Lógica combinacional

4

4-1 INTRODUCCION

Los circuitos lógicos para sistemas digitales pueden ser combinacionales o secuenciales. Un circuito combinacional consta de compuertas lógicas cuyas salidas en cualquier momento están determinadas en forma directa por la combinación presente de las entradas sin tomar en cuenta las entradas previas. Un circuito combinacional realiza una operación específica de procesamiento de información, especificada por completo en forma lógica por un conjunto de funciones booleanas. Los circuitos secuenciales emplean elementos de memoria (celdas binarias) además de las compuertas lógicas. Sus salidas son una función de las entradas y el estado de los elementos de memoria. El estado de los elementos de memoria, a su vez, es una función de las entradas previas. Como consecuencia, las salidas de un circuito secuencial dependen no sólo de las entradas presentes, sino también de las entradas del pasado y, el comportamiento del circuito debe especificarse en una secuencia de tiempo de entradas y de estados internos. En el Capítulo 6 se exponen los circuitos secuenciales.

En el Capítulo 1 se aprendió a reconocer los números binarios y los códigos binarios que representan cantidades discretas de información. Estas variables binarias se representan por voltajes eléctricos o alguna otra señal. Las señales pueden manipularse en las compuertas lógicas digitales para realizar las funciones requeridas. En el Capítulo 2 se introdujo el álgebra booleana como una forma para expresar de manera algebraica las funciones lógicas. En el Capítulo 3 se aprendió cómo simplificar las funciones booleanas para lograr la implementación económica de compuertas. El objetivo de este capítulo es usar el conocimiento adquirido en los capítulos previos y formular varios procedimientos sistemáticos de diseño y análisis de los circuitos combinacionales. La solución de algunos ejemplos típicos proporcionará un catálogo útil de funciones elementales importantes para el entendimiento de las computadoras y sistemas digitales.

Un circuito combinacional consta de variables de entrada, compuertas lógicas y variables de salida. Las compuertas lógicas aceptan las señales de las entradas y generan señales a las salidas. Este proceso transforma la información binaria de los datos dados de entrada en los datos requeridos de salida. En forma obvia, tanto los datos de entrada y

salida se representan por señales binarias, esto es, existen en dos valores posibles, uno representa la lógica 1 y el otro la lógica 0. En la Fig. 4-1, se muestra un diagrama de bloques de un circuito. Las n variables binarias de entrada provienen de una fuente externa; las m variables de salida van a un destino externo. En muchas aplicaciones, la fuente y/o destino son registros de almacenamiento (Sección 1-7) localizados ya sea en la proximidad del circuito combinacional o en un dispositivo externo remoto. Por definición, un registro externo no influencia el comportamiento del circuito combinacional ya que, si lo hace, el sistema total se vuelve un circuito secuencial.

Para las n variables de entrada, hay 2^n combinaciones posibles de los valores binarios de entrada. Para cada combinación posible de entrada, hay una y sólo una combinación posible de salida. Un circuito combinacional puede describirse por n funciones booleanas, una para cada variable de salida. Cada función de salida se expresa en términos de las n variables de entrada.

Cada variable de entrada a un circuito combinacional puede tener uno o dos alambres. Cuando está disponible sólo un alambre, puede representar la variable, ya sea en la forma normal (sin prima) o en la forma complementaria (con prima). Ya que una variable en una expresión booleana puede aparecer con prima y/o sin prima, es necesario proporcionar un inversor para cada literal que no está disponible en el alambre de entrada. Por otra parte, una variable de entrada puede aparecer en dos alambres, suministrando las formas tanto normal como complementaria a la entrada del circuito. En este caso, no es necesario incluir inversores para las entradas. El tipo de celdas binarias utilizadas en la mayoría de los sistemas digitales son circuitos flip-flop (Capítulo 6), que tienen salidas para los valores tanto normal como complementario de la variable binaria almacenada. En el trabajo subsecuente, se supondrá que cada variable de entrada aparece en dos alambres, suministrando en forma simultánea valores normal al igual que complementario. Debe tenerse en cuenta que un circuito inversor siempre puede suministrar el complemento de la variable si sólo está disponible un alambre.

4-2 PROCEDIMIENTO DE DISEÑO

El diseño de los circuitos combinacionales surge del planteamiento verbal del problema y termina en un diagrama de circuito lógico, o un conjunto de funciones booleaneas del cual puede obtenerse con facilidad el diagrama lógico. El procedimiento sigue estos pasos:

1. Se enuncia el problema.

Figura 4-1 Diagrama de bloques de un circuito combinacional.

2. Se determina el número de las variables de entrada disponibles y de las variables de salida requeridas.
3. Se asignan símbolos de letra a las variables de entrada y salida.
4. Se deriva la tabla de verdad que define las relaciones requeridas entre las entradas y las salidas.
5. Se obtiene la función booleana simplificada para cada salida.
6. Se dibuja el diagrama lógico.

Una tabla de verdad para un circuito combinacional consta de columnas de entrada y columnas de salida. Los 1 y 0 en las columnas de entrada se obtienen de las 2^n combinaciones binarias disponibles para las n variables de entrada. Los valores binarios para las salidas se determinan del examen del problema enunciado. Una salida puede ser igual ya sea a 0 o 1 para cada combinación válida de entrada. Sin embargo, las especificaciones pueden indicar que algunas combinaciones de entrada no ocurrirán. Estas combinaciones se vuelven condiciones no importa.

Las funciones de salida que se especifican en la tabla de verdad dan la definición exacta del circuito combinacional. Es importante que las especificaciones verbales se interpreten correctamente en una tabla de verdad. Algunas veces el diseñador debe usar su intuición y experiencia para llegar a la interpretación correcta. Las especificaciones verbales rara vez son muy completas y exactas. Cualquier interpretación equivocada que resulte en una tabla de verdad incorrecta producirá un circuito combinacional que no cubriría los requisitos enunciados.

Las funciones booleanas de salida de la tabla de verdad se simplifican por cualquier método disponible, como manipulación algebraica, el método de mapa, o el procedimiento de tabulación. Por lo común, habrá una variedad de expresiones simplificadas a elegir. No obstante, en cualquier aplicación particular ciertas restricciones, limitaciones y criterios servirán como guía en el proceso de escoger una expresión algebraica particular. Un método práctico de diseño sería tener que considerar tales restricciones como (1) número mínimo de compuertas, (2) número mínimo de entradas a una compuerta, (3) tiempo mínimo de propagación de la señal a través del circuito, (4) número mínimo de interconexiones y (5) limitaciones de las capacidades de impulsión de cada compuerta. Ya que todos estos criterios no pueden satisfacerse en forma simultánea, y ya que la importancia de cada restricción se dicta por la aplicación particular, es difícil hacer un enunciado general de lo que constituye una simplificación aceptable. En la mayoría de los casos, la simplificación principia por satisfacer un objetivo elemental, como producir una función booleana simplificada en una forma estándar y proceder de ese punto a cumplir cualesquiera otros criterios de comportamiento.

En la práctica, los diseñadores tienden a ir de la función booleana a una lista de alambrado que muestra las interconexiones entre varias compuertas lógicas estándar. En este caso, el diseño no va más allá de la función booleana simplificada de salida requerida. Sin embargo, un diagrama lógico es de ayuda para visualizar la implementación de compuertas de las expresiones.

4-3 SUMADORES

Las computadoras digitales realizan una variedad de tareas de procesamiento de información. Entre las funciones básicas encontradas están las diversas operaciones aritméticas. Sin duda, la operación aritmética más básica es la adición de dos dígitos binarios. Esta adición simple consta de cuatro operaciones elementales posibles, a saber, $0 + 0 = 0$, $0 + 1 = 1$, $1 + 0 = 1$ y $1 + 1 = 10$. Las primeras tres operaciones producen una suma cuya longitud es un dígito, pero cuando tanto los bits sumando como adendo son iguales a 1, la suma binaria consta de dos dígitos. El bit significativo más alto de este resultado se denomina acarreo. Cuando los números sumando y adendo contienen más dígitos significativos, la cuenta que se lleva obtenida por la adición de dos bits se añade al siguiente par de orden más alto de bits significativos. Un circuito combinacional que lleva a cabo la adición de dos bits se denomina *medio sumador*. Uno que lleva a cabo la adición de tres bits (dos bits significativos y una cuenta que se lleva previa) es un sumador completo. El nombre del primero proviene del hecho de que dos medios sumadores se emplean para implementar un adicionador completo. Los dos circuitos adicionadores son los primeros circuitos combinacionales que van a diseñarse.

Medio sumador

De la explicación verbal del medio sumador, se encuentra que este circuito necesita dos entradas binarias y dos salidas binarias. Las variables de entrada designan los bits sumando y adendo; las variables de salida producen la suma y el acarreo. Es necesario especificar dos variables de salida debido a que el resultado puede constar de dos dígitos binarios. Se asignan en forma arbitraria los símbolos x y y a las dos entradas y S (de suma) y C (para el acarrero) a las salidas.

Ahora que se han establecido el número y nombres de las variables de entrada y salida, ya puede formularse una tabla de verdad para identificar en forma exacta la función del medio sumador. Esta tabla de verdad se muestra a continuación:

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

El acarreo de salida es 0 a menos que ambas entradas sean 1. La salida S representa el bit menos significativo de la suma.

La función booleana simplificada de las dos salidas puede obtenerse de manera directa mediante la tabla de verdad. Las expresiones simplificadas en suma de productos son:

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

El diagrama lógico para esta implementación se muestra en la Fig. 4-2(a), lo mismo que otras cuatro implementaciones para un medio sumador. Todos logran el mismo resultado en lo que respecta al comportamiento de entrada-salida. Ilustran la flexibilidad de la que dispone el diseñador cuando implementa incluso una función lógica combinacional simple como ésta.

Como se mencionó antes, la Fig. 4-2(a) es la implementación del medio sumador en suma de productos. En la Fig. 4-2(b) se muestra la implementación en producto de sumas:

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

Para obtener la implementación de la Fig. 4-2(c), se observa que S es la OR excluyente de x y y. El complemento de S es la equivalencia de x y y (Sección 2-6):

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

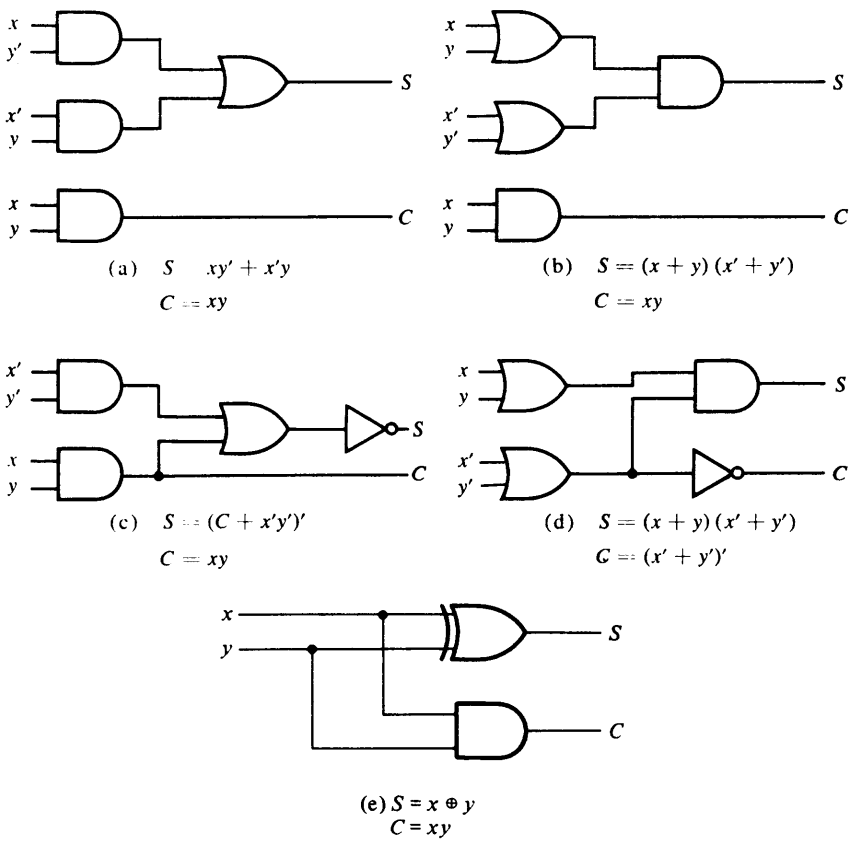


Figura 4-2 Varias implementaciones de un medio adicionador.

pero $C = xy$ y, por lo tanto, tenemos:

$$S = (C + x'y')'$$

En la Fig. 4-2(d) se utiliza la implementación de producto de sumas con C derivada como sigue:

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

el medio sumador puede implementarse con una compuerta OR excluyente y AND, como se muestra en la Fig. 4-2(c). Esta fórmula se usa posteriormente para mostrar que son necesarios dos circuitos medio sumadores para construir un circuito sumador completo.

Sumador completo

Un sumador completo es un circuito combinacional que formar la suma aritmética de tres bits de entrada. Consta de tres entradas y dos salidas. Dos de las variables de entrada, que se indican por x y y, representan los dos bits significativos que van a añadirse. La tercera entrada, z, representa la cuenta que se lleva de la posición previa significativa más baja. Son necesarias dos salidas debido a que la suma aritmética de tres dígitos binarios varía en valor desde 0 a 3 y el 2 o 3 binarios requieren dos dígitos. Las dos salidas se denotan por los símbolos S para suma y C para la cuenta que se lleva. La variable binaria S da el valor del bit menos significativo de la suma. La variable binaria C da la cuenta que se lleva de salida. La tabla de verdad del sumador completo es como sigue:

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

Los ocho renglones bajo las variables de entrada denotan todas las combinaciones posibles de 1 y 0 que pueden tener esas variables. Los 1 y 0 de las variables de salida se determinan de la suma aritmética de los bits de entrada. Cuando todos los bits de entrada son 0, la salida es 0. La salida S es igual a 1 sólo cuando una entrada es igual a 1, o cuando todas las tres entradas son iguales a 1. La salida C tiene una cuenta que se lleva de 1 si dos o tres entradas son iguales a 1.

Los bits de entrada y salida del circuito combinacional tienen diferentes interpretaciones en las diversas etapas del problema. En forma física, las señales binarias de

los alambres de entrada se consideran dígitos binarios agregados de manera aritmética para dar una suma de dos dígitos a los alambres de salida. Por otra parte, los mismos valores binarios se consideran variables de funciones booleanas cuando se expresan en la tabla de verdad o cuando el circuito se implementa con compuertas lógicas. Es importante darse cuenta de que se dan dos interpretaciones diferentes a los valores de los bits que se encuentran en este circuito.

La relación lógica de entrada-salida del circuito sumador completo puede expresarse en dos funciones booleanas, una para cada variable de salida. Cada función booleana de salida requiere un mapa único para su simplificación. Cada mapa debe tener ocho cuadros, ya que cada salida es una función de tres variables de entrada. Los mapas en la Fig. 4-3 se utilizan para simplificar las dos funciones de salida. Los 1 en los cuadros de los mapas de S y C se determinan en forma directa mediante la tabla de verdad. Los cuadros con 1 para la salida S no se combinan en cuadros adyacentes para dar una expresión simplificada en suma de productos. La salida C puede simplificarse a una expresión de seis literales. El diagrama lógico para el sumador completo implementado en suma de productos se muestra en la Fig. 4-4. En esta implementación se usan las expresiones booleanas siguientes:

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

Pueden desarrollarse otras configuraciones para un adicionador completo. La implementación en producto de sumas requiere el mismo número de compuerta como en la Fig. 4-4, con el número de compuertas AND y OR intercambiado. Un sumador completo puede implementarse con dos medios sumadores y una compuerta OR, como se muestra en la Fig. 4-5. La salida S del segundo medio sumador es la OR excluyente de z y la salida del primer medio sumador, dando:

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

y el acarreo de salida es:

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

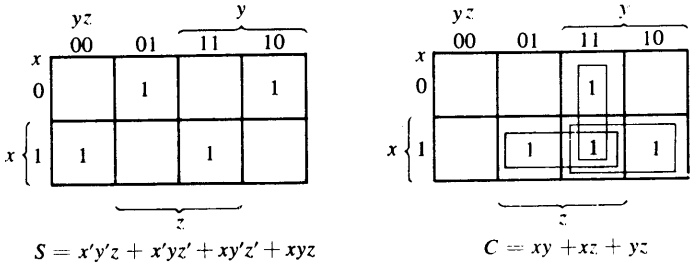


Figura 4-3 Mapas para un sumador completo.

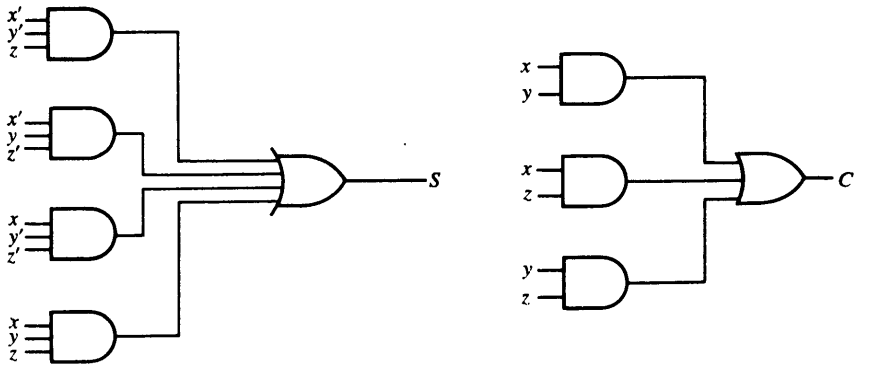


Figura 4-4 Implementación de un sumador completo en suma de productos.

4-4 RESTADORES

La sustracción de dos números binarios puede llevarse a cabo tomando el complemento del sustraendo y agregándolo al minuendo (Sección 1-5). Por este método, la operación de sustracción llega a ser una operación de división que requiere sumadores completos para su implementación en máquina. Es posible implementar la sustracción con circuitos lógicos en una forma directa, como se hace con lápiz y papel. Por este método cada bit sustraendo del número se sustrae de su bit minuendo correspondiente significativo para formar un bit de diferencia. Si el bit minuendo es menor que el bit sustraendo, se toma un 1 de la siguiente posición significativa. El hecho de que se ha tomado un 1 debe llevarse al siguiente par más alto de bit mediante una señal binaria que llega de fuera (salida) de una etapa dada y va a (entrada) la siguiente etapa más alta. En forma precisa así como hay medio sumadores y sumadores completos, hay medio restadores y restadores completos.

Medio restador

Un medio restador es un circuito combinacional que sustrae dos bits y produce su diferencia. También tiene una salida para especificar si se ha tomado un 1. Se designa el bit minuendo por x y el bit sustraendo mediante y. Para llevar a cabo $x - y$, tienen

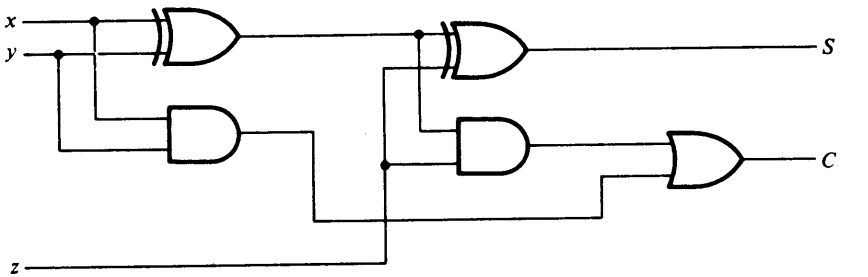


Figura 4-5 Implementación de un sumador completo con dos medio adicionadores y una compuerta OR.

que verificarse las magnitudes relativas de x y y. Si $x \geq y$, se tienen tres posibilidades; $0 - 0 = 0$, $1 - 0 = 1$ y, $1 - 1 = 0$. El resultado se denomina *bit de diferencia*. Si $x < y$, tenemos $0 - 1$ y es necesario tomar un 1 de la siguiente etapa más alta. El 1 que se toma de la siguiente etapa más alta añade 2 al bit minuendo, de la misma forma que en el sistema decimal lo que se toma añade 10 a un dígito minuendo. Con el minuendo igual a 2, la diferencia llega a ser $2 - 1 = 1$. El medio restador requiere dos salidas. Una salida genera la diferencia y se denotará por el símbolo D. La segunda salida, denotada B para lo que se toma, genera la señal binaria que informa a la siguiente etapa que se ha tomado un 1. La tabla de verdad para las relaciones de entrada-salida de un medio restador ahora puede derivarse como sigue:

x	y	B	D
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

La salida que toma B es un 0 en tanto que $x \geq y$. Es un 1 para $x = 0$ y $y = 1$. La salida D es el resultado de la operación aritmética $2B + x - y$.

Las funciones booleanas para las dos salidas del medio restador se derivan de manera directa de la tabla de verdad:

$$D = x'y + xy'$$
$$B = x'y$$

Es interesante observar que la lógica para D es exactamente la misma que la lógica para la salida S en el medio sumador.

Restador completo

Un restador completo es un circuito combinacional que lleva a cabo una sustracción entre dos bits, tomando en cuenta que un 1 se ha tomado por una etapa significativa más baja. Este circuito tiene tres entradas y dos salidas. Las tres entradas, x, y y z, denotan al minuendo, sustraendo y a la toma previa, respectivamente. Las dos salidas, D y B, representan la diferencia y la salida tomada, respectivamente. La tabla de verdad para el circuito es como sigue:

x	y	z	B	D
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Los ocho renglones bajo las variables de entrada designan todas las combinaciones posibles de 1 y 0 que pueden tomar las variables binarias. Los 1 y 0 para las variables de salida están determinados por la sustracción de $x - y - z$. Las combinaciones que tienen salida de toma $z = 0$ se reducen a las mismas cuatro condiciones del medio sumador. Para $x = 0$, $y = 0$ y $z = 1$, tiene que tomarse un 1 de la siguiente etapa, lo cual hace $B = 1$ y añade 2 a x. Ya que $2 - 0 - 1 = 1$, $D = 1$. Para $x = 0$ y $yz = 11$, necesita tomarse otra vez, haciendo $B = 1$ y $x = 2$. Ya que $2 - 1 - 1 = 0$, $D = 0$. Para $x = 1$ y $yz = 01$, se tiene $x - y - z = 0$, lo cual hace $B = 0$ y $D = 0$. Por último, para $x = 1, y = 1, z = 1$, tiene que tomarse 1, haciendo $B = 1$ y $x = 3$ y, $3 - 1 - 1 = 1$, haciendo $D = 1$.

La función booleana simplificada para las dos salidas del restador completo se derivan en los mapas de la Fig. 4-6. Las funciones simplificadas de salida en suma de productos son:

$$D = x'y'z + x'yz' + xy'z' + xyz$$
$$B = x'y + x'z + yz$$

De nuevo se observa que la función lógica para la salida D en el restador completo es exactamente la misma que para la salida S del sumador completo. Además, la salida B se asemeja a la función para C en el sumador completo, excepto que la variable de entrada x está complementada. Debido a estas similitudes, es posible convertir un sumador completo en un restador completo, complementando tan sólo la entrada x antes de su aplicación a las compuertas que forman la salida de acarreo.

4-5 CONVERSION DE CODIGO

La disponibilidad de una gran variedad de códigos para los mismos elementos discretos de información origina el uso de códigos diferentes por sistemas digitales diferentes. Algunas veces es necesario usar la salida de un sistema como la entrada a otro. Debe insertarse un circuito de conversión entre los dos sistemas si cada uno utiliza códigos diferentes para la misma información. Así que, un convertidor de código es un circuito que hace dos sistemas compatibles aun cuando cada uno use un código binario diferente.

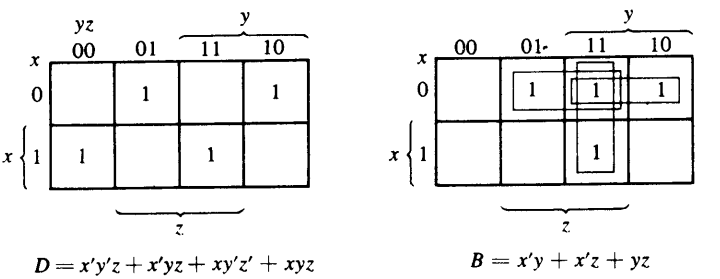


Figura 4-6 Mapas para un restador completo.

Para convertir un código binario *A* en el código binario *B*, las líneas de entrada deben suministrar la combinación bit de elementos como los especifica el código *A* y las líneas de salida, deben generar la combinación bit correspondiente del código *B*. Un circuito combinacional lleva a cabo esta transformación mediante compuertas lógicas. El procedimiento de diseño de los convertidores de código se ilustrará mediante un ejemplo específico de conversión del código BCD en el código exceso-3.

Las combinaciones bit para los códigos BCD y exceso-3 se listan en la Tabla 1-2 (Sección 1-6). Ya que cada código usa cuatro bits para representar un dígito decimal, debe haber cuatro variables de entrada y cuatro variables de salida. Permítase designar las cuatro variables de entrada con los símbolos *A, B, C* y *D* y las cuatro variables de salida por *w, x, y* y *z*. La tabla de verdad que relaciona las variables de entrada y salida se muestra en la Tabla 4-1. Las combinaciones bit de las entradas y sus correspondientes salidas se obtienen de manera directa de la Tabla 1-2. Se observa que cuatro variables binarias pueden tener 16 combinaciones bit, sólo 10 de las cuales se listan en la tabla de verdad. Las seis combinaciones bit que no se listan para las variables de *entrada* son combinaciones no importa. Ya que nunca ocurrirán, se tiene la libertad de asignar las variables de salida ya sea con un 1 o un 0, el que dé un circuito más simple.

Los mapas en la Fig. 4-7 están dibujados para obtener una función booleana simplificada para cada salida. Cada uno de los cuatro mapas en la Fig. 4-7 representa una de las cuatro salidas de este circuito como una función de las cuatro variables de entrada. Los 1 que se marcan en el interior de los cuadros se obtienen de los mintérminos que hacen la salida igual a 1. Los 1 se obtienen de la tabla de verdad pasando sobre las columnas de salida una a la vez. Por ejemplo, la columna bajo la salida *z* tiene cinco números 1; por tanto, el mapa para *z* puede tener cinco 1, cada uno en un cuadro correspondiente al mintérmino que hace que *z* sea igual a 1. Las seis combinaciones no importa se marcan con letras *X*. Una forma posible de simplificar las funciones en suma de productos se lista bajo el mapa de cada variable.

TABLA 4-1 Tabla de verdad para el ejemplo de conversión de código

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

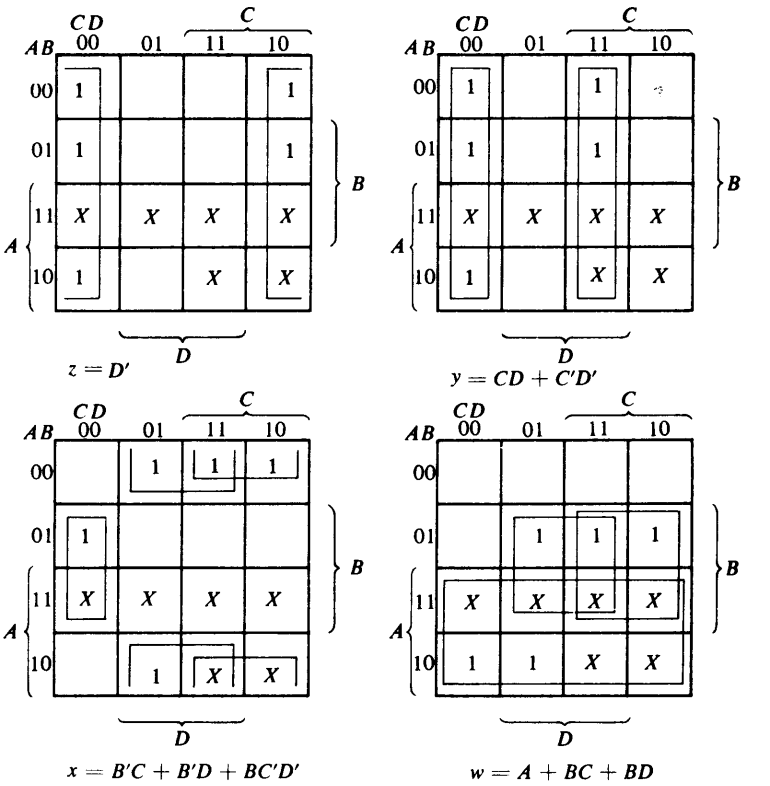


Figura 4-7 Mapas para un convertidor de código BCD-a-exceso-3.

Puede obtenerse de manera directa un diagrama lógico de dos niveles mediante las expresiones booleanas derivadas por los mapas. Hay otras posibilidades diferentes para un diagrama lógico que implemente este circuito. Las expresiones que se obtienen en la Fig. 4-7 pueden manipularse en forma algebraica con el objeto de usar compuertas comunes para dos o más salidas. Esta manipulación, que se muestra a continuación, ilustra la flexibilidad que se obtiene con sistemas de salidas múltiples cuando se implementan con tres o más niveles de compuertas.

$$z = D'$$
$$y = CD + C'D' = CD + (C + D)'$$
$$x = B'C + B'D + BC'D' = B'(C + D) + BC'D'$$
$$= B'(C + D) + B(C + D)'$$
$$w = A + BC + BD = A + B(C + D)$$

El diagrama lógico que implementa las expresiones anteriores se muestra en la Fig. 4-8. En él puede verse que la compuerta OR cuya salida es $C + D$ se ha utilizado para implementar parcialmente cada una de las tres salidas.

Sin contar los invertidores de entradas, la implementación en suma de productos requiere siete compuertas AND y tres compuertas OR. La implementación de la Fig. 4-8 requiere cuatro compuertas AND, cuatro OR y un inversor. Si sólo están disponibles entradas normales, la primera implementación requerirá invertidores para las variables *B, C* y *D,* en tanto que la segunda implementación necesita invertidores para las variables *B* y *D.*

4-6 PROCEDIMIENTO DE ANALISIS

El diseño de un circuito combinacional se inicia con las especificaciones verbales de una función requerida y culmina con un conjunto de funciones booleanas de salida o un diagrama lógico. El *análisis* de un circuito combinacional es en cierta forma el proceso inverso. Principia con un diagrama lógico dado y termina con un conjunto de funciones booleanas, una tabla de verdad o una explicación verbal de la operación del circuito. Si el diagrama lógico que va a analizarse se acompaña con una función nombre o una explicación de lo que se supone que realiza, entonces el problema del análisis se reduce a una verificación de la función enunciada.

El primer paso en el análisis es tener la seguridad de que el circuito dado es combinacional y no secuencial. El diagrama de un circuito combinacional tiene compuertas lógicas sin trayectorias de retroalimentación o elementos de memoria. Una trayectoria de retroalimentación es una conexión de la salida de una compuerta a

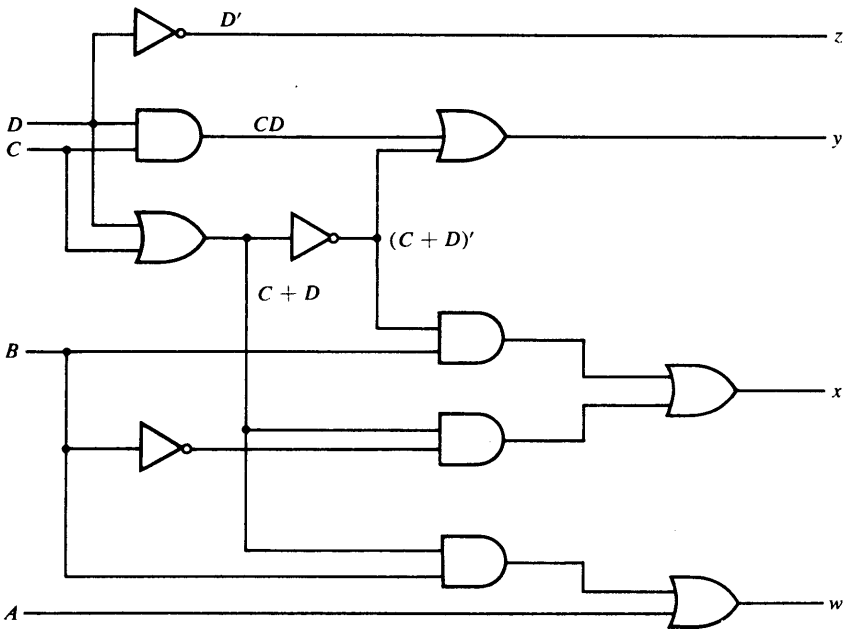


Figura 4-8 Diagrama lógico para un convertidor de código BCD-a-exceso-3.

la entrada de una segunda compuerta que forma parte de la entrada a la primera compuerta. Las trayectorias de retroalimentación o elementos de memoria en un circuito digital definen un circuito secuencial y deben analizarse de acuerdo con los procedimientos delineados en el Capítulo 6.

Una vez que se ha verificado que el diagrama lógico es un circuito combinacional, puede procederse a obtener las funciones booleanas de salida y/o la tabla de verdad. Si el circuito está acompañado por una explicación verbal de su función, entonces las funciones booleanas o la tabla de verdad son suficientes para la verificación. Si la función del circuito está bajo investigación, entonces es necesario interpretar la operación del circuito mediante la tabla de verdad derivada. El éxito de tal investigación se favorece si se tiene experiencia previa y familiaridad con una amplia variedad de circuitos digitales. La habilidad para correlacionar una tabla de verdad con una tarea de procesamiento de información es un arte que se adquiere con la experiencia.

Para obtener las funciones booleanas de salida de un diagrama lógico, se procede como sigue:

1. Se etiquetan con símbolos arbitrarios todas las salidas de compuerta que son una función de las variables de entrada. Se obtienen las funciones booleanas para cada compuerta.
2. Se etiquetan con otros símbolos arbitrarios las compuertas que son una función de las variables de entrada y/o compuertas previamente etiquetadas. Se encuentran las funciones booleanas para esas compuertas.
3. Se repite el proceso delineado en el paso 2 hasta que se han obtenido las salidas del circuito.
4. Por sustitución repetida de las funciones previamente definidas, se obtienen las funciones booleanas de salida en términos sólo de las variables de entrada.

El análisis del circuito combinacional en la Fig. 4-9 ilustra el procedimiento propuesto. Se observa que el circuito tiene tres entradas binarias, A, B y C, y dos salidas binarias, F_1 y F_2. Las salidas de las diversas compuertas se etiquetan con símbolos intermedios. Las salidas de las compuertas que son una función de las variables de entrada sólo son F_2, T_1 y T_2. Las funciones booleanas para estas tres salidas son:

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

A continuación se consideran las salidas de compuertas que son una función de los símbolos ya definidos:

$$T_3 = F_2' T_1$$
$$F_1 = T_3 + T_2$$

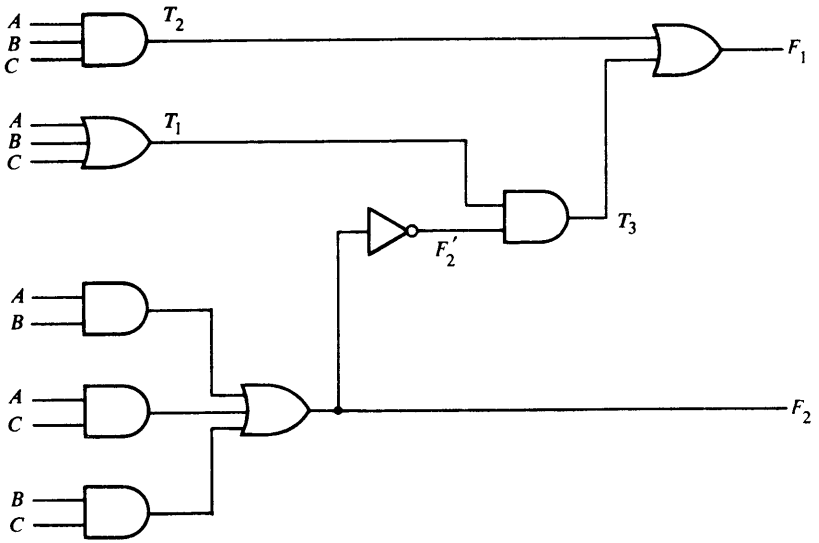


Figura 4-9 Diagrama lógico para el ejemplo de análisis.

La función booleana de salida F_2 expresada con anterioridad ya está dada como una función sólo de las entradas. Para obtener F_1 como una función de A, B y C se forma una serie de sustituciones como sigue:

$$\begin{aligned} F_1 &= T_3 + T_2 = F_2'T_1 + ABC = (AB + AC + BC)'(A + B + C) + ABC \\ &= (A' + B')(A' + C')(B' + C')(A + B + C) + ABC \\ &= (A' + B'C')(AB' + AC' + BC' + B'C) + ABC \\ &= A'BC' + A'B'C + AB'C' + ABC \end{aligned}$$

Si se desea proseguir la investigación y determinar la tarea de transformación de la información realizada por este circuito, puede derivarse la tabla de verdad en forma directa de las funciones booleanas y tratar de reconocer una operación familiar. Por este ejemplo, se observa que el circuito es un sumador completo, con F_1 como la salida de suma y F_2 la salida de la cuenta que se lleva. A, B y C son las tres entradas agregadas en forma aritmética.

La derivación de la tabla de verdad para el circuito es un proceso directo una vez que se conocen las funciones booleanas de salida. Para obtener la tabla de verdad de manera directa del diagrama lógico sin pasar a través de las derivaciones de las funciones booleanas, se procede como sigue:

1. Determínese el número de las variables de entrada al circuito. Para n entradas, fórmense las 2^n combinaciones posibles de entrada de 1 y 0 mediante el listado de los números binarios de 0 a $2^n - 1$.
2. Etiquétense las salidas de las compuertas seleccionadas con símbolos arbitrarios.

3. Obténgase la tabla de verdad para las salidas de las compuertas que son una función sólo de las variables de entrada.
4. Procédase a obtener la tabla de verdad para las salidas de las compuertas que son una función de los valores previamente definidos hasta que las columnas para todas las salidas estén determinadas.

Este proceso puede usarse utilizando el circuito en la Fig. 4-9. En la Tabla 4-2, se forman las ocho combinaciones posibles de las tres variables de entrada. La tabla de verdad para F_2 se determina de manera directa de los valores de A, B y C, con $F = 1$ para cualquier combinación que tenga dos de las tres entradas iguales a 1. La tabla de verdad para F'_2 es el complemento de F_2. Las tablas de verdad para T_1 y T_2 son las funciones OR y AND de las tres variables de entrada, respectivamente. Los valores para T_3 se derivan mediante T_1 y F'_2: T_3 es igual a 1 cuando tanto T_1 y F'_2 son iguales a 1 y a 0 de otra manera. Por último, F_1 es igual a 1 para las combinaciones en las cuales T_2 o T_3, o ambas, son iguales a 1. La inspección de las combinaciones en la tabla de verdad para A, B, C, F_1 y F_2 de la Tabla 4-2 muestra que es idéntica a la tabla de verdad del sumador completo que se presenta en la Sección 4-3 para x, y, z, S y C, respectivamente.

Considérese ahora un circuito combinacional que tiene combinaciones no importa de entrada. Cuando se diseña un circuito como éste, las combinaciones no importa se marcan con X en el mapa y se les asigna una salida de 1 o bien 0, lo que sea más conveniente para la simplificación de la función booleana de salida. Cuando se analiza, un circuito con combinaciones no importa la situación es por completo diferente. Aun cuando se supone que las combinaciones no importa de entrada nunca ocurrirán, el hecho es que si cualquiera de esas combinaciones se aplica a las entradas (intencionalmente o por error), estará presente una salida binaria. El valor de la salida dependerá de la elección para las X que se toma durante el diseño. Parte del análisis de tal circuito puede implicar la determinación de los valores de salida para las combinaciones no importa de entrada. Como ejemplo, considérese el convertidor de códigos BCD-a-exceso-3 diseñado en la Sección 4-5. Las salidas obtenidas cuando las seis combinaciones que no se utilizan del código BCD se aplican a las entradas son:

Entradas *BCD* sin uso				Salidas			
A	*B*	*C*	*D*	*w*	*x*	*y*	*z*
1	0	1	0	1	1	0	1
1	0	1	1	1	1	1	0
1	1	0	0	1	1	1	1
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	1
1	1	1	1	1	0	1	0

Estas salidas pueden derivarse mediante el método de análisis de la tabla como se delinea en esta sección. En este caso particular, las salidas pueden obtenerse de manera directa de los mapas en la Fig. 4-7. Por la inspección de los mapas, se determina cuándo las X en los cuadros del mintérmino correspondiente para cada salida se han

TABLA 4-2 Tabla de verdad para el diagrama lógico de la Fig. 4-9

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

incluido con los 1 o los 0. Por ejemplo, el cuadro para el mintérmino m_{10} (1010) se ha incluido con los 1 para las salidas w, x y z, pero no para y. Por lo tanto, las salidas para m_{10} son $wxyz = 1101$, como se lista en la tabla anterior. También se observa que las primeras tres salidas de la tabla no tienen significado en el código exceso-3, y que las últimas tres salidas corresponden al decimal 5, 6, y 7, respectivamente. Esta coincidencia es por entero una función de la elección para las X tomadas durante el diseño.

4-7 CURCUITOS NAND DE NIVEL MULTIPLE

Los circuitos combinacionales se construyen más a menudo con compuertas NAND o NOR, más bien que con compuertas AND y OR. Las compuertas NAND y NOR son más comunes desde el punto de vista del hardware, ya que están disponibles en la forma de circuitos integrados. Debido a la preeminencia de las compuertas NAND y NOR en el diseño de los circuitos combinacionales, es importante tener la capacidad de reconocer las relaciones que existen entre los circuitos construidos con compuertas AND-OR y sus diagramas equivalentes NAND o NOR.

La implementación de diagramas lógicos en dos niveles NAND y NOR se presentó en la Sección 3-6. Aquí se considera el caso más general de circuitos de niveles múltiples. El procedimiento para obtener circuitos NAND se presenta en esta sección, y para obtener los circuitos NOR en la siguiente sección.

Compuerta universal

La compuerta NAND se dice que es una compuerta universal porque cualquier sistema digital puede implementarse con ella. Los circuitos combinacionales al igual que los secuenciales pueden construirse con esta compuerta, debido a que el circuito flip-flop (el elemento de memoria de uso más frecuente en los circuitos secuenciales) puede construirse mediante dos compuertas NAND conectadas de la parte posterior de una a la de otra, como se muestra en la Sección 6-2.

Para mostrar que cualquier función booleana puede implementarse con compuertas NAND sólo se necesita mostrar que las operaciones lógicas AND, OR y NOT

pueden implementarse con compuertas NAND. La implementación de las operaciones AND, OR y NOT con compuertas NAND se muestra en la Fig. 4-10. La operación NOT se obtiene mediante una compuerta NAND de una entrada, que es en realidad otro símbolo para un circuito inversor. La operación AND requiere dos compuertas NAND. La primera produce el inversor AND y la segunda actúa como un inversor para producir la salida normal. La operación OR se lleva a cabo a través de una compuerta NAND con inversores adicionales en cada salida.

Una forma conveniente de implementar un circuito combinacional con compuertas NAND es obtener las funciones booleanas simplificadas en términos de AND, OR y NOT y convertir las funciones en la lógica NAND. La conversión de la expresión algebraica para operaciones AND, OR y NOT en operaciones NAND, por lo común es bastante complicada debido a que implica un gran número de aplicaciones del teorema de De Morgan. Esta dificultad se evita por el uso de simples manipulaciones de circuito y simples reglas, como se delinea abajo.

Implementación de una función booleana
Método de diagrama de bloques

La implementación de funciones booleanas con compuertas NAND puede obtenerse mediante una técnica de manipulación de un simple diagrama de bloques. El método requiere que se dibujen otros dos diagramas lógicos antes para obtener el diagrama lógico NAND. No obstante, el procedimiento es muy sencillo y directo.

1. Mediante la expresión algebraica, dibújese el diagrama lógico con compuertas AND, OR y NOT. Se supone que están disponibles entradas tanto normal como complementaria.
2. Dibújese un segundo diagrama lógico con la lógica NAND equivalente, como se da en la Fig. 4-10, para sustituir cada compuerta AND, OR y NOT.

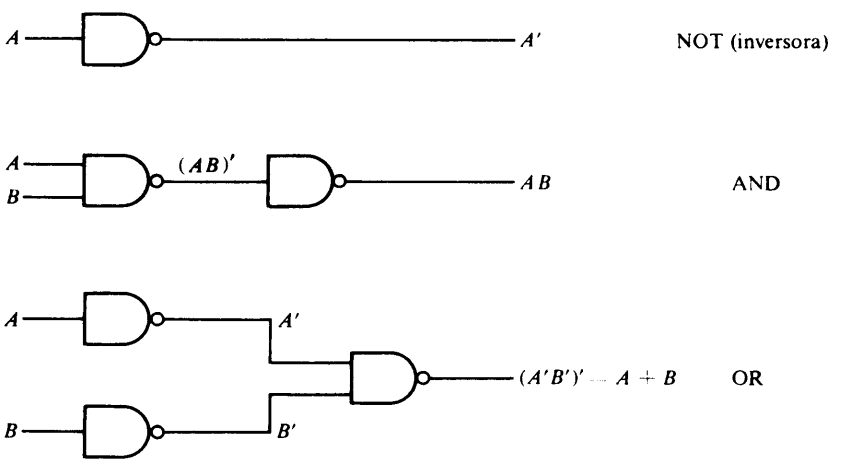


Figura 4-10 Implementación de compuertas NOT, AND u OR con compuertas.

3. Elimínense del diagrama cualesquiera dos inversores en cascada, ya que la inversión doble no realiza una función lógica. Elimínense los inversores conectados a entradas únicas alternas y compleméntense las variables de entrada correspondientes. El nuevo diagrama lógico que se obtiene es la implementación requerida en compuerta NAND.

Este procedimiento se ilustra en la Fig. 4-11 para la función:

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

La implementación AND-OR de esta función se muestra en el diagrama lógico de la Fig. 4-11(a). Para cada compuerta AND, se sustituye una compuerta AND seguida por un inversor; para cada compuerta OR, se sustituyen inversores de entrada seguidos por una compuerta NAND. Esta sustitución es consecuencia directa de las equivalencias lógicas de la Fig. 4-10 y se muestra en el diagrama en la Fig. 4-11(b). Este diagrama tiene siete inversores y cinco compuertas de dos entradas NAND listadas con números dentro del símbolo de compuerta. Los pares de inversores conectados en cascada (de cada casilla AND a cada casilla OR) se eliminan, ya que forman una inversión doble. El inversor conectado a la entrada B se elimina y la variable de entrada se designa por B'. El resultado es el diagrama lógico NAND que se muestra en la Fig. 4-11(c), con el número dentro de cada símbolo que identifica la compuerta en la Fig. 4-11(b).

En este ejemplo se demuestra que el número requerido de compuertas NAND para implementar la función booleana es igual al número de compuertas AND-OR, siempre que estén disponibles las entradas tanto normal como de complemento. Si sólo están disponibles las entradas normales, deben usarse inversores para generar cualesquiera entradas complementarias requeridas.

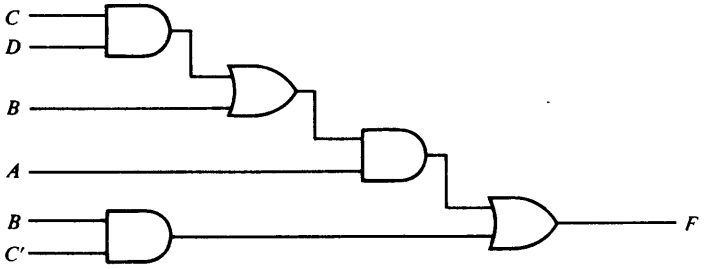
Un segundo ejemplo de implementación NAND se muestra en la Fig. 4-12. La función booleana que se implantará es:

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

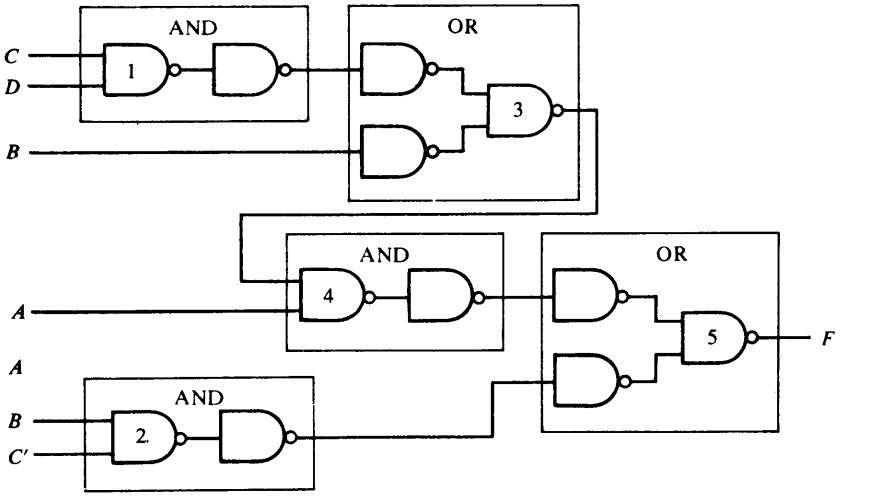
La implementación AND-OR se muestra en la Fig. 4-12(a) y su sustitución en lógica NAND, en la Fig. 4-12(b). Puede eliminarse un par de inversores en cascada. Las tres entradas externas E, A y B', las cuales van directamente a los inversores, se complementan y se eliminan los inversores correspondientes. La implementación final en compuerta NAND se muestra en la Fig. 4-12(c).

El número de compuertas NAND para el segundo ejemplo es igual al número de compuertas AND-OR más un inversor adicional en la salida (compuerta NAND 5). En general, el número de compuertas NAND requeridas para implementar una función es igual al número de compuertas AND-OR, excepto por un inversor ocasional. Esto es cierto siempre que estén disponibles las entradas tanto normal como complementaria, debido a que la conversión obliga a que se complementen ciertas variables de entrada.

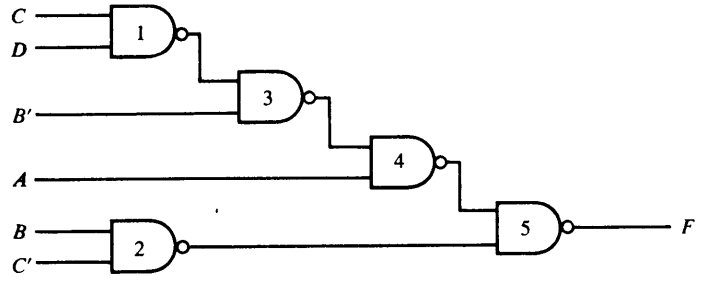
El método de diagrama de bloques es un poco cansado, ya que requiere el dibujo de dos diagramas lógicos para obtener la respuesta de un tercero. Con cierta experien-



(a) implementación AND/OR

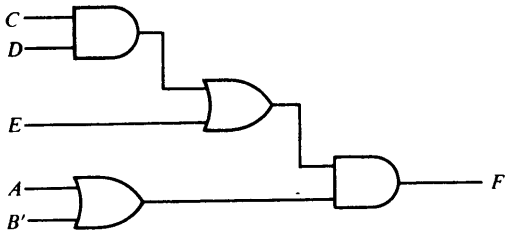


(b) Sustitución con funciones equivalentes NAND de las de la Fig. 5-8

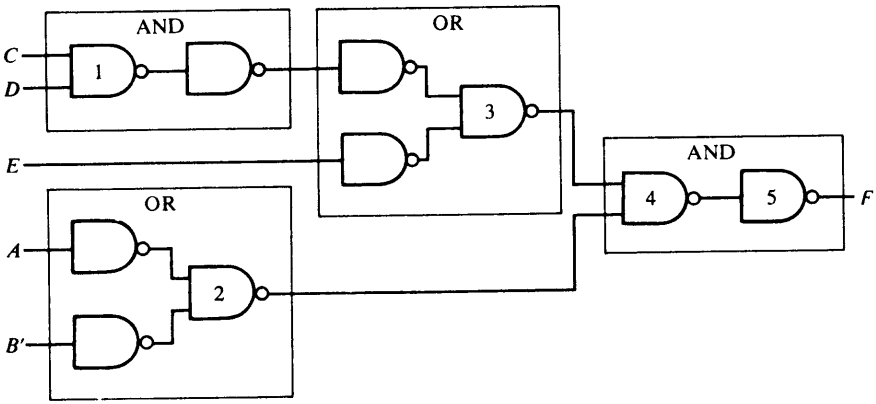


(c) Implementación NAND

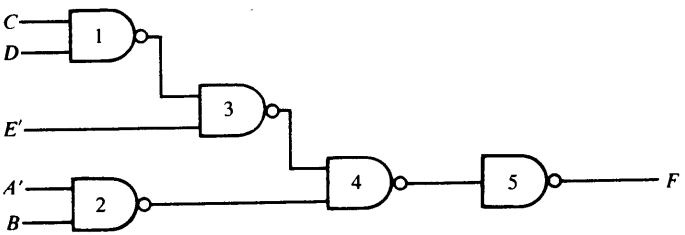
Figura 4-11 Implementación de $F = A(B + CD) + BC'$ con compuertas NAND.



(a) implementación AND/OR



(b) Sustitución con funciones equivalentes NAND



(c) Implementación NAND

Figura 4-12 Implementación de *(A + B') (CD + E)* con compuertas NAND.

cia, es posible reducir la cantidad de trabajo anticipando los pares de inversores en cascada y los inversores en las entradas. A partir del procedimiento que acaba de delinearse, no es difícil derivar reglas generales para la implementación de funciones booleanas con compuertas NAND en forma directa de una expresión algebraica.

Procedimiento de análisis

En el procedimiento anterior se consideró el problema de derivar un diagrama lógico NAND de una función booleana dada. El procedimiento inverso es el análisis del

problema que principia con un diagrama lógico NAND dado y termina con una expresión booleana o una tabla de verdad. El análisis de diagramas lógicos NAND sigue los mismos procedimientos que se presentaron en la Sección 4-6 para el análisis de circuitos combinacionales. La única diferencia es que la lógica NAND requiere una aplicación repetida del teorema de De Morgan. Ahora se demostrará la derivación de la función booleana mediante un diagrama lógico. Entonces se mostrará la derivación de la tabla de verdad de manera directa a partir del diagrama lógico NAND. Por último, se presenta un método para convertir un diagrama lógico AND-OR mediante la manipulación en diagrama de bloques.

Derivación de la función booleana por manipulación algebraica

El procedimiento para derivar la función booleana de un diagrama lógico se delineó en la Sección 4-6. Este procedimiento se demuestra para el diagrama lógico NAND que se ilustra en la Fig. 4-13 y es el mismo que se encuentra en la Fig. 4-11(c). Primero, todas las salidas de compuerta se etiquetan con símbolos arbitrarios. Segundo, se derivan las funciones booleanas para las salidas de compuerta que reciben solamente entradas externas:

$$T_1 = (CD)' = C' + D'$$
$$T_2 = (BC')' = B' + C$$

La segunda forma se sigue en forma directa del teorema de De Morgan y a veces puede ser de uso más conveniente. Tercero, las funciones booleanas de compuertas que tienen entrada de funciones derivadas previamente se determinan en orden consecutivo hasta que la salida se expresa en términos de las variables de entrada:

$$\begin{aligned} T_3 &= (B'T_1)' = (B'C' + B'D')' \\ &= (B + C)(B + D) = B + CD \\ T_4 &= (AT_3)' = [A(B + CD)]' \\ F &= (T_2T_4)' = \{(BC')'[A(B + CD)]'\}' \\ &= BC' + A(B + CD) \end{aligned}$$

Derivación de la tabla de verdad

El procedimiento para obtener en forma directa la tabla de verdad a partir de un diagrama lógico también se delinea en la Sección 4-6. Este procedimiento se demuestra para el diagrama lógico NAND que se ilustra en la Fig. 4-13. Primero, las cuatro variables de entrada, junto con sus 16 combinaciones de número 1 y 0, se listan como se muestra en la Tabla 4-3. Segundo, las salidas de todas las compuertas se etiquetan con símbolos arbitrarios como en la Fig. 4-13. Tercero, se obtiene la tabla de verdad para las salidas de las compuertas que son una función sólo de las variables de entrada.

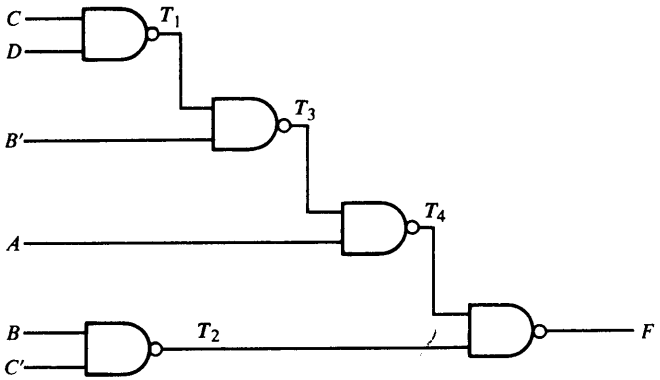
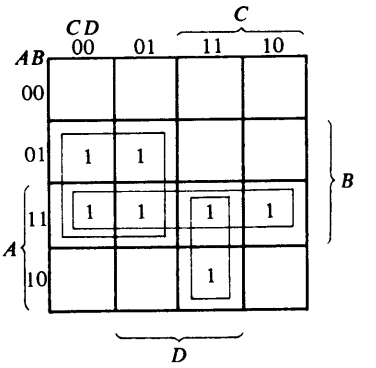


Figura 4-13 Ejemplo de análisis.

Estas son T_1 y T_2. $T_1 = (CD)'$; de modo que se marcan 0 en los renglones donde tanto C como D son iguales a 1 y se llenan los demás renglones de T_1 con números 1. También, $T_2 = (BC')'$; de modo que se marcan números 0 en los renglones donde $B = 1$ y $C = 0$, y se llenan los demás renglones de T_2 con números 1. Se procede entonces a obtener la tabla de verdad para las salidas de las compuertas que son una función de las salidas que se definieron con anterioridad hasta que la columna para la salida F queda determinada. Ahora es posible obtener una expresión algebraica para la salida mediante la tabla de verdad derivada. El mapa que se muestra en la Fig. 4-14 se obtiene de manera directa de la Tabla 4-3 y tiene números 1 en los cuadros de los mintérminos

TABLA 4-3 Tabla de verdad del circuito de la Figura 4-13

A	B	C	D	T_1	T_2	T_3	T_4	F
0	0	0	0	1	1	0	1	0
0	0	0	1	1	1	0	1	0
0	0	1	0	1	1	0	1	0
0	0	1	1	0	1	1	1	0
0	1	0	0	1	0	1	1	1
0	1	0	1	1	0	1	1	1
0	1	1	0	1	1	1	1	0
0	1	1	1	0	1	1	1	0
1	0	0	0	1	1	0	1	0
1	0	0	1	1	1	0	1	0
1	0	1	0	1	1	0	1	0
1	0	1	1	0	1	1	0	1
1	1	0	0	1	0	1	0	1
1	1	0	1	1	0	1	0	1
1	1	1	0	1	1	1	0	1
1	1	1	1	0	1	1	0	1



$$F = AB + BC' + ACD$$

Figura 4-14 Derivación de F a partir de la Tabla 4-3.

para los cuales F es igual a 1. La expresión simplificada que se obtiene mediante el mapa es:

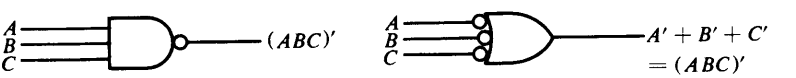
$$F = AB + ACD + BC' = A(B + CD) + BC'$$

Esta es igual a la expresión que se muestra en la Fig. 4-11, por tanto se verifica la respuesta correcta.

Transformación del diagrama de bloque

Algunas veces es conveniente convertir un diagrama lógico NAND en un diagrama lógico AND-OR equivalente para facilitar el procedimiento de análisis. Al hacer esto, la función booleana puede derivarse con más facilidad sin emplear el teorema de De Morgan. La conversión de diagramas lógicos se lleva a cabo a través de un proceso inverso del que se utiliza para la implementación. En la Sección 3-6, se mostraron dos símbolos gráficos alternos para la compuerta NAND. Estos símbolos se repiten en la Fig. 4-15 por motivos de comodidad. Por el uso juicioso de ambos símbolos, es posible convertir un diagrama NAND en una forma AND-OR equivalente.

La conversión de un diagrama lógico NAND en un diagrama AND-OR se lleva a cabo a través de un cambio en símbolos desde AND-invertida a OR- invertida en niveles alternos de compuertas. El primer nivel que va a cambiarse a un símbolo OR-invertido debe ser el último nivel. Estos cambios producen pares de círculos a lo

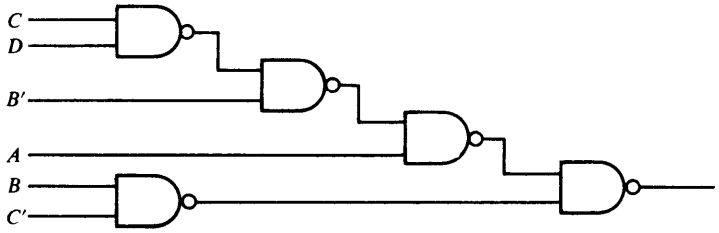


(a) AND-inversora (b) OR-inversora

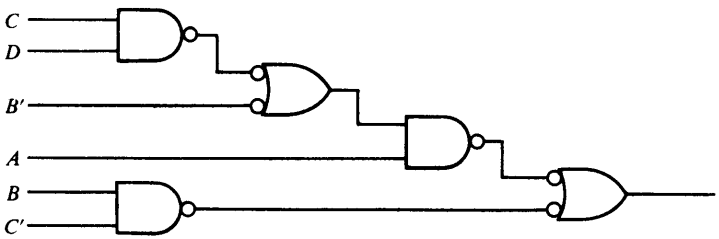
Figura 4-15 Dos símbolos para compuerta NAND.

largo de la misma línea, y esos pueden eliminarse ya que representan complementación doble. Además, una compuerta de una entrada AND u OR puede eliminarse ya que no realiza una función lógica. Una compuerta AND u OR de una entrada con un círculo en la entrada o salida se cambia a un circuito inversor.

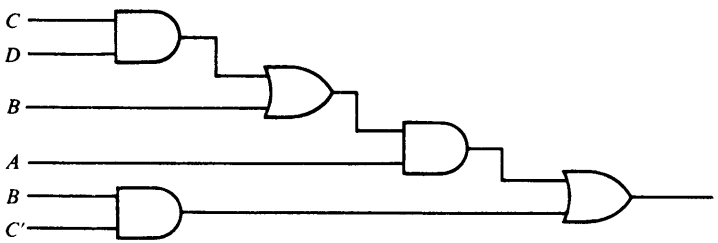
El procedimiento se demuestra en la Fig. 4-16. El diagrama lógico NAND en la Fig. 4-16(a) se convertirá en un diagrama AND-OR. El símbolo de la compuerta en el último nivel se cambia a un OR-invertido. Al buscar niveles alternos, se encuentra una compuerta más que requiere un cambio de símbolo como se muestra en la Fig. 4-16(b). Cualesquiera dos círculos a lo largo de la misma línea se eliminan. Los círculos que van a entradas externas también se eliminan, siempre que esté complementada la variable correspondiente de entrada. El diagrama lógico AND-OR requerido se muestra en la Fig. 4-16(c).



(a) Diagrama lógico NAND



(b) Sustitución con símbolos OR inversora en niveles alternos



(c) Diagrama lógico AND-OR

Figura 4-16 Conversión del diagrama lógico NAND en AND-OR.

4-8 CIRCUITOS NOR DE NIVELES MULTIPLES

La función NOR es la dual de la función NAND. Por esta razón, todos los procedimientos para la lógica NOR forman un dual de los procedimientos y reglas correspondientes desarrollados para la lógica NAND. En esta sección se enumeran diversos métodos para la implementación y análisis de la lógica NOR por el seguimiento de la misma lista de tópicos usados para la lógica NAND. Sin embargo, se incluyen explicaciones menos detalladas para evitar la repetición excesiva del material en la Sección 4-7.

Compuerta universal

La compuerta NOR es universal debido a que cualquier función booleana puede implementarse con ella, incluyendo un circuito flip-flop como se muestra en la Sección 6-2. La conversión de AND, OR y NOT en NOR se ilustra en la Fig. 4-17. La operación NOT se obtiene de una compuerta NOR de una entrada, todavía otro símbolo de un circuito inversor. La operación OR requiere dos compuertas NOR. La primera produce la OR-invertida y la segunda actúa como un inversor para obtener la salida normal. La operación AND se lleva a cabo a través de una compuerta NOR con inversores adicionales en cada entrada.

Implementación de una función booleana
Método de diagrama de bloques

El procedimiento de diagrama de bloques para implementar funciones booleanas con compuertas NOR es similar al procedimiento delineado en la sección anterior para las compuertas NAND.

1. Se dibuja el diagrama lógico AND-OR a partir de la expresión algebraica dada. Se supone que están disponibles las entradas tanto normal como complementaria.

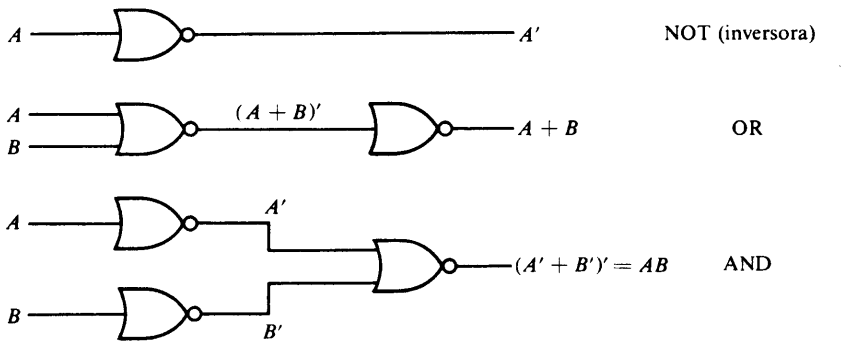
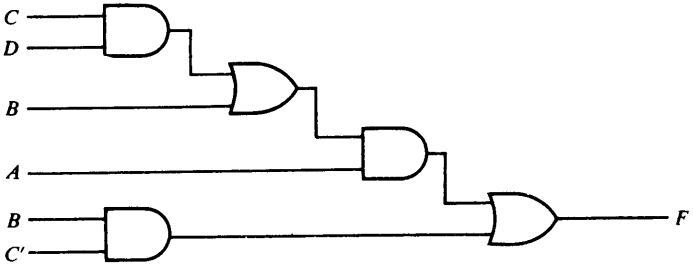
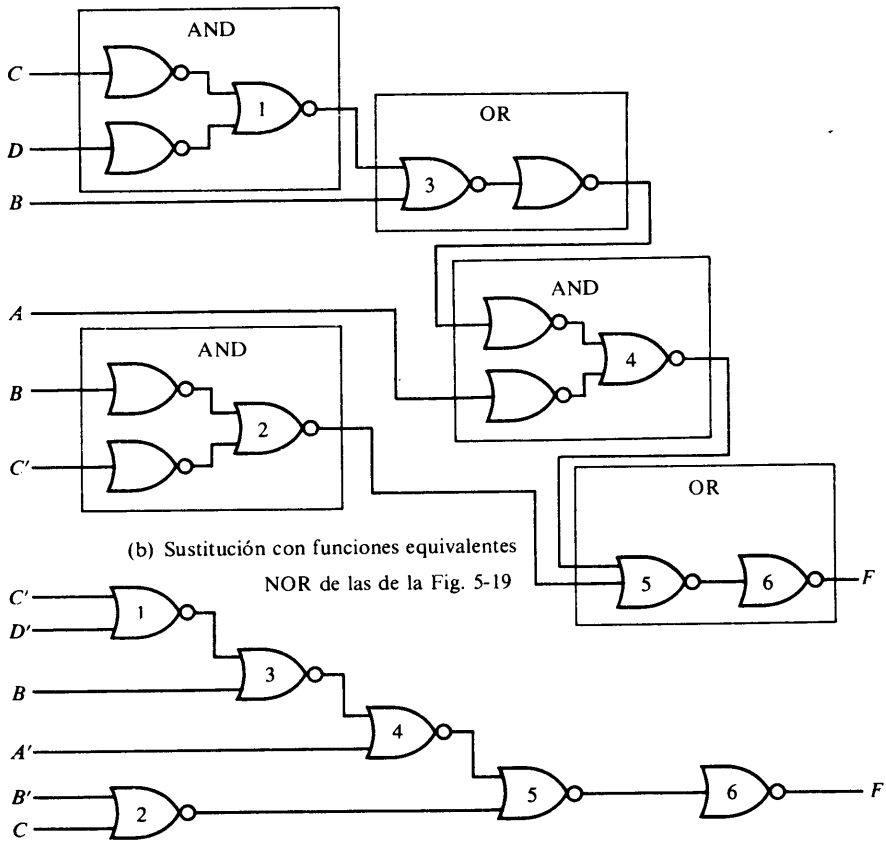


Figura 4-17 Implementación de compuertas NOT, OR y AND por compuertas NOR.



(a) Implementación AND/OR



(b) Sustitución con funciones equivalentes NOR de las de la Fig. 5-19

(c) Implementación NOR

Figura 4-18 Implementación de $F = A(B + CD) + BC'$ con compuertas NOR.

2. Se dibuja un segundo diagrama lógico con la lógica NOR equivalente, como se muestra en la Fig. 4-17, que sustituye a cada compuerta AND, OR y NOT.

3. Se eliminan del diagrama pares de inversores en cascada. Se eliminan inversores conectados a entradas externas únicas y se complementa la variable de entrada correspondiente.

El procedimiento se ilustra en la Fig. 4-18 para la función:

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

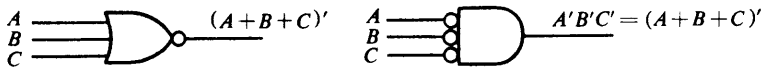
La implementación AND-OR de la función se muestra en el diagrama lógico en la Fig. 4-18(a). Para cada compuerta OR, se sustituye una compuerta NOR seguida por un inversor. Para cada compuerta AND, se sustituyen inversores de entrada seguidos por una compuerta NOR. El par de inversores en cascada de la casilla OR y de la casilla AND se eliminan. Los cuatro inversores conectados a entradas externas se eliminan y las variables de entrada se complementan. El resultado es el diagrama lógico NOR que se muestra en la Fig. 4-18(c). El número de compuertas NOR en este ejemplo es igual al número de compuertas AND-OR más un inversor adicional en la salida (compuerta NOR 6). En general, el número requerido de compuertas NOR para implementar una función booleana es igual al número de compuertas AND-OR, excepto por un inversor ocasional. Esto es cierto siempre que estén disponibles las entradas tanto normal como complementaria, debido a que la conversión obliga a que ciertas variables de entrada estén complementadas.

Procedimiento de análisis

El análisis de los diagramas lógicos NOR sigue los mismos procedimientos que se presentaron en la Sección 4-6 para el análisis de circuitos combinacionales. Para derivar la función booleana de un diagrama lógico, se marcan las salidas de las diversas compuertas con símbolos arbitrarios. Por sustituciones repetitivas, se obtiene la variable de salida como función de las variables de entrada. Para obtener la tabla de verdad de un diagrama lógico sin derivar primero la función booleana, se forma una tabla donde se listan las n variables de entrada con 2^n renglones de 1 y 0. Se deriva la tabla de verdad de las diversas salidas de compuerta NOR en sucesión, hasta que se obtiene la salida en la tabla de verdad. La función de salida de una compuerta NOR típica es de la forma $T + (A + B' + C)'$; de modo que la tabla de verdad para T está marcada con un 0 para las combinaciones donde $A = 1$ o $B = 0$ o $C = 1$. El resto de los renglones se llena con números 1.

Transformación del diagrama de bloques

Para convertir un diagrama lógico NOR en su diagrama lógico AND-OR equivalente, se usan los dos símbolos para las compuertas NOR que se muestran en la Fig. 4-19. El OR-invertido es el símbolo normal para una compuerta NOR y el AND-invertido es una alternativa conveniente en la que se utiliza el teorema de De Morgan y la convención de los círculos pequeños en las entradas denota complementación.



(a) OR-inversora (b) AND-inversora

Figura 4-19 Dos símbolos para compuerta NOR.

La conversión de un diagrama lógico NOR en un diagrama AND-OR se lleva a cabo a través de un cambio de símbolos de OR-invertido en AND-invertido principiando desde el último nivel y de niveles alternos. Los pares de círculos pequeños a lo largo de la misma línea se eliminan. Se elimina una compuerta de una entrada AND u OR, pero si tiene un pequeño círculo en la entrada o salida, se convierte en una inversora.

Este procedimiento se demuestra en la Fig. 4-20, donde el diagrama lógico NOR en (a) se convierte en un diagrama AND-OR. El símbolo de la compuerta en el último nivel (5) se cambia a un AND-invertido. Al buscar niveles alternos, se encuentra una compuerta en el nivel 3 y dos en el nivel 1. Estas tres compuertas pasan por un cambio de símbolo como se muestra en (b). Cualesquiera dos círculos a lo largo de la misma línea se eliminan. Los círculos que van a entradas externas también se eliminan, siempre que la variable de entrada correspondiente se complemente. La compuerta en el nivel 5 se vuelve una compuerta AND de una entrada y se elimina. El diagrama lógico AND-OR requerido se muestra en la Fig. 4-20(c).

4-9 OR-EXCLUYENTE Y FUNCIONES DE EQUIVALENCIA

Las operaciones binarias OR-excluyente y de equivalencia, denotadas por 0 y 0, respectivamente, realizan las siguientes funciones booleanas:

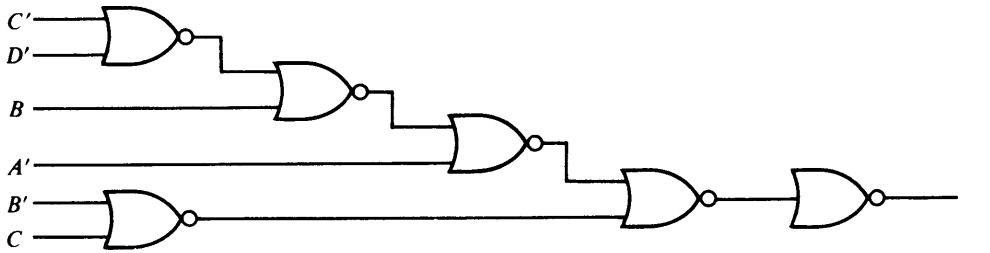
$$x \oplus y = xy' + x'y$$
$$x \odot y = xy + x'y'$$

Las dos operaciones son los complementos una de otra. Cada una es conmutativa y asociativa. Debido a estas dos propiedades, una función de tres o más variables puede expresarse sin paréntesis como se indica:

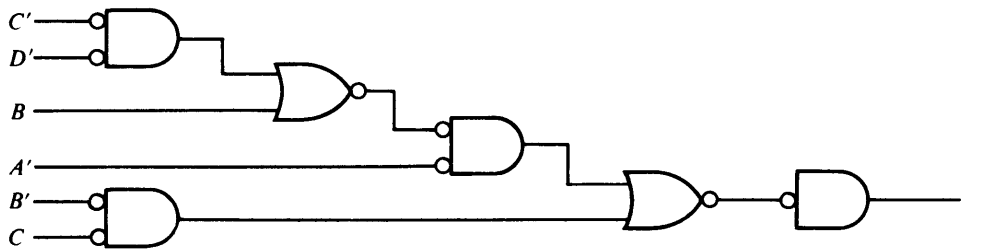
$$(A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C$$

Esto puede implicar la posibilidad de utilizar compuertas OR-excluyente (o equivalencia) con tres o más salidas. Sin embargo, las compuertas OR-excluyente de entradas múltiples son antieconómicas desde un punto de vista de hardware. De hecho, incluso una función de dos entradas por lo común se construye con otros tipos de compuertas. Por ejemplo, en la Fig. 4-21(a) se muestra la implementación de una función OR-excluyente de dos entradas con compuertas AND, OR y NOT. En la Fig. 4-21 (b) se muestra con compuertas NAND.

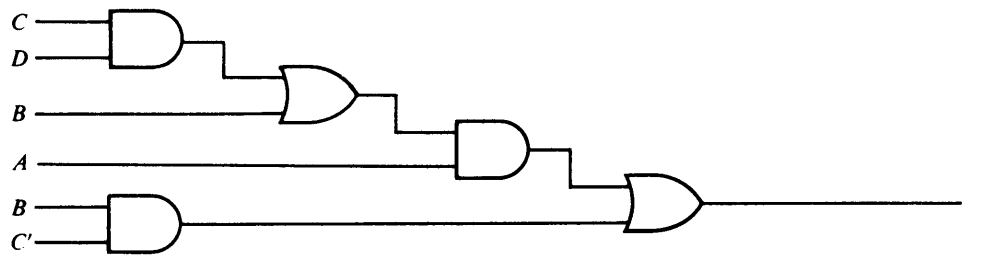
Sólo un número limitado de funciones booleanas puede expresarse exclusivamente en términos de operaciones OR-excluyente o de equivalencia. No obstante, estas funciones surgen con bastante frecuencia durante el diseño de sistemas digitales. Las



(a) Diagrama lógico NOR



(b) Sustitución con símbolos AND inversora en niveles alternos



(c) Diagrama lógico AND-OR

Figura 4-20 Conversión del diagrama lógico NOR en AND-OR.

dos funciones son de utilidad particular en operaciones aritméticas y en detección y corrección de errores.

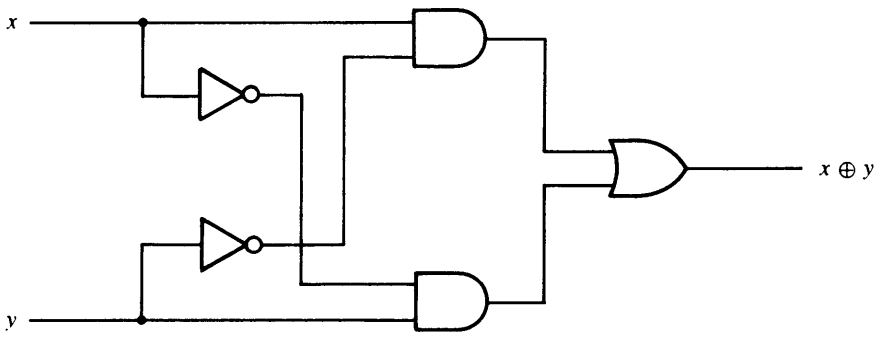
Una expresión OR-excluyente de n variables es igual a la función booleana con $2^n/2$ mintérminos cuyos números binarios equivalentes tienen un número impar de 1. Esto se demuestra en el mapa de la Fig. 4-22(a) para el caso de cuatro variables. Hay 16 mintérminos para cuatro variables. La mitad de los mintérminos tiene un valor numérico con número impar de 1; la otra mitad tiene un valor numérico con un número par de 1. El valor numérico de un mintérmino está determinado por los números de renglón y columna del cuadro que representa al mintérmino. El mapa de la Fig. 4-22(a) tiene 1 en los cuadros cuyos números de mintérmino tienen un número impar de 1. La función puede expresarse en términos de operaciones OR excluyentes de las cuatro variables. Esto se justifica por la siguiente manipulación algebraica:

$$A \oplus B \oplus C \oplus D = (AB' + A'B) \oplus (CD' + C'D)$$
$$= (AB' + A'B)(CD + C'D') + (AB + A'B')(CD' + C'D)$$
$$= \Sigma(1, 2, 4, 7, 8, 11, 13, 14)$$

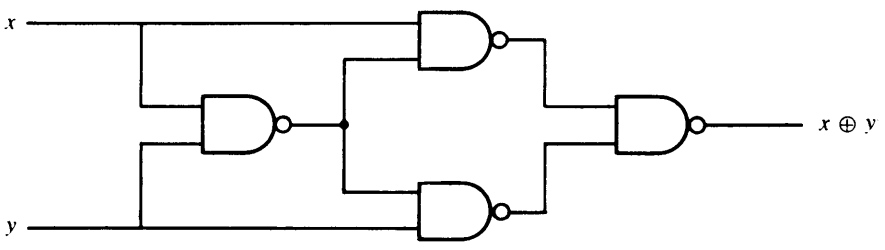
Una expresión de n variables de equivalencia es igual a la función booleana con $2^n/2$ mintérminos, cuyos números binarios equivalentes tienen un número par de 0. Esto se demuestra en el mapa en la Fig. 4-22(b) para el caso de cuatro variables. Los cuadros con 1 representan los ocho mintérminos con número par de 0, y la función puede expresarse en términos de las operaciones de equivalencia en las cuatro variables.

Cuando el número de variables en una función es impar, los mintérminos con un número par de 0 son los mismos que los mintérminos con un número impar de 1. Esto se demuestra en el mapa de tres variables de la Fig. 4-23(a). En consecuencia, una expresión OR-excluyente es igual a una expresión de equivalencia cuando ambas tienen el mismo número impar de variables. Sin embargo, forman los complementos una de otra cuando el número de variables es par, como se muestra en los dos mapas en la Fig. 4-22(a) y (b).

Cuando los mintérminos de una función con un número impar de variables tienen un número par de 1 (o en forma equivalente, un número impar de 0), la función



(a) con compuertas AND-OR-NOT



(b) con compuertas NAND

Figura 4-21 Implementación OR-excluyente.

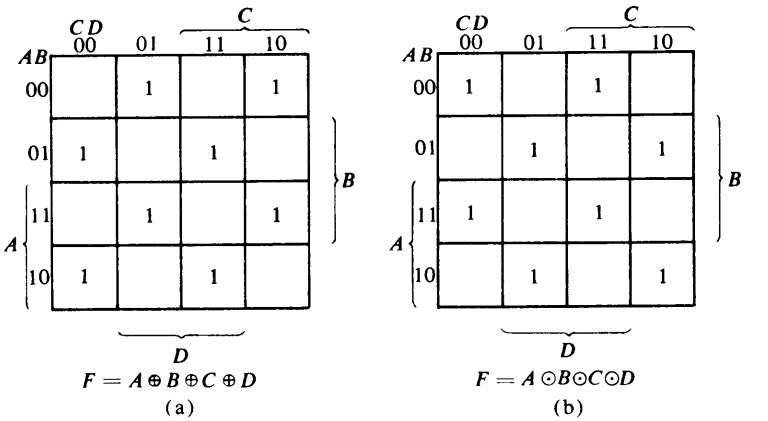


Figura 4-22 Mapa para (a) una función OR-excluyente y (b) una función equivalente, ambas de cuatro variables.

puede expresarse como el complemento ya sea de una expresión OR-excluyente o una expresión equivalente. Por ejemplo, la función de tres variables que se muestra en el mapa de la Fig. 4-23(b) puede expresarse de la siguiente forma:

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

o

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

La salida S de un sumador completo y la salida D de un restador completo (Sección 4-3) puede implementarse con funciones OR-excluyente debido a que cada función consta de cuatro mintérminos con valores numéricos que tienen un número impar de 1. La función OR-excluyente se usa en forma extensa en la implementación de operaciones aritméticas digitales, debido a que estas últimas por lo común se implantan a través de procedimientos que requieren una operación repetitiva de adición o sustracción.

Las funciones OR-excluyente y de equivalencia son muy útiles en sistemas que requieren códigos de detección de errores y corrección de errores. Como se expuso en

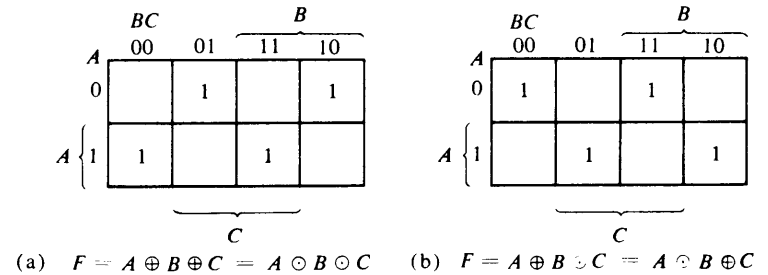


Figura 4-23 Mapa para funciones de tres variables.

la Sección 1-6, un bit de paridad es un esquema para detectar errores durante la transmisión de información binaria. Un bit de paridad es un bit adicional incluido con un mensaje binario para hacer que el número de los 1 sea impar o bien par. El mensaje, que incluye el bit de paridad, se transmite y entonces se verifica en la terminal receptora para buscar errores. Un error se detecta si la paridad verificada no corresponde con la transmitida. El circuito que genera el bit de paridad en el transmisor se conoce como *generador de paridad.* El circuito que verifica la pridad en el receptor se denomina *verificador de paridad.*

Como ejemplo, considérese un mensaje de tres bits que se transmite con un bit de impar-paridad. En la Tabla 4-4 se muestra la tabla de verdad para el generador de paridad. Los tres bits x, y y z constituyen el mensaje y son las entradas al circuito. El bit de paridad P es la salida. Para paridad impar, el bit P se genera de modo que el número total de 1 sea impar (incluyendo P). Mediante la tabla de verdad, se ve que $P = 1$, cuando el número de 1 en x, y y z es par. Esto corresponde al mapa de la Fig. 4-23(b); de modo que la función para P puede expresarse como sigue:

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

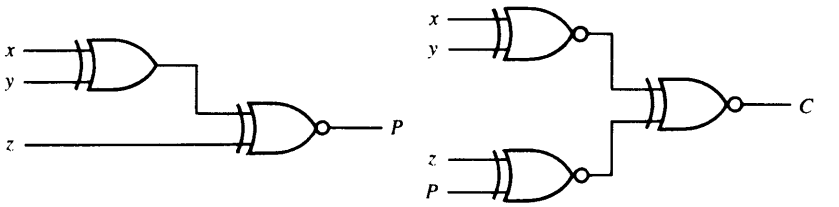
El diagrama lógico para el generador de paridad se muestra en la Fig. 4-24(a). Consta de una compuerta OR-excluyente de dos entradas y una compuerta de equivalencia de dos entradas. Las dos compuertas pueden intercambiarse y producir todavía la misma función, ya que P también es igual a:

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

El mensaje de tres bits y el bit de paridad se transmiten a su destino, donde se aplican a un circuito verificador de paridad. Un error ocurre durante la transmisión si la paridad de los cuatro bits recibidos es par, ya que la información binaria transmitida fue originalmente impar. La salida C del verificador de paridad debe ser un 1 cuando ocurre un error, esto es, cuando el número de 1 en las cuatro entradas es par. La tabla 4-5 es la tabla de verdad para el circuito verificador de impar-paridad. Mediante el cual

TABLA 4-4 Generación de paridad-impar

Mensaje de 3 bit			Bit de paridad generado
x	y	z	P
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0



(a) Generador paridad impar de 3 bit (b) Verificador paridad impar de 4 bit

Figura 4-24 Diagramas lógicos para generación y verificación de paridad.

se ve que la función para C consta de ocho mintérminos con valores numéricos que tienen un número par de 0. Esto corresponde al mapa de la Fig. 4-22(b); de modo que la función puede expresarse con operadores de equivalencia como sigue:

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

El diagrama lógico para el verificador de paridad se muestra en la Fig. 4-24(b) y consta de tres compuertas de equivalencia de dos entradas.

Es de interés observar que el generador de paridad puede implementarse con el circuito de la Fig. 4-24(b), si la entrada P se mantiene en forma permanente a lógica 0 y la salida se marca como P, en donde la ventaja es que el mismo circuito puede usarse para generación de paridad al igual que para verificación de paridad.

Del ejemplo anterior es obvio, que los circuitos de generación de paridad y verificación de paridad siempre tienen una función de salida, que incluye la mitad de

TABLA 4-5 Verificación de paridad-impar

Cuatro bits recibidos				Verificación de error de paridad
x	y	z	P	
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

los mintérminos cuyos valores numéricos tienen ya sea un número par o bien impar de 1. Como consecuencia, deben implementarse con compuertas de equivalencia y/o OR excluyente.

BIBLIOGRAFIA

1. Rhyne, V. T., *Fundamentals of Digital Systems Design.* Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1973.
2. Peatman, J. P., *The Design of Digital Systems.* New York: McGraw-Hill Book Co., 1972.
3. Nagle, H. T. Jr., B. D. Carrol, and J. D. Irwin, *An Introduction to Computer Logic.* Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1975.
4. Hill, F. J., y G. R. Peterson, *Introduction to Switching Theory and Logical Design,* 3a. ed. New York: John Wiley & Sons, Inc., 1981.
5. Maley, G. A., y J. Earle, *The Logic Design of Transistor Digital Computers.* Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1963.
6. Friedman, A. D., y P. R. Menon, *Theory and Design of Switching Circuits.* Woodland Hills, Calif.: Computer Science Press, Inc., 1975.

PROBLEMAS

4-1. Un circuito combinacional tiene cuatro entradas y una salida. La salida es igual a 1 cuando (1) todas las entradas son iguales a 1 o (2) ninguna de las entradas es igual a 1 o (3) un número impar de entradas son iguales a 1.
(a) Obtenga la tabla de verdad.
(b) Encuentre la función simplificada de salida en suma de productos.
(c) Encuentre la función simplificada de salida en producto de sumas.
(d) Dibuje los dos diagramas lógicos.

4-2. Diseñe un circuito combinacional que acepte un número de tres bits y genere una salida de número binario igual al cuadrado del número de entrada.

4-3. Es necesario multiplicar dos números binarios, cada uno de dos bits de largo, con objeto de formar su producto en binario. Permita que los dos números se representen por a_1, a_0 y b_1, b_0, donde el subíndice 0 denota el bit menos significativo.
(a) Determine el número de líneas de salida requerido.
(b) Encuentre las expresiones booleanas simplificadas para cada salida.

4-4. Repita el problema 4-3 para formar la suma (en lugar del producto) de los dos números binarios.

4-5. Diseñe un circuito combinacional con cuatro líneas de entrada que representen un dígito decimal en BCD y cuatro líneas de salida que generen el complemento a 9 del dígito de entrada.

4-6. Diseñe un circuito combinacional cuya entrada es un número de cuatro bits y cuya salida es el complemento a 2 del número de entrada.

4-7. Diseñe un circuito combinacional que multiplique por 5 un dígito decimal de entrada representado en BCD. La salida también está en BCD. Muestre que las salidas pueden obtenerse por las líneas de entrada sin utilizar ninguna compuerta lógica.

4-8. Diseñe un circuito combinacional que detecte un error en la representación de un dígito decimal en BCD. En otras palabras, obtenga un diagrama lógico cuya salida sea lógica 1 cuando las salidas contienen una combinación no usada en el código.

4-9. Implementar un restador completo con dos medios restadores y una compuerta OR.

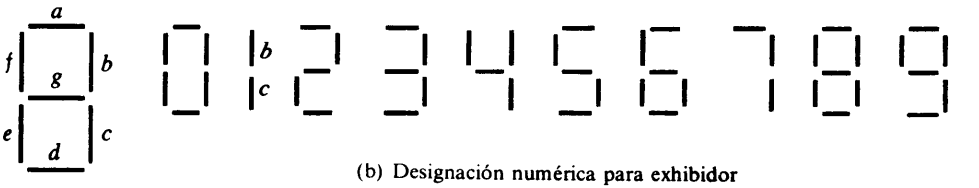
4-10. Muestre cómo puede convertirse un sumador completo en un restador completo con adición de un circuito inversor.

4-11. Diseñe un circuito combinacional que convierta un dígito decimal del código 8, 4, —2, —1 en el código BCD.

4-12. Diseñe un circuito combinacional que convierta un dígito decimal del código 2, 4, 2, 1 en el código 8, 4, —2, —1.

4-13. Obtenga el diagrama lógico que convierte un número binario de cuatro dígitos en un número decimal en BCD. Observe que son necesarios dos dígitos decimales ya que los números binarios están comprendidos entre 0 y 15.

4-14. Un decodificador de BCD-a-siete-segmentos es un circuito combinacional que acepta un dígito decimal en BCD y genera las salidas apropiadas para la selección de segmentos en



(a) Designación de segmentos

(b) Designación numérica para exhibidor

Figura P4-14.

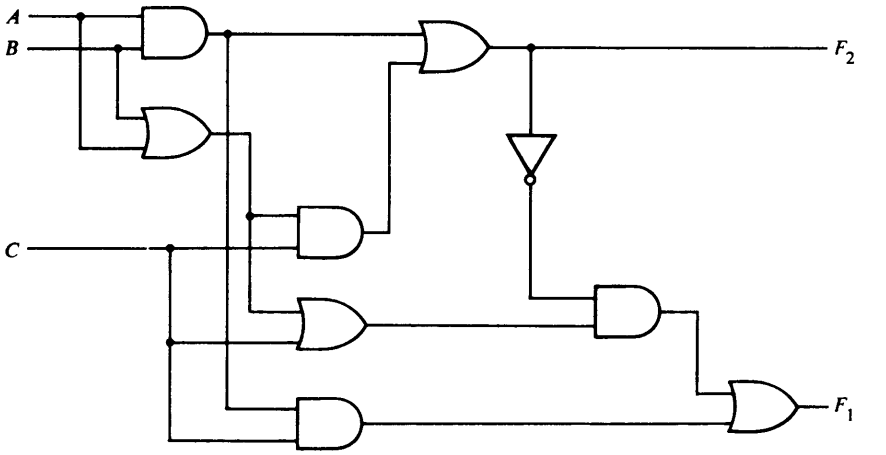


Figura P4-15.

un exhibidor indicador usado para mostrar el dígito decimal. Las siete salidas del decodificador *(a, b, c, d, e, f, g)* seleccionan los segmentos correspondientes al exhibidor como se muestra en la Fig. P4-14(a). La designación numérica escogida para representar el dígito decimal se muestra en la Fig. P4-14(b). Diseñe el circuito decodificador de BCD-a-siete-segmentos.

4-15. Analice los circuitos combinacionales de dos salidas que se muestran en la Fig. P4-15. Obtenga las funciones booleanas para las dos salidas y explique la operación del circuito.

4-16. Derive la tabla de verdad del circuito que se muestra en la Fig. P4-15.

4-17. Utilice el método de diagrama de bloques para convertir el diagrama lógico de la Fig. 4-8 en una implementación NAND.

4-18. Repita el problema 4-17 para la implementación NOR.

4-19. Obtenga el diagrama lógico NAND de un adicionador completo mediante las funciones booleanas:

$$C = xy + xz + yz$$

$$S = C'(x + y + z) + xyz$$

4-20. Determine la función booleana para la salida F del circuito que se muestra en la Fig. P4-20. Obtenga un circuito equivalente con menos compuertas NOR.

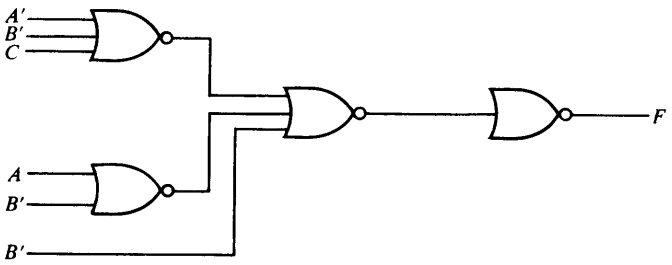


Figura P4-20.

4-21. Determine las funciones booleanas de salida de los circuitos en la Fig. P4-21.

4-22. Obtenga la tabla de verdad para los circuitos en la Fig. P4-21.

4-23. Obtenga el diagrama lógico AND-OR equivalente de la Fig. P4-21(a).

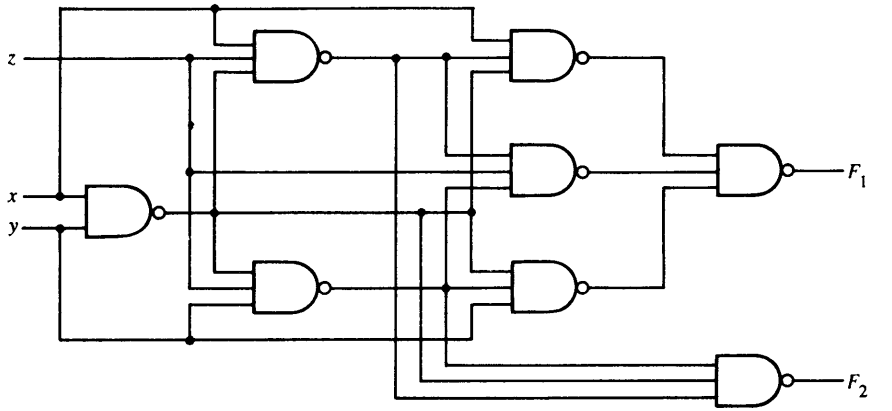
4-24. Obtenga el diagrama lógico AND-OR equivalente en la Fig. P4-21(b).

4-25. Obtenga el diagrama lógico de una función de equivalencia de dos entradas utilizando (a) compuertas AND, OR y NOT; (b) compuertas NOR; y (c) compuertas NAND.

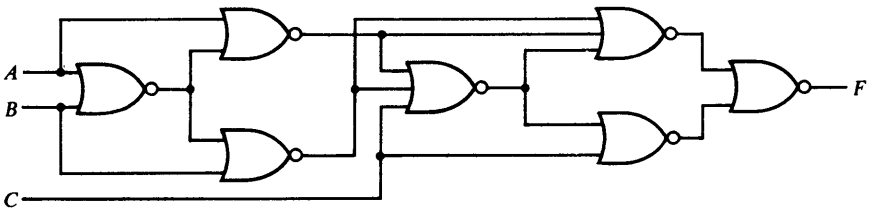
4-26. Muestre que el circuito en la Fig. 4-2(b) es un OR-excluyente.

4-27. Muestre que $A \odot B \odot C \odot D = \Sigma\,(0, 3, 5, 6, 9, 10, 12, 15)$.

4-28. Diseñe un circuito combinacional que convierta un número de cuatro bits en código reflejado (Tabla 1-4) en un número binario de cuatro bits. Implemente el circuito con compuertas OR-excluyentes.



(a)



(b)

Figura P4-21.

4-29. Diseñe un circuito combinacional para verificar paridad par de cuatro bits. Se requiere una salida de lógica 1 cuando los cuatro bits no constituyen una paridad par.

4-30. Implemente las cuatro funciones booleanas listadas usando tres circuitos medio adicionadores (Fig. 4-2e).

$D = A \oplus B \oplus C$

$E = A'BC + AB'C$

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

$G = ABC$

4-31. Implemente la función booleana:

$$F = AB'CD' + A'BCD' + AB'C'D + A'BC'D$$

con compuertas OR-excluyente y AND.