DISEÑO DIGITAL

TERCERA EDICIÓN

M. Morris Mano
CALIFORNIA STATE UNIVERSITY, LOS ANGELES

TRADUCCIÓN
Roberto Escalona García
Ingeniero Químico
Universidad Nacional Autónoma de México

REVISIÓN TÉCNICA
Gonzalo Duchén Sánchez
Sección de Estudios de Postgrado e Investigación
Escuela Superior de Ingeniería Mecánica y Eléctrica
Unidad Culhuacán
Instituto Politécnico Nacional

México • Argentina • Brasil • Colombia • Costa Rica • Chile • Ecuador
España • Guatemala • Panamá • Perú • Puerto Rico • Uruguay • Venezuela

www.FreeLibros.me

4 Lógica combinacional

4-1 CIRCUITOS COMBINACIONALES

Los circuitos lógicos para sistemas digitales pueden ser combinacionales o secuenciales. Un circuito combinacional consiste en compuertas lógicas cuyas salidas en cualquier momento están determinadas por la combinación actual de entradas. Un circuito combinacional realiza una operación que se puede especificar lógicamente con un conjunto de funciones booleanas. Los circuitos secuenciales usan elementos de almacenamiento además de compuertas lógicas, y sus salidas son función de las entradas y del estado de los elementos de almacenamiento. Esto último, a su vez, es función de entradas anteriores. Por ello, las salidas de un circuito secuencial dependen no sólo de los valores actuales de las entradas, sino también de entradas anteriores, y el comportamiento del circuito se debe especificar con una sucesión temporal de entradas y estados internos. Los circuitos secuenciales se estudiarán en los capítulos 5 y 9.

Un circuito combinacional consiste en variables de entrada, compuertas lógicas y variables de salida. Las compuertas lógicas aceptan señales de las entradas y generan señales para las salidas. Este proceso transforma información binaria, de los datos de entrada dados a los datos de salida requeridos. En la figura 4-1 se presenta un diagrama de bloques de un circuito combinacional. Las n variables binarias de entrada provienen de una fuente externa; las m variables de salida van a un destino externo. Cada variable de entrada y de salida existe físicamente como una señal binaria que representa 1 lógico y 0 lógico. En muchas aplicaciones, el origen

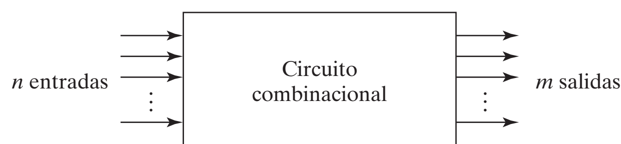


FIGURA 4-1
Diagrama de bloques de un circuito combinacional

www.FreeLibros.me

y el destino son registros de almacenamiento. Si los registros se incluyen con las puertas combinacionales, el circuito total se considera como un circuito secuencial.

Con n variables de entrada, hay 2^n posibles combinaciones de entradas binarias. Para cada una de esas combinaciones, hay un posible valor de salida. Por tanto, es posible especificar un circuito combinacional con una tabla de verdad que presenta los valores de salida para cada combinación de variables de entrada. También es factible describir un circuito combinacional con m funciones booleanas, una para cada variable de salida. Cada función de salida se expresa en términos de las n variables de entrada.

En el capítulo 1 se estudiaron los números binarios y los códigos binarios que representan cantidades discretas de información. Las variables binarias se representan físicamente con voltajes eléctricos o algún otro tipo de señal. Las señales se pueden manipular en compuertas lógicas digitales para efectuar las funciones requeridas. En el capítulo 2 se definió el álgebra booleana como una forma de expresar las funciones lógicas algebraicamente. En el capítulo 3 se explicó la manera de simplificar las funciones booleanas para lograr implementaciones económicas con compuertas. El propósito del presente capítulo es utilizar los conocimientos adquiridos en capítulos anteriores y formular procedimientos sistemáticos para el análisis y diseño de circuitos combinacionales. La resolución de algunos ejemplos representativos proporcionará un catálogo útil de funciones elementales importantes para entender los sistemas digitales.

Hay varios circuitos combinacionales que se usan ampliamente en el diseño de sistemas digitales. Esos circuitos pueden conseguirse en circuitos integrados y se clasifican como componentes estándar. Efectúan funciones digitales específicas que se necesitan a menudo en el diseño de sistemas digitales. En este capítulo presentaremos los circuitos combinacionales estándar más importantes, como los sumadores, restadores, comparadores, decodificadores, codificadores y multiplexores. Estos componentes se fabrican como circuitos MSI (de integración a mediana escala), y también se usan como *celdas estándar* en circuitos VLSI complejos como los circuitos integrados para aplicaciones específicas (ASIC). Las funciones de la celda estándar se interconectan dentro del circuito VLSI del mismo modo que se usan en el diseño MSI de múltiples CI.

4-2 PROCEDIMIENTO DE ANÁLISIS

El análisis de un circuito combinacional requiere deducir la función que realiza el circuito. Este proceso parte de un diagrama lógico dado y culmina en un conjunto de funciones booleanas, una tabla de verdad o una posible explicación del funcionamiento del circuito. Si el diagrama lógico a analizar va acompañado de un nombre de función o de una explicación de lo que se supone que hace, el problema de análisis se reducirá a una verificación de la función planteada. El análisis se efectúa manualmente encontrando las funciones booleanas o la tabla de verdad, o bien, utilizando un programa de simulación en computadora.

El primer paso del análisis consiste en asegurarse de que el circuito dado sea combinacional y no secuencial. El diagrama de un circuito combinacional tiene compuertas lógicas sin trayectorias de retroalimentación ni elementos de memoria. Una trayectoria de retroalimentación es una conexión de la salida de una compuerta a la entrada de una segunda compuerta que forma parte de la entrada a la primera compuerta. Las trayectorias de retroalimentación en un circuito digital definen a un circuito secuencial y deben analizarse según los procedimientos delineados en el capítulo 9.

Una vez que se verifica que el diagrama lógico representa un circuito combinacional, se procede a obtener las funciones booleanas de salida o la tabla de verdad. Si se está investigan-

www.FreeLibros.me

do la función del circuito, será necesario interpretar la operación de éste a partir de las funciones booleanas o la tabla de verdad obtenidas. El éxito de tal investigación será más asequible si tenemos experiencia previa con una amplia variedad de circuitos digitales.

Para obtener las funciones booleanas de salida a partir de un diagrama lógico, el procedimiento es el siguiente:

1. Rotule con símbolos arbitrarios todas las salidas de compuerta que son función de variables de entrada. Determine las funciones booleanas para cada salida de compuerta.
2. Rotule con otros símbolos arbitrarios las compuertas que son función de variables de entrada y de compuertas previamente rotuladas. Obtenga las funciones booleanas de estas compuertas.
3. Repita el proceso bosquejado en el paso 2 hasta obtener las salidas del circuito.
4. Por sustitución repetida de funciones previamente definidas, obtenga las funciones booleanas de salida en términos de variables de entrada.

El análisis del circuito combinacional de la figura 4-2 ilustra el procedimiento propuesto. Observe que el circuito tiene tres entradas binarias —A, B y C— y dos salidas binarias —F_1 y F_2. Las salidas de diversas compuertas están rotuladas con símbolos intermedios. Las salidas de compuertas que son función únicamente de variables de entrada son T_1 y T_2. La salida F_2 se deduce fácilmente de las variables de entrada. Las funciones booleanas de estas tres salidas son:

$$F_2 = AB + AC + BC$$
$$T_1 = A + B + C$$
$$T_2 = ABC$$

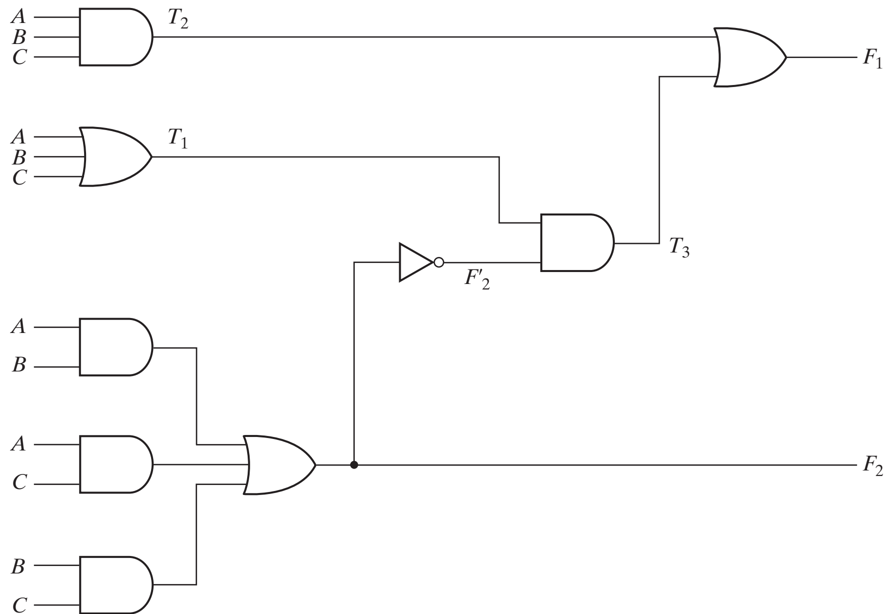


FIGURA 4-2
Diagrama lógico para el ejemplo de análisis

www.FreeLibros.me

A continuación, consideramos las salidas de compuertas que son función de símbolos ya definidos:

$$T_3 = F'_2 T_1$$
$$F_1 = T_3 + T_2$$

Para obtener F_1 en función de A, B y C, se realiza la siguiente serie de sustituciones:

$$\begin{aligned} F_1 &= T_3 + T_2 = F'_2 T_1 + ABC = (AB + AC + BC)'(A + B + C) + ABC \\ &= (A' + B')(A' + C')(B' + C')(A + B + C) + ABC \\ &= (A' + B'C')(AB' + AC' + BC' + B'C) + ABC \\ &= A'BC' + A'B'C + AB'C' + ABC \end{aligned}$$

Si se quiere investigar más a fondo y deducir la tarea de transformación de información que este circuito efectúa, habrá que dibujar el circuito a partir de las expresiones booleanas obtenidas y tratar de reconocer una operación conocida. Las funciones booleanas para F_1 y F_2 implementan el circuito que se muestra en la figura 4-7 (sección 4-4) y equivalen a un circuito sumador completo.

La deducción de la tabla de verdad del circuito es un proceso sencillo una vez que se conocen las funciones booleanas de salida. Para obtener la tabla de verdad directamente del diagrama lógico sin tener que deducir las funciones booleanas, se procede así:

1. Determine el número de variables de entrada del circuito. Para n entradas, forme las 2^n posibles combinaciones y haga una lista de los números binarios de 0 a $2^n - 1$ en una tabla.
2. Rotule las salidas de compuertas selectas con símbolos arbitrarios.
3. Obtenga la tabla de verdad para las salidas de aquellas compuertas que son función únicamente de las variables de entrada.
4. Obtenga la tabla de verdad para las salidas de aquellas compuertas que son función de valores previamente definidos, hasta llenar las columnas de todas las salidas.

Este proceso se ilustra empleando el circuito de la figura 4-2. En la tabla 4-1, formamos las ocho posibles combinaciones de las tres variables de entrada. La tabla de verdad para F_2 se de-

Tabla 4-1
Tabla de verdad para el diagrama lógico de la figura 4-2

A	B	C	F_2	F'_2	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

www.FreeLibros.me

termina directamente de los valores de A, B y C, siendo F_2 igual a 1 para cualquier combinación que tiene dos o tres entradas iguales a 1. La tabla de verdad para F'_2 es el complemento de F_2. Las tablas de verdad para T_1 y T_2 son las funciones OR y AND de las variables de entrada, respectivamente. Los valores para T_3 se deducen de T_1 y F'_2: T_3 es igual a 1 cuando tanto T_1 como F'_2 son 1, e igual a 0 en los demás casos. Por último, F_1 es igual a 1 para aquellas combinaciones en las que T_2 o T_3, o ambas, son 1. Una inspección de las combinaciones de A, B, C, F_1 y F_2 en la tabla de verdad revela que es idéntica a la tabla de verdad del sumador completo que se da en la sección 4-4 para x, y, z, S y C, respectivamente.

Otra forma de analizar un circuito combinacional es efectuando simulación lógica. En la sección 4-11 ilustraremos la simulación lógica y verificación del circuito de la figura 4-2 empleando Verilog HDL. (Véase el ejemplo HDL 4-10.)

4-3 PROCEDIMIENTO DE DISEÑO

El diseño de circuitos combinacionales parte de la especificación del problema y culmina en un diagrama lógico de circuitos o un conjunto de funciones booleanas a partir de las cuales se puede obtener el diagrama lógico. El procedimiento implica los pasos siguientes:

1. De las especificaciones del circuito, deduzca el número requerido de entradas y salidas; asigne un símbolo a cada una.
2. Deduzca la tabla de verdad que define la relación requerida entre las entradas y las salidas.
3. Obtenga las funciones booleanas simplificadas para cada salida en función de las variables de entrada.
4. Dibuje el diagrama lógico y verifique que el diseño sea correcto.

La tabla de verdad de un circuito combinacional consta de columnas de entrada y columnas de salida. Las columnas de entrada se obtienen de los 2^n números binarios para las n variables de entrada. Los valores binarios de las salidas se deducen de las especificaciones planteadas. Las funciones de salida especificadas en la tabla de verdad dan la definición exacta del circuito combinacional. Es importante interpretar correctamente las especificaciones verbales en la tabla de verdad. Tales especificaciones suelen ser incompletas, y cualquier interpretación errónea podría dar pie a una tabla de verdad incorrecta.

Las funciones binarias de salida enumeradas en la tabla de verdad se simplifican con cualquier método disponible, como manipulación algebraica, el método de mapa o un programa de simplificación para computadora. En muchos casos habrá diversas expresiones simplificadas para escoger. En cada aplicación dada, ciertos criterios servirán como guía para escoger una implementación. Un diseño práctico debe tomar en cuenta restricciones como el número de compuertas, el número de entradas de una compuerta, el tiempo de propagación de la señal a través de las compuertas, el número de interconexiones, las limitaciones de la corriente que proporciona cada compuerta y diversos criterios adicionales que es preciso considerar al diseñar con circuitos integrados. Puesto que la importancia de cada restricción depende de la aplicación específica, es difícil hacer recomendaciones generales acerca de lo que constituye una implementación aceptable. En la mayoría de los casos, la simplificación comienza por satisfacer un objetivo elemental, como producir las funciones booleanas simplificadas en una forma estándar, y luego efectúa otros pasos para cumplir con otros criterios de desempeño.

www.FreeLibros.me

Ejemplo de conversión de código

La disponibilidad de una gran variedad de códigos para los mismos elementos discretos de información hace que diferentes sistemas digitales usen códigos distintos. A veces es necesario usar la salida de un sistema como entrada de otro, y hay que insertar un circuito de conversión entre los dos sistemas si cada uno usa un código distinto para la misma información. Así pues, un convertidor de código es un circuito que hace compatibles a los dos sistemas aunque cada uno utilice un código binario distinto.

Para convertir del código binario A al código binario B, las líneas de entrada deberán proporcionar la combinación de elementos que especifica el código A, y las líneas de salida deberán generar las combinaciones de bits correspondientes del código B. Un circuito combinacional efectúa esta transformación con compuertas lógicas. Ilustraremos el procedimiento de diseño con un ejemplo que convierte el código BCD (decimal codificado en binario) en código exceso-3 para los dígitos decimales.

Las combinaciones de bits asignadas a los códigos BCD y exceso-3 se incluyen en la tabla 1-5 (sección 1-7). Puesto que ambos códigos usan cuatro bits para representar un dígito decimal, deberá haber cuatro variables de entrada y cuatro variables de salida. Designaremos a las primeras con A, B, C y D, y a las variables de salida, con w, x, y y z. La tabla de verdad que relaciona las variables de entrada y de salida se presenta en la tabla 4-2. Las combinaciones de bits para las entradas y sus salidas correspondientes se obtienen directamente de la sección 1-7. Cabe señalar que cuatro variables binarias pueden tener 16 combinaciones de bits, pero sólo 10 de ellas se presentan en la tabla de verdad. Las otras seis son combinaciones indiferentes. Esos valores carecen de significado en BCD y suponemos que nunca se presentarán. Por tanto, es posible asignar a las variables de salida 1 o 0, lo que produzca un circuito más simple.

Se han trazado los mapas de la figura 4-3 a fin de obtener funciones booleanas simplificadas para las salidas. Cada mapa representa una de las cuatro salidas del circuito en función de las cuatro variables de entrada. Los unos dentro de los cuadrados se obtienen de los minitér-

Tabla 4-2
Tabla de verdad para el ejemplo de conversión de código

Entrada BCD				Salida código exceso-3			
A	*B*	*C*	*D*	*w*	*x*	*y*	*z*
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

www.FreeLibros.me

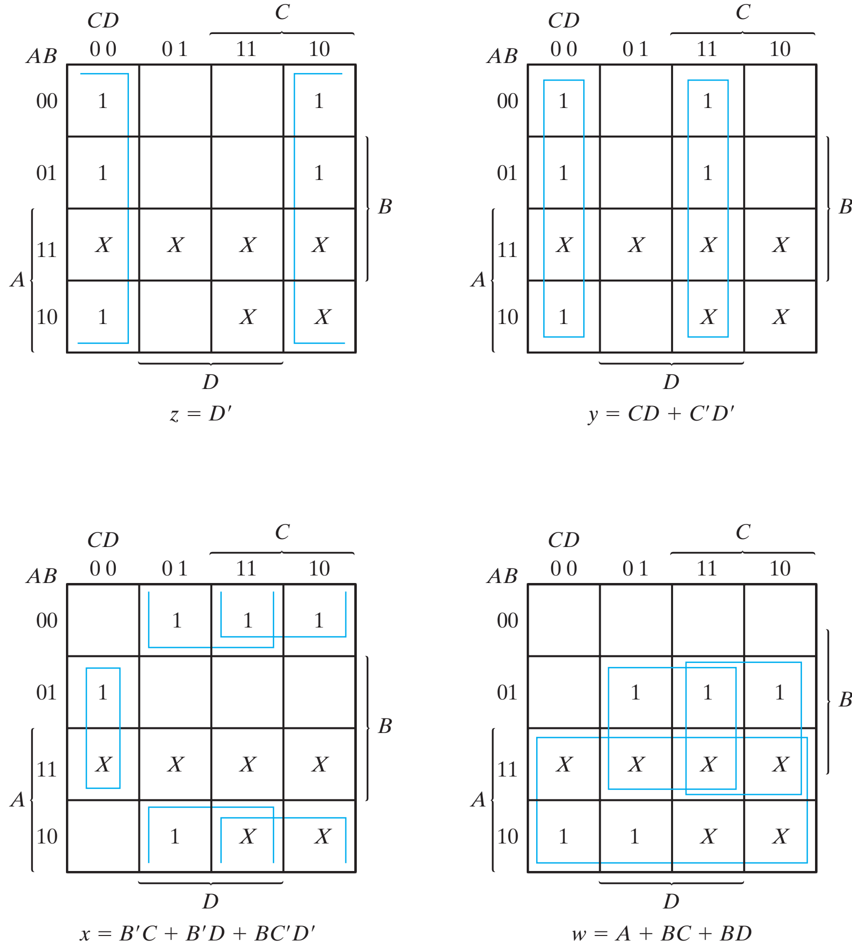


FIGURA 4-3
Mapas para el convertidor de código BCD a exceso-3

minos que hacen que la salida sea 1, y se obtienen de la tabla de verdad examinando las columnas de salida una por una. Por ejemplo, la columna de la salida *z* tiene cinco unos; por tanto, el mapa de *z* tiene cinco unos, cada uno en el cuadrado correspondiente al minitérmino que hace que *z* sea 1. Los seis minitérminos indiferentes, del 10 al 15, se han marcado con *X*. Bajo el mapa de cada variable se da una posible forma de simplificar la función en forma de suma de productos.

Es posible obtener un diagrama lógico de dos niveles directamente de las expresiones booleanas deducidas de los mapas. Hay otras posibilidades para un diagrama lógico que implemente este circuito. Las expresiones obtenidas en la figura 4-3 podrían manipularse algebraicamente con el fin de usar compuertas comunes para dos o más salidas. Esta manipu-

www.FreeLibros.me

lación que se presenta a continuación, ilustra la flexibilidad que se obtiene con sistemas de múltiples salidas implementados con tres o más niveles de compuertas:

$$
\begin{aligned}
z &= D' \\
y &= CD + C'D' = CD + (C + D)' \\
x &= B'C + B'D + BC'D' = B'(C + D) + BC'D' \\
&= B'(C + D) + B(C + D)' \\
w &= A + BC + BD = A + B(C + D)
\end{aligned}
$$

El diagrama lógico que implementa estas expresiones aparece en la figura 4-4. Observe que se ha usado la compuerta OR cuya salida es $C + D$ para implementar parcialmente cada una de las tres salidas.

Sin contar los inversores de entradas, la implementación en forma de suma de productos requiere siete compuertas AND y tres compuertas OR. La implementación de la figura 4-4 requiere cuatro compuertas AND, cuatro compuertas OR y un inversor. Si sólo se cuenta con las entradas normales, la primera implementación requerirá inversores para las variables *B*, *C* y *D*, y la segunda, para las variables *B* y *D*.

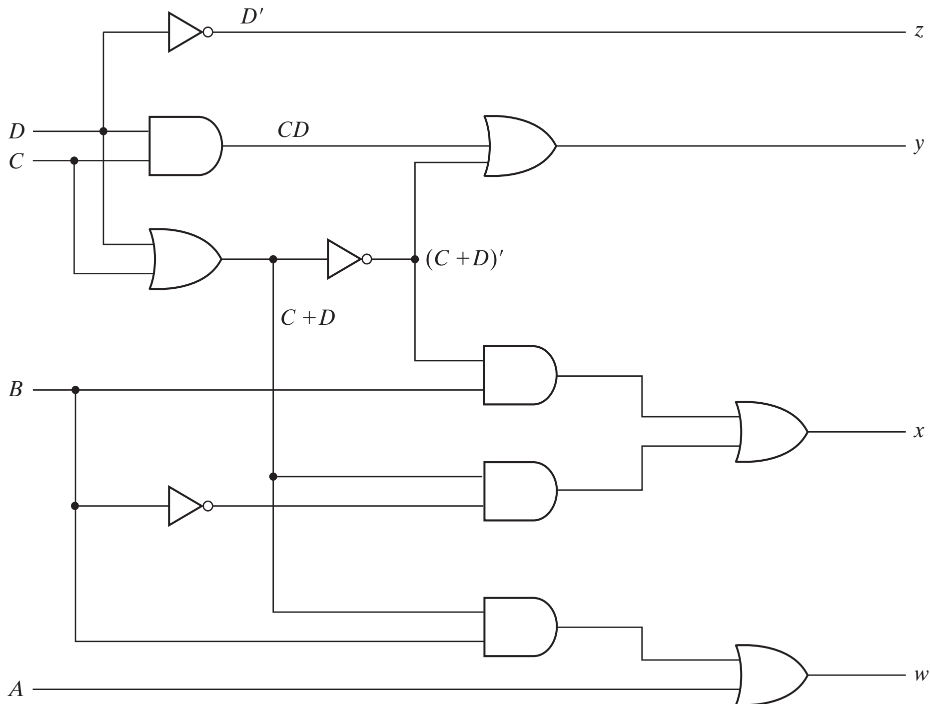


FIGURA 4-4
Diagrama lógico para el convertidor de código BCD a exceso-3

www.FreeLibros.me

4-4 SUMADOR-RESTADOR BINARIO

Las computadoras digitales efectúan diversas tareas de procesamiento de información. Entre esas funciones están las operaciones aritméticas. La operación aritmética más básica es la suma de dos dígitos binarios. Esta suma simple consiste en cuatro posibles operaciones elementales: $0 + 0 = 0$, $0 + 1 = 1$, $1 + 0 = 1$ y $1 + 1 = 10$. Las primeras tres operaciones producen una suma de un dígito, pero cuando ambos bits sumandos son 1, la suma binaria consta de dos dígitos. El bit más significativo de este resultado se denomina *acarreo* (*carry*, en inglés). Cuando ambos sumandos contienen más dígitos significativos, el acarreo obtenido de la suma de dos bits se suma al siguiente par más alto de bits significativos. Un circuito combinacional que realiza la suma de dos bits se denomina *semisumador;* uno que realiza la suma de tres bits (dos bits significativos y un acarreo previo) es un *sumador completo*. Los nombres de los circuitos provienen del hecho de que es posible usar dos semisumadores para implementar un sumador completo.

Un sumador-restador binario es un circuito combinacional que realiza las operaciones aritméticas de suma y resta con números binarios. Desarrollaremos este circuito utilizando un diseño jérárquico. Primero diseñaremos el semisumador, y a partir de él desarrollaremos el sumador completo. La conexión de n sumadores completos en cascada produce un sumador binario para números de n bits. Incluiremos el circuito de resta con la ayuda de un circuito complementador.

Semisumador

Por la descripción verbal del semisumador, se sabe que este circuito necesita dos entradas binarias y dos salidas binarias. Las variables de entrada designan los bits sumandos; las de salida, la suma y el acarreo. Asignaremos los símbolos x y y a las dos entradas y S (de suma) y C (de *carry*) a las salidas. La tabla de verdad del semisumador se presenta en la tabla 4-3. La salida C es 1 sólo cuando ambas entradas son 1. La salida S representa el bit menos significativo de la suma.

Las funciones booleanas simplificadas para las dos salidas se obtienen directamente de la tabla de verdad. Las expresiones simplificadas en suma de productos son

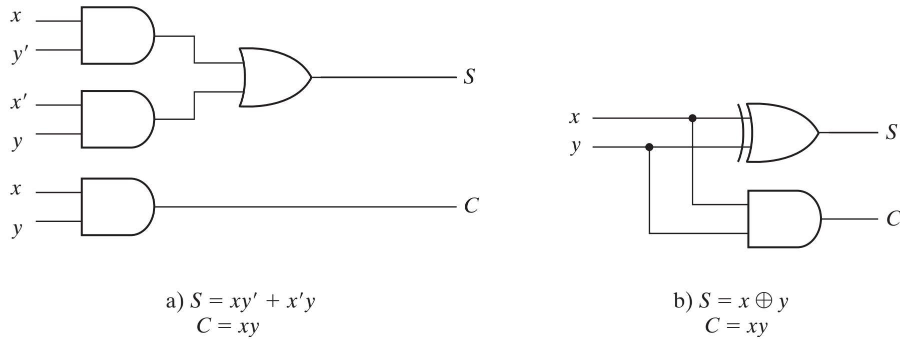
$$S = x'y + xy'$$
$$C = xy$$

El diagrama lógico del semisumador implementado como suma de productos se observa en la figura 4-5a). También se puede implementar con un OR exclusivo y una compuerta AND, como se indica en la figura 4-5b). Esta forma se utiliza para mostrar cómo dos semisumadores sirven para construir un sumador completo.

Tabla 4-3
Semisumador

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

www.FreeLibros.me



a) $S = xy' + x'y$
$C = xy$

b) $S = x \oplus y$
$C = xy$

FIGURA 4-5
Implementación de semisumador

Sumador completo

Un sumador completo es un circuito combinacional que forma la suma aritmética de tres bits. Tiene tres entradas y dos salidas. Dos de las variables de entrada, denotadas por x y y, representan los dos bits significativos que se sumarán. La tercera entrada, z, representa el acarreo de la posición significativa inmediata inferior. Se requieren dos salidas porque la suma aritmética de tres dígitos binarios puede tener valores entre 0 y 3, y el 2 o el 3 binarios requieren dos dígitos. Las dos salidas se designan otra vez con los símbolos S y C. La variable binaria S da el valor del bit menos significativo de la suma. La variable binaria C da el acarreo de salida. La tabla de verdad del sumador completo se presenta en la tabla 4-4. Las ocho filas bajo las variables de entrada dan todas las posibles combinaciones de las tres variables. Las variables de salida se determinan a partir de la suma aritmética de los bits de entrada. Si todos los bits de entrada son 0, la salida es 0. La salida S es 1 cuando sólo una entrada es 1 o cuando las tres entradas son 1. La salida C da un acarreo de 1 si dos o tres entradas son 1.

Los bits de entrada y de salida del circuito combinacional tienen diferentes interpretaciones en las distintas etapas del problema. Físicamente, las señales binarias de las entradas se consideran dígitos binarios que deben sumarse aritméticamente para formar una salida de dos dígitos. Por otra parte, los mismos valores binarios se consideran variables de funciones booleanas cuando se expresan en la tabla de verdad o cuando el circuito se implementa con compuertas

Tabla 4-4
Sumador completo

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

www.FreeLibros.me

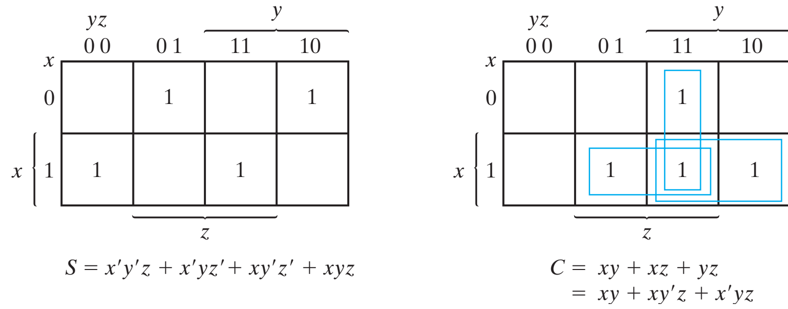


FIGURA 4-6
Mapas para el sumador completo

lógicas. Los mapas para las salidas del sumador completo aparecen en la figura 4-6. Las expresiones simplificadas son

$$S = x'y'z + x'yz' + xy'z' + xyz$$
$$C = xy + xz + yz$$

El diagrama lógico para el sumador completo implementado en forma de suma de productos se muestra en la figura 4-7. También puede implementarse con dos semisumadores y una compuerta OR, como se indica en la figura 4-8. La salida *S* del segundo semisumador es el OR exclusivo de *z* y la salida del primer semisumador, lo que da

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

La salida de acarreo es

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

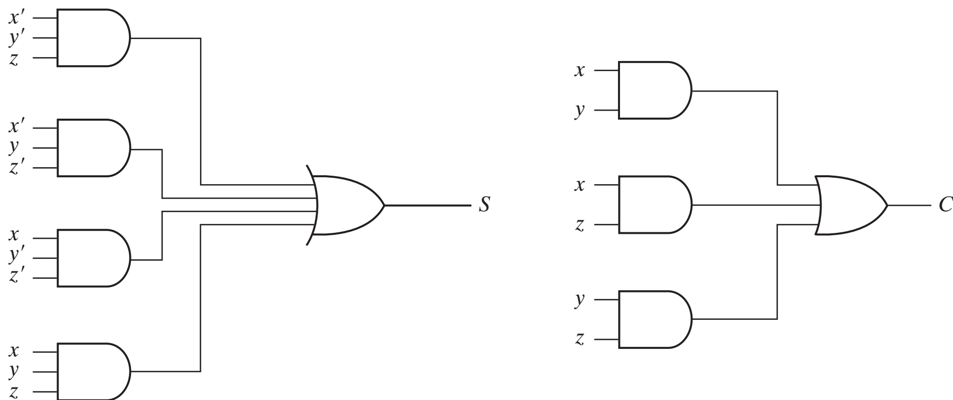


FIGURA 4-7
Implementación de un sumador completo como suma de productos

www.FreeLibros.me

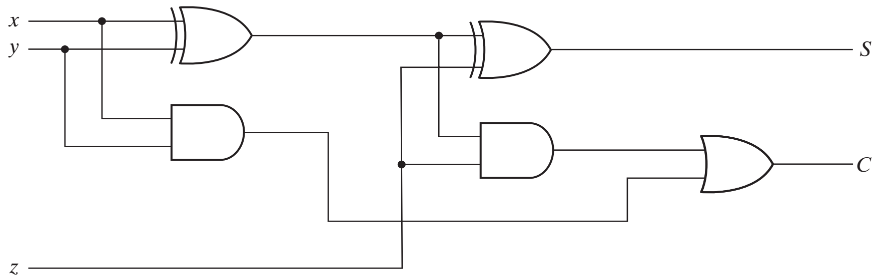


FIGURA 4-8
Implementación de un sumador completo con dos semisumadores y una compuerta OR

Sumador binario

Un sumador binario es un circuito digital que produce la suma aritmética de dos números binarios. Es posible construirlo con sumadores completos dispuestos en cascada, conectando el acarreo de salida de cada sumador completo al acarreo de entrada del siguiente sumador completo de la cadena. La figura 4-9 muestra la interconexión de cuatro circuitos sumadores completos (SC) para formar un sumador binario de cuatro bits con acarreo rizado. Los bits de los sumandos A y B se designan con subíndices de izquierda a derecha; el subíndice 0 denota el bit menos significativo. Los acarreos se conectan en una cadena a través de los sumadores completos. El acarreo de entrada del sumador es C_0 y se propaga a través de los sumadores completos hasta el acarreo de salida C_4. Las salidas S generan los bits de suma requeridos. Un sumador de n bits requiere n sumadores completos con cada acarreo de salida conectado al acarreo de entrada del siguiente sumador completo de orden superior.

Para ilustrar esto con un ejemplo específico, consideremos los dos números binarios $A = 1011$ y $B = 0011$. Su suma $S = 1110$ se forma con el sumador de cuatro bits así:

Subíndice i:	**3**	**2**	**1**	**0**	
Acarreo de entrada	0	1	1	0	C_i
Sumando	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}

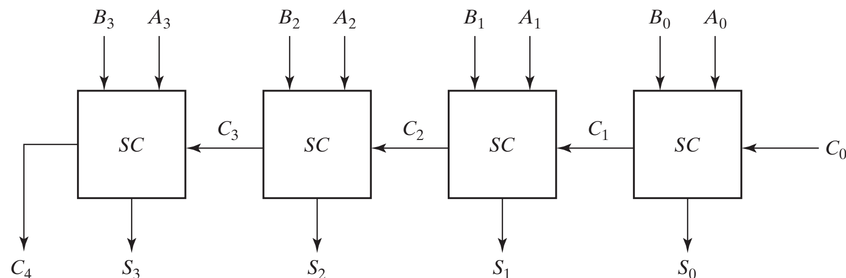


FIGURA 4-9
Sumador de cuatro bits

www.FreeLibros.me

Los bits se suman con sumadores completos, comenzando por la posición menos significativa (subíndice 0) para formar el bit de suma y el bit de acarreo. El acarreo de entrada C_0 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 de la siguiente posición significativa a la izquierda. Así, los bits de la suma se generan comenzando por la posición de la extrema derecha y están disponibles tan pronto como se genera el bit de acarreo anterior. Se deben generar todos los acarreos para que los bits de suma correctos aparezcan en las salidas.

El sumador de cuatro bits es un ejemplo representativo de un componente estándar. Se utiliza en muchas aplicaciones que implican operaciones aritméticas. Observe que el diseño de este circuito empleando el método clásico requeriría una tabla de verdad con $2^9 = 512$ entradas, ya que el circuito tiene nueve entradas. Al usar un método iterativo de conectar en cascada una función estándar, es posible obtener una implementación sencilla y directa.

Propagación del acarreo

La suma de dos números binarios en paralelo implica que todos los bits de los sumandos están disponibles al mismo tiempo para efectuar el cálculo. Como en cualquier circuito combinacional, la señal tiene que propagarse a través de las compuertas para que la salida correcta (la suma) esté disponible en las terminales de salida. El tiempo total de propagación es igual al retardo de propagación de una compuerta representativa multiplicado por el número de niveles de compuertas del circuito. El retardo de propagación más largo en un sumador es el tiempo que el acarreo tarda en propagarse a través de los sumadores completos. Dado que cada bit de la suma depende del valor del acarreo de entrada, el valor de S_i en cualquier etapa dada del sumador alcanzará su valor final de estado estable sólo hasta que el acarreo de entrada se haya propagado a esa etapa. Consideremos la salida S_3 de la figura 4-9. Las entradas A_3 y B_3 están disponibles tan pronto como se aplican señales de entrada al sumador. Sin embargo, el acarreo de entrada C_3 no se estabiliza en su valor final sino hasta después de que se cuenta con C_2 de la etapa anterior. Asimismo, C_2 tiene que esperar a C_1, y así hasta C_0. Por tanto, no será sino hasta que el acarreo se propague en rizo a través de todas las etapas cuando la última salida S_3 y el último acarreo C_4 se estabilicen en su valor final correcto.

Es posible calcular el número de niveles de compuerta para la propagación del acarreo a partir del circuito del sumador completo. La figura 4-10 reproduce otra vez el circuito. Las variables de entrada y salida llevan el subíndice i para denotar una etapa representativa del

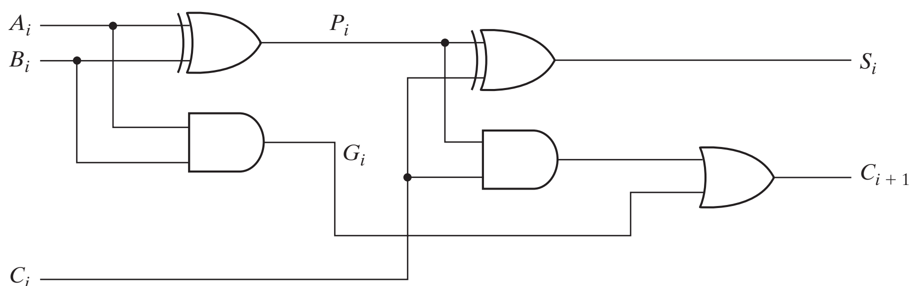


FIGURA 4-10
Sumador completo en el que se indican P y G

www.FreeLibros.me

sumador. Las señales en P_i y G_i se estabilizan en sus valores de estado estable después de que se propagan a través de sus respectivas compuertas. Todos los sumadores completos tienen en común estas dos señales y sólo dependen de los bits de entrada de los sumandos. 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 OR, lo que constituye dos niveles de compuertas. Si hay cuatro sumadores completos en el sumador, el acarreo de salida C_4 tendrá $2 \times 4 = 8$ niveles de compuerta desde C_0 hasta C_4. En el caso de un sumador de n bits, el acarreo tendrá que propagarse a través de $2n$ niveles de compuertas desde la entrada hasta la salida.

El tiempo de propagación del acarreo es un factor que limita la rapidez con que se suman dos números. Aunque el sumador, o cualquier circuito combinacional, siempre tendrá algún valor en sus terminales de salida, esos valores no serán correctos si no se da a las señales el tiempo suficiente para propagarse a través de las compuertas conectadas entre las entradas y las salidas. Puesto que todas las demás operaciones aritméticas se implementan con sumas sucesivas, el tiempo consumido durante el proceso de adición es crucial. Una solución obvia para reducir el retardo de propagación del acarreo es utilizar compuertas más rápidas con menor retardo. Sin embargo, los circuitos físicos tienen un límite en este sentido. Otra solución sería aumentar la complejidad del equipo de modo tal que el retardo del acarreo se reduzca. Existen varias técnicas para reducir el tiempo de propagación del acarreo en un sumador paralelo. La técnica más ampliamente utilizada se vale del principio de *acarreo anticipado*.

Considere el circuito del sumador completo que se aprecia en la figura 4-10. Si definimos dos nuevas variables binarias

$$P_i = A_i \oplus B_i$$
$$G_i = A_i B_i$$

la suma y el acarreo se expresarán así:

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_i C_i$$

G_i se llama *acarreo generado* y produce un acarreo de 1 si tanto A_i como B_i son 1, independientemente del acarreo de entrada C_i. P_i se llama *acarreo propagado* porque es el término asociado a la propagación del acarreo de C_i a C_{i+1}.

Ahora escribiremos las funciones booleanas para los acarreos de salida de cada etapa y sustituiremos C_i por el valor obtenido de las ecuaciones anteriores:

$$C_0 = \text{acarreo de entrada}$$
$$C_1 = G_0 + P_0 C_0$$
$$C_2 = G_1 + P_1 C_1 = G_1 + P_1(G_0 + P_0 C_0) = G_1 + P_1 G_0 + P_1 P_0 C_0$$
$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$$

Puesto que la función booleana para cada acarreo de salida se expresa en forma de suma de productos, cada función se puede implementar con un nivel de compuertas AND seguido de una compuerta OR (o con dos niveles de NAND). Las tres funciones booleanas de C_1, C_2 y C_3 se

www.FreeLibros.me

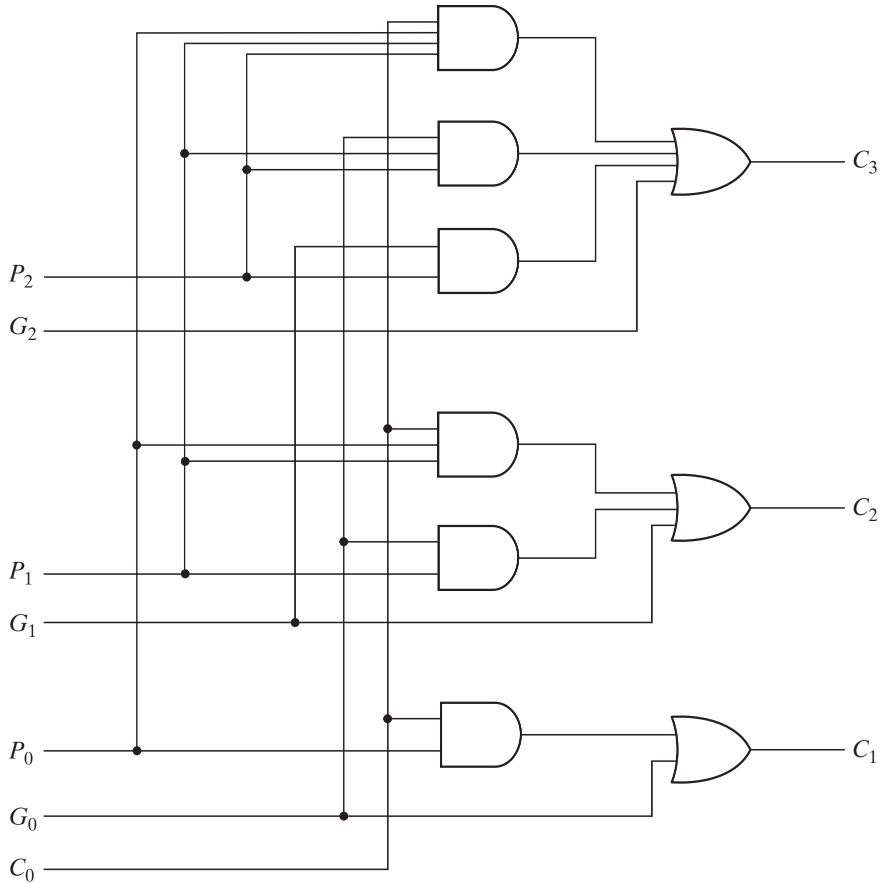


FIGURA 4-11
Diagrama lógico del generador de acarreo anticipado

implementan en el generador de acarreo anticipado que se observa en la figura 4-11. Advierta que C_3 no tiene que esperar a que C_2 y C_1 se propaguen; de hecho, C_3 se propaga al mismo tiempo que C_1 y C_2.

La construcción de un sumador de cuatro bits con un esquema de acarreo anticipado se representa en la figura 4-12. Cada salida de suma requiere dos compuertas OR exclusivo. La salida de la primera compuerta OR exclusivo genera la variable P_i y la compuerta AND genera la variable G_i. Los acarreos se propagan mediante el generador de acarreo anticipado (similar al de la figura 4-11) y se aplican como entradas a la segunda compuerta OR exclusivo. Todos los acarreos de salida se generan después de un retardo de dos niveles de compuertas. Así, las salidas S_1 a S_3 tienen el mismo tiempo de retardo por propagación. No se muestra el circuito de dos niveles para el acarreo de salida C_4. Este circuito se deduce fácilmente empleando el método de sustitución de ecuaciones.

www.FreeLibros.me

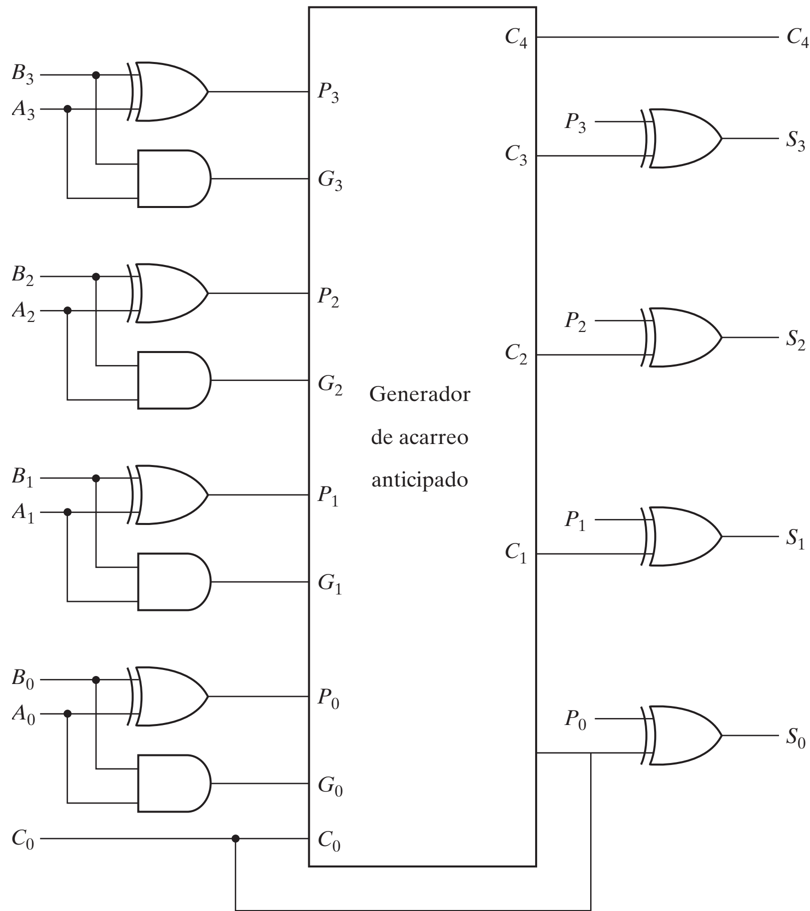


FIGURA 4-12
Sumador de cuatro bits con acarreo anticipado

Restador binario

La forma más conveniente de efectuar la resta de números binarios sin signo es utilizando complementos, como se explicó en la sección 1-5. Recuerde que la resta $A - B$ se efectúa obteniendo el complemento a dos de B y sumándolo a A. El complemento a dos se obtiene calculando el complemento a uno y sumando 1 al par de bits menos significativo. El complemento a uno se implementa con inversores, y el 1 se suma a través del acarreo de entrada.

El circuito para restar $A - B$ consiste en un sumador con inversores colocados entre cada entrada de datos B y la entrada correspondiente del sumador completo. El acarreo de entrada C_0 debe ser igual a 1 al restar. La operación se convierte entonces en A más el complemento a uno de B más 1. Esto es igual a A más el complemento a dos de B. En el caso de números sin signo, esto da $A - B$ si $A \geq B$, o el complemento a dos de $(B - A)$ si $A < B$. En el caso

www.FreeLibros.me

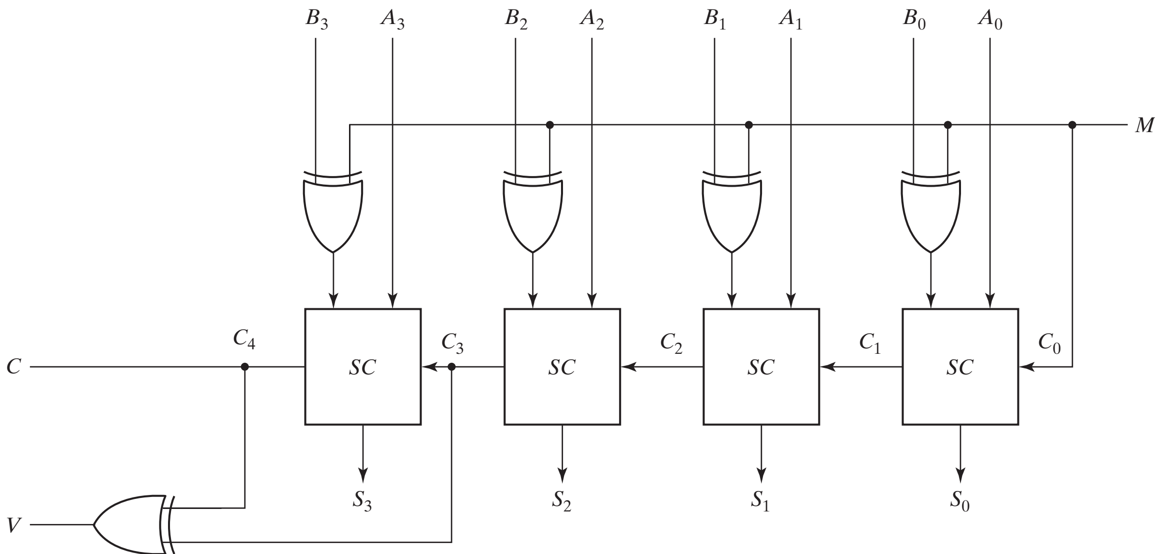


FIGURA 4-13
Sumador-restador de cuatro bits

de números con signo, el resultado es $A - B$, siempre que no haya desbordamiento. (Véase la sección 1-6.)

Las operaciones de suma y resta se pueden combinar en un solo circuito que tiene un sumador binario compartido. Esto se hace incluyendo una compuerta OR exclusivo con cada sumador completo. En la figura 4-13 se reproduce un circuito sumador-restador de cuatro bits. La entrada de modo M controla la operación. Si $M = 0$, el circuito es un sumador; y si $M = 1$, el circuito se convierte en un restador. Cada compuerta OR exclusivo recibe la entrada M y una de las entradas de B. Cuando $M = 0$, tenemos $B \oplus 0 = B$. Los sumadores completos reciben el valor de B, el acarreo de entrada es 0, y el circuito efectúa A más B. Cuando $M = 1$, tenemos $B \oplus 1 = B'$ y $C_0 = 1$. Todas las entradas de B se complementan y se suma un 1 a través del acarreo de entrada. El circuito efectúa la operación A más el complemento a dos de B. (El OR exclusivo con salida V es para detectar un desbordamiento.)

Vale la pena señalar que los números binarios en el sistema de complemento con signo se suman y restan con las mismas reglas básicas de suma y resta que los números sin signo. Por tanto, las computadoras sólo necesitan un circuito compartido en hardware para manejar ambos tipos de aritmética. El usuario o el programador deberá interpretar los resultados de tales sumas o restas de forma distinta, dependiendo de si se supone que los números tienen signo o no.

Desbordamiento

Cuando dos números de n dígitos cada uno se suman y la suma ocupa $n + 1$ dígitos, decimos que hubo un desbordamiento. Esto se cumple con los números binarios y decimales, con y sin signo. Cuando sumamos con lápiz y papel, el desbordamiento no causa problemas, porque la anchura del papel no limita la escritura de la suma. En las computadoras digitales el desbordamiento sí representa un problema porque el número de bits que contienen al número es finito, y un resultado que contiene $n + 1$ bits no cabe. Por ello, muchas computadoras detectan

www.FreeLibros.me

cuando ocurre un desbordamiento, y "encienden" un flip-flop específico que el usuario puede verificar.

La detección de un desbordamiento después de la suma de dos números binarios depende de si se considera que los números tienen signo o no. Cuando se suman dos números sin signo, el desbordamiento se detecta en el acarreo final de la posición más significativa. En el caso de números con signo, el bit de la extrema izquierda siempre representa al signo y los números negativos están en forma de complemento a dos. Cuando se suman dos números con signo, el bit de signo se trata como parte del número y el acarreo final no indica un desbordamiento.

No existe desbordamiento después de una suma si un número es positivo y el otro es negativo, ya que la suma de un número positivo y uno negativo produce un resultado más pequeño que el mayor de los dos números originales. Podría haber un desbordamiento si los dos números sumados son ambos positivos o ambos negativos. Para entender esto, considere el ejemplo siguiente. Dos números binarios con signo, $+70$ y $+80$ se almacenan en dos registros de ocho bits. El intervalo de números al que cada registro puede dar cabida es del $+127$ binario al -128 binario. Puesto que la suma de los dos números es $+150$, excederá la capacidad de un registro de 8 bits. Esto se cumple si ambos números son positivos o negativos. A continuación mostramos las dos sumas en binario, junto con los últimos dos acarreos;

acarreos:	0 1		acarreos	1 0	
+70	0	1000110	−70	1	0111010
+80	0	1010000	−80	1	0110000
+150	1	0010110	−150	0	1101010

Observe que el resultado de ocho bits que debería haber sido positivo tiene un bit de signo negativo, y el resultado de ocho bits que debería haber sido negativo tiene un bit de signo positivo. En cambio, si tomamos el acarreo de salida de la posición de bit de signo como bit de signo del resultado, la respuesta de nueve bits así obtenida será correcta. Puesto que la respuesta no cabe en ocho bits, decimos que se ha presentado un desbordamiento.

Es posible detectar la condición de desbordamiento observando el acarreo que llega a la posición de bit de signo y el acarreo que sale de ella. Si los dos acarreos son distintos, ha habido un desbordamiento. Esto se hace evidente en los ejemplos, donde se muestran explícitamente los dos acarreos. Si estos dos acarreos se aplican a una compuerta OR exclusivo, se detectará un desbordamiento cuando la salida de esa compuerta sea 1. Para que este método funcione correctamente, es preciso calcular el complemento a dos obteniendo el complemento a uno y sumándole 1. Esto da cuenta de la condición en la que se complementa el número negativo máximo.

El circuito sumador-restador binario con salidas C y V se representa en la figura 4-13. Si se considera que los dos números binarios carecen de signo, el bit C detectará un acarreo después de la suma o un préstamo después de la resta. Si se considera que los números tienen signo, el bit V detectará un desbordamiento. Si $V = 0$ después de una suma o resta, querrá decir que no hubo desbordamiento y que el resultado de n bits es correcto. Si $V = 1$, el resultado de la operación tiene $n + 1$ bits, pero sólo los n bits de la derecha del número caben en el espacio disponible, así que ha habido un desbordamiento. El $(n + 1)$*ésimo* bit es el signo real, que ha sido desplazado de su posición.

www.FreeLibros.me

4-5 SUMADOR DECIMAL

Las computadoras o calculadoras que realizan operaciones aritméticas directamente en el sistema numérico decimal representan los números decimales codificados en binario. Un sumador de una computadora así deberá utilizar circuitos de aritmética que acepten números decimales codificados y presenten los resultados en el mismo código. En el caso de la suma binaria, basta con considerar un par de bits significativos más un acarreo previo. Un sumador decimal requiere como mínimo nueve entradas y cinco salidas, ya que se requieren cuatro bits para codificar cada dígito decimal y el circuito necesita un acarreo de entrada y uno de salida. Hay una amplia variedad de posibles circuitos sumadores digitales, dependiendo del código empleado para representar los dígitos decimales. Aquí consideraremos un sumador decimal para el código BCD. (Véase la sección 1-7.)

Sumador BCD

Consideremos la suma aritmética de dos dígitos decimales en BCD, junto con un acarreo de entrada de una etapa anterior. Puesto que ninguno de los dígitos de entrada es mayor que 9, la suma de salida no puede ser mayor que $9 + 9 + 1 = 19$, donde el 1 de la suma es el acarreo de entrada. Suponga que aplicamos dos dígitos BCD a un sumador binario de cuatro bits. El sumador formará la suma en *binario* y producirá un resultado entre 0 y 19. Estos números binarios se presentan en forma de lista en la tabla 4-5 y se rotulan con los símbolos K, Z_8, Z_4, Z_2 y Z_1. K es el acarreo, y los subíndices de Z representan los pesos 8, 4, 2 y 1 que se pueden asignar a los cuatro bits en el código BCD. Las columnas bajo "Suma binaria" presentan el valor

Tabla 4-5
Deducción 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

www.FreeLibros.me

binario que aparece en las salidas del sumador binario de cuatro bits. La suma de dos dígitos decimales se debe representar en BCD, así que deberá aparecer en la forma presentada en la columna bajo "Suma BCD". El problema consiste en encontrar una regla para convertir la suma binaria en la representación BCD correcta de la suma.

Al examinar el contenido de la tabla, será evidente que cuando la suma binaria es 1001 o menos, el número BCD correspondiente es idéntico, de modo que no es necesaria conversión alguna. Cuando la suma binaria es mayor que 1001, se obtiene una representación no válida en BCD. La suma de 6 binario (0110) a la suma binaria la convierte en la representación BCD correcta y también produce el acarreo de salida necesario.

El circuito de lógica que detecta la corrección necesaria se deduce de las entradas de la tabla. Es obvio que se necesita una corrección cuando la suma binaria tiene un acarreo de salida $K = 1$. Las otras seis combinaciones (1010 a 1111) que necesitan una corrección tienen un 1 en la posición Z_8. Para distinguirlas de los números binarios 1000 y 1001, que también tienen un 1 en la posición Z_8, especificamos además que Z_4 o Z_2 deben tener un 1. La condición para la corrección y el acarreo de salida se expresa mediante la función booleana

$$C = K + Z_8Z_4 + Z_8Z_2$$

Cuando $C = 1$, es necesario sumar 0110 a la suma binaria y generar un acarreo de salida para la etapa siguiente.

En la figura 4-14 se observa un sumador BCD que suma dos dígitos BCD y produce un dígito de suma en BCD. Primero se suman los dos dígitos decimales, junto con el acarreo de entrada, en el sumador de cuatro bits de la parte de arriba, para producir la suma binaria. Si el

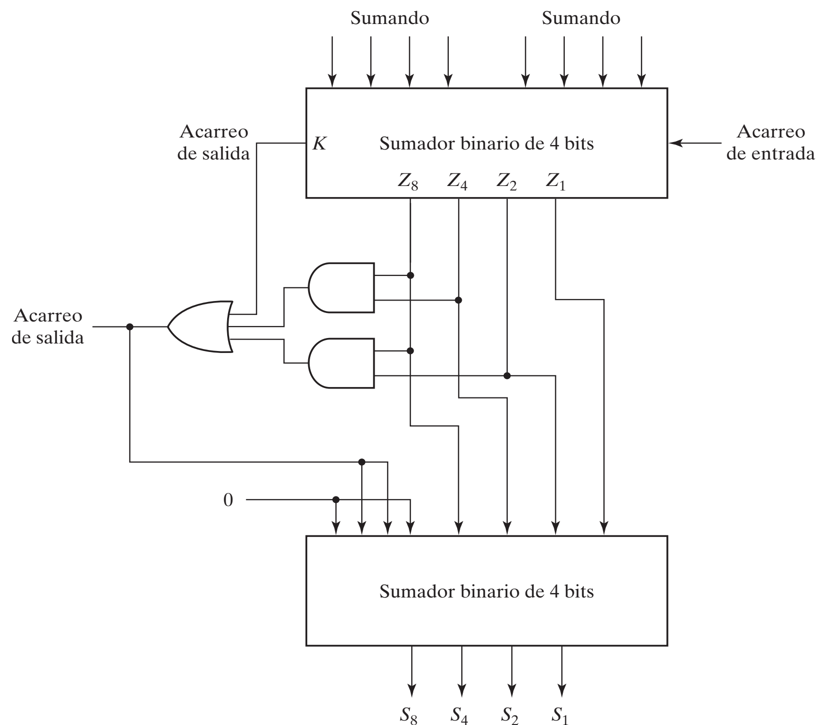


FIGURA 4-14
Diagrama de bloques de un sumador BCD

www.FreeLibros.me

acarreo de salida es 0, no se suma nada a la suma binaria. Si es 1, se suma 0110 binario a la suma binaria con el sumador de cuatro bits de la parte de abajo. El acarreo de salida generado por el sumador de abajo se desecha, pues proporciona información con que ya se cuenta en la terminal de acarreo de salida. Un sumador decimal en paralelo que suma *n* dígitos decimales necesita *n* etapas de sumador BCD. El acarreo de salida de una etapa deberá conectarse al acarreo de entrada de la siguiente etapa de orden superior.

4-6 MULTIPLICADOR BINARIO

La multiplicación de números binarios se efectúa igual que la de números decimales. El multiplicando se multiplica por cada bit del multiplicador, comenzando por el bit menos significativo. Cada una de estas multiplicaciones forma un producto parcial. Los productos parciales sucesivos se desplazan una posición a la izquierda. El producto final se obtiene sumando los productos parciales.

Para ver cómo puede implementarse un multiplicador binario con un circuito combinacional, consideremos la multiplicación de dos números de dos bits, como se muestra en la figura 4-15. Los bits del multiplicando son B_1 y B_0, los bits del multiplicador son A_1 y A_0, y el producto es $C_3C_2C_1C_0$. El primer producto parcial se forma multiplicando A_0 por B_1B_0. La multiplicación de dos bits como A_0 y B_0 produce 1 si ambos bits son 1; de lo contrario, produce 0. Esto es idéntico a la operación AND. Por tanto, el producto parcial puede implementarse con compuertas AND como se indica en el diagrama. El segundo producto parcial se forma multiplicando A_1 por B_1B_0 y se desplaza una posición a la izquierda. Los dos productos parciales se suman con dos circuitos de semisumador (SS). Por lo regular, los productos parciales tienen más bits, y ello obliga a usar sumadores completos para obtener la suma de los productos parciales. Observe que el bit menos significativo del producto no tiene que pasar por un sumador porque se forma con la salida de la primera compuerta AND.

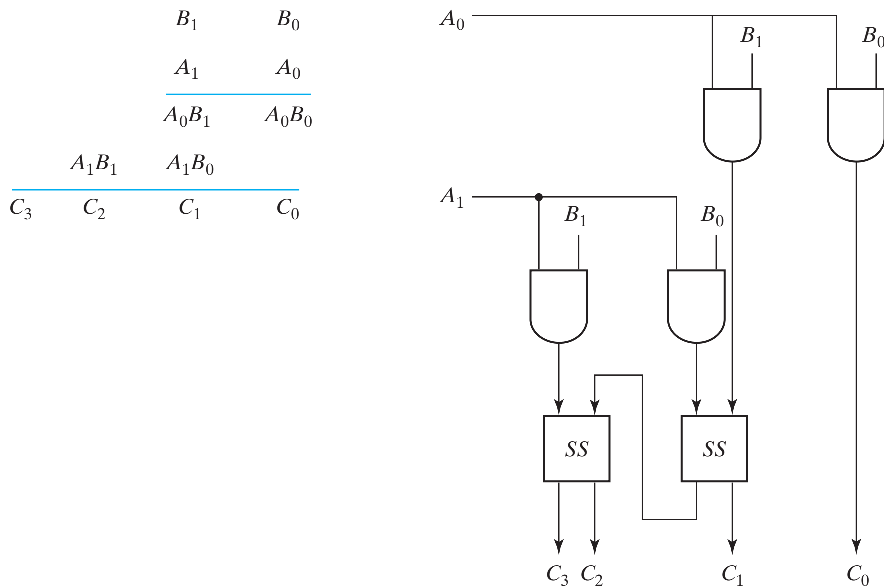


FIGURA 4-15
Multiplicador binario de dos bits por dos bits

www.FreeLibros.me

Podemos construir de forma similar un multiplicador binario de más bits con circuitos combinacionales. Se obtiene el AND de un bit del multiplicador y cada bit del multiplicando en tantos niveles como haya bits en el multiplicador. La salida binaria de cada nivel de compuertas AND se suma al producto parcial del nivel anterior para formar un nuevo producto parcial. El último nivel genera el producto. Si el multiplicador tiene J bits y el multiplicando tiene K bits, necesitaremos $(J \times K)$ compuertas AND y $(J - 1)$ sumadores de K bits para obtener un producto de $J + K$ bits.

Como segundo ejemplo, consideremos un circuito multiplicador que multiplica un número binario de cuatro bits por uno de tres bits. Representaremos el multiplicando con $B_3B_2B_1B_0$, y el multiplicador, con $A_2A_1A_0$. Puesto que $K = 4$ y $J = 3$, necesitaremos 12 compuertas AND y dos sumadores de cuatro bits para obtener un producto de siete bits. El diagrama lógico del multiplicador se presenta en la figura 4-16.

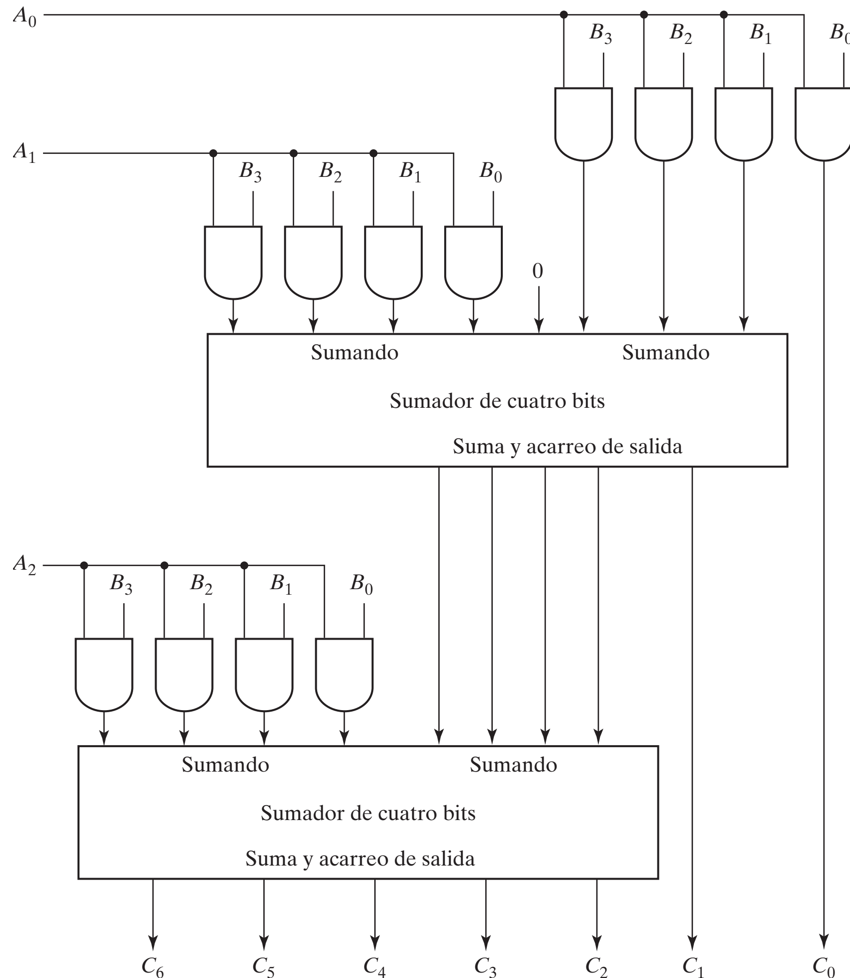


FIGURA 4-16
Multiplicador binario de 4 bits por 3 bits

www.FreeLibros.me

4-7 COMPARADOR DE MAGNITUDES

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 magnitudes* es un circuito combinacional que compara dos números, A y B, y determina sus magnitudes relativas. El resultado de la comparación se especifica con tres variables binarias que indican si $A > B$, $A = B$ o $A < B$.

El circuito para comparar dos números de n bits tiene 2^{2n} entradas en la tabla de verdad y resulta difícil de manejar incluso con $n = 3$. Por otra parte, como el lector seguramente sospechará, los circuitos comparadores poseen cierto grado de regularidad. Las funciones digitales que poseen una regularidad inherente bien definida por lo regular se diseñan empleando un procedimiento algorítmico. Un algoritmo es un procedimiento que especifica un conjunto finito de pasos que, si se siguen, producen la solución de un problema. Ilustraremos este método deduciendo un algoritmo para el diseño de un comparador de magnitudes de cuatro bits.

El algoritmo es una aplicación directa del procedimiento que una persona sigue para comparar las magnitudes relativas de dos números. Consideremos dos números, A y B, de cuatro dígitos cada uno. Escribiremos los coeficientes de los números del más al menos significativo:

$$A = A_3A_2A_1A_0$$
$$B = B_3B_2B_1B_0$$

Cada letra con subíndice representa uno de los dígitos del número. Los dos números son iguales si todos los pares de dígitos significativos son iguales; $A_3 = B_3$ y $A_2 = B_2$ y $A_1 = B_1$ y $A_0 = B_0$. Si los números son binarios, los dígitos son 1 o 0, y la relación de igualdad de cada par de bits se expresa lógicamente con una función OR exclusivo así:

$$x_i = A_iB_i + A_i'B_i' \qquad \text{para } i = 0, 1, 2, 3$$

donde $x_i = 1$ únicamente si los dos bits de la posición i son iguales (es decir, si ambos son 1 o ambos son 0).

La igualdad de los dos números, A y B, se indica en un circuito combinacional con una variable binaria de salida que designaremos con el símbolo $(A = B)$. Esta variable binaria es 1 si los números de entrada, A y B, son iguales, y es 0 en caso contrario. Para que exista la condición de igualdad, las x_i variables deberán ser todas 1. Esto implica una operación AND de todas las variables:

$$(A = B) = x_3x_2x_1x_0$$

La variable *binaria* $(A = B)$ es 1 únicamente 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, comenzando por la posición más significativa. Si los dos dígitos son iguales, se comparará el siguiente par de dígitos menos significativos. Esta comparación continuará hasta encontrar un par de dígitos distintos. Si el dígito correspondiente de A es 1 y el de B es 0, concluimos que $A > B$. Si el dígito correspondiente de A es 0 y el de B es 1, sabemos que $A < B$. La comparación sucesiva se expresa lógicamente con las dos funciones booleanas

$$(A > B) = A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$$
$$(A < B) = A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1 + x_3x_2x_1A_0'B_0$$

Los símbolos $(A > B)$ y $(A < B)$ son variables de salida *binarias* que valen 1 cuando $A > B$ o $A < B$, respectivamente.

La implementación con compuertas de las tres variables de salida que acabamos de deducir es más sencilla de lo que parece porque implica cierta repetición. Las salidas de desigual-

www.FreeLibros.me

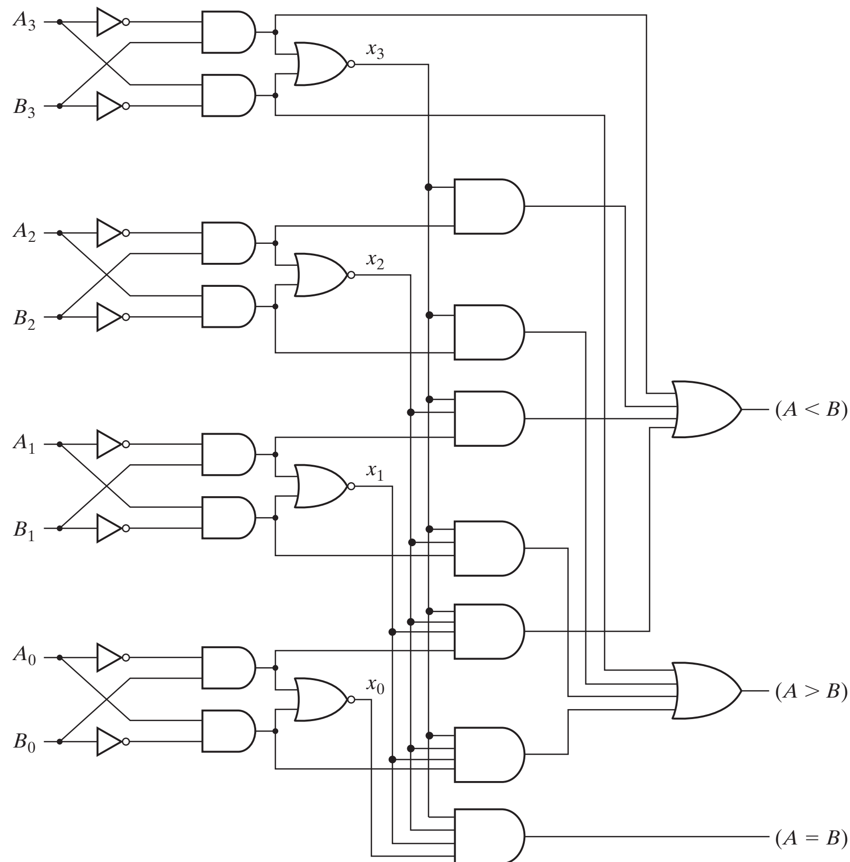


FIGURA 4-17
Comparador de magnitudes de cuatro bits

dad utilizan las mismas compuertas que generan la salida de igualdad. El diagrama lógico del comparador de magnitudes de cuatro bits se reproduce en la figura 4-17. Las cuatro salidas x se generan con circuitos NOR exclusivo y se aplican a una compuerta AND para dar la variable binaria de salida $(A = B)$. Las otras dos salidas utilizan las variables x para generar las funciones booleanas que presentamos antes. Ésta es una implementación multinivel y sigue un patrón regular. El procedimiento para obtener circuitos comparadores de magnitud para números binarios de más de cuatro bits se deduce fácilmente de este ejemplo.

4-8 DECODIFICADORES

En los sistemas digitales, las cantidades discretas de información se representan con códigos binarios. Un código binario de n bits puede 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 de salida distintas. Si la información codificada en n bits tiene combinaciones que no se usan, el decodificador podría tener menos de 2^n salidas.

www.FreeLibros.me

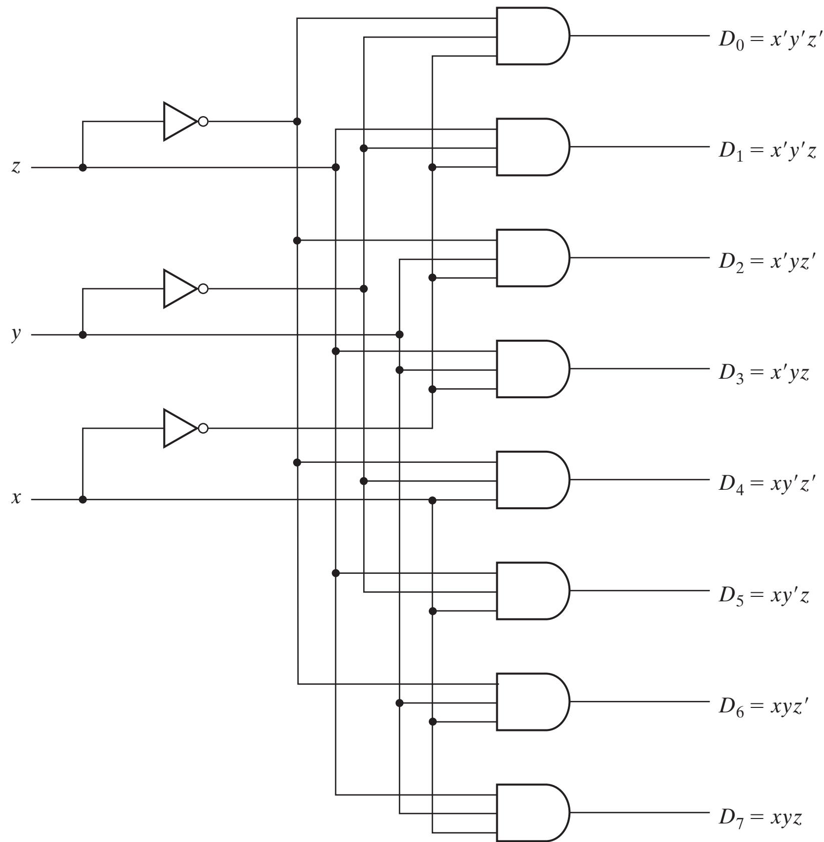


FIGURA 4-18
Decodificador de 3 a 8 líneas

Los decodificadores que presentamos aquí se describen como decodificadores de n a m líneas, donde $m \leq 2^n$. Su propósito es generar los 2^n (o menos) minitérminos de n variables de entrada. El nombre *decodificador* también se usa para referirse a otros convertidores de códigos, como un decodificador de BCD a siete segmentos.

Como ejemplo, consideremos el circuito decodificador de 3 a 8 líneas de la figura 4-18. Las tres entradas se decodifican para dar ocho salidas, cada una de las cuales representa uno de los minitérminos de las tres variables de entrada. Los tres inversores producen el complemento de las entradas, y cada una de las ocho compuertas AND genera uno de los minitérminos. Una aplicación específica de este decodificador es la conversión de binario a octal. Las variables de entrada representan un número binario, y las salidas, los ocho dígitos del sistema numérico octal. Sin embargo, un decodificador de 3 a 8 líneas puede servir para decodificar cualquier código de tres bits y obtener ocho salidas, una por cada elemento del código.

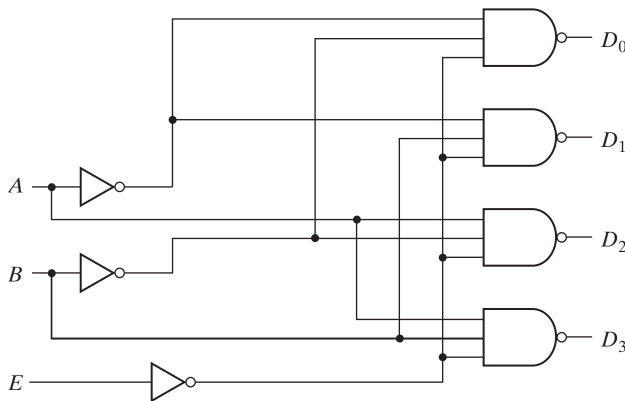
El funcionamiento del decodificador podría aclararse al examinar la tabla de verdad de la tabla 4-6. Para cada posible combinación de entrada, hay siete salidas que son 0 y sólo una igual a 1. La salida que vale 1 representa el minitérmino equivalente al número binario que se está alimentando a las líneas de entrada.

www.FreeLibros.me

Tabla 4-6
Tabla de verdad de un decodificador de 3 a 8 líneas

Entradas			Salidas							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Algunos decodificadores se construyen con compuertas NAND. Puesto que una compuerta NAND produce la operación AND con la salida invertida, resulta más económico generar los minitérminos del decodificador en su forma complementada. Además, los decodificadores incluyen una o más entradas *habilitadoras* (*enable*) que controlan el funcionamiento del circuito. En la figura 4-19 se aprecia un decodificador de 2 a 4 líneas con entrada de habilitación, construido con compuertas NAND. El circuito opera con salidas complementadas y una entrada de habilitación complementada. El decodificador se habilita cuando $E = 0$. Como indica la tabla de verdad, sólo una salida puede ser 0 en cualquier momento dado; todas las demás salidas son 1. La salida cuyo valor es 0 representa el minitérmino seleccionado por las entradas A y B. El circuito queda inhabilitado cuando $E = 1$, sean cuales sean los valores de las otras dos entradas. Cuando el circuito está inhabilitado, ninguna de las salidas es 0 y ninguno de los minitér-



a) Diagrama lógico

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

b) Tabla de verdad

FIGURA 4-19
Decodificador de 2 a 4 líneas con entrada habilitadora

www.FreeLibros.me

minos está seleccionado. En general, un decodificador podría operar con salidas complementadas o no complementadas. La entrada de habilitación podría activarse con una señal 0 o con una señal 1. Algunos decodificadores tienen dos o más entradas de habilitación que deben satisfacer una condición lógica dada para habilitar el circuito.

Un decodificador con entrada de habilitación puede funcionar como desmultiplexor. Un *desmultiplexor* es un circuito que recibe información de una sola línea y la dirige a una de 2^n posibles líneas de salida. La selección de una salida específica se controla con la combinación de bits de n líneas de selección. El decodificador de la figura 4-19 funciona como desmultiplexor de 1 a 4 líneas si E se toma como una línea de entrada de datos, y A y B se toman como entradas de selección. La variable única de entrada E tiene un camino a las cuatro salidas, pero la información de entrada se dirige a sólo una de las líneas de salida, especificada por la combinación binaria de las dos líneas de selección A y B. Esto se verifica examinando la tabla de verdad del circuito. Por ejemplo, si las líneas de selección $AB = 10$, la salida D_2 tendrá el mismo valor que la entrada E, mientras que todas las demás salidas se mantendrán en 1. Dado que se obtienen operaciones de decodificador y desmultiplexor con el mismo circuito, decimos que un decodificador con entrada de habilitación es un *decodificador/desmultiplexor*.

Es posible conectar los decodificadores con entradas de habilitación unos con otros para formar un circuito decodificador más grande. La figura 4-20 muestra dos decodificadores de 3 a 8 líneas con entradas de habilitación conectadas para formar un decodificador de 4 a 16 líneas. Cuando $w = 0$, el decodificador de arriba está habilitado y el otro está inhabilitado. Todas las salidas del decodificador de abajo son 0, y las ocho salidas del generador de arriba generan los minitérminos 0000 a 0111. Cuando $w = 1$, las condiciones de habilitación se invierten; las salidas del decodificador de abajo generan los minitérminos 1000 a 1111, mientras que todas las salidas del decodificador de arriba son 0. Este ejemplo ilustra la utilidad de las entradas de habilitación en los decodificadores y otros componentes de lógica combinacional. En general, las entradas de habilitación son una característica conveniente para interconectar dos o más componentes estándar y así expandir el componente a una función similar con más entradas y salidas.

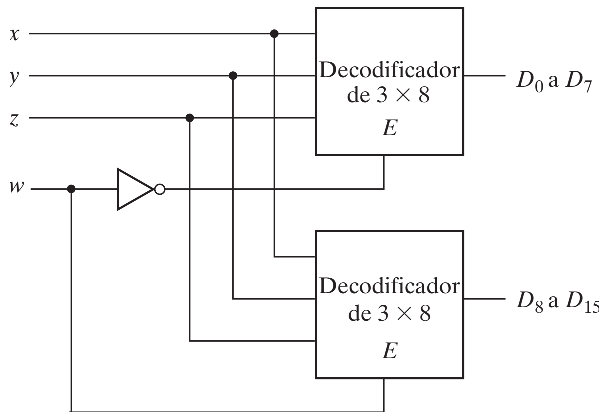


FIGURA 4-20
Decodificador 4 × 16 construido con dos decodificadores 3 × 8

www.FreeLibros.me

Implementación de lógica combinacional

Un decodificador produce los 2^n minitérminos de n variables de entrada. Puesto que cualquier función booleana es susceptible de expresarse como suma de minitérminos, es posible utilizar un decodificador para generar los minitérminos y una compuerta OR externa para formar la suma lógica. Así, cualquier circuito combinacional con n entradas y m salidas se puede implementar con un decodificador de n a 2^n líneas y m compuertas OR.

El procedimiento para implementar un circuito combinacional con un decodificador y compuertas OR requiere expresar la función booleana del circuito como suma de minitérminos. Entonces se escoge un decodificador que genere todos los minitérminos de las variables de entrada. Las entradas a cada compuerta OR se escogen de entre las salidas del decodificador, de acuerdo con la lista de minitérminos de cada función. Ilustraremos este procedimiento con un ejemplo que implementa un circuito sumador completo.

De la tabla de verdad del sumador completo (véase la tabla 4-4), obtenemos las funciones para el circuito combinacional en forma de suma de minitérminos:

$$S(x, y, z) = \Sigma(1, 2, 4, 7)$$
$$C(x, y, z) = \Sigma(3, 5, 6, 7)$$

Puesto que hay tres entradas y un total de ocho minitérminos, se necesita un decodificador de 3 a 8 líneas. La implementación se muestra en la figura 4-21. El decodificador genera los ocho minitérminos para x, y, z. La compuerta OR de la salida S forma la suma lógica de los minitérminos 1, 2, 4 y 7. La compuerta OR de la salida C forma la suma lógica de los minitérminos 3, 5, 6 y 7.

Una función con una lista larga de minitérminos requerirá una compuerta OR con un gran número de entradas. Una función con una lista de k minitérminos se expresa en su forma complementada F' empleando $2^n - k$ minitérminos. Si el número de minitérminos de una función es mayor que $2^n/2$, podremos expresar F' con menos minitérminos. En tal caso, resulta ventajoso utilizar una compuerta NOR para sumar los minitérminos de F'. La salida de la compuerta NOR complementa esta suma y genera la salida normal F. Si se usan compuertas NAND para el decodificador, como en la figura 4-19, las compuertas externas deberán ser NAND en lugar de OR. El motivo es que un circuito de compuertas NAND de dos niveles implementa una función de suma de minitérminos y equivale a un circuito AND-OR de dos niveles.

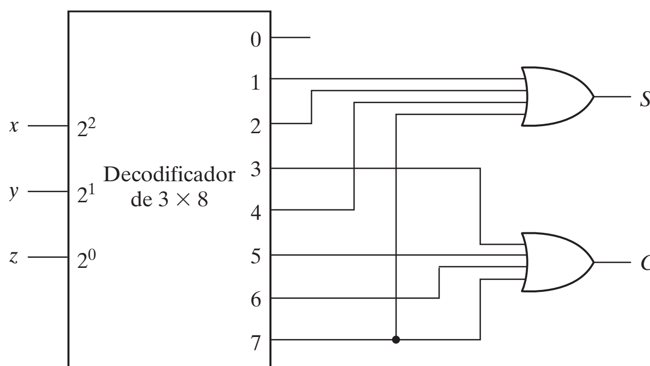


FIGURA 4-21
Implementación de un sumador completo con un decodificador

www.FreeLibros.me

4-9 CODIFICADORES

Un codificador es un circuito digital que efectúa la operación inversa de la que efectúa un decodificador. El codificador tiene 2^n (o menos) líneas de entrada y n líneas de salida. Estas últimas generan el código binario correspondiente al valor de entrada. Un ejemplo de codificador es el codificador de octal a binario cuya tabla de verdad se presenta en la tabla 4-7. Tiene ocho entradas (una para cada uno de los dígitos octales) y tres salidas que generan el número binario correspondiente. Se supone que sólo una entrada es igual a 1 en cualquier momento dado.

El codificador se puede implementar con compuertas OR cuyas salidas se determinan directamente de la tabla de verdad. La salida z es 1 cuando el dígito octal de entrada es 1, 3, 5 o 7. La salida y es 1 para los dígitos octales 2, 3, 6 o 7, y la salida x es 1 para los dígitos 4, 5, 6 o 7. Estas condiciones se expresan con las funciones booleanas de salida siguientes:

$$z = D_1 + D_3 + D_5 + D_7$$

$$y = D_2 + D_3 + D_6 + D_7$$

$$x = D_4 + D_5 + D_6 + D_7$$

El codificador se implementa con tres compuertas OR.

El codificador definido en la tabla 4-7 tiene la limitación de que sólo una entrada puede estar activa en un momento dado. Si dos entradas están activas simultáneamente, la salida producirá una combinación no definida. Por ejemplo, si D_3 y D_6 son 1 simultáneamente, la salida del decodificador será 111 porque las tres salidas son 1. Esto no representa ni el 3 binario ni el 6 binario. Para resolver esta ambigüedad, los circuitos codificadores deben establecer una prioridad de entrada que garantice que sólo se codificará una de las entradas. Si establecemos que las entradas con subíndice más alto tienen prioridad, y si tanto D_3 como D_6 son 1 al mismo tiempo, la salida será 110 porque D_6 tiene prioridad sobre D_3.

Otra ambigüedad en el codificador de octal a binario es que se genera una salida de tres ceros cuando todas las entradas son 0; esta salida es la misma que se produce cuando D_0 es igual a 1. La discrepancia se resuelve añadiendo una salida más que indique que por lo menos una entrada es 1.

Tabla 4-7
Tabla de verdad del 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

www.FreeLibros.me

Tabla 4-8
Tabla de verdad de un codificador con prioridad

Entradas				Salidas		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Codificador con prioridad

Un codificador con prioridad es un circuito codificador que incluye la función de prioridad. Su funcionamiento es tal que, si dos o más entradas son 1 al mismo tiempo, la salida prioritaria tendrá precedencia. En la tabla 4-8 se presenta la tabla de verdad de un codificador de cuatro entradas con prioridad. Además de las dos salidas, x y y, el circuito tiene una tercera salida designada V; ésta es un indicador de bit *válido* que adquiere el valor 1 cuando una o más entradas son 1. Si todas las entradas son 0, la entrada no será válida y V será 0. En tal caso, las otras dos salidas no se inspeccionarán y se especifican como condiciones de indiferencia. Advierta que, aunque las **X** en las columnas de salida representan condiciones de indiferencia, las **X** de las columnas de entrada sirven para representar una tabla de verdad en forma condensada. En vez de presentar los 16 minitérminos de cuatro variables, la tabla de verdad usa una **X** para representar tanto 1 como 0. Por ejemplo, X100 representa los dos minitérminos 0100 y 1100.

Según la tabla 4-8, cuanto más alto sea el subíndice de una entrada, mayor prioridad tendrá esa entrada. La entrada D_3 es la de mayor prioridad, así que, si es 1, la salida xy será 11 (3 binario) sin importar qué valor tengan las demás entradas. D_2 está en el siguiente nivel de prioridad. La salida es 10 si $D_2 = 1$, siempre que $D_3 = 0$ e independientemente del valor que tengan las otras dos entradas de menor prioridad. La salida para D_1 sólo se genera si las entradas con mayor prioridad son 0, y así hasta el nivel de prioridad más bajo.

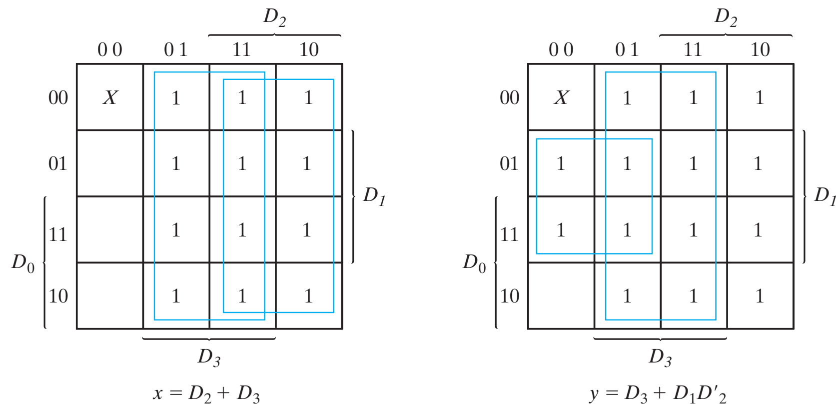


FIGURA 4-22
Mapas para el codificador con prioridad

www.FreeLibros.me

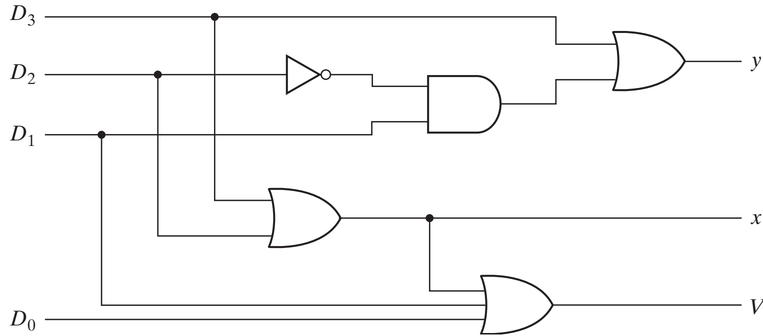


FIGURA 4-23
Codificador de cuatro entradas con prioridad

Los mapas para simplificar las salidas x y y aparecen en la figura 4-22. Los minitérminos para las dos funciones se deducen a partir de la tabla 4-8. Aunque la tabla sólo tiene cinco filas, al sustituir cada X de una fila, primero por 0 y después por 1, obtendremos las 16 posibles combinaciones de entrada. Por ejemplo, la cuarta fila de la tabla, que tiene XX10, representa los cuatro minitérminos 0010, 0110, 1010 y 1110. Las expresiones booleanas simplificadas para el codificador con prioridad se obtienen de los mapas. La condición para la salida V es una función OR de todas las variables de entrada. El codificador con prioridad se implementa en la figura 4-23 de acuerdo con las funciones booleanas siguientes:

$$x = D_2 + D_3$$
$$y = D_3 + D_1 D_2'$$
$$V = D_0 + D_1 + D_2 + D_3$$

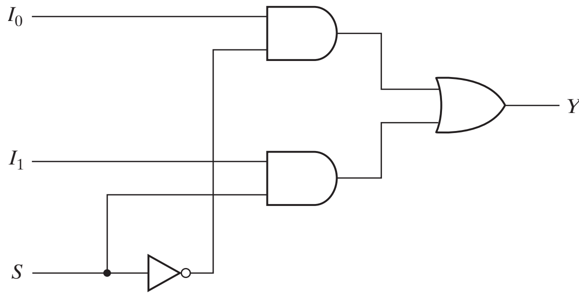
4-10 MULTIPLEXORES

Un multiplexor es un circuito combinacional que selecciona información binaria de una de muchas líneas de entrada y la envía a una sola línea de salida. La selección de una línea de entrada dada se controla con un conjunto de líneas de selección. Normalmente, hay 2^n líneas de entrada y n líneas de selección cuyas combinaciones de bits determinan cuál entrada se selecciona.

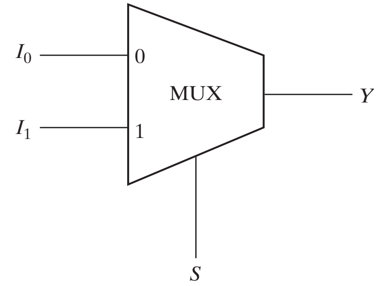
Un multiplexor de 2 líneas a 1 conecta una de dos fuentes de un bit a un destino común, como se indica en la figura 4-24. El circuito tiene dos líneas de entrada de datos, una línea de salida y una línea de selección S. Cuando $S = 0$, se habilita la compuerta AND de arriba e I_0 cuenta con una trayectoria hacia la salida. Cuando $S = 1$, la compuerta AND inferior está habilitada e I_1 tiene una trayectoria hacia la salida. El multiplexor actúa como un interruptor electrónico que selecciona una de dos fuentes. El diagrama de bloques de un multiplexor a veces se representa con un símbolo en forma de cuña, como en la figura 4-24b). Esto sugiere visualmente cómo una fuente de datos, seleccionada de entre varias, se dirige a un solo destino. En los diagramas de bloques es común rotular los multiplexores como MUX.

En la figura 4-25 se presenta un multiplexor de 4 líneas a 1. Cada una de las cuatro entradas, 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 determinada. Las salidas de las compuertas AND se aplican a una sola compuerta OR que genera la salida de una sola línea. La tabla de la función indica qué entrada se pasa a la salida con cada combinación de los valores binarios de selección. Para ilustrar el funcionamiento del circuito, consideremos el caso en que

www.FreeLibros.me

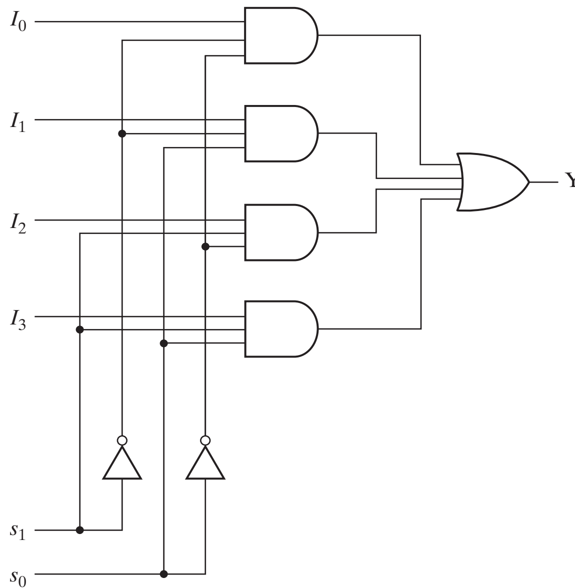


a) Diagrama lógico



b) Diagrama de bloque

FIGURA 4-24
Multiplexor de 2 líneas a 1



a) Diagrama lógico

s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

b) Tabla de función

FIGURA 4-25
Multiplexor de 4 líneas a 1

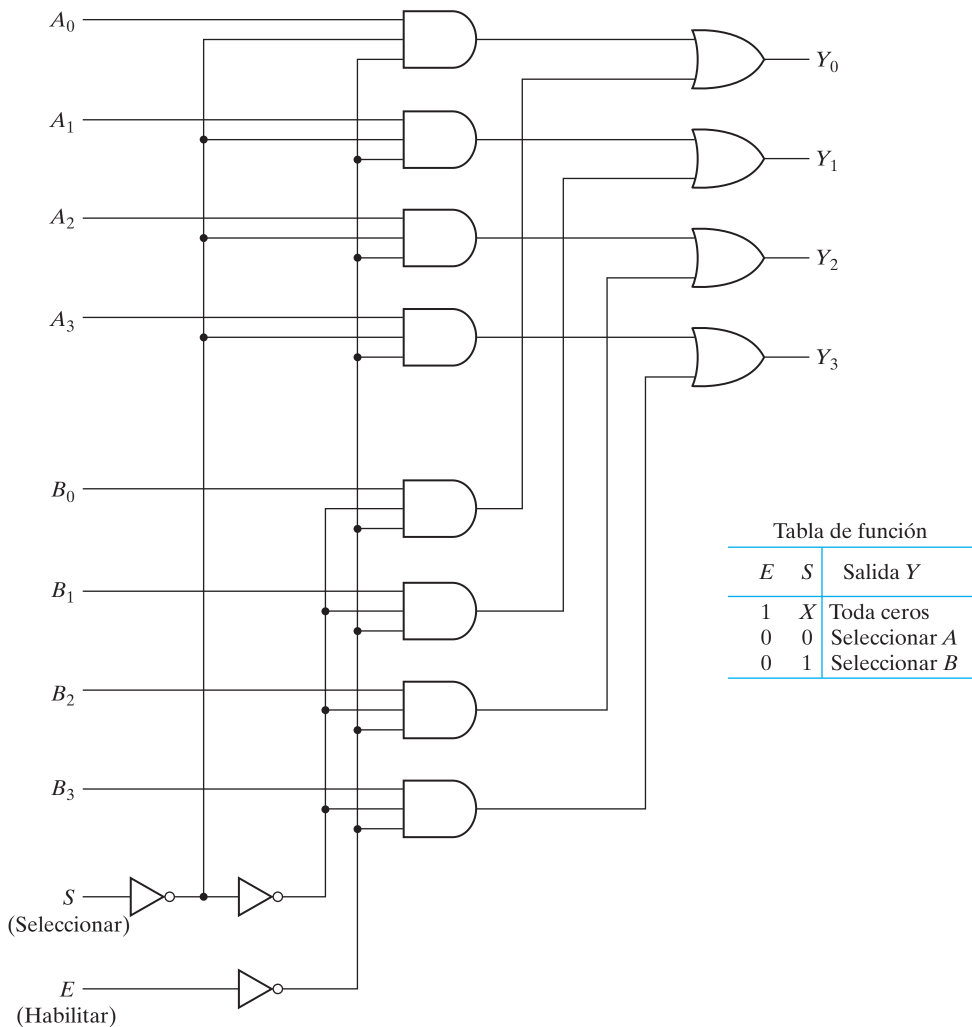
$S_1 S_0 = 10$. La compuerta AND asociada a la entrada I_2 tiene 1 en dos de sus entradas, y la tercera conectada a I_2. Las otras tres compuertas AND tienen 0 en por lo menos una de sus entradas, lo que hace que produzcan 0 como salida. Así, la salida de la compuerta OR tiene el mismo valor que I_2, así que constituye un camino de la entrada seleccionada hasta la salida. Los multiplexores también se denominan *selectores de datos*, pues seleccionan una de varias entradas y dirigen la información binaria a la línea de salida.

Las compuertas AND y los inversores del multiplexor semejan un circuito decodificador y, de hecho, decodifican las líneas de selección de entrada. En general, un multiplexor de 2^n líneas

www.FreeLibros.me

a 1 se construye a partir de un decodificador de n a 2^n líneas añadiéndole 2^n líneas de entrada, una para cada compuerta AND. Las salidas de las compuertas AND se aplican a una sola compuerta OR. El tamaño del multiplexor se especifica con el número de líneas de entrada de datos que tiene (2^n) y la única línea de salida. El número de líneas de selección (n) está implícito en el número de líneas de datos (2^n). Al igual que los decodificadores, los multiplexores podrían tener una entrada de habilitación que controla el funcionamiento de la unidad. Si dicha entrada está inactiva, las salidas quedarán inhabilitadas, y si está activa, el circuito funcionará como un multiplexor normal.

Los circuitos multiplexores se pueden combinar con entradas de selección comunes para crear una lógica de selección de múltiples bits. Por ejemplo, en la figura 4-26 se ilustra un multiplexor cuádruple de 2 líneas a 1. El circuito tiene cuatro multiplexores, cada uno de los



E	S	Salida Y
1	X	Toda ceros
0	0	Seleccionar A
0	1	Seleccionar B

FIGURA 4-26
Multiplexor cuádruple de 2 líneas a 1

www.FreeLibros.me

cuales puede seleccionar una de dos líneas de entrada. Podemos escoger que la salida Y_0 provenga de la entrada A_0 o bien de B_0. De igual manera, la salida Y_1 podría tener el valor de A_1 o B_1, y así sucesivamente. La línea de selección de entrada S selecciona una de las líneas en cada uno de los cuatro multiplexores. La entrada de habilitación E debe estar activa para que el funcionamiento sea normal. Aunque el circuito contiene cuatro multiplexores de 2 líneas a 1, seguramente lo veremos como un circuito que selecciona uno de dos conjuntos de líneas de datos de cuatro bits. Como indica la tabla de función, la unidad se habilita cuando $E = 0$. Entonces, si $S = 0$, las cuatro entradas A tendrán un camino a las cuatro salidas. En cambio, si $S = 1$, las cuatro entradas B se aplicarán a las salidas. Cuando $E = 1$, todas las salidas tienen 0, sin importar qué valor tenga S.

Implementación de funciones booleanas

En la sección 4-8 se explicó cómo utilizar un decodificador para implementar funciones booleanas añadiendo compuertas OR externas. Un examen del diagrama lógico de un multiplexor revela que básicamente es un decodificador con una compuerta OR incluida en la unidad. Los minitérminos de una función se generan en un multiplexor mediante el circuito asociado a las entradas de selección. Los minitérminos individuales se pueden seleccionar con las entradas de datos. Esto ofrece un método para implementar una función booleana de n variables con un multiplexor que tiene n entradas de selección y 2^n entradas de datos, una para cada minitérmino.

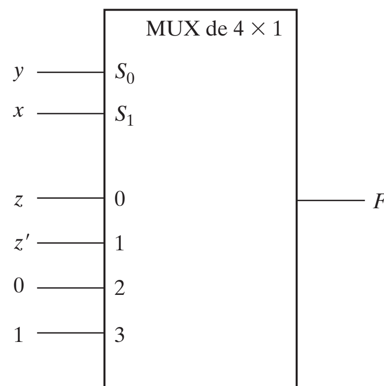
Ahora mostraremos un método más eficiente para implementar una función booleana de n variables con un multiplexor que tiene $n - 1$ entradas de selección. Las primeras $n - 1$ variables de la función se conectan a las entradas de selección del multiplexor. La variable restante de la función se utiliza para las entradas de datos. Si denotamos esa variable con z, cada entrada de datos del multiplexor será, z, z', 1 o 0. Para ilustrar este procedimiento, consideremos la función booleana de tres variables:

$$F(x, y, z) = \sum(1, 2, 6, 7)$$

La función puede implementarse con un multiplexor de 4 líneas a 1 como se indica en la figura 4-27. Las dos variables x y y se aplican a las líneas de selección en ese orden; x se conecta

x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

a) Tabla de verdad



b) Implementación con multiplexor

FIGURA 4-27
Implementación de una función booleana con un multiplexor

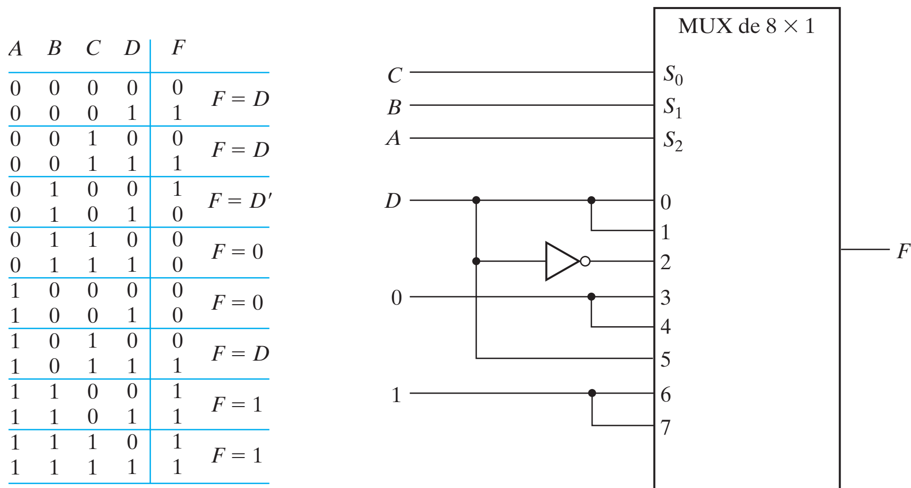
www.FreeLibros.me

a la entrada S_1 y y se conecta a S_0. Los valores de las líneas de entrada de datos se deducen de la tabla de verdad de la función. Cuando $xy = 00$, la salida F es igual a z porque $F = 0$ cuando $z = 0$ y $F = 1$ cuando $z = 1$. Esto requiere aplicar la variable z a la entrada de datos 0. El funcionamiento del multiplexor es tal que, cuando $xy = 00$, la entrada de datos 0 tiene una trayectoria hacia la salida y eso hace que F sea igual a z. De forma similar, podemos determinar las entradas que deben recibir las líneas de datos 1, 2 y 3, a partir del valor de F cuando $xy = 01$, 10 y 11, respectivamente. Este ejemplo específico muestra las cuatro posibilidades que podemos tener en las entradas de datos.

El procedimiento general para implementar cualquier función booleana de n variables con un multiplexor de $n - 1$ entradas de selección y 2^{n-1} entradas de datos se deduce del ejemplo anterior. Primero se enumera la función booleana en una tabla de verdad. Las primeras $n - 1$ variables de la tabla se aplican a las entradas de selección del multiplexor. Para cada combinación de las variables de selección, evaluamos la salida en función de la última variable. Esta función puede ser 0, 1, la variable o el complemento de la variable. Luego, estos valores se aplican a las entradas de datos en el orden correcto. Como segundo ejemplo, consideremos la implementación de la función booleana

$$F(A, B, C, D) = \sum(1, 3, 4, 11, 12, 13, 14, 15).$$

Esta función se implementa con un multiplexor con tres entradas de selección, como se ilustra en la figura 4-28. Observe que la primera variable, A, debe conectarse a la entrada de selección S_2 para que A, B y C correspondan a las entradas de selección S_2, S_1 y S_0, respectivamente. Los valores de las entradas de datos se determinan de la tabla de verdad que se presenta en la figura. El número de línea de datos correspondiente se determina a partir de la combinación binaria de ABC. Por ejemplo, cuando $ABC = 101$, la tabla indica que $F = D$, así que se aplica la variable de entrada D a la entrada de datos 5. Las constantes binarias 0 y 1 corresponden a dos valores de señal fijos. Cuando se usan circuitos integrados, el 0 lógico corresponde a la tierra de señal y el 1 lógico equivale a la señal de potencia, que por lo regular es de 5 volts.



A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	

FIGURA 4-28
Implementación de una función de cuatro entradas con un multiplexor

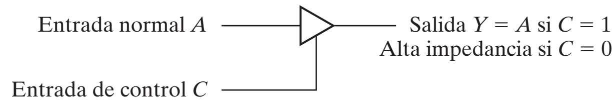


FIGURA 4-29
Símbolo gráfico para un búfer de tres estados

Compuertas de tres estados

Es posible construir un multiplexor con compuertas de tres estados. Una compuerta de tres estados es un circuito digital que exhibe tres estados. Dos de los estados son señales equivalentes al 1 y al 0 lógicos, como en las compuertas convencionales. El tercer estado es un estado de alta impedancia. El *estado de alta impedancia* se comporta como un circuito abierto, lo que implica que la salida parece estar desconectada y el circuito carece de significado lógico. Las compuertas de tres estados son capaces de realizar cualquier lógica convencional, como AND o NAND, pero la que se usa más comúnmente es la compuerta búfer.

En la figura 4-29 se observa el símbolo gráfico de un búfer de tres estados. Se distingue de un búfer normal con una línea de control de entrada que entra por la parte inferior del símbolo de compuerta. El búfer tiene una entrada normal, una salida y una entrada de control que determina el estado de la salida. Si la entrada de control es 1, la salida está habilitada y la compuerta se comporta como un búfer convencional, cuya salida es igual a la entrada normal. Cuando la entrada de control es 0, la salida se inhabilita y la compuerta pasa a un estado de alta impedancia, sea cual sea el valor en la entrada normal. El estado de alta impedancia de una compuerta de tres estados ofrece una característica especial que no ofrecen otras compuertas. Gracias a ella, un gran número de salidas de compuertas de tres estados se pueden conectar con alambres para formar una línea común sin correr riesgos por los efectos de carga.

En la figura 4-30 se ilustra la construcción de multiplexores con búferes de tres estados. La parte a) de la figura muestra la construcción de un multiplexor de 2 líneas a 1 con dos búferes

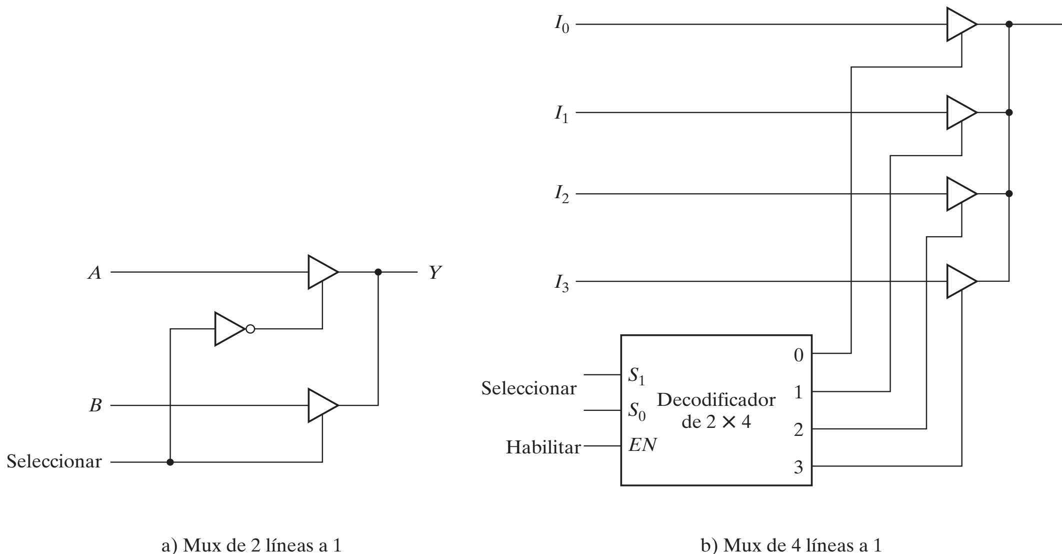


FIGURA 4-30
Multiplexores con compuertas de tres estados

www.FreeLibros.me

de tres estados y un inversor. Las dos salidas se conectan entre sí para formar una sola línea de salida. (Cabe señalar que este tipo de conexión no puede efectuarse con compuertas que no tengan salidas de tres estados.) Si la entrada de selección es 0, el búfer superior queda habilitado por su entrada de control, y el búfer inferior queda inhabilitado. Entonces, la salida Y es igual a la entrada A. Cuando la entrada de selección es 1, se habilita el búfer inferior y Y es igual a B.

En la figura 4-30b) se muestra la construcción de un multiplexor de 4 líneas a 1. Las salidas de cuatro búferes de tres estados se conectan entre sí para formar una sola línea de salida. Las entradas de control de los búferes determinan cuál de las cuatro entradas normales, I_0 a I_3, se conectará a la línea de salida. Nunca habrá más de un búfer en el estado activo a la vez. Los búferes conectados deben controlarse de modo que sólo un búfer de tres estados tenga acceso a la salida, mientras todos los demás búferes se mantienen en un estado de alta impedancia. Una forma de garantizar que no más de una entrada de control esté activa en un momento dado es utilizar un decodificador, como se indica en el diagrama. Si la entrada de habilitación del decodificador es 0, sus cuatro salidas son 0, y la línea de bus está en un estado de alta impedancia porque los cuatro búferes están inhabilitados. Cuando la entrada de habilitación está activa, uno de los búferes de tres estados estará activo, dependiendo del valor binario en las entradas de selección del decodificador. Una investigación cuidadosa revelará que este circuito es otra forma de construir un multiplexor de 4 líneas a 1.

4-11 HDL PARA CIRCUITOS COMBINACIONALES

Presentamos el lenguaje de descripción de hardware (HDL) Verilog en la sección 3-9. En esta sección se hablará de las opciones existentes para describir circuitos combinacionales en HDL. Presentaremos los circuitos secuenciales en el capítulo siguiente. Como ya se dijo, el módulo es el bloque de construcción básico en Verilog HDL. Es posible describir un módulo con cualquiera de las siguientes técnicas de modelado (o mediante una combinación de ellas):

- Modelado en el nivel de compuertas creando ejemplares de compuertas primitivas y módulos definidos por el usuario.
- Modelado de flujo de datos empleando enunciados de asignación continua con la palabra clave **assign.**
- Modelado de comportamiento utilizando enunciados de asignación procedimentales con la palabra clave **always.**

El modelado en el nivel de compuertas describe el circuito especificando las compuertas y cómo se conectan entre sí. El modelado de flujo de datos se utiliza primordialmente para describir circuitos combinacionales. El modelado de comportamiento sirve para describir sistemas digitales en un nivel de abstracción más alto. Existe otro estilo de modelado llamado modelado en el nivel de interruptores, que permite diseñar en el nivel de transistores MOS y que se analizará en la sección 10-10.

Modelado en el nivel de compuertas

Ya presentamos el modelado en el nivel de compuertas con un ejemplo sencillo en la sección 3-9. En este tipo de representación, los circuitos se especifican por sus compuertas lógicas y su interconexión. Es una descripción textual de un diagrama esquemático. Verilog reconoce 12 compuertas básicas como primitivas predefinidas. Cuatro compuertas primitivas son del tipo de tres estados. Las otras ocho son las que se presentaron en la sección 2-7, y se declaran con las palabras clave en minúsculas **and, nand, or, nor, xor, xnor, not** y **buf.** Cuando se simulan

www.FreeLibros.me

Tabla 4-9
Tablas de verdad para las compuertas primitivas predefinidas

and	0	1	x	z
0	0	0	0	0
1	0	1	x	x
x	0	x	x	x
z	0	x	x	x

or	0	1	x	z
0	0	1	x	x
1	1	1	1	1
x	x	1	x	x
z	x	1	x	x

xor	0	1	x	z
0	0	1	x	x
1	1	0	x	x
x	x	x	x	x
z	x	x	x	x

not	Entrada	Salida
	0	1
	1	0
	x	x
	z	x

las compuertas, el sistema asigna un conjunto lógico de cuatro valores a cada compuerta. Además de los dos valores lógicos 0 y 1, hay otros dos valores, *desconocido* y *alta impedancia*. Un valor desconocido se denota con **x,** y uno de alta impedancia, con **z.** Los valores desconocidos se consideran durante la simulación para el caso en que una entrada o salida es ambigua, por ejemplo, si todavía no se le ha asignado un valor de 0 o 1. La condición de alta impedancia se da en la salida de las compuertas de tres estados o si por descuido un alambre se deja desconectado. Las tablas de verdad de **and, or, xor** y **not** aparecen en la tabla 4-9. Las tablas de verdad para las demás compuertas son iguales, salvo que las salidas se complementan. Observe que, para la compuerta **and,** la salida es 1 sólo cuando ambas entradas son 1; la salida es 0 si cualquier entrada es 0. Por otra parte, si una entrada es **x** o **z,** la salida es **x.** La salida de la compuerta **or** es 0 si ambas entradas son 0, 1 si cualquier entrada es 1, y **x** en los demás casos.

Cuando una compuerta primitiva se incorpora en un módulo, decimos que se *crea un ejemplar* en el módulo. En general, los enunciados que crean ejemplares de componentes hacen referencia a componentes de nivel más bajo del diseño, con lo que básicamente se crean copias únicas (*ejemplares*) de esos componentes en el módulo de nivel más alto. Así, cuando un módulo usa una compuerta en su descripción decimos que *crea un ejemplar de* la compuerta.

A continuación se presentan dos ejemplos de modelado en el nivel de compuertas. Ambos ejemplos usan grupos de varios bits llamados *vectores*. Los vectores se especifican entre corchetes, con dos números separados por un signo de dos puntos. El código siguiente especifica dos vectores:

```
output [0:3]D;
wire [7:0]SUM;
```

El primer enunciado declara un vector de salida *D* de cuatro bits, 0 a 3. El segundo declara un vector alambrado *SUM* de ocho bits, numerados del 7 al 0. El primer número corresponde al bit más significativo del vector. Los bits individuales se especifican entre corchetes; por ejemplo *D*[2] especifica el bit 2 de *D*. También es posible direccionar partes de vectores. Por ejemplo, *SUM*[2:0] especifica los tres bits menos significativos del vector *SUM*.

El ejemplo HDL 4-1 es ilustrativo de la descripción en el nivel de compuertas de un decodificador de 2 a 4 líneas. Posee dos entradas de datos *A* y *B* y una entrada de habilitación *E*. Las cuatro salidas se especifican con el vector *D*. La declaración **wire** es para conexiones internas. Tres compuertas **not** producen el complemento de las entradas y cuatro compuertas **nand** generan las salidas para *D*. Recuerde que la salida siempre es el primer elemento en la lista de una compuerta, seguida de las entradas. Este ejemplo describe el decodificador de la fi-

www.FreeLibros.me

Ejemplo HDL 4-1

```
//Descripción a nivel de compuertas del decodificador 2 a 4
//de la figura 4-19
module decoder_gl (A,B,E,D);
   input A,B,E;
   output [0:3]D;
   wire Anot,Bnot,Enot;
   not
      n1 (Anot,A),
      n2 (Bnot,B),
      n3 (Enot,E);
   nand
      n4 (D[0],Anot,Bnot,Enot),
      n5 (D[1],Anot,B,Enot),
      n6 (D[2],A,Bnot,Enot),
      n7 (D[3],A,B,Enot);
endmodule
```

gura 4-19 y sigue los procedimientos establecidos en la sección 3-9. Advierta que las palabras clave **not** y **nand** se escriben una sola vez y no tienen que repetirse para cada compuerta, pero hay que insertar comas al final de cada serie de compuertas con la excepción del último enunciado, que debe terminar con un punto y coma.

Es posible combinar dos o más módulos para construir una descripción jérárquica de un diseño. Existen dos tipos básicos de metodologías de diseño: descendente y ascendente. En un diseño *descendente*, se define el bloque de más alto nivel y luego se identifican los sub-bloques requeridos para construir el bloque de más alto nivel. En un diseño *ascendente*, primero se identifican los bloques de construcción y luego se combinan para construir el bloque de más alto nivel. Tomemos como ejemplo el sumador binario de la figura 4-9. Podemos considerarlo como un componente de bloque superior construido con cuatro bloques de sumador completo, cada uno de los cuales se construye con dos bloques de semisumador. En un diseño descendente, primero se define un sumador de cuatro bits y luego se describen el sumador completo y el semisumador. En un diseño ascendente, se define el semisumador, luego se construye el sumador completo y al último se construye el sumador de cuatro bits con los sumadores completos.

En el ejemplo HDL 4-2 se muestra una descripción jérárquica ascendente de un sumador de cuatro bits. El semisumador se define con ejemplares de compuertas primitivas. El siguiente módulo describe el sumador completo con dos ejemplares de semisumador. El tercer módulo describe el sumador de cuatro bits con cuatro ejemplares de sumador completo. (Tenga presente que los identificadores no pueden comenzar con un número, pero sí con un subraya, así que el nombre del módulo es `_4bit_adder`.) Los ejemplares se crean utilizando el nombre del módulo ejemplarizado con un conjunto nuevo de nombres de puerto (o el mismo). Por ejemplo, el ejemplar del semisumador HA1 dentro del módulo del sumador completo se crea con los puertos $S1$, $D1$, x, y. Esto produce un semisumador con las salidas $S1$, $D1$ y las entradas x, y.

Cabe señalar que, en Verilog, no es posible colocar una definición de módulo dentro de otra. En otras palabras, no está permitido insertar un módulo entre las palabras clave **module** y **endmodule** de otro módulo. La única forma de incorporar una definición de módulo en otro módulo es creando un ejemplar del mismo. Así, se crean ejemplares de módulos dentro de otros módulos para crear una descripción jérárquica de un diseño. Además, tome nota de que hay que especificar nombres al crear ejemplares de módulos ya definidos (como FA0 para el

www.FreeLibros.me

Ejemplo HDL 4-2

```
//Descripción jerárquica a nivel de compuertas de un sumador de 4 bits
//Descripción del semisumador (véase la figura 4-5b)
module halfadder (S,C,x,y);
   input x,y;
   output S,C;
//Crear ejemplares de compuertas primitivas
   xor (S,x,y);
   and (C,x,y);
endmodule

//Descripción de sumador completo (véase la figura 4-8)
module fulladder (S,C,x,y,z);
   input x,y,z;
   output S,C;
   wire S1,D1,D2; //Salidas de primera XOR y dos compuertas AND
//Crear un ejemplar del semisumador
    halfadder HA1 (S1,D1,x,y),
              HA2 (S,D2,S1,z);
    or g1(C,D2,D1);
endmodule

//Descripción del sumador de 4 bits (véase la figura 4-9)
module _4bit_adder (S,C4,A,B,C0);
   input [3:0] A,B;
   input C0;
   output [3:0] S;
   output C4;
   wire C1,C2,C3;  //Acarreos intermedios
//Crear un ejemplar del sumador completo
   fulladder  FA0 (S[0],C1,A[0],B[0],C0),
              FA1 (S[1],C2,A[1],B[1],C1),
              FA2 (S[2],C3,A[2],B[2],C2),
              FA3 (S[3],C4,A[3],B[3],C3);
endmodule
```

primer sumador completo en el tercer módulo), pero el uso de nombres es opcional al crear ejemplares de compuertas primitivas.

Compuertas de tres estados

Como se mencionó en la sección 4-10, las compuertas de tres estados tienen una entrada de control que puede colocar a la compuerta en un estado de alta impedancia. Dicho estado se indica con **z** en HDL. Hay cuatro tipos de compuertas de tres estados, que se ilustran en la figura 4-31. La compuerta **bufif1** se comporta como un búfer normal si control = 1. La salida pasa a un estado de alta impedancia **z** cuando control = 0. La compuerta **bufif0** se comporta de

www.FreeLibros.me

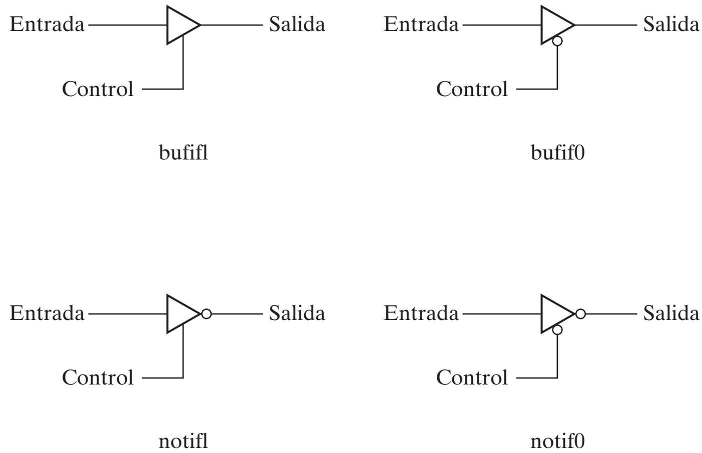


FIGURA 4-31
Compuertas de tres estados

forma similar excepto que el estado de alta impedancia se da cuando control = 1. Las dos compuertas **not** operan de forma similar, excepto que la salida es el complemento de la entrada cuando la compuerta no está en un estado de alta impedancia. Se crean ejemplares de las compuertas con el enunciado

nombre_compuerta (salida, entrada, control);

El nombre de la compuerta puede ser cualquiera de las cuatro compuertas de tres estados. La salida puede dar 0, 1 o **z.** Dos ejemplos de creación de ejemplares de compuertas son

```
bufif1  (OUT,A,control);
notif0  (Y,B,enable);
```

En el primer ejemplo, la entrada A se transfiere a OUT cuando control = 1. OUT pasa a **z** cuando control = 0. En el segundo ejemplo, la salida Y = **z** cuando enable = 1, y $Y = B'$ cuando enable = 0.

Las salidas de las compuertas de tres estados se pueden conectar entre sí para formar una línea de salida común. Para identificar semejante conexión, HDL usa la palabra clave **tri** de (triestado) para indicar que la salida tiene múltiples alimentaciones. Por ejemplo, consideremos el multiplexor de 2 líneas a 1 con compuertas de tres estados que se aprecia en la figura 4-32.

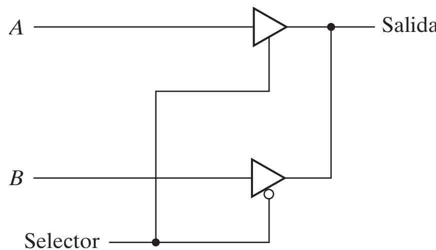


FIGURA 4-32
Multiplexor de 2 líneas a 1 con búferes de tres estados

www.FreeLibros.me

La descripción en HDL debe usar el tipo de datos **tri** para la salida.

```
module muxtri (A,B,select,OUT);
   input A,B,select;
   output OUT;
   tri OUT;
   bufif1 (OUT,A,select);
   bufif0 (OUT,B,select);
endmodule
```

Los dos búferes de tres estados tienen la misma salida. Para indicar que existe una conexión común, es necesario declarar *OUT* con la palabra clave **tri.**

Las palabras clave **wire** y **tri** son ejemplos del tipo de datos *net*. Las redes (*nets*) representan conexiones entre elementos de hardware. Su valor depende continuamente de la salida del dispositivo que representan. La palabra *net* no es una palabra clave, pero representa una clase de tipos de datos como **wire, wor, wand, tri, supply1** y **supply0.** La declaración **wire** es la que se usa con mayor frecuencia. La red **wor** modela la implementación en hardware de la configuración OR alambrada. La **wand** modela la configuración AND alambrada (véase la figura 3-28). Las redes **supply1** y **supply0** representan fuente de poder y tierra. Se utilizan en el modelado a nivel de interruptores (véase la sección 10-10).

Modelado de flujo de datos

El modelado de flujo de datos utiliza varios operadores que actúan sobre operandos para producir resultados deseados. Verilog HDL cuenta con cerca de 30 tipos de operadores. En la tabla 4-10 se presentan algunos de ellos, su símbolo y la operación que realizan. (En la tabla 8-1, sección 8-2, se da una lista completa de operadores.) Hay que distinguir entre las operaciones aritméticas y lógicas, por lo que se usan símbolos distintos para cada una. El símbolo más (+) se utiliza para la suma aritmética, y el AND lógico emplea el símbolo &. Hay símbolos especiales para OR, NOT y XOR. El símbolo de igualdad es doble (sin espacio intermedio) para distinguirlo del que se usa en el enunciado de asignación. El operador de concatenación permite juntar varios operandos. Por ejemplo, es posible concatenar dos operandos de dos bits cada uno para formar un operando de cuatro bits. El operador condicional se explicará más adelante junto con el ejemplo HDL 4-6.

Tabla 4-10
Operadores de Verilog HDL

Símbolo	Operación
+	Suma binaria
−	Resta binaria
&	AND bit por bit
|	OR bit por bit
^	XOR bit por bit
~	NOT bit por bit
==	Igualdad
>	Mayor que
<	Menor que
{ }	Concatenación
? :	Condicional

www.FreeLibros.me

Ejemplo HDL 4-3

```
//Descripción de flujo de datos del decodificador de 2 a 4
//Véase la figura 4-9
module decoder_df (A,B,E,D);
   input A,B,E;
   output [0:3] D;
   assign D[0] = ~(~A & ~B & ~E),
          D[1] = ~(~A & B & ~E),
          D[2] = ~(A & ~B & ~E),
          D[3] = ~(A & B & ~E);
endmodule
```

El modelado de flujo de datos utiliza asignaciones continuas y la palabra clave **assign.** Una asignación continua es un enunciado que asigna un valor a una *net*. El tipo de datos *net* se usa en Verilog HDL para representar una conexión física entre elementos de circuito. Una *net* define una salida de compuerta declarada por un enunciado **output** o **wire.** El valor asignado a la *net* se especifica con una expresión que utiliza operandos y operadores. Por ejemplo, suponiendo que ya se declararon las variables, un multiplexor de 2 líneas a 1 con entradas de datos *A* y *B*, entrada de selección *S* y salida *Y* se describe con la asignación continua

```
assign Y = (A & S) | (B & ~S);
```

Primero aparece la palabra clave **assign** seguida de la salida deseada *Y* y un signo de igual. Después viene una expresión booleana. En términos de hardware, esto equivaldría a conectar la salida de la compuerta OR (|) al alambre *Y*.

Los dos ejemplos que siguen muestran los modelos de flujo de datos de los dos ejemplos anteriores a nivel de compuertas. En el ejemplo HDL 4-3 se presenta la descripción de flujo de datos de un decodificador de 2 a 4 líneas. El circuito se define con cuatro enunciados de asignación continua empleando expresiones booleanas, una para cada salida. La descripción de flujo de datos del sumador de cuatro bits aparece en el ejemplo HDL 4-4. La lógica de suma se describe con un solo enunciado que usa los operadores de suma y concatenación. El símbolo más (+) especifica la suma binaria de los cuatro bits de *A* con los cuatro

Ejemplo HDL 4-4

```
//Descripción de flujo de datos de un sumador de 4 bits
module binary_adder (A,B,Cin,SUM,Cout);
   input [3:0] A,B;
   input Cin;
   output [3:0] SUM;
   output Cout;
   assign {Cout,SUM} = A + B + Cin;
endmodule
```

www.FreeLibros.me

Ejemplo HDL 4-5

```
//Descripción de flujo de datos de un comparador de 4 bits
module magcomp (A,B,ALSB,AGTB,AEQB);
   input [3:0] A,B;
   output ALTB,AGTB,AEQB;
   assign ALTB=(A < B),
          AGTB = (A > B),
          AEQB = (A == B);
endmodule
```

bits de *B* y el bit de *Cin*. La salida deseada es la concatenación del acarreo de salida *Cout* y los cuatro bits de *SUM*. La concatenación de operandos se expresa entre llaves, separando los operandos con comas. Así, {`Cout, SUM`} representa el resultado de cinco bits de la operación de suma.

El modelado de flujo de datos permite describir circuitos combinacionales por su función en vez de por su estructura de compuertas. Para mostrar cómo las descripciones de flujo de datos facilitan el diseño digital, consideremos el comparador de magnitudes de cuatro bits que se describe en el ejemplo HDL 4-5. El módulo especifica dos entradas de cuatro bits, *A* y *B*, y tres salidas. Una salida (ALTB) es 1 lógico si *A* es menor que *B*; otra salida (AGTB) es 1 lógico si *A* es mayor que *B*; y una tercera salida (AEQB) es 1 lógico si *A* es igual a *B*. (Advierta que la igualdad se representa con dos símbolos de igual.) Un compilador de síntesis Verilog HDL puede aceptar como entrada esta descripción de módulo y producir la lista de un circuito equivalente a la figura 4-17.

El siguiente ejemplo usa el operador condicional (?:). Este operador acepta tres operandos:

condición ? expresión-verdadera : expresión-falsa ;

La condición se evalúa. Si el resultado es 1 lógico, se evalúa la expresión verdadera. Si el resultado es 0 lógico, se evalúa la expresión falsa. Esto equivale a una condición **if-else.** El ejemplo HDL 4-6 presenta la descripción de un multiplexor de 2 a 1 empleando el operador condicional. La asignación continua

```
assign OUT = select ? A : B ;
```

especifica la condición OUT $= A$ si select $= 1$, si select $= 0$, entonces OUT $= B$.

Ejemplo HDL 4-6

```
//Descripción de flujo de datos del multiplexor 2 a 1
module mux2x1_df (A,B,select,OUT);
   input A,B,select;
   output OUT;
   assign OUT = select ? A : B;
endmodule
```

www.FreeLibros.me

Ejemplo HDL 4-7

```
//Descripción de comportamiento del multiplexor 2 a 1
module mux2x1_bh(A,B,select,OUT);
   input A,B,select;
   output OUT;
   reg OUT;
   always @ (select or A or B)
         if (select == 1) OUT = A;
         else OUT = B;
endmodule
```

Modelado de comportamiento

El modelado de comportamiento representa los circuitos digitales en un nivel funcional y algorítmico. Se le utiliza primordialmente para describir circuitos secuenciales, pero sirve también para describir circuitos combinacionales. Aquí se presentarán dos ejemplos sencillos de circuitos combinacionales como introducción al tema. Se analizará con mayor detalle el modelado de comportamiento en la sección 5-5, después de estudiar los circuitos secuenciales.

Las descripciones de comportamiento emplean la palabra clave **always** seguida de una lista de enunciados de asignación procedimentales. La salida deseada de los enunciados de asignación procedimentales debe ser del tipo de datos **reg.** A diferencia del tipo de datos **wire,** en el que la salida deseada de una asignación se puede actualizar continuamente, el tipo de datos **reg** conserva su valor hasta que se le asigna uno nuevo.

El ejemplo HDL 4-7 muestra la descripción de comportamiento de un multiplexor de 2 líneas a 1 (compare con el ejemplo HDL 4-6). Dado que la variable OUT es una salida deseada, debemos declararla como dato **reg** (además de la declaración **output**). Los enunciados de asignación procedimentales dentro del bloque **always** se ejecutan cada vez que hay un cambio en cualquiera de las variables indicadas después del símbolo @. (Observe que no se escribe un (;) al final del enunciado **always.**) En este caso, la lista incluye las variables de entrada *A* y *B*, y *select*. Advierta que se usa la palabra clave **or** entre las variables en lugar del operador de OR lógico "|". El enunciado condicional **if-else** permite tomar una decisión con base en el valor de la entrada *select*. El enunciado **if** se puede escribir sin el símbolo de igualdad:

```
if (select) OUT = A ;
```

Este enunciado implica que se examina *select* para ver si es 1 lógico.

El ejemplo HDL 4-8 describe la función de un multiplexor de 4 líneas a 1. La entrada *select* se define como un vector de dos bits, y la salida *y* se declara como dato **reg.** El enunciado **always** tiene un bloque secuencial delimitado por las palabras clave **case** y **endcase.** El bloque se ejecuta cada vez que cambia el valor de cualquiera de las entradas indicadas después del símbolo @. El enunciado **case** es una condición de ramificación condicional multivías. La expresión **case** (select) se evalúa y se compara con los valores de la lista de enunciados que siguen. Se ejecuta el primer valor que coincide con la condición verdadera. Puesto que *select* es un número de dos bits, puede ser igual a 00, 01, 10 o 11. Los números binarios se especifican con una **b** precedida por un apóstrofo. Primero se escribe el tamaño del número y luego su valor. Así, 2′b01 especifica un número binario de dos dígitos cuyo valor es 01. También pueden especificarse números en decimal, octal o hexadecimal, con las letras **′d, ′o** y **′h,** respectivamente. Si no se especifica la base del número, se toma como decimal por omisión. Si no se especifica el tamaño del número, el sistema supondrá que es de 32 bits.

www.FreeLibros.me

Ejemplo HDL 4-8

```
//Descripción de comportamiento del multiplexor 4 a 1
//Describe la tabla de función de la figura 4-25b).
module mux4x1_bh (i0,i1,i2,i3,select,y);
   input i0,i1,i2,i3;
   input [1:0] select;
   output y;
   reg y;
   always @ (i0 or i1 or i2 or i3 or select)
            case (select)
               2'b00: y = i0;
               2'b01: y = i1;
               2'b10: y = i2;
               2'b11: y = i3;
            endcase
endmodule
```

Hemos mostrado aquí ejemplos sencillos de descripciones del comportamiento de circuitos combinacionales. El modelado de comportamiento y los enunciados de asignación procedimentales requieren conocimientos de circuitos secuenciales y se tratarán más a fondo en la sección 5-5.

Cómo escribir un conjunto de pruebas sencillo

Un conjunto de pruebas es un programa en HDL que sirve para aplicar estímulos a un diseño HDL, a fin de probarlo y observar su respuesta durante una simulación. En ocasiones, los conjuntos de pruebas resultan muy complejos y largos, y su desarrollo podría ser más tardado que el del diseño que se prueba. Sin embargo, los que se incluyen aquí son relativamente sencillos, ya que lo único que queremos probar es circuitos combinacionales. Presentamos los ejemplos para ilustrar descripciones representativas de módulos de estímulo HDL.

Además del enunciado **always,** los conjuntos de pruebas utilizan el enunciado **initial** para estimular el circuito probado. El enunciado **always** se ejecuta repetidamente en un ciclo. El enunciado **initial** sólo se ejecuta una vez en el tiempo = 0 de la simulación y podría continuar con cualesquier operaciones que se retarden en cierto número de unidades de tiempo, especificadas por el símbolo #. Por ejemplo, consideremos el bloque **initial**

```
initial
     begin
             A = 0; B= 0;
        #10  A = 1;
        #20  A = 0; B=1;
     end
```

El bloque se encierra entre las palabras clave **begin** y **end.** En el tiempo $= 0$, A y B se ponen en 0. Diez unidades de tiempo después, A se cambia a 1. Veinte unidades de tiempo después (en $t = 30$), A se cambia a 0 y B se cambia a 1. Las entradas de una tabla de verdad de tres bits se generan con el bloque **initial**

www.FreeLibros.me

```
initial
    begin
             D = 3'b000;
             repeat (7)
        #10  D = D + 3'b001;
    end
```

El vector *D* de tres bits recibe el valor inicial 000 en el tiempo = 0. La palabra clave **repeat** especifica un enunciado cíclico: se suma 1 a *D* siete veces, una vez cada 10 unidades de tiempo. El resultado es una sucesión de números binarios de 000 a 111.

Un módulo de estímulo es un programa HDL que tiene la forma siguiente:

module nombreprueba.

Declarar identificadores locales **reg** y **wire.**

Crear ejemplares del módulo de diseño a probar.

Generar estímulos con enunciados **initial** y **always.**

Exhibir la respuesta de salida.

endmodule

Los módulos de prueba por lo regular carecen de entradas y de salidas. Las señales que se aplican como entradas al módulo de diseño simulado se declaran en el módulo de estímulo como del tipo de datos local **reg.** Las salidas del módulo de diseño que se exhiben para efectuar las pruebas se declaran en el módulo de estímulo como del tipo de datos local **wire.** Luego se crea un ejemplar del módulo a probar empleando los identificadores locales. La figura 4-33 aclara esta relación. El módulo de estímulo genera entradas para el módulo de diseño declarando los identificadores *TA* y *TB* como de tipo **reg,** y verifica la salida de la unidad de diseño con el identificador **wire** *TC*. Luego se usan los identificadores locales para crear el ejemplar del módulo de diseño a probar.

La respuesta al estímulo generado por los bloques **initial** y **always** aparecerá en la salida del simulador como diagramas de temporización. También es posible exhibir salidas numéricas empleando *tareas del sistema* de Verilog. Éstas son funciones integradas del sistema que se

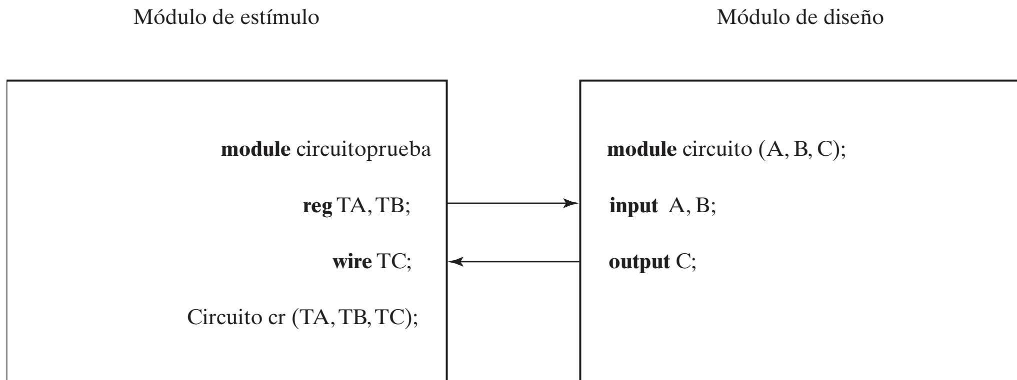


FIGURA 4-33
Interacción de los módulos de estímulo y de diseño

www.FreeLibros.me

reconocen por palabras clave que inician con el símbolo $. Algunas de las tareas del sistema que sirven para exhibición son

$display—exhibir una vez el valor de variables o cadenas con un retorno de fin de línea,
$write**—igual que **$display pero sin pasar a la siguiente línea,
$monitor—exhibe variables cada vez que un valor cambia durante una simulación,
$time—muestra el tiempo de simulación,
$finish—termina la simulación.

La sintaxis de **$display, $write** y **$monitor** tiene la forma

Nombre-tarea (especificación de formato, lista argumentos);

La especificación de formato incluye la base de los números que se exhiben, lo que se indica con el símbolo (%), y podría tener una cadena encerrada entre comillas ("). La base puede ser binaria, decimal, hexadecimal u octal, lo que se indica con los símbolos %b, %d, %h y %o, respectivamente. Por ejemplo, el enunciado

```
$display (%d %b %b, C,A,B);
```

especifica la exhibición de *C* en decimal, y de *A* y *B* en binario. Observe que no hay comas en la especificación de formato, que la especificación de formato y la lista de argumentos se separan con una coma, y que la lista de argumentos lleva comas entre las variables. Un ejemplo de enunciado que especifica una cadena encerrada entre comillas es

$display ("tiempo = %0d A = %b B = %b", **$tiempo**,A,B);

que produce

```
tiempo = 3  A = 10  B = 1
```

donde (tiempo =), (A =) y (B =) forman parte de la cadena que se exhibirá. El formato %0d, %b y %b especifica la base de **$time,** *A* y *B*, respectivamente. Al exhibir valores de tiempo, es recomendable usar el formato %0d en lugar del formato %d. Esto exhibe los dígitos significativos sin los espacios a la derecha que se exhiben cuando se usa %d. (%d exhibe unos 10 espacios a la derecha porque el tiempo se calcula como un número de 32 bits.)

En el ejemplo HDL 4-9 se presenta un módulo de estímulo. El circuito a probar es el multiplexor 2×1 que se describió en el ejemplo 4-6. El módulo `testmux` no tiene puertos. Las entradas del multiplexor se declaran con la palabra clave **reg,** y las salidas, con la palabra clave **wire.** Se crea un ejemplar del multiplexor con las variables locales. El bloque **initial** especifica una sucesión de valores binarios que se aplicarán durante la simulación. La respuesta de salida se verifica con la tarea del sistema **$monitor.** Cada vez que una variable cambia de valor, el simulador exhibe las entradas, la salida y el tiempo. El resultado de la simulación aparece en la bitácora de simulación (simulation log) del ejemplo. Ahí vemos que $OUT = A$ cuando $S = 1$ y $OUT = B$ cuando $S = 0$, lo que verifica el funcionamiento del multiplexor.

La simulación lógica es un método rápido y exacto para analizar circuitos combinacionales y verificar que funcionan correctamente. Hay dos tipos de verificación: funcional y de tiempos. En la verificación *funcional*, estudiamos la operación lógica del circuito, independientemente de consideraciones de temporización. Esto se hace deduciendo la tabla de verdad del circuito combinacional. En la verificación *de tiempos*, estudiamos el funcionamiento del circuito incluyendo el efecto de los retardos en las compuertas. Esto se hace observando las formas de onda en las salidas de las compuertas cuando responden a una entrada dada. Presentamos un

www.FreeLibros.me

Ejemplo HDL 4-9

```
//Estímulo para mux2x1_df.
module testmux;
  reg TA,TB,TS;  //entradas para mux
  wire Y;        //salida de mux
  mux2x1_df mx (TA,TB,TS,Y);  // crear un ejemplar mux
     initial
        begin
              TS = 1; TA = 0; TB = 1;
          #10 TA = 1; TB = 0;
          #10 TS = 0;
          #10 TA = 0; TB = 1;
        end
     initial
      $monitor("select = %b A = %b B = %b OUT = %b time = %0d",
               TS, TA, TB, Y, $time);
endmodule

//Descripción de flujo de datos de multiplexor de 2 a 1
//del ejemplo 4-6
module mux2x1_df (A,B,select,OUT);
   input A,B,select;
   output OUT;
   assign OUT = select ? A : B;
endmodule

Simulation log:

select = 1 A = 0 B = 1 OUT = 0 tiempo = 0
select = 1 A = 1 B = 0 OUT = 1 tiempo = 10
select = 0 A = 1 B = 0 OUT = 0 tiempo = 20
select = 0 A = 0 B = 1 OUT = 1 tiempo = 30
```

ejemplo de circuito con retardos de compuerta en la sección 3-9 (ejemplo HDL 3-3). Ahora presentaremos un ejemplo en HDL que produce la tabla de verdad de un circuito combinacional.

El análisis de circuitos combinacionales se explicó en la sección 4-2. Se analizó un circuito multinivel de un sumador completo y se dedujo su tabla de verdad por inspección. La descripción a nivel de compuertas de este circuito se ilustra en el ejemplo HDL 4-10. El circuito tiene tres entradas, dos salidas y nueve compuertas. La descripción del circuito sigue las interconexiones de las compuertas según el diagrama de la figura 4-2. El estímulo del circuito se da en el segundo módulo. Las entradas para estimular el circuito se especifican con un vector **reg** de tres bits llamado D. $D[2]$ equivale a la entrada A, $D[1]$ a la entrada B y $D[0]$ a la entrada C. Las salidas del circuito, F_1 y F_2, se declaran como **wire.** Este procedimiento sigue los pasos bosquejados en la figura 4-33. El ciclo **repeat** proporciona los siete números binarios que siguen a 000, para la tabla de verdad. El resultado de la simulación genera la tabla de verdad que se incluye junto con el ejemplo. Esa tabla demuestra que el circuito es un sumador completo.

www.FreeLibros.me

Ejemplo HDL 4-10

```
//Descripción a nivel de compuertas del circuito de la figura 4-2
module analysis (A,B,C,F1,F2);
   input   A,B,C;
   output  F1,F2;
   wire    T1,T2,T3,F2not,E1,E2,E3;
   or  g1 (T1,A,B,C);
   and g2 (T2,A,B,C);
   and g3 (E1,A,B);
   and g4 (E2,A,C);
   and g5 (E3,B,C);
   or  g6 (F2,E1,E2,E3);
   not g7 (F2not,F2);
   and g8 (T3,T1,F2not);
   or  g9 (F1,T2,T3);
endmodule

//Estímulo para analizar el circuito
module test_circuit;
  reg [2:0]D;
  wire F1,F2;
  analysis fig42(D[2],D[1],D[0],F1,F2);
  initial
    begin
     D = 3'b000;
     repeat(7)
      #10 D = D + 1'b1;
    end
  initial
     $monitor ("ABC = %b F1 = %b F2 =%b ",D, F1, F2);
endmodule


Simulation log:

ABC = 000 F1 = 0 F2 =0
ABC = 001 F1 = 1 F2 =0
ABC = 010 F1 = 1 F2 =0
ABC = 011 F1 = 0 F2 =1
ABC = 100 F1 = 1 F2 =0
ABC = 101 F1 = 0 F2 =1
ABC = 110 F1 = 0 F2 =1
ABC = 111 F1 = 1 F2 =1
```

www.FreeLibros.me

PROBLEMAS

4-1 Considere el circuito combinacional de la figura P4-1.

a) Deduzca las expresiones booleanas para T_1 a T_4. Evalúe las salidas F_1 y F_2 en función de las cuatro entradas.

b) Escriba la tabla de verdad con 16 combinaciones binarias de las cuatro variables de entrada. Luego dé en la tabla los valores binarios de T_1 a T_4 y las salidas F_1 y F_2.

c) Grafique en mapas las funciones booleanas de salida obtenidas en la parte b) y demuestre que las expresiones booleanas simplificadas son equivalentes a las obtenidas en la parte a).

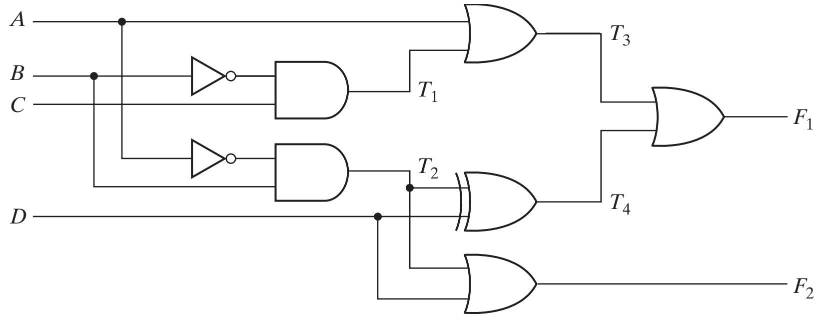


FIGURA P4-1

4-2 Obtenga las expresiones booleanas simplificadas para las salidas F y G en términos de las variables de entrada del circuito de la figura P4-2.

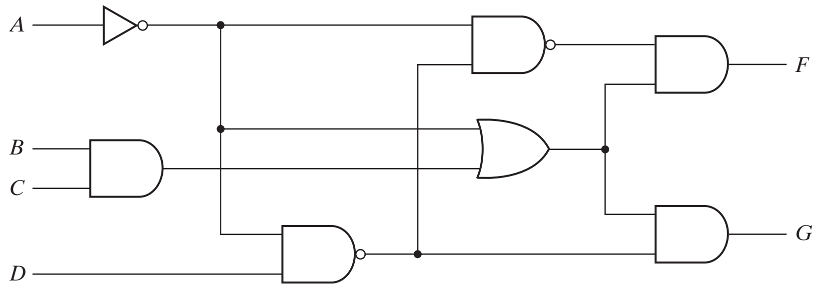


FIGURA P4-2

4-3 Para el circuito de la figura 4-26 (sección 4-10),

a) Escriba las funciones booleanas de las cuatro salidas en función de las variables de entrada.

b) Si el circuito se presenta como tabla de verdad, ¿cuántas filas y columnas tendría la tabla?

4-4 Diseñe un circuito combinacional con tres entradas y una salida. La salida es 1 cuando el valor binario de las entradas es menos que 3, y es 0 en los demás casos.

4-5 Diseñe un circuito combinacional con tres entradas, x, y y z, y tres salidas, A, B y C. Cuando la entrada binaria es 0, 1, 2 o 3, la salida binaria es uno más que la entrada. Si la entrada binaria es 4, 5, 6 o 7, la salida binaria es uno menos que la entrada.

www.FreeLibros.me

4-6 Un circuito de mayoría es un circuito combinacional cuya salida es 1 si las variables de entrada tienen más unos que ceros. La salida es 0 en caso contrario. Diseñe un circuito de mayoría de tres entradas.

4-7 Diseñe un circuito combinacional que convierta un código Gray de cuatro bits (tabla 1-6) en un número binario de cuatro bits. Implemente el circuito con compuertas OR exclusivo.

4-8 Diseñe un convertidor de código que convierta un dígito decimal del código 8, 4, −2, −1 a BCD (véase la tabla 1-5).

4-9 Un decodificador de BCD a siete segmentos es un circuito combinacional que convierte un dígito decimal BCD en un código apropiado para seleccionar segmentos de un indicador que exhibe los dígitos decimales en la forma acostumbrada. Las siete salidas del decodificador (*a*, *b*, *c*, *d*, *e*, *f*, *g*) seleccionan los segmentos correspondientes del indicador, como se indica en la figura P4-9a). La forma de representar los dígitos decimales con el indicador se muestra en la figura P4-9b). Diseñe un decodificador de BCD a siete segmentos empleando el mínimo de compuertas. Las seis combinaciones no válidas deberán dejar el indicador en blanco.

a) Designación de segmentos　　b) Designación numérica para exhibición

FIGURA P4-9

4-10 Diseñe un circuito combinacional complementador a dos, de cuatro bits. (La salida genera el complemento a dos del número binario de entrada.) Demuestre que es posible construir el circuito con compuertas OR exclusivo. ¿Puede predecir las funciones de salida para un complementador a dos de cinco bits?

4-11 Diseñe un circuito combinacional incrementador de cuatro bits. (Un circuito que suma 1 a un número binario de cuatro bits.) El circuito puede diseñarse con cuatro semisumadores.

4-12 a) Diseñe un circuito semirrestador con entradas *x* y *y*, y salidas *D* y *B*. El circuito resta los bits $x - y$, y coloca la diferencia en *D* y el préstamo (*borrow*) en *B*.

b) Diseñe un circuito restador completo con tres entradas, *x*, *y* y *z*, y dos salidas, *D* y *B*. El circuito resta $x - y - z$, donde *z* es el préstamo de entrada, *B* es el préstamo de salida y *D* es la diferencia.

4-13 El circuito sumador-restador de la figura 4-13 recibe los valores siguientes para la entrada de modo *M* y las entradas de datos *A* y *B*. En cada caso, determine los valores de las cuatro salidas *SUM*, el acarreo *C* y el desbordamiento *V*.

	M	*A*	*B*
a)	0	0111	0110
b)	0	1000	1001
c)	1	1100	1000
d)	1	0101	1010
e)	1	0000	0001

www.FreeLibros.me

4-14 Suponga que la compuerta OR exclusivo tiene un retardo de propagación de 20 ns y que las compuertas AND y OR tienen un retardo de 10 ns. Calcule el retardo de propagación total del sumador de cuatro bits de la figura 4-12.

4-15 Deduzca la expresión booleana de dos niveles para el acarreo de salida C_4 que se muestra en el generador de acarreo anticipado de la figura 4-12.

4-16 Demuestre que es posible expresar el acarreo de salida de un circuito sumador completo en la forma AND-OR-INVERT

$$C_{i+1} = G_i + P_i C_i = (G_i' P_i + G_i' C_i')'$$

El CI tipo 74182 es un circuito generador de acarreo anticipado que genera los acarreos con compuertas AND-OR-INVERT (véase la sección 3-7). El circuito supone que las terminales de entrada tienen los complementos de las G, las P y C_1. Deduzca las funciones booleanas para los acarreos anticipados C_2, C_3 y C_4 en este CI. (*Sugerencia:* Use el método de sustitución de ecuaciones para deducir los acarreos en términos de C_1'.)

4-17 Defina el acarreo propagado y el acarreo generado como

$$P_i = A_i + B_i$$
$$G_i = A_i B_i$$

respectivamente. Demuestre que el acarreo de salida y la suma de salida de un sumador completo es

$$C_{i+1} = (C_i' G_i' + P_i')'$$
$$S_i = (P_i G_i') \oplus C_i$$

El diagrama lógico de la primera etapa de un sumador paralelo de cuatro bits como el implementado en el CI tipo 74283 se reproduce en la figura P4-17. Identifique las terminales P_i' y G_i' y demuestre que el circuito implementa un sumador completo.

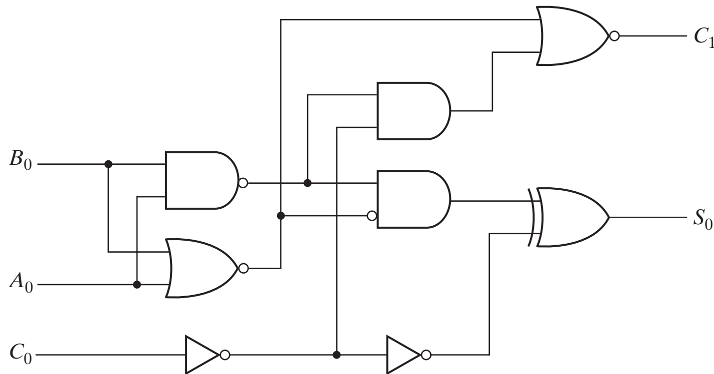


FIGURA P4-17
Primera etapa de un sumador paralelo

www.FreeLibros.me

4-18 Diseñe un circuito combinacional que genere el complemento a nueve de un dígito BCD.

4-19 Construya un circuito sumador-restador BCD. Utilice el sumador BCD de la figura 4-14 y el complementador a nueve del problema 4-18. Utilice diagramas de bloque para los componentes.

4-20 Diseñe un multiplicador binario que multiplique dos números de cuatro bits. Utilice compuertas AND y sumadores binarios.

4-21 Diseñe un circuito combinacional que compare dos números de cuatro bits para ver si son iguales. La salida del circuito es 1 si los dos números son iguales, y 0 en caso contrario.

4-22 Diseñe un decodificador de exceso-3 a binario empleando las combinaciones no utilizadas del código como condiciones de indiferencia.

4-23 Dibuje el diagrama lógico de un decodificador de 2 a 4 líneas empleando únicamente compuertas NOR. Incluya una entrada de habilitación.

4-24 Diseñe un decodificador de BCD a decimal empleando las combinaciones no utilizadas del código BCD como condiciones de indiferencia.

4-25 Construya un decodificador de 5 a 32 líneas con cuatro decodificadores de 3 a 8 líneas provistos de habilitación y un decodificador de 2 a 4 líneas. Use diagramas de bloque para los componentes.

4-26 Construya un decodificador de 4 a 16 líneas con cinco decodificadores de 2 a 4 líneas provistos de habilitación.

4-27 Se especifica un circuito combinacional con estas tres funciones booleanas:

$$F_1(A, B, C) = \Sigma(2, 4, 7)$$
$$F_2(A, B, C) = \Sigma(0, 3)$$
$$F_3(A, B, C) = \Sigma(0, 2, 3, 4, 7)$$

Implemente el circuito con un decodificador construido con compuertas NAND (similar a la figura 4-19) y compuertas NAND o AND conectadas a las salidas del decodificador. Utilice un diagrama de bloque para el decodificador. Use el mínimo de entradas en las compuertas externas.

4-28 Se define un circuito combinacional con las tres funciones booleanas siguientes:

$$F_1 = x'y'z' + xz$$
$$F_2 = xy'z' + x'y$$
$$F_3 = x'y'z + xy$$

Diseñe el circuito con un decodificador y compuertas externas.

4-29 Diseñe un codificador prioritario con las cuatro entradas de la tabla 4-8, pero asignando a la entrada D_0 la prioridad más alta, y a D_3, la más baja.

4-30 Especifique la tabla de verdad de un codificador prioritario de octal a binario. Incluya una salida V para indicar que al menos una de las entradas está presente. La entrada con el subíndice más alto tendrá prioridad. ¿Qué valor tendrán las cuatro salidas si las entradas D_5 y D_3 son 1 al mismo tiempo?

4-31 Construya un multiplexor 16×1 con dos multiplexores 8×1 y uno 2×1. Use diagramas de bloque.

4-32 Implemente la función booleana siguiente con un multiplexor:

$$F(A, B, C, D) = \Sigma(0, 1, 3, 4, 8, 9, 15)$$

4-33 Implemente un sumador completo con dos multiplexores 4×1.

www.FreeLibros.me

4-34 Un multiplexor 8×1 tiene las entradas A, B y C conectadas a las entradas de selección S_2, S_1 y S_0, respectivamente. Las entradas de datos I_0 a I_7 son: $I_1 = I_2 = I_7 = 0$; $I_3 = I_5 = 1$; $I_0 = I_4 = D$; e $I_6 = D'$. Determine la función booleana que implementa el multiplexor.

4-35 Implemente la siguiente función booleana con un multiplexor 4×1 y compuertas externas. Conecte las entradas A y B a las líneas de selección. Los requisitos de entrada de las cuatro líneas de datos serán función de las variables C y D. Estos valores se obtienen expresando F en función de C y D para cada uno de los cuatro casos en que $AB = 00, 01, 10$ y 11. Podría ser necesario implementar estas funciones con compuertas externas.

$$F(A, B, C, D) = \sum(1, 3, 4, 11, 12, 13, 14, 15)$$

4-36 Escriba la descripción HDL en el nivel de compuertas del circuito codificador prioritario de la figura 4-23.

4-37 Escriba la descripción jérárquica HDL en el nivel de compuertas de un sumador-restador de cuatro bits para números binarios sin signo. El circuito es similar a la figura 4-13 pero sin la salida V. Se puede usar un ejemplar del sumador completo de cuatro bits que se describe en el ejemplo HDL 4-2.

4-38 Escriba la descripción HDL de flujo de datos de un multiplexor cuádruple de 2 líneas a 1 con habilitación (véase la figura 4-26).

4-39 Escriba una descripción HDL del comportamiento de un comparador de cuatro bits con una salida de seis bits $Y[5:0]$. El bit 5 de Y es para igualdad, el bit 4 para desigualdad, el bit 3 para mayor que, el bit 2 para menor que, el bit 1 para mayor o igual que, y el bit 0 para menor o igual que.

4-40 Escriba una descripción HDL de flujo de datos de un sumador-restador de números sin signo de cuatro bits. Utilice el operador condicional (?:).

4-41 Repita el problema 4-40 empleando modelado de comportamiento.

4-42
a) Escriba una descripción HDL en el nivel de compuertas del circuito convertidor de BCD a exceso-3 que se ilustra en la figura 4-4.
b) Escriba una descripción de flujo de datos del convertidor de BCD a exceso-3 utilizando las expresiones booleanas de la figura 4-3.
c) Escriba una descripción HDL del comportamiento de un convertidor de BCD a exceso-3.
d) Escriba un conjunto de pruebas para simular y probar el circuito convertidor de BCD a exceso-3 y verificar la tabla de verdad. Compruebe los tres circuitos.

4-43 Explique la función del circuito especificado por la descripción HDL siguiente:

```
module Prob438 (A,B,S,E,Q);
   input  [1:0] A, B;
   input  S, E;
   output [1:0] Q;
   assign Q = E ? (S ? A : B) : 'bz;
endmodule
```

4-44 Escriba una descripción HDL del comportamiento de una unidad de aritmética-lógica (ALU) de cuatro bits. El circuito efectúa dos operaciones aritméticas y dos lógicas que se seleccionan con una entrada de dos bits. Las cuatro operaciones son suma, resta, AND y OR.

4-45 Escriba una descripción HDL del comportamiento de un codificador prioritario de cuatro entradas. Use un vector de cuatro bits para las entradas D y un bloque **always** con enunciados **if-else.** Suponga que la entrada $D[3]$ es prioritaria.

www.FreeLibros.me

REFERENCIAS

1. Dietmeyer, D. L. 1988. *Logic Design of Digital Systems,* 3a. ed. Boston: Allyn Bacon.
2. Gajski, D. D. 1997. *Principles of Digital Design.* Upper Saddle River, NJ: Prentice-Hall.
3. Hayes, J. P. 1993. *Introduction to Digital Logic Design.* Reading, MA: Addison-Wesley.
4. Katz, R. H. 1994. *Contemporary Logic Design.* Upper Saddle River, NJ: Prentice-Hall.
5. Mano, M. M. y C. R. Kime. 2000. *Logic and Computer Design Fundamentals,* 2a. ed. Upper Saddle River, NJ: Prentice-Hall.
6. Nelson V. P., H. T. Nagle, J. D. Irwin y B. D. Carroll. 1995. *Digital Logic Circuit Analysis and Design.* Upper Saddle River, NJ: Prentice-Hall.
7. Roth, C. H. 1992. *Fundamentals of Logic Design,* 4a. ed. St. Paul: West.
8. Wakerly, J. F. 2000. *Digital Design: Principles and Practices,* 3a. ed. Upper Saddle River, NJ: Prentice-Hall.
9. Bhasker, J. 1997. A *Verilog HDL Primer.* Allentown, PA: Star Galaxy Press.
10. Bhasker, J. 1998. *Verilog HDL Synthesis.* Allentown, PA: Star Galaxy Press.
11. Ciletti, M. D. 1999. *Modeling, Synthesis, and Rapid Prototyping with Verilog HDL.* Upper Saddle River, NJ: Prentice-Hall.
12. Palnitkar, S. 1996. *Verilog HDL: A Guide to Digital Design and Synthesis.* SunSoft Press (un título Prentice-Hall).
13. Thomas, D. E. y P. R. Moorby. 1998. *The Verilog Hardware Description Language,* 4a. ed. Boston: Kluwer Academic Publishers.

www.FreeLibros.me