
(a) Diagrama lógico con compuertas NAND



(b) Símbolo gráfico

Q	D	$Q(t+1)$
0	0	0
0	1	1
1	0	0
1	1	1

(c) Tabla característica



(d) Ecuación característica

Figura 6-5 Flip-flop *D* con pulsos de reloj.

Flip-flop *JK*

Un flip-flop *JK* es un refinamiento del flip-flop *RS* ya que el estado indeterminado del tipo *RS* se define en el tipo *JK*. Las entradas *J* y *K* se comportan como las entradas *S* y *R* para ajustar y despejar el flip-flop (obsérvese que en un flip-flop *JK*, la letra *J* es para ajustar y la letra *K* es para el *despeje*). Cuando se aplican señales de entrada en forma simultánea a *J* como a *K*, el flip-flop cambia a su estado complementario, esto es, si $Q = 1$, cambia a $Q = 0$ y viceversa.

Un flip-flop *JK* temporizado se muestra en la Fig. 6-6(a). La salida *Q* opera AND con las entradas *K* y *CP*, de modo que el flip-flop se despeja durante un pulso de reloj sólo si *Q* era previamente 1. En forma similar, la salida *Q'* opera AND con las entradas *J* y *CP* de modo que el flip-flop se ajusta con un pulso de reloj sólo si *Q'* era previamente 1.

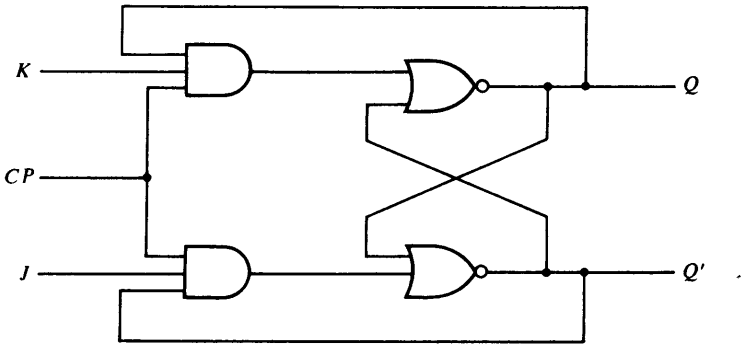
Como se muestra en la tabla característica en la Fig. 6-6(c), el flip-flop *JK* se comporta como un flip-flop *RS* excepto cuando tanto *J* como *K* son iguales a 1. Cuando *J* y *K* son 1, el pulso de reloj se transmite sólo a través de una compuerta AND (la que tenga conectada su entrada a la salida del flip-flop que al presente sea igual a 1). Por tanto, si $Q = 1$, la salida de la compuerta superior AND llega a ser 1 bajo la aplicación de un pulso de reloj, y el flip-flop se despeja. Si $Q' = 1$, la salida de la compuerta AND inferior llega a ser un 1 y el flip-flop se ajusta. En cualquier caso, el estado de la salida del flip-flop se complementa.

Las entradas en el símbolo gráfico para el flip-flop *JK* deben marcarse con una *J* (bajo *Q*) y *K* (bajo *Q'*). La ecuación característica se da en la Fig. 6-4(d) y se deriva mediante el mapa de la tabla característica.

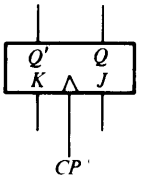
Obsérvese que debido a la conexión de retroalimentación en el flip-flop *JK*, una señal *CP* que permanece en 1 (en tanto $J = K = 1$) una vez que las salidas se han complementado provocará transiciones repetidas y continuas de las salidas. Para evitar esta operación indeseable, los pulsos de reloj deben tener una duración más corta que el retardo de propagación a través del flip-flop. Este es un requisito de restricción, ya que la operación del circuito depende del ancho de los pulsos. Por esta razón, los flip-flop *JK* nunca se construyen como se muestra en la Fig. 6-6(a). La restricción en el ancho del pulso puede eliminarse con una construcción de maestro-esclavo o de disparo en borde, como se expone en la siguiente sección. El mismo razonamiento se aplica al flip-flop *T* que se presenta a continuación.

Flip-flop *T*

El flip-flop *T* es una versión de una sola entrada del flip-flop *JK*. Como se muestra en la Fig. 6-7(a), el flip-flop *T* se obtiene mediante un tipo *JK* si ambas entradas se ligan. La denominación *T* proviene de la capacidad del flip-flop para "conmutar" (de la inicial del término en inglés: *toggle*), o cambiar de estado. Sin importar el estado presente del flip-flop, asume el estado complementario cuando ocurre el pulso de reloj mientras la entrada *T* es lógica 1. El símbolo, la tabla característica y la ecuación característica del flip-flop *T* se muestran en la Fig. 6-7, partes (b), (c) y (d), respectivamente.



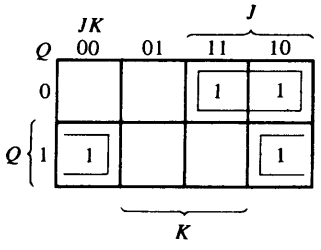
(a) Diagrama lógico



(b) Símbolo gráfico

Q	J	K	Q(t + 1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

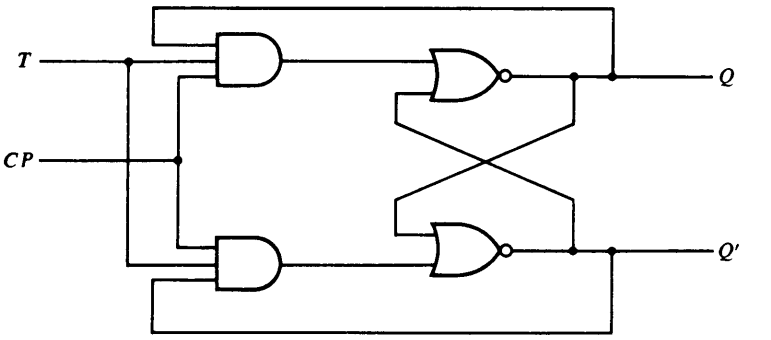
(c) Tabla característica



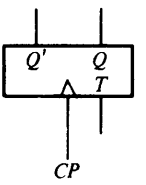
$Q(t + 1) = JQ' + K'Q$

(d) Ecuación característica

Figura 6-6 Flip-flop *JK* con pulsos de reloj.



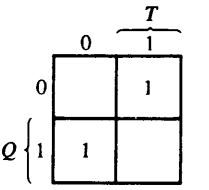
(a) Diagrama lógico



(b) Símbolo gráfico

Q	T	$Q(t+1)$
0	0	0
0	1	1
1	0	1
1	1	0

(c) Tabla característica



$Q(t+1) = TQ' + T'Q$

(d) Ecuación característica

Figura 6-7 Flip-flop T con pulsos de reloj.

Los flip-flop que se introducen en esta sección son los tipos disponibles más comunes en el comercio. Los procedimientos de análisis y diseño que se desarrollan en este capítulo son aplicables para cualquier flip-flop temporizado una vez que se define su tabla característica.

6-3 DISPARO DEL FLIP-FLOP

El estado de un flip-flop se cambia por una modificación momentánea en la señal de entrada. Este cambio momentáneo se denomina *gatillo* y la transición que provoca se dice que dispara al flip-flop. Los flip-flop asíncronos, como los circuitos básicos en la Fig. 6-2 y 6-3, requieren una entrada de gatillo definida por un cambio de *nivel* de señal. Este nivel debe volver a su valor inicial (0 en el flip-flop NOR y 1 en el NAND) antes de que aplique un segundo gatillo. Los flip-flops temporizados se disparan por pulsos. Un pulso comienza desde un valor inicial de 0, pasa en forma momentánea a 1 y después de un corto tiempo, regresa a su valor 0 inicial. El intervalo de tiempo desde la aplicación del pulso hasta que ocurre la transición de la salida es un factor crítico que requiere más investigación.

Como se observó en el diagrama de bloques en la Fig. 6-1, un circuito secuencial tiene una trayectoria de retroalimentación entre el circuito combinacional y los

elementos de memoria. Esta trayectoria puede producir inestabilidad y las salidas de los elementos de memoria (flip-flops) se cambian, mientras las salidas del circuito combinacional que van a las entradas de los flip-flops se muestrean por el pulso de reloj. Este problema de temporizado puede evitarse si las salidas de los flip-flops no inician el cambio sino hasta que el pulso de entrada ha regresado a 0. Para asegurar tal operación, un flip-flop debe tener un retardo de propagación de señal desde la entrada a la salida que exceda la duración del pulso. Este retardo por lo común es muy difícil de controlar si el diseñador depende por completo del retardo de propagación de las compuertas lógicas. Una forma de asegurar el retardo apropiado es incluir dentro del circuito flip-flop una unidad de retardo física que tenga un retardo igual o mayor que la duración del pulso. Una manera adecuada de resolver el problema del temporizado de la retroalimentación es hacer sensitivo al flip-flop a la *transición* del pulso más que a la duración del pulso.

Un pulso de reloj puede ser positivo o bien negativo. Una fuente positiva de reloj permanece en 0 durante el intervalo entre pulsos y pasa a 1 al ocurrir un pulso. El pulso pasa a través de dos transiciones de señal: desde 0 a 1 y el regreso de 1 a 0. Como se muestra en la Fig. 6-8, la transición positiva se define como el *borde positivo* y la transición negativa como el *borde negativo*. Esta definición también se aplica a los pulsos negativos.

Los flip-flops temporizados que se introdujeron en la Sección 6-2 se disparan durante el borde positivo del pulso, y la transición de estado principia tan pronto el pulso alcanza el nivel lógico 1. El nuevo estado del flip-flop puede aparecer en las terminales de salida mientras el pulso de entrada todavía esté en 1. Si las otras entradas del flip-flop cambian mientras el reloj todavía está en 1, el flip-flop iniciará la respuesta a esos nuevos valores y puede ocurrir un nuevo estado de salida. Cuando esto sucede, la salida de un flip-flop no puede aplicarse a las entradas de otro flip-flop cuando el mismo pulso de reloj los dispara a ambos. Sin embargo, si puede hacerse que el flip-flop responda a la transición de borde positiva (o negativa) *solamente*, en lugar de la duración completa del pulso, entonces puede eliminarse el problema de transición múltiple.

Una forma de hacer que el flip-flop responda sólo a una transición de pulso es utilizar un acoplamiento capacitor. En esta configuración, un circuito *RC* (resistor-capacitor) se inserta en la salida de reloj del flip-flop. Este circuito genera un pico

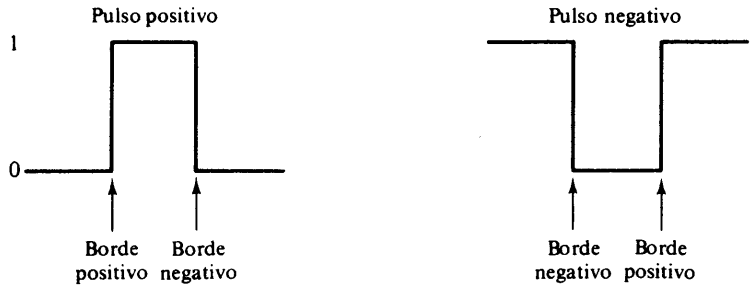


Figura 6-8 Definición de la transición del pulso de reloj.

como respuesta a un cambio momentáneo de la señal de entrada. Un borde positivo emerge de dicho circuito con un pico positivo, y un borde negativo emerge con un pico negativo. El disparo por bordes se logra diseñando el flip-flop de modo que desprecie un pico y dispare una ocurrencia de otro pico. Una forma para lograr el disparo por borde es utilizar un flip-flop maestro-esclavo o disparado por borde como se expone a continuación.

Flip-flop maestro-esclavo

Un flip-flop maestro-esclavo se construye mediante dos flip-flops separados. Un circuito sirve como un maestro y el otro como un esclavo, y el circuito global se conoce como un *flip-flop maestro-esclavo*. El diagrama lógico de un flip-flop maestro-esclavo *RS* se muestra en la Fig. 6-9. Consta de un flip-flop maestro, un flip-flop esclavo y un inversor. Cuando el pulso de reloj *CP* es 0, la salida del inversor es 1. Ya que la entrada de reloj del esclavo es 1, el flip-flop está habilitado si la salida Q es igual a Y, en tanto que Q' es igual a Y'. El flip-flop maestro se habilita porque $CP = 0$. Cuando el pulso llega a 1, entonces la información en las entradas externas R y S se transmite al flip-flop maestro. Sin embargo, el flip-flop esclavo está aislado mientras el pulso esté en su nivel 1, ya que la salida del inversor es 0. Cuando el pulso regresa a 0, el flip-flop maestro está aislado, lo cual evita que lo afecten las entradas externas. El flip-flop esclavo pasa entonces al mismo estado que el del flip-flop maestro.

Las relaciones de temporizados que se muestran en la Fig. 6-10 ilustran la secuencia de eventos que ocurren en un flip-flop maestro esclavo. Se supone que el flip-flop está en el estado despejado antes de la ocurrencia de un pulso, de modo que $Y = 0$ y $Q = 0$. Las condiciones de entrada son $S = 1$, $R = 0$, y el siguiente pulso de reloj cambiará el flip-flop al estado ajustar con $Q = 1$. Mediante la transición de un pulso de 0 a 1, el flip-flop maestro está restaurado y cambia Y a 1. El flip-flop esclavo no es aceptado porque su entrada *CP* es 0. Ya que el flip-flop maestro es un circuito interno, su cambio de estado no es obvio en las salidas Q y Q'. Cuando el pulsor regresa a 0, se permite que la información del maestro pase al esclavo, haciendo que la salida externa

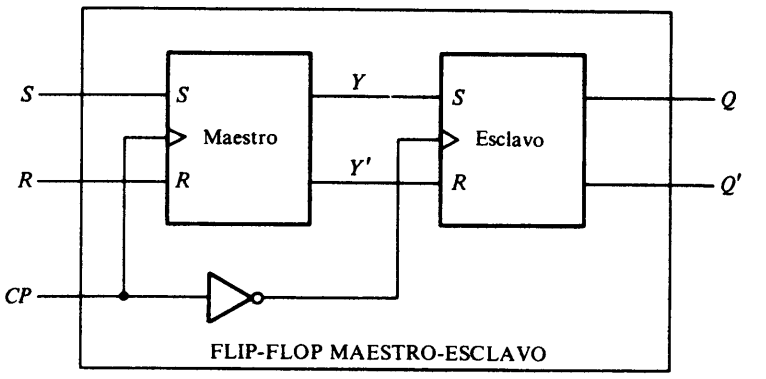


Figura 6-9 Diagrama lógico del flip-flop maestro-esclavo.

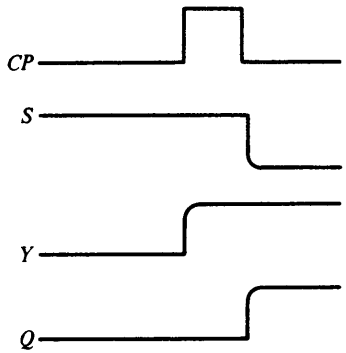


Figura 6-10 Relaciones de tiempos en un flip-flop maestro-esclavo.

sea $Q = 1$. Obsérvese que la entrada externa S debe cambiarse al mismo tiempo que el pulso pasa a través de su transición de borde negativo. Esto se debe a que una vez que la entrada CP alcanza 0, el maestro está inhabilitado y sus entradas R y S no tienen influencia hasta que ocurre el siguiente pulso de reloj. Por eso, en un flip-flop maestro-esclavo es posible cambiar la salida del flip-flop y su información de entrada con el mismo pulso de reloj. Debe tomarse en cuenta que la entrada S puede llegar mediante la salida de otro flip-flop maestro-esclavo que se cambió con el mismo pulso de reloj.

El comportamiento del flip-flop maestro-esclavo que acaba de describirse dicta que los cambios de estado en todos los flip-flops coincidan con la transición de borde negativo del pulso. No obstante, algunos flip-flops maestro-esclavo IC cambian los estados de salida en la transición de borde positivo de los pulsos de reloj. Esto sucede en flip-flops que tienen un inversor adicional entre la CP terminal y la entrada del maestro. Tales flip-flops se disparan con pulsos negativos (véase la Fig. 6-8), de modo que el borde negativo del pulso afecte al maestro y el borde positivo afecte al esclavo y las terminales de salida.

La combinación maestro-esclavo puede construirse para cualquier tipo de flip-flop por la adición de un flip-flop RS temporizado con un reloj invertido para formar el esclavo. Un ejemplo de un flip-flop JK maestro-esclavo construido con compuertas NAND se muestra en la Fig. 6-11. Consta de dos flip-flops; las compuertas 1 a la 4 forman el flip-flop maestro, y las compuertas 5 a la 8 forman el flip-flop esclavo. La información presente en las entradas J y K se transmite al flip-flop maestro en el borde positivo de un pulso de reloj y se sostiene hasta que ocurre el borde negativo del pulso de reloj, después del cual se permite que pase a través del flip-flop esclavo. La entrada de reloj normalmente es 0, lo cual mantiene las salidas de las compuertas 1 y 2 en el nivel. Esto evita que las entradas J y K afecten el flip-flop maestro. El flip-flop esclavo es un tipo RS temporizado, con el flip-flop suministrando las entradas y con la entrada de reloj invertida por la compuerta 9. Cuando el reloj es 0, la salida de la compuerta 9 es 1, de modo que la salida Q es igual a Y y Q' es igual a Y'. Cuando ocurre el borde positivo de un pulso de reloj, el flip-flop maestro se afecta y puede cambiar estados. El flip-flop esclavo está aislado mientras que el reloj esté en el

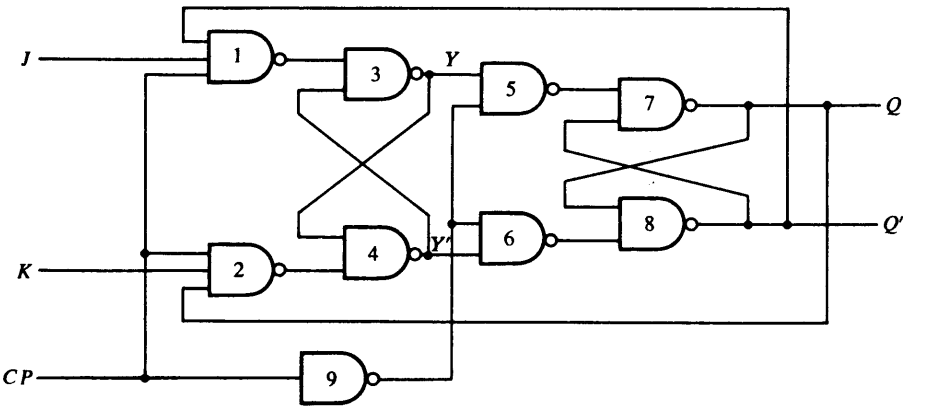


Figura 6-11 Fli-flop maestro-esclavo *JK* con pulsos de reloj.

nivel 1, ya que la salida de la compuerta 9 proporciona un 1 a ambas entradas de las compuertas 7 y 8 NAND flip-flop básico. Cuando la entrada de reloj regresa a 0, el flip-flop maestro está aislado mediante las entradas *J* y *K* y el flip-flop esclavo pasa al mismo estado del flip-flop maestro.

Se considera ahora un sistema digital que contiene muchos flip-flops maestro-esclavo, con las salidas de algunos flip-flops que van a las entradas de otros flip-flops. Se supone que las entradas de pulso de reloj a todos los flip-flops están sincronizadas (ocurren al mismo tiempo). Al principio de cada pulso de reloj, algunos de los elementos maestro cambian estado, pero todas las salidas flip-flop permanecen en sus valores previos. Después de que el pulso de reloj regresa a 0, algunas de las salidas cambian de estado, pero ninguno de los nuevos estados tiene efecto en cualquiera de los elementos maestro hasta el siguiente pulso de reloj. Así que, los estados de flip-flops en el sistema pueden cambiarse en forma simultánea durante el mismo pulso de reloj, aun cuando las salidas de los flip-flops estén conectadas a entradas de flip-flop. Esto es posible ya que el nuevo estado aparece en las terminales de salida sólo después de que el pulso de reloj ha regresado a 0. En consecuencia, el contenido binario de un flip-flop puede transferirse a un segundo flip-flop y el contenido del segundo transferirse al primero, y ambas transferencias pueden ocurrir durante el mismo pulso de reloj.

Flip-flop disparado por borde

Otro tipo de flip-flop que sincorniza los cambios de estado durante la transición de pulsos de reloj es el flip-flop *disparado por borde*. En este tipo de flip-flop, las transiciones de salida ocurren en un nivel específico del pulso de reloj. Cuando el nivel del pulso de entrada excede el nivel umbral, las entradas están bloqueadas y, de este modo, el flip-flop no responde a cambios adicionales en las entradas hasta que el pulso de reloj regresa a 0 y ocurre otro pulso. Algunos flip-flops disparados por borde provocan una transición en el borde positivo del pulso, y otros causan una transición en el borde negativo del pulso.

El diagrama lógico de un flip-flop tipo D disparado por borde positivo se muestra en la Fig. 6-12. Consta de tres flip-flops básicos del tipo que se muestra en la Fig. 6-3. Las compuertas NAND 1 y 2 conforman un flip-flop básico y las compuertas 3 y 4 conforman otro. El tercer flip-flop básico que comprende las compuertas 5 y 6 proporciona las salidas al circuito. Las entradas S y R del tercer flip-flop básico deben mantenerse en lógica 1 para que las salidas permanezcan en sus valores de estado estacionario. Cuando $S = 0$ y $R = 1$, la salida pasa al estado establecido con $Q = 1$. Cuando $S = 1$ y $R = 0$, la salida pasa al estado despejado con $Q = 0$. Las entradas S y R están determinadas mediante los estados de los otros dos flip-flops básicos. Estos dos flip-flops básicos responden a las entradas externas D (datos) y CP (pulso de reloj).

La operación del circuito se explica en la Fig. 6-13, donde las compuertas 1-4 vuelven a dibujarse para mostrar todas sus transiciones posibles. Las salidas S y R de las compuertas 2 y 3 van a las compuertas 5 y 6, como se muestra en la Fig. 6-12, para proporcionar las salidas reales del flip-flop. En la Fig. 6-13(a) se muestran los valores binarios en las salidas de las cuatro compuertas cuando $CP = 0$. La entrada D puede ser igual a 0 o 1. En cualquier caso, un CP de 0 provoca que las salidas de las compuertas 2 y 3 pasen a 1, y así hacen que $S = R = 1$, que es la condición para una salida de estado estacionario. Cuando $D = 0$, la compuerta 4 tiene una salida 1, la cual causa que la salida de la compuerta 1 vaya a 0. Cuando $D = 1$, la compuerta 4 va a 0, la cual provoca que la salida de la compuerta 1 pase a 1. Estas son las dos condiciones posibles cuando el CP terminal, que es 0, inhabilita cualesquiera cambios en las salidas del flip-flop, sin importar cuál es el valor de D.

Hay un tiempo definido, llamado tiempo de *disposición*, en el cual la entrada D debe mantenerse en un valor constante antes de la aplicación del pulso. El tiempo de disposición es igual al retardo de propagación a través de las compuertas 4 y 1, ya que un cambio en D provoca un cambio en las salidas de esas dos compuertas. Ahora se supone que D no cambia durante el tiempo de disposición y que la entrada CP llega

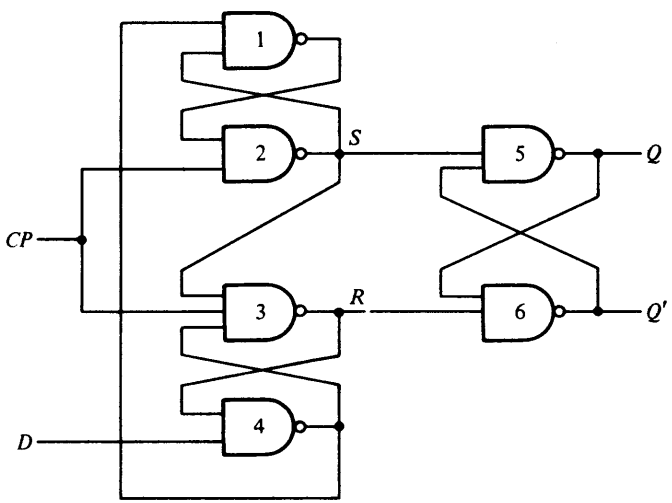
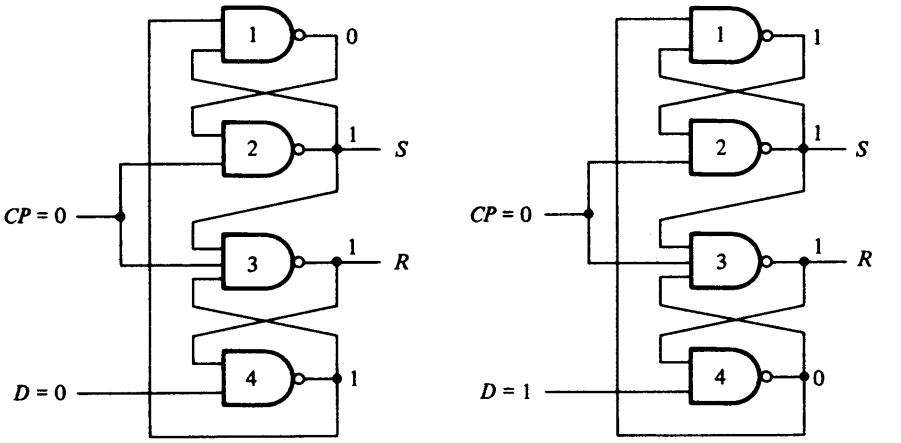
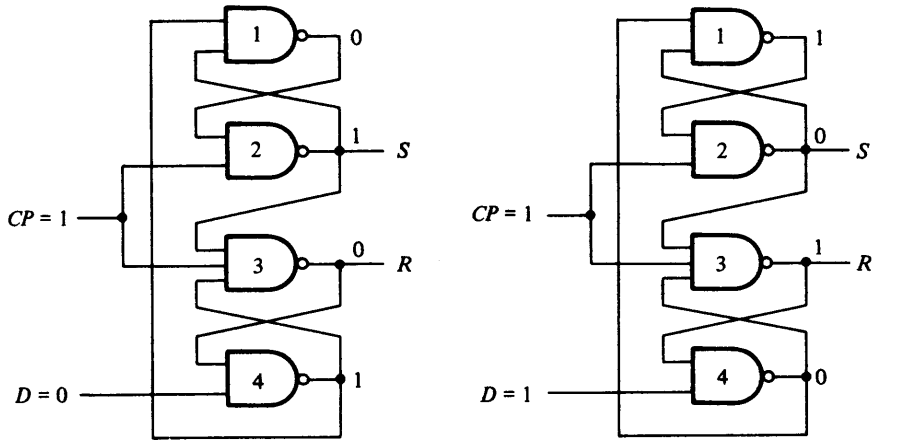


Figura 6-12 Flip-flop tipo D con disparo en borde positivo.



(a) Con $CP = 0$



(b) Con $CP = 1$

Figura 6-13 Operación del flip-flop tipo D con disparo en borde.

a ser 1. Esta situación se indica en la Fig. 6-13(b). Si D=0 cuando el CP llega a ser 1, entonces S permanece en 1 pero R cambia a 0. Esto causa que la salia del flip-flop Q vaya a 0 (en la Fig. 6-12). Si ahora, mientras $CP = 1$, hay un cambio en la entrada D, la salida de la compuerta 4 permanecerá en 1 (incluso si D va a 1), ya que una de las entradas de compuertas viene de R la cual se mantiene en 0. Sólo cuando el CP regresa a 0 puede cambiar la salida de la compuerta 4; pero entonces tanto R como S se vuelven 1, inhabilitando cualquier cambio en la salida de flip-flop. Sin embargo, hay un tiempo definido, llamado el *tiempo de conservación*, en el que la entrada D no debe cambiar después de la aplicación de la transición que va a positivo del pulso. El tiempo de conservación es igual al retardo de propagación de la compuerta 3, ya que debe

asegurarse que R se vuelva 0 con objeto de mantener la salida de la compuerta 4 en 1, con independencia del valor de D.

Si $D = 1$, cuando $CP = 1$, entonces S cambia a 0, pero R permanece en 1, lo cual provoca que las salidas del flip-flop Q vaya a 1. Un cambio en D mientras $CP = 1$ no altera S y R porque la compuerta 1 se mantiene en 1 por la señal 0 de S. Cuando el CP va a 0, tanto S como R van a 1 para evitar que la salida tenga cambios.

En resumen, cuando el pulso de reloj de entrada realiza una transición que va a positivo, el valor de D se transfiere a Q. Los cambios en D cuando CP se mantiene en un valor sostenido de 1, no afectan Q. Por otra parte, una transición de pulso a negativa no afecta la salida, y tampoco cuando $CP = 0$. Siendo así, el flip-flop disparado por borde elimina cualquier problema de retroalimentación en los circuitos secuenciales precisamente como lo hace un flip-flop maestro-esclavo. El tiempo de disposición y el tiempo de conservación deben tomarse en consideración cuando se usa este tipo de flip-flop.

Cuando se usan tipos diferentes de flip-flop en el mismo circuito secuencial, debe tenerse la seguridad de que todas las salidas de los flip-flops hacen sus transiciones al mismo tiempo, esto es, durante ya sea el borde negativo o el positivo del pulso. Los flip-flops que se comportan en forma opuesta respecto a la transición de polaridad adoptada pueden cambiarse con facilidad por la adición de inversores en sus entradas de reloj. Un procedimiento alterno es proporcionar pulsos positivos y negativos (mediante un inversor), y aplicar entonces los pulsos positivos a los flip-flops que disparan durante el borde negativo y pulsos negativos a los flip-flops que disparan durante el borde positivo, o viceversa.

Entradas directas

Los flip-flops disponibles en paquetes IC algunas veces proporcionan entradas especiales para ajustar o despejar el flip-flop en forma asíncrona. Estas entradas por lo común se llaman *preajuste directo* y despeje directo. Afectan el flip-flop en un valor positivo (o negativo) de la señal de entrada sin la necesidad de un pulso de reloj. Estas entradas son útiles para conducir todos los flip-flops a un estado inicial antes de su operación temporizada. Por ejemplo, después que se conecta la potencia en un sistema digital, los estados de sus flip-flops son indeterminados. Un interruptor de despeje limpia todos los flips-flops a un estado inicial despejado y un interruptor de inicio principia la operación temporizada del sistema. El interruptor de despeje debe limpiar todos los flip-flops en forma asíncrona sin la necesidad de un pulso.

El símbolo gráfico de un flip-flop maestro-esclavo con despeje directo se muestra en la Fig. 6-14. El reloj o la entrada CP tiene un círculo bajo el triángulo pequeño para indicar que las salidas cambian durante la transición negativa del pulso (la ausencia de un círculo pequeño indicaría un flip-flop disparado por borde positivo). La entrada de despeje directo también tiene un pequeño círculo para indicar que, en forma normal, esta entrada debe mantenerse en 1. Si la entrada de despeje se mantiene en 0, el flip-flop permanece limpio, independiente de las otras entradas o del pulso de reloj. La tabla de función especifica la operación del circuito. Las X son condiciones no importa que indican que un 0 en la entrada de despeje directo inhabi-

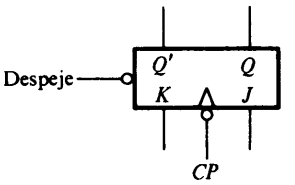


Tabla de función

Entradas				Salidas	
Despeje	Reloj	J	K	Q	Q'
0	X	X	X	0	1
1	↓	0	0	Sin cambio	
1	↓	0	1	0	1
1	↓	1	0	1	0
1	↓	1	1	Lengüeta	

Figura 6-14 Flip-flop *JK* con despeje directo.

lita todas las demás entradas. Sólo cuando la entrada de despeje es 1 puede tener efecto una transición negativa del reloj en las salidas. Las salidas no cambian si $J = K = 0$. El flip-flop cambia o complementa cuando $J = K = 1$. Algunos flip-flops es posible que también tengan una entrada de preajuste directo, la cual establece la salida Q en 1 (y Q' en 0) en forma asíncrona.

Cuando están disponibles entradas directas asíncronas en un flip-flop maestro-esclavo, deben conectarse tanto al maestro como al esclavo con objeto de sobrepasar las otras entradas y el reloj. Un despeje directo en el flip-flop maestro-esclavo *JK* en la Fig. 6-10 se conecta a las entradas de las compuertas 1, 4 y 8. Un despeje directo en el flip-flop disparado en borde *D* en la Fig. 6-12 se conecta a las entradas de las compuertas 2 y 6.

6-4 ANALISIS DE CIRCUITOS SECUENCIALES TEMPORIZADOS

El comportamiento de un circuito secuencial se determina mediante las entradas, las salidas y los estados de sus flip-flops. Tanto las salidas como el estado siguiente son función de las entradas y del estado presente. El análisis de los circuitos secuenciales consiste en obtener una tabla o un diagrama de las secuencias de tiempo de las entradas, salidas y los estados internos. También es posible escribir expresiones booleanas que describen el comportamiento de los circuitos secuenciales. Sin embargo, esas expresiones deben incluir la secuencia de tiempo necesaria ya sea en forma directa o indirecta.

Un diagrama lógico se reconoce como el circuito de un circuito secuencial e incluye los flip-flops. Los flip-flops puede ser de cualquier tipo y el diagrama lógico puede o no incluir las compuertas combinacionales. En esta sección, se introduce primero un ejemplo específico de un circuito secuencial temporizado y entonces se presentan diversos métodos para describir el comportamiento de los circuitos secuenciales. El ejemplo específico se usará en la exposición para ilustrar diversos métodos.

Ejemplo de un circuito secuencial

En la Fig. 6-15 se muestra un ejemplo de un circuito secuencial temporizado. Tiene una variable de entrada x, una variable de salida y y dos flip-flops *RS* temporizados

etiquetados *A* y *B*. Las conexiones cruzadas de las salidas de los flip-flops a las entradas de las compuertas no se muestran por líneas de dibujo, de modo que se facilite el trazado del circuito. En lugar de esto, se reconocen las conexiones por el símbolo de letra que se marca en cada entrada. Por ejemplo, la entrada marcada x' en la compuerta 1 indica una entrada del complemento de x. La segunda entrada marcada A indica una conexión a la salida normal de flip-flop A.

Se supone un disparo de borde negativo tanto en los flip-flops como en la fuente que produce la entrada externa x. En este caso, las señales para un estado presente dado están disponibles durante el tiempo desde la terminación de un pulso de reloj hasta la terminación del siguiente pulso de reloj, a cuvo tiempo el circuito pasa al estado siguiente.

Tabla de estado

La secuencia en tiempo de las entradas, salidas y estados de flip-flop pueden enumerarse en una *tabla de estado*.* La tabla de estado para el circuito en la Fig. 6-15 se muestra en la Tabla 6-1. Consta de tres secciones etiquetadas *estado presente*, *estado siguiente* y *salida*. El *estado presente* indica los estados de los flip-flops antes de la ocurrencia del pulso de reloj. El *estado siguiente* muestra los estados de los flip-flops después de la aplicación de un pulso de reloj, y la sección *de salida* lista los valores de las variables de salida durante el estado presente. Las secciones de estado siguiente al igual que la de salida tienen dos columnas, una para $x = 0$ y la otra para $x = 1$.

* En los libros que tratan la teoría de la conmutación esta tabla se denomina una *tabla de transición*. Reservan el nombre de *tabla de estados* para una tabla con estados internos representados por símbolos arbitrarios.

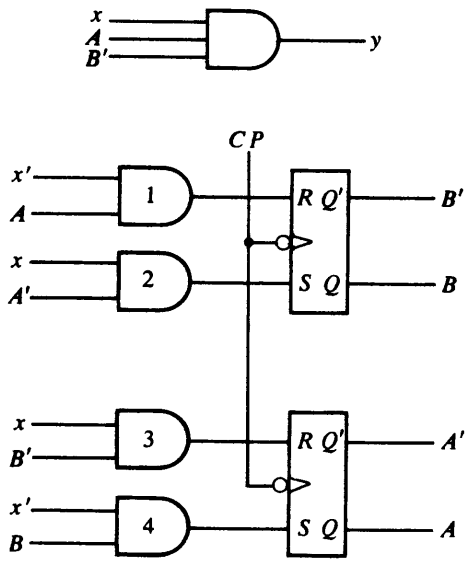


Figura 6-15 Ejemplo de un circuito secuencial con pulsos de reloj.

TABLA 6-1 Tabla de estados para el circuito de la Fig 6-15

	Estado siguiente		Salida	
Estado presente	$x = 0$	$x = 1$	$x = 0$	$x = 1$
AB	AB	AB	y	y
00	00	01	0	0
01	11	01	0	0
10	10	00	0	1
11	10	11	0	0

La derivación de la tabla de estado principia desde un estado inicial supuesto. El estado inicial de la mayoría de los circuitos secuenciales prácticos se define como el estado con números 0 en todos los flip-flops. Algunos circuitos secuenciales tienen un estado inicial diferente y otros no tienen ninguno en absoluto. En cualquier caso, los análisis siempre pueden principiar desde cualquier estado arbitrario. En este ejemplo se principia derivando la tabla de estado desde el estado inicial 00.

Cuando el estado presente es 00, $A = 0$ y $B = 0$. Mediante el diagrama lógico, se ve que ambos flip-flops están despejados y $x = 0$. Ninguna de las compuertas AND produce una señal lógica 1. Por lo tanto, el estado siguiente permanece sin cambio. Con $AB = 00$ y $x = 1$, la compuerta 2 produce una señal lógica 1 y la entrada S del flip-flop B y la compuerta 3 produce una señal lógica 1 a la entrada del flip-flop A. Cuando un pulso de reloj dispara los flip-flops, A se despeja y B está ajustado, haciendo que el estado siguiente sea 01. Esta información se lista en el primer renglón de la tabla de estado.

En forma similar, puede derivarse el siguiente estado principiando desde los otros tres posibles estados presentes. En general, el estado siguiente es una función de las entradas, del estado presente y del tipo de flip-flop que se utilice. Con flip-flops RS, por ejemplo, debe recordarse que un 1 en la entrada S establece el flip-flop y un 1 en la entrada R despeja el flip-flop, sin importar su estado previo. Un 0 en las entradas S y R deja el flip-flop sin cambio, mientras que un tanto en la entrada S como en la R evidencia un mal diseño y una tabla de estado indeterminada.

Las entradas para la sección de salida son fáciles de derivar. En este ejemplo, la salida y es igual a 1 sólo cuando $X = 1$, $A = 1$ y $B = 0$. Por tanto, las columnas de salida se marcan con 0, excepto cuando el estado presente es 10 y la entrada $x = 1$, por lo cual y se marca con un 1.

La tabla de estado de cualquier circuito secuencial se obtiene por el mismo procedimiento que se usa en el ejemplo. En general, un circuito secuencial con m flip-flops y n variables de entrada tendrá 2^m renglones, uno para cada estado. Cada una de las secciones de estado siguiente y salida tendrán 2^n columnas, una para cada combinación de entrada.

Las salidas externas de un circuito secuencial pueden tener procedencia de compuertas lógicas o de elementos de memoria. La sección de salida en la tabla

de estado es necesaria sólo si hay salidas de compuertas lógicas. Cualquier salida externa que se toma en forma directa de un flip-flop ya está listada en la columna de estado presente de la tabla de estado. Por lo tanto, la sección de salida de la tabla de estado puede excluirse si no hay salidas externas de compuertas lógicas.

Diagrama de estado

La información disponible en una tabla de estado puede representarse en forma gráfica en un *diagrama de estado*. En este diagrama, un estado se representa con un círculo y la transición entre estados se indica con líneas dirigidas que conectan los círculos. El diagrama de estado de un círculo secuencial en la Fig. 6-15 se muestra en la Fig. 6-16. El número binario dentro de cada círculo identifica el estado que representa el círculo. Las líneas dirigidas están etiquetadas con dos números binarios separados por una /. El valor de entrada que provoca la transición de estado se etiqueta primero; el número después del símbolo / da el valor de la salida durante el estado presente. Por ejemplo, la línea dirigida desde el estado 00 al 01 se etiqueta 1/0, lo cual significa que el circuito secuencial está en un estado presente 00 mientras $x = 1$ y $y = 0$, y que a la terminación del siguiente pulso de reloj, el circuito pasa al siguiente estado 01. Una línea dirigida que conecta un círculo con sí misma indica que ocurre cambio de estado. El diagrama de estado proporciona la misma información que la tabla de estado y se obtiene en forma directa de la Tabla 6-1.

No hay diferencia entre una tabla de estado y un diagrama de estado excepto en la forma de representación. La tabla de estado es más fácil de derivar mediante un diagrama lógico dado y en el diagrama de estado continúa en forma directa de una tabla de estado. El diagrama de estado da una imagen de las transiciones de estado y se encuentra en una forma adecuada para la interpretación humana de la operación del circuito. El diagrama de estado se utiliza con frecuencia como la especificación inicial de diseño de un circuito secuencial.

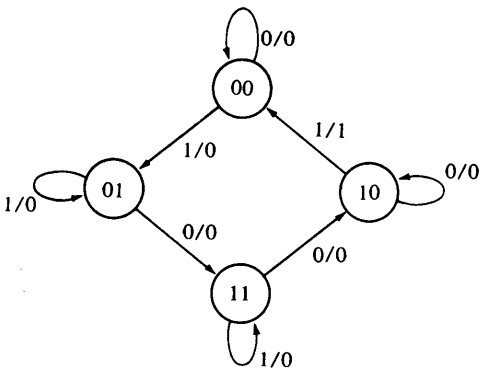


Figura 6-16 Diagrama de estado para el circuito en la Fig. 6-15.

Ecuaciones de estado

Una *ecuación de estado* (también conocida como una *ecuación de aplicación*) es una expresión algebraica que especifica las condiciones para una transición de estado de flip-flop. El primer miembro de la ecuación denota el estado siguiente de un flip-flop en el segundo miembro, una función booleana que especifica las condiciones de estado siguientes que hacen que el siguiente estado sea igual a 1. Una ecuación de estado es similar en forma a una ecuación característica de flip-flop, excepto que especifica las condiciones de estado siguiente en términos de las variables externas de entrada y otros valores del flip-flop. La ecuación de estado se deriva en forma directa mediante una tabla de estado. Por ejemplo, la ecuación de estado para el flip-flop A se deriva mediante la inspección de la Tabla 6-1. En las columnas del estado siguiente, se observa que el flip-flop A pasa al estado 1 cuatro veces: cuando $x = 0$ y $AB = 01$ o 10, o 11 o cuando $x = 1$ y $AB = 11$. Esto puede expresarse en forma algebraica en una ecuación de estado como sigue:

$$A(t + 1) = (A'B + AB' + AB)x' + ABx$$

El segundo miembro de la ecuación de estado es una función booleana para un estado presente. Cuando esta función es igual a 1, la ocurrencia de un pulso de reloj provoca que el flip-flop tenga un estado siguiente de 1. Cuando la función es igual a 0, el pulso de reloj causa que A tenga un estado siguiente de 0. El primer miembro de la ecuación identifica al flip-flop por su símbolo de letra, seguido por la función de tiempo designada $(t + 1)$, para hacer énfasis en que el flip-flop alcanzará este valor una secuencia de pulso después.

La ecuación de estado es una función booleana con tiempo incluido. Es aplicable sólo en los circuitos secuenciales temporizados, ya que $A(t + 1)$ se define para cambiar valor con la ocurrencia de un pulso de reloj en instantes discretos de tiempo.

La ecuación de estado para el flip-flop A se simplifica mediante un mapa como se muestra en la Fig. 6-17(a). Con alguna manipulación algebraica, la función puede expresarse en la siguiente forma:

$$A(t + 1) = Bx' + (B'x)'A$$

Si se hace $Bx' = S$ y $B'x = R$, se obtiene la relación:

$$A(t + 1) = S + R'A$$

que es la ecuación característica de un flip-flop RS [Fig. 6-4(d)]. Esta relación entre la ecuación de estado y la ecuación característica del flip-flop puede simplificarse por la inspección del diagrama lógico en la Fig. 6-19. De esta manera puede verse que la entrada S del flip-flop es igual a la función booleana Bx' y que la entrada R es igual a $B'x$. La sustitución de esas funciones en la ecuación característica del flip-flop produce la ecuación de estado para este circuito secuencial.

La ecuación de estado para un flip-flop en un circuito secuencial puede derivarse de una tabla de estado o de un diagrama lógico. La derivación mediante la tabla de estado consiste en obtener la función booleana especificando las condiciones que hacen que el siguiente estado del flip-flop sea un 1. La derivación mediante un diagrama lógico consiste en obtener las funciones de las entradas del flip-flop y sustituirlas en la ecuación característica de flip-flop.

La derivación de la ecuación de estado para el flip-flop B mediante la tabla de estado se muestra en el mapa de la Fig. 6-17(b). Los 1 marcados en el mapa son el estado presente y las combinaciones de entrada que provocan que el flip-flop pase a un estado siguiente de 1. Estas condiciones se obtienen de manera directa mediante la Tabla 6-1. La forma simplificada que se obtiene en el mapa se manipula algebraicamente, y la ecuación de estado obtenida es:

$$B(t + 1) = A'x + (Ax')'B$$

La ecuación de estado puede derivarse en forma directa mediante el diagrama lógico. A partir de la Fig. 6-15, puede verse que la señal para la entrada S del flip-flop B se genera por la función $A'x$ y la señal para la entrada R por la función Ax'. La sustitución de $S = A'x$ y $R = Ax'$ en la ecuación característica de un flip-flop RS dada por:

$$B(t + 1) = S + R'B$$

se obtiene la ecuación de estado derivada antes.

Las ecuaciones de estado de todos los flip-flops junto con las funciones de salida, especifican en forma completa un circuito secuencial. Representan, algebraicamente, la misma información que una tabla de estado presenta la forma tabular y un diagrama de estado representa en forma gráfica.

Funciones de entrada de un flip-flop

El diagrama lógico de un circuito secuencial consta de elementos de memoria y compuertas. El tipo de flip-flops y sus tablas características especifican las propieda-

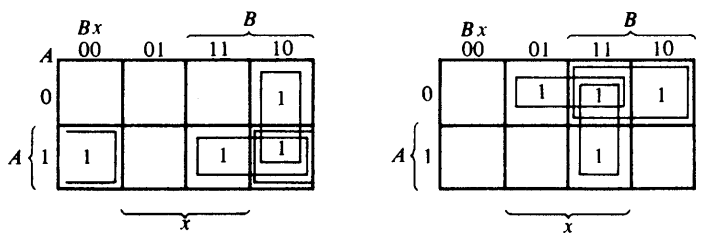


Figura 6-17 Ecuaciones de estado para los flip-flop A y B.

des lógicas de los elementos de memoria. Las interconexiones entre las compuertas forman un circuito combinacional que puede especificarse en forma algebraica con funciones booleanas. Por consiguiente, el conocimiento del tipo flip-flops y una lista de funciones booleanas del circuito combinacional proporcionan toda la información necesaria para dibujar el diagrama lógico de un circuito secuencial. La parte del circuito combinacional que genera las salidas externas se describe en forma algebraica por las *funciones de salida del circuito.* La parte del circuito que genera las entradas a flip-flops se describe de manera algebraica por un conjunto de funciones booleanas llamadas *funciones de entradas de flip-flops* o, algunas veces, *ecuaciones de entrada.*

Se adoptará la convención de usar dos letras para denotar una variable de entrada flip-flop: la primera para designar el nombre de la entrada y la segunda el nombre del flip-flop. Como un ejemplo, considérense las siguientes funciones de entrada flip-flop:

$$JA = BC'x + B'Cx'$$
$$KA = B + y$$

JA y KA denotan dos variables booleanas. La primera letra en cada una denota la entrada J y K, respectivamente, de un flip-flop JK. La segunda letra A es el símbolo del nombre del flip-flop. El segundo miembro de cada ecuación es una función booleana para la variable correspondiente de entrada al flip-flop. La implicación de las dos funciones de entrada se muestra en el diagrama lógico en la Fig. 6-18. El flip-flop JK tiene un símbolo de salida A y las dos entradas etiquetadas J y K. El circuito combinacional dibujado en el diagrama es la implantación de la expresión algebraica dada por las funciones de entrada. Las salidas del circuito combinacional se denotan por JA y KA en las funciones de entrada si van a las entradas J y K, respectivamente, del flip-flop A.

Mediante este ejemplo, puede verse que una función de entrada flip-flop es una expresión algebraica para un circuito combinacional. La designación de dos letras es un nombre de una variable para una *salida* del circuito combinacional. Esta salida siempre está conectada a la *entrada* (denotada por la primera letra) de un flip-flop (designado por la segunda letra).

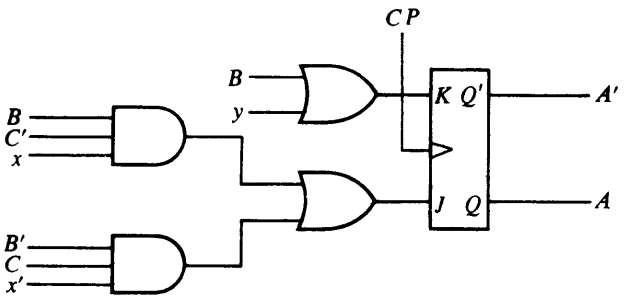


Figura 6-18 Implementación de las funciones de entrada al flip-flop $JA = BC'x + B'Cx'$ y $KA = B + y$.

El circuito secuencial en la Fig. 6-15 tiene una entrada x, una salida y, y dos flip-flops RS denotados por A y B. El diagrama lógico puede expresarse en forma algebraica con cuatro funciones de entrada flip-flop y una función de salida del circuito como sigue:

$$SA = Bx' \qquad RA = B'x$$
$$SB = A'x \qquad RB = Ax'$$
$$y = AB'x$$

Este conjunto de funciones booleanas especifica por completo el diagrama lógico. Las variables SA y RA especifica un flip-flop RS etiquetado A; las variables SB y RB especifican un segundo flip-flop RS etiquetado B. La variable y denota la salida. Las expresiones booleanas para las variables especifican el circuito combinacional que es parte del circuito secuencial.

Las funciones de entrada flip-flop constituyen una forma algebraica conveniente para especificar un diagrama lógico de un circuito secuencial. Implican el tipo de flip-flop mediante la primera letra de las variables de entrada y especifican por completo el circuito combinacional que impulsa al flip-flop. El tiempo no se incluye en forma explícita en estas ecuaciones, pero se implica mediante la operación del pulso de reloj. Algunas veces es conveniente especificar en forma algebraica un circuito secuencial con las funciones de salida del circuito y las funciones de entrada del flip-flop, en lugar de dibujar el diagrama lógico.

6-5 REDUCCION Y ASIGNACION DE ESTADO

El análisis de los circuitos secuenciales principia mediante un diagrama de circuito y culmina en una tabla de estado o diagrama. El diseño de un circuito secuencial se inicia mediante un conjunto de especificaciones y termina en un diagrama lógico. Los procedimientos de diseño se presentan comenzando desde la Sección 6-7. En esta sección se exponen ciertas propiedades de los circuitos secuenciales que pueden usarse para reducir el número de compuertas y de flip-flops durante el diseño.

Reducción de estado*

En cualquier proceso de diseño debe considerarse el problema de minimizar el costo del circuito final. Las dos reducciones de costo más obvias son las reducciones en el número de flip-flops y el número de compuertas. Debido a que estos dos detalles parecen los más evidentes, se han estudiado e investigado extensamente. De hecho, una gran parte del tema de la teoría de conmutación se dedica a la busqueda de algoritmos para minimizar el número de flip-flops y compuertas en los circuitos secuenciales.

* En la Sección 9-5 pueden encontrarse más explicaciones y más ejemplos de reducción de estados.

La reducción del número de flip-flops en un circuito secuencial se conoce como el problema de *reducción de estado*. Los algoritmos de reducción de estado tratan con procedimientos para reducir el número de estados en una tabla de estados mientras se mantienen sin cambio los requisitos de entrada-salida externa. Ya que m flip-flops producen 2^m estados, una reducción en el número de estados puede (o no puede) dar por resultado una reducción en el número de flip-flops. Un efecto no predecible al reducir el número de flip-flops es que algunas veces el circuito equivalente (con menos flip-flops) puede requerir más compuertas combinacionales.

Se ilustrará la necesidad de la reducción de estado con un ejemplo. Se principia con un circuito secuencial cuya especificación está dada en el diagrama de estado en la Fig. 6-19. En este ejemplo, sólo son importantes las secuencias de entrada-salida; los estados internos sólo se usan para proporcionar las secuencias requeridas. Por esta razón, los estados que se marcan dentro de los círculos se denotan por símbolos alfabéticos en lugar de sus valores binarios. Esto es en contraste a un contador binario, donde la secuencia de valores binarios de los estados por sí mismos se toman como las salidas.

Hay un número infinito de secuencias de entrada que pueden aplicarse al circuito, cada una conduce a una secuencia única de salidas. Como ejemplo, considérese la secuencia de entrada 01010110100 principiando desde el estado inicial *A*. Cada entrada de 0 o 1 produce una salida de 0 o 1 y causa que el circuito pase al estado siguiente. Mediante el diagrama de estado, se obtiene la secuencia de salida y estado para la secuencia dada de entrada como sigue: en el circuito en el estado inicial *A*, una entrada de 0 produce una salida de 0 y el circuito permanece en el estado *a*. Con el estado presente *a* y entrada de 1, la salida es 0 y el estado siguiente es *b*. Con el estado presente *b* y la entrada de 0, la salida es 0 y el estado siguiente es *b*. Continuando este proceso se encuentra la secuencia completa como sigue:

estado	*a*	*a*	*b*	*c*	*d*	*e*	*f*	*f*	*g*	*f*	*g*	*a*
entrada	0	1	0	1	0	1	1	0	1	0	0	
salida	0	0	0	0	0	1	1	0	1	0	0	

En cada columna, se tiene el estado presente, el valor de entrada y el valor de salida. El estado siguiente se escribe en la parte superior de la siguiente columna. Es importante tomar en cuenta que en este circuito los estados por sí mismos son de importancia secundaria, ya que se tiene interés sólo en las secuencias de salida que provocan las secuencias de entrada.

Ahora se supone que se ha encontrado un circuito secuencial cuyo diagrama de estado tiene menos de siete estados y se desea compararlos con el circuito cuyo diagrama de estado está dado en la Fig. 6-19. Si se aplican secuencias de entrada idénticas a los dos circuitos y ocurren salidas idénticas para todas las secuencias de entrada, entonces se dice que los dos circuitos son equivalentes (en lo que respecta a la entrada-salida) y uno puede reemplazarse por el otro. El problema de reducción de estado es encontrar formas de reducir el número de estados en un circuito secuencial y alterar las relaciones de entrada-salida.

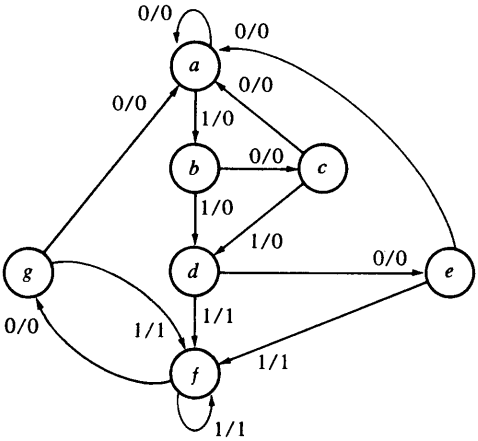


Figura 6-19 Diagrama de estado.

Ahora se procede a reducir el número de estados para este ejemplo. Primero, se necesita la tabla de estado; es más conveniente aplicar los procedimientos para reducción de estados aquí que en los diagramas de estado. La tabla de estado del circuito se lista en la Tabla 6-2 y se obtienen en forma directa mediante el diagrama de estado en la Fig. 6-19.

Aquí se presenta, sin prueba, un algoritmo para la reducción de estado de una tabla de estado por completo especificada: "Se dice que dos estados son equivalentes si, para cada miembro del conjunto de entradas, dan exactamente la misma salida y envían al circuito ya sea al mismo estado o a un estado equivalente. Cuando dos estados son equivalentes, uno de ellos puede eliminarse sin alterar las relaciones de entrada-salida".

Se aplica este algoritmo a la Tabla 6-2. Al pasar a través de la tabla de estado, se buscan dos estados presentes que vayan al mismo estado siguiente que tengan la misma salida para ambas combinaciones de entrada. Los estados *g* y *e* son dos de

TABLA 6-2 Tabla de estados

	Estado siguiente		Salida	
Estado presente	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	*a*	*b*	0	0
b	*c*	*d*	0	0
c	*a*	*d*	0	0
d	*e*	*f*	0	1
e	*a*	*f*	0	1
f	*g*	*f*	0	1
g	*a*	*f*	0	1

dichos estados; ambos van a los estados a y f y tienen salidas de 0 y 1 para $x=0$ y $x=1$, respectivamente. Por eso, los estados g y e son equivalentes; puede eliminarse uno. El procedimiento de eliminar un estado y reemplazarlo por su equivalente se demuestra en la Tabla 6-3. El renglón con el estado presente g se cruza y el estado g se reemplaza por el estado e cada vez que ocurre en las columnas de estados siguientes.

El estado presente f ahora tiene estados siguientes e y f y salidas 0 y 1 para $x=0$ y $x=1$, respectivamente. Los mismos estados siguientes y salidas aparecen en el renglón con el estado presente d. Por lo tanto, los estados f y g son equivalentes. El estado f puede eliminarse y reemplazarse por d. La tabla repetida final se muestra en la Tabla 6-4. El diagrama de estado para la tabla reducida consta sólo de cinco estados y se muestra en la Fig. 6-20. Este diagrama de estado satisface las especificaciones originales de entrada-salida y producirá la secuencia referida de salida para cualquier secuencia dada de entrada. La siguiente lista que se deriva mediante el diagrama de estado en la Fig. 6-20 es para la secuencia de entrada que se utilizó con anterioridad. Se observa que resulta la misma secuencia de salida aunque la secuencia de estado sea diferente:

estado	*a*	*a*	*b*	*c*	*d*	*e*	*d*	*d*	*e*	*d*	*e*	*a*
entrada	0	1	0	1	0	1	1	0	1	0	0	
salida	0	0	0	0	0	1	1	0	1	0	0	

De hecho, esta secuencia es exactamente la misma que se obtuvo de la Fig. 6-19, y se reemplaza e por g y d por f.

Vale la pena observar que la reducción en el número de estados de un circuito secuencial es posible si se tiene sólo interés en las relaciones externas de salida-entrada. Cuando se toman en forma directa salidas externas de los flip-flops, las salidas deben ser independientes del número de estados antes de que se apliquen algoritmos de reducción de estado.

El circuito secuencial de este ejemplo se redujo de siete a cinco estados. En cualquier caso, la representación de los estados con componentes físicos requiere que

TABLA 6-3 Reducción de la tabla de estados

	Estado siguiente		Salida	
Estado presente	$x=0$	$x=1$	$x=0$	$x=1$
a	*a*	*b*	0	0
b	*c*	*d*	0	0
c	*a*	*d*	0	0
d	*e*	*f d*	0	1
e	*a*	*f d*	0	1
f	*g e*	*f*	0	1
g	*a*	*f*	0	1

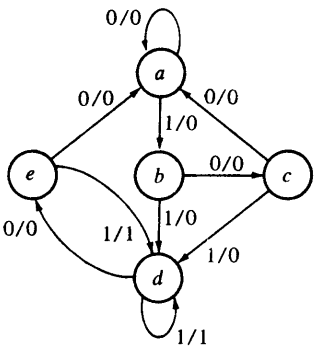


Figura 6-20 Diagrama reducido de estado.

se usen tres flip-flops, ya que m flip-flops pueden representar hasta 2^m estados distintos. Con tres flip-flops, pueden formularse hasta ocho estados binarios denotados por números binarios 000 hasta 111, con cada bit designando el estado de un flip-flop. Si se utiliza la tabla de estado en la Tabla 6-2, deben asignarse valores binarios a siete estados; el estado restante no se usa. Si se utiliza la tabla de estado en la Tabla 6-4, sólo cinco estados necesitan asignación binaria, y quedan tres estados sin uso. Los estados sin uso se tratan como condiciones no importa durante el diseño del circuito. Ya que las condiciones no importa por lo común ayudan a obtener funciones booleanas más simples, es más probable que el circuito con cinco estados requiera menos compuertas combinacionales que el circuito con siete estados. En cualquier caso, la reducción de siete a cinco estados no reduce el número de flip-flops. En general, la reducción del número de compuertas en una tabla de estado es probable que resulte en un circuito con menos equipo. Sin embargo, el hecho de que una tabla de estado se ha reducido a menos estados no garantiza un ahorro en el número de flip-flops o de compuertas.

Asignación de estado

El costo de un circuito combinacional parte de un circuito secuencial puede reducirse por el uso de métodos conocidos de simplificación para los circuitos combinacionales.

Tabla 6-4 Tabla reducida de estados

	Estado siguiente		Salida	
Estado presente	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	*a*	*b*	0	0
b	*c*	*d*	0	0
c	*a*	*d*	0	0
d	*e*	*d*	0	1
e	*a*	*d*	0	1

Sin embargo, hay otro factor, conocido como el problema de *asignación de estado*, que entra en juego al minimizar las compuertas combinacionales. Los procedimientos de asignación de estado se ocupan con métodos para asignar valores binarios a los estados, en tal forma que reducen el costo de un circuito combinacional que impulsa a los flip-flops. Esto es de particular ayuda cuando se considera un circuito secuencial desde sus terminales externas de entrada-salida. Tal circuito puede seguir una secuencia de estados internos, pero los valores binarios de los estados individuales puede no ser de consecuencia mientras el circuito produzca la secuencia referida de salida para una secuencia dada de entradas. Esto no se aplica a circuitos cuyas salidas externas se aplican de manera directa mediante flip-flops con secuencias binarias especificadas por completo.

Las alternativas disponibles de asignación al estado binario pueden demostrarse junto con el circuito secuencial que se especifica en la Tabla 6-4. Recuérdese que, en este ejemplo, los valores binarios de los estados son inmateriales mientras que su secuencia mantenga las relaciones apropiadas de entrada-salida. Por esta razón, cualquier asignación de número binario es satisfactoria en tanto que cada estado esté asignado a un número único. En la Tabla 6-5 se muestran tres ejemplos de asignaciones binarias posibles para los cinco estados de la tabla reducida. La asignación 1 es una asignación binaria directa para la secuencia de estados desde *a* hasta *e*. Las otras dos asignaciones se eligen en forma arbitraria. De hecho, hay 140 diferentes asignaciones distintas para este circuito (11).

La Tabla 6-6 es la tabla de estado reducida con asignación binaria 1 sustituida por los símbolos de letra de los cinco estados.* Es obvio que una asignación binaria diferente causará una tabla de estado con distintos valores binarios para los estados, en tanto las relaciones de entrada-salida permanezcan iguales. La forma binaria de la tabla de estado se utiliza para derivar el circuito combinacional parte del circuito secuencial. La complejidad del circuito combinacional depende de la asignación binaria de estado que se escoja. El diseño del circuito secuencial que se presenta en esta sección se completa en el Ejemplo 6-1 de la Sección 6-7.

Se han sugerido diversos procedimientos que conducen a una asignación binaria particular de las muchas disponibles. El criterio más común es que la asignación que se escoja debe producir un circuito combinacional simple para las entradas flip-flop. Sin

* Una tabla de estado estable con asignación binaria se denomina en ocasiones *tabla de transición*.

TABLA 6-5 Asignaciones de tres estados posibles

Estado	Asignación 1	Asignación 2	Asignación 3
a	001	000	000
b	010	010	100
c	011	011	010
d	100	101	101
e	101	111	011

embargo, a la fecha, no hay procedimientos de asignación de estado que garanticen un circuito combinacional de mínimo costo. La asignación de estado es uno de los problemas de reto de la teoría de conmutación. El lector interesado encontrará una rica y cada vez más abundante literatura sobre este tema. Las técnicas para tratar los problemas de asignación de estado rebasan el alcance de este libro.

6-6 TABLAS DE EXCITACION FLIP-FLOP

Las tablas características de los diversos flip-flops se presentaron en la Sección 6-2. Una tabla característica define la propiedad lógica del flip-flop y caracteriza por completo su operación. Los circuitos integrados flip-flops algunas veces se definen por una tabla característica tabulada en forma un poco diferente. Esta segunda forma de las tablas características para los flip-flops *RS*, *JK*, *D* y *T* se muestra en la Tabla 6-7. Representa la misma información que las tablas características en las Figs. 6-4(c) a la 6-7(c).

En la Tabla 6-7 se define el estado de cada flip-flop como una función de sus entradas y su estado previo. $Q(t)$ se refiere al estado presente y $Q(t+1)$ al siguiente estado después de la ocurrencia de un pulso de reloj. La tabla característica para los flip-flops *RS* muestra que el estado siguiente es igual al estado presente cuando tanto la entrada *S* como la *R* son 0. Cuando la entrada *R* es igual a 1, el siguiente pulso de reloj despeja el flip-flop. Cuando la entrada *S* es igual a 1, el siguiente pulso de reloj establece el flip-flop. El signo de interrogación para el siguiente estado cuando tanto *S* como *R* son iguales a 1 en forma simultánea designa un estado siguiente indeterminado.

La tabla para el flip-flop *JK* es la misma que para el *RS*, cuando *J* y *K* se reemplazan por *S* y *R*, respectivamente, excepto para el caso indeterminado. Cuando tanto *J* como *K* son iguales a 1, el estado siguiente es igual al complemento del estado presente, esto es, $Q(t+1) = Q'(t)$. El estado siguiente del flip-flop *D* depende por completo de la entrada *D* y es independiente del estado presente. El siguiente estado del flip-flop *T* es el mismo que el estado presente si $T = 0$ y se complementa si $T = 1$.

La tabla característica es útil para análisis y para definir la operación del flip-flop. Especifica el estado siguiente cuando las entradas del estado presente se conocen. Durante el proceso de diseño, por lo común se conoce la transición del

TABLA 6-6 Tabla reducida de estados con la asignación binaria 1

	Estado siguiente		Salida	
Estado presente	$x = 0$	$x = 1$	$x = 0$	$x = 1$
001	001	010	0	0
010	011	100	0	0
011	001	100	0	0
100	101	100	0	1
101	001	100	0	1

TABLA 6-7 Tablas flip-flop características

S	R	$Q(t+1)$
0	0	$Q(t)$
0	1	0
1	0	1
1	1	?

(a) RS

J	K	$Q(t+1)$
0	0	$Q(t)$
0	1	0
1	0	1
1	1	$Q'(t)$

(b) JK

D	$Q(t+1)$
0	0
1	1

(c) D

T	$Q(t+1)$
0	Q(t)
1	Q'(t)

(d) T

estado presente al estado siguiente y se desea encontrar las condiciones de entrada del flip-flop, que provocarán la transición requerida. Por esta razón, se necesita una tabla que liste las entradas requeridas para un cambio dado de estado. Dicha lista se denomina tabla de excitación.

En la Tabla 6-8 se presentan las tablas de excitación para los cuatro flip-flops. Cada tabla consta de dos columnas, $Q(t)$ y $Q(t+1)$, y una columna para cada entrada para mostrar cómo se logra la transición requerida. Hay cuatro transiciones posibles desde el estado presente al estado siguiente. Las condiciones requeridas de entrada para cada una de las cuatro transiciones se derivan mediante la información disponible en la tabla característica. El símbolo X en las tablas representa condiciones no importa, esto es, no importa si la entrada es 1 o 0.

Flip-flop *RS*

La tabla de excitación para el flip-flop RS se muestra en la Tabla 6-8(a). En el primer renglón se muestra el flip-flop en el estado 1 en el tiempo t. Se desea dejarlo en el estado 0 después de la ocurrencia del pulso. Mediante la tabla característica, se encuentra que si S al igual que R son 0, el flip-flop no cambiará de estado. En consecuencia, tanto la entrada S como la R deben 0. Sin embargo, en realidad no importa si R se hace un 1, cuando ocurre el pulso, ya que resulta en que deja el flip-flop en el estado, 0. Por eso R puede ser 1 o 0 y el flip-flop permanecerá en el estado 0 en $t+1$. Así, la entrada bajo R se marca X como condición no importa.

Si el flip-flop está en el estado 0 y se desea que pase al estado 1, entonces mediante la tabla característica, se encuentra que la única forma de hacer $Q(t+1)$ igual a 1 es hacer $S=1$ y $R=0$. Si el flip-flop va a tener una transición del estado 1 al estado 0, debe tenerse $S=0$ y $R=1$.

La última condición que puede ocurrir es para que el flip-flop esté en el estado 1 y permanezca en el estado 1. Por supuesto R debe ser 0; no se desea despejar el flip-flip. Sin embargo, S puede ser ya sea un 0 o un 1. Si es 0, el flip-flop no cambia y permanece en el estado 1; si es un 1, se establece el flip-flop en el estado 1, como se desea. Así que, S se lista como una condición no importa.

Flip-flop *JK*

La tabla de excitación para el flip-flip JK se muestra en la Tabla 6-8(b). Cuando tanto el estado presente como el estado siguiente son 0, la entrada J deberá permanecer en 0 y la entrada K podrá ser 0 o bien 1. En forma similar, cuando tanto el estado presente como el siguiente son 1, la entrada K debe permanecer en 0 mientras la entrada J puede ser 0 o 1. Si el flip-flop va a tener una transición del estado 0 al estado 1, J debe ser igual a 1 ya que la entrada J establece el flip-flop. No obstante, la entrada K puede ser 0 o bien un 1. Si $K = 0$, condición la $J = 1$ establece el flip-flop cuando se requiere; si $K = 1$ y $J = 1$, el flip-flop está complementado y pasa del estado 0 al estado 1 cuando se requiere. En este caso, la entrada K se marça con una condición no importa para la transición de 0-a-1. Para una transición del estado 1 al estado 0, debe tenerse $K = 1$, ya que la entrada K despeja el flip-flop. Sin embargo la entrada J puede ser 0 o bien 1, ya que $J = 0$ no tiene efecto, $J = 1$ junto con $K = 1$ complementa el flip-flop con una transición resultante del estado 1 al estado 0.

La tabla de excitación para el flip-flop JK ilustra la ventaja de usar este tipo cuando se diseña en circuitos secuenciales. El hecho de que tiene muchas condiciones no importa indica que los circuitos combinacionales para las funciones de entrada son

TABLA 6-8 Tablas de excitación flip-flop

$Q(t)$	$Q(t+1)$	S	R
0	0	0	*X*
0	1	1	0
1	0	0	1
1	1	*X*	0

(a) *RS*

$Q(t)$	$Q(t+1)$	J	K
0	0	0	*X*
0	1	1	*X*
1	0	*X*	1
1	1	*X*	0

(b) *JK*

$Q(t)$	$Q(t+1)$	D
0	0	0
0	1	1
1	0	0
1	1	1

(c) *D*

$Q(t)$	$Q(t+1)$	T
0	0	0
0	1	1
1	0	1
1	1	0

(d) *T*

susceptibles de ser más simples debido a que los términos no importa por lo común simplifican a una función.

Flip-flop *D*

La tabla de excitación para el flip-flop *D* se muestra en la Tabla 6-8(c). Mediante la tabla característica, Tabla 6-7(c), se observa que el estado siguiente siempre es igual a la entrada *D* y es independiente del estado presente. De este modo, *D* debe ser 0 si $Q(t + 1)$ ha de ser 0, y 1 si $Q(t + 1)$ tiene que ser 1, sin importar el valor de $Q(t)$.

Flip-flop *T*

La tabla de excitación para el flip-flop *T* se muestra en la Tabla 6-8(d). Mediante la tabla característica, Tabla 6-7(d), se encuentra que cuando la entrada $T = 1$, el estado del flip-flop se complementa; cuando $T = 0$, el estado del flip-flop permanece sin cambio. En consecuencia, cuando el estado del flip-flop debe permanecer igual, el requisito es que $T = 0$. Cuando el estado del flip-flop tiene que complementarse, *T* debe ser igual a 1.

Otros flip-flops

El procedimiento de diseño que se describe en este capítulo puede usarse con cualquier flip-flop. Es necesario conocer la tabla característica del flip-flop de la cual es posible desarrollar una nueva tabla de excitación. La tabla de excitación se utiliza entonces para determinar las funciones de entrada del flip-flop, como se explica en la siguiente sección.

6-7 PROCEDIMIENTO DE DISEÑO

El diseño de un circuito secuencial temporizado principia mediante un conjunto de especificaciones y culmina en un diagrama lógico o en una lista de funciones booleanas, mediante las cuales puede obtenerse el diagrama lógico. En contraste con un circuito combinacional, que está especificado por completo por una tabla de verdad, un circuito secuencial requiere una tabla de estado para su especificación. El primer paso en el diseño de circuitos secuenciales es obtener una tabla de estado o una reprentación equivalente, como por ejemplo un diagrama de estado o ecuaciones de estado.

Un circuito secuencial síncrono está hecho de flip-flops y compuertas combinacionales. El diseño del circuito consiste en escoger los flip-flops y entonces encontrar una estructura de compuertas combinacionales que, junto con los flip-flops, producen un circuito que cumple con las especificaciones establecidas. El número de flip-flops se determina mediante el número de estados necesarios en el circuito. El circuito combinacional se deriva mediante la tabla de estado por métodos que se presentan en este capítulo. De hecho, una vez que se determina el tipo y número de flip-flops, el proceso

de diseño implica una transformación del problema del circuito secuencial en un problema de circuito combinacional. En esta forma, pueden aplicarse las técnicas del circuito combinacional.

En esta sección se presenta un procedimiento para el diseño de circuitos secuenciales. Aunque se intenta que sirva como una guía para el principiante, este procedimiento puede reducirse con la experiencia. El procedimiento primero se resume en una lista de pasos consecutivos que se recomiendan como sigue:

1. La descripción verbal del comportamiento del circuito se establece. Esta descripción puede ir acompañada con un diagrama de estado, un diagrama de temporizado o bien otra información pertinente.
2. A partir de la información dada sobre el circuito, se obtiene la tabla de estado.
3. El número de estados puede reducirse por los métodos de reducción de estado y el circuito secuencial puede caracterizarse por las relaciones entrada-salida independientemente del número de estados.
4. Se asignan valores binarios a cada estado si la tabla de estado obtenida en los pasos 2 o 3 contiene símbolos alfabéticos.
5. Se determina el número de flip-flops necesarios y se asigna un símbolo alfabético a cada uno.
6. Se escoge el tipo de flip-flop que va a usarse.
7. Mediante la tabla de estado se derivan las tablas de excitación y salida del circuito.
8. Por el uso del método de mapa o cualquier otro método de simplificación, se derivan las funciones de salida y las de entrada del flip-flop.
9. Se dibuja el diagrama lógico.

La especificación verbal del comportamiento del circuito por lo común supone que el lector está familiarizado con la terminología de la lógica digital. Es necesario que el diseñador utilice su intuición y experiencia para llegar a la interpretación correcta de las especificaciones del circuito, debido a que las descripciones verbales pueden ser incompletas e inexactas. Sin embargo, una vez que se ha establecido dicha especificación y se ha obtenido la tabla de estado, es posible hacer uso del procedimiento formal para diseñar el circuito.

La reducción del número de estados y la asignación de valores binarios a los estados se expuso en la Sección 6-5. En los ejemplos que siguen se supone que se conocen el número de estado y la asignación binaria para los estados. Como consecuencia, los pasos 3 y 4 del diseño no se considerarán en las exposiciones subsecuentes.

Ya se ha mencionado que el número de flip-flops está determinado por el número de estados. Un circuito puede tener estados binarios sin usar, si el número total de estados es menor que 2^m. Los estados que no se usan se toman como condiciones no importa durante el diseño del circuito combinacional parte del circuito.

El tipo de flip-flop que va a usarse puede incluirse en las especificaciones de diseño o puede depender de lo que esté disponible para el diseñador. Muchos sistemas digitales se construyen por completo con flip-flops *JK* porque son el tipo más versátil disponible. Cuando están disponibles muchos tipos de flip-flops, es aconsejable usar el flip-flop *RS* o *D* para aplicaciones que requieren transferencia de datos (como registradores con corrimiento), el tipo *T* para aplicaciones que implican complementación (como contadores binarios) y el tipo *JK* para aplicaciones generales.

La información de la salida externa se especifica en la sección de salida en la tabla de estado. Mediante ella pueden derivarse las funciones de salida del circuito. La tabla de excitación para el circuito es similar a la de los flip-flops individuales, excepto que las condiciones de entrada están listadas por la información disponible en las columnas de estado presente y estado siguiente de la tabla de estado. El método para obtener la tabla de excitación y las funciones simplificadas de entrada flip-flops se ilustran mejor con un ejemplo.

Se desea diseñar el circuito secuencial temporizado cuyo diagrama de estado está dado en la Fig. 6-21. El tipo de flip-flop que va a usarse es *JK*.

El diagrama de estado consta de cuatro estados con valores binarios ya asignados. Ya que las líneas dirigidas están marcadas con un solo dígito binario sin una /, se concluye que hay una variable de entrada y no hay variables de salida. (El estado de los flip-flops puede considerarse como las salidas del circuito.) Los dos flip-flops necesarios para representar los cuatro estados se denotan *A* y *B*. La variable de estado se denota *x*.

La tabla de estado para este circuito, derivada mediante el diagrama de estado, se muestra en la Tabla 6-9. Obsérvese que no hay sección de salida para este circuito. Ahora se mostrará el procedimiento para obtener la tabla de excitación y la estructura de compuertas combinacionales.

La derivación de la tabla de excitación se facilita si se ordena la tabla de estado en una forma diferente. Esta forma se muestra en la Tabla 6-10, donde el estado presente y las variables de entrada están ordenadas en la forma de una tabla de verdad. El valor del estado siguiente para cada estado presente y las condiciones de entrada se copian de la Tabla 6-9. La tabla de excitación de un circuito es una lista de las

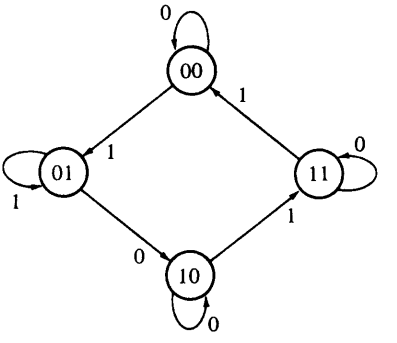


Figura 6-21 Diagrama de estado.

TABLA 6-9 Tabla de estado

Estado presente		Estado siguiente			
		$x = 0$		$x = 1$	
A	B	A	B	A	B
0	0	0	0	0	1
0	1	1	0	0	1
1	0	1	0	1	1
1	1	1	1	0	0

condiciones de entrada flip-flop que causarán las transiciones requeridas de estado y es una función del tipo de flip-flop que se utilice. Ya que este ejemplo especifica flip-flops JK, se necesitan columnas para las entradas J y K de los flip-flops A (denotadas por JA y KA) y B (denotadas por JB y KB).

La tabla de excitación para el flip-flop JK se derivó en la Tabla 6-8(b). Esta tabla se usa ahora para derivar la tabla de excitación del circuito. Por ejemplo, en el primer renglón de la Tabla 6-10 se tiene una transición para el flip-flop A desde 0 en el estado presente hasta 0 en el estado siguiente. En la Tabla 6-8(b) se encuentra que una transición de estado desde 0 hasta 0 requiere que la entrada $J = 0$ y la entrada $K = X$, de modo que 0 y X se copian del primer renglón bajo JA y KA, respectivamente. Ya que el primer renglón también muestra una transición para el flip-flop B desde 0 en el estado presente hasta 0 en el estado siguiente, 0 y X se copian en el primer renglón bajo JB y

TABLA 6-10 Tabla de excitación

Entradas del circuito combinacional			Estado siguiente		Salidas del circuito combinacional			
Estado presente		Entrada			Entradas flip-flop			
A	B	x	A	B	JA	KA	JB	KB
0	0	0	0	0	0	X	0	X
0	0	1	0	1	0	X	1	X
0	1	0	1	0	1	X	X	1
0	1	1	0	1	0	X	X	0
1	0	0	1	0	X	0	0	X
1	0	1	1	1	X	0	1	X
1	1	0	1	1	X	0	X	0
1	1	1	0	0	X	1	X	1

KB. El segundo renglón de la Tabla 6-10 muestra una transición para el flip-flop *B* desde 0 en el estado presente hasta 1 en el estado siguiente. Mediante la Tabla 6-8(b) se encuentra que una transición desde 0 hasta 1 requiere que la entrada $J = 1$ y la entrada $K = X$. De modo que 1 y *X* se copian en el segundo renglón bajo *JB* y *KB*, respectivamente. Este proceso se continúa para cada renglón de la tabla y para cada flip-flop, con las condiciones de entrada como se especifican en la Tabla 6-8(b) que se copian en el renglón apropiado del flip-flop particular que se esté considerando.

Se hace ahora una pausa y se considera la información disponible en una tabla de excitación como la Tabla 6-10. Se sabe que un circuito secuencial consta de un número de flip-flops y un circuito combinacional. En la Fig. 6-22 se muestran los dos flip-flops *JK* necesarios para el circuito y una caja que representa el circuito combinacional. Mediante el diagrama de bloques, es claro que las salidas del circuito combinacional van a las entradas flip-flop y las salidas externas (si se especifica). Las entradas al circuito combinacional son las entradas externas y los valores de estado presentes de los flip-flops. Además, las funciones booleanas que especifican un circuito combinacional se derivan mediante una tabla de verdad que muestra las relaciones de entrada-salida del circuito. La tabla de verdad que describe al circuito combinacional está disponible en la tabla de excitación. Las *entradas* al circuito combinacional se especifican bajo las columnas de estado presente y entrada, y las *salidas* del circuito combinacional se especifican bajo las columnas de entrada flip-flop. Por lo tanto, una tabla de excitación transforma un diagrama de estado en la tabla de verdad necesaria para el diseño del circuito combinacional parte del circuito secuencial.

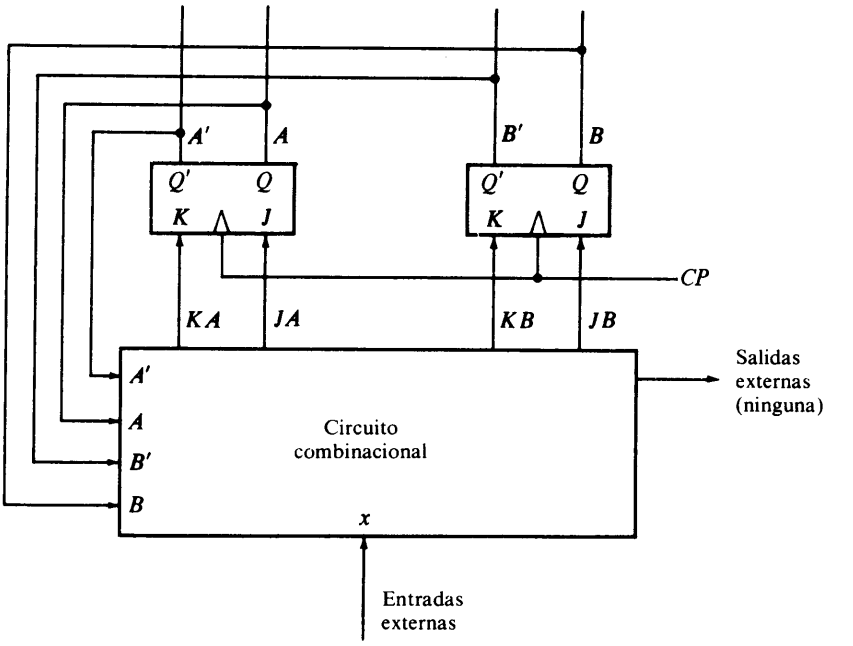


Figura 6-22 Diagrama de bloques del circuito secuencial.

Las funciones booleanas simplificadas para el circuito combinacional pueden derivarse ahora. Las entradas son las variables A, B y x; las salidas son las variables JA, KA, JB y KB. La información de la tabla de verdad se transfiere a los mapas en la Fig. 6-23, donde se derivan las cuatro funciones simplificadas de entrada flip-flop:

$$JA = Bx' \qquad KA = Bx$$
$$JB = x \qquad KB = A \odot x$$

El diagrama lógico se dibuja en la Fig. 6-24 y consta de dos flip-flops, dos compuertas AND, una compuerta de equivalencia y un inversor.

Con alguna experiencia, es posible reducir la cantidad de trabajo implicado en el diseño del circuito combinacional. Por ejemplo, es posible obtener la información de los mapas en la Fig. 6-23 en forma directa en la Tabla 6-9, sin tener que derivar la Tabla 6-10. Esto se hace por el paso sistemático a través de cada estado presente y combinación de entrada en la Tabla 6-9 y comparando con los valores binarios del estado siguiente correspondiente. Entonces, se determinan las condiciones requeridas de entrada como las especifica la excitación flip-flop en la Tabla 6-8. En lugar de insertar los 0, 1 o X obtenidos dentro de la tabla de excitación, pueden escribirse en forma directa en el cuadro adecuado del mapa apropiado.

La tabla de excitación de un circuito secuencial con m flip-flops, k entradas por flip-flop y n impulsos externos está formada por $m + n$ columnas para el estado presente y variables de entrada y hasta 2_{m+n} renglones listados en alguna cuenta binaria conveniente. La sección de estado siguiente tiene m columnas, una para cada flip-flop. Los valores de entrada flip-flop se listan en mk columnas, una para cada entrada de cada flip-flop. Si el circuito contiene j salidas, la tabla debe incluir j columnas. La tabla de verdad del circuito combinacional se toma de la tabla de excitación considerando los $m + n$ estados presentes y columnas de entrada como entradas y los $mk + j$ valores de entrada flip-flop y las salidas externas como salidas.

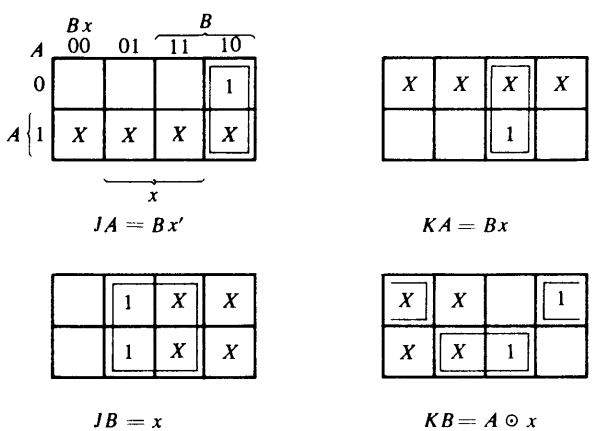


Figura 6-23 Mapas para el circuito combinacional.

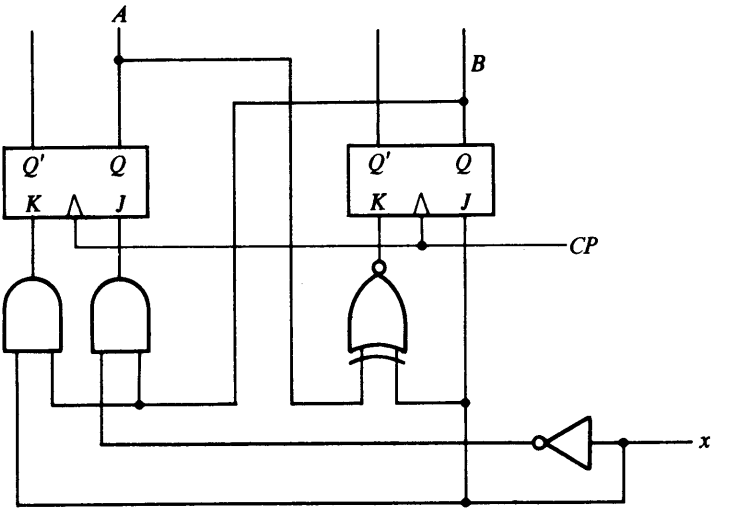


Figura 6-24 Diagrama lógico del circuito secuencial.

Diseño con estados sin uso

Un circuito con m flip-flops puede tener 2^m estados. Hay ocaciones en las que un circuito secuencial puede utilizar menos de este número máximo de estado. Los estados que no se usan en la especificación del circuito secuencial no se listan en la tabla de estado. Cuando se simplifican las funciones de entrada a los flip-flops, los estados sin uso pueden tratarse como condiciones no importa.

EJEMPLO 6-1: Complete el diseño del circuito secuencial que se presenta en la Sección 6-5. Use la tabla de estado reducida con asignación 1 como se da en la Tabla 6-6. El circuito empleará flip-flops *RS*.

La tabla de estados de la Tabla 6-6 vuelve a dibujarse en la Tabla 6-11 en la forma conveniente para obtener la tabla de excitación. Las condiciones de entrada flip-flop se derivan mediante las columnas de estado presente y estado siguiente de la tabla de estado. Ya que se usan flip-flops *RS*, se necesita consultar la Tabla 6-8(a) para las condiciones de excitación de este tipo de flip-flops. Los tres flip-flops reciben nombres de variables *A*, *B* y *C*. La variable de entrada es *x* y la variable de salida es *y*. La tabla de excitación del circuito proporciona toda la información necesaria para el diseño.

Hay tres estados sin usar de este circuito: estados binarios 000, 110 y 111. Cuando se incluye una entrada de 0 o 1 con estos estados no usados, se obtienen seis mintérminos no importa 0, 1, 12, 13, 14 y 15. Estas seis combinaciones binarias no se listan en la tabla bajo estado presente y entrada y se tratan como términos no importa.

El circuito combinacional parte del circuito secuencial se simplifica en los mapas de la Fig. 6-25. Hay siete mapas en el diagrama: seis mapas son para simplificar las funciones de entrada para los tres flip-flops *RS*. El séptimo mapa es para simplificar la salida *y*. Cada mapa tiene seis *X* en los cuadros de los mintérminos no importa 0, 1, 2, 13, 14 y 15. Los otros términos no importa en el mapa provienen de las *X* en las columnas de entrada flip-flop de la tabla. Las funciones simplificadas se listan bajo cada mapa. El diagrama lógico obtenido mediante esas funciones booleanas se dibuja en la Fig. 6-26.

Un factor que hasta este punto no se ha considerado en el diseño es el estado inicial de un circuito secuencial. Cuando se conecta primero la potencia en un circuito digital, no se conoce en qué estado se asentarán los flip-flops. Es costumbre proporcionar una entrada maestra de restaurar cuyo propósito es inicializar los estados de todos los flip-flops en el sistema. En forma típica, la señal maestra de restaurar se aplica a todos los flip-flops en forma síncrona antes de iniciar las operaciones temporizadas. En la mayoría de los casos los flip-flops se despejan en 0 por la señal maestra de restaurar pero algunos pueden ajustarse en 1. Por ejemplo, el circuito en la Fig. 6-26 puede restaurarse en forma inicial en un estado $ABC = 001$, ya que el estado 000 no es un estado válido para este circuito.

¿Pero, qué pasa si un circuito no se restaura a un estado inicial válido? O peor aún, ¿qué sucede si debido a una señal de ruido o cualquier otra razón no prevista el circuito mismo se encuentra en uno de sus estados inválidos? En ese caso es necesario asegurar que el circuito en forma eventual pasará a uno de los estados válidos de modo que pueda reasumir la operación normal. En otra forma, si el circuito secuencial circula entre estados inválidos, no habrá forma de devolverlo a su secuencia intentada de transiciones de estado. Aun cuando puede suponerse que esta condición indeseable no

TABLA 6-11 Tabla de excitación para el Ejemplo 6-1

Estado presente			Entrada	Estado siguiente			Entradas flip-flop						Salida
A	*B*	*C*	*x*	*A*	*B*	*C*	*SA*	*RA*	*SB*	*RB*	*SC*	*RC*	*y*
0	0	1	0	0	0	1	0	*X*	0	*X*	*X*	0	0
0	0	1	1	0	1	0	0	*X*	1	0	0	1	0
0	1	0	0	0	1	1	0	*X*	*X*	0	1	0	0
0	1	0	1	1	0	0	1	0	0	1	0	*X*	0
0	1	1	0	0	0	1	0	*X*	0	1	*X*	0	0
0	1	1	1	1	0	0	1	0	0	1	0	1	0
1	0	0	0	1	0	1	*X*	0	0	*X*	1	0	0
1	0	0	1	1	0	0	*X*	0	0	*X*	0	*X*	1
1	0	1	0	0	0	1	0	1	0	*X*	*X*	0	0
1	0	1	1	1	0	0	*X*	0	0	*X*	0	1	1

se supone que ocurra, un diseñador cuidadoso debe asegurarse de que esta situación no ocurra jamás.

Se estableció previamente que los estados sin usar en un circuito secuencial pueden tratarse como condiciones no importa. Una vez que se diseña el circuito, los m flip-flops en el sistema pueden estar en cualquiera de los 2^m estados posibles. Si alguno de los estados se toma como condiciones no importa, debe investigarse el circuito para determinar el efecto de estos estados sin usar. El siguiente estado de los estados inválidos puede determinarse mediante el análisis del circuito. En cualquier caso, siempre es

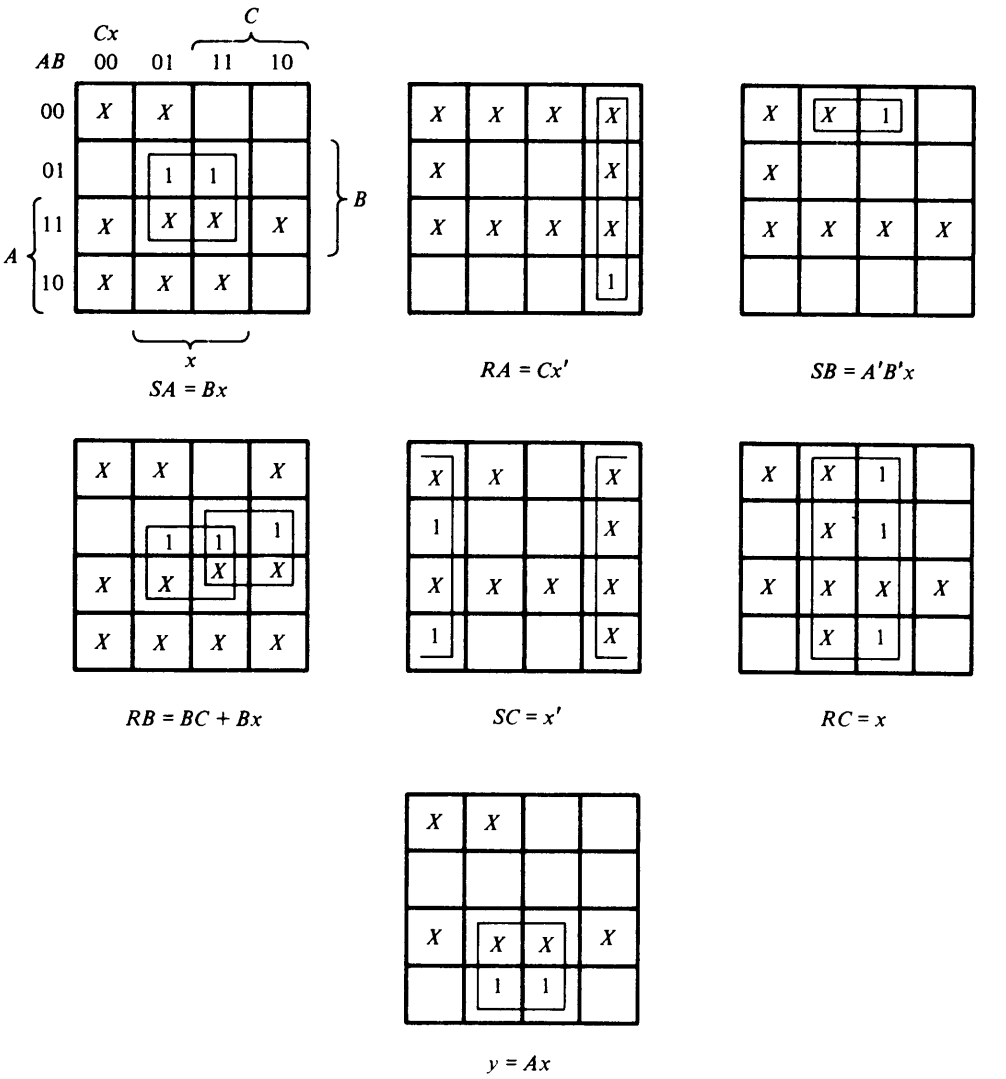


Figura 6-25 Mapas para simplificar el circuito secuencial del Ejemplo 6-1.

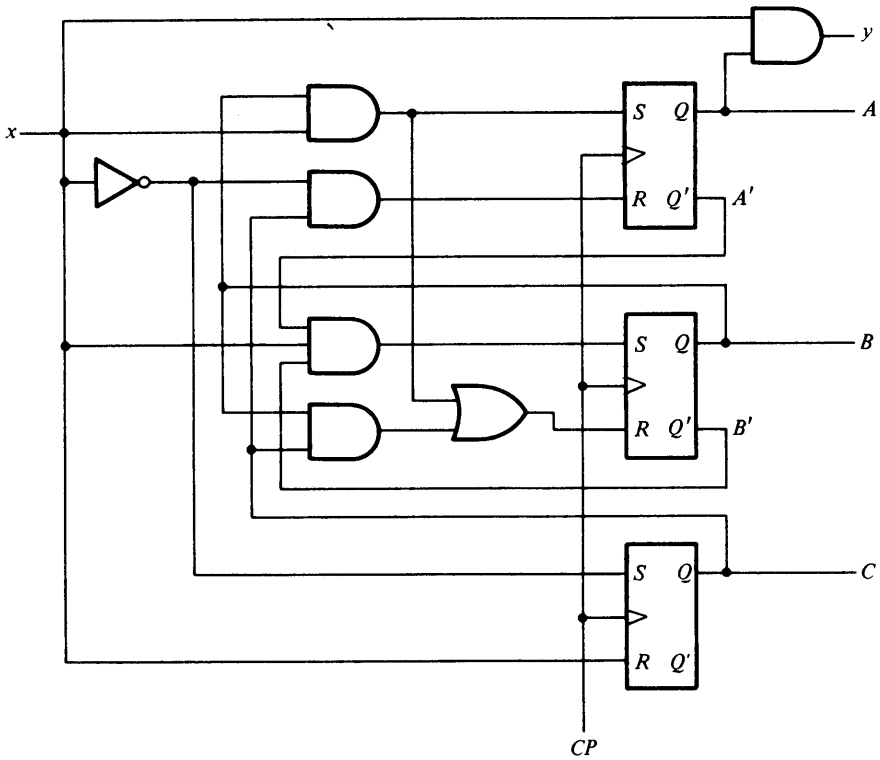


Figura 6-26 Diagrama lógico para el Ejemplo 6-1.

prudente analizar un circuito obtenido mediante un diseño para asegurar que no se incurrió en errores durante el proceso de diseño.

EJEMPLO 6-2: Analice el circuito secuencial que se obtuvo en el Ejemplo 6-1 y determine el efecto de los estados sin usar.

Los estados sin uso son 000, 110 y 111. El análisis del circuito se realiza por el método delineado en la Sección 6-4. Los mapas en la Fig. 6-25 también pueden ayudar en el análisis. Lo que se necesita aquí es principiar con el diagrama del circuito en la Fig. 6-26 y derivar la tabla de estado o el diagrama. Si la tabla de estado es idéntica a la Tabla 6-6 (o a la tabla de estados que es parte de la Tabla 6-11), entonces se sabe que el diseño es correcto. Además, deben determinarse los estados siguientes mediante los estados sin uso 000, 110 y 111.

Los mapas en la Fig. 6-25 pueden ayudar a encontrar el estado siguiente mediante cada uno de los estados sin uso. Tome, por ejemplo, el estado sin uso 000. Si el circuito, por alguna razón, está en el estado presente 000, una entrada $x = 0$ transferirá el circuito a algún estado siguiente y una entrada $x = 1$ lo transferirá a otro (o el mismo) esta-

do siguiente. Se investigará primero el mintérmino $ABCx = 0000$. Mediante los mapas, se ve que este mintérmino no está incluido en alguna función excepto para SC, esto es, la entrada establecida del flip-flop C. Por tanto, los flip-flops A y B no cambiarán pero el flip-flop C se establecerá en 1. Ya que el estado presente es $ABC = 000$, el estado siguiente será $ABC = 001$. Los mapas también muestran que el mintérmino $ABCx = 0001$ está incluido en las funciones para SB y RC. De este modo, B se establecerá y C se despejará. Al principiar con $ABC = 000$ y establecer B, se obtiene el estado siguiente $ABC = 010$ (C está ya despejada). La investigación en el mapa para la salida y muestra que ésta será 0 para estos dos mintérminos.

En el diagrama de estado en la Fig. 6-27 se muestra el resultado del procedimiento de análisis. El circuito opera como debe, mientras permanezca dentro de los estados 001 010, 011, 100 y 101. Aun si se encuentra en uno de los estados inválidos 000, 110 o 111, pasa a uno de los estados válidos dentro de uno o dos pulsos de reloj. Así que, el circuito arranca y se corrige por sí mismo, ya que con el tiempo pasa a un estado válido desde el cual continúa operando como se requiere.

Una situación indeseable puede haber ocurrido si el estado siguiente de 110 para $x = 1$ resulta ser 111 y el siguiente estado de 111 para $x = 0$ o 1 resulta que es 110. Entonces, si el circuito principia desde 110 o 111, circulará y permanecerá entre esos dos estados para siempre. Deben evitarse los estados sin uso que provocan tal comportamiento indeseable; si se encuentra que existen, el circuito debe rediseñarse. Esto puede hacerse con mayor facilidad al especificar un estado siguiente válido para cualquier estado sin uso que se encuentre que circule entre los estados inválidos.

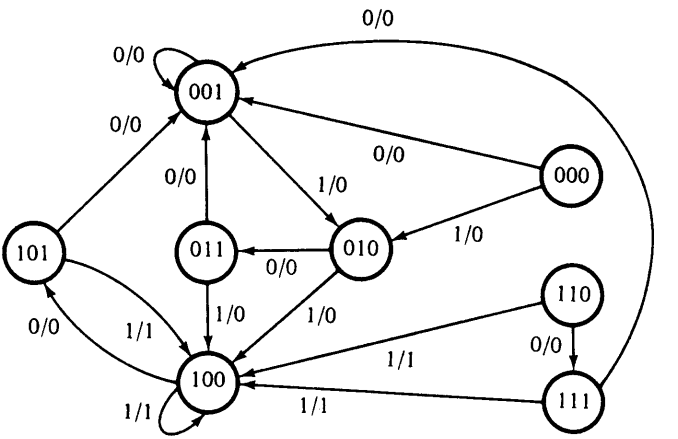


Figura 6-27 Diagrama de estado para el circuito de la Fig. 6-26.

6-8 DISEÑO DE CONTADORES

Un circuito secuencial que pasa a través de una secuencia prescrita de estados bajo la aplicación de pulsos de entrada se denomina *contador*. Los pulsos de entrada, llamados *pulsos de conteo*, pueden ser pulsos de reloj, o pueden originarse en una fuente externa y pueden ocurrir a intervalos de tiempo prescritos o aleatorios. En un contador, la secuencia de estados puede seguir un conteo binario o cualquier otra secuencia de estados. Los contadores se encuentran en casi todo el equipo que contiene lógica digital. Se usa para contar el número de ocurrencias de un evento y son útiles para generar secuencias de temporizado para controlar operaciones en un sistema digital.

De las diversas secuencias que puede seguir un contador, la secuencia binaria directa es la más simple y la más directa. Un contador que sigue la secuencia binaria se denomina *contador binario*. Un contador binario de n-bit consta de n flip-flops y puede contar en binario desde 0 hasta $2^n - 1$. Como ejemplo, el diagrama de estados de un contador de 3-bit se muestra en la Fig. 6-28. Como se ve mediante los estados binarios indicados dentro de los círculos, las salidas flip-flop repiten la secuencia de conteo binario con un retorno a 000 después de 111. Las líneas dirigidas entre los círculos no se marcan con valores de entrada-salida como en los otros diagramas de estado. Recuérdese que las transiciones de estado en los circuitos secuenciales temporizados ocurren durante un pulso de reloj; los flip-flops permanecen en sus estados presentes si no ocurre pulso. Por esta razón la variable de pulso de reloj CP no aparece en forma explícita como una variable de entrada en un diagrama de estado o tabla de estado. Desde este punto de vista, el diagrama de estado de un contador no tiene que mostrar los valores de entrada-salida junto con las líneas dirigidas. La única entrada al circuito es el pulso de conteo, y las salidas están especificadas de manera directa por los estados presentes de los flip-flops. El siguiente estado de un contador depende por completo de su estado presente, y la transición de estado ocurre cada vez que ocurre el pulso. Debido a esta propiedad, un contador se especifica en forma completa por una lista de la *secuencia de conteo*, esto es, la secuencia de los estados binarios por los que pasa.

La secuencia de conteo de un contador binario de 3-bit se da en la Tabla 6-12. El número siguiente en la secuencia representa el estado siguiente que alcanza el circuito bajo la aplicación de un pulso de conteo. La secuencia de conteo se repite después de que alcanza el último valor, de modo que el estado 000 es el estado siguiente después de 111. La secuencia de conteo da toda la información necesaria para diseñar el circuito. No es necesario listar los estados siguientes en una columna separada porque pueden leerse en el número siguiente en la secuencia. El diseño de contadores sigue el mismo procedimiento que se delineó en la Sección 6-7, excepto que la tabla de excitación puede obtenerse de manera directa mediante la secuencia de conteo.

La tabla 6-12 es la tabla de excitación para el contador binario de 3-bit. Los tres flip-flops reciben las designaciones de variables A_2, A_1 y A_0. Los contadores binarios se construyen en la forma más eficiente con flip-flops T (o flip-flop JK con J y K unidas). La excitación flip-flop para las entradas T se derivan mediante la tabla de excitación del flip-flop T y mediante la inspección de la transición de estado desde un conteo dado (estado presente) al siguiente bajo él (estado siguiente). Como ilustración, se consi-

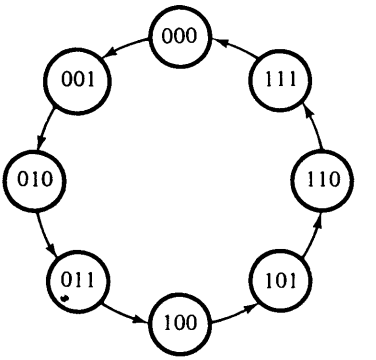


Figura 6-28 Diagrama de estado de un contador binario de 3-bit.

deran las entradas flip-flop para el renglón 001. El estado presente aquí es 001 y el estado siguiente es 010, que es el conteo siguiente en la secuencia. Al comparar estos dos conteos, se observa que A_2 pasa desde 0 a 0; de modo que TA_2 se marca con un 0 porque el flip-flop A_2 debe permanecer sin cambio cuando ocurre el pulso de reloj. A_1 pasa de 0 a 1; de modo que TA_1 se marca con un 1 porque este flip-flop debe complementarse en el siguiente pulso de reloj. En forma similar, A_0 pasa de 1 a 0, indicando que debe complementarse; de modo que TA_0 se marca con un 1. El último renglón con el estado presente 111 se compara con el primer conteo 000 del cual es su estado siguiente. El paso de todos los 1 a todos los 0 requiere que los tres flip-flops se complementen.

Las funciones flip-flop de entrada para las tablas de excitación se simplifican en los mapas de la Fig. 6-29. Las funciones booleanas que se listan bajo cada mapa especifican la parte del circuito combinacional del contador. Al incluir estas funciones con los tres flip-flops, se obtiene el diagrama lógico del contador como se muestra en la Fig. 6-30.

TABLA 6-12 Tabla de excitación para un contador binario de 3-bit

Secuencia de cuenta			Entradas flip-flop		
A_2	A_1	A_0	TA_2	TA_1	TA_0
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	0	0	1
0	1	1	1	1	1
1	0	0	0	0	1
1	0	1	0	1	1
1	1	0	0	0	1
1	1	1	1	1	1

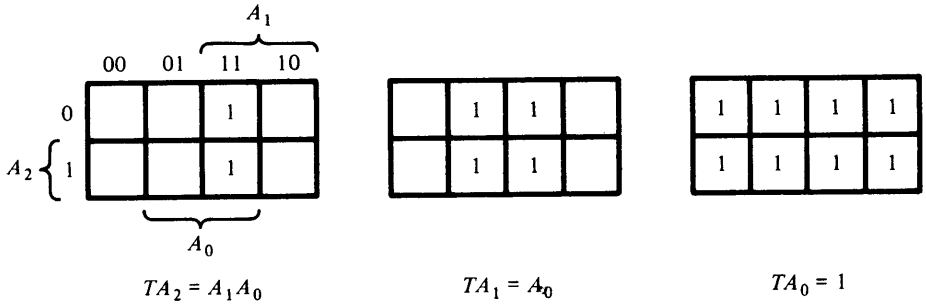


Figura 6-29 Mapas para un contador binario de 3-bit.

Un contador con n flip-flops puede tener una secuencia binaria de menos de 2^n números. Un contador BCD cuenta la secuencia binaria desde 0000 hasta 1001 y regresa a 0000 para repetir la secuencia. Otros contadores pueden seguir una secuencia arbitraria que es posible no sea la secuencia binaria directa. En cualquier caso, el procedimiento de diseño es el mismo. La secuencia de conteo se lista en la tabla de excitación que se obtiene al comparar un conteo presente con el siguiente conteo que se lista bajo él. Una secuencia de conteo tabulada siempre supone una cuenta repetida, de modo que el siguiente estado de la última entrada es el primer conteo listado.

EJEMPLO 6-3: Diseñe un contador que tenga una secuencia repetida de seis estados como se lista en la Tabla 6-13.

En esta secuencia, los flip-flops B y C repiten la cuenta binaria 00, 01, 10, mientras el flip-flop alterna entre 0 y 1 cada tres conteos. La secuencia de conteo para A, B, C no es binaria directa y no se usan los dos estados, 011 y 111. La elección flip-flops JK es resultado de la tabla de excitación en la Tabla 6-13. Las entradas KB y KC tienen sólo 1 y X en sus columnas, de modo que estas entradas siempre son 1. Las otras funciones de entrada flip-flop pueden simplificarse usando los mintérminos 3 y 7 como condiciones no importa. Las funciones simplificadas son:

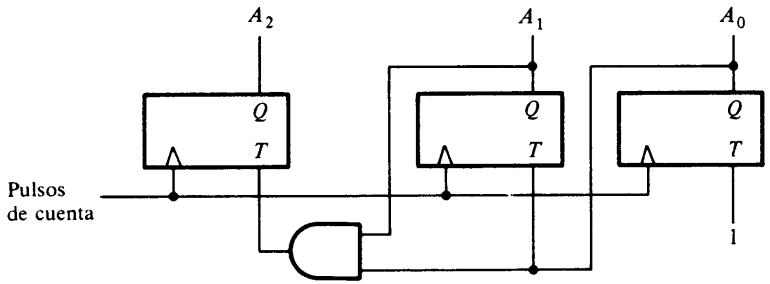


Figura 6-30 Diagrama logico de un contador binario de 3-bit.

$$JA = B \qquad KA = B$$
$$JB = C \qquad KB = 1$$
$$JC = B' \qquad KC = 1$$

El diagrama lógico del contador se muestra en la Fig. 6-31(a). Ya que hay dos estados sin usar, se analiza el circuito para determinar su efecto. El diagrama de estado que se obtiene así se analiza en la Fig. 6-31(b). Si el circuito alguna vez pasa a un estado inválido, el pulso siguiente de conteo lo transfiere a uno de los estados válidos y continúa el conteo en forma correcta. El contador que arranca por sí mismo y puede arrancar desde cualquier estado, pero con el tiempo alcanza la secuencia normal de conteo.

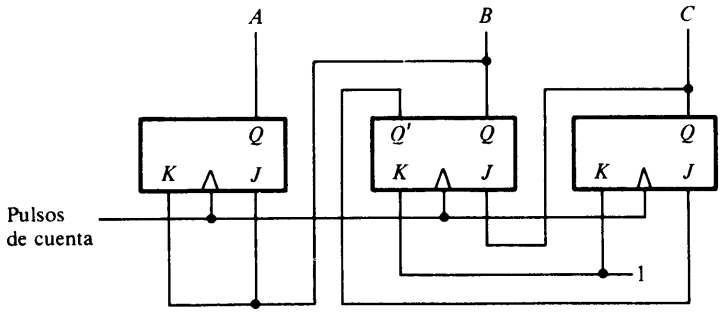
6-9 DISEÑO MEDIANTE LAS ECUACIONES DE ESTADO

Un circuito secuencial puede diseñarse mediante las ecuaciones de estado en lugar de la tabla de excitación. Como se muestra en la Sección 6-4, una ecuación de estado es una expresión algebraica que proporciona las condiciones para el siguiente estado como función del estado presente y las variables de entrada. Las ecuaciones de estado de un circuito secuencial se expresan en forma algebraica la misma información que se expresa en forma tabular en una tabla de estado.

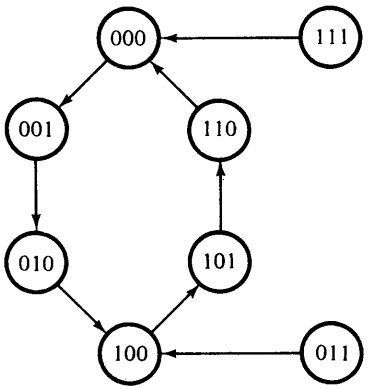
El método con ecuaciones de estado es conveniente cuando el circuito ya está especificado en esta forma o cuando las ecuaciones de estado se derivan con facilidad de la tabla de estado. Este es el método preferido cuando se utilizan flip-flops *D*. Algunas veces puede ser conveniente usar este método con los flip-flops *JK*. La aplicación de este procedimiento a circuitos con flip-flops *RS* o *T* es posible pero implica una cantidad considerable de manipulación algebraica. Aquí se mostrará la aplicación de este método a circuitos secuenciales que emplean flip-flops *D* o *JK*. El punto de inicio en cada caso es la ecuación característica de flip-flop derivada en la Sección 6-2.

TABLA 6-13 Tabla de excitación para el Ejemplo 6-3

Secuencia de cuenta			Entradas flip-flop					
A	*B*	*C*	*JA*	*KA*	*JB*	*KB*	*JC*	*KC*
0	0	0	0	*X*	0	*X*	1	*X*
0	0	1	0	*X*	1	*X*	*X*	1
0	1	0	1	*X*	*X*	1	0	*X*
1	0	0	*X*	0	0	*X*	1	*X*
1	0	1	*X*	0	1	*X*	*X*	1
1	1	0	*X*	1	*X*	1	0	*X*



(a) Diagrama lógico del contador



(b) Diagrama de estado del contador

Figura 6-31 Solución al Ejemplo 6-3.

Circuitos secuenciales con flip-flops *D*

La ecuación característica del flip-flop *D* se deriva en la Fig. 6-5(d):

$$Q(t + 1) = D$$

Esta ecuación establece que el siguiente estado del flip-flop es igual al valor presente de su entrada *D* y es independiente del valor del estado presente. Esto significa que las entradas para el siguiente estado en la tabla de estados son exactamente las mismas que las entradas *D*. Por tanto, no es necesario derivar las condiciones de entrada flip-flop para la tabla de excitación, ya que esta información ya está disponible en las columnas de los estados siguientes.

Se toma, por ejemplo, la tabla de excitación en la Tabla 6-10. La siguiente columna de estado para *A* tiene cuatro números 1, y sucede lo mismo en la columna para el estado siguiente de *B*. Para diseñar este circuito con flip-flops *D*, se escriben las ecuaciones de estado y se igualan a las entradas *D* correspondientes:

$$A(t + 1) = DA(A, B, x) = \Sigma(2, 4, 5, 6)$$
$$B(t + 1) = DB(A, B, x) = \Sigma(1, 3, 5, 6)$$

donde DA y DB son las funciones de entrada flip-flop para los flip-flops D denotados A y B, respectivamente, y cada función se expresa como la suma de cuatro mintérminos. Las funciones simplificadas pueden obtenerse mediante dos mapas de tres variables. Las funciones de entrada flip-flop simplificadas son:

$$DA = AB' + Bx'$$
$$DB = A'x + B'x + ABx'$$

Si hay estados sin usar en el circuito secuencial, deben considerarse junto con las entradas, o combinacionales no importa. Los mintérminos no importa así obtenidos pueden utilizarse para simplificar las ecuaciones de estado de las funciones de entrada flip-flop D.

EJEMPLO 6-4: Diseñe un circuito secuencial con cuatro flip-flops, A, B, C y D. Los siguientes estados de B, C y D son iguales a los estados presentes de A, B y C, respectivamente. El siguiente estado de A es igual a la OR-excluyente de los estados presentes C y D.

Mediante el planteamiento del problema, es conveniente escribir primero las ecuaciones de estado para el circuito:

$$A(t + 1) = C \oplus D$$
$$B(t + 1) = A$$
$$C(t + 1) = B$$
$$D(t + 1) = C$$

Este circuito especifica un *registro de corrimiento con retroalimentación.* En un registro de corrimiento con retroalimentación, cada flip-flop transfiere o corre su contenido al siguiente flip-flop cuando ocurre el pulso de reloj, pero el estado siguiente del primer flip-flop (A en este caso) es cierta función del estado presente de los otros flip-flops. Ya que las ecuaciones de estado son muy simples, el flip-flop de uso más conveniente es el tipo D.

Las funciones de entrada flip-flop para estos circuitos se toman de manera directa mediante las ecuaciones de estado, con la variable del estado siguiente reemplazada por la variable de entrada flip-flop:

$$DA = C \oplus D$$
$$DB = A$$
$$DC = B$$
$$DD = C$$

El circuito puede construirse con cuatro flip-flops D y una compuerta OR-excluyente.

Ecuaciones de estado con flip-flops *JK**

La ecuación característica para el flip-flop *JK* se deriva en la Fig. 6-6(d):

$$Q(t + 1) = (J)Q' + (K')Q$$

Las variables de entrada *J* y *K* están encerradas entre paréntesis para no confundir los términos AND de la ecuación característica con la convención de dos letras, que se ha usado para representar las variables de entrada flip-flop.

El circuito secuencial puede derivarse de manera directa mediante las ecuaciones de estado sin tener que dibujar la tabla de excitación. Esto se hace mediante un procedimiento de comparación entre las ecuaciones de estado para cada flip-flop y la ecuación característica general del flip-flop *JK*. El proceso de comparación consiste en manipular cada ecuación de estado hasta que esté en la forma de la ecuación característica. Una vez que esto se ha hecho, las funciones para las entradas *J* y *K* pueden extraerse y simplificarse. Esto debe hacerse para cada ecuación de estado que se liste, y su nombre de variable flip-flop *A*, *B*, *C*, etc., debe reemplazar la letra *Q* en la ecuación característica.

Una ecuación dada de estado para $Q(t + 1)$ ya puede expresarse como una función de Q y Q'. Con más frecuencia, Q o bien Q', o ambas, pueden estar ausentes en la expresión booleana. Entonces es necesario manipular en forma algebraica la expresión hasta que tanto Q como Q' estén incluidas en la expresión. El siguiente ejemplo demuestra todas las posibilidades que pueden encontrarse.

EJEMPLO 6-5: Diseñe un circuito secuencial con flip-flops *JK* para satisfacer las siguientes ecuaciones de estado:

$$A(t + 1) = A'B'CD + A'B'C + ACD + AC'D'$$
$$B(t + 1) = A'C + CD' + A'BC'$$
$$C(t + 1) = B$$
$$D(t + 1) = D'$$

Las funciones de entrada para el flip-flop *A* se derivan por este método ordenando la ecuación de estado y comparándola con la ecuación característica como sigue:

$$\begin{aligned} A(t + 1) &= (B'CD + B'C)A' + (CD + C'D')A \\ &= (J)A' + (K')A \end{aligned}$$

Mediante la igualdad de las dos ecuaciones, se deduce que las funciones de entrada para el flip-flop *A* son:

$$J = B'CD + B'C = B'C$$
$$K = (CD + C'D')' = CD' + C'D$$

* Esta parte puede omitirse sin pérdida de la continuidad.

La ecuación de estado para el flip-flop B puede ordenarse como sigue:

$$B(t + 1) = (A'C + CD') + (A'C')B$$

No obsante, esta forma no es adecuada para compararla con la ecuación característica ya que falta la variable B'. Si la primera cantidad entre paréntesis se trata con el operador AND con $(B' + B)$, la ecuación permanece igual pero con la variable B' incluida. Por lo tanto:

$$\begin{aligned} B(t + 1) &= (A'C + CD')(B' + B) + (A'C')B \\ &= (A'C + CD')B' + (A'C + CD' + A'C')B \\ &= (J)B' + (K')B \end{aligned}$$

Mediante la igualdad de las dos ecuaciones, se deducen las funciones de entrada para el flip-flop B:

$$\begin{aligned} J &= A'C + CD' \\ K &= (A'C + CD' + A'C')' = AC' + AD \end{aligned}$$

Las ecuaciones de estado para el flip-flop C pueden manipularse como sigue:

$$\begin{aligned} C(t + 1) = B &= B(C' + C) = BC' + BC \\ &= (J)C' + (K')C \end{aligned}$$

Las funciones de entrada para el flip-flop C son:

$$\begin{aligned} J &= B \\ K &= B' \end{aligned}$$

Por último, la ecuación de estado para el flip-flop C puede manipularse con el objeto de comparación como sigue:

$$\begin{aligned} D(t + 1) &= D' = 1.D' + 0.D \\ &= (J)D' + (K')D \end{aligned}$$

lo cual da la función de entrada:

$$J = K = 1$$

Las funciones derivadas de entrada pueden acumularse y listarse juntas. La convención de dos letras para denotar la variable de entrada flip-flop que no se usa en la derivación anterior está a continuación:

$$\begin{aligned} JA &= B'C & KA &= CD' + C'D \\ JB &= A'C + CD' & KB &= AC' + AD \\ JC &= B & KC &= B' \\ JD &= 1 & KD &= 1 \end{aligned}$$

El procedimiento de diseño que aquí se introduce es un método alterno para determinar las funciones de entrada flip-flop de un circuito secuencial cuando se emplean flip-flops *JK*. Para usar este procedimiento cuando inicialmente se especifica un diagrama de estado o una tabla de estado, es necesario que las ecuaciones de estado se deriven por el procedimiento delineado en la Sección 6-4. El método de las ecuaciones de estado para encontrar las funciones de entrada flip-flop puede ampliarse para cubrir los estados sin uso que se consideran como condiciones no importa. Los mintérminos no importa se escriben en la forma de una ecuación de estado y se manipulan hasta que están en la forma de la ecuación característica para el flip-flop particular que se considere. Las funciones J y K en la ecuación de estado no importa se toman entonces como mintérminos no importa cuando se simplifican las funciones de entrada para un flip-flop particular.

BIBLIOGRAFIA

1. Marcus, M. P., *Switching Circuits for Engineers*, 3a. ed. Englewood Cliffs, N.J.: Prentice-Hall, 1975.
2. McCluskey, E. J., *Introduction to the Theory of Switching Circuits*. New York: McGraw-Hill Book Co., 1965.
3. Miller, R. E., *Switching Theory*, dos volúmenes. New York: John Wiley and Sons, 1965.
4. Krieger, M., *Basic Switching Circuit Theory*. New York: The Macmillan Co., 1967.
5. Hill, F. J., and G. R. Peterson, *Introduction to Switching Theory and Logical Design*. 3a. ed. New York: John Wiley and Sons, 1981.
6. Givone, D. D., *Introduction to Switching Circuit Theory*. New York: McGraw-Hill Book Co., 1970.
7. Kohavi, Z., *Switching and Finite Automata Theory*, 2a. ed. New York: McGraw-Hill Book Co., 1978.
8. Phister M., *The Logical Design of Digital Computers*. New York: John Wiley and Sons, 1958.
9. Paull, M. C., and S. H. Unger, "Minimizing the Number of States in Incompletely Specified Sequential Switching Functions." *IRE Trans. on Electronic Computers*, Vol. EC-8, No. 3 (September 1959), 356-66.
10. Hartmanis, J., "On the State Assignment Problem for Sequential Machines I." *IRE Trans. on Electronic Computers*, Vol. EC-10, No. 2 (June 1961), 157-65.

11. McCluskey, E. J., and S. H. Unger, "A Note on the Number of Internal Assignments for Sequential Circuits." *IRE Trans. on Electronic Computer*, Vol. EC-8, No. 4 (December 1959), 439-40.

PROBLEMAS

6-1. Muestre el diagrama lógico de un flip-flop *RS* temporizado con cuatro compuertas NAND.

6-2. Muestre el diagrama lógico de un flip-flop *D* temporizado con compuertas AND y NOR.

6-3. Muestre que el flip-flop *D* temporizado en la Fig. 6-5(a) puede reducirse por una compuerta.

6-4. Considere un flip-flop *JK'*, esto es, un flip-flop *JK* con un inversor entre la salida externa *K'* y la entrada interna *K*.
(a) Obtenga la tabla característica flip-flop.
(b) Obtenga la ecuación característica.
(c) Muestre que al ligar las dos entradas externas se forma un flip-flop *D*.

6-5. Un flip-flop con prioridad de ajuste tiene una entrada de ajuste y de restaurar. Difiere de un flip-flop *RS* convencional en que un intento de ajuste y restaurar en forma simultánea da por resultado el ajuste del flip-flop.
(a) Obtenga la tabla característica y la ecuación característica para el flip-flop con prioridad de ajuste.
(b) Obtenga un diagrama lógico para un flip-flop asíncrono con prioridad de ajuste.

6-6. Obtenga el diagrama lógico de un flip-flop *JK* maestro-esclavo con compuertas AND y NOR. Incluye una previsión para establecer y despejar el flip-flop en forma asíncrona (sin un reloj).

6-7. Este problema investiga la operación del flip-flop *JK* maestro-esclavo a través de la transición binaria en las compuertas internas en la Fig. 6-11. Evalúe los valores binarios (0 o 1) en las salidas de las nueve compuertas cuando las entradas al circuito pasan a través de la secuencia siguiente:
(a) $CP = 0$, $Y = 0$, $Q = 0$ y $J = K = 1$.
(b) Después *CP* pasa a 1 (*Y* debe ir a 1; *Q* permanece en 0).
(c) Después *CP* pasa a 0 e inmediatamente después que *J* pasa a 0 (*Q* debe ir a 1; *Y* no se afecta).
(d) Después *CP* pasa a 1 otra vez (*Y* debe ir a 0).
(e) Después *CP* regresa a 0 e inmediatamente después que *K* pasa a 0 (*Q* debe ir a 0).
(f) Todos los pulsos siguientes no tienen efecto mientras que *J* y *K* permanezcan en 0.

6-8. Dibuje el diagrama lógico (mostrando todas las compuertas) de un flip-flop *D* maestro-esclavo. Use compuertas NAND.

6-9. Conecte una terminal despejada asíncrona a las entradas de las compuertas 2 y 6 del flip-flop en la Fig. 6-12.
(a) Muestre que cuando la entrada despejada es 0, el flip-flop está despejado, y permanece despejado, sin importar los valores de las entradas *CP* y *D*.
(b) Muestre que cuando la entrada despejada es 1, no tiene efecto en las operaciones temporizadas normales.

6-10. El sumador completo en la Fig. P6-10 recibe dos entradas externas x y y; la tercera entrada z proviene de la salida de un flip-flop D. El acarreo de salida se transfiere al flip-flop cada pulso de reloj. La salida externa S da la suma de x, y y z. Obtenga la tabla de estado y el diagrama de estado del circuito secuencial.

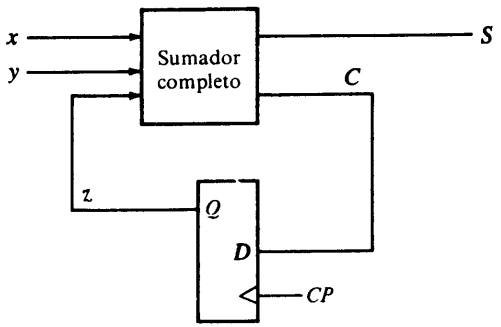


Figura P6-10.

6-11. Derive la tabla de estado y el diagrama de estado del circuito secuencial en la Fig. P6-11. ¿Cuál es la función del circuito?

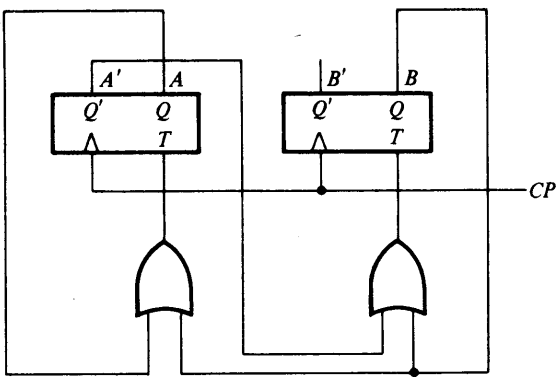


Figura P6-11.

6-12. Un circuito secuencial tiene cuatro flip-flops A, B, C, D y una entrada x. Está descrito por la siguiente ecuación de estado:

$$A(t+1) = (CD' + C'D)x + (CD + C'D')x'$$
$$B(t+1) = A$$
$$C(t+1) = B$$
$$D(t+1) = C$$

(a) Obtenga la secuencia de estados cuando $x = 1$, principiando desde el estado $ABCD = 0001$.

(b) Obtenga la secuencia de estados cuando $x = 0$, principiando desde el estado $ABCD = 0000$.

6-13. Un circuito secuencial tiene dos flip-flops (A y B), dos entradas (x y y), y una salida (z). Las funciones de entrada flip-flop y la función de salida del circuito son las siguientes:

$$JA = xB + y'B' \qquad KA = xy'B'$$
$$JB = xA' \qquad KB = xy' + A$$
$$z = xyA + x'y'B$$

Obtenga el diagrama lógico, la tabla de estado, el diagrama de estado y las ecuaciones de estado.

6-14. Reduzca el número de estados en la siguiente tabla de estados y tabule la tabla de estados reducida.

Estado presente	Estado siguiente		Salida	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	f	b	0	0
b	d	c	0	0
c	f	e	0	0
d	g	a	1	0
e	d	c	0	0
f	f	b	1	1
g	g	h	0	1
h	g	a	1	0

6-15. Principiando desde el estado a de la tabla de estado en el problema 6-14, encuentre la secuencia de salida generada con una secuencia de entrada 01110010011.

6-16. Repita el problema 6-15 utilizando la tabla reducida del problema 6-14. Muestre que se obtiene la misma secuencia de salida.

6-17. Sustituya la asignación binaria 2 de la Tabla 6-5 en los estados en la Tabla 6-4 y obtenga la tabla de estado binaria. Repita esto con la asignación binaria 3.

6-18. Obtenga la tabla de excitación del flip-flop JK' descrita en el problema 6-4.

6-19. Obtenga la tabla de excitación para el flip-flop con prioridad de ajuste que se describe en el problema 6-5.

6-20. Un circuito secuencial tiene una entrada y una salida. El diagrama de estado se muestra en la Fig. P6-20. Diseñe el circuito secuencial con (a) flip-flops T, (b) flip-flops RS y (c) flip-flops JK.

6-21. Diseñe el circuito de un registro de 4-bit que convierte el número binario almacenado en el registro en su valor de complemento 2 cuando la entrada $x = 1$. Los flip-flops del registro son del tipo RST. Este flip-flop tiene tres entradas: dos entradas tienen capacidades RS y uno tiene una capacidad T. Las entradas RS se usan para transferir el número binario de 4-bit cuando una entrada $y = 1$. Use la entrada T para la conversión.

6-22. Repita el Ejemplo 6-1 con la asignación binaria 3 de la Tabla 6-5. Utilice flip-flops JK.

6-23. Diseñe un contador BCD con flip-flops JK.

6-24. Diseñe un contador que cuente los dígitos decimales de acuerdo con el código 2, 4, 2, 1 (Tabla 1-2). Use flip-flops T.

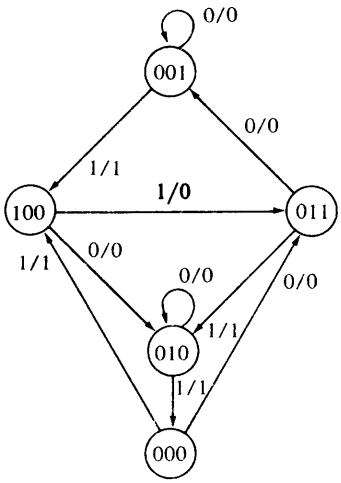


Figura P6-20.

6-25. Diseñe los contadores binarios que tienen la secuencia binaria repetida siguiente. Use flip-flops *JK*.

(a) 0, 1, 2

(b) 0, 1, 2, 3, 4

(c) 0, 1, 2, 3, 4, 5, 6

6-26. Diseñe un contador con la siguiente secuencia binaria: 0, 1, 3, 2, 6, 4, 5, 7 y se repite. Utilice flip-flops *RS*.

6-27. Diseñe un contador con la siguiente secuencia binaria: 0, 1, 3, 7, 6, 4 y se repite. Use flip-flops *T*.

6-28. Diseñe un contador con la siguiente secuencia binaria: 0, 4, 2, 1, 6 y se repite. Utilice flip-flops *JK*.

6-29. Repita el Ejemplo 6-5 usando flip-flops *D*.

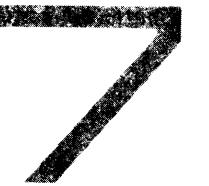
6-30. Verifique el circuito obtenido en el Ejemplo 6-5 usando el método de tabla de excitación.

6-31. Diseñe el circuito secuencial de pulso por las siguientes ecuaciones de estado. Utilice flip-flops *JK*.

$$A(t+1) = xAB + yA'C + xy$$
$$B(t+1) = xAC + y'BC'$$
$$C(t+1) = x'B + yAB'$$

6-32. (a) Derive las ecuaciones de estado para el circuito secuencial especificado por la Tabla 6-6, Sección 6-5. Liste los términos no importa. (b) Derive las funciones de entrada flip-flop mediante las ecuaciones de estado (y los términos no importa) utilizando el método delineado en el Ejemplo 6-5. Use flip-flops *JK*.

Registros, contadores y unidad de memoria



7-1 INTRODUCCION

Un circuito secuencial temporizado consta de un grupo de flip-flops y compuertas combinacionales conectadas para formar una trayectoria de retroalimentación. Los flip-flops son esenciales porque, cuando están ausentes, el circuito se reduce a un circuito combinacional puro (siempre que no haya trayectoria de retroalimentación). Un circuito solo con flip-flops se considera un circuito secuencial incluso cuando están ausentes las compuertas combinacionales.

Un circuito MSI que contiene celdas de almacenamiento en su interior es, por definición, un circuito secuencial. Los circuitos MSI que incluyen flip-flops u otras celdas de almacenamiento por lo común se clasifican según la función que realizan más que por el nombre "circuito secuencial". Estos circuitos MSI se clasifican en una de tres categoría: registros, contadores o memoria de acceso aleatorio. En este capítulo se presentan varios registros y contadores disponibles en la forma IC y se explica su operación. También se presenta la organización de la memoria de acceso aleatorio.

Un *registro* es un grupo de celdas de almacenamiento binario adecuadas para mantener información binaria. Un grupo de flip-flops constituye un registro, ya que cada flip-flop es una celda binaria capaz de almacenar un bit de información. Un registro de n-bit tiene un grupo de n flip-flops y es capaz de almacenar cualquier información binaria que contenga n bits. Además de los flip-flops, un registro puede tener compuertas combinacionales que realicen ciertas tareas de procesamiento de datos. En su definición más amplia, un registro consta de un grupo de flip-flops y compuertas que efectúan su transición. Los flip-flops mantienen información binaria y las compuertas controlan cuando y cómo se transfiere información nueva al registro.

Los contadores están en la Sección 6-8. Un contador es un registro que pasa a través de una secuencia determinada de estados bajo la aplicación de pulsos de entrada. Las compuertas en un contador están conectadas de tal forma que producen una secuencia prescrita de estados binarios en el registro. Aunque los contadores son un tipo especial de registro, es costumbre diferenciarlos dándoles un nombre especial.

Una unidad de memoria es una colección de celdas de almacenamiento junto con circuitos asociados necesarios para transferir la información de entrada y salida

del almacenamiento. Una memoria de acceso aleatorio (RAM) difiere de una memoria de sólo lectura (ROM), en que una RAM puede transferir la información almacenada a una salida (lectura) y también es capaz de recibir información nueva de entrada para almacenamiento (escritura). Un nombre más apropiado para dicha memoria sería *memoria de lectura-escritura.*

Los registros, contadores y memorias tienen amplia aplicación en el diseño de sistemas digitales en general y en computadoras digitales en particular. Los registros también pueden usarse para facilitar el diseño de circuitos secuenciales. Los contadores son útiles para generar variables temporizadoras para las operaciones de secuencia y control en un sistema digital. Las memorias son esenciales para el almacenamiento de programas e información en una computadora digital. El conocimiento de la operación de estos componentes es indispensable para comprender la organización y diseño de los sistemas digitales.

7-2 REGISTROS

Están disponibles varios tipos de registros en la forma de circuitos MSI. El registro más simple posible es el que consta de flip-flops solos sin ninguna compuerta externa. En la Fig. 7-1 se muestra uno de dichos registros construido con cuatro flip-flops tipo D y una entrada común de pulsos de reloj. La entrada de pulsos de reloj, CP, capacita todos los flip-flops, de modo que la información presente disponible en las cuatro entradas puede transferirse al registro de 4-bit. Las cuatro salidas pueden muestrearse para obtener la información almacenada en ese momento en el registro.

La forma en que los flip-flops en un registro se disparan es de importancia primaria. Si los flip-flops están construidos con seguros de compuertas tipo D como en la Fig. 6-5, entonces la información presentada en una entrada de información (D) se transfiere a la salida Q cuando el pulso de habilitación (CP) es 1, y la salida Q sigue los datos de entrada en tanto que la señal CP permanezca 1. Cuando CP pasa a 0, la información que estaba presente en la entrada de información precisamente antes de la transición se retiene en la salida Q. En otras palabras, los flip-flops son sensitivos a la duración del pulso y el registro se habilita mientras $CP = 1$. Un registro que responde a la duración del pulso en forma común se denomina *compuerta con seguro* y la entrada

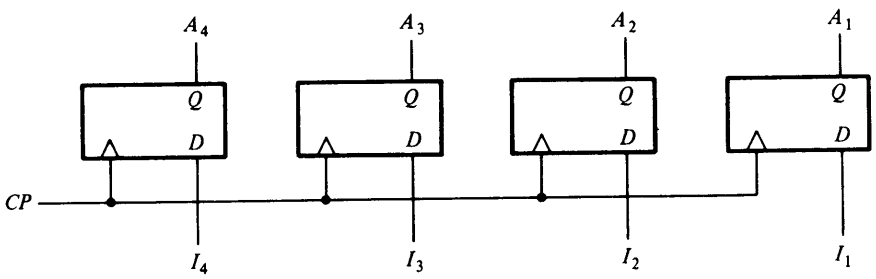


Figura 7-1 Registro de 4-bit.

CP con frecuencia se etiqueta con la variable *G* (en lugar de *CP*). Los seguros son adecuados para usarse como almacenamiento temporal de información binaria que va a transferirse a un destino externo. No deben utilizarse en el diseño de circuitos secuenciales que tienen conexiones de retroalimentación.

Como se explicó en la Sección 6-3, un flip-flop puede usarse en el diseño de circuitos secuenciales temporizados siempre que sea sensitivo a la transición de pulso más que a la duración del pulso. Esto significa que los flip-flops en el registro deben ser del tipo de disparo de borde o maestro-esclavo. En forma normal, no es posible distinguir mediante un diagrama lógico cuando un flip-flop es de seguro con compuerta, disparado por borde, o maestro-esclavo, debido a que los símbolos gráficos para todos los tres son iguales. La distinción debe hacerse en el nombre dado a la unidad. Un grupo de flip-flops sensitivos a la duración del pulso por lo común se denomina como *seguro*, mientras que un grupo de flip-flops sensitivo a una transición de pulso se conoce como *registro.** Un registro siempre puede reemplazar a un seguro, pero la situación inversa debe tomarse con precaución para tener la seguridad de que las salidas de un seguro nunca van a otras entradas flip-flop que se disparan con el mismo pulso de reloj común. En las exposiciones que siguen, se supone que cualquier grupo de flip-flops que se dibuje constituye un *registro* y que todos los flip-flops son de los tipos de disparo por borde o maestro-esclavo. Si el registro es sensitivo a la duración del pulso, se referirá como un *seguro.*

Registro con carga paralela

La transferencia de información nueva a un registro se conoce como *cargar* el registro. Si todos los bits del registro se cargan en forma simultánea con un solo pulso de reloj, se dice que la carga se hace en paralelo. Un pulso aplicado a la entrada *CP* del registro en la Fig. 7-1 cargará las cuatro entradas en paralelo. En esta configuración, el pulso de reloj debe inhibirse desde la terminal *CP* si el contenido del registro debe dejarse sin cambio. En otras palabras, la entrada *CP* actúa como una señal de habilitación que controla la carga de nueva información en el registro. Cuando *CP* pasa a 1, la información de entrada se carga en el registro. Si *CP* permanece en 0, el contenido del registro no se cambia. Obsérvese que el cambio de estado en las salidas ocurre en el borde positivo del pulso. Si un flip-flop cambia de estado al borde negativo, habrá un pequeño círculo bajo el símbolo de triángulo en la entrada *CP* del flip-flop.

La mayoría de los sistemas digitales tienen un reloj generador maestro que suministra un tren continuo de pulsos de reloj. Todos los pulsos de reloj se aplican a los flip-flops y registros en el sistema. El reloj generador maestro actúa como una bomba que suministra un ritmo constante a todas las partes del sistema. Una señal separada de control decide entonces qué pulsos específicos de reloj tendrán un efecto en un registro particular. En dicho sistema, los pulsos de reloj deben combinarse con una compuerta AND con la señal de control, y la salida de la compuerta AND se aplica entonces a la terminal *CP* del registro que se muestra en la Fig. 7-1. Cuando la señal de control es 0, la salida de la compuerta AND es 0 y la información almacenada en el

* Por ejemplo, el IC tipo 7475 es un seguro de 4-bit, en tanto el tipo 74175 es un registro de 4-bit.

registro permanece sin cambio. Sólo cuando la señal de control es un 1 el pulso de reloj pasa a través de la compuerta AND y a la terminal *CP* para cargar información nueva dentro del registro. Tal variable de control se denomina control de entrada de *carga.*

La inserción de una compuerta AND en la trayectoria de pulsos de reloj significa que la lógica se lleva a cabo con pulsos de reloj. La inserción de compuertas lógicas produce retardos de propagación entre el reloj generador maestro y las entradas de reloj de los flip-flops. Para sincronizar el sistema en forma completa, debe tenerse la seguridad de que todos los pulsos de reloj llegan al mismo tiempo a todas las entradas de todos los flip-flops, de modo que todos puedan cambiar en forma simultánea. La realización de la lógica con pulsos de reloj inserta retardos variables y puede lanzar al sistema fuera de sincronismo. Por esta razón, es aconsejable (pero no necesario, en tanto los retardos se toman en consideración) aplicar los pulsos de reloj directamente a todos los flip-flops y controlar la operación del registro con otras entradas, como las entradas *R* y *S* de un flip-flop *RS*.

Un registro de 4-bit con una entrada de control de carga usando flip-flops *RS* se muestra en la Fig. 7-2. La entrada *CP* del registro recibe pulsos sincronizados continuos que se aplican a todos los flip-flops. El inversor en la trayectoria *CP* provoca que todos los flip-flops se disparen por el borde negativo de los pulsos que llegan. El objetivo del inversor es reducir la carga del reloj generador maestro. Esto se debe a que la entrada *CP* está conectada sólo a una compuerta (el inversor) en lugar de las cuatro entradas de compuertas que se requerirían si las conexiones se hicieran directamente en las entradas de reloj flip-flop (marcadas con triángulos pequeños).

La entrada de *despeje* va a una terminal especial en cada flip-flop a través de una compuerta buffer no inversora. Cuando esta terminal pasa a 0, el flip-flop se despeja en forma asíncrona. La entrada de despeje es útil para dejar el registro de todos los 0 antes de su operación temporizada. La entrada de despeje debe mantenerse en 1 durante las operaciones temporizadas normales (véase la Fig. 6-14).

La entrada de *carga* pasa a través de una compuerta buffer (para reducir el cargado) y a través de una serie de compuertas AND a las entradas *R* y *S* de cada flip-flop. Aunque están presentes en forma continua pulsos de reloj, la entrada de carga es la que controla la operación del registro. Las dos compuertas AND y el inversor asociado con cada entrada IC determinan los valores de *R* y *S*. Si la carga de entrada es 0, tanto *R* como *S* son 0, y no ocurre cambio de estado con ningún pulso de reloj. Por consiguiente, la carga de entrada es una variable de control que puede evitar cualquier cambio de información en el registro en tanto que su entrada sea 0. Cuando el control de carga pasa a 1, las entradas I_1 a la I_4 especifican qué información binaria se carga en el registro en el siguiente pulso de reloj. Para cada *I* que es igual a 1, las entradas flip-flop correspondientes son $S = 1$, $R = 0$. Para cada *I* que es igual a 0, las entradas flip-flop correspondientes son $S = 0$, $R = 1$. Por eso, el valor de entrada se transfiere al registro siempre que la carga de entrada es 1. La entrada de despeje es 1 y un pulso de reloj pasa de 1 a 0. Este tipo de transferencia se denomina transferencia *carga-paralela*, ya que todos los bits del registro se cargan en forma simultánea. Si la compuerta buffer asociada con la etrada de carga se cambia a una compuerta inversora, entonces el registro se carga cuando la entrada de carga es 0 y se inhibe cuando la entrada de carga es 1.

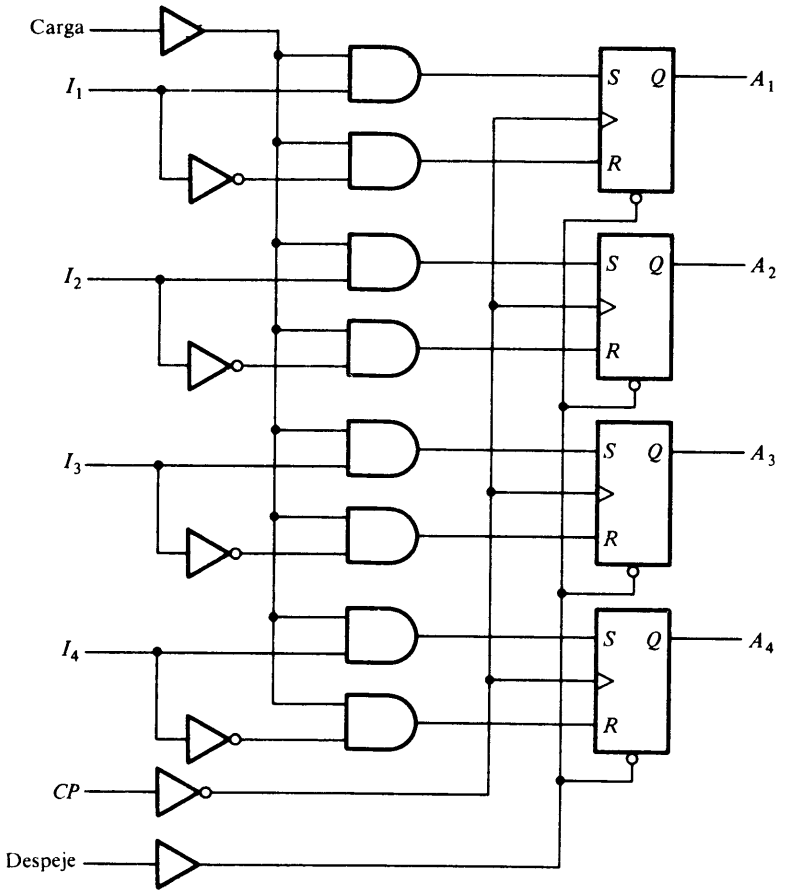


Figura 7-2 Registro de 4-bit con carga paralela.

Un registro con carga paralela puede construirse con flip-flops *D* como se muestra en la Fig. 7-3. Las entradas de reloj y despeje son las mismas que antes. Cuando la entrada de carga es 1, las entradas *I* se transfieren al registro en el siguiente pulso de reloj. Cuando la entrada de carga es 0, las entradas del circuito están inhibidas y los flip-flops *D* están recargados con el estado presente, manteniendo por tanto el contenido del registro. La conexión de retroalimentación en cada flip-flop es necesaria cuando se usa el tipo *D* ya que un flip-flop *D* tiene una condición "sin cambio" de entrada. Con cada pulso de reloj, la entrada *D* determina el estado siguiente de la salida. Para dejar la salida sin cambio, es necesario hacer la entrada *D* igual a la salida *Q* presente en cada flip-flop.

Implementación de lógica secuencial

En el capítulo 6 se vio que un circuito secuencial temporizado consta de un grupo de flip-flops y compuertas combinacionales. Ya que los registros están disponibles como

circuitos MSI, a veces se vuelve conveniente emplear un registro como parte del circuito secuencial. En la Fig. 7-4 se muestra un diagrama de bloques de un circuito secuencial que usa un registro. El estado presente del registro y las entradas externas determinan el estado siguiente del registro y los valores de las salidas externas. Parte del circuito combinacional determina el estado siguiente y la otra parte genera las salidas. El valor del estado siguiente del circuito combinacional se carga en el registro con un pulso de reloj. Si el registro tiene una entrada de carga, debe establecerse en 1. En otra forma, si el registro no tiene entrada de carga (como en la Fig. 7-1), el valor del estado siguiente se transferirá en forma automática cada pulso de reloj.

La parte del circuito combinacional de un circuito secuencial puede implementarse por cualquiera de los métodos expuestos en el Capítulo 5. Puede construirse con

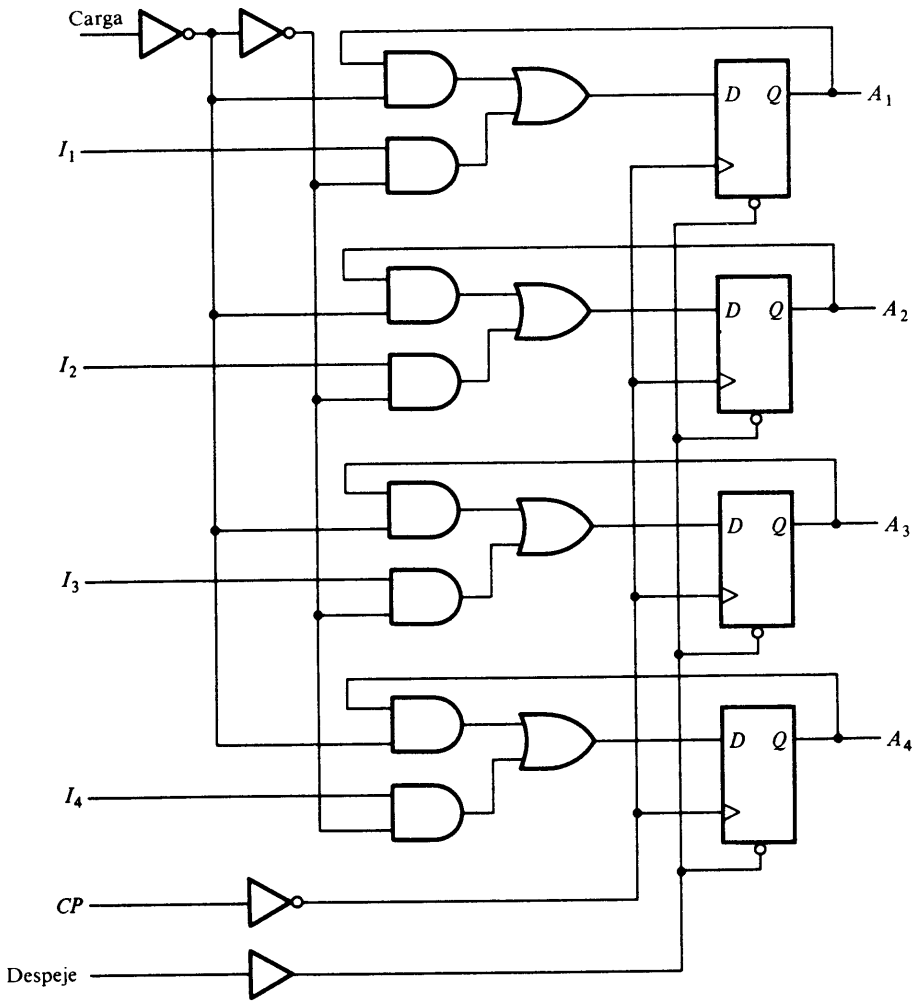


Figura 7-3 Registro con carga paralela que usa flip-flops *D*.

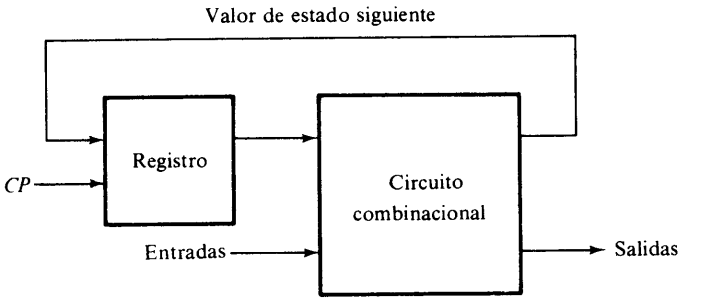


Figura 7-4 Diagrama de bloques de un circuito secuencial.

compuertas SSI, con ROM, o con un arreglo lógico programable (PLA). Mediante el uso de un registro, es posible reducir el diseño de un circuito secuencial al de un circuito combinacional conectado a un registro.

EJEMPLO 7-1: Diseñe el circuito secuencial cuya tabla de estado se lista en la Fig. 7-5(a).

La tabla de estado especifica dos flip-flops A_1 y A_2, una entrada x y una salida y. El siguiente estado y la información de salida se obtienen de manera directa mediante la tabla:

$$A_1(t+1) = \Sigma(4, 6)$$
$$A_2(t+1) = \Sigma(1, 2, 5, 6)$$
$$y(A_1, A_2, x) = \Sigma(3, 7)$$

Los valores mintérminos son para las variables A_1, A_2 y x, las cuales son el estado presente y las variables de entrada. Las funciones para el estado siguiente y salida pueden simplificarse mediante mapas para dar:

$$A_1(t+1) = A_1x'$$
$$A_2(t+1) = A_2 \oplus x$$
$$y = A_2x$$

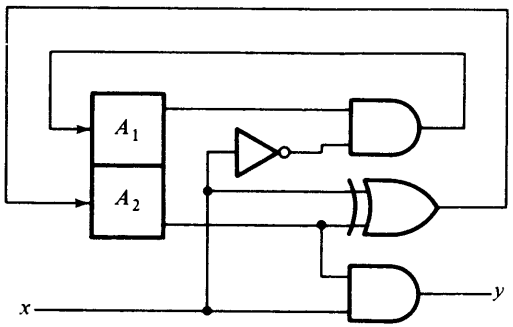
El diagrama lógico se muestra en la Fig. 7-5(b).

EJEMPLO 7-2: Repita el Ejemplo 7-1, pero ahora use una ROM y un registro.

La ROM puede utilizarse para implementar el circuito combinacional y el registro proporcionará los flip-flops. El número de entradas a la ROM es igual al número de flip-flops más el número de las entradas externas. El número de salidas de la ROM es igual al número de flip-flops más el número de las entradas externas. En este caso se tienen tres entradas y tres salidas de la ROM, de modo que su tamaño debe ser 8×3. La implementación se muestra en la Fig. 7-6. La tabla de verdad

Estado presente		Entrada	Estado siguiente		Salida
A_1	A_2	x	A_1	A_2	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	0	1
1	0	0	1	0	0
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	0	0	1

(a) Tabla de estados



(b) Diagrama lógico

Figura 7-5 Ejemplo de la implementación de un circuito secuencial.

ROM es idéntica a la tabla de estado con "estado presente" y "entradas" especificando la dirección de la ROM y "estado siguiente" y "salidas" especificando las salidas ROM. Los valores de estado siguiente deben conectarse desde las salidas ROM a las entradas de registro.

7-3 REGISTROS CON CORRIMIENTO

Un registro capaz de correr su información binaria ya sea a la derecha o a la izquierda se denomina *registro con corrimiento*. La configuración lógica de un registro con corrimiento consta de una cadena de flip-flops conectados en cascada, con la salida de un flip-flop conectada a la entrada del siguiente flip-flop. Todos los flip-flops reciben un pulso común de reloj que causa el corrimiento de una etapa a la siguiente.

El registro con corrimiento más simple posible es uno que usa sólo flip-flops, como se muestra en la Fig. 7-7. La salida Q de un flip-flop dado se conecta a la entrada D del flip-flop a su derecha. Cada pulso de reloj corre el contenido del registrador una posición de bit a la derecha. La *entrada serial* determina qué pasa al flip-flop de la extrema izquierda durante el corrimiento. La *salida serial* se toma de la salida del

Tabla de verdad de una ROM

Dirección			Salidas		
1	2	3	1	2	3
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	0	1
1	0	0	1	0	0
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	0	0	1

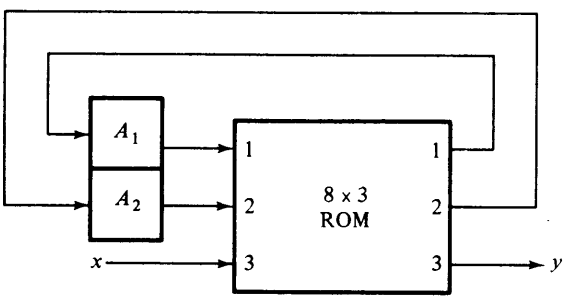


Figura 7-6 Circuito secuencial que usa un registro y una ROM.

flip-flop de la extrema derecha antes de la aplicación de un pulso. Aunque este registro corre su contenido a la derecha, si se gira la página de arriba hacia abajo, se encuentra que el registro corre su contenido a la izquierda. En consecuencia, un registro con corrimiento unidireccional puede funcionar ya sea como un registro de corrimiento a la derecha o como un registro de corrimiento a la izquierda.

El registro en la Fig. 7-7 corre su contenido con cada pulso de reloj durante el borde negativo de la transición del pulso. (Esto se indica por el círculo pequeño asociado con la entrada de reloj en todos los flip-flops.) Si se desea controlar el corrimiento de modo que ocurra sólo con ciertos pulsos pero no con otros, se debe controlar la entrada *CP* del registro. Se mostrará después que las operaciones de corrimiento pueden controlarse a través de las entradas *D* de los flip-flops más bien que a través de la entrada *CP*. Sin embargo, si se usa el registro con corrimiento mostrado en la Fig. 7-7, el corrimiento puede controlarse con facilidad mediante una compuerta externa AND como se muestra a continuación.

Transferencia serial

Se dice que un sistema digital opera en modo serial cuando la información se transfiere y se manipula un bit a la vez. El contenido de un registro se transfiere a otro corriendo los bits de un registro a otro. La información se transfiere un bit a la vez corriendo los bits fuera del registro fuente al registro de destino.

La transferencia serial de información desde el registro *A* al registro *B* se hace con registros con corrimiento, como se muestra en el diagrama de bloque en la Fig. 7-8(a). La salida serial (*SO*) del registro *A* pasa a la entrada serial (*SI*) del registro *B*. Para evitar la pérdida de información almacenada en el registro fuente, el registro *A* se hace que circule su información por la conexión de salida serial a su terminal de entrada serial. El contenido inicial del registro *B* se corre saliendo a través de su salida serial y se pierde a menos que se transfiera a un tercer registro con corrimiento. La entrada de control de corrimiento determina cuándo y por cuántas veces se corren los registros. Esto se hace por la compuerta AND que permite que los pulsos de reloj pasen a las terminales *CP* sólo cuando el control de corrimiento es 1.

Se supone que los registros con corrimiento tienen cuatro bits cada uno. La unidad de control que supervisa la transferencia debe diseñarse de tal forma que capacite los registros con corrimiento, a través de la señal de control de corrimiento, por un tiempo de duración fija igual a cuatro pulsos de reloj. Esto se muestra en el diagrama de temporizado en la Fig. 7-8(b). La señal de control de corrimiento se

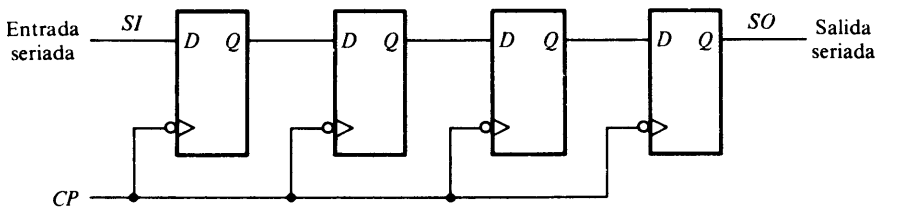
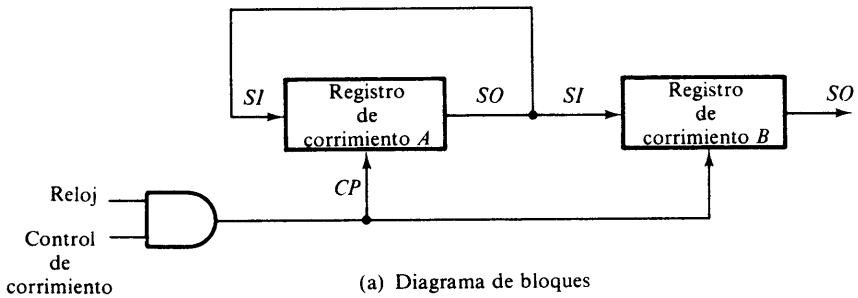
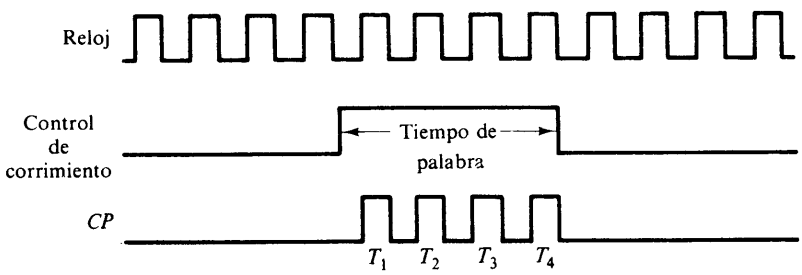


Figura 7-7 Registro de corrimiento.



(a) Diagrama de bloques



(b) Diagrama de temporizado

Figura 7-8 Transferencia seriada del registro A al registro B.

sincroniza con el reloj y cambia de valor precisamente después del borde negativo de un pulso de reloj. Los siguientes cuatro pulsos de reloj encuentran la señal de control de corrimiento en el estado 1, de modo que la salida de compuerta AND conectada a las terminales CP produce los cuatro pulsos T_1, T_2, T_3 y T_4. El cuarto pulso cambia el control de corrimiento a 0 y se inhabilitan los cuatro registros con corrimiento.

Se supone que el contenido binario de A antes del corrimiento es 1011 y el de B 0010. La transferencia serial desde A a B ocurrirá en cuatro pasos como se muestra en la Tabla 7-1. Después del primer pulso T_1, el bit de la extrema derecha de A se corre al bit de la extrema izquierda de B y, al mismo tiempo, este bit se circula a la posición de la extrema izquierda de A. Los otros bits de A y B se corren un lugar a la derecha. La

TABLA 7-1 Ejemplo de transferencia en serie

Pulso temporizador	Registro A de corrimiento				Registro B de corrimiento				Salida en serie de B
Valor inicial	1	0	1	1	0	0	1	0	0
Después de T_1	1	1	0	1	1	0	0	1	1
Después de T_2	1	1	1	0	1	1	0	0	0
Después de T_3	0	1	1	1	0	1	1	0	0
Después de T_4	1	0	1	1	1	0	1	1	1

salida serial previa desde *B* se pierde y su valor cambia de 0 a 1. Los siguientes tres pulsos realizan operaciones idénticas, corriendo los bits de *A* a *B* uno a la vez. Después del cuarto corrimiento, el control de corrimiento pasa a 0 y tanto el registro *A* como el *B* tienen el valor 1011. De este modo, el contenido de *A* se transfiere a *B* en tanto que el contenido a *A* permanece sin cambio.

La diferencia entre los modos de operación serial y paralelo se hace aparente mediante este ejemplo. En el modo paralelo, está disponible la información de todos los bits de un registro y todos los bits pueden transferirse en forma simultánea durante un pulso de reloj. En el modo serial, los registros tienen una sola entrada serial y una sola salida serial. La información se transfiere un bit a la vez, mientras los registros están corriendo en la misma dirección.

Las computadoras pueden operar en un modo serial, un modo paralelo o en una combinación de ambos. Las operaciones seriales son más lentas debido al tiempo que toma transferir la información de entrada y salida de los registros con corrimiento. Sin embargo, las computadoras seriales requieren menos hardware para realizar operaciones, porque un circuito común puede usarse una y otra vez para manipular los bits que salen de los registros con corrimiento en una forma secuencial. El intervalo de tiempo entre los pulsos de reloj se denomina *tiempo de bit*, y el tiempo requerido para correr el contenido completo de un registro con corrimiento se conoce como *tiempo de palabra*. Estas secuencias de tiempo se generan por las secciones de control del sistema. En una computadora en paralelo, las señales de control se habilitan durante un intervalo de pulso de reloj. Las transferencias en los registros son en paralelo y ocurren bajo la aplicación de un solo pulso de reloj. En una computadora serial, las señales de control deben mantenerse por un periodo igual a un tiempo de palabra. El pulso aplicado cada tiempo de bit transfiere el resultado de la operación, uno a la vez, dentro de un registro con corrimiento. La mayoría de las computadoras operan en un modo paralelo ya que es un modo más rápido de operación.

Registro con corrimiento bidireccional con carga paralela

Los registros con corrimiento pueden usarse para convertir datos seriales en datos en paralelo y viceversa. Si se tiene acceso a todas las salidas flip-flop de un registro con corrimiento, entonces la información que se introduce de manera serial por corrimiento puede tomarse en salida en paralelo mediante las salidas de los flip-flops. Si se agrega capacidad de carga en paralelo a un registro con corrimiento, entonces la información que se introduce en paralelo puede tomarse en salida en forma serial corriendo la información almacenada en el registro.

Algunos registros con corrimiento proporcionan las terminales necesarias de entrada y salida para la transferencia en paralelo. También pueden tener capacidades tanto de corrimiento a la derecha como de corrimiento a la izquierda. El registro con corrimiento más general tiene todas las capacidades que se listan más adelante. Otros pueden tener algunas de esas funciones, cuando menos con una operación de corrimiento.

1. Un control de *despeje* para despejar el registro a 0.
2. Una entrada *CP* para los pulsos de reloj que sincronizan todas las operaciones.
3. Un control de corrimiento a la derecha para habilitar la operación de *corrimiento a la derecha* y las líneas de entrada serial en líneas de salida asociadas con el corrimiento a la derecha.
4. Un control de *corrimiento a la izquierda* para habilitar la operación de corrimiento a la izquierda y las líneas de *entrada serial* y las líneas de *salida* asociadas con el corrimiento a la izquierda.
5. Un control de *carga en paralelo* para habilitar una transferencia en paralelo de las n líneas de entrada asociadas con la transferencia en paralelo.
6. n líneas de salida en paralelo.
7. Un estado de control que deja la información sin cambio en el registro aun cuando se apliquen en forma continua pulsos de reloj.

Un registro capaz de correr tanto a la izquierda como a la derecha se denomina *registro de corrimiento bidireccional.* Uno que puede correr en una sola dirección se conoce como *registro de corrimiento unidireccional.* Si el registro tiene tanto capacidades de corrimiento como de carga en paralelo, se llama *registro con corrimiento y carga paralela.*

El diagrama de un registro con corrimiento que tiene todas las capacidades que se listaron con anterioridad se muestra en la Fig. 7-9.* Consta de cuatro flip-flops *D*, aunque pueden usarse flip-flops *RS* siempre que se inserte un inversor entre las terminales *S* y *R*. Los cuatro multiplexores (MUX) son parte del registro y aquí se dibujan en forma de diagrama de bloques. (Véase la Fig. 5-16 para el diagrama lógico del multiplexor.) Los cuatro multiplexores tienen dos variables de selección comunes, s_1 y s_0. La entrada 0 en cada MUX se selecciona cuando $s_1s_0 = 00$, la entrada 1 se selecciona cuando $s_1s_0 = 1$ y en forma similar para las otras dos entradas de los multiplexores.

Las entradas s_1 y s_0 controlan el modo de operación del registro como se especifica en las funciones listadas en la Tabla 7-2. Cuando $s_1s_0 = 00$, el valor presente del registro se aplica a las entradas *D* de los flip-flops. Esta condición forma una trayectoria desde la salida de cada flip-flop hasta la entrada del mismo flip-flop. El siguiente pulso de reloj transfiere en cada flip-flop el valor binario que tenía previamente y no ocurre cambio de estado. Cuando $s_1s_0 = 01$, las terminales 1 de las entradas del multiplexor tienen una trayectoria a las entradas *D* de los flip-flops. Esto causa una operación de corrimiento a la derecha, con la entrada serial transferida al flip-flop A_4. Cuando $s_1s_0 = 10$, resulta una operación de corrimiento a la izquierda, con la otra entrada serial pasando al flip-flop A_1. Por último, cuando $s_1s_0 = 11$, la información binaria en las líneas de entrada en paralelo se transfiere al registro en forma simultánea durante el siguiente pulso de reloj.

* Este registro es similar al IC tipo 74194.

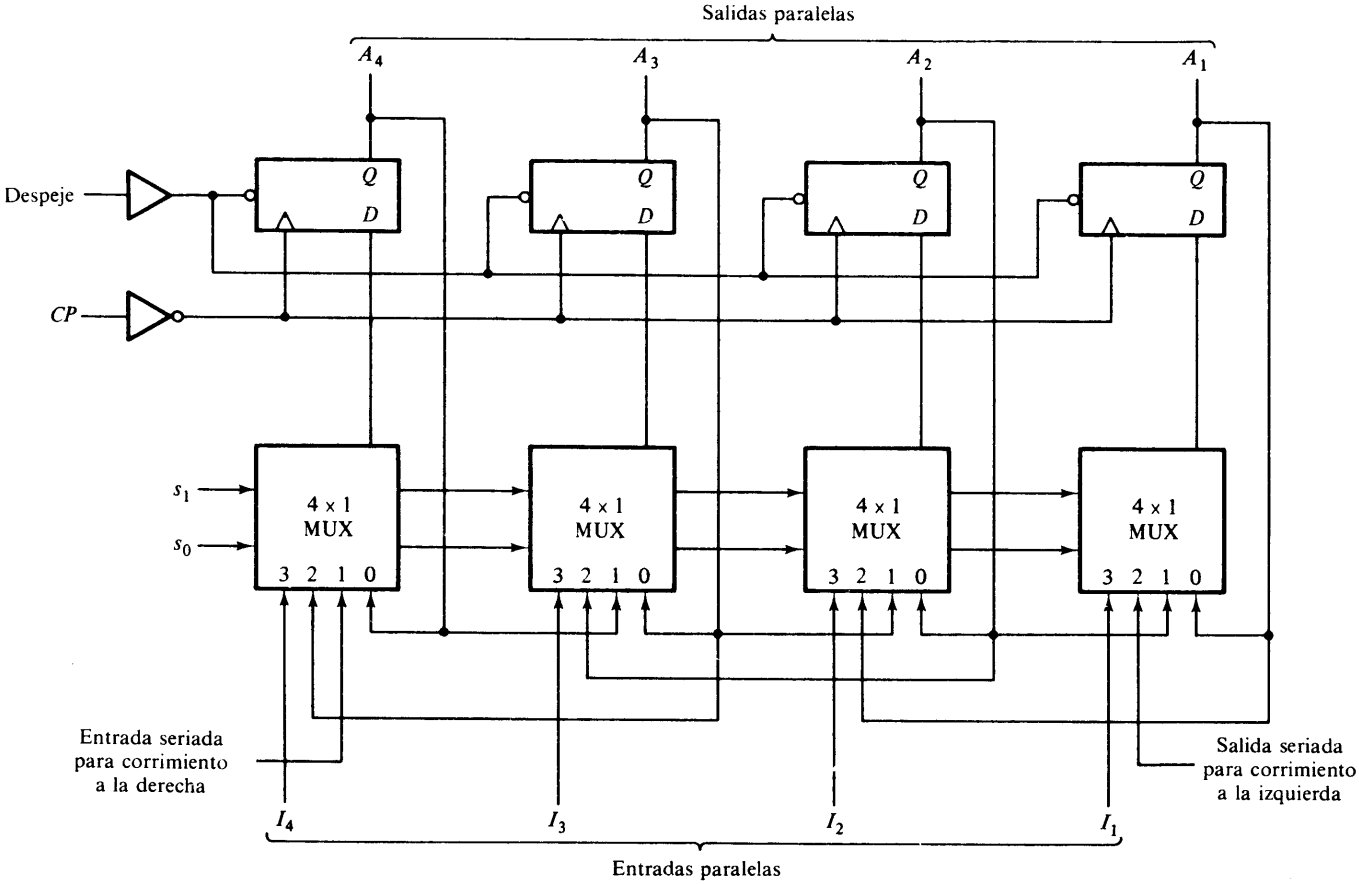


Figura 7-9 Registrador de corrimiento bidireccional de 4-bit con carga paralela.

Un registro con corrimiento bidireccional con carga paralela es un registro de propósito general capaz de realizar tres operaciones: corrimiento a la izquierda, corrimiento a la derecha y carga paralela. No todos los registros con corrimiento disponibles en los circuitos MSI tienen todas estas capacidades. La aplicación particular determina la elección de un registro de corrimiento MSI con preferencia a otro.

Adición serial

Las operaciones en las computadoras digitales se hacen principalmente en paralelo debido a que es el modo de operación más rápido. Las operaciones seriales son lentas pero requieren menos equipo. Para demostrar el modo serial de operación, se presenta aquí el diseño de un adicionador serial. La contraparte en paralelo se expuso en la Sección 5-2.

Los dos números binarios que van a sumarse en forma serial se almacenan en dos registros de corrimiento. Se añaden bits en un par a la vez, en forma secuencial, a través de un circuito sumador completo (*FA*), como se muestra en la Fig. 7-10. El acarreo de salida del sumador completo se transifere a un flip-flop *D*. La salida de este flip-flop se usa entonces como un acarreo de entrada para el siguiente par de bits significativos. Los dos registros de corrimiento se corren a la derecha por el periodo de un tiempo de palabra. Los bits suma de la salida *S* del sumador completo pueden transferirse a un tercer registro con corrimiento. Por el corrimiento de la suma a *A* mientras los bits de *A* se corren saliendo, es posible utilizar un registro para almacenar tanto los bits de adendo como los de sumando. La entrada serial (SI) del registro *B* es capaz de recibir un nuevo número binario mientras los bits sumando se corren hacia afuera durante la adición.

El sumador serial opera como sigue. En forma inicial, el registro *A* retiene el sumando, el registro *B* retiene el adendo y el flip-flop de acarreo, se despeja a 0. Las salidas seriales (*SO*) de *A* y *B* proporcionan un par de bits significativos para el sumador completo *x* y *y*. La salida *Q* del flip-flop de la entrada de acarreo a *z*. El control de corrimiento a la derecha habilita tanto a los registros como al flip-flop de acarreo de modo que al siguiente pulso de reloj, ambos registros se corren un lugar a la derecha, el bit de suma de *S* entra al flip-flop de la extrema izquierda de *A*. Y el acarreo de salida se transfiere al flip-flop *Q*. El control de corrimiento a la derecha habilita los registros por un número de pulsos de reloj igual al número de bits en los registros. Para

TABLA 7-2 Tabla de función para el registro en la Fig. 7-9

Modo de control		Operación del registro
s_1	s_0	
0	0	Sin cambio
0	1	Corrimiento a la derecha
1	0	Corrimiento a la izquierda
1	1	Carga paralela

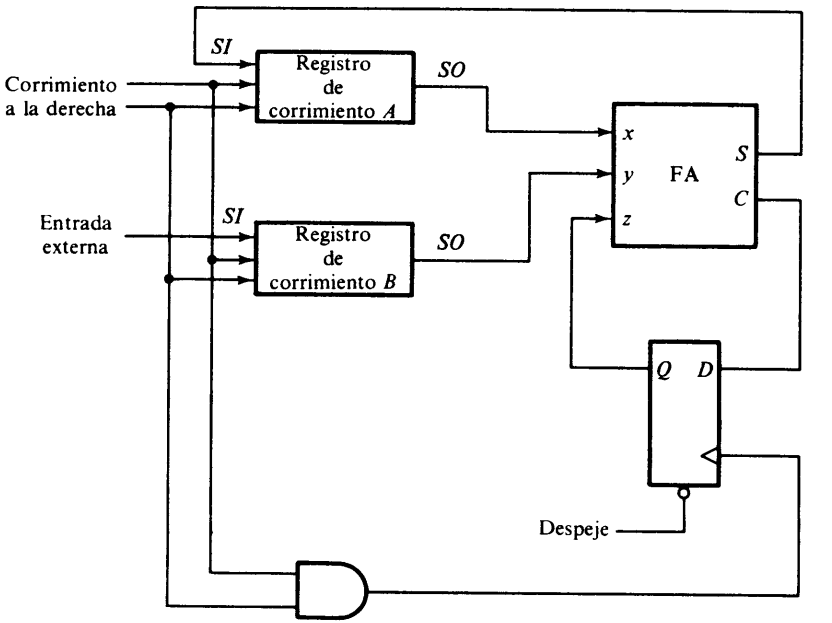


Figura 7-10 Sumador serial.

cada pulso de reloj subsecuente, se transfiere un nuevo bit de suma a A, un acarreo se transfiere a Q y ambos registros se corren un lugar a la derecha. Este proceso continúa hasta que el control de corrimiento a la derecha se habilita. Por tanto, la adición se lleva a cabo pasando cada par de bits junto con la cuenta previa que se lleva a través de un solo circuito sumador completo y se transfiere la suma, un bit a la vez, al registro A.

Si tiene que sumarse un nuevo número a los contenidos del registro A, este número debe transferirse serialmente primero al registro B. La repetición del proceso una vez más agregará el segundo número al número previo en A.

Al comparar el sumador serial con el sumador paralelo que se describe en la Sección 5-2, se observan las siguientes diferencias. El sumador paralelo debe usar registros con capacidad de carga en paralelo, en tanto que el sumador serial utiliza registros de corrimiento. El número de circuitos sumadores completos en el sumador paralelo es igual al número de bits en los números binarios, en tanto que el sumador serial requiere sólo un circuito sumador completo y un flip-flop de acarreo. Excluyendo los registros, el sumador paralelo es un circuito combinacional puro, en tanto que el sumador serial es un circuito secuencial. El circuito secuencial en el sumador serial consta de un circuito sumador completo y un flip-flop que almacena la salida de acarreo. Esto es típico en las operaciones seriales ya que el resultado de una operación de un bit a la vez puede depender no sólo de las entradas presentes, sino también de las entradas previas.

Para mostrar que las operaciones de un bit a la vez en las computadoras seriales pueden requerir un circuito secuencial, se rediseñará el sumador serial considerándolo como un circuito secuencial.

EJEMPLO 7-3: Diseñe un sumador serial usando un procedimiento de lógica secuencial.

Primero, debe estipularse que los dos registros de corrimiento están disponibles para almacenar los números binarios que van a sumarse en forma serial. Las salidas seriales de los registros se designan por las variables x y y. El circuito secuencial que va a diseñarse no incluirá los registros de corrimiento; se insertarán después para mostrar la unidad completa. El propio circuito secuencial tiene dos entradas, x y y, que proporcionan un par de bits significativos, y una salida S que genera el bit de suma, y el flip-flop Q para almacenar el acarreo. El estado presente de Q proporciona el valor presente del acarreo. El pulso de reloj que corre los registros habilita el flip-flop Q para cargar el siguiente acarreo. Este acarreo se utiliza entonces con el siguiente par de bits en x y y. La tabla de estado que especifica el circuito secuencial se da en la Tabla 7-3.

El estado presente de Q es el valor presente del acarreo. El acarreo presente Q se suma junto con las entradas x y y para producir el bit de suma en la salida S. El siguiente estado de Q es equivalente a la salida del acarreo. Recuérdese que los listados de la tabla de estado son idénticos a los listados en la tabla de verdad del sumador completo, excepto que la entrada del acarreo, ahora es el estado presente de Q y el acarreo de salida, ahora es el estado siguiente de Q.

Si se usa un flip-flop D para Q, se obtiene el mismo circuito que en la Fig. 7-10 ya que los requisitos de entrada de la entrada D son los mismos que para los valores de estado siguiente. Si se utiliza un flip-flop JK para Q, se obtienen los requisitos de entrada de excitación que se listan en la Tabla 7-3. Las tres funciones booleanas de interés son las funciones de entrada flip-flop para JK y KQ y salida S. Estas funciones

TABLA 7-3 Tabla de excitación para un adicionador serial

Estado presente	Entradas		Estado siguiente	Salida	Entradas flip-flop	
Q	x	y	Q	S	JQ	KQ
0	0	0	0	0	0	X
0	0	1	0	1	0	X
0	1	0	0	1	0	X
0	1	1	1	0	1	X
1	0	0	0	1	X	1
1	0	1	1	0	X	0
1	1	0	1	0	X	0
1	1	1	1	1	X	0

se especifican en la tabla de excitación y puede simplificarse mediante mapas:

$$JQ = xy$$
$$KQ = x'y' = (x + y)'$$
$$S = x \oplus y \oplus Q$$

Como se muestra en la Fig. 7-11, el circuito consta de tres compuertas y un flip-flop *JK*. Los dos registros de corrimiento también se incluyen en el diagrama para mostrar el sumador completo. Observe que la salida *S* es una función no sólo de *x* y *y*, sino también del estado presente de *Q*. El estado siguiente de *Q* es una función de los valores presentes de *x* y *y* que provienen de las salidas seriales de los registros de corrimiento.

7-4 CONTADORES DE ONDULACION O PULSACION

Los contadores MSI se dividen en dos categorías: contadores de ondulación o pulsación y contadores síncronos. En un contador de ondulación, la transición de salida del flip-flop sirve como una fuente para disparar otro flip-flop. En otras palabras, las entradas *CP* de todos los flip-flops (excepto el primero) se disparan no por los pulsos que llegan, sino más bien por la transición que ocurre en otros flip-flops. En un contador síncrono, los pulsos de entrada se aplican a todas las entradas *CP* de todos los flip-flops. El cambio de estado de un flip-flop particular depende del estado presente en otros flip-flops. Los contadores MSI síncronos se exponen en la sección siguiente. Aquí se presentan algunos contadores de ondulación MSI comunes y se explica su operación.

Contador binario de ondulación

Un contador binario de ondulación consta de una conexión en serie de flip-flops complementarios (tipo *C* o *JK*), con la salida de cada flip-flop conectada a la entrada *CP* del siguiente flip-flop de orden más alto. El flip-flop que retiene el bit menos significativo recibe los pulsos de acarreo que llegan. El diagrama de un contador de ondulación binario de 4-bit se muestra en la Fig. 7-12. Todas las entradas *J* y *K* son iguales a 1. El círculo pequeño en la entrada *CP* indica que el flip-flop se complementa durante una transición que pasa a negativo o cuando la salida a la cual está conectado pasa de 1 a 0. Para entender la operación del contador binario, hay que referirse a su secuencia de conteo dada en la Tabla 7-4. Es obvio que el bit A_1 del orden más bajo debe complementarse con cada pulso de conteo. Cada vez que A_1 pasa de 1 a 0, se complementa a A_2. Cada vez que A_2 pasa de 1 a 0, se complementa a A_3 y así sucesivamente. Por ejemplo, tómese la transición de la cuenta 0111 a 1000. Las flechas en la tabla indican las transiciones en este caso. A_1 se complementa con el pulso de conteo. Ya que A_1 pasa de 1 a 0, dispara A_2 y lo complementa. Como resultado, A_2 pasa de 1 a 0, lo cual a su vez complementa a A_3.A_3 ahora pasa de 1 a 0, lo cual

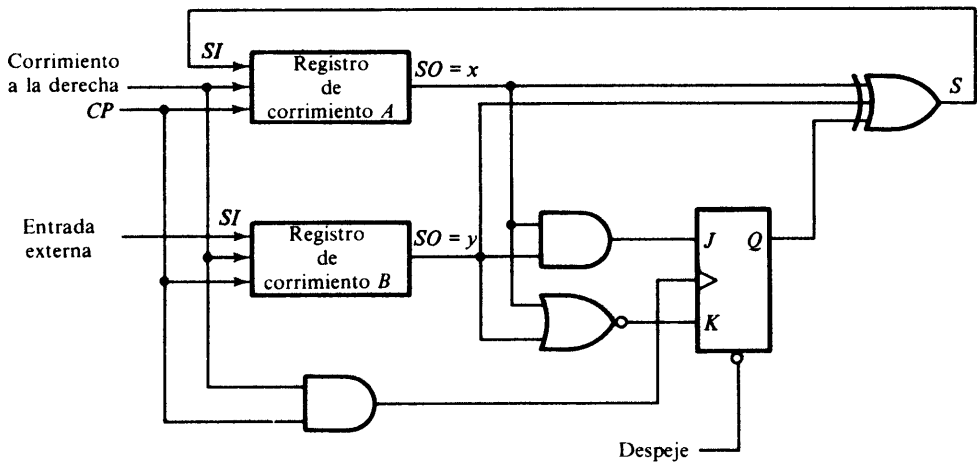


Figura 7-11 Segunda forma de un sumador serial.

complementa a A_4. La transición de salida de A_4, si se conecta a una siguiente etapa, no disparará el flip-flop siguiente ya que pasa de 0 a 1. El flip-flop tiene un cambio a la vez en una sucesión rápida, y la señal se propaga a través del contador en una forma de *ondulación*. Los conductores de ondulación algunas veces se denominan *contadores asíncronos*.

Un contador binario con un conteo en reversa se conoce como contador binario *de decremento*. En un contador de decremento, el conteo binario se disminuye en 1 con cada pulso de entrada de conteo. El conteo de un contador de decremento de 4-bit principia del binario 15 y continúa al conteo binario 14, 13, 12, ..., 0 y entonces regresa a 15. El circuito en la Fig. 7-12 funcionará como un contador de decremento binario si las salidas se toman de las terminales complemento Q' de todos los flip-flops. Si sólo están disponibles las salidas normales de los flip-flops, el circuito debe modificarse ligeramente como se describe a continuación.

Una lista de la secuencia de conteo de un contador binario de decremento muestra que el bit de orden más bajo debe complementarse con cada pulso de conteo. Cualquier otro bit en la secuencia se complementa si su bit previo de orden más bajo pasa de 0 a 1. En consecuencia, el diagrama de un contador binario de decremento es el mismo que en la Fig. 7-12, siempre que todos los flip-flops disparen en el borde

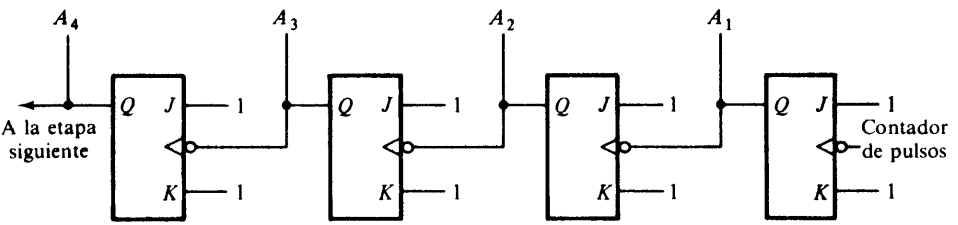


Figura 7-12 Contador binario de ondulación de 4-bit.

TABLA 7-4 Secuencia de cuenta para un contador binario de ondulación

Secuencia de cuenta				Condiciones para complementar flip-flops	
A_4	A_3	A_2	A_1		
0	0	0	0	Complemento A_1	
0	0	0	1	Complemento A_1	A_1 irá de 1 a 0 y complemento A_2
0	0	1	0	Complemento A_1	
0	0	1	1	Complemento A_1	A_1 ira de 1 a 0 y complemento A_2; A_2 irá de 1 a 0 y complemento A_3
0	1	0	0	Complemento A_1	
0	1	0	1	Complemento A_1	A_1 irá de 1 a 0 y complemento A_2
0	1	1	0	Complemento A_1	
0	1	1	1	Complemento A_1	A_1 ira de 1 a 0 y complemento A_2; A_2 ira de 1 a 0 y complemento A_3; A_3 irá de 1 a 0 y complemento A_4
1	0	0	0	y así sucesivamente...	

positivo del pulso. (Los círculos pequeños en las entradas *CP* deben estar ausentes.) Si se usan flip-flops con disparo en borde negativo, entonces la entrada *CP* de cada flip-flop debe conectarse a la salida *Q'* del flip-flop previo. Entonces, cuando *Q* pasa de 0 a 1, *Q'* pasará de 1 a 0 y complementa el siguiente flip-flop como se requiere.

Contador de ondulación BCD

Un contador decimal sigue una secuencia de diez estados y regresa a 0 después del conteo de 9. Dicho contador debe tener cuando menos cuatro flip-flops para representar cada dígito decimal, ya que un dígito decimal se representa por un código binario cuando menos con cuatro bits. La secuencia de estados en un contador decimal está determinada por el código binario que se utiliza para representar un bit decimal. Si se usa el BCD, la secuencia de estados es como se muestra en el diagrama de estados en la Fig. 7-13. Este es similar a un contador binario, excepto que el estado después de 1001 (código para el dígito decimal 9) es 0000 (código para el dígito decimal 0).

El diseño de un contador de ondulación decimal o de cualquier contador de ondulación que no siga la secuencia binaria no es un procedimiento directo. Las herramientas formales del diseño lógico pueden servir sólo como una guía. Un producto final satisfactorio requiere el ingenio y la imaginación del diseñador.

El diagrama lógico de un contador de ondulación BCD se muestra en la Fig. 7-14.* Las cuatro salidas se denotan con el símbolo *Q* con un subíndice numérico igual al peso binario del bit correspondiente en el código BCD. Los flip-flops disparan en el borde negativo, esto es, cuando la señal *CP* pasa de 1 a 0. Obsérvese que la salida de Q_1 se aplica a las entradas *CP* tanto de Q_2 como de Q_8 y la salida de Q_2 se aplica a la

* Este circuito es similar al IC tipo 7490.

entrada CP de Q_4. Las entradas J y K se conectan ya sea a una señal 1 permanente o a las salidas de los flip-flops como se muestra en el diagrama.

Un contador de ondulación es un circuito secuencial asíncrono y no puede describirse por ecuaciones booleanas desarrolladas para describir los circuitos secuenciales temporizados. Las señales que afectan la transición del flip-flop dependen del orden en el cual cambian de 1 a 0. La operación del contador puede explicarse con una lista de condiciones para transiciones de flip-flop. Estas condiciones se derivan mediante el diagrama lógico y mediante el conocimiento de cómo opera un flip-flop JK. Recuérdese que cuando la entrada CP pasa de 1 a 0, el flip-flop se ajusta si $J = 1$, se despeja si $K = 1$, se complementa si $J = K = 1$ y se deja sin cambio si $J = K = 0$. Se requieren las siguientes condiciones para cada transición de estado del flip-flop:

1. Q_1 se complementa en el borde negativo de cada pulso de conteo.
2. Q_2 se complementa si $Q_8 = 0$ y Q_1 pasa de 1 a 0. Q_2 se despeja si $Q_8 = 1$ y Q_1 pasa de 1 a 0.
3. Q_4 se complementa cuando Q_2 pasa de 1 a 0.
4. Q_8 se complementa cuando $Q_4Q_2 = 11$ y Q_1 pasa de 1 a 0. Q_8 se despeja ya sea si Q_4 o Q_2 es 0 y Q_1 pasa de 1 a 0.

Para verificar que estas condiciones resultan en la secuencia requerida por un contador de ondulación BCD, es necesario verificar que todas las transiciones de los flip-flops por supuesto siguen una secuencia de estado, como se especifica por el diagrama de estado en la Fig. 7-13. Otra forma de verificar la operación del contador es derivar el diagrama de tiempos para cada flip-flop para las condiciones listadas con anterioridad. Este diagrama se muestra en la Fig. 7-15 con los estados binarios listados después de cada pulso de reloj. Q_1 cambia de estado después de cada pulso de reloj. Q_2 se complementa cada vez que Q_1 pasa de 1 a 0 en tanto que $Q_8 = 0$. Cuando Q_8 llega a ser 1, Q_2 permanece despejado a 0. Q_4 se complementa cada vez que Q_2 pasa de 1 a 0. Q_8 permanece despejado mientras que Q_2 o Q_4 esté en 0. Cuando tanto Q_2 como Q_4 llegan a ser 1, Q_8 se complementa cuando Q_1 pasa de 1 a 0. Q_8 se despeja en la siguiente transición de Q_1.

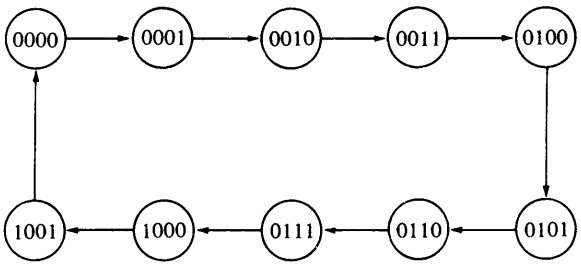


Figura 7-13 Diagrama de estado de un contador BCD decimal.

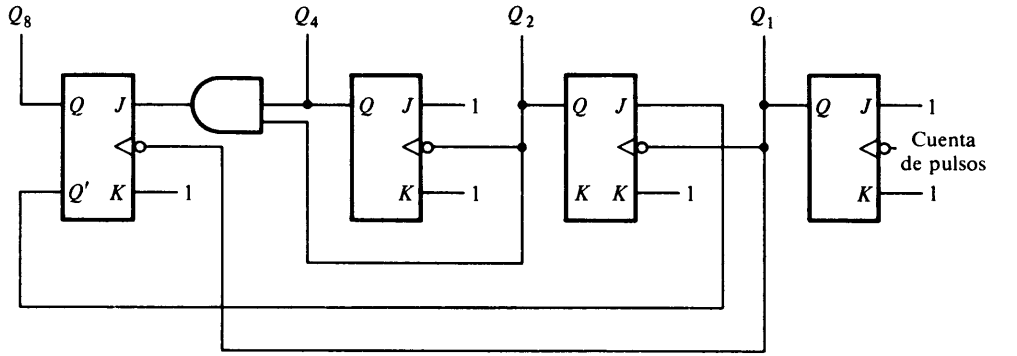


Figura 7-14 Diagrama lógico de un contador BCD de ondulación.

El contador BCD de la Fig. 7-14 es un contador de *década*, ya que cuenta desde 0 a 9. Para contar en decimal desde 0 a 99, se necesita un contador de dos décadas. Para contar desde 0 a 999, se requiere un contador de tres décadas. Los contadores de décadas múltiples pueden construirse conectando contadores BCD en cascada, uno para cada década. Un contador de tres décadas se muestra en la Fig. 7-16. Las entradas a la segunda y tercera década vienen de Q_8 de la década previa. Cuando Q_8 en una década pasa de 1 a 0, dispara la cuenta para la década siguiente de orden más alto, mientras su propia década pasa de 9 a 0. Por ejemplo, el conteo después de 399 será 400.

7-5 CONTADORES SINCRONOS

Los contadores síncronos se distinguen de los contadores de pulsación en que los pulsos de reloj se aplican a las entradas *CP* de *todos* los flip-flops. El pulso común dispara en forma simultánea todos los flip-flops, en lugar de uno a la vez en sucesión como en un contador de pulsación. La decisión de cuándo un flip-flop va a complementarse o no se determina por los valores de las entradas J y K al tiempo del pulso. Si $J = K = 0$, el flip-flop permanece sin cambio. Si $J = K = 1$, el flip-flop se complementa.

Un procedimiento de diseño para cualquier tipo de contador síncrono se presentó en la Sección 6-8. El diseño de un contador binario de 3-bit se llevó a cabo en detalle y se ilustró en la Fig. 6-30. En esta sección, se presentan algunos contadores síncronos MSI típicos y se explica su operación. Debe tomarse en cuenta que no es necesario diseñar un contador si ya está disponible en el comercio en la forma IC.

Contador binario

El diseño de contadores binarios síncronos es tan simple que no es necesario pasar a través de un proceso de diseño riguroso de lógica secuencial. En un contador binario síncrono, el flip-flop en la posición del más bajo orden se complementa con cada pulso. Esto significa que sus entradas J y K deben mantenerse en la lógica 1. Un

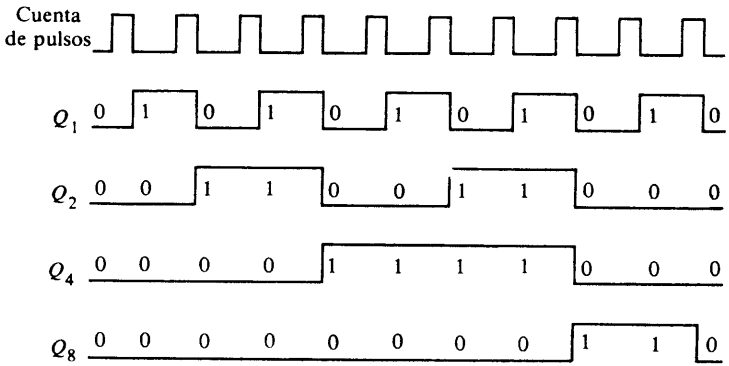


Figura 7-15 Diagrama de temporizado para el contador decimal de la Fig. 7-14.

flip-flop en cualquier otra posición se complementa con un pulso si todos los bits en las posiciones de orden más bajo son iguales a 1, porque los bits de más bajo orden (cuando todos son 1) cambiarán a 0 en el siguiente pulso de conteo. El conteo binario dicta que el siguiente bit de orden más alto se complemente. Por ejemplo, si el estado presente de un contador de 4-bit es $A_4A_3A_2A_1 = 0011$, el siguiente conteo será 0100. A_1 se complementa siempre. A_2 se complementa ya que el estado presente de $A_1 = 1$. A_3 se complementa porque el estado presente de $A_2A_1 = 11$. Pero A_4 no se complementa ya que el estado presente de $A_3A_2A_1 = 011$, lo cual no da la condición de todos 1.

Los contadores síncronos binarios tienen un patrón regular y pueden construirse con facilidad con flip-flops que se complementan y compuertas. El patrón regular puede verse con claridad mediante el contador de 4-bit que se muestra en la Fig. 7-17. Las terminales *CP* de todos los flip-flops se conectan a una fuente común de pulsos de reloj. La primera etapa A_1 tiene sus entradas J y K iguales a 1 si el contador está habilitado. Las otras entradas J y K son iguales a 1 si todos los bits previos de orden bajo son iguales a 1 y el conteo está habilitado. La cadena de compuertas AND genera la lógica requerida para las entradas J y K en cada etapa. El contador puede extenderse a cualquier número de etapas, y cada etapa tiene un flip-flop adicional y una compuerta AND que da una salida de 1 si todas las salidas previas de los flip-flops son 1.

Obsérvese que los flip-flops disparan en el borde negativo del pulso. Esto no es esencial aquí como era en el caso del contador de pulsación. El contador también puede dispararse en el borde positivo del pulso.

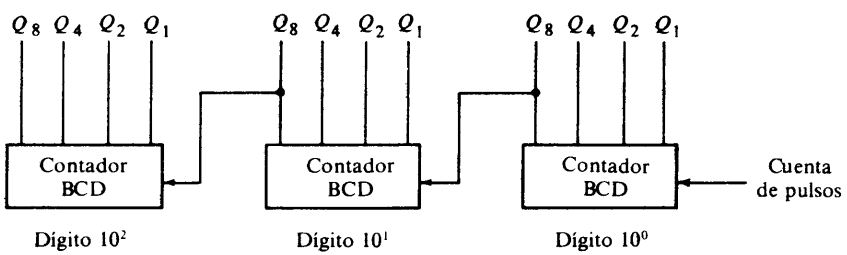


Figura 7-16 Diagrama de bloques de un contador decimal BCD de 3-décadas.

Contador binario de incremento-decremento

En un contador binario síncrono de decremento, el flip-flop en la posición de orden más bajo se complementa con cada pulso. Un flip-flop en cualquier otra posición se complementa con un pulso siempre que todos los bits de orden más bajo sean iguales a 0. Por ejemplo, si el estado presente de un contador binario de decremento de 4-bit es $A_4A_3A_2A_1 = 1100$, el siguiente conteo será 1011. A_1 está siempre complementado. A_2 está complementado ya que el estado presente de $A_1 = 0$. A_3 está complementado porque el estado presente de $A_2A_1 = 00$. Pero A_4 no esta complementado ya que el estado presente de $A_3A_2A_1 = 100$, lo cual no es una condición de todos 0.

Un contador binario de decremento puede construirse como se muestra en la Fig. 7-17, excepto que las entradas a las compuertas AND deben llegar de las salidas complementadas Q' y no de las salidas normales Q de los flip-flops previos. Las dos operaciones pueden combinarse en un circuito. Un contador binario capaz de contar ya sea hacia arriba o hacia abajo se muestra en la Fig. 7-18. Los flip-flops T que se emplean en este circuito pueden considerarse como flip-flops JK con las terminales J y K ligadas. Cuando el control de entrada de incremento es 1, el circuito cuenta hacia arriba, ya que las entradas T están determinadas por los valores previos de las salidas normales en Q. Cuando el control de entrada de decremento es 1, el circuito cuenta hacia *abajo*, ya que las salidas complementarias Q' determinan los estados de las entradas T. Cuando tanto las señales hacia *arriba* como las hacia *abajo* son 0, el registro no cambia de estado y permanece en la misma cuenta.

Contador BCD

Un contador BCD cuenta en código decimal binario desde 0000 a 1001 y de vuelta a 0000. Debido al retorno a 0 después de una cuenta de 9, un contador BCD no tiene un patrón regular como en una cuenta binaria directa. Para derivar el circuito de un contador síncrono BCD, es necesario pasar a través de un procedimiento de diseño como se expuso en la Sección 6-8.

La secuencia de conteo de un contador BCD se da en la Tabla 7-5. La excitación para los flip-flops T se obtiene mediante la secuencia de conteo. En la tabla también se muestra una salida y. Esta salida es igual a 1 cuando el estado presente en el contador es 1001. En esta forma y puede habilitar el conteo de la siguiente década de orden más alto, mientras el mismo pulso cambia la década presente desde 1001 hasta 0000.

Las funciones de entrada flip-flop de la tabla de excitación pueden simplificarse mediante mapas. Los estados sin uso para los mintérminos 10 al 15 se toman como términos no importa. Las funciones simplificadas se listan a continuación:

$$TQ_1 = 1$$
$$TQ_2 = Q_8'Q_1$$
$$TQ_4 = Q_2Q_1$$
$$TQ_8 = Q_8Q_1 + Q_4Q_2Q_1$$
$$y = Q_8Q_1$$

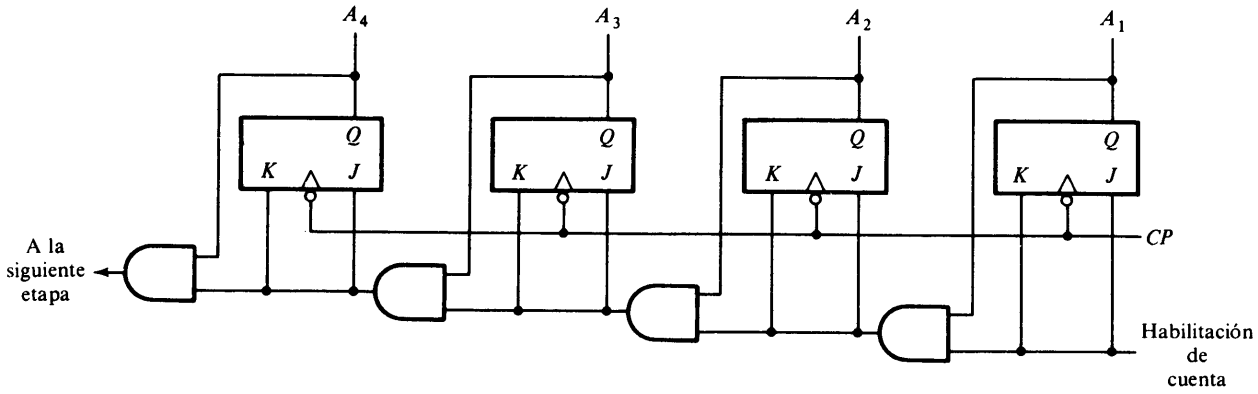


Figura 7-17 Contador síncrono binario de 4-bit.

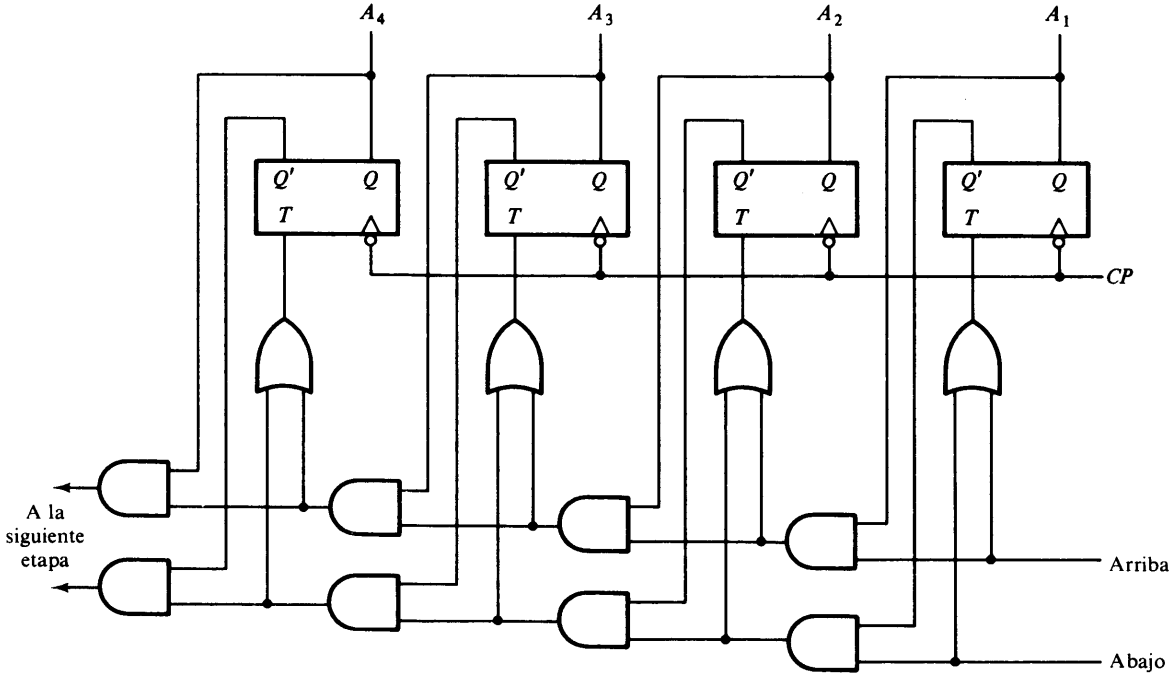


Figura 7-18 Contador binario de incremento decremento de 4-bit.

TABLA 7-5 Tabla de excitación para un contador BCD

Secuencia de cuenta				Entradas flip-flop				Salida de acarreo
Q_8	Q_4	Q_2	Q_1	TQ_8	TQ_4	TQ_2	TQ_1	y
0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	1	1	0
0	0	1	0	0	0	0	1	0
0	0	1	1	0	1	1	1	0
0	1	0	0	0	0	0	1	0
0	1	0	1	0	0	1	1	0
0	1	1	0	0	0	0	1	0
0	1	1	1	1	1	1	1	0
1	0	0	0	0	0	0	1	0
1	0	0	1	1	0	0	1	1

El circuito puede dibujarse fácilmente con cuatro flip-flops T, cinco compuertas AND y una compuerta OR.

Los contadores síncronos BCD pueden configurarse en cascada para formar un contador para números decimales de cualquier longitud. La configuración en cascada se hace como en la Fig. 7-16, excepto que la salida y debe conectarse a la entrada de cuenta de la década siguiente de orden más alto.

Contador binario con carga en paralelo

Los contadores empleados en los sistemas digitales con frecuencia requieren una capacidad de carga en paralelo para transferir un número inicial binario antes de la operación de conteo. En la Fig. 7-19 se muestra en diagrama lógico de un registro que tiene una capacidad de carga en paralelo y también puede operar como un contador.* El control de la carga de entrada, cuando es igual a 1, inhabilita la secuencia de conteo y provoca una transferencia de datos de las entradas I_1 hasta I_4 a los flip-flops A_1 hasta A_4, respectivamente. Si la carga de entrada es 0 y el control de cuenta de entrada es 1, el circuito opera como un contador. Los pulsos de reloj entonces deben provocar que el estado de los flip-flops cambie de acuerdo con la secuencia de conteo binario. Si ambas entradas de control son 0, los pulsos de reloj no cambian el estado del registro.

La terminal de salida de la cuenta que se lleva llega a ser 1 si todos los flip-flops son iguales a 1 mientras se habilita la entrada de conteo. Esta es la condición para complementar el flip-flop que retiene el bit siguiente de orden más alto. Esta salida es útil para aumentar el contador a más de cuatro bits. La velocidad del contador se aumenta si este acarreo se genera directamente de las salidas de todos los cuatro flip-flops en lugar de pasar a través de una cadena de compuertas AND. En forma similar, cada flip-flop está asociado con una compuerta AND que recibe todas las

* Este es similar pero no idéntico al IC tipo 74161.

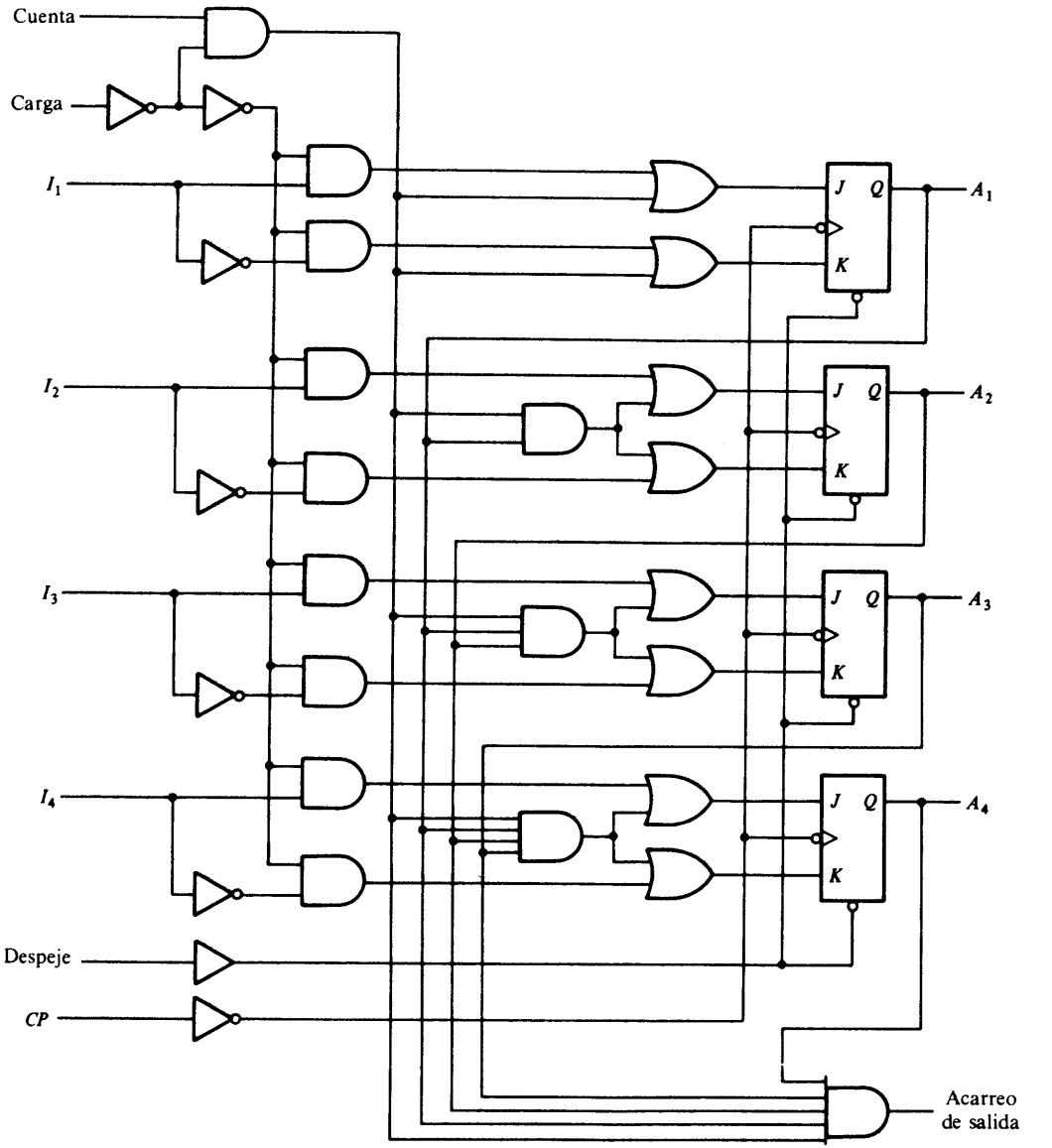


Figura 7-19 Contador binario de 4-bit con carga paralela.

salidas flip-flop previas directamente para determinar cuándo debe complementarse el flip-flop.

La operación del contador se resume en la Tabla 7-6. Las cuatro entradas de control: despeje, *CP*, carga y conteo determinan el siguiente estado de salida. El despeje de entrada es asíncrono y, cuando es igual a 0, causa que el contador se despeje a todos 0, sin importar la presencia de pulsos de reloj u otras entradas. Esto se indica en la tabla con las *X* anotadas, que simbolizan condiciones no importa para las otras entradas, de modo que su valor puede ser 0 o bien 1. La entrada despejada debe pasar al estado 1 para las operaciones temporizadas que se listan en las siguientes tres anotaciones en la tabla. Con las entradas tanto de carga como de cuenta en 0, las salidas no cambian, ya sea que se aplique o no un pulso a la terminal *CP*. Una carga en la entrada de 1 causa una transferencia de las entradas I_1—I_4 al registro durante el borde positivo de un pulso de entrada. La información de entrada se carga en el registro sin importar el valor de la entrada de cuenta, ya que la entrada de cuenta está dividida cuando la entrada de carga es 1. Si la carga de entrada se mantiene en 0, la entrada de conteo controla la operación del contador. Las salidas cambian al siguiente conteo binario en la transición de borde positivo de cada pulso de reloj, pero no ocurre cambio de estado si la entrada de cuenta es 0.

El contador de 4-bit que se muestra en la Fig. 7-19 puede encerrarse en un paquete IC. Son necesarios dos IC para la construcción de un contador de 8-bit; cuatro IC para un contador de 16-bit; y así sucesivamente. El acarreo de salida de un IC debe conectarse a la entrada de cuenta del IC que retiene los bits siguientes de orden más alto del contador.

Los contadores con capacidad de carga paralela que tienen un número especificado de bits son muy útiles en el diseño de sistemas digitales. Se hará referencia después a estos como registros con capacidades de carga y de incremento. La función de *incremento* es una operación que añade 1 al contenido presente de un registro. Al capacitar el control de cuenta durante el periodo de un pulso de reloj, el contenido del registro puede incrementarse en 1.

Un contador con carga paralela puede utilizarse para generar cualquier número deseado de secuencias de conteo. Un contador módulo-*N* (abreviado mod *N*) es un contador que pasa a través de una secuencia repetida de *N* cuentas. Por ejemplo, un contador binario de 4-bit es un contador mod-16. Un contador BCD es un contador mod-10. En algunas aplicaciones, puede ser innecesario preocuparse de los *N* estados particulares que usa un contador mod-*N*. Si este es el caso, entonces puede emplearse

TABLA 7-6 Tabla de función para el contador en la Fig. 7-19

Despeje	*CP*	Carga	Cuenta	Función
0	*X*	*X*	*X*	Despeje a 0
1	*X*	0	0	Sin cambio
1	↑	1	*X*	Entradas de carga
1	↑	0	1	Cuenta en el siguiente estado binario

un contador con carga paralela para construir cualquier contador mod-N, con N correspondiendo a cualquier valor deseado. Esto se muestra en el siguiente ejemplo.

EJEMPLO 7-4: Construya un contador mod-6 usando el circuito MSI especificado en la Fig. 7-19.

En la Fig. 7-20 se muestran cuatro formas en las cuales puede usarse un contador con carga paralela para generar una secuencia de seis conteos. En cada caso el control de cuenta se establece en 1 para habilitar la cuenta a través de pulsos en la entrada *CP*. También se utilizan los hechos de que el control de carga inhibe el conteo y que la operación de despeje es independiente de otras entradas de control.

La compuerta AND en la Fig. 7-20(a) detecta la ocurrencia del estado 0101 en la salida. Cuando el contador se encuentra en ese estado, se habilita la entrada de carga y una entrada de todos a 0 se carga en el registro. Por tanto, el contador pasa a través de los estados binarios 0, 1, 2, 3, 4 y 5 y, entonces, regresa a 0. Este procedimiento produce una secuencia de seis conteos.

La entrada de despeje del registro es asíncrona, esto es, no depende del reloj. En la Fig. 7-20(b), la compuerta NAND detecta la cuenta de 0110, pero tan pronto como ocurre esta cuenta, el registro se despeja. La cuenta 0110 no tiene oportunidad de permanecer por algún tiempo apreciable ya que el registro pasa inmediatamente a 0. Ocurre un pico momentáneo en la salida A_2 conforme la cuenta pasa desde 0101 hasta 0110 e inmediatamente a 0000. Este pico momentáneo puede ser indeseable y por esta razón no se recomienda esta configuración. Si el contador tiene una entrada asíncrona de despeje, es posible despejar el contador con el reloj después de una ocurrencia de la cuenta 0101.

En lugar de usar los primeros seis conteos, puede desearse elegir los últimos seis conteos desde 10 a 15. En este caso es posible tomar ventaja del acarreo de salida, para cargar un número en el registro. En la Fig. 7-20(c), el contador principia con la cuenta 1010 y continúa hasta 1111. El acarreo de salida generado durante el último estado habilita el control de carga, el cual, carga la entrada que se establece en 1010.

También es posible escoger cualquier cuenta intermedia de seis estados. El contador mod-6 en la Fig. 7-20(d) pasa a través de la secuencia de conteo 3, 4, 5, 6, 7 y 8. Ya obtenido el último conteo 1000 la salida A_4 pasa a 1 y se capacita el control de carga. Esto carga en el registro el valor 0011 y la cuenta binaria continúa desde este estado.

7-6 SECUENCIAS DE TEMPORIZADO

La secuencia de las operaciones en un sistema digital se especifica por una unidad de control. La unidad de control que supervisa las operaciones en un sistema digital en forma normal consta de señales de temporizado que determinan la secuencia del

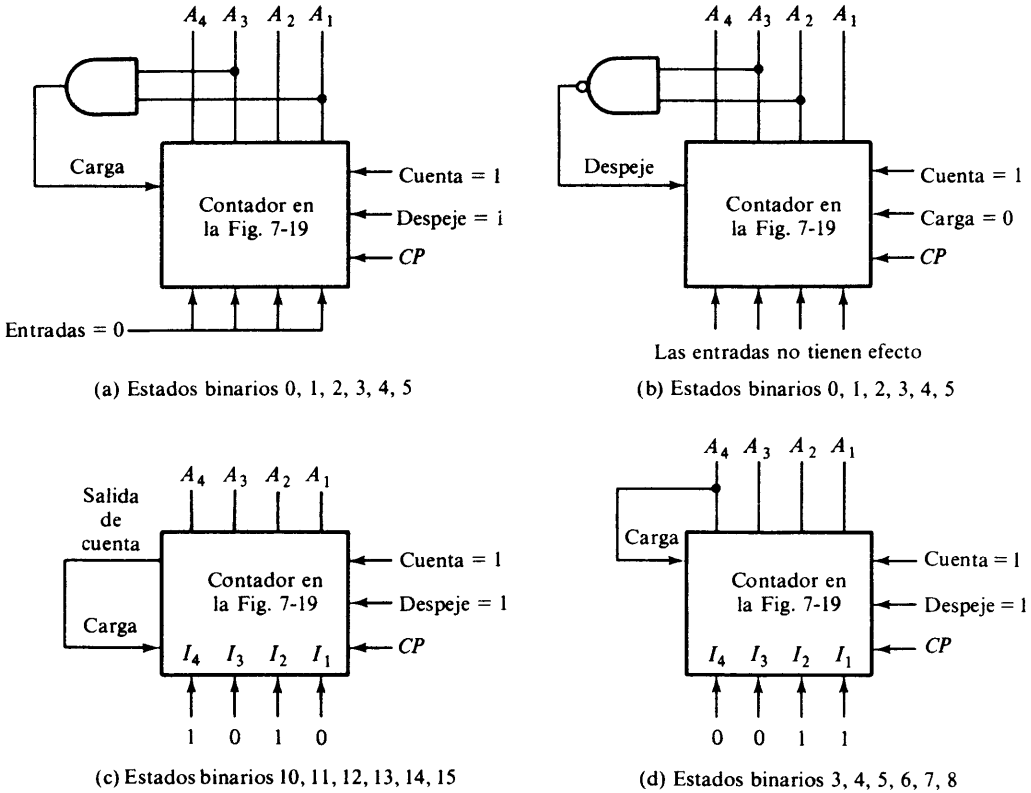


Figura 7-20 Cuatro formas de realizar un contador mod-6 usando un contador con carga paralela.

tiempo en el cual se ejecutan las operaciones. Las secuencias de tiempo en la unidad de control pueden generarse con facilidad mediante contadores o registros de corrimiento. Esta sección demuestra el uso de esas funciones MSI en la generación de señales de temporizado para una unidad de control.

Generación de tiempo de palabra

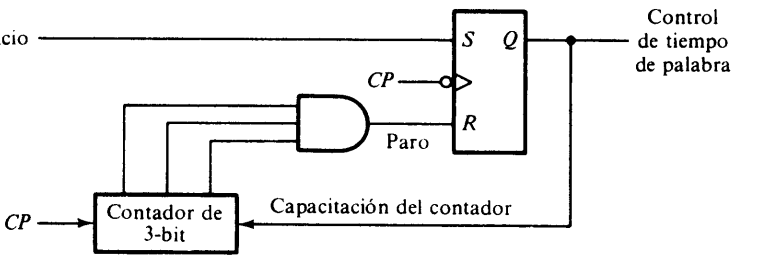
Primero, se demuestra un circuito que genera la señal de temporizado requerida para el modo serial de operación. La transferencia serial de información se expuso en la Sección 7-3, con un ejemplo mostrado en la Fig. 7-8. La unidad de control en una computadora serial debe generar una señal de *tiempo de palabra* que permanezca activa por un número de pulsos igual al número de bits en los registros de corrimiento. La señal de tiempo de palabra puede generarse mediante un contador que cuente el número requerido de pulsos.

Se supone que la señal de tiempo de palabra que va a generarse debe permanecer activa por un periodo de ocho pulsos de reloj. En la Fig. 7-21(a) se muestra un circuito contador que lleva a cabo esta tarea. Inicialmente, el contador de 3-bit se despeja a 0.

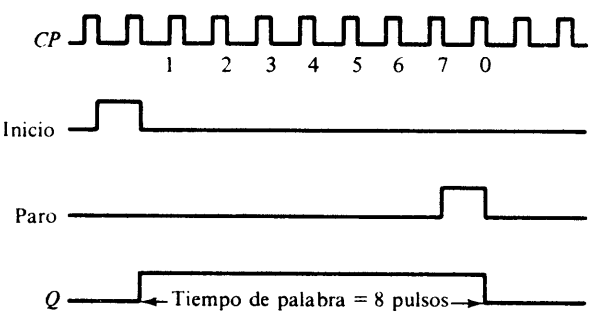
Una señal de arranque ajustará el flip-flop Q. La salida de este flip-flop suministra el control de tiempo de palabra y también habilita el contador. Después de la cuenta de ocho pulsos, el flip-flop se restaura y Q pasa a 0. El diagrama en tiempo de la Fig. 7-21(b) demuestra la operación del circuito. La señal de arranque se sincroniza con el reloj y permanece activa por un periodo de pulso de reloj. Después Q se establece en 1, el contador principia a contar los pulsos de reloj. Cuando el contador alcanza la cuenta de 7 (binario 111), envía una señal de paro a la entrada restaurar del flip-flop. La señal de paro llega a ser un 1 después de la transición de borde negativo del pulso 7. El siguiente pulso de reloj cambia el contador al estado 000 y también depeja Q. Ahora el contador está inhabilitado y la señal de tiempo de palabra permanece en 0. Obsérvese que el contador de tiempo de palabra permanece activo por un periodo de ocho pulsos. Obsérvese también que la señal de paro en este circuito puede usarse para arrancar otro control de cuenta de palabra en otro circuito, precisamente como la señal de arranque se usa en este circuito.

Señales de temporizado

En un modo paralelo de operación, un sólo pulso de reloj puede especificar el tiempo al cual debe ejecutarse una operación. La unidad de control en un sistema digital que opera en el modo paralelo debe generar señales de tiempo que permanecen activas sólo



(a) Diagrama del circuito



(b) Diagrama de temporizado

Figura 7-21 Generación de un control de tiempo de palabra para operaciones seriadas.

por un periodo de pulso de reloj, pero estas señales de tiempo deben diferenciarse unas de otras.

Las señales de temporizado que controlan la secuencia de operaciones en un sistema digital pueden generarse con un registro de corrimiento o un contador con un decodificador. Un *contador de anillo* es un registro de corrimiento circular con sólo un flip-flop que se ajusta en cualquier tiempo particular; todos los otros están despejados. El único bit se corre de un flip-flop a otro para producir la secuencia de señales de temporizado. En la Fig. 7-22(a) se muestra un registro de corrimiento de 4-bit conectado como un contador en anillo. El valor inicial del registro es 1000, lo cual produce la variable T_0. El bit único se corre a la derecha con cada pulso de reloj y circula regresando desde T_3 a T_0. Cada flip-flop está en el estado 1 una vez cada cuatro pulsos de reloj y produce una de las cuatro señales de temporizado que se muestan en la Fig. 7-22(c). Cada salida se vuelve un 1 después de la transición de borde negativo de un pulso de reloj y permanece en 1 durante el siguiente pulso de reloj.

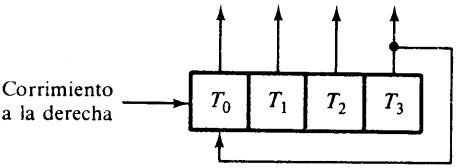
Las señales de temporizado pueden generarse también habilitando en forma continua un contador de 2-bit que pasa a través de cuatro estados distintos. El decodificador que se muestra en la Fig. 7-22(b) decodifica los cuatro estados del contador y genera la secuencia requerida de señales de temporizado.

Las señales de temporizado, cuando se habilita por los pulsos de reloj, proporcionarán pulsos de reloj de fases múltiples. Por ejemplo, si T_0 se opera AND con CP, la salida de la compuerta AND generará pulsos de reloj a un cuarto de la frecuencia de los pulsos del reloj maestro. Los pulsos de reloj de fase múltiple pueden usarse para controlar registros diferentes con escalas de tiempo distintas.

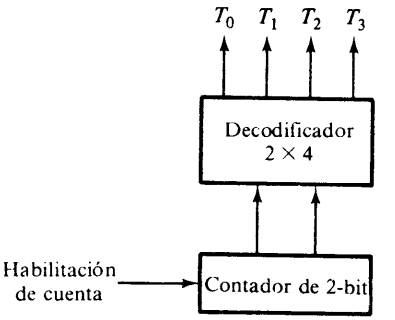
Para generar 2^n señales de temporizado, se necesita ya sea un registro de corrimiento con 2^n flip-flops o un contador de n-bit junto con un decodificador de n-a-2^n líneas. Por ejemplo, pueden generarse 16 señales temporizadoras con un registro de corrimiento de 16-bit conectado con un contador de anillo o con un contador de 4-bit y un decodificador 4-a-16 líneas. En el primer caso, se necesitan 16 flip-flops. En el segundo caso, se necesitan 4 flip-flops y 16 compuertas AND de cuatro entradas para el decodificador. También es posible generar las señales de temporizado con una combinación de un registro de corrimiento y un decodificador. En esta forma, el número de flip-flops es menos que en un contador de anillo, y el decodificador requiere sólo compuertas de dos entradas. Esta combinación algunas veces se denomina *contador Johnson*.

Contador Johnson

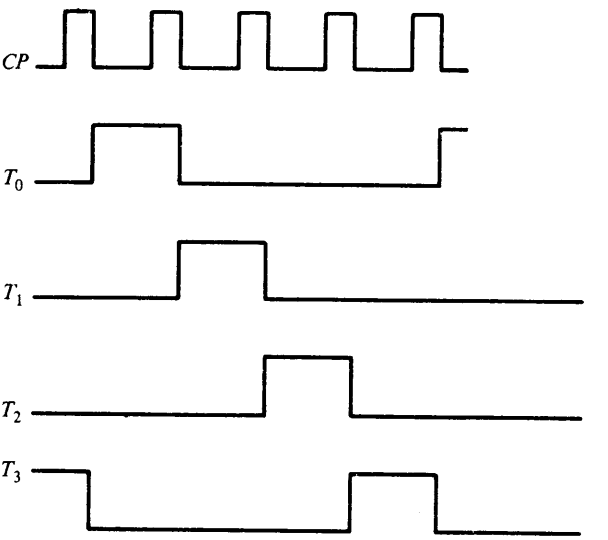
Un contador de anillo de k-bit circula un bit único entre los flip-flops para proporcionar k estados distintos. El número de estados puede duplicarse si el registro de corrimiento se conecta como un contador de anillo con *cambio en cola*. Un contador de anillo con cambio en cola es un registro de corrimiento circular con la salida complementaria del último flip-flop conectada a la entrada del primer flip-flop. En la Fig. 7-23(a) se muestra dicho registro de corrimiento. La conexión circular se hace desde la salida complementaria del flip-flop de la extrema derecha a la entrada del flip-flop de la extrema izquierda. El registro corre su contenido una posición a la derecha con cada



(a) Contador en anillo (valor inicial = 1000)



(b) Contador y decodificador



(c) Secuencia de cuatro señales de temporizado

Figura 7-22 Generador de señales de temporizado.

pulso de reloj, y al mismo tiempo, el valor complementario del flip-flop E se transfiere al flip-flop A. Principiando desde un estado despejado, el contador de anillo de cambio en cola pasa a través de una secuencia de ocho estados como se lista en la Fig. 7-23(b). En general un contador de anillo de cambio en cola de k-bit pasará a través de una secuencia de $2k$ estados. Principiando desde todos 0, cada operación de corrimiento inserta 1 desde la izquierda hasta que el registro está lleno con todos los 1. En las siguientes secuencias, se insertan los 0 desde la izquierda hasta que el registro se llena otra vez con todos los 0.

Un contador Johnson es un contador de anillo que cambia en cola de k-bit con $2k$ compuertas decodificadoras para proporcionar salidas para las $2k$ señales de temporizado. Las compuertas decodificadoras se muestran en la Fig. 7-23, pero están especificadas en la última columna de la tabla. Las ocho compuertas AND listadas en la tabla, cuando se conectan al circuito, complementarán la construcción del contador Johnson. Ya que cada compuerta se habilita durante una secuencia de estado particular, las salidas de las compuertas generarán ocho secuencias de temporizado en sucesión.

La decodificación de un contador de anillo de cambio en cola de k-bit para obtener $2k$ secuencias de temporizado sigue un patrón regular. El estado de todos 0 se decodifica tomando el complemento de las salidas de los dos flip-flops extremos. El estado de todos 1 se decodifica tomando las salidas normales de los dos flip-flops extremos. Todos los demás estados se decodifican de un patrón adyacente 1, 0 o 0, 1 en la secuencia. Por ejemplo, la secuencia 7 tiene un patrón adyacente 0, 1 en los flip-flops B y C. La salida decodificada se obtiene entonces tomando el complemento de B y salida normal de C, o $B'C$.

Una desventaja del circuito que se muestra en la Fig. 7-23(a) es que, si se encuentra en un estado sin uso por sí mismo, persistirá en moverse de un estado inválido a otro y nunca encontrará su camino a un estado válido. Esta dificultad puede corregirse modificando el circuito para evitar esta condición indeseable. Un procedimiento de corrección es desconectar la salida del flip-flop B que va a la entrada D del flip-flop C y habilitar entonces la entrada del flip-flop C por la función:*

$$DC = (A + C)B$$

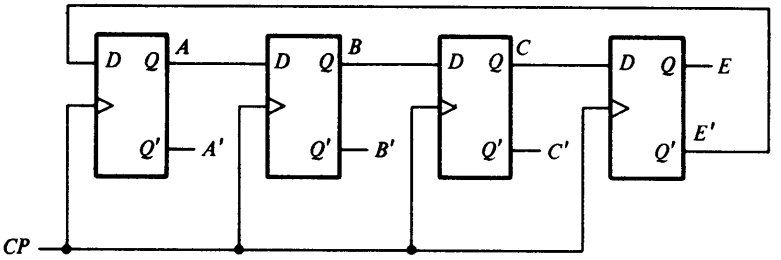
en donde DC es la función de entrada del flip-flop para la entrada D del flip-flop C.

Los contadores Johnson pueden construirse para cualquier número de secuencias de temporizado. El número de flip-flops necesarios es la mitad del número de las señales de temporizado. El número de compuertas decodificadoras es igual al número de señales de temporizado y sólo se emplean compuertas de dos entradas.

7-7 UNIDAD DE MEMORIA

Los registros en una computadora digital pueden clasificarse ya sea en el tipo operacional o de almacenamiento. Un registro *operacional* es capaz de almacenar información binaria en sus flip-flops y, además, tiene compuertas combinacionales capaces de

* Esta es la forma en que está hecho el IC tipo 4022.



(a) Contador en anillo de cola-cambio de 4 etapas

Número de secuencia	Salidas flip-flop *A*	*B*	*C*	*E*	Compuerta AND requerida para la salida
1	0	0	0	0	*A'E'*
2	1	0	0	0	*AB'*
3	1	1	0	0	*BC'*
4	1	1	1	0	*CE'*
5	1	1	1	1	*AE*
6	0	1	1	1	*A'B*
7	0	0	1	1	*B'C*
8	0	0	0	1	*C'E*

(b) Secuencia de cuenta y decodificación requerida

Figura 7-23 Construcción de un contador Johnson.

realizar las tareas de procesamiento de datos. Un registro de *almacenamiento* se utiliza sólo para almacenamiento temporal de la información binaria. Esta información no puede alterarse cuando se transfiere dentro y fuera del registro. Una *unidad de memoria* es una colección de registros de almacenamiento junto con los circuitos asociados necesarios para transferir la información dentro y fuera de los registros. Los registros de almacenamiento en una unidad de memoria se denominan *registros de memoria*.

La mayor parte de los registros en una computadora digital son registros de memoria, a los cuales se transfiere para almacenamiento la información y de los cuales está disponible la información cuando se necesita para procesarla. En la unidad de proceso, en forma comparativa, se encuentran pocos registros operacionales. Cuando tiene lugar el procesamiento de datos, la información de registros seleccionados en la unidad de memoria se transfiere primero a los registros operacionales en la unidad de procesamiento. Los resultados intermedios y final obtenidos en los registros operacionales se transfieren devolviéndolos a registros de memoria seleccionados. En forma similar, la información binaria recibida de los dispositivos de entrada se almacena primero en los registros de memoria; la información transferida a los dispositivos de salida se toma de los registros en la unidad de memoria.

El componente que forma las celdas binarias de los registros en una unidad de memoria debe tener ciertas propiedades básicas; las más importantes de estas son: (1) Debe tener una propiedad confiable de dos estados para la representación binaria. (2) Debe tener tamaño pequeño. (3) El costo por bit de almacenamiento debe ser tan

bajo como sea posible. (4) El tiempo de acceso a un registro de memoria debe ser razonablemente rápido. Los ejemplos de componentes de unidades de memoria son núcleos magnéticos, IC semiconductores, y superficies magnéticas en cintas, tambores o discos.

Una unidad de memoria almacena información binaria en grupos llamados *palabras* y cada palabra se almacena en un registro de memoria. Una palabra en memoria es una entidad de n bits que se mueven hacia adentro y hacia afuera del almacén como una unidad. Una palabra de memoria puede representar un operando, una instrucción, un grupo de caracteres alfanuméricos, o cualquier información codificada en binario. La comunicación entre una unidad de memoria y su medio ambiente se logra a través de dos señales de control y dos registros externos. Las señales de control especifican la dirección de la transferencia requerida, esto es, ya sea que una palabra se almacene en un registro de memoria o que una palabra previamente almacenada se transfiere fuera del registro de memoria. Un registro externo especifica el registro particular de memoria escogido entre los miles disponibles; el otro especifica la configuración particular de bits de la palabra en cuestión. Las señales de control y los registros se muestran en el diagrama de bloques en la Fig. 7-24.

El *registro de dirección* de memoria especifica la memoria de palabras seleccionada. Cada palabra en una memoria está asignada a un número de especificación que principia desde 0 hasta el número máximo de palabras disponibles. Para comunicarse con una palabra de memoria específica, su número de localización, o *dirección*, se transfiere al registro de dirección. Los circuitos internos de la unidad de memoria aceptan esta dirección del registro y abren las trayectorias necesarias para seleccionar

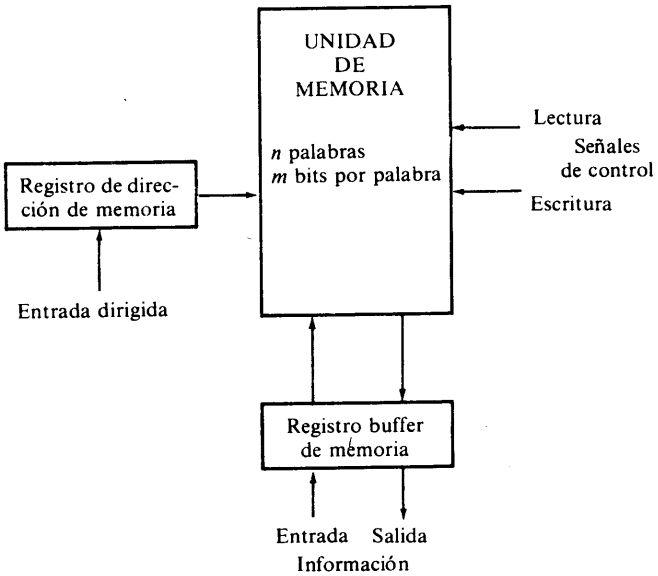


Figura 7-24 Diagrama de bloques de una memoria mostrando la comunicación con el medio ambiente.

la palabra llamada. Un registro de dirección con n bits puede especificar hasta 2^n palabras de memoria. Las unidades de memoria de computadora pueden variar desde 1 024 palabras, que requieren un registro de dirección de 10 bits, a 1 048 576 = 2^{20} palabras, que requiere un registro de dirección de 20-bit.

Las dos señales de control aplicadas a la unidad de memoria se denominan *lectura* y *escritura*. Una señal de escritura especifica una función de transferencia interna; una señal de lectura especifica una función de transferencia externa. Cada una se referencia desde la unidad de memoria. Al aceptar una de las señales de control, los circuitos internos de la unidad de memoria proporcionan la función deseada. Ciertos tipos de unidades de almacenamiento, debido a las características de su componente, destruyen la información almacenada en una celda cuando el bit en esa celda se lee al exterior. Dicha unidad se dice que es una memoria de lectura destructiva, en contraposición a una memoria no destructiva donde la información permanece en la celda después de que se lee al exterior. En cualquier caso, la información anterior siempre se destruye cuando se escribe información nueva. La secuencia del control interno en una memoria de lectura destructiva debe proporcionar señales de control que provoquen que la palabra se restablezca en sus celdas binarias si la aplicación exige una función no destructiva.

La transferencia de información hacia y desde los registros en memoria y el medio externo se comunica a través de un registro común llamado *registro buffer* de memoria (otros nombres son *registros de información* y *registro de almacén*). Cuando la unidad de memoria recibe una señal de control para *escritura*, el control interno interpreta el contenido de registro buffer como la configuración de bits de la palabra que va a almacenarse en un registro de memoria. Con una señal de control de *lectura*, el control interno envía la palabra de un registro de memoria al registro buffer. En cada caso el contenido del registro de dirección especifica el registro particular de memoria referenciado para escritura o lectura.

Ahora se resumen las características de transferencia de información de una unidad de memoria mediante un ejemplo. Considérese una unidad de memoria de 1 024 palabras con ocho bits por palabra. Para especificar 1 024 palabras, se necesita una dirección de diez bits, ya que 2^{10} = 1 024. Por consiguiente, el registro de dirección debe contener diez flip-flops. El registro buffer debe tener ocho flip-flops para almacenar el contenido de palabras transferidas hacia adentro y hacia afuera de la memoria. La unidad de memoria tiene 1 024 registros, con números asignados de dirección desde 0 hasta 1 023.

En la Fig. 7-25 se muestra el contenido inicial de tres registros: registro de dirección de memoria (MAR), registro buffer de memoria (MBR), y registro de memoria dirigida por MAR. Ya que el número binario equivalente en MAR es decimal 42, el registro de memoria dirigida por MAR es el que tiene el número de dirección 42.

La secuencia de las operaciones necesarias para comunicarse con la unidad de memoria con el propósito de transferir una palabra a la MBR es:

1. Transferencia de los bits de dirección de la palabra seleccionada al MAR.
2. Activación del control de entrada de *lectura*.

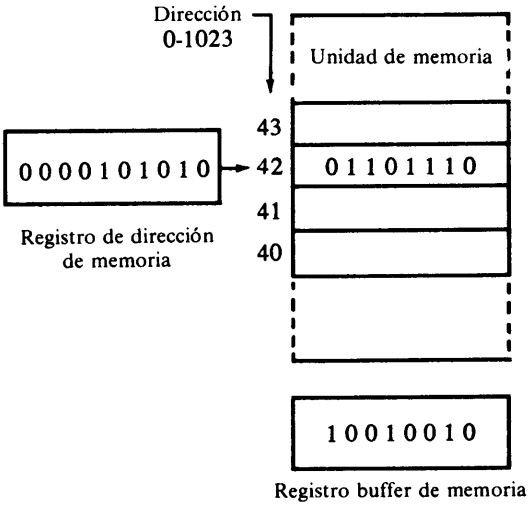


Figura 7-25 Valores iniciales de registros.

El resultado de la operación de lectura se indica en la Fig. 7-26(a). La información binaria al presente almacenada en el registro de memoria 42 se transfiere al registro MBR.

La secuencia de las operaciones necesarias para almacenar una palabra nueva en la memoria es:

1. Transferencia de los bits de dirección de la palabra seleccionada al registro MAR.
2. Transferencia de los bits de información de la palabra al registro MBR.
3. Activación de la entrada de control de *escritura.*

El resultado de la operación de escritura se indica en la Fig. 7-26(b). Los bits de información que provienen del MBR se almacenan en el registro de memoria 42.

En el ejemplo anterior, se supone una unidad de memoria con la propiedad de lectura al exterior no destructiva. Tales memorias pueden construirse con IC semiconductores. Retienen la información en el registro de memoria cuando el registro se muestrea durante el proceso de lectura de modo que no ocurre pérdida de información. Otro componente de uso común en las unidades de memoria es el núcleo magnético. En forma característica un núcleo magnético tiene lectura destructiva, esto es, pierde la información binaria almacenada durante el proceso de lectura. En la Sección 7-8 se presentan ejemplos de memoria de semiconductor y de núcleo magnético.

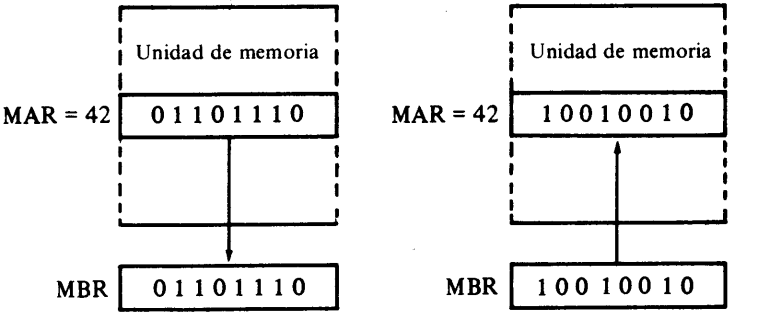
Debido a su propiedad de lectura destructiva, una memoria de núcleo magnético debe proporcionar una función adicional de control para restablecer la palabra en el registro de memoria. Una señal de control de lectura aplicada a una memoria de núcleo magnético transfiere el contenido de la palabra dirigida a un registro externo y,

al mismo tiempo, el registro de memoria se despeja en forma automática. La secuencia del control interno en una memoria de núcleo magnético entonces proporciona señales apropiadas que provocan el restablecimiento de la palabra en el registro de memoria. La transferencia de información en una memoria de núcleo magnético durante una operación de lectura se indica en la Fig. 7-27. Una operación de lectura destructiva transfiere la palabra seleccionada a un MBR pero deja el registro de memoria con todos los 0. La operación normal de memoria requiere que el contenido de la palabra seleccionada permanezca en la memoria después de una operación de lectura. Por tanto, es necesario pasar a través de una operación de *restablecimiento* que escribe el valor en el MBR al registro seleccionado de memoria. Durante la operación de restablecimiento, el contenido del MAR y el MBR debe permanecer sin cambio.

Una entrada de control de escritura aplicada a una memoria de núcleo magnético provoca una transferencia de información como se indica en la Fig. 7-28. Para transferir información nueva a un registro seleccionado, la información anterior debe borrarse primero despejando todos los bits de la palabra a 0. Después de que se ha hecho esto, el contenido del MBR puede transferirse a la palabra seleccionada. El MAR no debe cambiar durante la operación para asegurar que la misma palabra seleccionada que se despeja es la que recibe la información nueva.

Una memoria de núcleo magnético requiere dos medios ciclos, ya sea para lectura o escritura. El tiempo que necesita la memoria para pasar a través de ambos medios ciclos se conoce como tiempo de *ciclo de memoria.*

El modo de acceso de un sistema de memoria se determina por el tipo de componente usado. En una memoria de *acceso aleatorio*, debe considerarse que los registros están separados en el espacio, con cada registro ocupando una localización especial particular como en una memoria de núcleo magnético. En una memoria de *acceso secuencial*, la información almacenada en cierto medio no es accesible de inmediato, pero está disponible sólo a ciertos intervalos de tiempo. Una unidad de cinta magnética es de este tipo. Cada localización de memoria pasa las cabezas de lectura en turno, pero la información se lee hacia afuera sólo cuando se ha alcanzado la palabra requerida. El *tiempo de acceso* de una memoria es el tiempo requerido para



(a) Operación de lectura (b) Operación de escritura

Figura 7-26 Transferencia de información durante las operaciones de lectura y escritura.

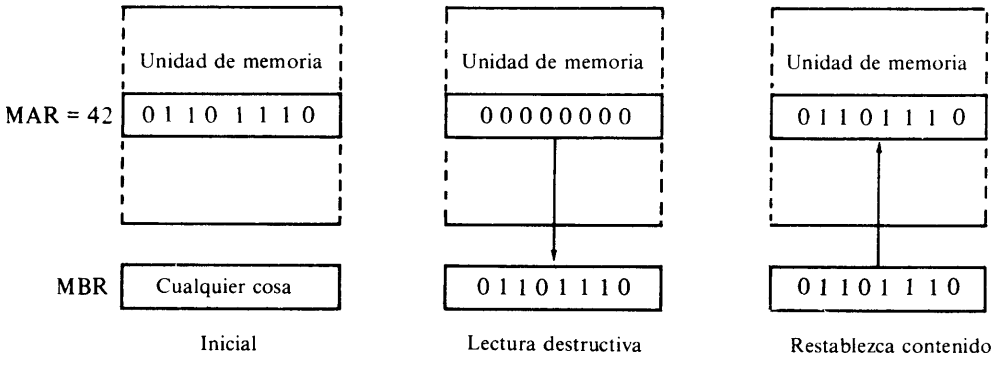


Figura 7-27 Transferencia de información en una memoria de núcleo magnético durante una operación de lectura.

seleccionar una palabra y leerla o bien escribirla. En una memoria de acceso aleatorio, el tiempo de acceso es siempre el mismo sin importar la localización particular de la palabra en el espacio. En una memoria secuencial, el tiempo de acceso depende de la posición de la palabra al tiempo del requerimiento. Si la palabra está surgiendo justamente del almacenamiento al tiempo que se le requiere, el tiempo de acceso es precisamente el tiempo necesario para leerla o escribirla. Pero si sucede que la palabra está en la última posición, el tiempo de acceso también incluye el tiempo requerido para que las otras palabras se muevan pasando las terminales. En consecuencia, el tiempo de acceso en una memoria secuencial es variable.

Las unidades de memoria cuyos componentes pierden la información almacenada con el tiempo o cuando se desconecta la potencia se dice que son *volátiles*. Una unidad semiconductora de memoria es de esta categoría ya que sus celdas binarias necesitan potencia externa para mantener las señales necesarias. En contraste, una unidad de memoria no volátil, como un núcleo magnético o discos magnéticos, retiene su información almacenada después de la remoción de la potencia. Esto se debe a que la información almacenada en los componentes magnéticos se manifiesta por la dirección de magnetización, la cual se retiene cuando se desconecta la potencia. En las computadoras digitales es deseable una propiedad no volátil porque muchos progra-

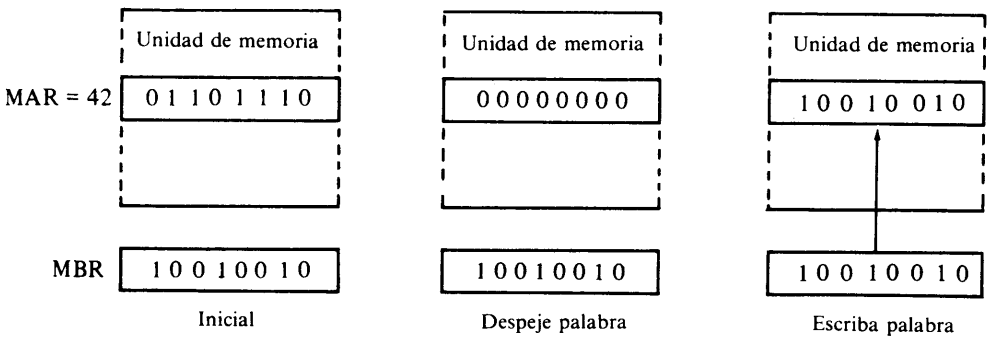


Figura 7-28 Transferencia de información en una memoria de núcleo magnético durante una operación de escritura.

mas útiles se dejan en forma permanente en la unidad de memoria. Cuando la potencia se desconecta y se vuelve a conectar otra vez, los programas previamente almacenados y otra información no se pierde, sino que continúan su residencia en la memoria.

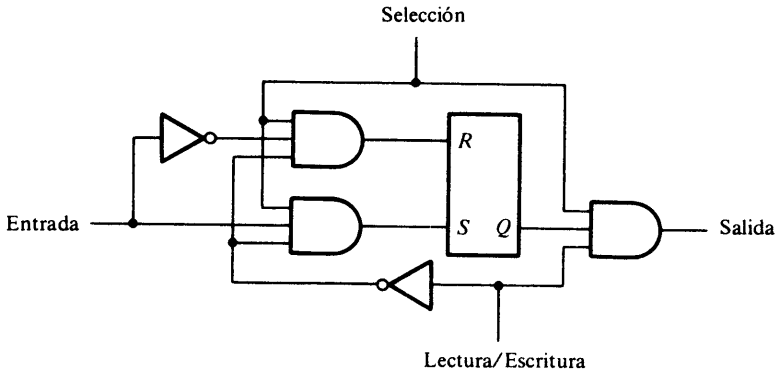
7-8 EJEMPLOS DE MEMORIAS DE ACCESO ALEATORIO

La construcción interna de dos tipos diferentes de memorias de acceso aleatorio se presenta en forma de diagrama en esta sección. El primer tipo está construido con flip-flops y compuertas y el segundo con núcleos magnéticos. Para facilitar que se incluya completa la unidad de memoria en un diagrama, debe usarse una capacidad de almacenamiento indicada. Por esta razón, las unidades de memoria que se presentan aquí tienen una capacidad pequeña de 12 bits arreglados en cuatro palabras de tres bits cada una. Las memorias de acceso aleatorio comerciales pueden tener una capacidad de miles de palabras y cada palabra puede variar entre 8 y 64 bits. La construcción lógica de las unidades de memoria de gran capacidad sería una extensión directa de la configuración que aquí se muestra.

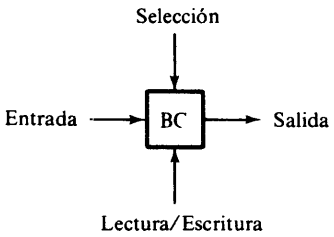
Circuito integrado de memoria

La construcción interna de una memoria de acceso aleatorio de m palabras con n bits por palabra consta de $m \times n$ celdas de almacenamiento binario y la lógica asociada para seleccionar palabras individuales. La celda de almacenamiento binario es el bloque básico de construcción de una unidad de memoria. El circuito equivalente de una celda binaria que almacena un bit de información se muestra en la Fig. 7-29. Aunque la celda se muestra para incluir compuertas y un flip-flop, está construida en forma interna con dos transistores que tienen entradas múltiples. Una celda de almacenamiento binario debe ser muy pequeña con objeto de poder agrupar tantas celdas como sea posible en la pequeña área disponible en la pastilla de circuito integrado. La celda binaria tiene tres entradas y una salida. La entrada de selección capacita la celda para lectura o escritura. La entrada de lectura/escritura determina la operación de la celda cuando se selecciona. Un 1 en la entrada de lectura/escritura forma una trayectoria desde el flip-flop a la terminal de salida. La información en la terminal de entrada se transfiere al flip-flop cuando el control de lectura/escritura es 0. Obsérvese que el flip-flop opera sin pulsos de reloj y que su propósito es almacenar el bit de información en la celda binaria.

Las memorias en el circuito integrado algunas veces tienen una línea única para el control de lectura y escritura. Un estado binario en la línea única especifica una operación de lectura y el otro estado especifica una operación de escritura. Además, se incluye una o más líneas de habitación para proporcionar los medios para seleccionar el IC y para la expansión de varios paquetes en una unidad de memoria con un número más grande de palabras. En la Fig. 7-30 se muestra la construcción lógica de un IC RAM. Consta de 4 palabras de 3 bits cada una, para un total de 12 celdas binarias. Las casillas pequeñas etiquetadas BC representan celdas binarias, y las tres entradas y la salida única en cada BC son como se especificó en el diagrama de la Fig. 7-29.



(a) Diagrama lógico



(b) Diagrama de bloque

Figura 7-29 Celda de memoria.

Las dos líneas de entrada de dirección van a través de un decodificador interno de 2-a-4 línea. El decodificador se habilita con la entrada de habilitación de memoria. Cuando la habilitación de memoria es 0, todas las salidas de decodificador son 0 y ninguna de las palabras de memoria se selecciona. Cuando la habilitación de la memoria es 1, se selecciona una de cuatro palabras, dependiendo del valor de las dos líneas de dirección. Ahora, con el control de lectura/escritura en 1, los bits de la palabra seleccionada pasan a través de las tres compuertas OR a las terminales de salida. Las celdas binarias no seleccionadas producen 0 en las entradas de las compuertas OR y no tienen efecto en las salidas. Con el control de lectura/escritura en 0, la información disponible en las líneas de entrada se transfiere a las celdas binarias de la palabra seleccionada. Las celdas binarias no seleccionadas en las otras palabras están inhabilitadas por sus entradas de selección y sus valores previos permanecen sin cambio. Con el control de habilitación de memoria en 0, el contenido de todas las celdas en la memoria permanece sin cambio, sin importar el valor del control de lectura/escritura.

Los IC RAM tienen construcción interna con celdas que tienen una capacidad OR alambrada. Esto elimina la necesidad de las compuertas OR en el diagrama. Las

líneas de las salidas externas también pueden formar lógica alambrada para facilitar la conexión de dos o más paquetes IC para formar una unidad de memoria con un número mayor de palabras.

Memoria de núcleo magnético

Una memoria de núcleo magnético usa núcleos magnéticos para almacenar la información binaria. Un núcleo magnético tiene forma toroidal y está hecho de material magnético. En contraste con un flip-flop semiconductor que necesita sólo una cantidad física, como por ejemplo voltaje, para su operación un núcleo magnético emplea

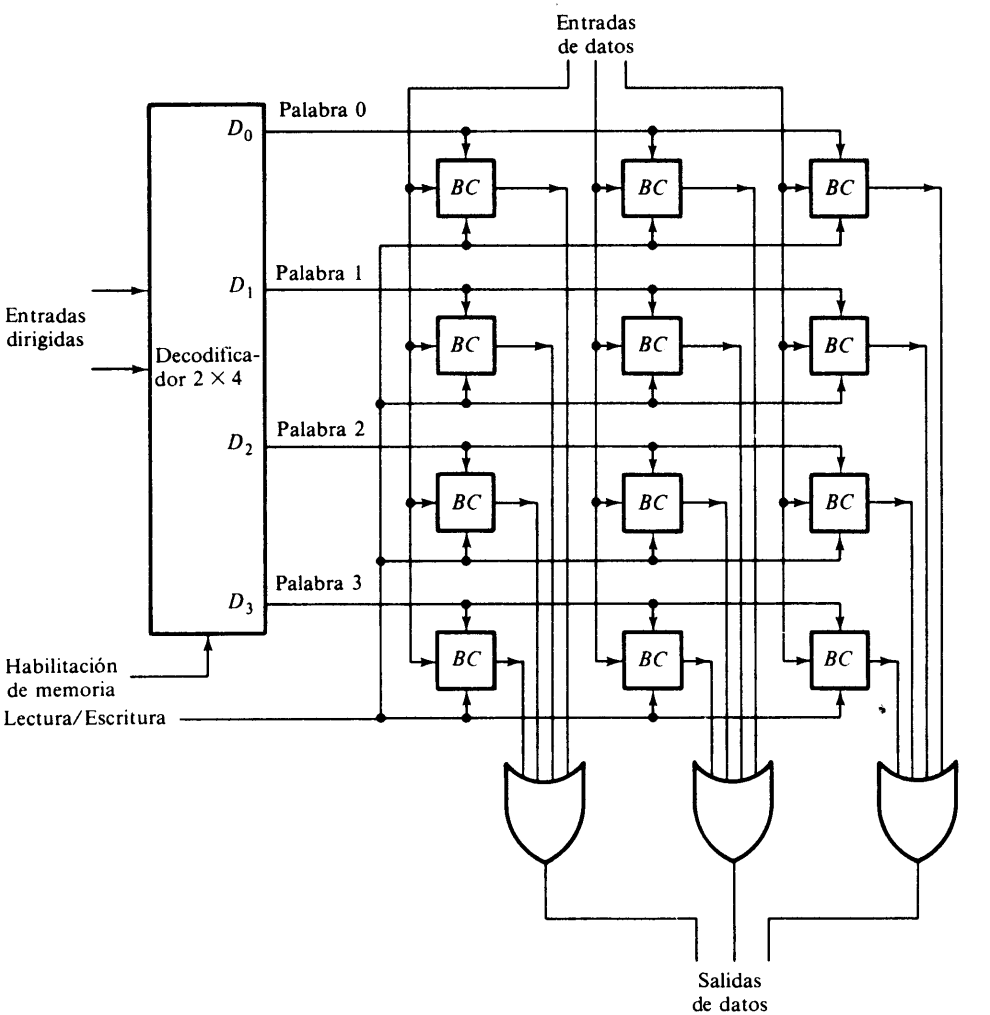


Figura 7-30 Memoria en circuito integrado.

tres cantidades físicas: corriente, flujo magnético y voltaje. La señal que excita al núcleo es un pulso de *corriente* en un alambre que pasa a través del núcleo. La información binaria almacenada se representa por la dirección del *flujo magnético* dentro del núcleo. La información binaria de salida se extrae mediante un alambre que se eslabona al núcleo en la forma de un pulso de *voltaje*.

La propiedad física que hace que un núcleo magnético sea adecuado para el almacenamiento binario es su circuito de histéresis, que se muestra en la Fig. 7-31(c). Este circuito es una gráfica de corriente comparada con el flujo magnético y tiene la forma de un circuito cuadrado. Con corriente cero, un flujo que es positivo (dirección en sentido contrario a las manecillas del reloj) o bien negativo (dirección en el sentido de las manecillas del reloj) permanece en el núcleo magnetizado. Una dirección, por ejemplo magnetización contraria al sentido de las manecillas del reloj, se usa para representar un 1 y la otra para representar un 0.

Un pulso de corriente aplicado al devanado a través del núcleo puede cambiar la dirección de magnetización. Como se muestra en la Fig. 7-31(a), la corriente en la dirección hacia abajo produce flujo en la dirección de las manecillas del reloj, provocando que el núcleo pase al estado 0. En la Fig. 7-31(b) se muestra la corriente y la dirección del flujo para almacenar un 1. La trayectoria que toma el flujo cuando se aplica el pulso de corriente se indica con flechas en el circuito de histéresis.

La toma de la lectura de la información binaria almacenada en el núcleo se complica por el hecho de que el flujo no puede detectarse cuando no tiene cambio. Sin embargo, si el flujo cambia con respecto al tiempo, induce un voltaje en un alambre que se enlaza con el núcleo. Por tanto, la toma de lectura puede realizarse con la aplicación de una corriente en la dirección negativa como se muestra en la Fig. 7-32. Si el núcleo está en el estado 1, la corriente invierte la dirección de magnetización, y el cambio resultante del flujo produce un pulso de voltaje en el alambre sensor. Si el núcleo ya está en el estado 0, la corriente negativa deja magnetizado al núcleo en la misma dirección, causando una perturbación muy ligera del flujo magnético, lo cual resulta en un voltaje de salida muy pequeño en el alambre sensor. Obsérvese que ésta es una toma de lectura destructiva, ya que la corriente leída siempre devuelve el núcleo al estado 0. Se pierde el valor almacenado previamente.

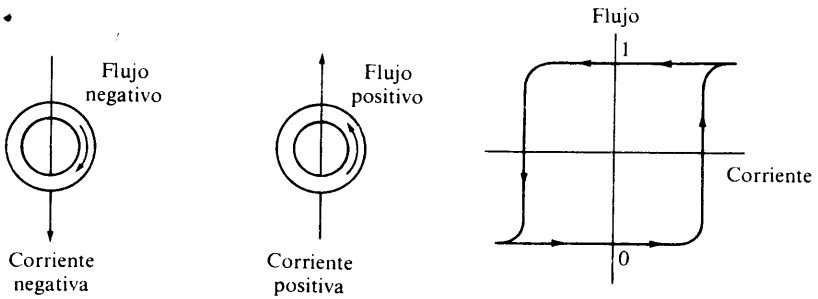


Figura 7-31 Almacén de un bit en un núcleo magnético.

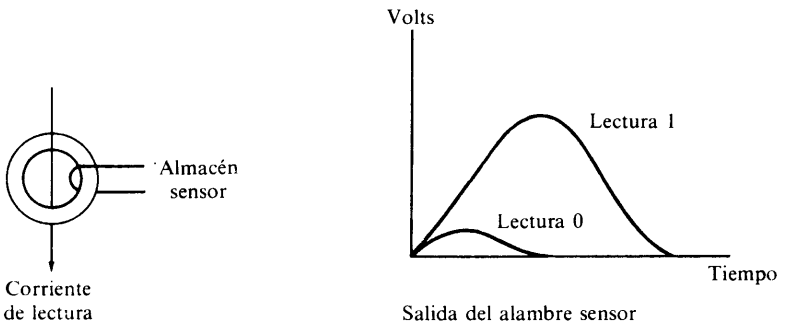


Figura 7-32 Lectura de un bit a partir de un núcleo magnético.

En la Fig. 7-33 se muestra la organización de una memoria de núcleo magnético que contiene cuatro palabras con tres bits cada una. Al compararla con la unidad de memoria IC que se muestra en la Fig. 7-30, se observa que la celda binaria ahora es un núcleo magnético con los alambres eslabonados. La excitación del núcleo se lleva a cabo mediante un pulso de corriente generado en un impulsor (DR). La información de salida pasa a través de un amplificador sensor (SA) cuyas salidas establecen los flip-flops correspondientes en el registro buffer. Se eslabonan tres alambres a cada núcleo. El alambre de palabras se excita por un impulsor de palabra que pasa a través de los tres núcleos de una palabra. Un alambre de bit se excita con un impulsor de bit y pasa a través de cuatro núcleos en la misma posición de bit. Los alambres sensores eslabonan los mismos núcleos como el alambre de bit y se aplica a un amplificador sensor que forma el pulso de voltaje cuando se lee un 1 y rechaza la pequeña perturbación cuando se lee un 0.

Durante una operación de lectura, un pulso de corriente del impulsor de palabra se aplica a los núcleos de la palabra seleccionada por el decodificador. La corriente de lectura está en la dirección negativa (Fig. 7-32) y causa que todos los núcleos de la palabra seleccionada pasen al estado 0, sin importar su estado previo. Los núcleos que previamente contienen un 1 cambian su flujo e inducen un voltaje en su alambre sensor. Los flujos de los núcleos que ya contienen un 0 no cambian. El pulso de voltaje en un alambre sensor de núcleos con un 1 previo se amplifica en el amplificador sensor y establece el flip-flop correspondiente en el registro buffer.

Durante una operación de escritura, el registro buffer mantiene la información por almacenarse en la palabra especificada por el registro de dirección. Se supone que todos los núcleos en la palabra seleccionada están despejados inicialmente, esto es, todos están en el estado 0, de modo que los núcleos que requieren un 1 necesitan pasar por un cambio de estado. Se genera en forma simultánea un pulso de corriente en el impulsor de palabra seleccionado por el decodificador y en el impulsor de bit, cuyo flip-flop correspondiente de registro buffer contiene un 1. Ambas corrientes están en la dirección positiva, pero su magnitud es sólo la mitad de la necesaria para cambiar el flujo al estado 1. Esta media corriente es demasiado pequeña por sí misma para cambiar la dirección de magnetización. Pero la suma de dos medias corrientes es suficiente para cambiar la dirección de la magnetización al estado 1. Un núcleo cambia

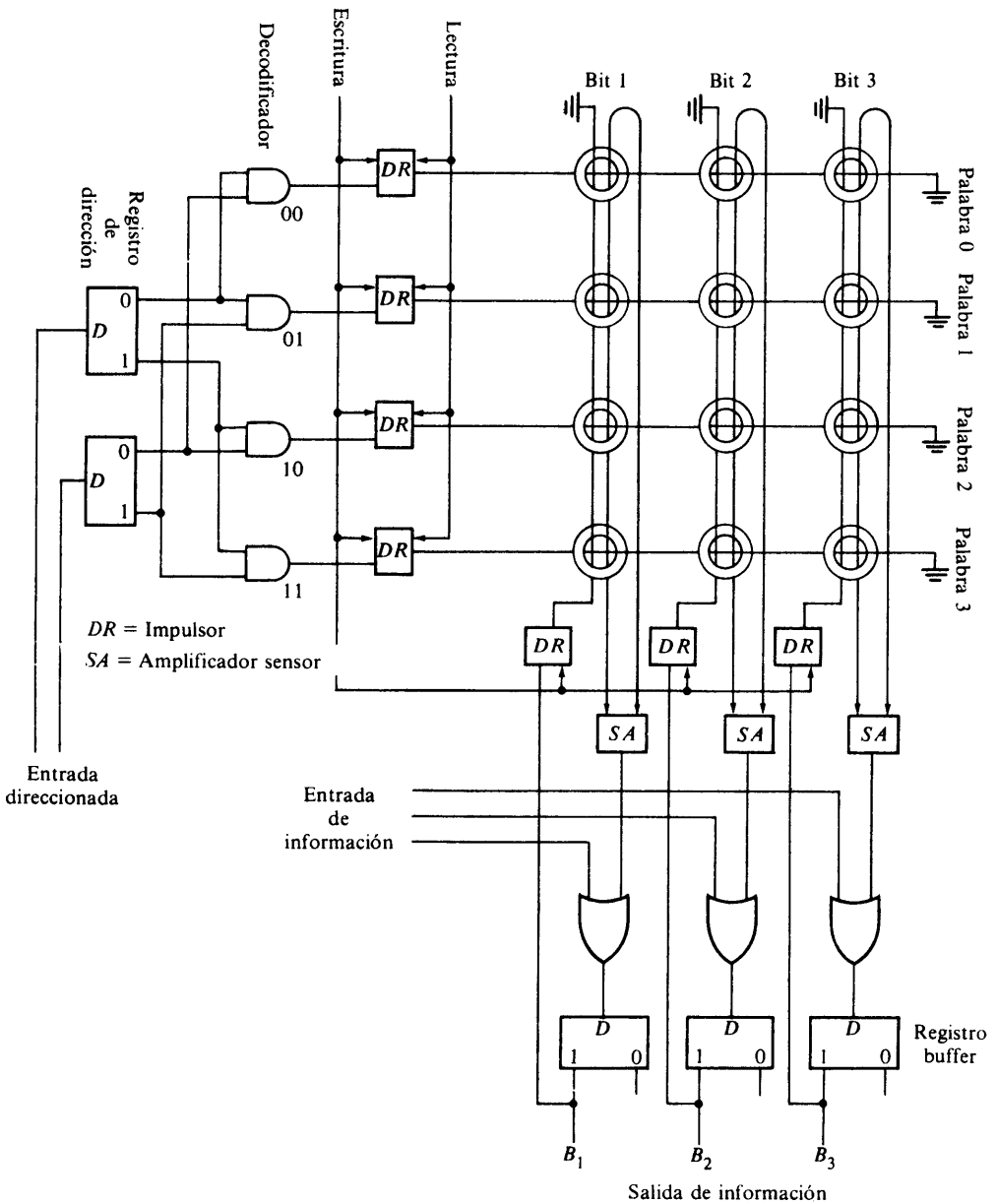


Figura 7-33 Unidad de memoria en núcleo magnético.

al estado 1 sólo si hay una coincidencia de dos medias corrientes que provienen de un impulsor de palabra y un impulsor de bit. La dirección de magnetización de un núcleo no cambia si recibe sólo media corriente de uno de los impulsores. El resultado es que la magnetización de los núcleos se cambia al estado 1 sólo si se intersecan los alambres de palabra y bit esto es, sólo en la palabra seleccionada y sólo en la posición de bit en la cual el registro buffer está en 1.

Las operaciones de lectura y de escritura que se describieron con anterioridad son incompletas, porque la información almacenada en la palabra que se selecciona se destruye por el proceso de lectura y la operación de escritura trabaja en forma apropiada sólo si los núcleos están despejados inicialmente. Como se mencionó en la Sección 7-7, una operación de lectura debe ir seguida por otro ciclo que establece valores previamente almacenados en los núcleos. Una operación de escritura está precedida por un ciclo que despeja los núcleos de la palabra seleccionada.

El restablecimiento de la operación durante un ciclo de lectura es equivalente a una operación de escritura, la cual, en efecto, escribe la información leída previamente desde el registro buffer devolviéndola a la palabra seleccionada. La operación de despeje durante un ciclo de escritura es equivalente a una operación de lectura que destruye la información almacenada pero evita que la información de lectura alcance el registro buffer al inhibir el amplificador sensor. Los ciclos de restablecimiento y despeje normalmente se inician por el control interno de memoria, de modo que la unidad de memoria tiene la apariencia exterior de poseer una propiedad de extracción de lectura no destructiva.

BIBLIOGRAFIA

1. *The TTL Data Book for Design Engineers.* Dallas: Texas Instruments, Inc., 1976.
2. Blakeslee, T. R., *Digital Design with Standard MSI and LSI.* New York: John Wiley & Sons, 1975.
3. Barna A., y D. I. Porat, *Integrated Circuits in Digital Electronics.* New York: John Wiley & Sons, 1973.
4. Taub, H., y D. Schilling, *Digital Integrated Electronics.* New York: McGraw-Hill Book Co., 1977.
5. Grinich, V. H., y H. G. Jackson, *Introduction to Integrated Electronics.* New York: McGraw-Hill Book Co., 1975.
6. Kostopoulos, G. K., *Digital Engineering.* New York: McGraw-Hill Book Co., 1975.
7. Scott, N. R., *Electronic Computer Technology.* New York: McGraw-Hill Book Co., 1970, cap. 10.
8. Kline, R. M., *Digital Computer Design.* Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1977, cap. 9.

PROBLEMAS

7-1. El registro en la Fig. 7-1 transfiere la información de entrada a los flip-flops cuando la entrada *CP* pasa a través de una transición de borde positivo. Modifique el circuito de modo que la información de entrada se transfiera al registro cuando un pulso de reloj pasa a través de una transición de borde negativo, siempre que una entrada de carga de control es igual al binario 1.

7-2. El registro en la Fig. 7-3 carga las entradas durante una transición negativa de un pulso de reloj. ¿Qué cambios internos son necesarios para que las entradas se carguen durante el borde negativo del pulso?

7-3. Verifique el circuito en la Fig. 7-5 usando mapas para simplificar las ecuaciones del estado siguiente.

7-4. Diseñe el circuito secuencial cuya tabla de estado se da a continuación, usando un registro de 2-bit y compuertas combinacionales.

Estado presente		Entrada	Estado siguiente	
A	*B*	*x*	*A*	*B*
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	1	1
1	1	0	1	0
1	1	1	0	1

7-5. Diseñe un circuito secuencial cuyo diagrama de estado está dado en la Fig. 6-27 usando un registro de 3-bit y una ROM 16×4.

7-6. El contenido de un registro de corrimiento de 4-bit en la forma inicial es 1101. El registro se corre seis veces a la derecha, con la entrada serial de 101101. ¿Cuál es el contenido del registro después de cada corrimiento?

7-7. ¿Cuál es la diferencia entre las transferencias serial y paralela? ¿Qué tipo de registro se usa en cada caso?

7-8. El registro de corrimiento bidireccional de 4-bit que se muestra en la Fig. 7-9 está encerrado dentro de un paquete IC.
(a) Dibuje un diagrama de bloques del IC mostrando todas las entradas y salidas.
(b) Dibuje un diagrama de bloques usando tres IC para producir un registro de corrimiento bidireccional de 12-bit.

7-9. El adicionador serial que se muestra en la Fig. 7-10 usa registros de corrimiento de 4-bit. El registro *A* mantiene el número binario 0101 y el registro *B* mantiene 0111. El flip-flop *Q* de la cuenta que se lleva está despejado al inicio. Haga la lista de los valores binarios en el registro *A* y el flip-flop *Q* después de cada corrimiento.

7-10. ¿Qué cambio se necesita en el circuito que se muestra en la Fig. 7-11 para convertirlo en un circuito que reste el contenido de *B* del contenido de *A*?

7-11. Diseñe un contador serial; en otras palabras, determine el circuito que debe incluirse en forma externa con un registro de corrimiento con objeto de obtener un contador que opere en una forma serial.

7-12. Dibuje el diagrama de un contador binario de ondulación de 4-bit usando flip-flops que disparen en el borde positivo.

7-13. Un flip-flop tiene un retardo de 20-ns desde el tiempo en que su entrada *CP* pasa desde 1 a 0 al tiempo que se complementa la salida. ¿Cuál es el retardo máximo en un contador binario de ondulación de 10-bit que usa esos flip-flops? ¿A qué frecuencia máxima puede operar el contador con confiabilidad?

7-14. ¿Cuántos flip-flops deben complementarse en un contador binario de ondulación de 10-bit para alcanzar la cuenta siguiente después de 0111111111?

7-15. Dibuje el diagrama de un contador binario de ondulación de 4-bit hacia abajo usando flip-flops que disparan en la (a) transicición de borde positivo y (b) transición de borde negativo.

7-16. Dibuje un diagrama de temporizado similar al que se muestra en la Fig. 7-15 para el contador binario de ondulación que se muestra en la Fig. 7-12.

7-17. Determine el estado siguiente para cada uno de los seis estados sin uso en el contador de ondulación BCD que se muestra en la Fig. 7-14. ¿El contador arranca por sí mismo?

7-18. El contador de ondulación que se muestra en la Fig. P7-18 usa flip-flop que disparan en la transición de borde negativo de la entrada *CP*. Determine la secuencia de conteo del contador. ¿El contador arranca por si solo?

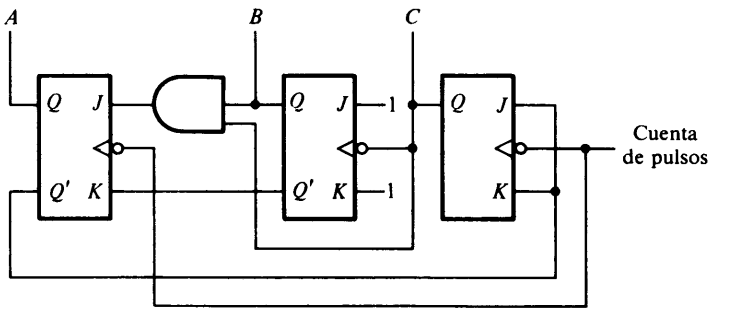


Figura P7-18 Contador de ondulación.

7-19. ¿Qué sucede en el contador en la Fig. 7-18 si tanto la entrada de incremento como la entrada de decremento son iguales a 1 al mismo tiempo? Modifique el circuito de modo de que cuente hacia arriba si ocurre esta condición.

7-20. Verifique las funciones de entrada del flip-flop del contador síncrono BCD especificado en la Tabla 7-5. Dibuje el diagrama lógico del contador BCD e incluya una entrada de control para habilitar el conteo.

7-21. Diseñe un contador síncrono BCD con flip-flops *JK*.

7-22. Muestre las conexiones externas para cuatro IC contadores binarios con carga paralela (Fig. 7-19) para producir un contador binario de 16-bit. Use un diagrama de bloques para cada IC.

7-23. Construya un contador BCD usando el circuito MSI que se muestra en la Fig. 7-19.

7-24. Construya un contador mod-12 usando el circuito MSI especificado en la Fig. 7-19. Dé cuatro alternativas.

7-25. Mediante el uso de dos circuitos MSI como se especifica en la Fig. 7-19, construya un contador binario que cuente desde 0 hasta el binario 64.

7-26. Utilizando la variable *paro* de la Fig. 7-21 como señal de arranque, construya un segundo control de tiempo de palabra que permanezca encendido por un periodo de 16 pulsos de reloj.

7-27. Muestre un contador binario de n-bit conectado a un decodificador de n-a-2^n líneas es equivalente a un contador de anillo con 2^n flip-flops. Dibuje los diagramas de bloque de ambos circuitos para $n = 3$. ¿Cuántas señales de temporizado se generan?

7-28. Incluya una entrada de habilitación a decodificador en la Fig. 7-22(b) y conéctelo a los pulsos de reloj. Dibuje las señales de temporizado que se generan ahora en las salidas del decodificador.

7-29. Complete el diseño del contador Johnson que se muestra en la Fig. 7-23, ilustrando las salidas de las ocho señales de temporizado.

7-30. (a) Haga la lista de los ocho estados sin uso en el contador de anillo con conmutación en cola en la Fig. 7-23. Determine el estado siguiente para cada estado sin uso y muestre que, si el circuito se encuentra por sí mismo en un estado inválido, no regresa a un estado inválido. (b) Modifique el circuito como se recomienda en el texto y muestre que (1) el circuito produce la misma secuencia de estados como se muestra en la lista que se incluye en la Fig. 7-23(b) y (2), el circuito alcanza un estado válido desde cualquiera de los estados sin uso.

7-31. Construya un contador Johnson con diez señales de temporizado.

7-32. (a) La unidad de memoria que se muestra en la Fig. 7-24 tiene una capacidad de 8 192 palabras de 32 bits por palabra. ¿Cuántos flip-flops se necesitan para el registro de dirección de memoria y el registro buffer de memoria? (b) ¿Cuántas palabras contendrá la unidad de memoria si el registro de dirección tiene 15 bits?

7-33. Cuando el número de palabras que van a seleccionarse en una memoria es demasiado grande, es conveniente usar una celda de almacenamiento binario con dos entradas seleccionadas: una X (horizontal) y una Y (vertical) entrada seleccionada. Tanto X como Y deben habilitarse para seleccionar la celda.
(a) Dibuje una celda binaria similar a la que se muestra en la Fig. 7-29 con X y Y entradas seleccionadas.
(b) Muestre cómo dos decodificadores 4×16 pueden usarse para seleccionar una palabra en una memoria de 256-palabras.

7-34. (a) Dibuje un diagrama de bloques de la memoria 4×3 que se muestra en la Fig. 7-30. Muestre todas las entradas y todas las salidas. (b) Construya una memoria 8×3 usando dos de esas unidades. Utilice una construcción de diagrama de bloques.

7-35. Se requiere construir una memoria con 256 palabras, 16 bits por palabra, organizada como se muestra en la Fig. 7-33. Están disponibles núcleos en una matriz de 16 renglones y 16 columnas.

(a) ¿Cuántas matrices se necesitan?

(b) ¿Cuántos flip-flops están en los registros de dirección y buffer?

(c) ¿Cuántos núcleos reciben corriente durante un ciclo de lectura?

(d) ¿Cuántos núcleos reciben cuando menos media corriente durante un ciclo de escritura?

Máquinas de estado algorítmico (ASM)

8

8-1 INTRODUCCION

La información binaria almacenada en un sistema digital puede clasificarse ya sea como datos o control de información. Los datos son elementos discretos de información que se manipulan para realizar tareas de aritmética, lógica, corrimiento y otras tareas similares de procesamiento de datos. Estas operaciones se implantan con componentes digitales como sumadores, decodificadores, multiplexores, contadores y registros de corrimiento. La información de control proporciona señales de mando que supervisan las diversas operaciones de la sección de datos con objeto de llevar a cabo las tareas deseadas de procesamiento de datos. El diseño lógico de un sistema digital puede dividirse en dos partes distintas. Una parte se ocupa del diseño de los circuitos digitales que llevan a cabo las operaciones de procesamiento de datos. La otra parte se ocupa del diseño del circuito de control que supervisa las operaciones y sus secuencias.

Las relaciones entre el control lógico y el procesador de datos en un sistema digital se muestran en la Fig. 8-1. El subsistema procesador de datos manipula los datos en los registros de acuerdo con los requisitos del sistema. El control lógico inicia los mandos en secuencia apropiada al procesador de datos. El control lógico usa las

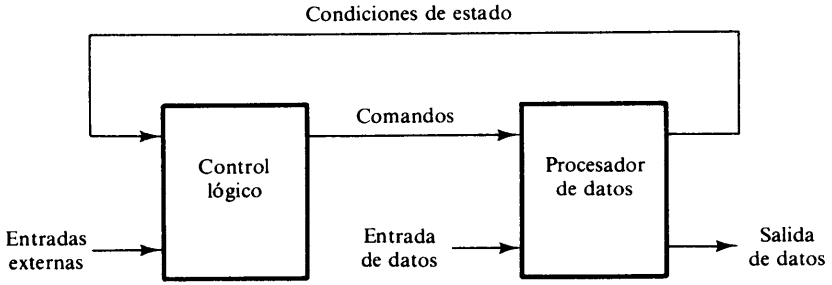


Figura 8-1 Interacción del control y el procesador de datos.

condiciones de estado del procesador de datos para servir como variables de decisión para determinar la secuencia de las señales de control.

El control lógico que genera las señales para dar la secuencia de las operaciones en el procesador de datos es un circuito secuencial cuyos estados internos dictan los mandos de control para el sistema. En cualquier momento el estado del control secuencial inicia un conjunto prescrito de mandos. Dependiendo de las condiciones de estado y otras entradas externas, el circuito secuencial pasa al estado siguiente para iniciar otras operaciones. Los circuitos digitales que actúan como el control lógico proporcionan una secuencia de tiempo de señales para iniciar las operaciones en el procesador de datos y determinar el siguiente estado del mismo subsistema de control.

La secuencia de control y las tareas de procesamiento de datos de un sistema digital se especifican mediante un algoritmo en el hardware. Un algoritmo consta de un número finito de pasos de procedimiento que especifican como obtener una solución a un problema. Un algoritmo de hardware es un procedimiento para implantar el problema con una pieza dada de equipo. La parte del diseño digital más creativa y más desafiante es la formulación de algoritmos de hardware para lograr los objetivos requeridos.

Una forma conveniente de especificar la secuencia de los pasos de proceso y las trayectorias de decisión para un algoritmo es un diagrama de flujo. El diagrama de flujo para un algoritmo de hardware traduce la estipulación en palabras a un diagrama de información que enumera la secuencia de las operaciones junto con las condiciones necesarias para su ejecución. Un diagrama especial de flujo que ha sido desarrollado específicamente para definir algoritmos de hardware digitales se denomina diagrama de máquina de estado algorítmico (ASM). Una máquina de estado es otro término para un circuito secuencial, el cual es la estructura básica de un sistema digital.

El diagrama ASM se asemeja a un diagrama convencional de flujo pero se interpreta en forma algo diferente. Un diagrama convencional de flujo describe la secuencia de los pasos de procedimiento y las trayectorias de decisión para un algoritmo sin ocuparse de sus relaciones en el tiempo. El diagrama ASM describe la secuencia de eventos lo mismo que las relaciones de temporizado entre los estados de un controlador secuencial y los eventos que ocurren cuando pasa de un estado al siguiente. En forma específica está adaptado para especificar con precisión la secuencia de control y las operaciones de procesamiento de datos en un sistema digital, tomando en consideración las restricciones del hardware digital.

Este capítulo presenta un método de diseño digital lógico que usa el diagrama ASM. Los diversos bloques que componen el diagrama se definen primero. Las relaciones de temporizado entre los bloques se explican entonces mediante ejemplos. Las diversas formas de implantar el control lógico se exponen junto con ejemplos de diagramas ASM y los sistemas digitales correspondientes que representan.

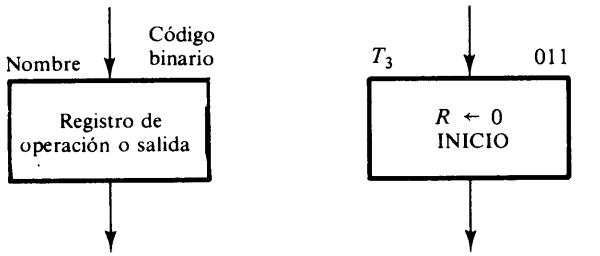
8-2 DIAGRAMA ASM

El diagrama ASM es un tipo especial de diagrama de flujo adecuado para describir las operaciones secuenciales en un sistema digital. El diagrama está compuesto de tres

elementos básicos: la casilla de estado, la casilla de decisión y la casilla condicional. Un estado en la secuencia de control se indica con una casilla de estado, como se muestra en la Fig. 8-2. La forma de la casilla de estado es rectangular dentro de la cual se escriben operaciones de registro o nombres de señal de salida que el control genera mientras se encuentra en este estado. El estado recibe un nombre simbólico, el cual se coloca en la esquina superior izquierda de la casilla. El código binario asignado al estado se coloca en la esquina superior de la derecha. En la Fig. 8-2(b) se muestra un ejemplo específico de una casilla de estado. El estado tiene el nombre simbólico T_3, y el código binario asignado a él es 011. Dentro de la casilla se escribe la operación del registro $R \leftarrow 0$, la cual indica que el registro R se despeja a 0 cuando el sistema está en el estado T_3. Por ejemplo, el nombre START dentro de la casilla puede indicar una señal de salida que inicia cierta operación.

La casilla de decisión escribe el efecto de una entrada en el subsistema de control. Es una casilla con forma de rombo con dos o más trayectorias de salida, como se muestra en la Fig. 8-3. La condición de entrada que va a probarse está escrita dentro de la casilla. Una trayectoria de salida se toma si la condición es cierta y la otra cuando la condición es falsa. Cuando una condición de entrada está asignada a un valor binario, las dos trayectorias se indican por 1 y 0.

Las casillas de estado y decisión se conocen por su uso en los diagramas convencionales de flujo. El tercer elemento, la casilla condicional, es de uso exclusivo en el diagrama ASM. La forma ovalada de la casilla condicional se muestra en la Fig. 8-4. Los lados redondeados la diferencían de la casilla de estado. La



(a) Descripción general (b) Ejemplo específico

Figura 8-2 Caja de estado.

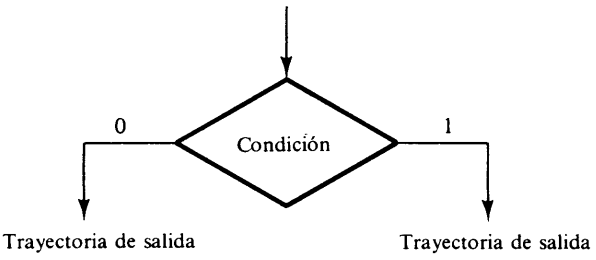


Figura 8-3 Caja de decisión.

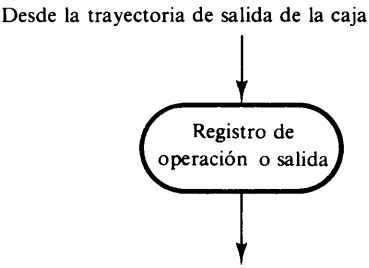


Figura 8-4 Caja condicional.

trayectoria de entrada a la casilla condicional debe llegar desde una de las trayectorias de salida de una casilla de decisión. Las operaciones de registro o salidas listadas dentro de la casilla condicional se generan durante un estado dado siempre que se satisfaga la condición de entrada. En la Fig. 8-5 se muestra un ejemplo con una casilla condicional. El control genera una señal de salida de START cuando se encuentra en el estado T_1. Mientras se encuentra en el estado T_1, el control verifica el estado de la entrada E. Si $E = 1$, entonces R se despeja a 0; en otra forma, R permanece sin cambio. En cualquier caso, el estado siguiente es T_2.

Bloque ASM

Un bloque ASM es una estructura que consta de una casilla de estado y todas las casillas de decisión y condicionales conectadas a sus trayectorias de salida. Un bloque ASM tiene una entrada y cualquier número de trayectorias de salida representadas por la estructura de las casillas de decisión. Un diagrama ASM consta de uno o más bloques interconectados. En la Fig. 8-6 se muestra un ejemplo de un bloque ASM.

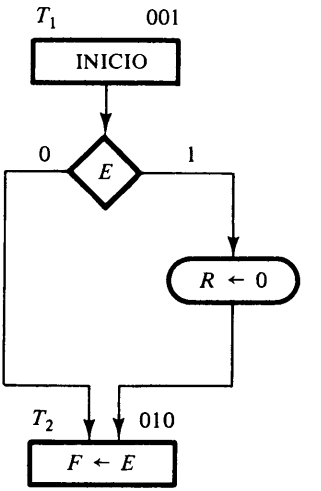


Figura 8-5 Ejemplo con caja condicional.

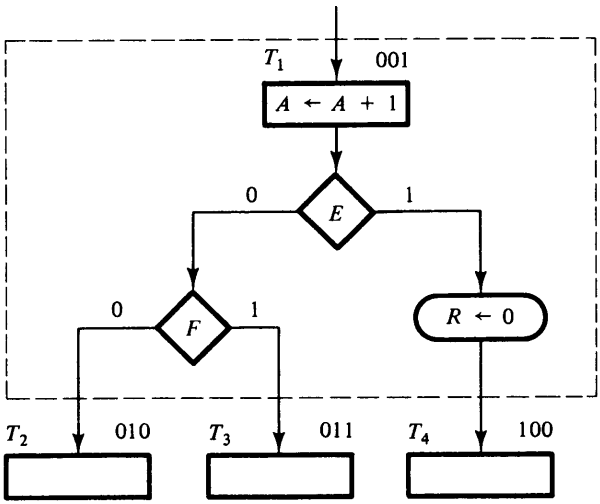


Figura 8-6 Bloque ASM.

Asociadas con el estado T_1 se encuentran dos casillas de decisión y una casilla condicional. El diagrama distingue el bloque con líneas punteadas alrededor de la estructura entera, pero esto por lo común no se hace, ya que el diagrama ASM define en forma única cada bloque mediante su estructura. Una casilla de estado sin casillas de decisión o condicionales constituye un bloque simple.

Cada bloque en el diagrama ASM describe el estado del sistema durante el intervalo de un pulso de reloj. Las operaciones dentro de las casillas de estado y condicionales en la Fig. 8-6 se ejecutan con un pulso común de reloj mientras el sistema se encuentra en el estado T_1. El mismo pulso de reloj también transfiere el sistema controlador a uno de los estados siguientes, T_2, T_3 o T_4, como dictan los valores binarios de E y F.

El diagrama ASM es muy similar a un diagrama de estado. Cada bloque de estado es equivalente a un estado en un circuito secuencial. La casilla de decisión es equivalente a la información binaria escrita a lo largo de las líneas dirigidas que conectan dos estados en un diagrama de estado. Como consecuencia, algunas veces es conveniente convertir el diagrama en un diagrama de estado y entonces usar los procedimientos de circuito secuencial para diseñar el control lógico. Como una ilustración, el diagrama ASM en la Fig. 8-6 se dibuja como un diagrama de estado en la Fig. 8-7. Los tres estados están simbolizados por círculos con su valor binario escrito dentro de cada círculo. Las líneas dirigidas indican las condiciones que determinan el Estado siguiente. Las operaciones incondicionales y condicionales que deben llevarse a cabo no se indican en el diagrama de estado.

Operaciones de registro

Un sistema digital con bastante frecuencia se define por los registros que contiene y las operaciones que se realizan en los datos almacenados en ellos. Un registro en su sentido

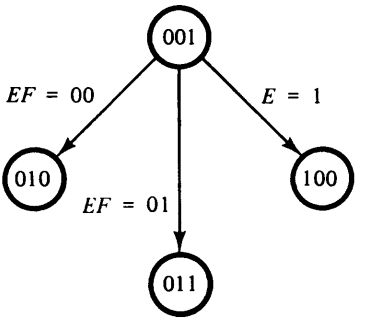


Figura 8-7 Diagrama de estado equivalente a la carta ASM en la Fig. 8-6.

más amplio incluye registros de almacenamiento, registros de corrimiento, contadores y flip-flops solos. Los ejemplos de operaciones de registros son corrimiento, incremento, adición, despeje y transferencia de datos. Algunas veces es conveniente adoptar una notación adecuada para describir las operaciones llevadas a cabo entre los registros.

En la Tabla 8-1 se dan ejemplos de notación simbólica para algunas operaciones de registro. Un registro se indica por una o más letras mayúsculas como A, B, o RA. Las celdas individuales o flip-flops dentro de un registro de n-bit se numeran en secuencia desde 1 a n o desde 0 a $n-1$. Un flip-flop sólo se considera como un registro de 1-bit. La transferencia de datos de un registro a otro se simboliza por una flecha dirigida que denota una transferencia de contenido de un registro fuente a un registro de destino. La operación de despejar registro se simboliza por una transferencia de 0 al registro. Un Flip-flop sólo puede establecerse en 1 o despejarse a 0. Para incrementar un registro en 1, es necesario que el registro sea capaz de contar hacia arriba como en un contador binario. La operación de decremento requiere un contador descendente. El contenido de dos registros puede agregarse mediante un circuito sumador. Algunas operaciones, como por ejemplo la operación de corrimiento, no tienen símbolo conocido. En tal caso se usan las palabras "corrimiento a la derecha R" para denotar un corrimiento a la derecha del registro R.

TABLA 8-1 Notación simbólica para las operaciones de registro.

Notación simbólica	Descripción
	Transferencia del contenido del registro
$A \leftarrow B$	B al registro A
$R \leftarrow 0$	Despejar el registro R
$F \leftarrow 1$	Establecer el flip-flop en 1
$A \leftarrow A + 1$	Incrementar el registro A en 1 (cuenta arriba)
$A \leftarrow A - 1$	Disminuir el registro A en 1 (cuenta abajo)
$A \leftarrow A + B$	Agregar el contenido del registro B al registro A

8-3 CONSIDERACIONES DE TEMPORIZADO

El temporizado de todos los registros y flip-flops en un sistema digital se controla por un reloj generador maestro. Los pulsos de reloj se aplican no sólo a los registros de la subsección del procesador de datos, sino también a todos los flip-flops en el circuito lógico. Las entradas también están sincronizadas con los pulsos de reloj porque normalmente se generan como salidas de otro circuito que usa las mismas señales del reloj. Si la señal de entrada cambia en un tiempo arbitrario independientemente del reloj, se le llama una entrada asíncrona. Las entradas asíncronas pueden causar una variedad de problemas, como se expone en el Capítulo 9. Para simplificar el diseño, se supone que todas las entradas están sincronizadas con el reloj y que cambian de estado como respuesta a una transición de borde del pulso de reloj. En forma similar, cualquier salida que es una función del estado presente y una entrada síncrona también estarán sincronizadas.

La mayor diferencia entre un diagrama de flujo convencional y un diagrama ASM está en la interpretación de las relaciones de tiempo entre las diversas operaciones. Por ejemplo, si la Fig. 8-6 fuera un diagrama de flujo convencional, entonces se consideraría que las operaciones listadas siguen una después de otra en secuencia de tiempo: el registro A se incrementa primero y sólo entonces E se evalúa. Si $E = 1$, entonces el registro R se despeja y el control pasa al estado T_4. En otra forma, si $E = 0$, el paso siguiente es evaluar F y pasar al estado T_2 o T_3. En contraste, un diagrama ASM considera el bloque entero como una unidad. Todas las operaciones que están especificadas dentro del bloque deben ocurrir en sincronismo durante la transición en borde del mismo pulso de reloj, mientras el sistema cambia desde T_1 al estado siguiente. Esto se presenta en forma gráfica en la Fig. 8-8. Se supone disparo de borde positivo para todos los flip-flops. La primera transición positiva del reloj transfiere el circuito de control al estado T_1. Mientras se encuentra en el estado T_1, los circuitos de control verifican las entradas E y F para generar las señales apropiadas en acuerdo. Las operaciones siguientes ocurren en forma simultánea durante la siguiente transición positiva del pulso de reloj:

1. El registro A se incrementa.
2. Si $E = 1$, el registro R se despeja.
3. Dependiendo de los valores de E y F, el control se transfiere al estado siguiente, T_2 o T_3 o T_4.

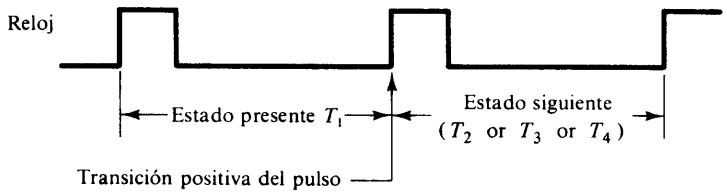


Figura 8-8 Transición entre estados.

Obsérvese que las operaciones en la subsección del procesador de datos y el cambio de estado en el control lógico ocurren al mismo tiempo.

Ahora se demostrará la relación de tiempo entre los componentes de un diagrama ASM al pasar a un ejemplo específico de diseño. El ejemplo no tiene ninguna aplicación conocida y sólo se formula para mostrar la utilidad del diagrama ASM. Se principia desde las especificaciones iniciales y se procede con el desarrollo de un diagrama ASM apropiado mediante el cual puede derivarse el hardware digital.

Ejemplo de diseño

Se desea diseñar un sistema digital con dos flip-flops E y F y un contador A binario de 4-bit. Los flip-flops individuales en A se denotan por A_4, A_3, A_2 y A_1, con A_4 que mantiene el bit más significativo de la cuenta. Una señal de inicio S pone en operación el sistema despejando el contador A y el flip-flop E. Entonces el contador se aumenta en 1 principiando desde el siguiente pulso de reloj y continúa incrementando hasta que la operación se detiene. Los contadores de bits A_3 y A_4 determinan la secuencia de operaciones:

Si $A_3 = 0$, E se despeja a 0 y continúa el conteo.

Si $A_3 = 1$, E se ajusta en 1; entonces si $A_4 = 0$, el conteo continúa, pero si $A_4 = 1$, F se ajusta en 1 en el siguiente pulso de reloj y el sistema detiene el conteo.

Diagrama ASM

El diagrama ASM se muestra en la Fig. 8-9. Cuando no se realizan operaciones, el sistema está en el estado inicial T_0, esperando la señal de inicio S. Cuando la entrada S es igual a 1, el contador A y el flip-flop F se despejan a 0 y el controlador pasa al estado T_1. Obsérvese la casilla condicional que sigue a la casilla de decisión para S. Esto significa que el contador y el flip-flop se despejarán durante T_0 si $S = 1$; y al mismo tiempo, el control se transfiere al estado T_1. El bloque asociado con el estado T_1 tiene dos casillas de decisión y dos casillas condicionales. El contador se incrementa con cada pulso de reloj. Al mismo tiempo, una de tres operaciones posibles ocurre durante la transición del mismo pulso de reloj:

Ya sea que E se despeje y el control permanezca en el estado T_1 ($A_3 = 0$);

o E se ajusta y el control permanece en el estado T_1($A_3A_4 = 10$);

o E se ajusta y el control pasa al estado T_2($A_3A_4 = 11$).

Cuando el control está en el estado T_2, el flip-flop F se ajusta en 1 y el circuito retorna a su estado inicial, T_0.

Los diagramas ASM constan de tres estados y tres bloques. El bloque asociado con T_0 consta de la casilla de estado, una casilla de decisión y una casilla condicional. El bloque asociado con T_2 consta sólo de la casilla de estado. El control lógico tiene una salida externa, S y dos entradas de estado, A_3 y A_4.

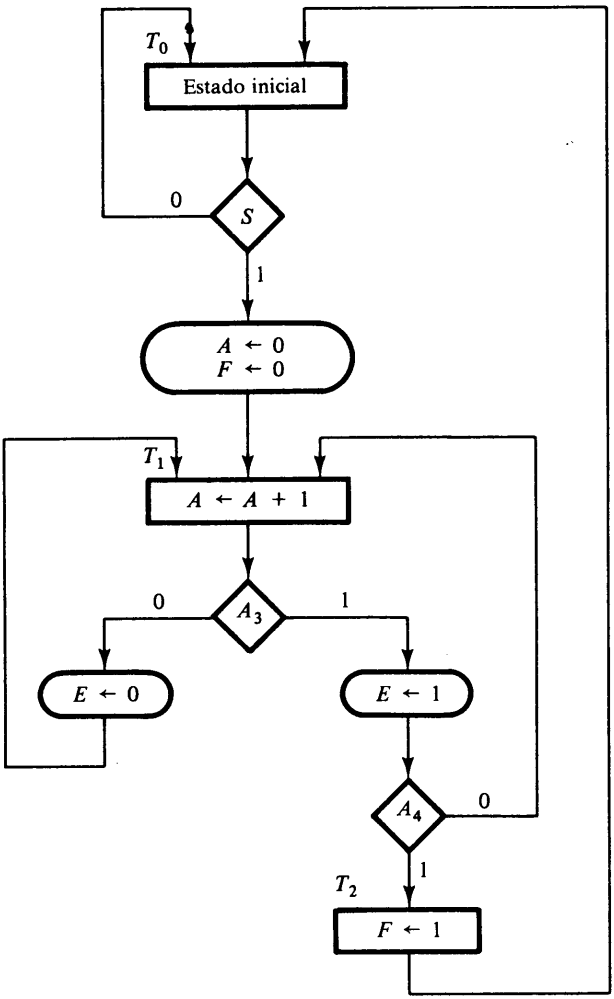


Figura 8-9 Carta ASM para ejemplo de diseño.

Secuencia de temporizado

Cada bloque en un diagrama ASM especifica las operaciones que van a realizarse durante un pulso común de reloj. Las operaciones especificadas dentro de las casillas de estado y condicional en el bloque se llevan a cabo en la subsección del procesador de datos. El cambio de un estado al siguiente se lleva a cabo en el control lógico. Con objeto de apreciar la relación de tiempo implicada, se lista la secuencia de paso por paso de las operaciones después de cada pulso de reloj desde el tiempo en que ocurre la señal de inicio hasta que el sistema regresa a su estado inicial.

En la Tabla 8-2 se muestran los valores binarios del contador y los dos flip-flops después de cada pulso de reloj. En la tabla también se muestran por separado los

estados de A_3 y A_4, al igual que el estado presente en el controlador. Se principia con el estado T_1 precisamente después de que la señal de entrada S ha causado que el contador y el flip-flops F se despejen. El valor de E se supone que es 1, porque E es igual a 1 al T_0 (como se muestra al final de la tabla) y porque E no cambia durante la transición desde T_0 hasta T_1. El sistema permanece en el estado T_1 durante los siguientes trece pulsos de reloj. Cada pulso incrementa el contador y despeja o bien ajusta E. Obsérvese la relación entre el tiempo al cual E_3 llega a ser un 1 y el tiempo al cual E se ajusta en 1. Cuando $A = 0011$, el siguiente pulso de reloj incrementa el contador a 0100, pero el mismo pulso de reloj ve el valor de A_3 como 0, de modo que E se despeja. El siguiente pulso cambia el contador desde 0100 a 0101 y, ahora, A_3 es inicialmente igual a 1, de modo que E se ajusta en 1. En forma similar, E se despeja a 0 no cuando la cuenta pasa desde 0011 a 1000, sino cuando pasa desde 1000 a 1001, lo cual es cuando A_3 es 0 en el valor presente del contador.

Cuando la cuenta alcanza 1100, tanto A_3 como A_4 son iguales a 1. El siguiente pulso de reloj incrementa A en 1, ajusta E en 1, y transfiere el control al estado T_2. El control permanece en T_2 sólo por un periodo del reloj. La transición de pulso asociada con T_2 ajusta el flip-flop F en 1 y transfiere el control al estado T_0. El sistema permanece en el estado inicial T_0 en tanto que S sea igual a 0.

Por la observación de la Tabla 8-2 puede parecer que las operaciones realizadas en E se retardan por un pulso de reloj. Esta es la diferencia entre un diagrama ASM y un diagrama de flujo convencional. Si la Fig. 8-9 fuera un diagrama de flujo convencional, podría suponerse que A se incrementa primero y que el valor incrementado podría haberse utilizado para verificar el estado de A_3. Las operaciones que se realizan

TABLA 8-2 Secuencia de las operaciones en el diseño de ejemplo

Contador				Flip-flops			
A_4	A_3	A_2	A_1	E	F	Condiciones	Estado
0	0	0	0	1	0	$A_3 = 0, A_4 = 0$	T_1
0	0	0	1	0	0		
0	0	1	0	0	0		
0	0	1	1	0	0		
0	1	0	0	0	0	$A_3 = 1, A_4 = 0$	
0	1	0	1	1	0		
0	1	1	0	1	0		
0	1	1	1	1	0		
1	0	0	0	1	0	$A_3 = 0, A_4 = 1$	
1	0	0	1	0	0		
1	0	1	0	0	0		
1	0	1	1	0	0		
1	1	0	0	0	0	$A_3 = 1, A_4 = 1$	
1	1	0	1	1	0		T_2
1	1	0	1	1	1		T_0

en el hardware digital como se especifica por un bloque en el diagrama ASM ocurren durante el mismo periodo de reloj y no en una secuencia de operaciones que siguen una a otra en el tiempo, como se interpreta por lo común en un diagrama de flujo convencional. Por lo tanto, el valor de A_3 que se considera en la casilla de decisión se toma del valor del contador en el estado presente y antes de que se incremente. Esto es porque la casilla de decisión para E pertenece al mismo bloque como en el estado T_1. Los circuitos digitales en el control generan las señales para todas las operaciones especificadas en el bloque presente antes de la llegada del siguiente pulso de reloj. La transición siguiente del reloj ejecuta todas las operaciones en los registros y flip-flops, incluyendo los flip-flops en el controlador que determina el estado siguiente.

Procesador de datos

El diagrama ASM da toda la información necesaria para diseñar el sistema digital. Los requisitos para el diseño del subsistema procesador de datos se especifican dentro de las casillas de estado y condicionales. El control lógico se determina mediante las casillas de decisión y las transiciones de estado requeridas. Un diagrama que muestra el hardware para el ejemplo de diseño se muestra en la Fig. 8-10. El subsistema de control se ilustra sólo con sus entradas y salidas. El diseño detallado de control se considera en la siguiente sección. El procesador de datos consta de un contador binario de 4-bit, dos flip-flops y un número de compuertas. El contador es similar al que se muestra en la Fig. 7-17 excepto que se requieren compuertas adicionales para la operación síncrona de despeje. El contador se incrementa con cada pulso de reloj cuando el control está en el estado T_1. Se despeja sólo cuando el control se encuentra en el estado T_0 y S es igual a 1. Esta operación condicional requiere una compuerta AND para garantizar que ambas condiciones estén presentes. Las otras dos operaciones condicionales usan otras dos compuertas AND para ajustar despejar el flip-flop E. El flip-flop F se ajusta incondicionalmente durante el estado T_2. Obsérvese que todos los flip-flops y registros, incluyendo los flip-flops en el control, utilizan una fuente común de pulso de reloj.

Este ejemplo demuestra un método de diseño digital empleando el diagrama ASM. El diseño del subsistema procesador de datos requiere una interpretación de las operaciones de registros y su implementación mediante los componentes que se expusieron en los Capítulos 5 y 7, como por ejemplo registros, contadores, multiplexores y adicionadores.El diseño del subsistema de control requiere la aplicación de procedimientos de diseño basados en la teoría de la lógica secuencial. Las siguientes tres secciones presentan algunas de las alternativas que están disponibles para diseñar el control lógico.

8-4 IMPLANTACION DEL CONTROL

La sección de control de un sistema digital es en esencia un circuito secuencial que puede diseñarse por el procedimiento delineado en el Capítulo 6. Sin embargo, en la mayor parte de los casos este método no es práctico debido al gran número de estados

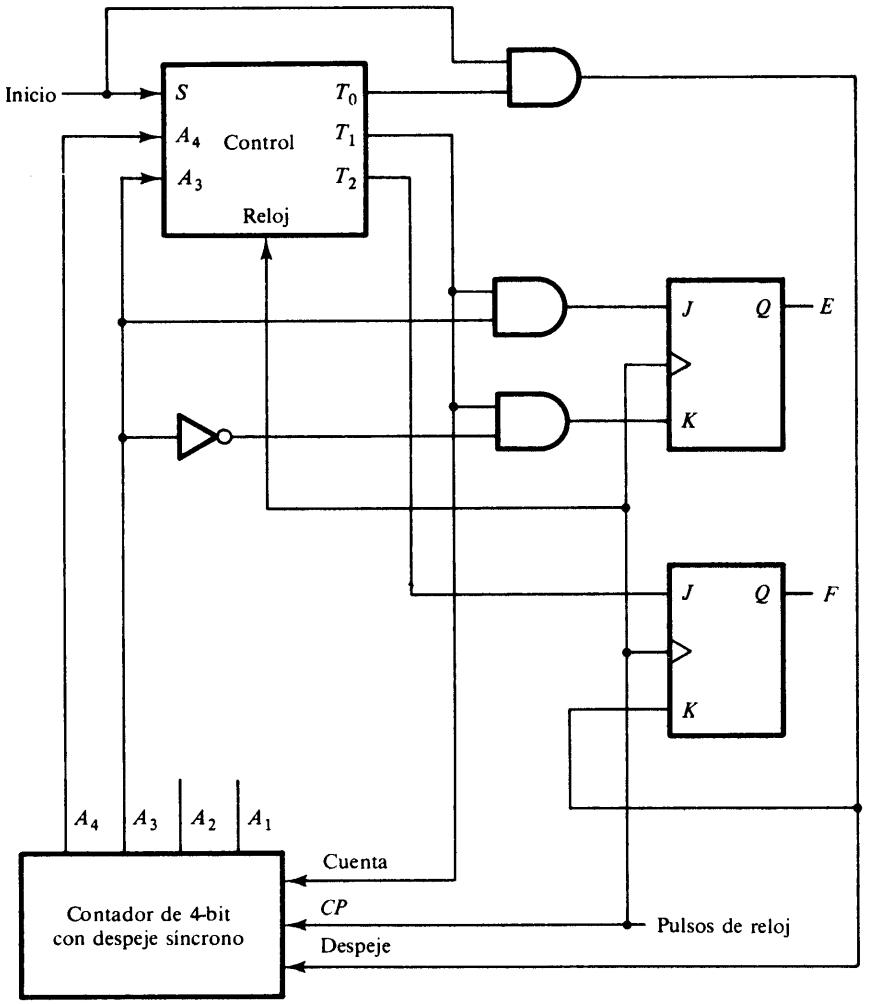


Figura 8-10 Procesador de datos para el ejemplo de diseño.

y salidas que puede tener un circuito de control típico. Excepto por controladores muy simples, el método de diseño que usa tablas de estado y exitación es engorroso y difícil de manejar. Los diseñadores experimentados de sistemas digitales usan métodos especializados para el diseño del control lógico que pueden considerarse como una extensión del método secuencial clásico combinado con otras suposiciones de simplificación. Dos de estos métodos especializados se presentan en esta sección y un tercer método se explica en la Sección 8-5. Otra alternativa es utilizar una ROM o un PLA para diseñar el control lógico. Esto se cubre en la Sección 8-6.

Tabla de estado

Como se mencionó con anterioridad, el diagrama ASM se asemeja a un diagrama de estado con cada casilla de estado representando un estado. El diagrama de estado puede convertirse en una tabla de estado mediante la cual puede diseñarse el circuito secuencial del controlador. Primero deben asignarse valores binarios a cada estado en el diagrama ASM. Para n flip-flops en el circuito secuencial de control, el diagrama ASM puede acomodar hasta 2^n estados. Un diagrama con tres o cuatro estados requiere un circuito secuencial con dos flip-flops. Con cinco a ocho estados, se necesitan tres flip-flops. Cada combinación de valores flip-flop representa un número binario para uno de los estados.

Una tabla de estados para un controlador es una lista de los estados y entrada spresentes y sus correspondientes estados siguientes y salidas. En la mayoría de los casos hay muchas condiciones no importa de entrada que deben incluirse, de modo que es aconsejable arreglar la tabla de estado para tomar esto en consideración. Con objeto de aclarar el procedimiento, se ilustrará por la obtención de la tabla de estado del controlador definido en el ejemplo de la sección anterior.

El diagrama ASM del diseño de ejemplo se muestra en la Fig. 8-9. Se asignan los siguientes valores binarios a los tres estados: $T_0 = 00$, $T_1 = 01$, $T_2 = 11$. El estado binario 10 no se utiliza y se tratará como una condición no importa. La tabla de estado correspondiente al diagrama ASM se muestra en la Tabla 8-3. Son necesarios dos flip-flops, y se etiquetan G_1 y G_2. Hay tres entradas y tres salidas. Las entradas se toman mediante las condiciones en las casillas de decisión. Las salidas son equivalentes al estado presente de control. Obsérvese que hay un renglón en la tabla para cada transición posible entre estados. El estado inicial 00 pasa al estado 01 o permanece en 00 dependiendo del valor de la entrada S. Las otras dos entradas se marcan con X sin preocupación, ya que no determinan en este caso el estado siguiente. En tanto el sistema está en el estado binario 00, el control proporciona una salida etiquetada T_0 para iniciar las operaciones requeridas de registro. La transición del estado binario 01 depende de las entradas A_3 y A_4. El sistema pasa al estado binario 11 sólo si $A_3A_4 = 11$; en otra forma, permanece en el estado binario 01. Al final, el estado binario 11 pasa a 00 en forma independiente de las variables de entrada.

TABLA 8-3 Tabla de estados para control en la Fig. 8-10

Símbolo del estado presente	Estado presente		Entradas			Estado siguiente		Salidas		
	G_1	G_2	S	A_3	A_4	G_1	G_2	T_0	T_1	T_2
T_0	0	0	0	X	X	0	0	1	0	0
T_0	0	0	1	X	X	0	1	1	0	0
T_1	0	1	X	0	X	0	1	0	1	0
T_1	0	1	X	1	0	0	1	0	1	0
T_1	0	1	X	1	1	1	1	0	1	0
T_2	1	1	X	X	X	0	0	0	0	1

Este ejemplo demuestra una tabla de estado para un controlador secuencial. Obsérvese otra vez el gran número de condiciones no importa bajo las entradas. El número de renglones en la tabla de estado es igual al número de trayectorias distintas entre los estados en el diagrama ASM.

Diagrama lógico con flip-flops *JK*

El procedimiento para diseñar un circuito secuencial principiando desde una tabla de estado se presenta en la Sección 6-7. Este procedimiento requiere que se obtenga la tabla de excitación de las entradas flip-flop y entonces se simplifica la parte del circuito combinacional del circuito secuencial. Si se aplica este procedimiento a la Tabla 8-3, se necesita utilizar mapas de cinco variables (véase la Fig. 3-11) para simplificar las funciones de entrada. Esto es porque hay cinco variables listadas bajo las columnas de "estado presente" y "entrada". Ya que este procedimiento se explicó en el Capítulo 6, no se mostrará aquí el trabajo de detalle. Las funciones de entrada flip-flop obtenidas por este método, si se suponen flip-flops *JK*, son:

$$JG_1 = G_2A_3A_4 \qquad JG_2 = S$$
$$KG_1 = 1 \qquad KG_2 = G_1$$

Para derivar las tres funciones de salida, se utiliza el hecho de que el estado binario 10 no se usa y se obtienen las siguientes funciones simplificadas:

$$T_0 = G_2'$$
$$T_1 = G_1'G_2$$
$$T_2 = G_1$$

El diagrama lógico del control se muestra en la Fig. 8-11. Este circuito reemplaza el bloque control en la Fig. 8-10.

Flip-flops *D* y decodificador

Cuando el número de flip-flops más entradas en una tabla de estado es mayor de cinco, es necesario usar grandes mapas para simplificar las funciones de entrada. Esto es engorroso y difícil de lograr, como se explicó en el Capítulo 3. En consecuencia es necesario encontrar vías alternas para diseñar controladores excepto cuando son muy simples. Una posibilidad es usar flip-flips tipo *D* y obtener las funciones de entrada directamente mediante la tabla de estado sin que sea necesaria una tabla de excitación. Esto se debe a que el estado siguiente es el mismo que los requisitos de entrada para los flip-flops *D* (véase la Sección 6-9). Para diseñar el circuito secuencial con flip-flops *D*, es necesario pasar a la siguiente columna de estado en la tabla de estado y derivar todas las condiciones que deben establecer en 1 a cada flip-flop. Mediante la Tabla 8-3 se observa que la siguiente columna de estado de G_1 tiene un solo 1 en el quinto renglón. La entrada T de flip-flop G_1 debe ser igual a 1 durante el estado presente $T_1 = G_1'G_2$

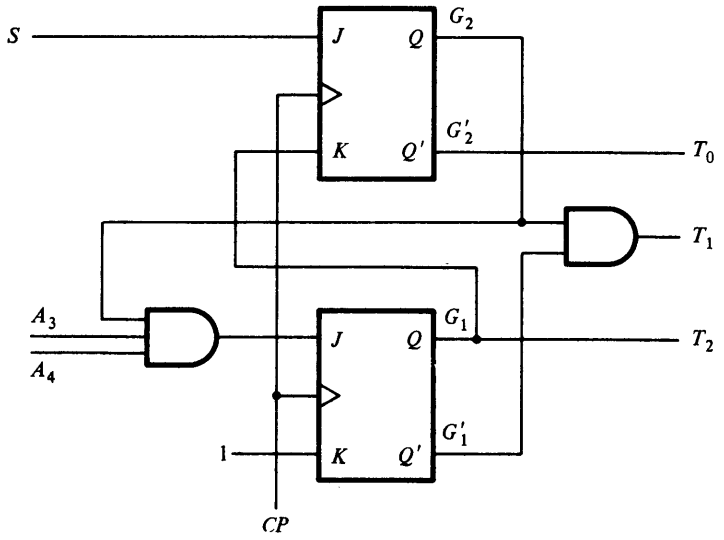


Figura 8-11 Diagrama lógico de control con uso de flip-flops *JK*.

cuando tanto la entrada A_3 como la A_4 son iguales a 1. Esto se expresa con la función de entrada del flip-flop *D*:

$$DG_1 = G'_1G_2A_3A_4$$

En forma similar, la siguiente columna de estado de G_2 tiene cuatro número 1 y la condición para ajustar este flip-flop es:

$$DG_2 = G'_1G'_2S + G'_1G_2$$

Puede avanzarse un paso más e insertar un decodificador a la salida de los flip-flops para obtener las tres salidas necesarias, T_0, T_1 y T_2. Entonces, en lugar de usar las salidas de flip-flop como la condición de estado presente, también pueden usarse las salidas del decodificador para suministrar esta información. Las funciones de entrada a los flip-flops *D* pueden expresarse ahora como sigue:

$$DG_1 = A_3A_4T_1$$
$$DG_2 = ST_0 + T_1$$

El diagrama lógico alterno se muestra en la Fig. 8-12. El decodificador proporciona las tres salidas de control, y esas salidas también se usan para determinar el estado siguiente de cada flip-flop. El segundo circuito de control requiere más componentes que el primero, pero tiene la ventaja de que puede derivarse por inspección mediante la tabla de estado.

Un flip-flop por estado

Otro método posible de diseño del control lógico es usar un flip-flop por estado en el circuito secuencial. Sólo un flip-flop se establece en cualquier momento particular; todos los otros se despejan a 0. El único bit se hace para propagarse de un flip-flop al otro bajo el control de lógica de decisión. En dicho arreglo, cada flip-flop representa un estado que es activado sólo cuando el bit de control se transfiere a él.

Es obvio que en este método no se usa un número mínimo de flip-flops para circuito secuencial. De hecho, se utiliza un número máximo de flip-flops. Por ejemplo, un circuito secuencial con 12 estados requiere un mínimo de cuatro flip-flops. Sin embargo, por este método el circuito necesita 12 flip-flops, uno para cada estado.

Una organización de control que utiliza un flip-flop por estado tiene la característica conveniente de que el circuito puede derivarse de manera directa mediante el diagrama de estado sin necesidad de tablas de estado por excitación. Considérese, por ejemplo, el diagrama de estado de la Fig. 8-13. Este diagrama es equivalente al diagrama ASM del ejemplo de diseño en la Fig. 8-9 en lo que respecta a las transiciones de control de estado. Ya que el diagrama tiene tres estados, se asignan tres flip-flops al circuito y se etiquetan T_0, T_1 y T_2. El controlador puede diseñarse por inspección del diagrama de estado si se usan flip-flops tipo D. La función booleana para ajustar el flip-flop se determina mediante el estado presente y las condiciones de entrada a lo largo de las líneas dirigidas. Por ejemplo, el flip-flop T_0 se ajusta con el siguiente pulso de reloj si el estado presente $T_2 = 1$ o si el estado presente $T_0 = 1$ y la entrada $S = 0$. Esta condición se define por la función de entrada flip-flop:

$$DT_0 = T_2 + S'T_0$$

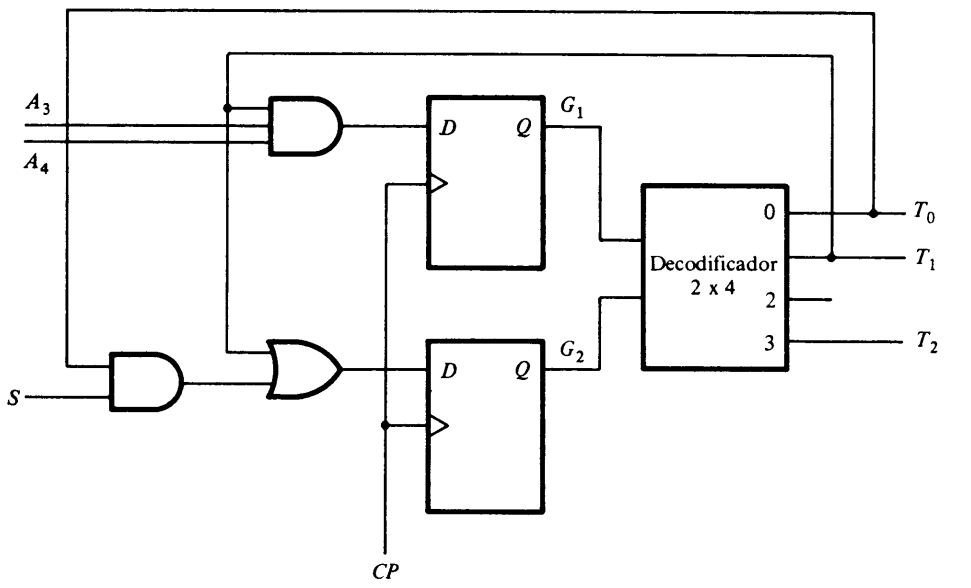


Figura 8-12 Diagrama lógico alterno de control con uso de flip-flops D y un decodificador.

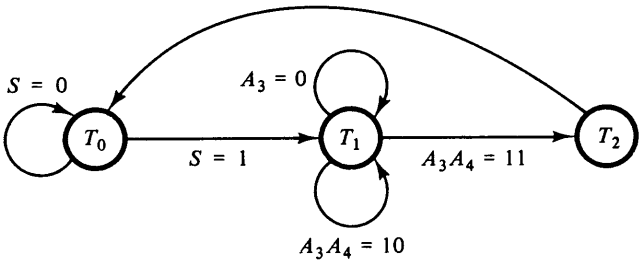


Figura 8-13 Diagrama de estado de un controlador.

en donde DT_0 denota la entrada D del flip-flop T_0. De hecho, la condición para ajustar un flip-flop en 1 se obtiene en forma directa del diagrama de estado por la condición especificada en las líneas dirigidas, que van al estado correspondiente flip-flop combinado en lógica AND con el estado previo flip-flop. Si hay más de una línea dirigida que va a un estado, todas las condiciones deben combinarse en lógica OR. Por el uso de este procedimiento para los otros dos flip-flops, se obtienen las funciones de entrada:

$$DT_1 = ST_0 + A_3'T_1 + A_3A_4'T_1 = ST_0 + (A_3A_4)'T_1$$
$$DT_2 = A_3A_4T_1$$

El diagrama lógico se muestra en la Fig. 8-14. Consta de tres flip-flops tipo D: T_0, T_1 y T_2, y las compuertas asociadas especificadas por las funciones de entrada listadas arriba.

Al inicio, el flip-flop T_0 debe ajustarse en 1 y todos los otros flip-flops se despejan a 0 de modo que el flip-flop que representa el estado inicial es igual a 1 y todos los otros estados son iguales a 0. Una vez iniciado, el flip-flop controlador uno por estado se propagará por sí mismo de estado a estado en la forma apropiada. Para un registro con una entrada común asíncrona de despeje, como se muestra en la Fig. 8-14, todos los flip-flops incluyendo la salida Q de T_0 se despejan a 0. Tomando la salida de T_0 de la salida complemento Q' proporciona la señal inicial requerida de 1 para T_0. Con objeto de mantener Q' como la salida de T_0, es necesario que la función de entrada de la entrada D se complemente. Esto se hace por el inversor adicional que se coloca a la entrada D del flip-flop T_0.

8-5 DISEÑO CON MULTIPLEXORES

Un importante objetivo del diseño de control lógico es el desarrollo de un circuito que implemente la secuencia de control deseada en una forma lógica y directa. El intento de minimizar el número de compuertas tiende a producir un circuito irregular, lo que hace difícil para cualquiera, excepto para el diseñador, identificar la secuencia de eventos que emprende el control. Como consecuencia, es difícil alterar, dar servicio o mantener el equipo después del diseño inicial. La secuencia de estados en el control debe ser evidente en forma clara de la configuración del circuito aun si esto requiere componentes adicionales y resulta en un circuito con número no mínimo de componentes. Una implantación tal es el método de diseño con multiplexores.

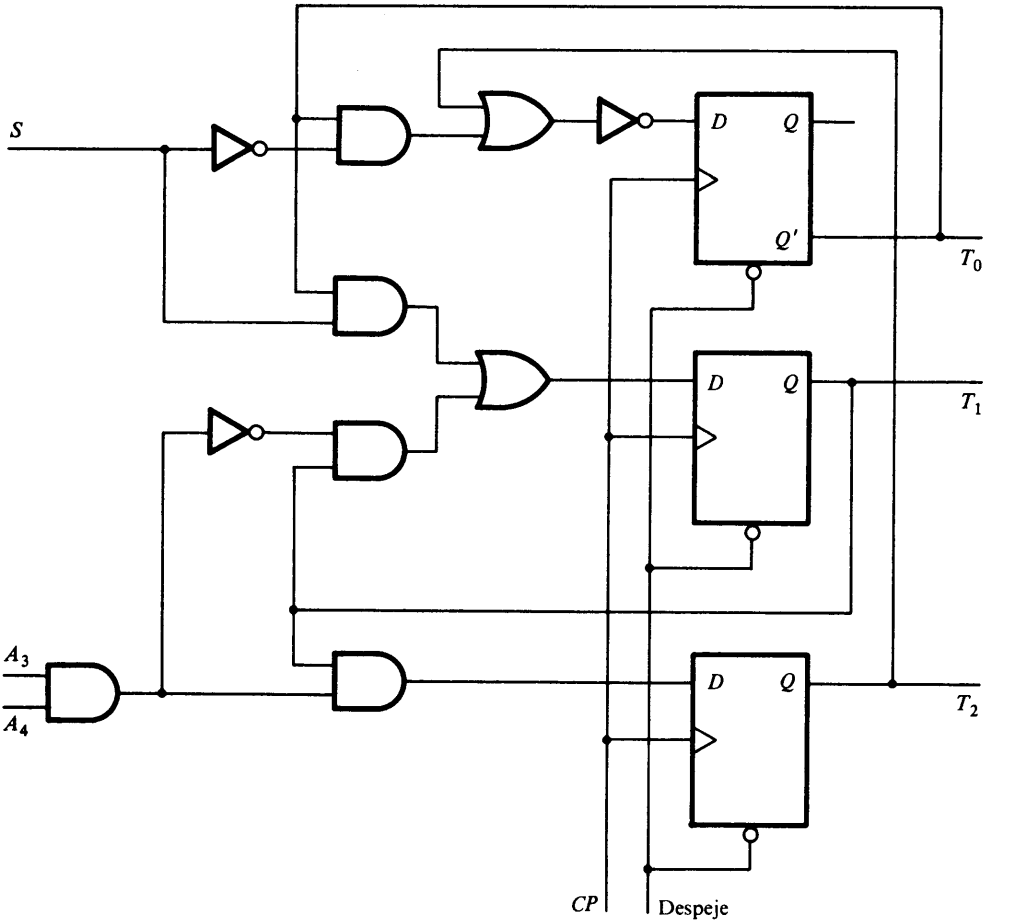


Figura 8-14 Diagrama de una tercera alternativa de control con uso de un flip-flop por estado.

El circuito de control que se muestra en la Fig. 8-12 consta de tres componentes: los flip-flops que mantienen el valor binario de estado, el decoficador que genera las salidas de control, y las compuertas que determinan el estado siguiente. Ahora se reemplazan las compuertas con multiplexores y se usa un registro para los flip-flops individuales. Este método de diseño da por resultado un patrón regular de tres niveles de componentes. El primer nivel consta de multiplexores que determinan el estado siguiente del registro. El segundo nivel contiene un registro que mantiene el estado binario presente. El tercer nivel tiene el decodificador que proporciona una salida separada para cada estado de control.

Por ejemplo, considérese el diagrama ASM en la Fig. 8-15. Consta de cuatro estados y cuatro entradas de control. Las casillas de estado se dejan vacías en este caso porque se tiene interés sólo en la secuencia de control, la cual es independiente de las operaciones del registro. La asignación binaria para cada estado se indica en la

esquina superior derecha de las casillas de estado. Las casillas de decisión especifican las transiciones de estado como una función de las cuatro entradas de control w, x, y y z. La implementación del control de tres niveles se muestra en la Fig. 8-16. Consta de dos multiplexores MUX1 y MUX2, un registro con dos flip-flops G_1 y G_2 y un decodificador con cuatro salidas. Las salidas del registro se aplican a las entradas del decodificador y también a las entradas seleccionadas de los multiplexores. En esta forma, el estado presente del registro se usa para seleccionar una de las entradas para cada multiplexor. Las salidas de los multiplexores se aplican entonces a las entradas D de G_1 y G_2. El propósito de cada multiplexor es producir una entrada a su flip-flop correspondiente igual al valor binario del estado siguiente.

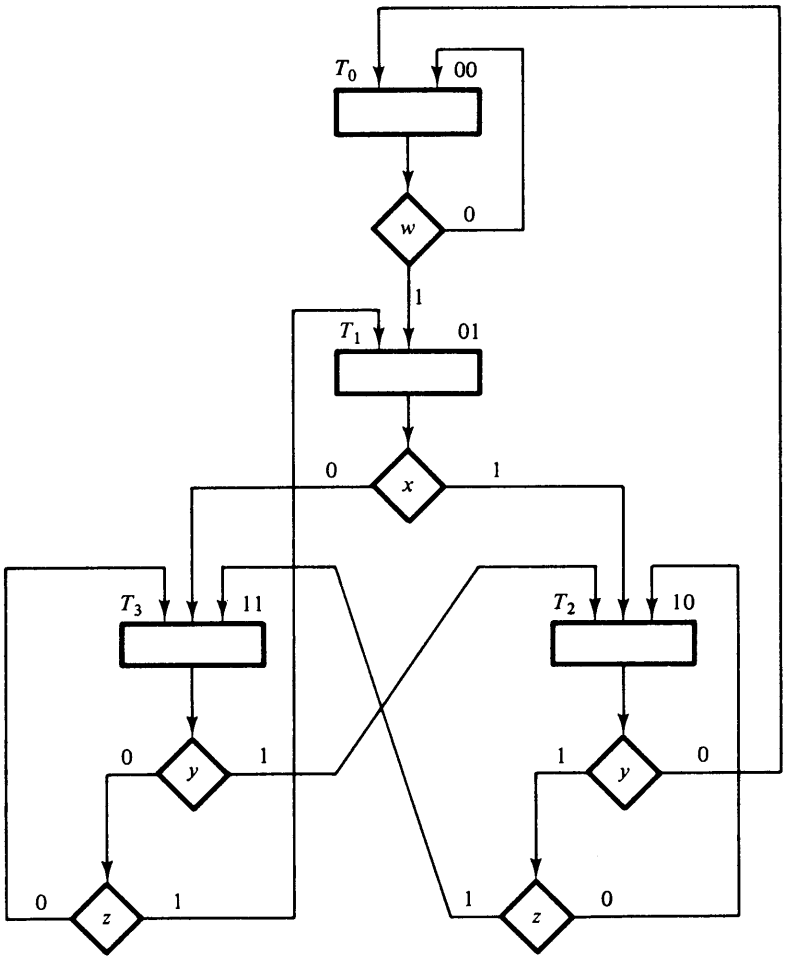


Figura 8-15 Ejemplo de una carta ASM con cuatro entradas de control.

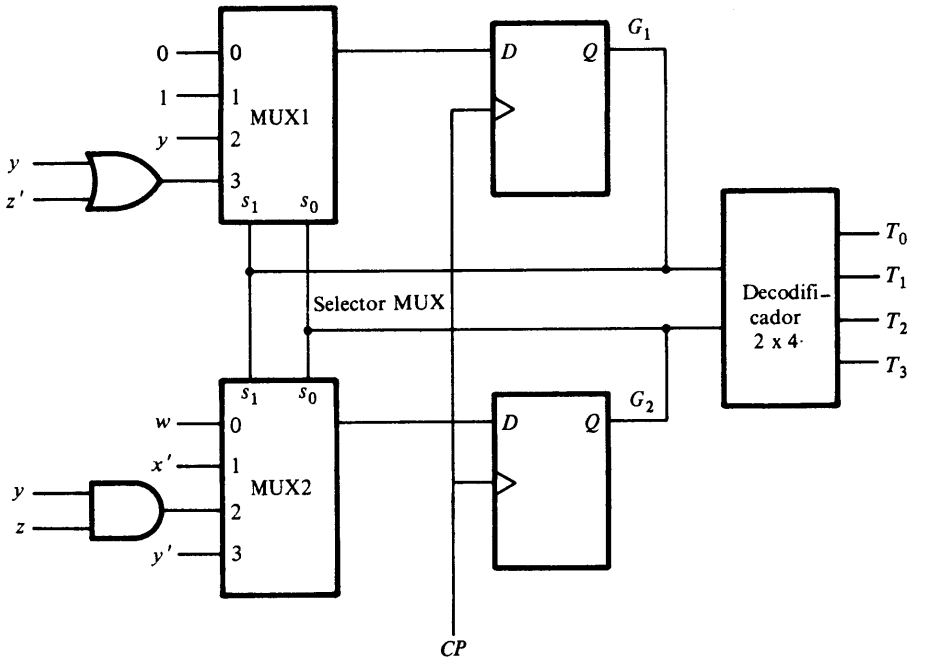


Figura 8-16 Implementación de un control con multiplexores.

Las entradas de los multiplexores se determinan mediante las casillas de decisión y las transiciones de estado dadas en el diagrama ASM. Por ejemplo, el estado 00 permanece en 00 o pasa a 01 dependiendo del valor de la entrada w. Ya que el estado siguiente de G_1 es 0 en cualquier caso, se coloca una señal equivalente a la lógica 0 en la entrada 0 del MUX1. El estado siguiente de G_2 es 0 si $w = 0$ y 1 si $w = 1$. Ya que el estado siguiente de G_2 es igual a w, se aplica el control de entrada w a la entrada 0 del MUX2. Esto significa que cuando las entradas seleccionadas de los multiplexores son iguales al estado presente 00, las salidas de los multiplexores proporcionan el valor binario que se transfiere al registro durante el siguiente pulso de reloj.

Para facilitar la evaluación de las entradas de multiplexor, se prepara una tabla mostrando las condiciones de entrada para cada transición posible en el diagrama ASM. En la Tabla 8-4 se da esta información para el diagrama ASM en la Fig. 8-15. Hay dos transiciones desde el estado presente 00 o 01 y tres transiciones desde el estado presente 10 o 11. Estas transiciones están separadas por líneas horizontales a través de la tabla. Las condiciones de entrada que se listan en la tabla se obtienen mediante las casillas de decisión en el diagrama ASM. Por ejemplo, en la Fig. 8-15 se observa que el estado presente 01 pasará al estado siguiente 10 si $x = 1$, o al estado siguiente 11 si $x = 0$. En la tabla se marcan estas condiciones de entrada como x y x', respectivamente. Las dos columnas bajo "entradas multiplexoras" en la tabla especifican los valores de entrada que deben aplicarse a MUX1 y MUX2. La entrada de multiplexor para cada estado presente se determina mediante las condiciones de entrada cuando el estado

siguiente del flip-flop es igual a 1. Por tanto, después del estado presente 01, el estado siguiente de G_1 siempre es igual a 1 y el estado siguiente de G_2 es igual al valor complemento de x. Así que, la entrada de MUX1 se hace igual a 1 y la de MUX2 a x' cuando el estado presente del registro es 01. Como otro ejemplo, después del estado presente 10, el estado siguiente de G_1 debe ser igual a 1 si las condiciones de entrada son yz' o yz. Cuando estos dos términos booleanos se combinan juntos en lógica OR y se simplifican entonces, se obtiene la sola variable binaria y, como se indica en la tabla. El estado siguiente de G_2 es igual a 1 si las condiciones de entrada son $yz = 11$. Si el estado siguiente de G_1 permanece en 0 después de un estado presente dado, se coloca un 0 en la entrada del multiplexor como se muestra en el estado presente 00 para el MUX1. Si el estado siguiente de G_1 siempre es 1, se coloca un 1 en la entrada del multiplexor como se muestra en el estado presente 01 para el MUX1. Las otras anotaciones para MUX1 y MUX2 se derivan de manera semejante. Las entradas a los multiplexores mediante la tabla se usan entonces en la implementación del control en la Fig. 8-16. Obsérvese que si el estado siguiente de un flip-flop es una función de dos o más variables de control, el multiplexor puede requerir una o más compuertas en su entrada. De otra forma, la entrada del multiplexor es igual a la variable de control, o el complemento de la variable de control, o 0 o 1.

Ejemplo de diseño

Se demostrará la implementación de un multiplexor de control mediante un segundo ejemplo de diseño. En el ejemplo también se demostrará la formulación del diagrama ASM y la implementación del subsistema procesador de datos.

El sistema digital que se diseñará consta de dos registros $R1$ y $R2$ y un flip-flop E. El sistema cuenta el número de los 1 en el número cargado dentro del registro $R1$ y

TABLA 8-4 Condiciones de entrada al multiplexor

Estado presente		Estado siguiente		Condiciones de entrada	Entradas al multiplexor	
G_1	G_2	G_1	G_2		MUX1	MUX2
0	0	0	0	w'		
0	0	0	1	w	0	w
0	1	1	0	x		
0	1	1	1	x'	1	x'
1	0	0	0	y'		
1	0	1	0	yz'		
1	0	1	1	yz	$yz' + yz = y$	yz
1	1	0	1	$y'z$		
1	1	1	0	y		
1	1	1	1	$y'z'$	$y + y'z' = y + z'$	$y'z + y'z' = y'$

establece el registro $R2$ en ese número. Por ejemplo, si el número binario que se carga dentro de $R1$ es 10111001, el circuito cuenta los cinco 1 en $R1$ y establece el registro $R2$ en la cuenta binaria 101. Esto se hace corriendo cada bit desde el registro $R1$ uno a la vez dentro del flip-flop E. El valor en E se verifica por el control, y cada vez que es igual a 1, el registro $R2$ se incrementa en 1.

El subsistema de control usa una entrada S externa para iniciar la operación y dos entradas de estados E y Z desde el procesador de datos. E es la salida del flip-flop. Z es la salida de un circuito que verifica el contenido del registro $R1$ para todos los 0. El circuito produce una salida $Z = 1$ cuando $R1$ es igual a 0.

El diagrama ASM para el ejemplo de diseño se muestra en la Fig. 8-17. El número binario se carga dentro de $R1$ y el registro $R2$ se ajusta en un valor por

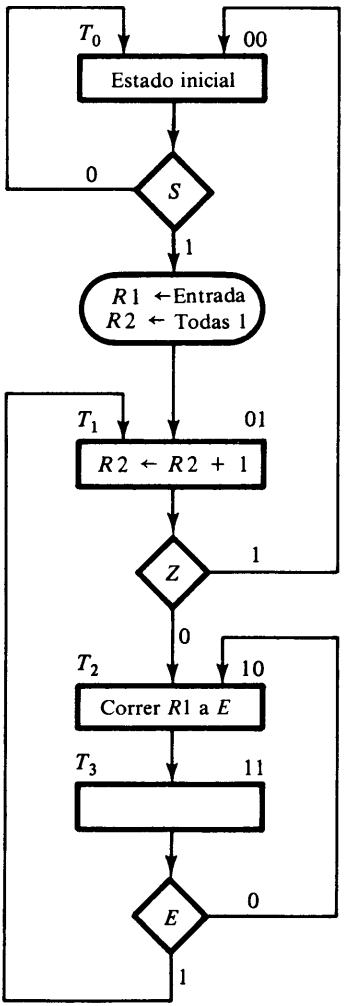


Figura 8-17 Carta ASM para el ejemplo de diseño.

completo en número 1. Obsérvese que un número con todos 1 en un registro cuando se incrementa produce un número por completo en 0. En el estado T_1, el registro $R2$ se incrementa si el contenido de $R1$ se examina. Si el contenido es 0, entonces $Z = 1$, y esto significa que no hay 1 almacenados en el registro; de modo que la operación termina con $R2$ igual a 0. Si el contenido de $R1$ no es cero, entonces $Z = 0$ y esto indica que hay algunos 1 almacenados en el registro. El número en $R1$ se corre y su bit de la extrema izquierda se transfiere dentro de E. Esto se hace cuantas veces sea necesario hasta que se transfiere un 1 dentro de E. Por cada 1 detectado en E, el registro $R2$ se incrementa y el registro $R1$ se verifica otra vez para buscar más números 1. El lazo mayor se repite hasta que todos los 1 en $R1$ están contados. Obsérvese que la casilla de estado de T_3 no tiene operaciones de registro, pero el bloque asociado con ella contiene la casilla de decisión para E. También obsérvese que la entrada serial al registro de corrimiento $R1$ debe ser igual a 0 porque no se desea correr 1 externos dentro de $R1$.

El subsistema procesador de datos se muestra en la Fig. 8-18. El control tiene tres entradas y cuatro salidas. Sólo se usan tres salidas por el procesador de datos. El registro $R1$ es un registro de corrimiento similar al que se ilustra en la Fig. 7-9. El registro $R2$ es un contador con carga paralela parecido al que se ilustra en la Fig. 7-19. Con objeto de no complicar el diagrama, no se muestran los pulsos de reloj, pero deben aplicarse a los dos registros, al flip-flop E y a los flip-flops en el control. El circuito que verifica para 0 es una compuerta NOR. Por ejemplo, si $R1$ es un registro de cuatro bits con salidas R_1, R_2, R_3, R_4, entonces Z se genera con la función booleana

$$Z = R_1' R_2' R_3' R_4' = (R_1 + R_2 + R_3 + R_4)'$$

la cual comprende las funciones NOR de todos los bits en el registro.

Las condiciones de entrada del multiplexor para el control se determinan mediante la Tabla 8-5. Las condiciones de entrada se obtienen mediante el diagrama ASM para cada posible transición de estado binario. La asignación binaria a cada estado está escrita en la esquina superior derecha de las casillas de estado. La transición del estado presente 00 depende de S, del estado presente 01 depende de Z y del estado presente de E. El estado presente 10 pasa al estado siguiente 11 de manera incondicional. Los valores bajo MUX1 y MUX2 en la tabla se determinan mediante las condiciones booleanas de entrada para el estado siguiente de G_1 y G_2, respectivamente.

La implementación de control del ejemplo de diseño se muestra en la Fig. 8-19. Esta es una implementación de tres niveles con los multiplexores en el primer nivel. Las entradas a los multiplexores se obtienen mediante la Tabla 8-5.

8-6 CONTROL PLA

Mediante los ejemplos que se presentaron en este capítulo se ha visto que el diseño de un circuito de control es en forma esencial un problema de lógica secuencial. En la Sección 7-2 se mostró que un circuito secuencial puede construirse mediante un registro conectado a un circuito combinacional. En la Sección 5-8 se investigó el arreglo lógico programable (PLA) y se mostró que puede utilizarse para implementar

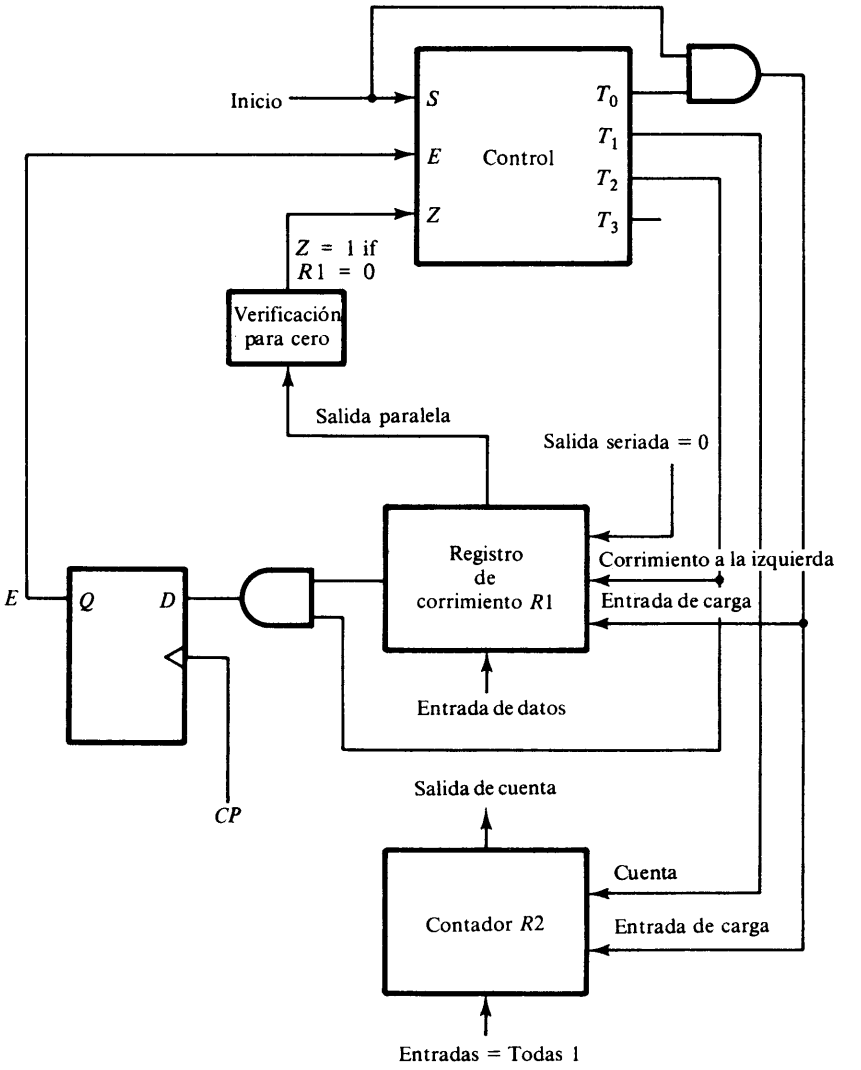


Figura 8-18 Subsistema procesador de datos para el ejemplo de diseño.

cualquier circuito combinacional. Ya que el control lógico es un circuito secuencial, entonces es posible diseñar el circuito de control con un registro conectado a un PLA. El diseño de un control PLA requiere que se obtenga la tabla de estado del circuito. El método PLA debe usarse si la tabla de estado contiene muchas anotaciones no importa en otra forma, puede ser ventajoso utilizar una ROM en lugar de un PLA.

El control PLA se desmotrará mediante un tercer ejemplo de diseño. Este ejemplo es un circuito que modifica dos números binarios sin signo y produce su producto binario.

TABLA 8-5 Condiciones de entrada al multiplexor del diseño de ejemplo

Estado presente		Estado siguiente		Condiciones de entrada	Entradas al multiplexor	
G_1	G_2	G_1	G_2		MUX1	MUX2
0	0	0	0	S'		
0	0	0	1	S	0	S
0	1	0	0	Z		
0	1	1	0	Z'	Z'	0
1	0	1	1	Ninguna	1	1
1	1	1	0	E'		
1	1	0	1	E	E'	E

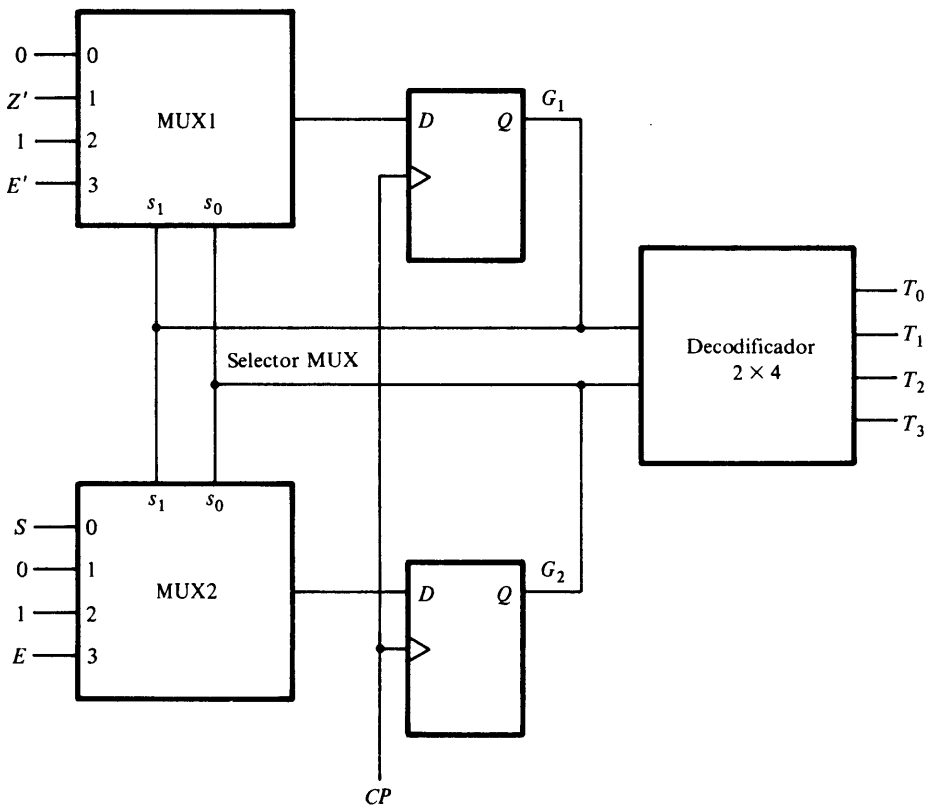


Figura 8-19 Implementación de control para ejemplo de diseño.

Multiplicador binario

La multiplicación de dos números binarios se hace con papel y lápiz por adiciones sucesivas y corrimientos. Este proceso se ilustra de mejor manera con un ejemplo numérico. Permítase multiplicar los dos números binario 10111 y 10011.

23	10111	multiplicando
19	10011	multiplicador
	10111	
	10111	
	00000	
	00000	
	10111	
437	110110101	producto

El proceso consiste de observar los bits sucesivos del multiplicador, primero el bit menos significativo. Si el bit de multiplicador es un 1, el multiplicando se copia abajo. En otra forma, los 0 se copian abajo. Los números que se copian abajo en líneas sucesivas se corren una posición a la izquierda del número previo. Por último, los números se adicionan y sus sumas forman el producto. Obsérvese que el producto obtenido mediante la multiplicación de dos números binarios de n bits cada uno puede ser hasta 2^n de largo.

Cuando se implementa el proceso anterior con hardware digital, es conveniente cambiar ligeramente el proceso. Primero, en lugar de proporcionar circuitos digitales para almacenar y sumar en forma simultánea tantos números binarios como haya números 1 en el multiplicador, es conveniente proporcionar circuitos para la suma sólo de dos números binarios y acumular en dorma sucesiva los productos parciales en un registro. Segundo, en lugar de correr el multiplicando a la izquierda, el producto parcial se corre a la derecha, lo cual resulta en dejar el producto parcial y el multiplicando en las posiciones relativas requeridas. Tercero, cuando el bit correspondiente del multiplicador es un 0, no hay necesidad de sumar todos los 0 al producto parcial, ya que esto no altera su valor.

El subsistema procesador de datos para el multiplicador binario se muestra en la Fig. 8-20. El multiplicando se almacena en el registro B. El multiplicador se almacena en el registro Q, y el producto parcial se forma en el registro A. Un adicionador paralelo similar al circuito mostrado en la Fig. 5-1 se utiliza para añadir los contenidos del registro B al registro A. El flip-flop A almacena la cuenta que se lleva después de la adición. El contador P inicialmente se establece para mantener un número binario igual al número de bits en el multiplicador. Este contador se decrementa después de la formación de cada producto parcial. Cuando el contenido del contador alcanza cero, el producto se forma en el registro doble A y Q y se detiene el proceso.

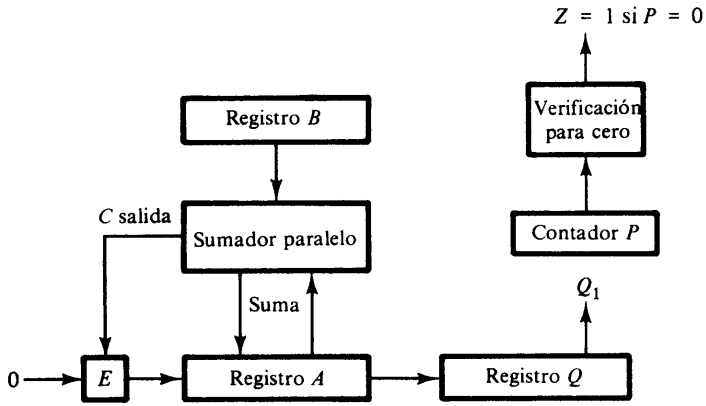


Figura 8-20 Procesador de datos para multiplicador binario.

El control lógico permanece en un estado inicial hasta que la señal de inicio S llega a ser un 1. El sistema realiza entonces la multiplicación. La suma de A y B forma un producto parcial, el cual se transfiere a A. La salida del acarreo mediante la adición ya sea 0 o 1 se transfiere a E. Tanto el producto parcial en A como el multiplicador en Q se corren a la derecha. El bit menos significativo de A se corre a la posición más significativa de Q; el acarreo de E se corre a la posición más significativa de A; y 0 se corre dentro de E. Después de la operación de corrimiento a la derecha, un bit del producto parcial se transfiere dentro de Q en tanto los bits del multiplicador en Q se corren una posición a la derecha. En esta forma, el bit de la extrema derecha del registro Q, designado por Q_1 siempre mantiene el bit del multiplicador que debe inspeccionarse a continuación.

Diagrama ASM

El diagrama ASM para el multiplicador binario se muestra en la Fig. 8-21. Al inicio, el multiplicando está en B y el multiplicador en Q. El proceso de multiplicación se inicia cuando $S = 1$. El registro A y el flip-flop E se despejan y el contador de secuencia P se ajusta en un número binario n, el cual es igual al número de bits en el multiplicador.

A continuación se entra en un lazo que mantiene formando los productos parciales. El bit multiplicador en Q_1 se verifica, y si es igual a 1, el multiplicando en B se agrega al producto parcial en A. La cuenta que se lleva de la adición se transfiere a E. El producto parcial en A se deja sin cambio si $Q_1 = 0$. El contador P se decrementa en 1 sin importar el valor de Q_1. Los registros E, A y Q se combinan en un registro compuesto EAQ, el cual se corre entonces un lugar a la derecha para obtener un nuevo producto parcial.

El valor en el contador P se verifica después de la formación de cada producto parcial. Si el contenido de P no es cero, la entrada de control Z es igual a 0 y el proceso se repite para formar un nuevo producto parcial. El proceso se detiene cuando el contador P llega a ser 0 y la entrada de control Z es igual a 1. Obsérvese que el producto parcial formado en A se corre dentro de Q un bit a la vez y, a la larga, repone el

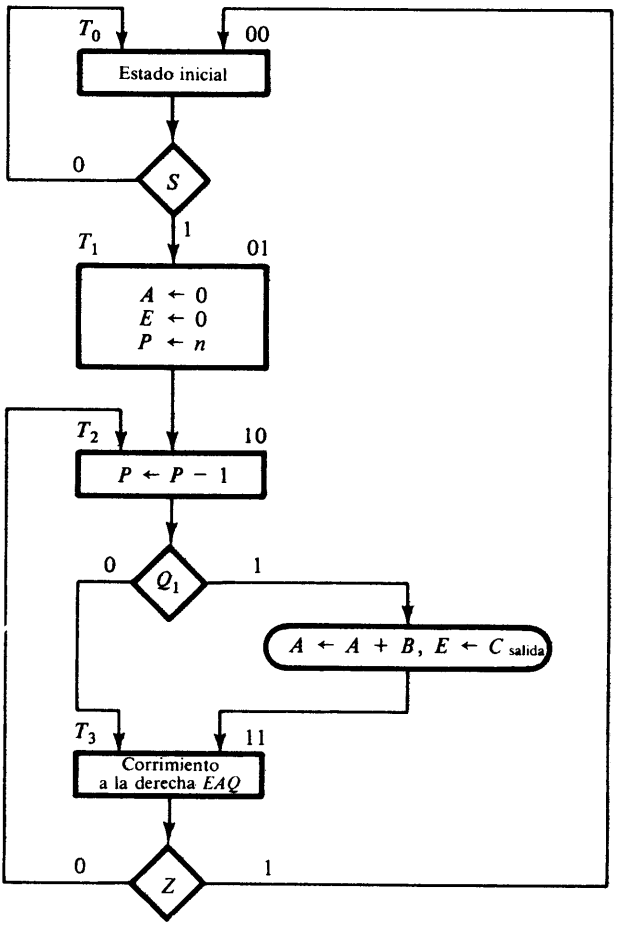


Figura 8-21 Carta ASM para multiplicador binario.

multiplicador. El producto final está disponible en A y Q, con A manteniendo los bits más significativos y Q los bits menos significativos.

El ejemplo numérico anterior se repite en la Fig. 8-22 para aclarar el proceso de multiplicación. El procedimiento sigue los pasos delineados en el diagrama ASM.

Control PLA

El control para el multiplicador binario tiene cuatro estados y tres entradas. La asignación del estado binario se muestra en el diagrama ASM sobre cada casilla de estado. Las tres entradas de control son S, Q_1 y Z. El diseño de una unidad de control con un PLA es similar al diseño que usa flip-flops D y un decodificador. La única diferencia estriba en la forma en que el circuito combinacional parte del control de implanta. El PLA en forma esencial reemplaza al decodificador y todas las compuertas en las entradas de los flip-flops.

Multiplicando $B = 10111$

	E	A	Q	P
Multiplicador en Q	0	00000	10011	101
$Q_1 = 1$; añadir B		10111		
Primer producto parcial	0	10111		100
correr a la derecha EAQ	0	01011	11001	
$Q_1 =$: añadir B		10111		
Segundo producto parcial	1	00010		011
Correr a la derecha EAQ	0	10001	01100	
$Q_1 = 0$; correr a la derecha EAQ	0	01000	10110	010
$Q_1 = 0$; correr a la derecha EAQ	0	00100	01011	001
$Q_1 = 1$; añadir B		10111		
Quinto producto parcial	0	11011		000
Correr a la derecha EAQ	0	01101	10101	
Producto final en $AQ = 0110110101$				

Figura 8-22 Ejemplo de multiplicación binaria.

El diagrama de bloques del control PLA se muestra en la Fig. 8-23. El PLA se conecta a un registro con dos flip-flops, G_1 y G_2. Las entradas al PLA son los valores del estado presente en el registro y las tres entradas de control. Las salidas del PLA proporcionan los valores para el estado siguiente en el registro y las variables de salida del control. Hay una salida para cada estado presente y una salida adicional para la operación condicional $D = Q_1T_2$. Ya que el PLA implanta el circuito combinacional de control, bien pueden incluirse dentro de él las compuertas para todas las operaciones condicionales. En el multiplicador binario hay una operación condicional para añadir B a A durante el estado T_2 siempre que $Q_1 = 1$. La salida D del PLA activará esta operación en el procesador de datos

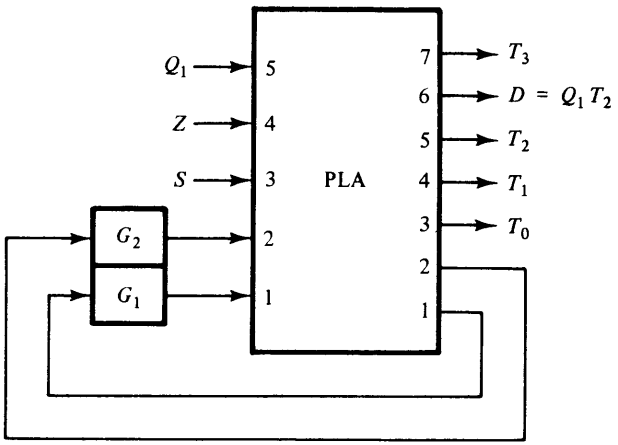


Figura 8-23 Diagrama de bloques del control PLA.

En cualquier momento dado, el estado presente del registro junto con las condiciones de entrada determinan los valores de salida y el estado siguiente para el registro. El siguiente pulso de reloj inicia las operaciones en el registro especificadas por las salidas PLA y transfiere el siguiente valor de estado dentro del registro. Esto proporciona un nuevo estado de control y posibles valores diferentes de entrada. Por eso, el PLA actúa como la parte del circuito combinacional del circuito secuencial para generar las salidas de control y los valores del estado siguiente para el registro.

Tabla del programa PLA

La organización interna del PLA se presentó en la Sección 5-8. También se mostró cómo obtener la tabla del programa PLA. Se aconseja al lector que repase esta sección para tener la seguridad de que se ha comprendido el significado de una tabla de programa PLA. Las trayectorias internas del PLA están construidas de acuerdo con las especificaciones que se dan en la tabla del programa. El diseño de un control PLA requiere que se obtenga la tabla de estado para el circuito. La tabla de estado da esencialmente toda la información necesaria para obtener la tabla del programa PLA.

La tabla de programa PLA puede obtenerse en forma directa mediante la tabla de estado sin necesidad de procedimientos de simplificación. La tabla del programa PLA que se lista en la Tabla 8-7 especifica siete términos productos, uno para cada puede ser una función de una de las variables de entrada de control o puede ser independiente de cualquier entrada. Si una variable de entrada no influencia el estado siguiente, se marca con una condición no importa, X. Si hay dos transiciones diferentes desde el mismo estado presente, el estado presente se repite en la tabla pero los estados siguientes tienen asignados diferentes valores binarios. En la tabla también se listan todas las salidas de control como una función del estado presente. Obsérvese que la entrada Q_1 no afecta el estado siguiente sino sólo determina el valor de la salida D durante el estado T_2.

La tabla de programa PLA puede obtenerse en forma directa mediante la tabla de estado sin necesidad de procedimientos de simplificación. La tabla del programa PLA que se lista en la Tabla 8-7 especifica siete términos productos, uno para cada

TABLA 8-6 Tabla de estado para el circuito de control

Estado presente		Entradas			Estado siguiente		Salidas				
G_1	G_2	S	Z	Q_1	G_1	G_2	T_0	T_1	T_2	D	T_3
0	0	0	X	X	0	0	1	0	0	0	0
0	0	1	X	X	0	1	1	0	0	0	0
0	1	X	X	X	1	0	0	1	0	0	0
1	0	X	X	0	1	1	0	0	1	0	0
1	0	X	X	1	1	1	0	0	1	1	0
1	1	X	0	X	1	0	0	0	0	0	1
1	1	X	1	X	0	0	0	0	0	0	1

TABLA 8-7 Tabla del programa para el PLA

Término producto	Entradas 1	2	3	4	5	Salidas 1	2	3	4	5	6	7	Comentarios
1	0	0	0	—	—	—	—	1	—	—	—	—	$T_0 = 1$, $S = 0$
2	0	0	1	—	—	—	1	1	—	—	—	—	$T_0 = 1$, $S = 1$
3	0	1	—	—	—	1	—	—	1	—	—	—	$T_1 = 1$
4	1	0	—	—	0	1	1	—	—	1	—	—	$T_2 = 1$, $Q_1 = 0$
5	1	0	—	—	1	1	1	—	—	1	1	—	$T_2 = 1$, $D = 1$,
6	1	1	—	0	—	1	—	—	—	—	—	1	$T_3 = 1$, $Z = 0$
7	1	1	—	1	—	—	—	—	—	—	—	1	$T_3 = 1$, $Z = 1$

renglón en la tabla de estado. Las terminales de entrada y salida están marcadas con número, y las variables aplicadas a esas terminales numeradas se indican en el diagrama de bloques en la Fig. 8-23. Los comentarios no son parte de la tabla, pero se incluyen para mayor claridad.

De acuerdo con las reglas establecidas en la Sección 5-8, una trayectoria sin conexión para un PLA se indica por un guión (-) en la tabla. Las *X* en la tabla de estado denotan condiciones no importa que implican que no hay conexión para el PLA. Los 0 en las columnas de salida también indican que no hay conexiones en las compuertas OR dentro del PLA. La traducción de la tabla de estado a una tabla de programa PLA es muy simple. Las *X* en las columnas de "entrada" y los 0 en las columnas de "estado siguiente" y "salidas" se cambian a guiones, y todas las demás anotaciones permanecen igual. Las entradas al PLA son las mismas que en el estado presente y las entradas en la tabla de estado. Las salidas del PLA son iguales que en el estado siguiente y las salidas en la tabla de estado.

En el ejemplo anterior se demuestra el procedimiento para diseñar el control lógico con un PLA. Mediante las especificaciones del sistema, se obtiene primero una tabla de estado para el controlador. El número de estados determina el número de flip-flops para el registro. Entonces se conecta el PLA al registro y a las variables de entrada y salida. La tabla de programa PLA se obtiene de manera directa mediante la tabla de estado.

Los ejemplos que se presentaron en este capítulo demuestran cinco métodos de diseño de control lógico. No deben considerarse estos métodos como los únicos posibles. Un diseñador con recursos puede ser capaz de formular una configuración de control para adecuarla a una aplicación particular. Esta configuración puede constar de una configuración de métodos o puede constituir una organización de control diferente de las que se presentan aquí.

BIBLIOGRAFIA

1. Clare, C. R., *Designing Logic Systems Using State Machines.* New York: McGraw-Hill Book Co., 1973.

2. Winkel, D., y F. Prosser, *The Art of Digital Design.* Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1980.

3. Peatman, J. B., *Digital Hardware Design.* New York: McGraw-Hill Book Co., 1980

4. Wiatrowski, C. A., C. H. House, *Logic Circuits and Microcomputer Systems.* New York: McGraw-Hill Book Co., 1980.

5. Fletcher, W. I., *An Engineering Approach to Digital Design.* Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1980.

6. Rhyne, V. T., *Fundamentals of Digital Systems Design.* Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1973.

PROBLEMAS

8-1. Dibuje la porción de un diagrama ASM que especifica una operación condicional para incrementar el registro R durante el estado T_1 y transferir el estado T_2 si las entradas de control z y y son iguales a 1 y 0, respectivamente.

8-2. Muestre ocho trayectorias de salida en un bloque ASM que emanan desde las casillas de decisión que verifican los ocho valores binarios posibles de tres variables de control x, y y z.

8-3. Obtenga el diagrama ASM para las siguientes transiciones de estado:
(a) Si $x = 0$, el control pasa desde el estado T_1 al estado T_2; si $x = 1$, genere una operación condicional y pase desde T_1 a T_2.
(b) Si $x = 1$, el control pasa desde T_1 a T_2 y entonces a T_3; si $x = 0$, el control pasa desde T_1 a T_3.
(c) Principie desde el estado T_1; entonces: si $xy = 00$, pasa a T_2; si $xy = 01$, pasa a T_3; si $xy = 10$, pasa a T_1; en otra forma, pasa a T_3.

8-4. Construya un diagrama ASM para un sistema digital que cuenta el número de personas en un cuarto. Las personas entran a la habitación por una puerta con una fotocelda que cambia una señal x desde 1 a 0 cuando se interrumpe la luz. Salen del cuarto por una segunda puerta con una fotocelda similar con una señal y. Tanto x como y están sincronizadas con el reloj, pero pueden permanecer encendidas o apagadas por más de un periodo de pulso de reloj. El subsistema procesador de datos consta de un contador hacia abajo con un exhibidor de su contenido.

8-5. Explique cómo difiere el diagrama ASM de un diagrama de flujo convencional. Utilice la Fig. 8-5 como una ilustración, muestre la diferencia en interpretación.

8-6. Diseñe el contador de 4-bit con despeje síncrono especificado en la Fig. 8-10.

8-7. Use mapas de cinco variables, derive las funciones booleanas de entrada a los flip-flops JK en la Fig. 8-11.

8-8. Diseñe el control cuya tabla de estado está dada en la Tabla 8-3 utilizando dos multiplexores, un registro y un decodificador.

8-9. Diseñe el control del ejemplo de diseño en la Sección 8-5 por el método de un flip-flop por estado.

8-10. El diagrama de estado de una unidad de control se muestra en la Fig. P8-10. Tiene cuatro estados y dos entradas, x y y.
(a) Dibuje el diagrama ASM equivalente, dejando vacías las casillas de estado.
(b) Diseñe el control con multiplexores.

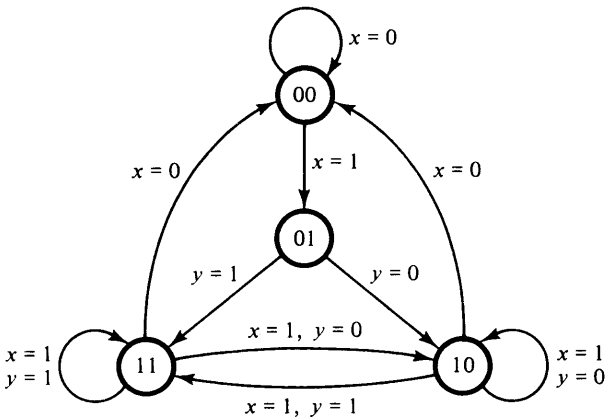


Figura P8-10 Diagrama de estados de control para el problema 8-10.

8-11. Suponga que $R1$ en la Fig. 8-18 es el registro de corrimiento de 4-bit que se muestra en la Fig. 7-9. Muestre cómo las entradas de corrimiento y carga en la Fig. 8-18 están conectadas a las entradas s_1 y s_0 del registro de corrimiento.

8-12. Diseñe un sistema digital con tres registros de 4-bit, A, B y C, para realizar las siguientes operaciones:
1. Transferencia de dos números binarios A y B cuando se habilita una señal de inicio.
2. Si $A < B$, se corren a la izquierda los contenidos de A y se transfiere el resultado al registro B.
3. Si $A > B$, se corren a la derecha los contenidos de B y se transfiere el resultado al registro C.
4. Si $A = B$, se transfiere el número al registro C sin cambio.

8-13. Diseñe un sistema digital que multiplique dos números binarios por el método de adición repetida. Por ejemplo, para multiplicar 5 X 4, el sistema digital evalúa el producto por la adición del multiplicando cuatro veces: $5 + 5 + 5 + 5 = 20$. Permita que el multiplicando esté en el registro BR, el multiplicador en el registro AR y el producto en el registro PR. Un circuito sumador añade los contenidos de BR a PR. Un circuito para detección de cero Z verifica cuando AR llega a ser 0 después de cada vez que se decrementa.

8-14. Demuestre que la multiplicación de dos números de n-bit da un producto de longitud menor que o igual a $2n$ bits.

8-15. En la Fig. 8-20, el registro Q mantiene el multiplicador y el registro B mantiene el multiplicando. Suponga que cada número consta de 15 bits.
(a) ¿Cuántos bits pueden esperarse en el producto y dónde está disponible?
(b) ¿Cuántos bits hay en el contador P y cuál es el número binario cargado dentro de él al inicio?
(c) Diseñe el circuito que verifique para cero en el contador P.

8-16. Liste los contenidos de los registros E, A, Q y P similar a la Fig. 8-22 durante el proceso de multiplicar los dos números 11111 (multiplicando) y 10101 (multiplicador).

8-17. Determine el tiempo que toma procesar la operación de multiplicación en el multiplicador binario que se describe en la Sección 8-6. Suponga que el registro Q tiene n bits y el periodo de reloj es T nanosegundos.

8-18. Diseñe el circuito de control del multiplicador binario especificado por el diagrama ASM en la Fig. 8-21 utilizando cada uno de los siguientes métodos:
(a) Flip-Flops *JK* y compuertas.
(b) Flip-flops *D* y un decodificador.
(c) Multiplexores de entrada y un registro.
(d) Un flip-flop por estado.

8-19. Diseñe el control cuya tabla de estado se muestra en la Tabla 8-3 usando el método PLA.

8-20. Considere el diagrama ASM de la Fig. P8-20. Las operaciones de registro no se especifican, porque sólo se tiene interés en diseñar el control lógico.
(a) Dibuje el diagrama de estado equivalente.
(b) Diseñe el control con un flip-flop por estado.

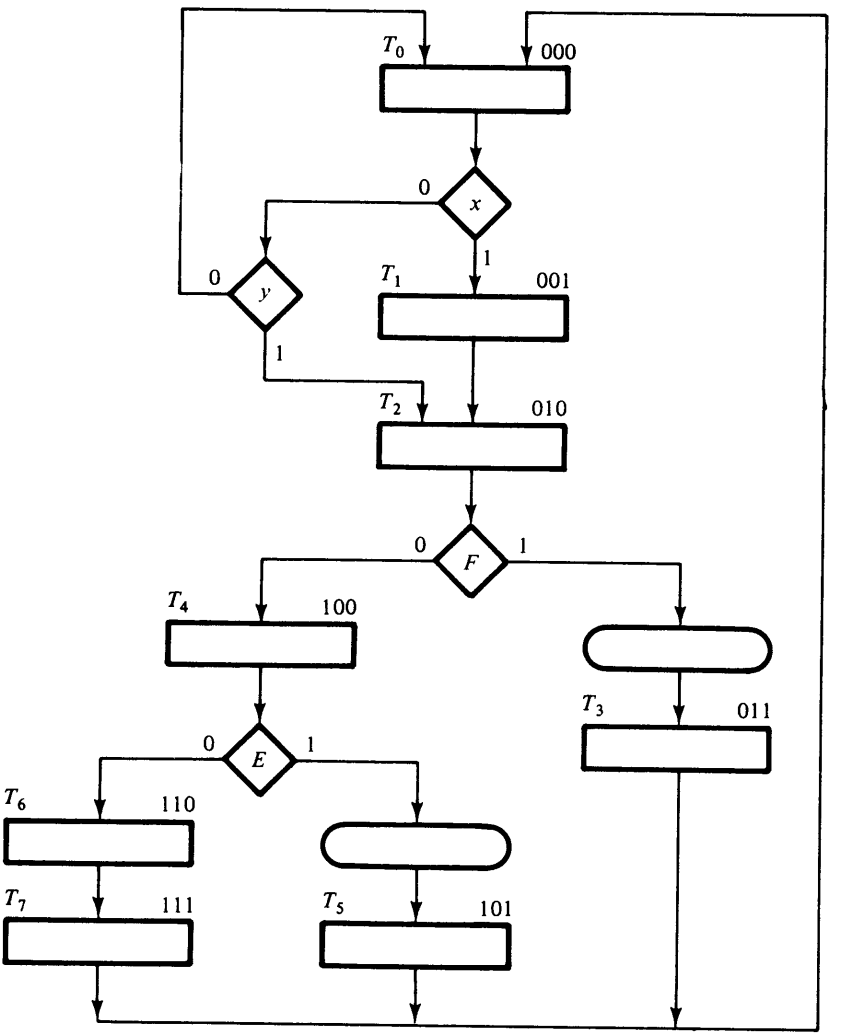


Figura P8-20 Diagrama ASM para los problemas 8-20 al 8-22 inclusive.

8-21. (a) Derive la tabla de estado para el diagrama ASM en la fig. P8-20.
(b) Diseñe el control con tres flip-flops *D*, un decodificador y compuertas.
(c) Diseñe el control con un registro y un PLA. Liste la tabla de programa PLA.

8-22. (a) Derive una tabla mostrando las condiciones de entrada del multiplexor para el control especificado en el diagrama ASM en la Fig. P8-20.
(b) Diseñe el control con tres multiplexores, un registro con tres flip-flops, y un decodificador 3×8.