
Lógica combinacional con MSI y LSI



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 es 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:

<i>Subíndice i</i>	<u>4 3 2 1</u>	<i>Sumador completo de la Fig. 4-5</i>
Acarreo de entrada	0 1 1 0	C_i
Consumando	1 0 1 1	A_i
Sumando	0 0 1 1	B_i
Suma	1 1 1 0	S_i
Acarreo de salida	0 0 1 1	C_{i+1}
		z
		x
		y
		S
		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

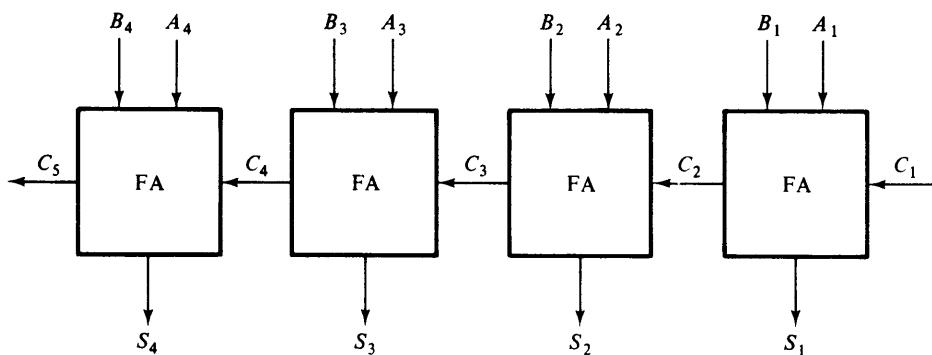


Figura 5-1 Sumadores completos de 4-bit.

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 MSI. 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.

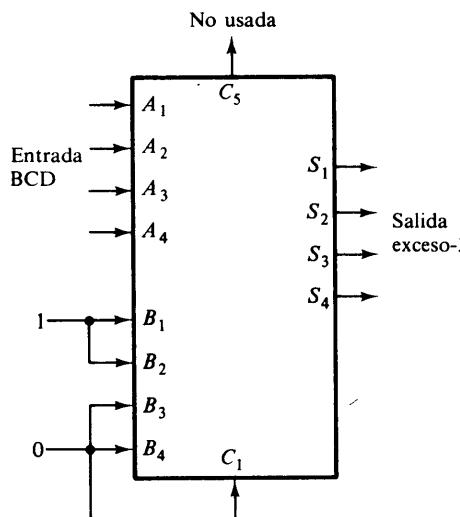


Figura 5-2 Convertidor de código BCD-a-exceso-3.

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

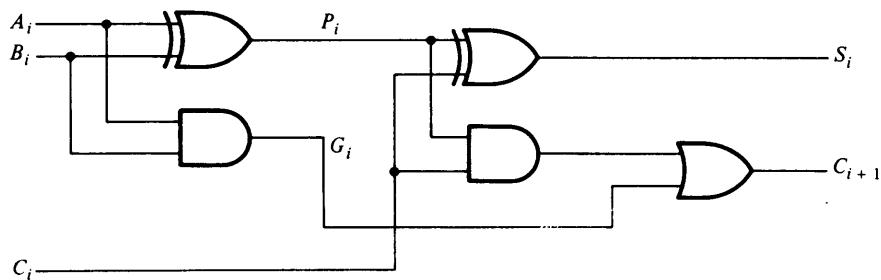


Figura 5-3 Circuito sumador completo.

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:

$$\begin{aligned}P_i &= A_i \oplus B_i \\G_i &= A_i B_i\end{aligned}$$

la suma y acarreo de salida pueden expresarse como:

$$\begin{aligned}S_i &= P_i \oplus C_i \\C_{i+1} &= G_i + P_i C_i\end{aligned}$$

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:

$$C_2 = G_1 + P_1 C_1$$

$$C_3 = G_2 + P_2 C_2 = G_2 + P_2(G_1 + P_1 C_1) = G_2 + P_2 G_1 + P_2 P_1 C_1$$

$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_1$$

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 $C_s = G + PC_1$.

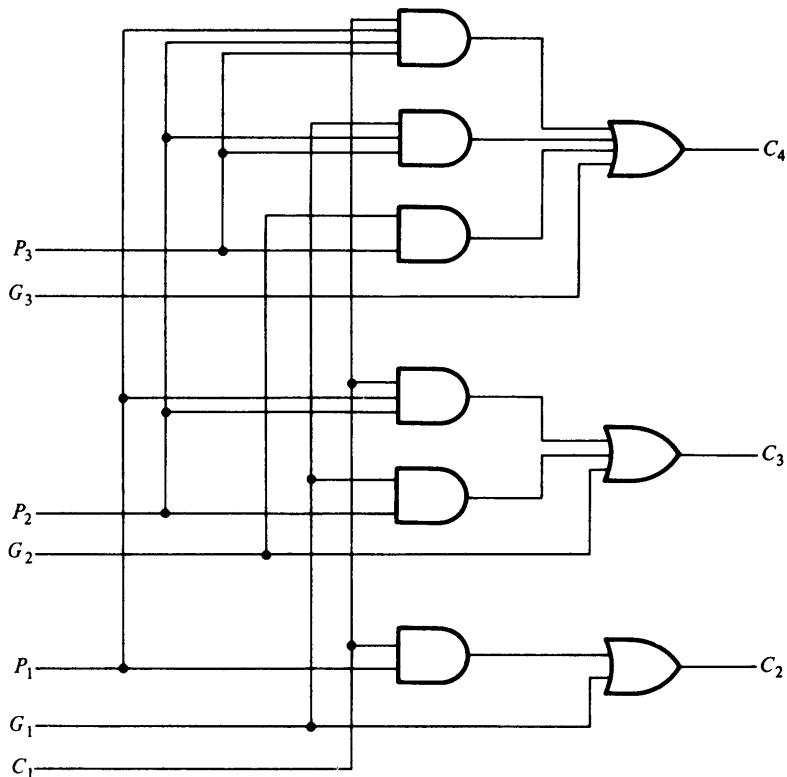


Figura 5-4 Diagrama lógico de un generador de acarreo anticipado.

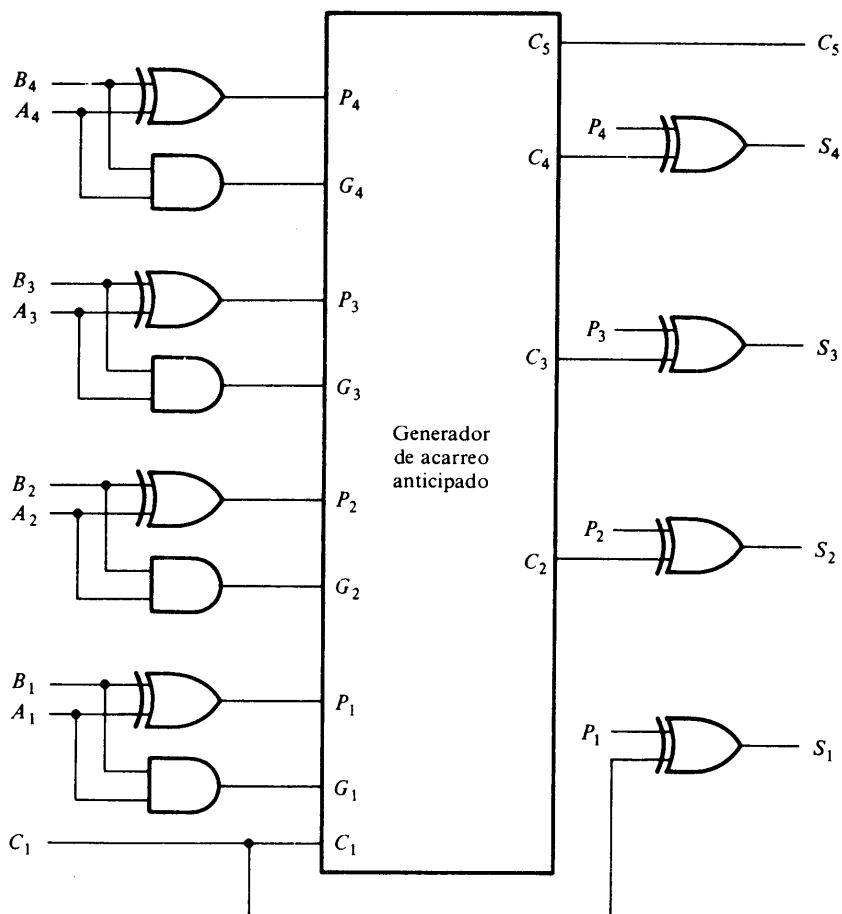


Figura 5-5 Sumadores completos de 4-bit con acarreo anticipado.

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-exclusivo. 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 dígito 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.

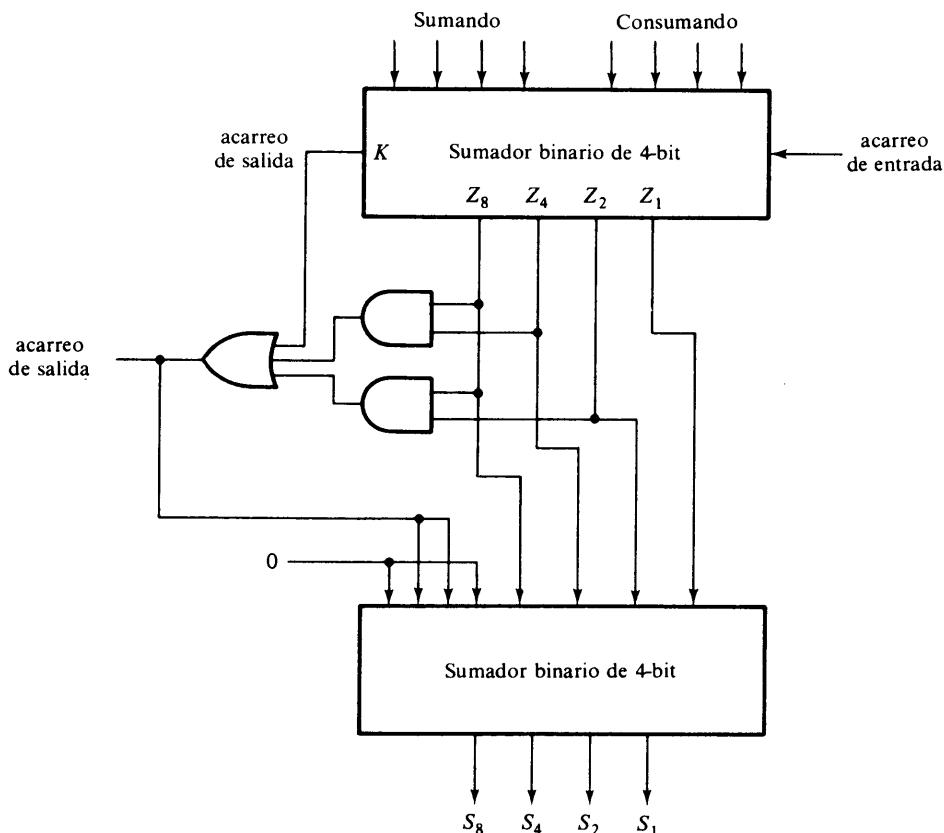


Figura 5-6 Diagrama de bloques de un sumador BCD.

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:

$$\begin{aligned} A &= A_3A_2A_1A_0 \\ B &= B_3B_2B_1B_0 \end{aligned}$$

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:

$$x_i = A_iB_i + A'_iB'_i \quad i = 0, 1, 2, 3$$

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 salidas 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).

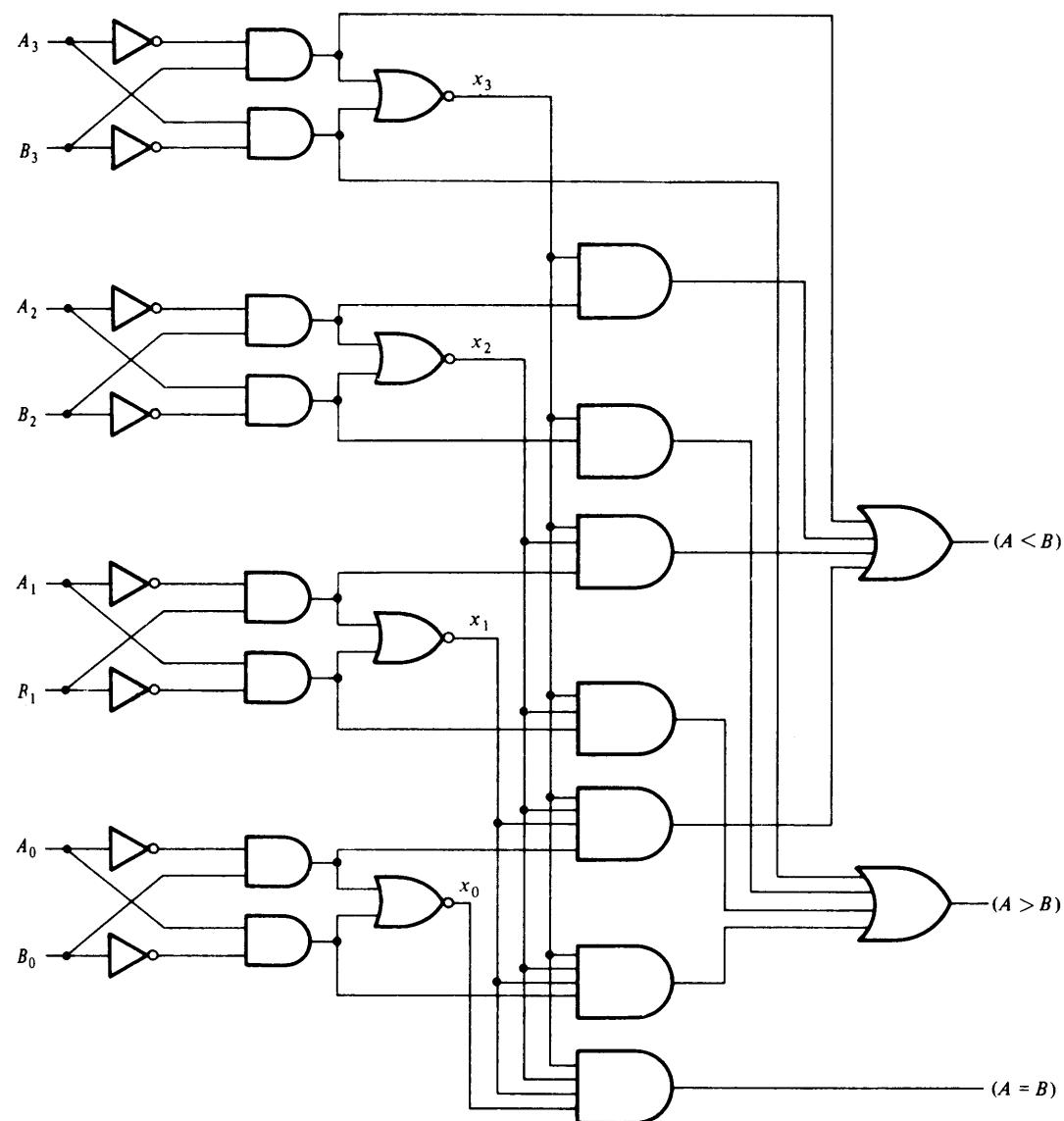


Figura 5-7 Comparador de magnitud de 4-bit.

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.

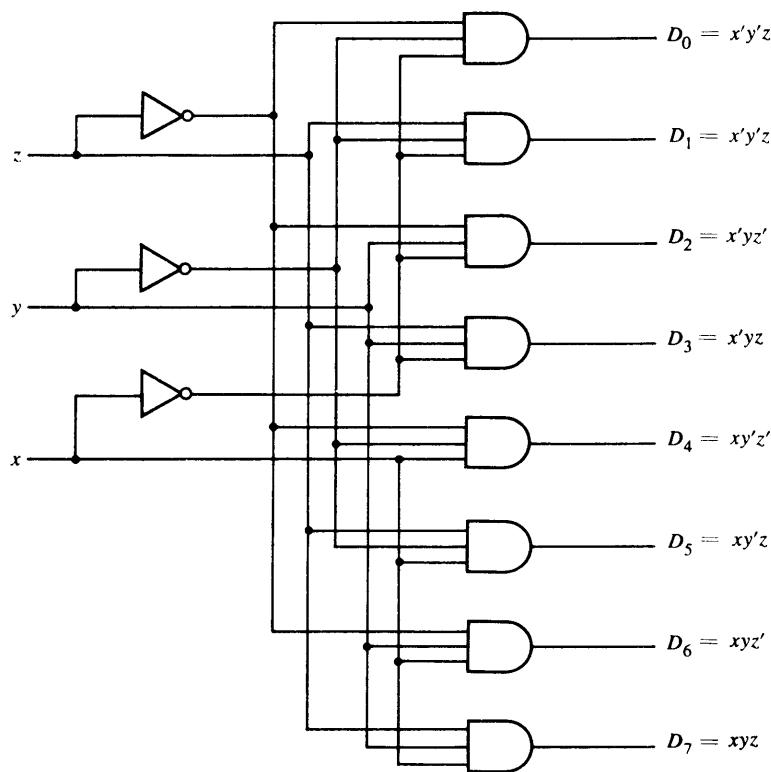


Figura 5-8 Un decodificador 3-a-8 linea.

TABLA 5-2 Tabla de verdad de un decodificador de 3-a-8 linea

Entradas <i>x y z</i>	Salidas							
	<i>D</i> ₀	<i>D</i> ₁	<i>D</i> ₂	<i>D</i> ₃	<i>D</i> ₄	<i>D</i> ₅	<i>D</i> ₆	<i>D</i> 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:

$$D_2 = x'yz'$$

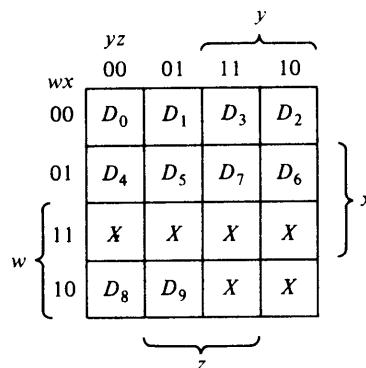


Figura 5-9 Mapa para simplificar un decodificador BCD-a-decimal.

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.

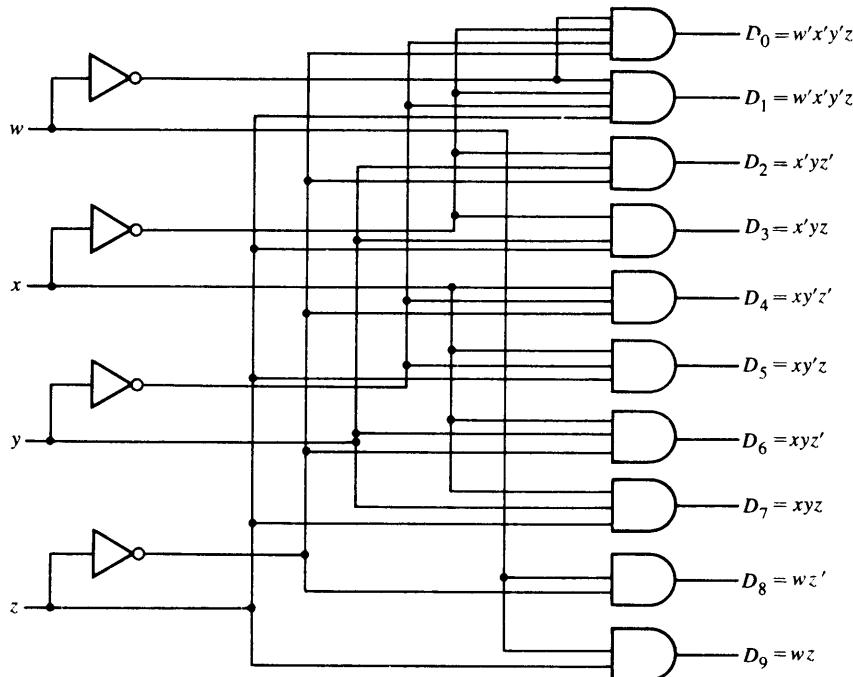


Figura 5-10 Decodificador BCD-a decimal.

TABLA 5-3 Tabla de verdad parcial para el circuito de la Fig. 5-10

Entradas				Salidas									
<i>w</i>	<i>x</i>	<i>y</i>	<i>z</i>	<i>D</i> ₀	<i>D</i> ₁	<i>D</i> ₂	<i>D</i> ₃	<i>D</i> ₄	<i>D</i> ₅	<i>D</i> ₆	<i>D</i> ₇	<i>D</i> ₈	<i>D</i> 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 decodificador 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 adiconador 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.

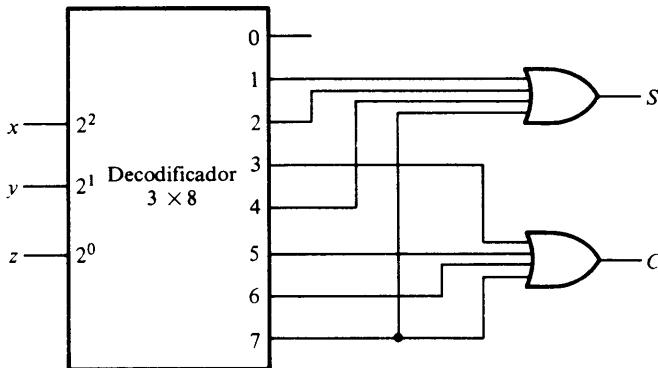


Figura 5-11 Implementación de un sumador completo con un decodificador.

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 desprecindibles. 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-

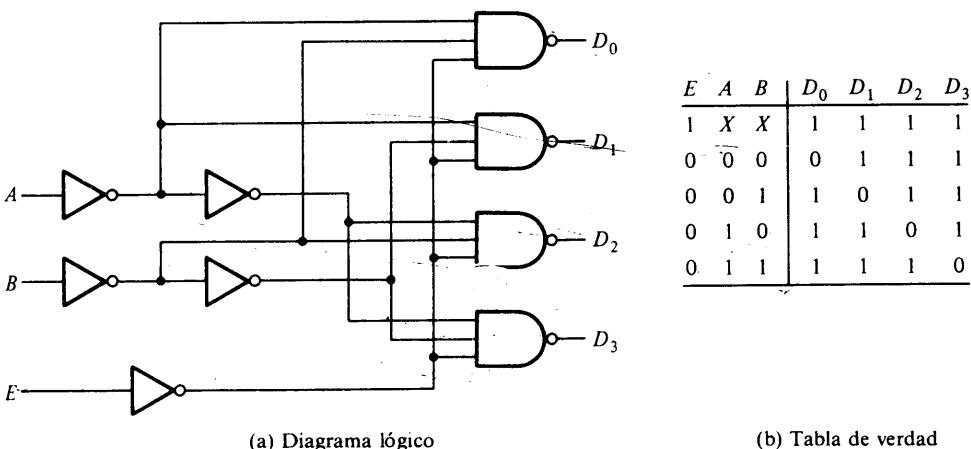


Figura 5-12 Un decodificador 2-a-4 línea con habilitación de entrada (E).

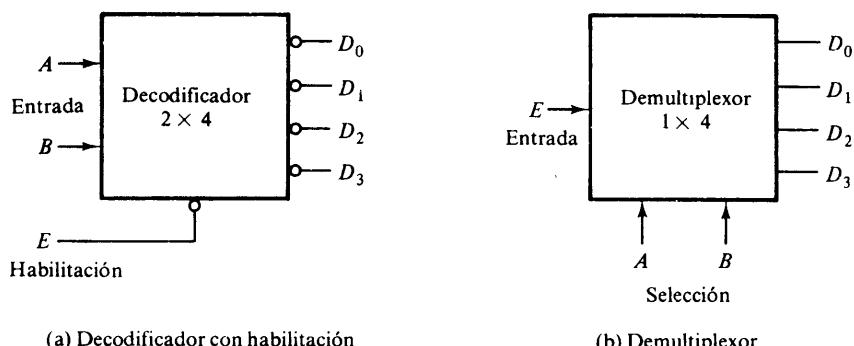


Figura 5-13 Diagramas de bloque para el circuito de la Fig. 5-12.

mimos 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

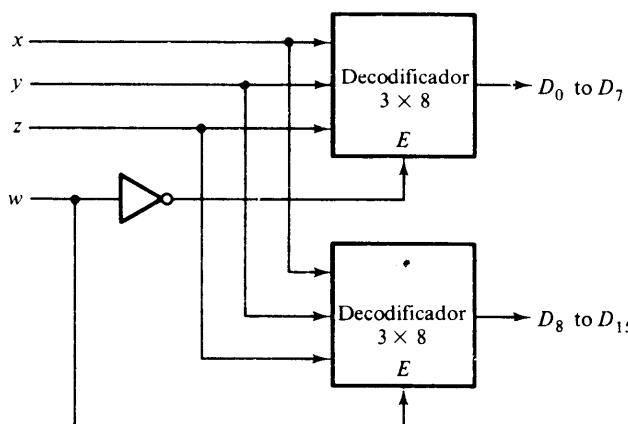


Figura 5-14 Un decodificador 4×16 construido con dos decodificadores 3×8 .

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.

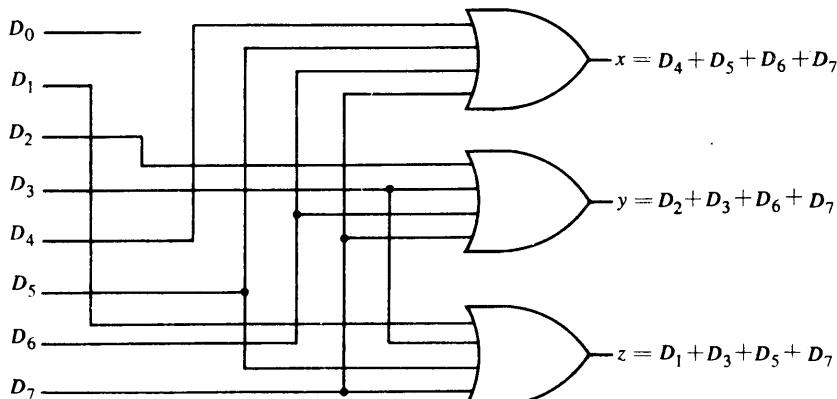


Figura 5-15 Codificador octal-a-binario.

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 expandir 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.

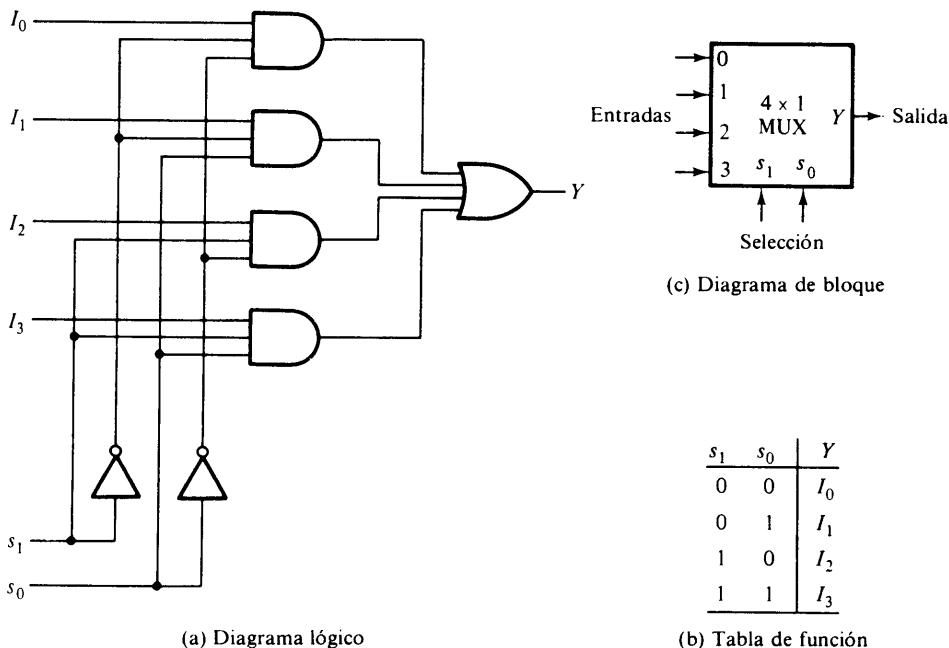


Figura 5-16 Un multiplexor de 4-a-1 línea.

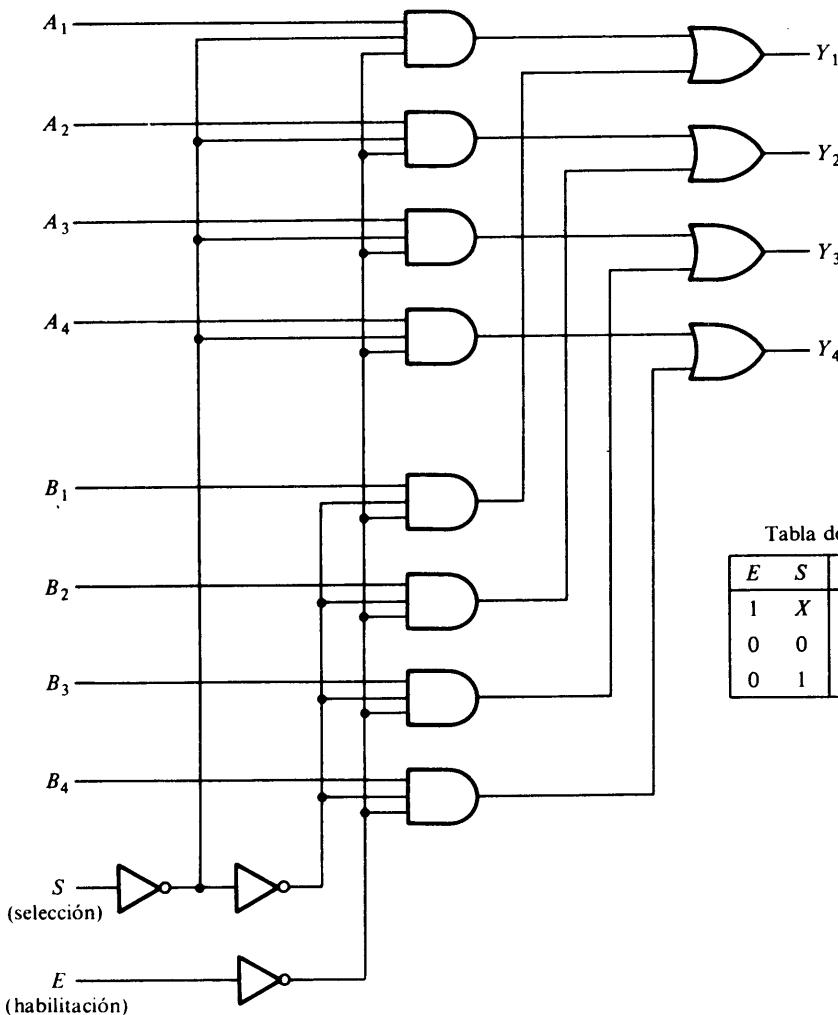


Tabla de función

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.

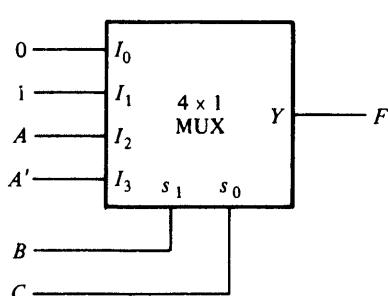
Para demostrar este procedimiento con un ejemplo concreto, considérese la función de tres variables:

$$F(A, B, C) = \Sigma(1, 3, 5, 6)$$

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 $ABCD \dots$, en donde A es la variable más a la izquierda en la secuencia ordenada de las n variables y $BCD\dots$ 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

Mintermino	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 minterminos. La segunda mitad de los minterminos tendrá su variable A sin complementar. Para una función de tres variables, A, B, C , se tienen ocho minterminos. La variable A se complementa en minterminos 0 a 3 y está sin complementar en los minterminos de 4 a 7.

Se listan las entradas del multiplexor y bajo ellas se listan todos los minterminos en dos renglones. En el primer renglón se listan todos los minterminos donde A está complementada, en el segundo renglón todos los minterminos con A sin complementar, como se muestra en la Fig. 5-18(c). Se encierran dentro de un círculo todos los minterminos de la función y se inspecciona por separado cada columna.

Se listan las entradas del multiplexor y bajo ellas se listan todos los minterminos en dos renglones. En el primer renglón se listan todos los minterminos donde A está complementada, en el segundo renglón todos los minterminos con A sin complementar, como se muestra en la Fig. 5-18(c). Se encierran dentro de un círculo todos los minterminos de la función y se inspecciona por separado cada columna.

Si los dos minterminos en una columna no están dentro de un círculo, aplíquese 0 a la entrada correspondiente del multiplexor.

Si los dos minterminos están dentro de un círculo, se aplica 1 a la entrada correspondiente del multiplexor.

Si el mintermino 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 .

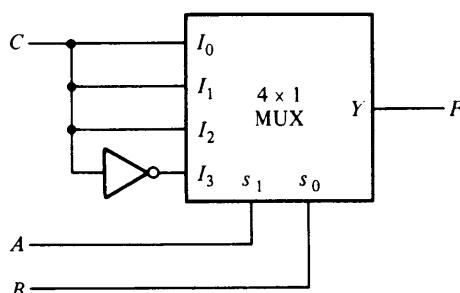
No es necesario escoger la variable de la extrema izquierda en la secuencia ordenada de una lista de variables para las entradas de multiplexor. De hecho, puede elegirse cualquiera de las variables para las entradas del multiplexor, siempre que se multiplique la tabla de implementación del multiplexor. 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



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

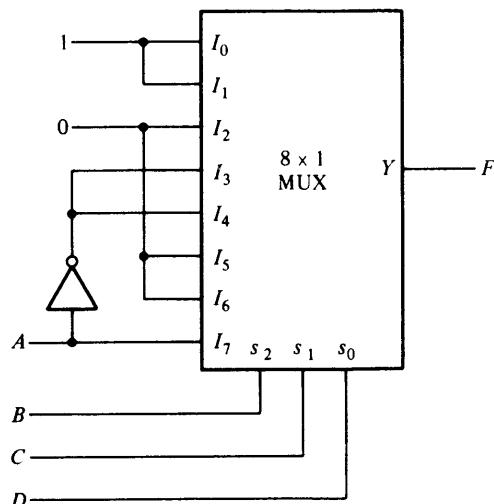


Figura 5-20 Implementación de $F(A, B, C, D) = \Sigma(0, 1, 3, 4, 8, 9, 15)$.

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.

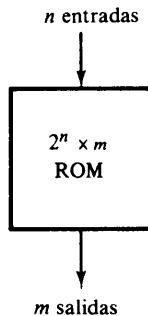


Figura 5-21 Diagrama de bloques de una ROM.

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 9 líneas de salida y 5 líneas de entrada para especificar $2^5 = 32$ 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

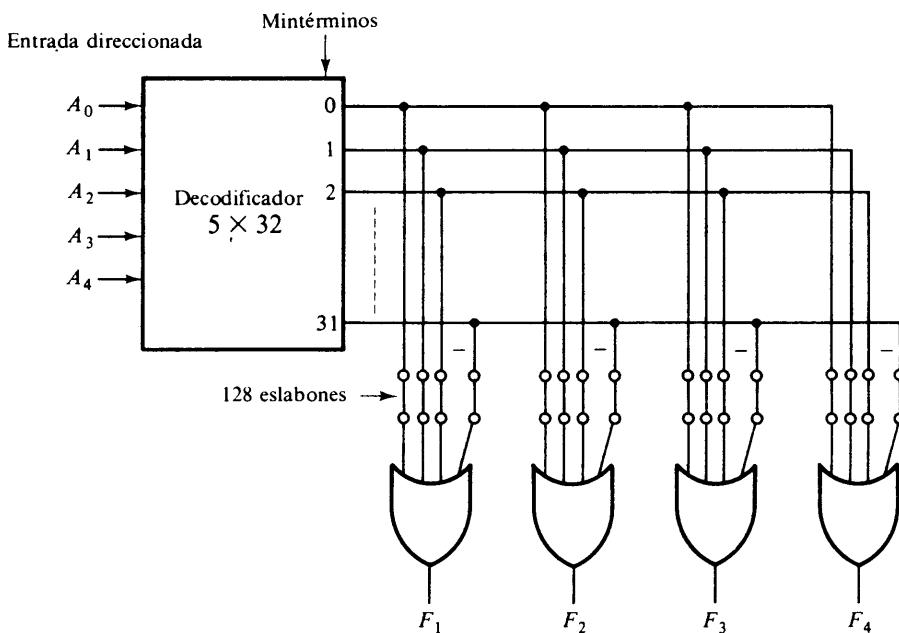


Figura 5-22 Construcción lógica de una ROM de 32×4 .

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.

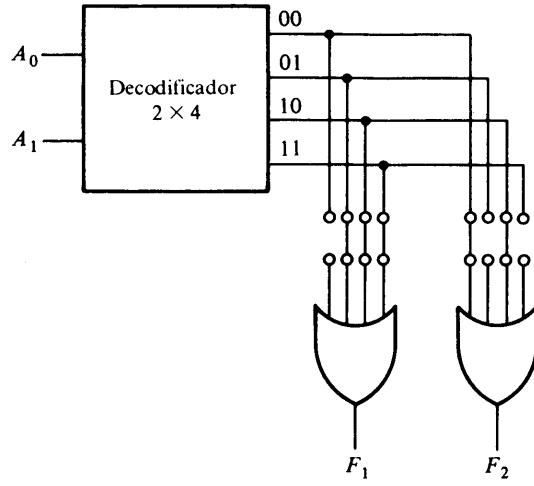
En la práctica, cuando se diseña un circuito mediante una ROM, no es necesario mostrar las conexiones internas en la compuerta de los eslabones dentro de la unidad como se hizo en la Fig. 5-23. Esto se mostró aquí tan solo para propósitos de demostración. Todo lo que el diseñador tiene que hacer es especificar 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.

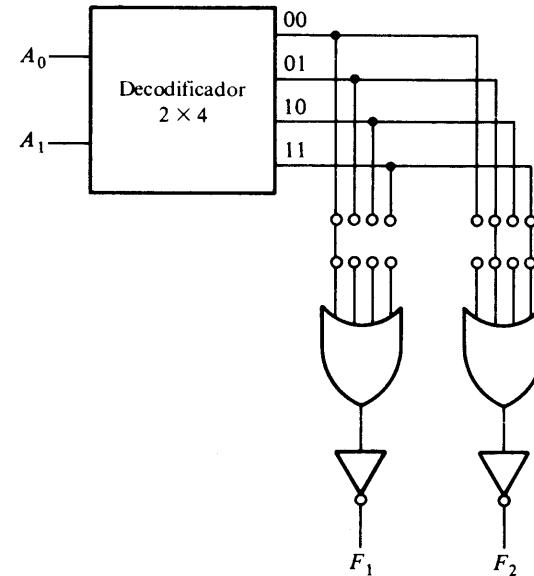
El primer paso es derivar la tabla de verdad para el circuito combinacional. En la mayoría de los casos esto es todo lo que se necesita. En algunos casos puede ajustarse una tabla de verdad más pequeña para la ROM por el uso de ciertas propiedades en la tabla de verdad del circuito combinacional. La 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



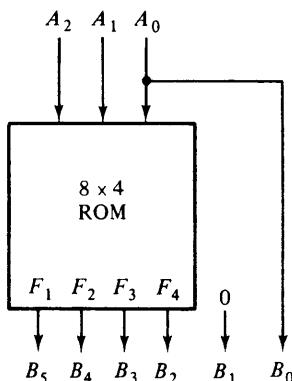
(c) ROM con compuertas AND-OR-INVERSORA

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 8×4 . 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 EEAROM.

La función de una ROM puede interpretarse en dos formas diferentes. La primera interpretación es de una unidad que implementa cualquier circuito combinatorial. 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 las 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.

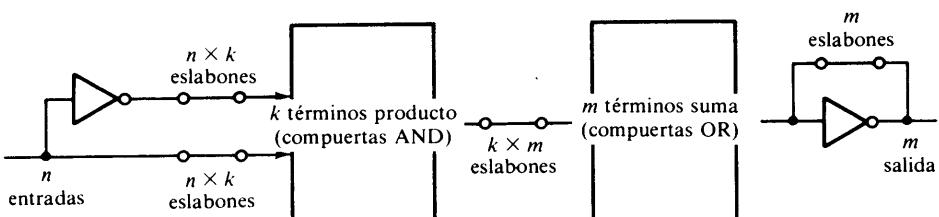


Figura 5-25 Diagrama de bloques del arreglo PLA.

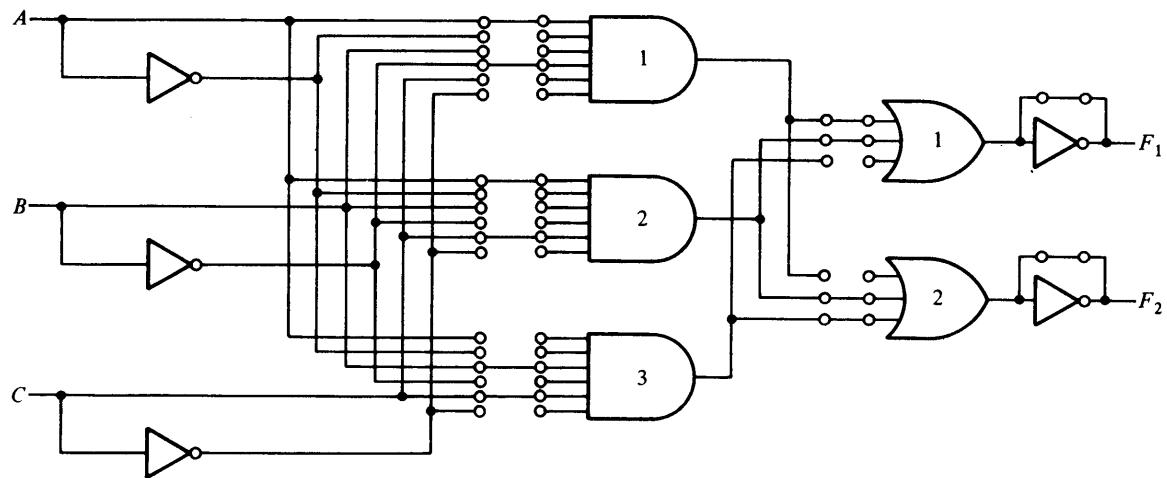


Figura 5-26 Arreglo PLA con 3 entradas, 3 términos producto y 2 salidas; implementa el circuito combinacional especificado en la Fig. 5-27.

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):

$$F_1 = AB' + AC$$

$$F_2 = AC + BC$$

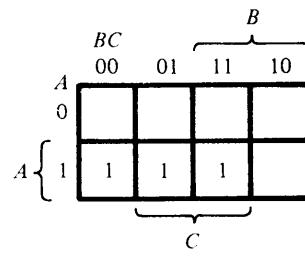
Hay tres distintos términos producto en este circuito combinacional: AB' , AC y BC . El circuito tiene tres entradas y dos salidas; de modo que el PLA de la Fig. 5-26 puede usarse para implementar este circuito combinacional.

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.

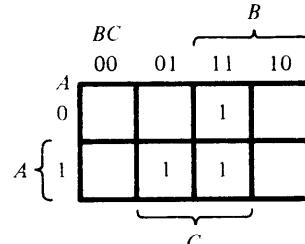
Para cada término producto, las entradas se marcan con 1, 0, o - (guion). Si una variable en el producto término aparece en su forma normal (sin prima), la variable correspondiente de entrada se marca con un 1. Si aparece complementada (con prima), la variable de entrada correspondiente se marca con un 0. Si la variable está ausente en el término producto, se marca con un guion. 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 guion 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			Salidas		T	T/C
	A	B	C	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 F₁ está marcada con 1 para los términos producto 1 y 2 con un guion 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 guion 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

		BC		B	
		00	01	11	10
A		0			
A	1			1	
	1		1	1	1

C

$$F_1 = AC + AB + BC$$

		BC		B	
		00	01	11	10
A		0	1		
A	1		1		1
	1			1	

C

$$F_2 = B'C' + A'C' + ABC$$

		BC		B	
		00	01	11	10
A		0	0		
A	1		0		
	1				

C

$$F'_1 = B'C' + A'C' + A'E'$$

		BC		B	
		00	01	11	10
A		0	0	0	
A	1		0		0
	1			0	0

C

$$F'_2 = B'C + A'C + ABC'$$

Tabla de programa del PLA

Término producto	Entradas			Salidas	
	A	B	C	F_1	F_2
$B'C'$	1	—	0	0	1
$A'C'$	2	0	—	0	1
$A'B'$	3	0	0	—	1
ABC	4	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:
- $$C_{i+1} = G_i + P_i C_i = (G'_i P'_i + G'_i C'_i)'$$

(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 , 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:

$$P_i = A_i + B_i$$

$$G_i = A_i B_i$$

Muestre que el acarreo de salida y la salida de suma de un sumador completo llega a ser:

$$C_{i+1} = (C'_i G'_i + P'_i)' = G_i + P_i C_i$$

$$S_i = (P_i G_i) \oplus C_i$$

(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 muestre que el circuito implementa un circuito sumador completo.

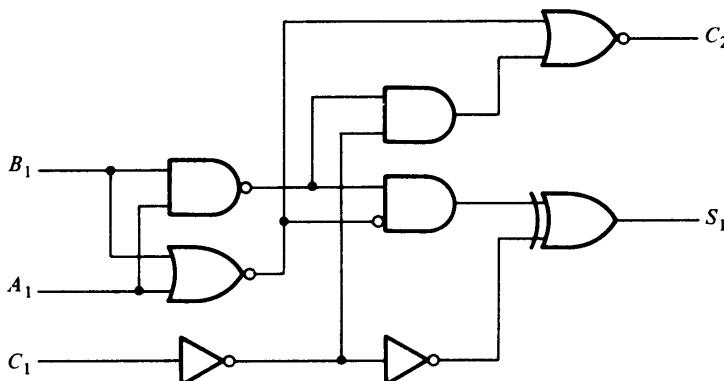


Figura P5-6 Primera etapa de un sumador paralelo.

- (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:
- El acarreo de salida es igual al acarreo de salida del sumador binario.
 - Si el acarreo de salida = 1, agregue 0011.
 - 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 significativos.

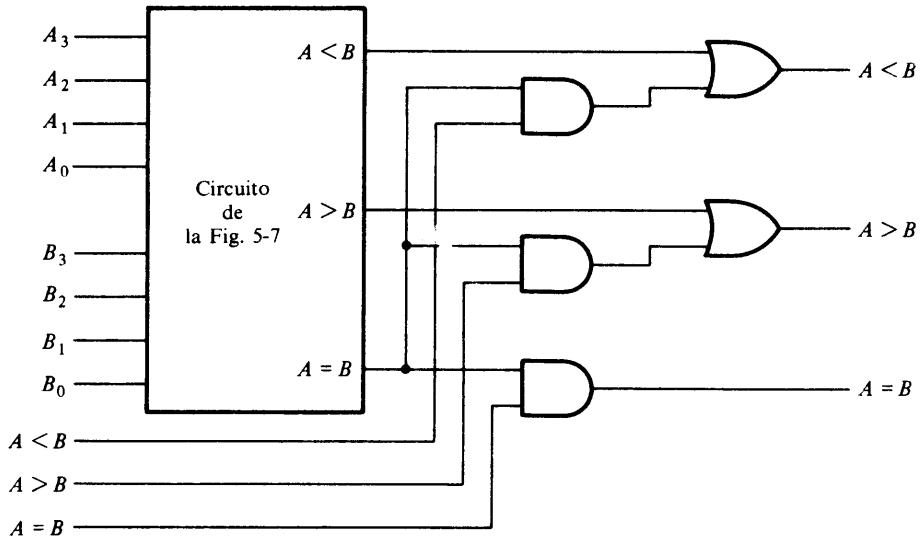


Figura P5-14 Circuito lógico equivalente del IC tipo 74L85.

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 combinación 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-linea 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.

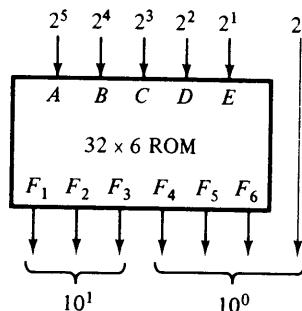


Figura P5-27 Convertidor de binario en decimal.

- 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:
 - Un sumador/restador BCD con un control de entrada para seleccionar entre la adición y la resta.
 - Un multiplicador binario que multiplique dos números de 4-bit.
 - 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.