DÉCIMA EDICIÓN

Sistemas digitales

Principios y aplicaciones

Ronald J. Tocci
Monroe Community College

Neal S. Widmer
Purdue University

Gregory L. Moss
Purdue University

Traducción

Alfonso Vidal Romero Elizondo
Ingeniero en Electrónica y Comunicación
Instituto Tecnológico y de Estudios Superiores de Monterrey - Campus Monterrey

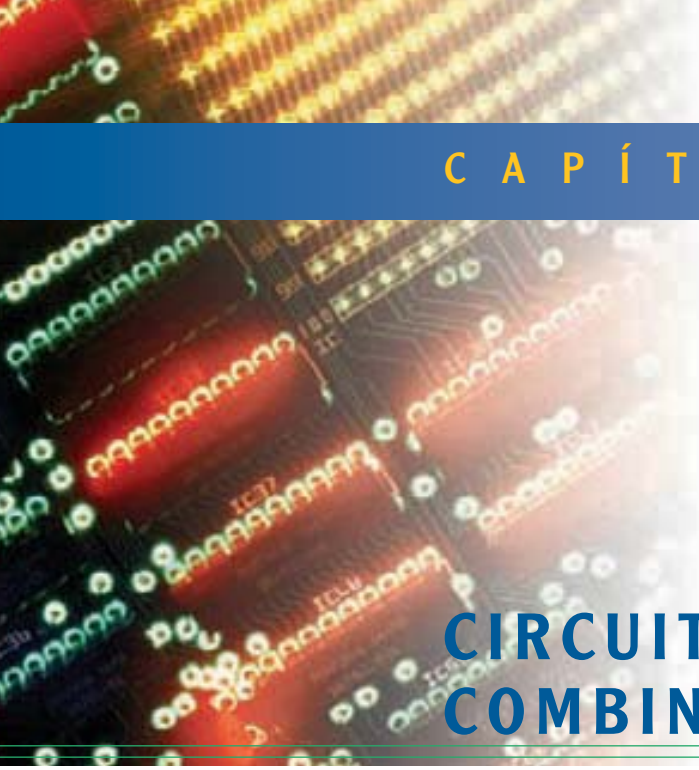
Revisión técnica

Reynaldo Félix Acuña
Profesor investigador
Departamento de Ingeniería Eléctrica y Electrónica
Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Estado de México

Marcos de Alba
Profesor investigador
Departamento de Ingeniería Eléctrica y Electrónica
Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Estado de México

PEARSON Educación ®

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



CAPÍTULO 4

CIRCUITOS LÓGICOS COMBINACIONALES

CONTENIDO

OBJETIVOS

Al terminar este capítulo, usted podrá:

- Convertir una expresión lógica en una expresión de suma de productos.
- Realizar los pasos necesarios para reducir una expresión de suma de productos a su forma más simple.
- Utilizar el álgebra booleana y el mapa de Karnaugh como herramientas para simplificar y diseñar circuitos lógicos.
- Explicar la operación de los circuitos OR exclusivo y NOR exclusivo.
- Diseñar circuitos lógicos simples sin la ayuda de una tabla de verdad.
- Implementar circuitos de habilitación.
- Citar las características básicas de los CIs digitales TTL y CMOS.
- Utilizar las reglas básicas de diagnóstico de fallas de los sistemas digitales.
- Deducir las fallas a partir de los resultados observados en circuitos lógicos combinacionales.
- Describir la idea fundamental de los dispositivos lógicos programables (PLDs).
- Describir los pasos implicados en la programación de un PLD para realizar una función lógica combinacional simple.
- Consultar los manuales de usuario de Altera para adquirir la información necesaria para realizar un experimento de programación simple en el laboratorio.
- Describir los métodos de diseño jerárquico.
- Identificar los tipos de datos apropiados para las variables con valores de un solo bit, arreglos de bits y numéricas.
- Describir los circuitos lógicos mediante el uso de las estructuras de control de HDL IF/ELSE, IF/ELSIF y CASE.
- Seleccionar la estructura de control apropiada para un problema dado.

INTRODUCCIÓN

En el capítulo 3 estudiamos la operación de todas las compuertas lógicas básicas, y utilizamos el álgebra booleana para describir y analizar circuitos formados de combinaciones de compuertas lógicas. Estos circuitos pueden clasificarse como circuitos lógicos *combinacionales* ya que, en cualquier momento, el nivel lógico de la salida depende de la combinación de los niveles lógicos presentes en las entradas. Un circuito combinacional no tiene característica de *memoria*, por lo que su salida depende *sólo* del valor actual de sus entradas.

En este capítulo continuaremos con nuestro estudio de los circuitos combinacionales. Para empezar, veremos más detalles sobre la simplificación de los circuitos lógicos. Analizaremos dos métodos: el primero utiliza los teoremas del álgebra booleana; el segundo utiliza una técnica de *mapeo*. Además,

estudiaremos técnicas simples de diseño de circuitos lógicos combinacionales para satisfacer un conjunto dado de requerimientos. El estudio completo del diseño de circuitos lógicos no es uno de nuestros objetivos, pero los métodos que presentaremos le brindarán una excelente introducción al diseño lógico.

Una buena porción de este capítulo está dedicada al diagnóstico de fallas de los circuitos combinacionales. Con esta primera exposición al diagnóstico de fallas, usted podrá empezar a desarrollar el tipo de habilidades analíticas necesarias para tener éxito al realizar sus diagnósticos de fallas. Para que este material sea lo más práctico posible, primero presentaremos algunas de las características básicas de los CIs de compuertas lógicas en las familias lógicas TTL y CMOS, junto con una descripción de los tipos más comunes de fallas que se encuentran en los circuitos integrados digitales.

En las últimas secciones de este capítulo ampliaremos nuestro conocimiento sobre los dispositivos lógicos programables y los lenguajes de descripción de hardware. Reforzaremos el concepto de las conexiones de hardware programables y brindaremos más detalles en relación con la función del sistema de desarrollo. Usted conocerá los pasos que se siguen actualmente para el diseño y desarrollo de los sistemas digitales. Le proporcionaremos suficiente información para que pueda elegir los tipos de datos correctos para usarlos en proyectos simples que presentaremos más adelante en este capítulo. Por último explicaremos varias estructuras de control, junto con algunas instrucciones relacionadas con su uso apropiado.

4-1 FORMA DE SUMA DE PRODUCTOS

Los métodos de simplificación y diseño de circuitos lógicos que estudiaremos requieren que la expresión lógica se encuentre en forma de **suma de productos (SOP)**. Algunos ejemplos de esta forma son:

1. $ABC + \overline{A}B\overline{C}$
2. $AB + \overline{A}B\overline{C} + \overline{C}\,\overline{D} + D$
3. $\overline{A}B + C\overline{D} + EF + GK + H\overline{L}$

Cada una de estas expresiones de suma de productos consiste de dos o más términos AND (productos) a los que se les aplica la operación OR. Cada término AND consiste de una o más variables que aparecen de manera *individual*, ya sea en forma complementada o no complementada. Por ejemplo, en la expresión de suma de productos $ABC + \overline{A}B\overline{C}$ el primer producto AND contiene las variables A, B y C en su forma no complementada (no invertida). El segundo término AND contiene a A y C en su forma complementada (invertida). En una expresión de suma de productos, un signo de inversión *no puede* cubrir más de una variable en un término (por ejemplo, no podemos tener $\overline{ABC}$ o $\overline{RST}$).

Producto de las sumas

Algunas veces se utiliza otra forma general para las expresiones lógicas en el diseño de circuitos lógicos. A esta forma se le llama **producto de las sumas (POS)** y consiste de dos o más términos OR (sumas) a los que se les aplica una operación AND. Cada término OR consiste de una o más variables en su forma complementada o no complementada. He aquí algunas expresiones de producto de sumas:

1. $(A + \overline{B} + C)(A + C)$
2. $(A + \overline{B})(\overline{C} + D)F$
3. $(A + C)(B + \overline{D})(\overline{B} + C)(A + \overline{D} + \overline{E})$

Los métodos de simplificación y diseño de circuitos que utilizaremos se basan en la forma de suma de productos (SOP), por lo que no veremos muchos ejemplos

con la forma de producto de sumas (POS). No obstante, de vez en cuando se presentará esta forma para algunos circuitos lógicos que tengan una cierta estructura.

PREGUNTAS DE REPASO

1. ¿Cuál de las siguientes expresiones se encuentra en la forma SOP?
   (a) $AB + CD + E$
   (b) $AB(C + D)$
   (c) $(A + B)(C + D + F)$
   (d) $\overline{MN} + PQ$
2. Repita la pregunta 1 para la forma POS.

4-2 SIMPLIFICACIÓN DE CIRCUITOS LÓGICOS

Una vez que se obtiene la expresión para un circuito lógico, podemos reducirla a una forma más simple que contenga menos términos, o menos variables en uno o más términos. Así, la nueva expresión puede utilizarse para implementar un circuito equivalente al circuito original, pero que contenga menos compuertas y conexiones.

Para ilustrar esto, el circuito de la figura 4-1(a) puede simplificarse para producir el circuito de la figura 4-1(b). Ambos circuitos realizan la misma lógica, por lo que debe ser obvio que el más simple es más conveniente, ya que contiene menos compuertas y, por lo tanto, será más pequeño y económico que el original. Lo que es más, la confiabilidad del circuito aumentará, ya que hay menos interconexiones que pueden provocar fallas potenciales en el circuito.

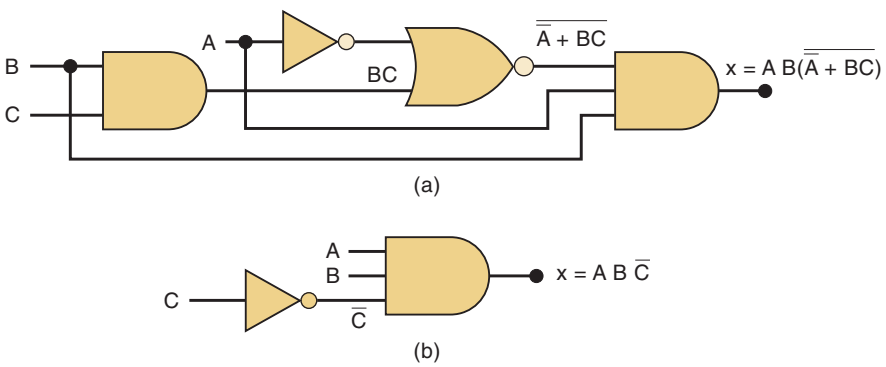


FIGURA 4-1 A menudo es posible simplificar un circuito lógico de tal forma que en la parte (a) se produzca una implementación más eficiente, la cual se muestra en (b).

En las siguientes secciones estudiaremos dos métodos para simplificar los circuitos lógicos. Uno de ellos utiliza los teoremas de álgebra booleana y, como veremos, depende mucho de la inspiración y la experiencia. El otro método (mapeo de Karnaugh) tiene un enfoque sistemático, paso a paso. Tal vez algunos instructores deseen omitir este último método debido a que es algo mecánico y es posible que no contribuya a una mejor comprensión del álgebra booleana. Esto puede hacerse sin afectar la continuidad o la claridad del resto del libro.

4-3 SIMPLIFICACIÓN ALGEBRAICA

Podemos utilizar los teoremas de álgebra booleana que estudiamos en el capítulo 3 para que nos ayuden a simplificar la expresión para un circuito lógico. Desafortunadamente, no siempre es obvio cuáles teoremas deben aplicarse para producir

el resultado más simple. Lo que es más, no hay una manera sencilla de saber si la expresión simplificada se encuentra en su forma más simple o si todavía puede simplificarse más. Por ende, la simplificación algebraica se vuelve a menudo un proceso de prueba y error. Sin embargo, con experiencia uno puede volverse un adepto para obtener resultados bastante razonables.

Los ejemplos que se muestran a continuación ilustrarán muchas de las maneras en las que pueden aplicarse los teoremas booleanos para tratar de simplificar una expresión. Estos ejemplos contienen dos pasos esenciales:

1. La expresión original se coloca en forma SOP mediante la aplicación repetida de los teoremas de DeMorgan y la multiplicación de los términos.
2. Una vez que la expresión original se encuentre en la forma SOP, se comprueba si hay factores comunes en los términos de productos y se lleva a cabo la factorización en donde sea posible. Este proceso de factorización deberá ayudar a eliminar uno o más términos.

EJEMPLO 4-1

Simplifique el circuito lógico que se muestra en la figura 4-2(a).

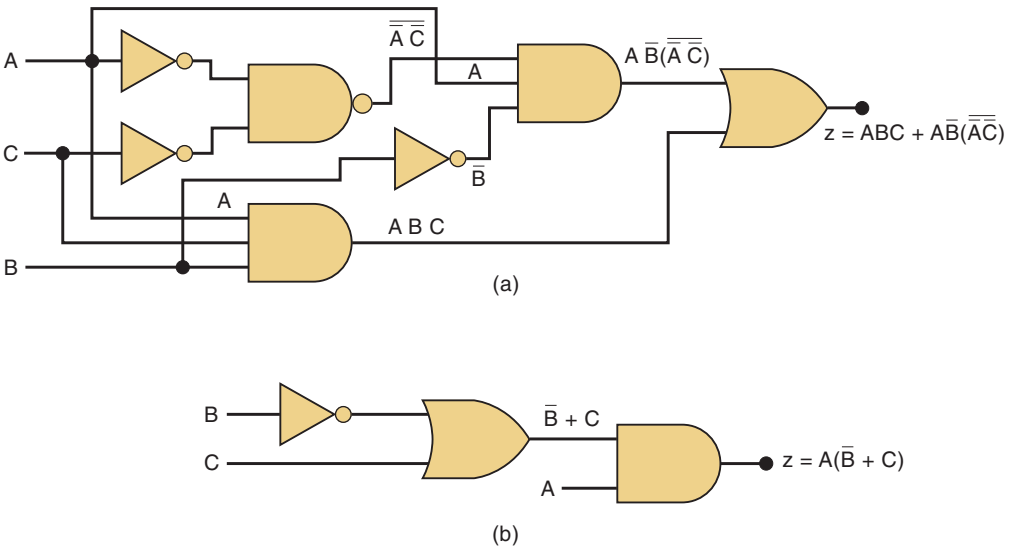


FIGURA 4-2 Ejemplo 4-1.

Solución

El primer paso es determinar la expresión para la salida, utilizando el método que se presentó en la sección 3-6. El resultado es

$$z = ABC + A\overline{B} \cdot (\overline{\overline{A}\,\overline{C}})$$

Una vez que se obtiene la expresión, por lo general, es conveniente descomponer todos los signos inversores grandes mediante el uso de los teoremas de DeMorgan, y después multiplicar todos los términos.

$$\begin{aligned} z &= ABC + A\overline{B}(\overline{\overline{A}} + \overline{\overline{C}}) && \text{[teorema (17)]} \\ &= ABC + A\overline{B}(A + C) && \text{[se cancelan las inversiones dobles]} \\ &= ABC + A\overline{B}A + A\overline{B}C && \text{[se realizan las multiplicaciones]} \\ &= ABC + A\overline{B} + A\overline{B}C && [A \cdot A = A] \end{aligned}$$

Ahora que la expresión se encuentra en la forma SOP, debemos buscar variables comunes entre los diversos términos con la intención de factorizar. Los términos primero y tercero de arriba tienen a AC en común, lo cual puede factorizarse:

$$z = AC(B + \overline{B}) + A\overline{B}$$

Como $B + \overline{B} = 1$, entonces

$$\begin{aligned} z &= AC(1) + A\overline{B} \\ &= AC + A\overline{B} \end{aligned}$$

Ahora podemos factorizar A, lo cual produce

$$z = A(C + \overline{B})$$

Este resultado ya no puede simplificarse más. En la figura 4-2(b) se muestra la implementación de su circuito. Es fácil observar que el circuito de la figura 4-2(b) es mucho más simple que el de la figura 4-2(a).

EJEMPLO 4-2

Simplifique la expresión $z = A\overline{B}\,\overline{C} + A\overline{B}C + ABC$.

Solución

La expresión ya se encuentra en la forma SOP.

Método 1: Los primeros dos términos de la expresión tienen el producto AB en común. Por lo tanto,

$$\begin{aligned} z &= A\overline{B}(\overline{C} + C) + ABC \\ &= A\overline{B}(1) + ABC \\ &= A\overline{B} + ABC \end{aligned}$$

Podemos factorizar la variable A de ambos términos:

$$z = A(\overline{B} + BC)$$

Utilizando el teorema (15b):

$$z = A(\overline{B} + C)$$

Método 2: La expresión original es $z = A\overline{B}\,\overline{C} + A\overline{B}C + ABC$. Los primeros dos términos tienen a $A\overline{B}$ en común. Los últimos dos tienen a AC en común. ¿Cómo podemos saber si debemos factorizar $A\overline{B}$ de los primeros dos términos, o AC de los últimos dos términos? En realidad podemos hacer ambas cosas si utilizamos el término $A\overline{B}C$ *dos veces*. En otras palabras, podemos reformular la expresión de la siguiente manera:

$$z = A\overline{B}\,\overline{C} + A\overline{B}C + A\overline{B}C + ABC$$

en donde hemos agregado un término $A\overline{B}C$. Esto es válido y no cambia el valor de la expresión, ya que $A\overline{B}C + A\overline{B}C = A\overline{B}C$ [(teorema (7)]. Ahora podemos factorizar $A\overline{B}$ de los primeros dos términos y AC de los últimos dos:

$$\begin{aligned} z &= A\overline{B}(C + \overline{C}) + AC(\overline{B} + B) \\ &= A\overline{B} \cdot 1 + AC \cdot 1 \\ &= A\overline{B} + AC = A(\overline{B} + C) \end{aligned}$$

Desde luego que este resultado es el mismo que se obtuvo con el método 1. Este truco de usar el mismo término dos veces puede usarse siempre. De hecho, el mismo término puede usarse más veces si es necesario.

EJEMPLO 4-3

Simplifique $z = \overline{A}C(\overline{\overline{A}BD}) + \overline{A}B\overline{C}\,\overline{D} + A\overline{B}C$.

Solución

Primero, utilizamos el teorema de DeMorgan en el primer término:

$$z = \overline{A}C(A + \overline{B} + \overline{D}) + \overline{A}B\overline{C}\,\overline{D} + A\overline{B}C \qquad \textbf{(paso 1)}$$

La multiplicación produce lo siguiente:

$$z = \overline{A}CA + \overline{A}C\overline{B} + \overline{A}C\overline{D} + \overline{A}B\overline{C}\,\overline{D} + A\overline{B}C \qquad \textbf{(2)}$$

Como $\overline{A} \cdot A = 0$, se elimina el primer término:

$$z = \overline{A}\,\overline{B}C + \overline{A}C\overline{D} + \overline{A}B\overline{C}\,\overline{D} + A\overline{B}C \qquad \textbf{(3)}$$

Ésta es la forma SOP deseada. Ahora debemos buscar factores comunes de entre los diversos términos de productos. La idea es buscar el factor común más grande entre dos o más términos de productos. Por ejemplo, los términos primero y último tienen el factor común $\overline{B}C$ y los términos segundo y tercero tienen el factor común $\overline{A}\,\overline{D}$. Podemos factorizar estos términos de la siguiente manera:

$$z = \overline{B}C(\overline{A} + A) + \overline{A}\,\overline{D}(C + B\overline{C}) \qquad \textbf{(4)}$$

Ahora, como $\overline{A} + A = 1$, y $C + B\overline{C} = C + B$ [teorema (15a)], tenemos que

$$z = \overline{B}C + \overline{A}\,\overline{D}(B + C) \qquad \textbf{(5)}$$

Este mismo resultado podría obtenerse si eligiéramos otros términos para la factorización. Por ejemplo, podríamos haber factorizado C de los términos de productos primero, segundo y cuarto, en el paso 3, para obtener:

$$z = C(\overline{A}\,\overline{B} + \overline{A}\,\overline{D} + A\overline{B}) + \overline{A}B\overline{C}\,\overline{D}$$

La expresión dentro de los paréntesis puede factorizarse aún más:

$$z = C(\overline{B}[\overline{A} + A] + \overline{A}\,\overline{D}) + \overline{A}B\overline{C}\,\overline{D}$$

Y como $\overline{A} + A = 1$, esta expresión se convierte en:

$$z = C(\overline{B} + \overline{A}\,\overline{D}) + \overline{A}B\overline{C}\,\overline{D}$$

Después de multiplicar, nos queda

$$z = \overline{B}C + \overline{A}C\,\overline{D} + \overline{A}B\overline{C}\,\overline{D}$$

Ahora podemos factorizar $\overline{A}\,\overline{D}$ de los términos segundo y tercero para obtener

$$z = \overline{B}C + \overline{A}\,\overline{D}(C + B\overline{C})$$

Si utilizamos el teorema (15a), la expresión entre paréntesis se convierte en $B + C$. Por lo tanto, queda

$$z = \overline{B}C + \overline{A}\,\overline{D}(B + C)$$

Este resultado es el mismo que obtuvimos antes, pero nos llevó muchos más pasos. Aquí se ilustra el porqué debemos buscar los factores comunes más grandes: por lo general, nos llevará a la expresión final en menos pasos.

El ejemplo 4-3 ilustra la frustración que se encuentra a menudo en la simplificación booleana. Como hemos llegado a la misma ecuación (que parece irreducible) por dos métodos distintos, podría parecer razonable concluir que esta ecuación final es la forma más simple. De hecho, la forma más simple de esta ecuación es

$$z = \overline{A}B\overline{D} + \overline{B}C$$

Pero no hay una manera aparente de reducir el paso (5) para llegar a esta versión más simple. En este caso nos faltó una operación en el proceso, la cual nos podría haber conducido a la forma más simple. La pregunta es, "¿Cómo hubiéramos podido saber que omitimos un paso?" Más adelante examinaremos una técnica de mapeo que siempre nos llevará a la forma SOP más simple.

EJEMPLO 4-4

Simplifique la expresión $x = (\overline{A} + B)(A + B + D)\overline{D}$.

Solución

La expresión puede colocarse en la forma de suma de productos mediante la multiplicación de todos los términos. El resultado es

$$x = \overline{A}A\overline{D} + \overline{A}B\overline{D} + \overline{A}D\overline{D} + BA\overline{D} + BB\overline{D} + BD\overline{D}$$

El primer término puede eliminarse, ya que $\overline{A}A = 0$. De igual forma, los términos tercero y sexto pueden eliminarse debido a que $D\overline{D} = 0$. El quinto término puede simplificarse a $B\overline{D}$, ya que $BB = B$. Esto nos da

$$x = \overline{A}B\overline{D} + AB\overline{D} + B\overline{D}$$

Podemos factorizar $B\overline{D}$ de cada término para obtener

$$x = B\overline{D}(\overline{A} + A + 1)$$

Es evidente que el término dentro de los paréntesis siempre es 1, por lo que al último nos queda

$$x = B\overline{D}$$

EJEMPLO 4-5

Simplifique el circuito de la figura 4-3(a).

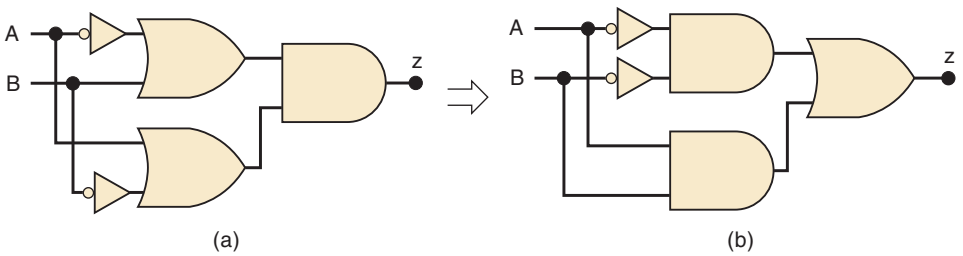


FIGURA 4-3 Ejemplo 4-5.

Solución

La expresión para la salida z es

$$z = (\overline{A} + B)(A + \overline{B})$$

Si multiplicamos para obtener la forma de suma de productos, obtenemos

$$z = \overline{A}A + \overline{A}\,\overline{B} + BA + B\overline{B}$$

Podemos eliminar $\overline{A}A = 0$ y $B\overline{B} = 0$ para terminar con

$$z = \overline{A}\,\overline{B} + AB$$

Esta expresión se implementa en la figura 4-3(b), y si la comparamos con el circuito original veremos que ambos circuitos contienen el mismo número de compuertas y conexiones. En este caso, el proceso de simplificación produjo un circuito equivalente pero más simple.

EJEMPLO 4-6

Simplifique $x = A\overline{B}C + \overline{A}BD + \overline{C}\,\overline{D}$.

Solución

Inténtelo, pero no podrá simplificar más esta expresión.

PREGUNTAS DE REPASO

1. Indique cuáles de las siguientes expresiones *no* se encuentran en la forma de suma de productos:
   (a) $RS\overline{T} + \overline{R}S\overline{T} + \overline{T}$
   (b) $A\overline{DC} + \overline{A}DC$
   (c) $MN\overline{P} + (M + \overline{N})P$
   (d) $AB + \overline{A}B\overline{C} + A\,\overline{B}\,\overline{C}\,D$
2. Simplifique el circuito de la figura 4-1(a) para obtener el circuito de la figura 4-1(b).
3. Cambie cada compuerta AND en la figura 4-1(a) por una compuerta NAND. Determine la nueva expresión para x y simplifíquela.

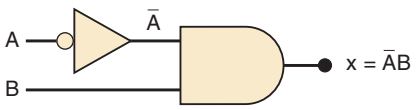
4-4 DISEÑO DE CIRCUITOS LÓGICOS COMBINACIONALES

Cuando se da el nivel de salida deseado de un circuito lógico para todas las posibles condiciones de entrada, los resultados pueden mostrarse de manera conveniente en una tabla de verdad. La expresión booleana para el circuito requerido puede entonces derivarse de la tabla de verdad. Por ejemplo, considere la figura 4-4(a), en donde se muestra la tabla de verdad para un circuito que tiene dos entradas A y B, y la salida x. La tabla muestra que la salida x estará en el nivel 1 *sólo* para el caso en el que $A = 0$ y $B = 1$. Ahora lo que resta es determinar qué circuito lógico producirá esta operación deseada. Debería ser evidente que una de las posibles soluciones es la que se muestra en la figura 4-4(b). Aquí se utiliza una compuerta AND con las entradas $\overline{A}$ y B, de manera que $x = \overline{A} \cdot B$. Es obvio que x será 1 *sólo* si ambas entradas de la compuerta AND son 1, a saber, $\overline{A} = 1$ (lo cual significa que $A = 0$) y $B = 1$. Para todos los demás valores de A y B, la salida x será 0.

FIGURA 4-4 Circuito que produce una salida de 1 sólo para la condición en la que $A = 0$ y $B = 1$.

A	B	x
0	0	0
0	1	1
1	0	0
1	1	0

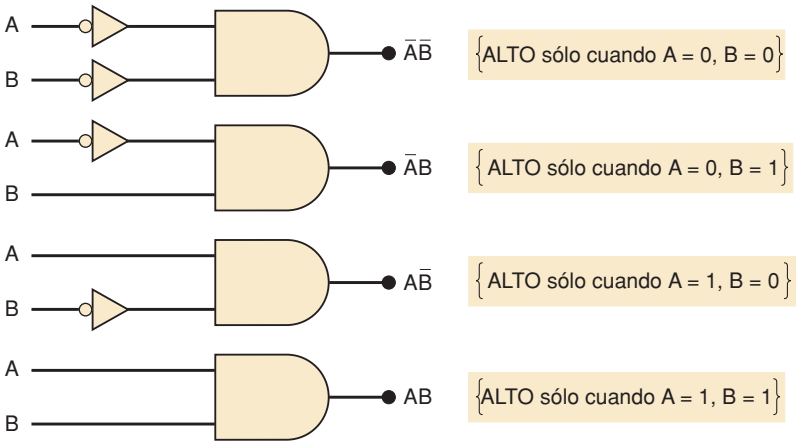
(a)



(b)

Puede usarse un enfoque similar para las demás condiciones de entrada. Por ejemplo, si x fuera a estar en nivel alto sólo para la condición $A = 1$, $B = 0$, el circuito resultante sería una compuerta AND con entradas A y $\overline{B}$. En otras palabras, para cualquiera de las cuatro posibles condiciones de entrada, podemos generar una salida x en nivel alto mediante el uso de una compuerta AND con las entradas apropiadas para generar el producto AND requerido. En la figura 4-5 se muestran los cuatro casos. Cada una de las compuertas AND que se muestran genera una salida que es 1 *sólo* para una condición de entrada dada y la salida es 0 para todas las demás condiciones. Hay que recalcar que las entradas AND son invertidas o no invertidas, dependiendo de los valores que tengan las variables para la condición dada. Si la variable es 0 para la condición dada, se invierte antes de entrar a la compuerta AND.

FIGURA 4-5 Una compuerta AND con las entradas apropiadas puede usarse para producir una salida de 1 para un conjunto específico de niveles de entrada.



Ahora consideremos el caso que se muestra en la figura 4-6(a), en donde tenemos una tabla de verdad que indica que la salida x debe ser 1 para dos casos distintos: $A = 0, B = 1$ y $A = 1, B = 0$. ¿Cómo puede implementarse esto? Sabemos que el término AND $\overline{A} \cdot B$ generará un 1 sólo para la condición $A = 0$, $B = 1$, y que el término AND $A \cdot \overline{B}$ generará un 1 para la condición $A = 1, B = 0$. Como x debe estar

en ALTO para *cualquiera* de esas condiciones, debe quedar claro que se debe aplicar una operación OR a estos términos para producir la salida x deseada. Esta implementación se muestra en la figura 4-6(b), en donde la expresión resultante para la salida es $x = \overline{A}B + A\overline{B}$.

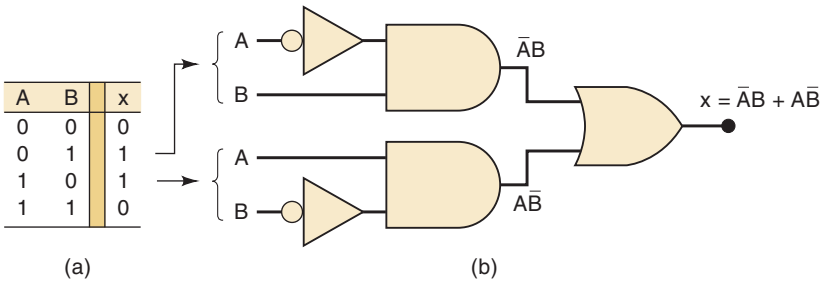


FIGURA 4-6 Cada conjunto de condiciones de entrada que debe producir una salida en ALTO se implementa mediante una compuerta AND separada. Se aplica una operación OR a las salidas de la compuerta AND para producir la salida final.

En este ejemplo se genera un término AND para cada caso en la tabla donde la salida x va a ser un 1. Después se aplica un OR a las salidas de la compuerta AND para producir la salida total x, que será 1 cuando cualquiera de los términos de la operación AND sea 1. Este mismo procedimiento puede extenderse a los ejemplos con más de dos entradas. Considere la tabla de verdad para un circuito de tres entradas (tabla 4-1). Aquí hay tres casos en donde la salida x debe ser 1. Se muestra el término AND requerido para cada uno de estos casos. Observe nuevamente que para cada caso en el que una variable es 0, ésta aparece invertida en el término AND. La expresión de suma de productos para x se obtiene aplicando una operación OR a los tres términos AND.

$$x = \overline{A}B\overline{C} + \overline{A}BC + ABC$$

TABLA 4-1

A	*B*	*C*	*x*	
0	0	0	0	
0	0	1	0	
0	1	0	1	$\rightarrow \overline{A}B\overline{C}$
0	1	1	1	$\rightarrow \overline{A}BC$
1	0	0	0	
1	0	1	0	
1	1	0	0	
1	1	1	1	$\rightarrow ABC$

Procedimiento completo de diseño

Cualquier problema lógico puede resolverse mediante el uso del siguiente procedimiento:

1. Interprete el problema y establezca una tabla de verdad para describir su operación.
2. Escriba el término AND (producto) para cada caso en el que la salida sea 1.
3. Escriba la expresión de suma de productos (SOP) para la salida.
4. Simplifique la expresión de salida, si es posible.
5. Implemente el circuito para la expresión final simplificada.

El siguiente ejemplo ilustra el procedimiento completo de diseño.

EJEMPLO 4-7

Diseñe un circuito lógico que tenga tres entradas A, B y C, y cuya salida esté en ALTO sólo cuando la mayoría de sus entradas estén en ALTO.

Solución

Paso 1. Establezca la tabla de verdad.

Con base en el enunciado del problema, la salida x deberá ser 1 siempre que dos o más entradas sean 1; para todos los demás casos, la salida deberá ser 0 (tabla 4-2).

TABLA 4-2

A	B	C	x	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\rightarrow \overline{A}BC$
1	0	0	0	
1	0	1	1	$\rightarrow A\overline{B}C$
1	1	0	1	$\rightarrow AB\overline{C}$
1	1	1	1	$\rightarrow ABC$

Paso 2. Escriba el término AND para cada caso en el que la salida sea un 1.

Hay cuatro casos así. Los términos AND se muestran enseguida de la tabla de verdad (tabla 4-2). Observe de nuevo que cada término AND contiene cada variable de entrada en su forma invertida o no invertida.

Paso 3. Escriba la expresión de suma de productos para la salida.

$$x = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

Paso 4. Simplifique la expresión de salida.

Esta expresión puede simplificarse de varias formas. Tal vez la más rápida sea considerar que el último término ABC tiene dos variables en común con cada uno de los otros términos. Por ende, podemos usar el término ABC para factorizarlo con cada uno de los demás términos. La expresión se reformula de manera que el término ABC ocurra tres veces (recuerde del ejemplo 4-2 que es posible hacer esto en el álgebra booleana):

$$x = \overline{A}BC + ABC + A\overline{B}C + ABC + AB\overline{C} + ABC$$

Si factorizamos los pares de términos apropiados, tenemos que

$$x = BC(\overline{A} + A) + AC(\overline{B} + B) + AB(\overline{C} + C)$$

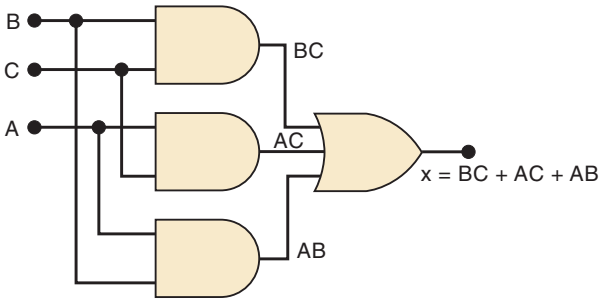
Cada término en paréntesis es igual a 1, por lo que nos queda

$$x = BC + AC + AB$$

Paso 5. Implemente el circuito para la expresión final.

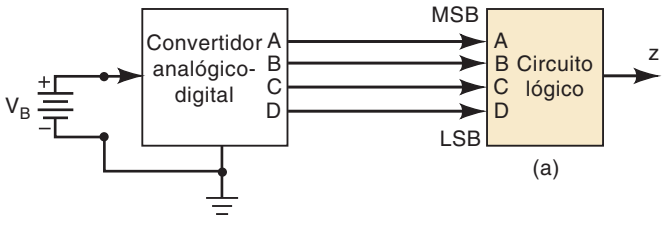
Esta expresión se implementa en la figura 4-7. Como la expresión se encuentra en la forma SOP, el circuito consiste de un grupo de compuertas AND conectadas con una sola compuerta OR.

FIGURA 4-7 Ejemplo 4-7.



EJEMPLO 4-8

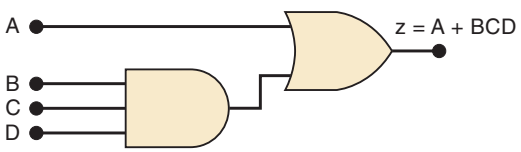
Consulte la figura 4-8(a), en donde un convertidor analógico-digital está monitoreando el voltaje de corriente directa de una batería de almacenamiento de 12 V en una nave espacial en órbita. La salida del convertidor es un número binario de 4 bits identificado como *ABCD*, que corresponde al voltaje de la batería en intervalos de 1 V, en donde *A* es el MSB. Las salidas binarias del convertidor se alimentan a un circuito lógico que debe producir una salida en ALTO siempre y cuando el valor binario sea mayor que $0110_2 = 6_{10}$; esto es, que el voltaje de la batería sea mayor que 6 V. Diseñe este circuito lógico.



(a)

	A	B	C	D	z
(0)	0	0	0	0	0
(1)	0	0	0	1	0
(2)	0	0	1	0	0
(3)	0	0	1	1	0
(4)	0	1	0	0	0
(5)	0	1	0	1	0
(6)	0	1	1	0	0
(7)	0	1	1	1	1→ $\bar{A}BCD$
(8)	1	0	0	0	1→ $A\bar{B}\bar{C}\bar{D}$
(9)	1	0	0	1	1→ $A\bar{B}\bar{C}D$
(10)	1	0	1	0	1→ $A\bar{B}C\bar{D}$
(11)	1	0	1	1	1→ $A\bar{B}CD$
(12)	1	1	0	0	1→ $AB\bar{C}\bar{D}$
(13)	1	1	0	1	1→ $AB\bar{C}D$
(14)	1	1	1	0	1→ $ABC\bar{D}$
(15)	1	1	1	1	1→ $ABCD$

(b)



(c)

FIGURA 4-8 Ejemplo 4-8.

Solución

La tabla de verdad se muestra en la figura 4-8(b). Para cada caso en la tabla de verdad hemos indicado el equivalente decimal del número binario representado por la combinación *ABCD*.

La salida *z* es igual a 1 para todos aquellos casos en los que el número binario sea mayor que 0110. Para todos los demás casos, *z* es igual a 0. Esta tabla de verdad nos da la siguiente expresión de suma de productos:

$$z = \bar{A}BCD + A\bar{B}\,\bar{C}\,\bar{D} + A\bar{B}\,\bar{C}D + A\bar{B}C\bar{D} + A\bar{B}CD + AB\bar{C}\,\bar{D} + AB\bar{C}D + ABC\bar{D} + ABCD$$

La simplificación de esta expresión será una formidable tarea, pero con un poco de cuidado puede lograrse. El proceso paso a paso implica factorizar y eliminar términos de la forma $A + \overline{A}$:

$$\begin{aligned} z &= \overline{A}BCD + A\overline{B}\,\overline{C}(\overline{D} + D) + A\overline{B}C(\overline{D} + D) + AB\overline{C}(\overline{D} + D) + ABC(\overline{D} + D) \\ &= \overline{A}BCD + A\overline{B}\,\overline{C} + A\overline{B}C + AB\overline{C} + ABC \\ &= \overline{A}BCD + A\overline{B}(\overline{C} + C) + AB(\overline{C} + C) \\ &= \overline{A}BCD + A\overline{B} + AB \\ &= \overline{A}BCD + A(\overline{B} + B) \\ &= \overline{A}BCD + A \end{aligned}$$

Esta expresión puede reducirse aún más si aplicamos el teorema (15a), el cual nos dice que $x + \overline{x}y = x + y$. En este caso $x = A$ y $y = BCD$. Por lo tanto,

$$z = \overline{A}BCD + A = BCD + A$$

Esta expresión final se implementa en la figura 4-8(c).

Como lo demuestra este ejemplo, el método de simplificación algebraica puede ser bastante largo cuando la expresión original contiene un gran número de términos. Ésta es una limitación que no comparte el método de mapeo de Karnaugh, como veremos más adelante.

EJEMPLO 4-9

Observe la figura 4-9(a). En una copiadora simple, se debe generar una señal de paro S para detener la operación de la máquina y encender una luz indicadora cada vez que exista una de las siguientes condiciones: (1) que no haya papel en la bandeja alimentadora; o (2) que se activen los dos microinterruptores en la ruta del papel, lo cual indica un atasco. La presencia de papel en la bandeja alimentadora se indica

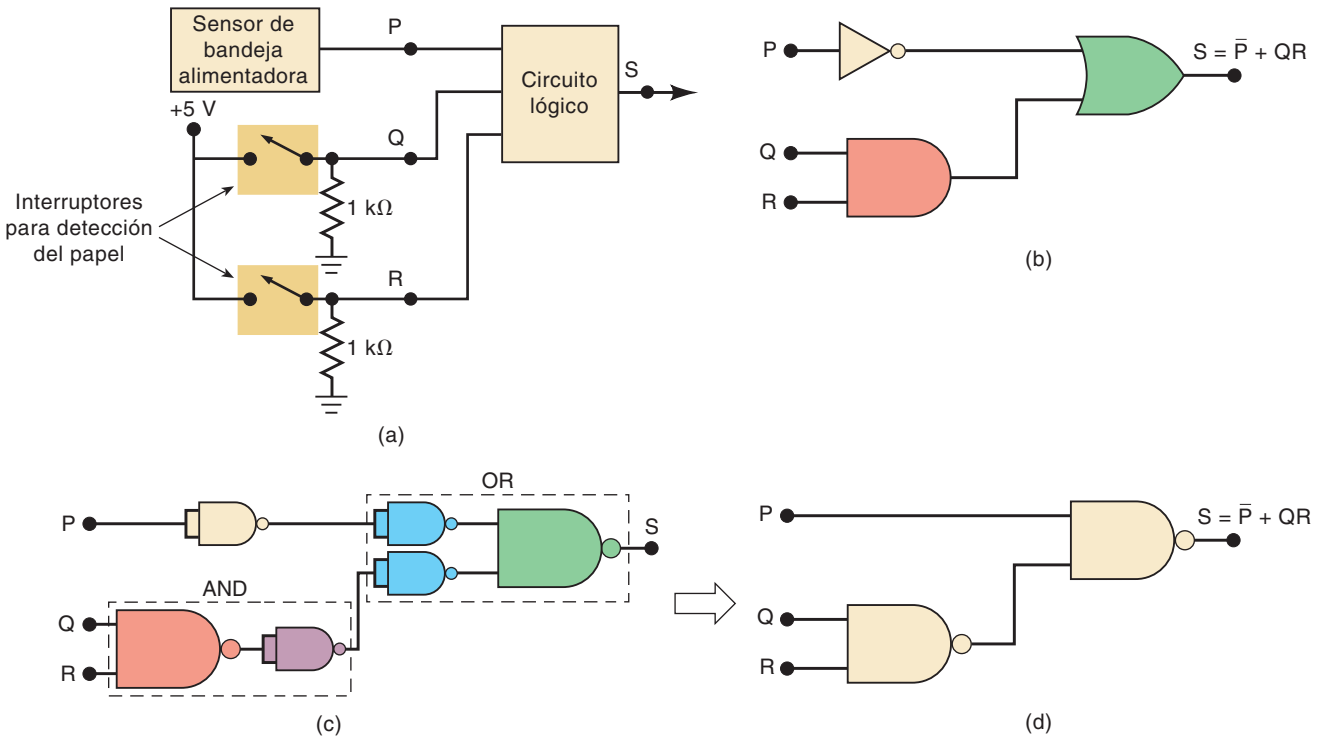


FIGURA 4-9 Ejemplo 4-9.

mediante un nivel ALTO en la señal lógica *P*. Cada microinterruptor produce una señal lógica (*Q* y *R*, *respectivamente*) que cambia a ALTO cada vez que el papel pasa por el interruptor y lo activa. Diseñe el circuito lógico para producir un nivel ALTO en la señal de salida *S* para las condiciones antes mencionadas, e impleméntelo utilizando el chip con cuatro compuertas NAND de dos entradas 74HC00 CMOS.

Solución

Utilizaremos el proceso de cinco pasos indicado en el ejemplo 4-7. La tabla de verdad se muestra en la figura 4-3. La salida *S* será un 1 lógico siempre que $P = 0$, ya que esto indica que no hay papel en la bandeja alimentadora. *S* también será un 1 para los dos casos en los que *Q* y *R* sean ambas 1, lo cual indica un atasco de papel. Como la tabla indica, hay cinco condiciones de entrada distintas que producen una salida en ALTO. **(Paso 1)**

TABLA 4-3

P	*Q*	*R*	*S*	
0	0	0	1	$\overline{P}\,\overline{Q}\,\overline{R}$
0	0	1	1	$\overline{P}\,\overline{Q}R$
0	1	0	1	$\overline{P}Q\overline{R}$
0	1	1	1	$\overline{P}QR$
1	0	0	0	
1	0	1	0	
1	1	0	0	
1	1	1	1	PQR

Se muestran los términos AND para cada uno de estos casos. **(Paso 2)**
La expresión de suma de productos sería

$$S = \overline{P}\,\overline{Q}\,\overline{R} + \overline{P}\,\overline{Q}R + \overline{P}Q\overline{R} + \overline{P}QR + PQR \qquad \textbf{(Paso 3)}$$

Podemos empezar la simplificación factorizando $\overline{P}\,\overline{Q}$ de los términos 1 y 2, y factorizando $\overline{P}Q$ de los términos 3 y 4:

$$S = \overline{P}\,\overline{Q}(\overline{R} + R) + \overline{P}Q(\overline{R} + R) + PQR \qquad \textbf{(Paso 4)}$$

Ahora podemos eliminar los términos $R + R$ ya que son iguales a 1:

$$S = \overline{P}\,\overline{Q} + \overline{P}Q + PQR$$

Si factorizamos $\overline{P}$ de los términos 1 y 2 podemos eliminar *Q*:

$$S = \overline{P} + PQR$$

Aquí, podemos aplicar el teorema (15b) ($\overline{x} + xy = \overline{x} + y$) para obtener

$$S = \overline{P} + QR$$

Como una comprobación adicional de esta ecuación booleana simplificada, veamos si concuerda con la tabla de verdad con la que comenzamos. Esta ecuación dice que la salida *S* estará en ALTO siempre que *P* esté en BAJO OR cuando *Q* AND *R* estén en ALTO. Consulte la tabla 4-3 y observe que la salida está en ALTO para los cuatro casos en los que *P* está en BAJO. *S* también está en ALTO cuando *Q* AND *R*

están ambas en ALTO, sin importar el estado de *P*. Lo anterior concuerda con la ecuación.

La implementación AND/OR para este circuito se muestra en la figura 4-9(b).

(Paso 5)

Para implementar este circuito usando el chip NAND de dos entradas 74HC00, debemos convertir cada una de las compuertas y el INVERSOR en sus compuertas NAND equivalentes (con base en la sección 3-12). Esto se muestra en la figura 4-9(c). Es evidente que podemos eliminar los inversores dobles para producir la implementación con compuertas NAND que se muestra en la figura 4-9(d).

El circuito alambrado final se obtiene mediante la conexión de dos de las compuertas NAND en el chip 74HC00. Este chip CMOS tiene la misma configuración de compuertas y los mismos números de terminales que el chip 74LS00 TTL de la figura 3-31. La figura 4-10 muestra el circuito alambrado con números de terminales, incluyendo las terminales +5 V y tierra (GND). También incluye un transistor excitador de salida y un LED para indicar el estado de la salida *S*.

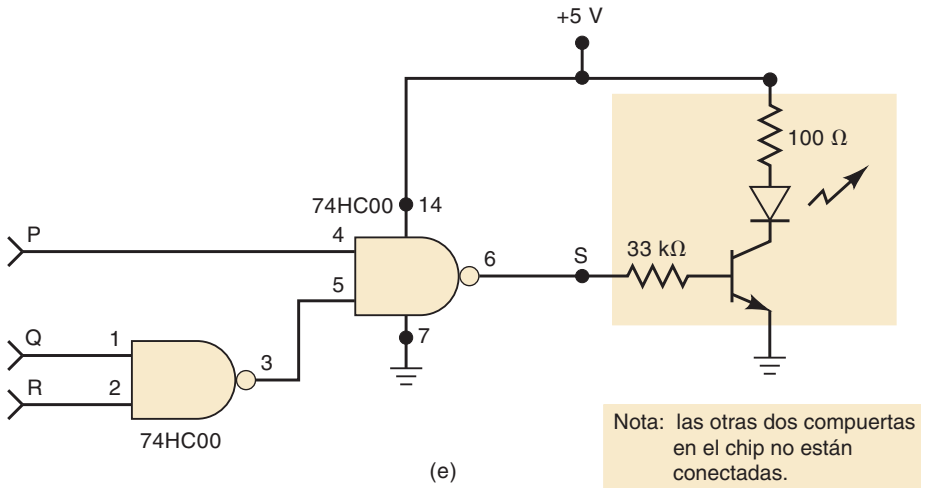


FIGURA 4-10 Circuito para la figura 4-9(d) implementado mediante el uso del chip NAND 74HC00.

PREGUNTAS DE REPASO

1. Escriba la expresión de suma de productos para un circuito con cuatro entradas y una salida que debe estar en ALTO sólo cuando la entrada *A* esté en BAJO al mismo tiempo que dos de las otras entradas estén en BAJO.
2. Implemente la expresión de la pregunta 1 utilizando sólo compuertas NAND de cuatro entradas. ¿Cuántas se requieren?

4-5 MÉTODO DE MAPAS DE KARNAUGH

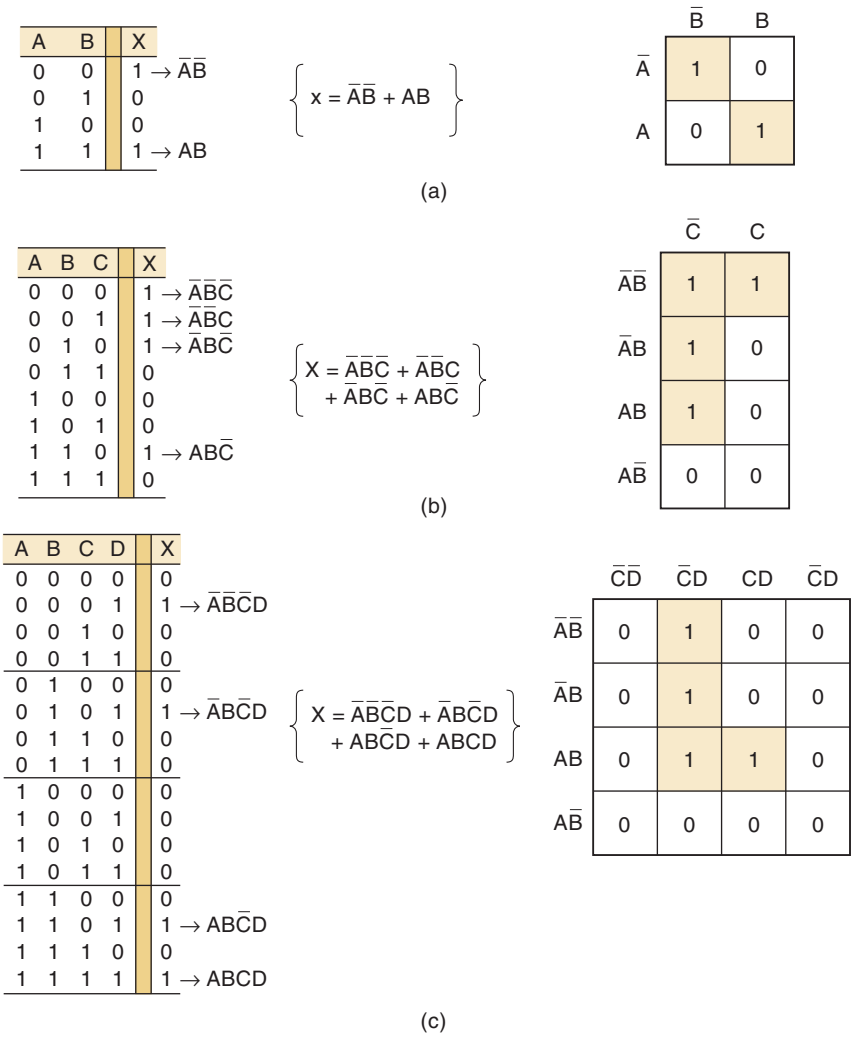
El **mapa de Karnaugh (mapa K)** es una herramienta gráfica que se utiliza para simplificar una ecuación lógica o convertir una tabla de verdad en su correspondiente circuito lógico mediante un proceso simple y ordenado. Aunque un mapa K puede usarse para problemas en los que se involucre cualquier número de variables de entrada, su utilidad práctica está limitada a cinco o seis variables. Los siguientes problemas estarán limitados a un máximo de cuatro entradas, ya que los problemas con cinco o más entradas son demasiado complicados y se resuelven mejor mediante el uso de un programa de computadora.

Formato del mapa de Karnaugh

Al igual que una tabla de verdad, el mapa K es un medio para mostrar la relación entre las entradas lógicas y la salida deseada. La figura 4-11 muestra tres ejemplos de mapas K para dos, tres y cuatro variables, junto con sus correspondientes tablas de verdad. Estos ejemplos ilustran los siguientes puntos importantes:

1. La tabla de verdad proporciona el valor de la salida X para cada combinación de valores de entrada. El mapa K proporciona la misma información en un formato distinto. Cada caso en la tabla de verdad corresponde a una casilla en el mapa K. Por ejemplo, en la figura 4-11(a) la condición $A = 0, B = 0$ corresponde la casilla $\overline{A}\,\overline{B}$ en el mapa K. Como la tabla de verdad muestra $X = 1$ para este caso, se coloca un 1 en la casilla $\overline{A}\,\overline{B}$ del mapa K. De manera similar, la condición $A = 1, B = 1$ en la tabla de verdad corresponde a la casilla AB del mapa K. Como $X = 1$ para este caso, se coloca un 1 en la casilla AB. Todas las demás casillas se llenan con 0s. Esta misma idea se utiliza en los mapas con tres y cuatro variables que se muestran en la figura.
2. Las casillas del mapa K se etiquetan de manera que las casillas adyacentes en forma horizontal difieran sólo por una variable. Por ejemplo, la casilla de la esquina superior izquierda en el mapa de cuatro variables es $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$, mientras que la casilla que se encuentra justo a su derecha es $\overline{A}\,\overline{B}\,\overline{C}\,D$ (sólo la variable D es distinta). De manera similar, las casillas adyacentes verticalles sólo difie-

FIGURA 4-11 Mapas de Karnaugh y tablas de verdad para (a) dos, (b) tres y (c) cuatro variables.



ren por una variable. Por ejemplo, la casilla de la esquina superior izquierda es $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$, mientras que la casilla que está justo debajo es $\overline{A}\,\overline{B}\,\overline{C}\,D$ (sólo la variable B es distinta).

Observe que cada casilla en la fila superior se considera como adyacente a una casilla correspondiente en la fila inferior. Por ejemplo, la casilla $\overline{A}\,\overline{B}CD$ en la fila superior es adyacente a la casilla $A\overline{B}CD$ en la fila inferior, ya que sólo difieren por la variable A. Podemos considerar que la parte superior del mapa se dobla para tocar su parte inferior. De manera similar, las casillas de la columna más a la izquierda son adyacentes a las correspondientes en la columna más a la derecha.

3. Para que las casillas adyacentes en forma vertical y horizontal difieran sólo por una variable, el etiquetado de arriba hacia abajo debe realizarse en el orden mostrado: $\overline{A}\,\overline{B}$, $\overline{A}B$, AB, $A\overline{B}$. Lo mismo aplica para el etiquetado de izquierda a derecha: $\overline{C}\,\overline{D}$, $\overline{C}D$, CD, $C\overline{D}$.
4. Una vez que se ha llenado un mapa K con 0s y 1s, puede obtenerse la expresión de suma de productos para la salida X mediante la aplicación de la operación OR a todas las casillas que contengan un 1. En el mapa de tres variables de la figura 4-11(b), las casillas $\overline{A}\,\overline{B}\,\overline{C}$, $\overline{A}\,\overline{B}C$, $\overline{A}B\overline{C}$ y $AB\overline{C}$ contienen un 1, de manera que $X = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + AB\overline{C}$.

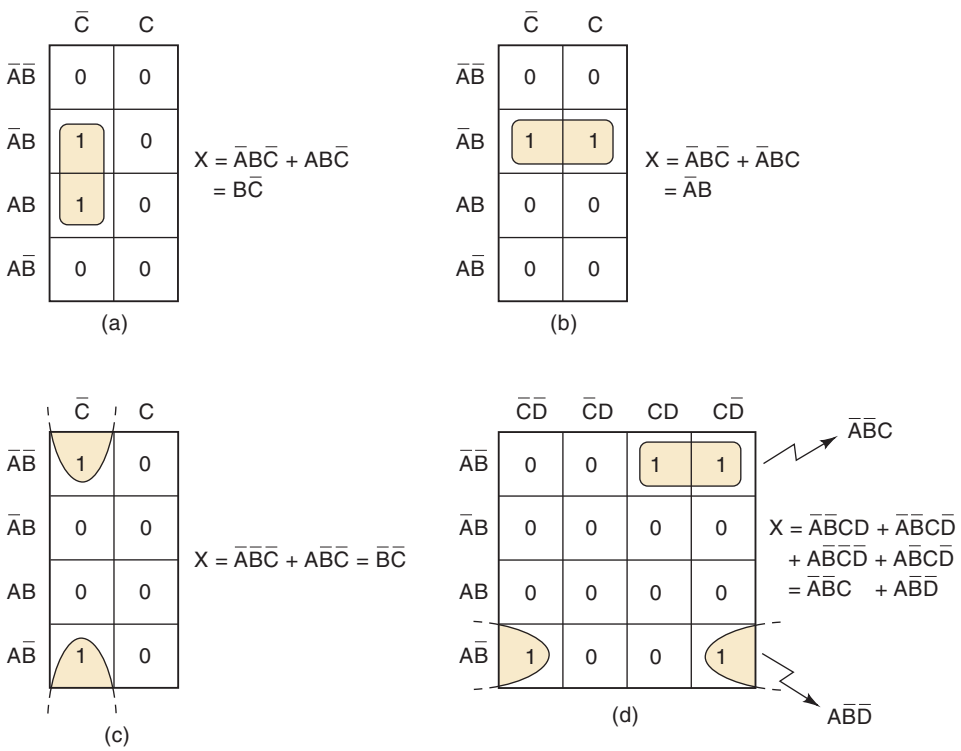
Agrupamiento

La expresión para la salida X puede simplificarse mediante la combinación apropiada de las casillas en el mapa K que contengan 1s. Al proceso para combinar estos 1s se le conoce como **agrupamiento**.

Agrupamiento de pares (grupos de dos)

La figura 4-12(a) es el mapa K para cierta tabla de verdad de tres variables. Este mapa contiene un par de 1s que son adyacentes en forma vertical; el primero representa a $\overline{A}B\overline{C}$ y el segundo a $AB\overline{C}$. Observe que en estos dos términos, sólo la variable A aparece

FIGURA 4-12
Ejemplos de agrupamientos de pares de 1s adyacentes.



tanto en forma normal como complementada (invertida), mientras que B y $\overline{C}$ permanecen sin cambios. Estos dos términos pueden agruparse (combinarse) para obtener un resultante que elimine la variable A, ya que aparece tanto en forma complementada como no complementada. Es fácil demostrarlo de la siguiente manera:

$$\begin{aligned} X &= \overline{A}B\overline{C} + AB\overline{C} \\ &= B\overline{C}(\overline{A} + A) \\ &= B\overline{C}(1) = B\overline{C} \end{aligned}$$

Este mismo principio se aplica para cualquier par de 1s adyacentes en forma vertical u horizontal. La figura 4-12(b) muestra un ejemplo de dos 1s adyacentes en forma horizontal; los cuales pueden agruparse, y la variable C puede eliminarse ya que aparece tanto en su forma no complementada como en su forma complementada, para obtener un resultante de $X = AB$.

La figura 4-12(c) muestra otro ejemplo. En un mapa K, la fila superior y la fila inferior de casillas se consideran adyacentes. Por ende, los dos 1s en este mapa pueden agruparse para obtener un resultante de $\overline{A}\,\overline{B}\,\overline{C} + A\overline{B}\,\overline{C} = \overline{B}\,\overline{C}$.

La figura 4-12(d) muestra un mapa K que tiene dos pares de 1s que pueden agruparse. Los dos 1s en la fila superior son adyacentes en forma horizontal. Los dos 1s en la fila inferior también son adyacentes ya que, en un mapa K la columna más a la izquierda y la columna más a la derecha se consideran adyacentes. Cuando se agrupa el par de 1s de la parte superior se elimina la variable D (ya que aparece como D y como $\overline{D}$) para producir el término $\overline{A}\,\overline{B}C$. Al agrupar el par de la parte inferior se elimina la variable C para producir el término $A\overline{B}\,\overline{D}$. Después se aplica una operación OR a estos dos términos para obtener el resultado final para X.

En resumen:

Al agrupar un par de 1s adyacentes en un mapa K se elimina la variable que aparece tanto en forma no complementada como en forma complementada.

Agrupamiento de cuádruples (grupos de cuatro)

Un mapa K puede contener un grupo de cuatro 1s que sean adyacentes. A este grupo se le conoce como *cuádruple*. La figura 4-13 muestra varios ejemplos de este tipo. En la figura 4-13(a), los cuatro 1s son adyacentes en forma vertical y en la figura 4-13(b) son adyacentes en forma horizontal. El mapa K de la figura 4-13(c) contiene cuatro 1s en una casilla y se consideran adyacentes entre sí. Los cuatro 1s de la figura 4-13(d) también son adyacentes, al igual que los de la figura 4-13(e) ya que, como dijimos antes, las filas superior e inferior se consideran como adyacentes entre sí, al igual que las columnas más a la izquierda y más a la derecha.

Cuando se agrupa un cuádruple, el término resultante sólo contendrá las variables que no cambian su forma en todas las casillas del cuádruple. Por ejemplo, en la figura 4-13(a) las cuatro casillas que contienen un 1 son $\overline{A}\,\overline{B}C$, $\overline{A}BC$, ABC y $A\overline{B}C$. Si examinamos estos términos descubriremos que sólo la variable C permanece sin cambios (tanto A como B aparecen en forma complementada y no complementada). Por ende, la expresión resultante para X es tan sólo $X = C$. Esto puede demostrarse de la siguiente manera:

$$\begin{aligned} X &= \overline{A}\,\overline{B}C + \overline{A}BC + ABC + A\overline{B}C \\ &= \overline{A}C(\overline{B} + B) + AC(B + \overline{B}) \\ &= \overline{A}C + AC \\ &= C(\overline{A} + A) = C \end{aligned}$$

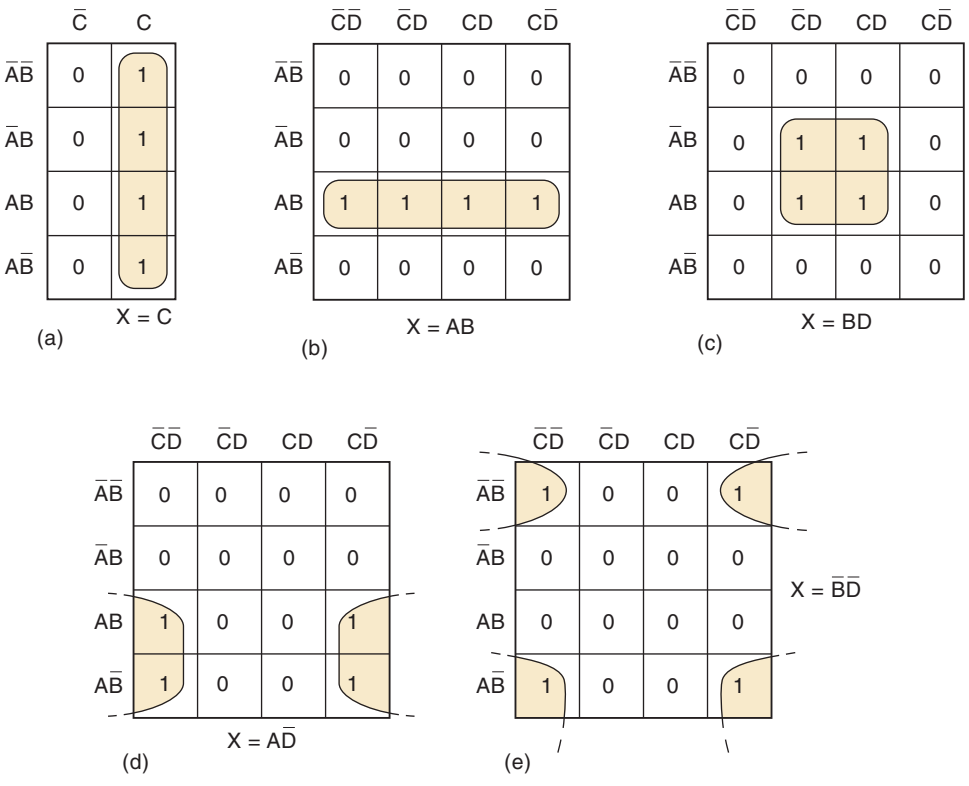


FIGURA 4-13
Ejemplos de agrupamiento de cuádruples.

Como otro ejemplo, considere la figura 4-13(d), en donde las cuatro casillas que contienen 1s son $ABC\overline{D}$, $A\overline{B}\,\overline{C}\,\overline{D}$, $ABC\overline{D}$ y $A\overline{B}C\overline{D}$. Al analizar estos términos podemos ver que sólo las variables A y D permanecen sin cambios, de manera que la expresión simplificada para X es

$$X = A\overline{D}$$

Lo que puede demostrarse de la misma forma que se hizo antes. El lector deberá comprobar cada uno de los otros casos de la figura 4-13 para verificar las expresiones indicadas para X.

En resumen:

Al agrupar un cuádruple de 1s adyacentes se eliminan las dos variables que aparecen tanto en forma complementada como en forma no complementada.

Agrupamiento de octetos (grupos de ocho)

A un grupo de ocho 1s adyacentes entre sí se le conoce como *octeto*. La figura 4-14 muestra varios ejemplos de octetos. Cuando se agrupa un octeto en un mapa de cuatro variables se eliminan tres de ellas, ya que sólo una permanece sin cambios. Por ejemplo, si examinamos las ocho casillas agrupadas en la figura 4-14(a) podremos ver que sólo la variable B se encuentra en la misma forma para las ocho casillas: las demás variables aparecen en su forma complementada y no complementada. En consecuencia, para este mapa $X = B$. El lector puede verificar los resultados para los demás ejemplos de la figura 4-14.

En resumen:

Al agrupar un octeto de 1s adyacentes se eliminan las tres variables que aparecen tanto en su forma complementada como en su forma no complementada.

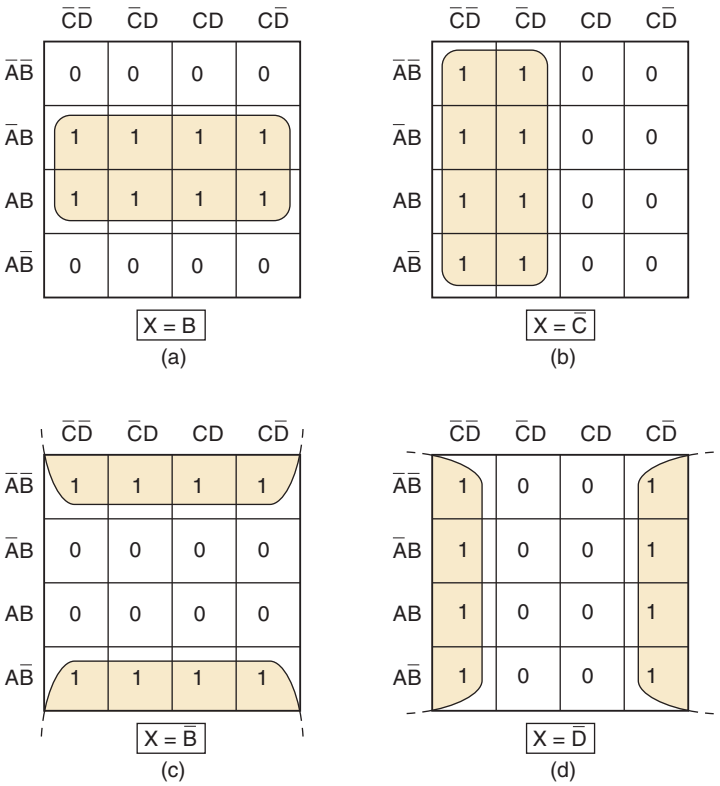


FIGURA 4-14
Ejemplos de agrupamiento de octetos.

Proceso completo de simplificación

Hemos visto cómo se puede utilizar el agrupamiento de pares, cuádruples y octetos en un mapa K para obtener una expresión simplificada. Podemos resumir la regla para los agrupamientos de *cualquier* tamaño, de la siguiente manera:

> **Cuando una variable aparece tanto en su forma complementada como no complementada dentro de un grupo, esa variable se elimina de la expresión. Las variables que son iguales para todas las casillas del grupo deben aparecer en la expresión final.**

Debe quedar claro que un agrupamiento mayor de 1s elimina más variables. Para ser exacto, un agrupamiento de dos elimina una variable, un agrupamiento de cuatro elimina dos variables y un agrupamiento de ocho elimina tres. Ahora utilizaremos este principio para obtener una expresión lógica simplificada a partir de un mapa K que contenga cualquier combinación de 1s y 0s.

Primero describiremos el procedimiento y después lo aplicaremos en varios ejemplos. Los siguientes pasos son el procedimiento mediante el uso del método del mapa K, para simplificar una expresión booleana:

Paso 1 Construya el mapa K y coloque 1s en las casillas que correspondan a los 1s en la tabla de verdad. Coloque 0s en las demás casillas.

Paso 2 Examine el mapa en busca de 1s adyacentes y marque los que *no* sean adyacentes con cualquier otro 1. A éstos se les conoce como 1s *aislados*.

Paso 3 A continuación busque los 1s que sean adyacentes sólo con otro 1. Agrupe *cualquier* par que contenga este tipo de 1s.

Paso 4 Agrupe cualquier octeto, aún y cuando contenga algunos 1s que ya se hayan agrupado.

Paso 5 Agrupe cualquier cuádruple que contenga uno o más 1s que no se hayan agrupado ya, *asegurándose de utilizar el número mínimo de grupos*.

Paso 6 Agrupe cualquier par necesario para incluir todos los 1 que no se hayan agrupado todavía, *asegurándose de utilizar el número mínimo de agrupamientos.*

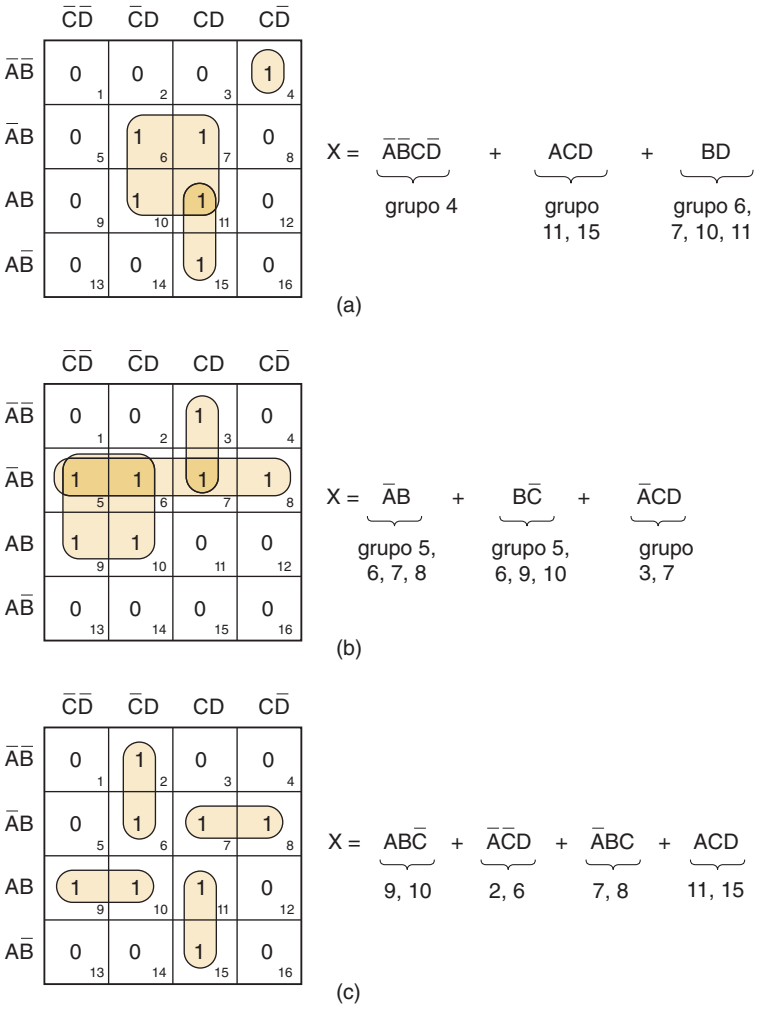
Paso 7 Forme la suma OR de todos los términos generados, uno por cada grupo.

En los siguientes ejemplos seguiremos al pie de la letra cada uno de estos pasos y haremos referencia a ellos. En cada caso, la expresión lógica resultante estará en su forma de suma de productos más simple.

EJEMPLO 4-10

La figura 4-15(a) muestra el mapa K para un problema con cuatro variables. Supondremos que el mapa se obtuvo de la tabla de verdad del problema (**paso 1**). Las casillas están numeradas para identificar cada uno de los grupos.

FIGURA 4-15
Ejemplos 4-10 al 4-12.



Paso 2 La casilla 4 es la única que contiene un 1 que no es adyacente con cualquier otro 1. Se agrupa y se identifica como grupo 4.

Paso 3 La casilla 15 es adyacente *sólo* con la casilla 11. Este par se agrupa y se identifica como grupo 11, 15.

Paso 4 No hay octetos.

Paso 5 Las casillas 6, 7, 10 y 11 forman un cuádruple. Este cuádruple se agrupa (grupo 6, 7, 10, 11). Observe que la casilla 11 se utiliza de nuevo, aun y cuando forma parte del grupo 11, 15.

Paso 6 Ya se han agrupado todos los 1s.

Paso 7 Cada grupo genera un término en la expresión para X. El grupo 4 es $\overline{A}\,\overline{B}C\overline{D}$. El grupo 11, 15 es ACD (se elimina la variable B). El grupo 6, 7, 10, 11 es BD (se eliminan A y C).

EJEMPLO 4-11

Considere el mapa K de la figura 4-15(b). Una vez más podemos suponer que ya se ha realizado el paso 1.

Paso 2 No hay 1s aislados.

Paso 3 El 1 en la casilla 3 es adyacente *sólo* con el 1 en la casilla 7. Al agrupar este par (grupo 3, 7) se produce el término $\overline{A}CD$.

Paso 4 No hay octetos.

Paso 5 Hay dos cuádruples: El primero lo forman los cuadros 5, 6, 7 y 8 . Al agrupar este cuádruple se produce el término $\overline{A}B$. El segundo está compuesto por las casillas 5, 6, 9 y 10. Se debe agrupar este cuádruple, ya que contiene dos casillas que no se han agrupado antes. Al agruparlo se produce $\overline{B}C$.

Paso 6 Ya se han agrupado todos los 1s.

Paso 7 Se aplica la operación OR a los términos generados por los tres grupos y se obtiene la expresión para X.

EJEMPLO 4-12

Considere el mapa K de la figura 4-15(c):

Paso 2 No hay 1s aislados.

Paso 3 El 1 en la casilla 2 es adyacente sólo para el 1 en la casilla 6. Este par se agrupa para producir $\overline{A}\,\overline{C}D$. De manera similar, la casilla 9 es adyacente sólo con la casilla 10. Al agrupar este par se produce $AB\overline{C}$. De igual forma, los grupos 7, 8 y 11, 15 producen los términos $\overline{A}BC$ y ACD, en forma correspondiente.

Paso 4 No hay octetos.

Paso 5 Sólo hay un cuádruple formado por las casillas 6, 7, 10 y 11. No obstante, este cuádruple *no* se agrupa debido a que todos los 1s que contiene ya se han incluido en otros grupos.

Paso 6 Ya se han agrupado todos los 1s.

Paso 7 La expresión para X se muestra en la figura.

EJEMPLO 4-13

Considere el mapa K de la figura 4-16(a).

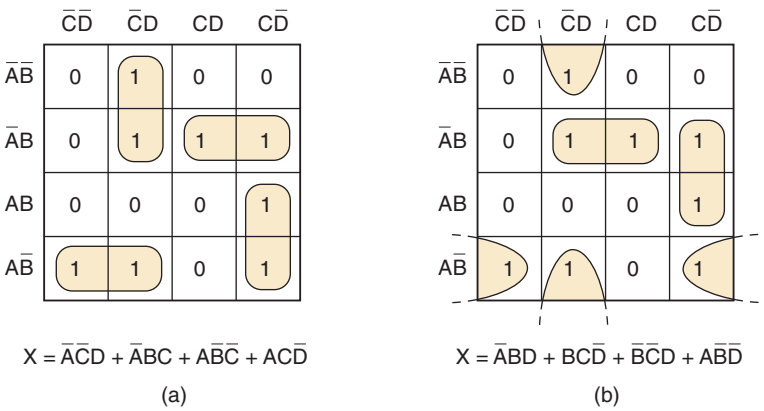


FIGURA 4-16 El mismo mapa K con dos soluciones correctas similares.

Paso 2 No hay 1s aislados.

Paso 3 No hay 1s que sean adyacentes sólo con otro 1.

Paso 4 No hay octetos.

Paso 5 No hay cuádruples.

Pasos 6 y 7 Hay muchos pares posibles. El agrupamiento debe utilizar el número mínimo de grupos para tomar en cuenta a todos los 1s. Para este mapa hay *dos* grupos posibles, los cuales requieren sólo cuatro pares agrupados. La figura 4-16(a) muestra una solución y su expresión resultante. La figura 4-16(b) muestra la otra. Observe que ambas expresiones son de la misma complejidad, por lo cual ninguna es mejor que la otra.

Cómo llenar un mapa K a partir de una expresión de salida

Cuando la salida deseada se presenta como expresión booleana en vez de tabla de verdad, el mapa K puede llenarse mediante el uso de los siguientes pasos:

1. Cambie la expresión a su forma SOP, en caso de que no se encuentre ya en esa forma.
2. Para cada término de productos en la expresión SOP, coloque un 1 en cada casilla del mapa K cuya etiqueta contenga la misma combinación de variables de entrada. Coloque un 0 en todas las demás casillas.

El siguiente ejemplo ilustra este procedimiento.

EJEMPLO 4-14

Use un mapa K para simplificar la expresión $y = \overline{C}(\overline{A}\,\overline{B}\,\overline{D} + D) + A\overline{B}C + \overline{D}$.

Solución

1. Multiplique el primer término para obtener $y = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{C}D + A\overline{B}C + \overline{D}$, que se encuentra ahora en la forma SOP.
2. Para el término $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$ sólo necesita colocar un 1 en la casilla $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$ del mapa K (figura 4-17). Para el término $\overline{C}D$ coloque un 1 en todas las casillas que tengan $\overline{C}D$ en sus etiquetas: $\overline{A}\,\overline{B}\,\overline{C}D$, $\overline{A}B\overline{C}D$, $AB\overline{C}D$, $A\overline{B}\,\overline{C}D$. Para el término $A\overline{B}C$ coloque un 1 en todas las casillas que tengan un $A\overline{B}C$ en sus etiquetas: $A\overline{B}C\overline{D}$, $A\overline{B}CD$. Para el término $\overline{D}$ coloque un 1 en todas las casillas que tengan una $\overline{D}$ en sus etiquetas: en todas las casillas de las columnas más a la izquierda y más a la derecha.

FIGURA 4-17 Ejemplo 4-14.

	$\overline{C}\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\overline{B}$	1	1	0	1
$\overline{A}B$	1	1	0	1
AB	1	1	0	1
$A\overline{B}$	1	1	1	1

$y = A\overline{B} + \overline{C} + \overline{D}$

Ahora el mapa K está lleno y puede agruparse para simplificar la expresión. Verifique que un agrupamiento apropiado produzca la expresión $y = A\overline{B} + \overline{C} + \overline{D}$.

Condiciones "No importa"

Algunos circuitos lógicos pueden diseñarse de manera que haya ciertas condiciones de entrada para las cuales no existan niveles de salida especificados, por lo general, debido a que estas condiciones de entrada nunca ocurrirán. En otras palabras, habrá ciertas combinaciones de niveles de entrada en las que "no importa" si la salida está en ALTO o en BAJO. Esto se ilustra en la tabla de verdad de la figura 4-18(a).

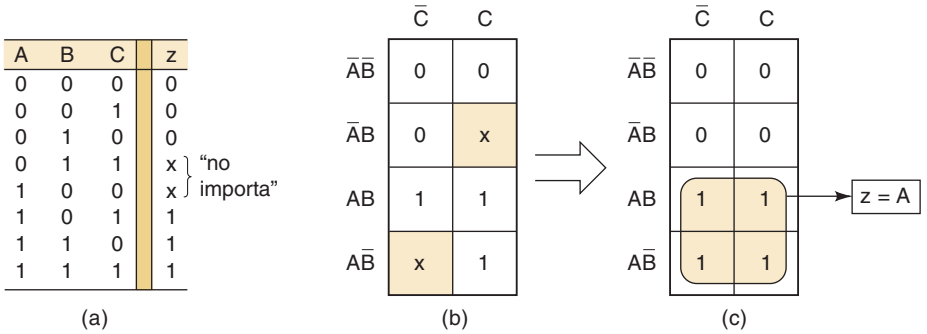


FIGURA 4-18 Las condiciones "No importa" deben cambiarse por 0 o 1 para producir un agrupamiento del mapa K que genere la expresión más simple.

Aquí la salida z no se especifica como 0 o 1 para las condiciones $A, B, C = 1, 0, 0$ y $A, B, C = 0, 1, 1$. En vez de ello se muestra una x para estas condiciones. La x representa la **condición de "no importa"**. Este tipo de condición puede surgir debido a varias razones; es la más común que en algunas situaciones nunca podrán ocurrir ciertas combinaciones de entradas, por lo que no hay una salida especificada para estas condiciones.

Un diseñador de circuitos tiene la libertad de hacer que la salida para cualquier condición de "no importa" sea un 0 o un 1 para producir la expresión de salida más simple. Por ejemplo, el mapa K para esta tabla de verdad se muestra en la figura 4-18(b), con una x en las casillas $A\overline{B}\,\overline{C}$ y $\overline{A}BC$. Aquí la mejor opción para el diseñador sería cambiar la x de la casilla $A\overline{B}\,\overline{C}$ por un 1 y la x de la casilla $\overline{A}BC$ por un 0, ya que esto produciría un cuádruple que puede agruparse para producir $z = A$, como se muestra en la figura 4-18(c).

Siempre que ocurran condiciones de "no importa", debemos decidir cuál x se va a cambiar por 0 y cuál por 1 para producir el mejor agrupamiento del mapa K (es decir, el grupo más grande que resulta en la expresión más simple). Esta decisión no siempre es fácil. Varios de los problemas al final del capítulo le ayudarán a aumentar su experiencia para tratar los casos de "no importa". He aquí otro ejemplo.

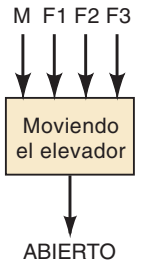
EJEMPLO 4-15

Vamos a diseñar un circuito lógico que controla la puerta de un elevador en un edificio de tres pisos. El circuito de la figura 4-19(a) tiene cuatro entradas. M es una señal lógica que indica cuando se mueve el elevador ($M = 1$) o cuando está detenido ($M = 0$). $F1$, $F2$ y $F3$ son señales indicadoras de cada piso que, por lo general, están en BAJO, y cambian a ALTO sólo cuando el elevador está posicionado en ese piso. Por ejemplo, cuando el elevador está alineado con el segundo piso, $F2 = 1$ y $F1 = F3 = 0$. La salida del circuito es la señal *ABIERTO* que, por lo general, está en BAJO y cambia a ALTO cuando se va a abrir la puerta del elevador.

Podemos llenar la tabla de verdad para la salida *ABIERTO* [Figura 4-19(b)] de la siguiente manera:

1. Como el elevador no puede alinearse con más de un piso a la vez, sólo una de las entradas de los pisos puede estar en ALTO en un momento dado. Esto significa que todos aquellos casos en la tabla de verdad en los que más de una entrada de piso esté en 1 serán condiciones de "no importa". Podemos colocar una x en la

FIGURA 4-19 Ejemplo 4-15.



(a)

M	F1	F2	F3	ABIERTO
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	X
0	1	0	0	1
0	1	0	1	X
0	1	1	0	X
0	1	1	1	X
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	X
1	1	0	0	0
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

(b)

	$\overline{F2}\,\overline{F3}$	$\overline{F2}F3$	$F2F3$	$F2\overline{F3}$
$\overline{M}\,\overline{F1}$	0	1	X	1
$\overline{M}F1$	1	X	X	X
$M F1$	0	X	X	X
$M\overline{F1}$	0	0	X	0

(c)

	$\overline{F2}\,\overline{F3}$	$\overline{F2}F3$	$F2F3$	$F2\overline{F3}$
$\overline{M}\,\overline{F1}$	0	1	1	1
$\overline{M}F1$	1	1	1	1
$M F1$	0	0	0	0
$M\overline{F1}$	0	0	0	0

$\text{OPEN} = \overline{M}\,(F1 + F2 + F3)$

(d)

columna de la salida *ABIERTO* para los ocho casos en los que más de una entrada *F* es 1.

2. Si analizamos los otros ocho casos, cuando $M = 1$ el elevador se está moviendo, por lo que *ABIERTO* debe ser un 0 ya que no deseamos que se abra la puerta del elevador. Cuando $M = 0$ (el elevador está detenido) queremos que *ABIERTO* = 1 siempre y cuando una de las entradas de piso sea 1. Cuando $M = 0$ y todas las entradas de piso son 0, el elevador está detenido pero no está alineado en forma apropiada con ninguno de los pisos, por lo que queremos que *ABIERTO* = 0 para mantener la puerta cerrada.

Ahora la tabla de verdad está completa y podemos transferir su información al mapa K de la figura 4-19(c). El mapa sólo tiene tres 1s, pero ocho condiciones de "no importa". Al cambiar cuatro de estas casillas de "no importa" por 1s, podemos producir grupos de cuádruples que contengan los 1s originales [Figura 4-19(d)]. Esto es lo mejor que podemos hacer en cuanto a minimizar la expresión de salida. Verifique que los agrupamientos produzcan la expresión de salida que se muestra para *ABIERTO*.

Resumen

El proceso de mapa K tiene varias ventajas en comparación con el método algebraico. El mapeo K es un proceso más ordenado, con pasos bien definidos en comparación con el proceso de prueba y error que se utiliza algunas veces en la simplificación algebraica. Por lo general, el mapeo K requiere menos pasos, en especial para las expresiones que contienen muchos términos, y siempre produce una expresión mínima.

Sin embargo, algunos instructores prefieren el método algebraico debido a que requiere un profundo conocimiento del álgebra booleana y no es tan sólo un procedimiento mecánico. Cada método tiene sus ventajas y, aunque la mayoría de los diseñadores lógicos son adeptos en ambos, ser proficiente en uno de ellos es todo lo que se necesita para producir resultados aceptables.

Existen otras técnicas más complejas que utilizan los diseñadores para minimizar circuitos lógicos con más de cuatro entradas. Estas técnicas se adecuan en forma especial a los circuitos con grandes cantidades de entradas, en donde no puede considerarse el método algebraico ni el mapeo K. La mayoría de estas técnicas puede traducirse a un programa de computadora que realizará la minimización con base en los datos de entrada que suministre la tabla de verdad o la expresión sin simplificar.

PREGUNTAS DE REPASO

1. Utilice el mapeo K para obtener la expresión del ejemplo 4-7.
2. Utilice el mapeo K para obtener la expresión del ejemplo 4-8. Aquí se debe enfatizar la ventaja del mapeo K para expresiones que contengan muchos términos.
3. Obtenga la expresión del ejemplo 4-9, utilizando un mapa K.
4. ¿Qué es una condición de "no importa"?

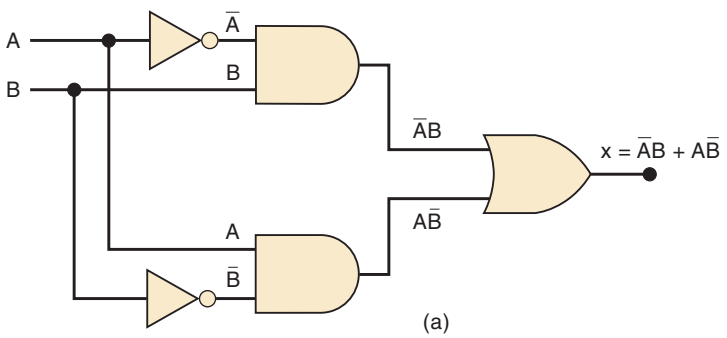
4-6 CIRCUITOS OR EXCLUSIVO Y NOR EXCLUSIVO

Dos de los circuitos lógicos especiales que se presentan con mucha frecuencia en los sistemas digitales son el *OR exclusivo* y el *NOR exclusivo*.

OR exclusivo

Considere el circuito lógico de la figura 4-20(a). La expresión de salida de este circuito es

$$x = \overline{A}B + A\overline{B}$$



A	B	x
0	0	0
0	1	1
1	0	1
1	1	0

Símbolos de compuerta XOR

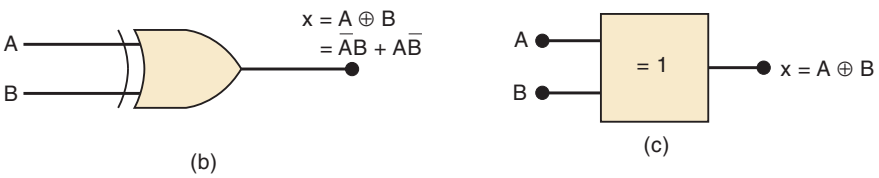


FIGURA 4-20
(a) Circuito OR exclusivo y su tabla de verdad; (b) símbolo tradicional de la compuerta XOR; (c) símbolo IEEE/ANSI para la compuerta XOR.

La tabla de verdad que acompaña a este circuito muestra que $x = 1$ para dos casos: $A = 0, B = 1$ (el término $\overline{A}B$) y $A = 1, B = 0$ (el término $A\overline{B}$). En otras palabras:

Este circuito produce una salida en ALTO siempre que las dos entradas se encuentran en los niveles opuestos.

Éste es el circuito **OR exclusivo**, que de aquí en adelante se abreviará como **XOR**.

Esta combinación específica de compuertas lógicas se produce con mucha frecuencia y es muy útil en ciertas aplicaciones. De hecho, al circuito XOR se le ha otorgado su propio símbolo, el cual se muestra en la figura 4-20(b). Se asume que este símbolo contiene toda la lógica dentro del circuito XOR y, por lo tanto, tiene la misma expresión lógica y la misma tabla de verdad. Por lo general, al circuito XOR se le conoce como *compuerta* XOR, y lo consideramos como otro tipo de compuerta lógica. En la figura 4-20(c) se muestra el símbolo IEEE/ANSI para una compuerta XOR. El símbolo de notación de dependencia (= 1) dentro del bloque indica que la salida será activa en ALTO *sólo* cuando una de las entradas esté en ALTO.

Una compuerta XOR sólo tiene *dos* entradas; no hay compuertas XOR de tres ni de cuatro entradas. Las dos entradas se combinan de manera que $x = \overline{A}B + A\overline{B}$. Una forma abreviada que se utiliza algunas veces para indicar la expresión de salida XOR es

$$x = A \oplus B$$

en donde el símbolo $\oplus$ representa la operación de la compuerta XOR.

A continuación se sintetizan las características de una compuerta XOR:

1. Sólo tiene dos entradas y su salida es

$$x = \overline{A}B + A\overline{B} = A \oplus B$$

2. Su salida está en *ALTO* sólo cuando las dos entradas se encuentran en niveles *distintos*.

Hay varios CIs disponibles que contienen compuertas XOR. Los que se listan a continuación son chips que contienen cuatro compuertas XOR.

74LS86	Chip con cuatro compuertas XOR (familia TTL)
74C86	Chip con cuatro compuertas XOR (familia CMOS)
74HC86	XOR (CMOS de alta velocidad)

NOR exclusivo

El circuito **NOR exclusivo** (que se abrevia como **XNOR**) opera en forma completamente opuesta al circuito XOR. La figura 4-21(a) muestra un circuito XNOR y su tabla de verdad correspondiente. La expresión de salida es

$$x = AB + \overline{A}\,\overline{B}$$

lo cual indica junto con la tabla de verdad que x será 1 para dos casos: $A = B = 1$ (el término AB) y $A = B = 0$ (el término $\overline{A}\,\overline{B}$). En otras palabras:

El circuito XNOR produce una salida en ALTO siempre que las dos entradas se encuentran en el mismo nivel.

De todo esto podemos deducir que la salida del circuito XNOR es el inverso exacto de la salida del circuito XOR. El símbolo tradicional para una compuerta

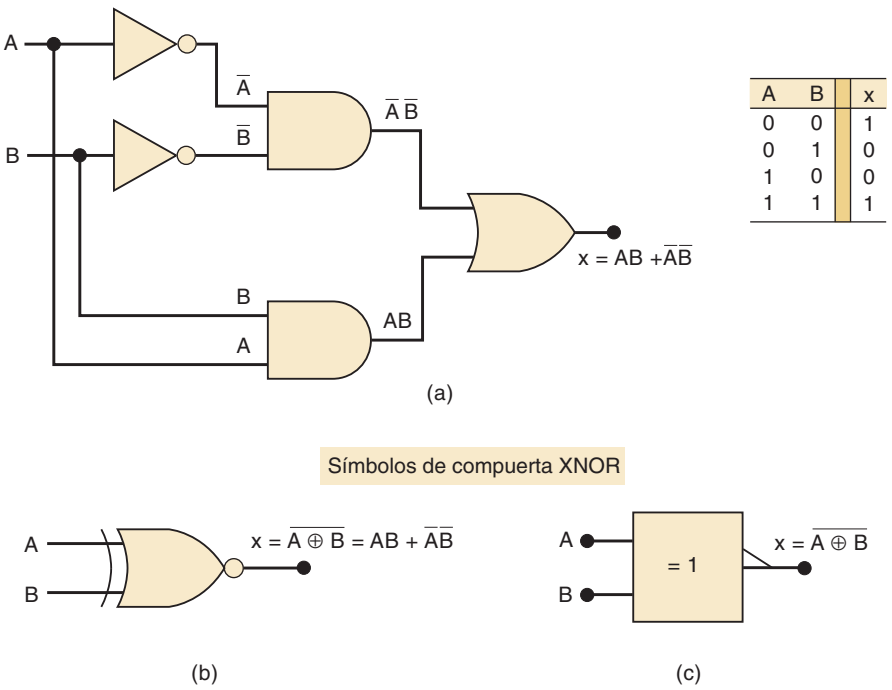


FIGURA 4-21
(a) Circuito NOR exclusivo; (b) símbolo tradicional para la compuerta XNOR; (c) símbolo IEEE/ANSI.

XNOR se obtiene con sólo agregar un pequeño círculo en la salida del símbolo XOR [figura 4-21(b)]. El símbolo IEEE/ANSI agrega el pequeño triángulo en la salida del símbolo XOR. Ambos símbolos indican una salida que cambia a su estado de activo en BAJO cuando *sólo una* de las entradas está en ALTO.

La compuerta XNOR también tiene *sólo dos* entradas, y las combina de manera que su salida sea

$$x = AB + \overline{A}\,\overline{B}$$

Una forma abreviada de indicar la expresión de salida de la compuerta XNOR es

$$x = \overline{A \oplus B}$$

la cual es el inverso de la operación XOR. La compuerta XNOR se puede sintetizar de la siguiente manera:

1. Sólo tiene dos entradas y su salida es

$$x = AB + \overline{A}\,\overline{B} = \overline{A \oplus B}$$

2. Su salida está en ALTO sólo cando las dos entradas se encuentran en el *mismo* nivel.

Hay varios CIs disponibles que contienen compuertas XNOR. Los que se listan a continuación son chips que contienen cuatro compuertas XNOR.

74LS266	Chip con cuatro compuertas XNOR (familia TTL)
74C266	Chip con cuatro compuertas XNOR (CMOS)
74HC266	Chip con cuatro compuertas XNOR (CMOS de alta velocidad)

Sin embargo, cada uno de estos chips consta de circuitos con salida especial que limita su uso a ciertos tipos especiales de aplicaciones. Es muy común que un diseñador lógico obtenga la función XNOR con sólo conectar la salida de una compuerta XOR a un INVERSOR.

EJEMPLO 4-16

Determine la forma de onda de salida para las formas de onda de entrada que se muestran en la figura 4-22.

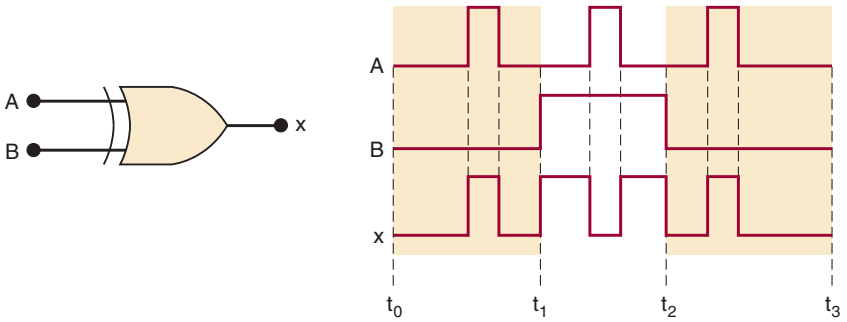


FIGURA 4-22 Ejemplo 4-16.

Solución

La forma de onda de salida se obtiene mediante el hecho de que la salida XOR estará en ALTO sólo cuando sus entradas se encuentren en distintos niveles. La forma de onda de salida resultante revela varios puntos interesantes:

1. La forma de onda de x concuerda con la forma de onda de entrada A durante los intervalos de tiempo en los que $B = 0$. Esto ocurre durante los intervalos de tiempo t_0 a t_1 y t_2 a t_3.
2. La forma de onda de x es el *inverso* de la forma de onda de entrada A durante los intervalos de tiempo en los que $B = 1$. Esto ocurre durante el intervalo t_1 a t_2.
3. Estas observaciones muestran que una compuerta XOR puede utilizarse como *INVERSOR CONTROLADO*; es decir, que una de sus entradas puede utilizarse para controlar si se va a invertir o no la señal de la otra entrada. Esta propiedad puede ser útil en ciertas aplicaciones.

EJEMPLO 4-17

La notación x_1x_0 representa un número binario de dos bits que puede tener cualquier valor (00, 01, 10 o 11); por ejemplo, cuando $x_1 = 1$ y $x_0 = 0$ el número binario es 10, y así sucesivamente. De manera similar, y_1y_0 representa otro número binario de dos bits. Diseñe un circuito lógico en el que utilice las entradas x_1, x_0, y_1 y y_0, y cuya salida esté en ALTO sólo cuando los dos números binarios x_1x_0 y y_1y_0 sean *iguales*.

Solución

El primer paso es construir una tabla de verdad para las 16 condiciones de entrada (tabla 4-4). La salida z debe estar en ALTO siempre que los valores de x_1x_0 concuerden con los valores de y_1y_0; esto es, siempre que $x_1 = y_1$ y $x_0 = y_0$. La tabla muestra que hay cuatro de esos casos. Ahora podríamos continuar con el procedimiento normal, que sería obtener una expresión de suma de productos para z, tratar de simplificarla y después implementar el resultado. No obstante, la naturaleza de este problema lo hace ideal para que se implemente mediante el uso de compuertas XNOR, y con un poco de pensamiento se producirá una solución simple con el mínimo esfuerzo. Consulte la figura 4-23; en este diagrama lógico, x_1 y y_1 se alimentan de

TABLA 4-4

x_1	x_0	y_1	y_0	z (Salida)
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

una compuerta XNOR y x_0 y y_0 se alimentan a otra compuerta XNOR. La salida de cada compuerta XNOR estará en ALTO sólo cuando sus entradas sean iguales. Por ende, para $x_0 = y_0$ y $x_1 = y_1$ las salidas de ambas compuertas XNOR estarán en ALTO. Ésta es la condición que estamos buscando, ya que significa que los dos números de dos bits son iguales. La salida de la compuerta AND estará en ALTO sólo para este caso, con lo cual se producirá el resultado deseado.

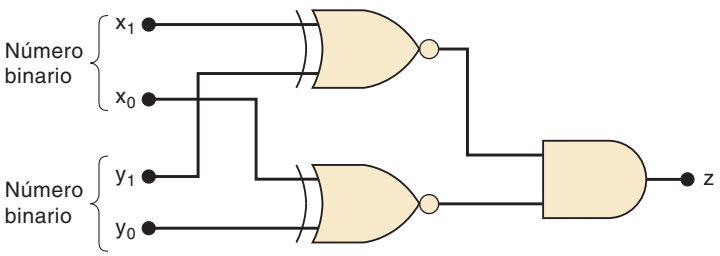


FIGURA 4-23 Circuito para detectar la igualdad de dos números binarios de dos bits.

EJEMPLO 4-18

Al simplificar la expresión para la salida de un circuito lógico combinacional, tal vez se encuentre con las operaciones XOR o XNOR cuando esté factorizando. A menudo esto nos lleva a utilizar compuertas XOR o XNOR en la implementación del circuito final. Para ilustrar lo anterior, simplifique el circuito de la figura 4-24(a).

Solución

La expresión sin simplificar para el circuito se obtiene como

$$z = ABCD + A\,\overline{B}\,\overline{C}\,D + \overline{A}\,\overline{D}$$

Podemos factorizar AD de los primeros dos términos:

$$z = AD(BC + \overline{B}\,\overline{C}) + \overline{A}\,\overline{D}$$

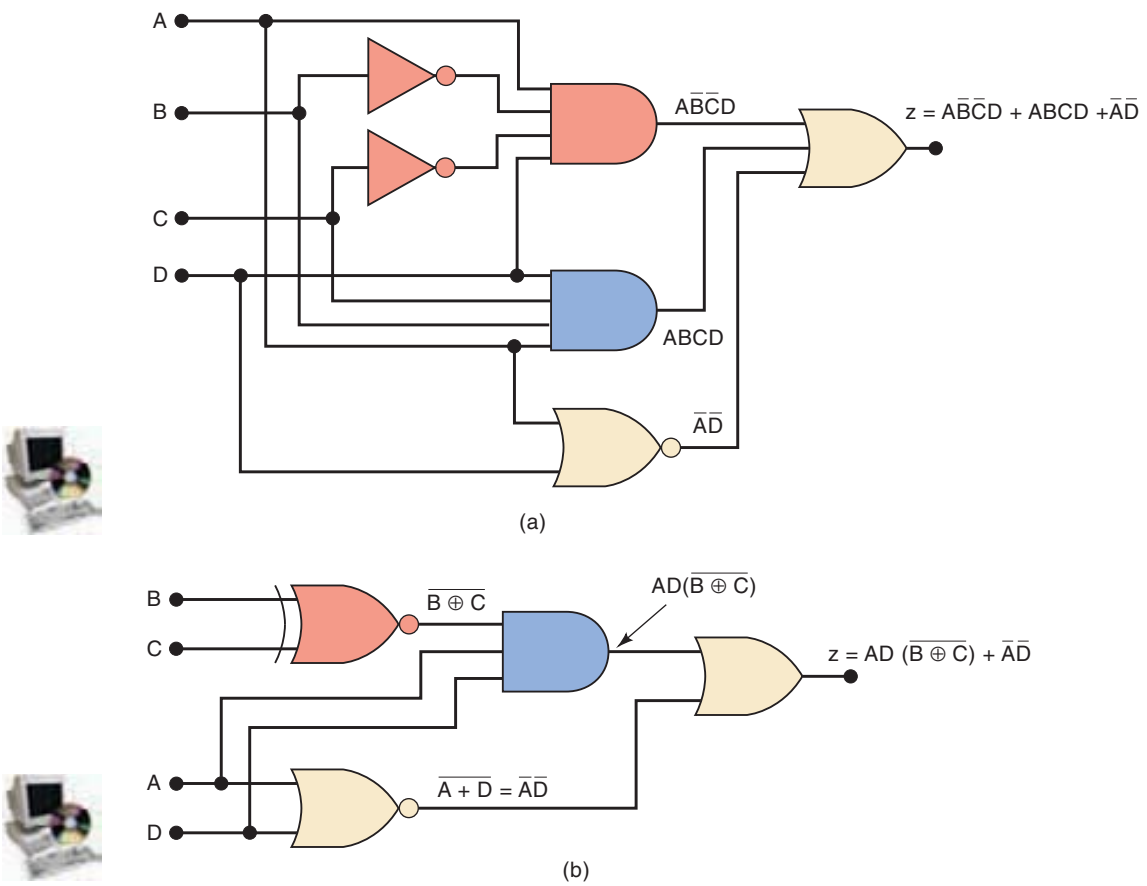


FIGURA 4-24 El ejemplo 4-18, que muestra cómo puede usarse una compuerta XNOR para simplificar la implementación de un circuito.

A primera instancia podría pensar que la expresión entre paréntesis puede sustituirse por un 1. Pero eso sería cierto sólo si la expresión fuera $BC + \overline{BC}$. Debería reconocer la operación entre paréntesis como la combinación XNOR de B y C. Este hecho puede usarse para volver a implementar el circuito como se muestra en la figura 4-24(b). Este circuito es mucho más simple que el original, ya que utiliza compuertas con menos entradas y se han eliminado dos INVERSORES.

PREGUNTAS DE REPASO

1. Use álgebra booleana para demostrar que la expresión de salida de la compuerta XNOR es el inverso exacto de la expresión de salida de la compuerta XOR.
2. ¿Cuál es la salida de una compuerta XNOR cuando se conectan a sus entradas una señal lógica y su inverso exacto?
3. Un diseñador lógico necesita un INVERSOR, y todo lo que hay disponible es una compuerta XOR de un chip 74HC86. ¿Necesita otro chip?

4-7 GENERADOR Y COMPROBADOR DE PARIDAD

En el capítulo 2 vimos que un transmisor puede adjuntar un bit de paridad a un conjunto de bits de datos antes de transmitirlos a un receptor. También vimos cómo esto permite al receptor detectar cualquier error de un solo bit que pueda haber ocurrido durante la transmisión. La figura 4-25 muestra un ejemplo de un tipo de

circuito lógico que se utiliza para la **generación de paridad** y la **comprobación de paridad**. Este ejemplo específico utiliza un grupo de cuatro bits como los datos que se van a transmitir, y utiliza un bit de paridad par. Puede adaptarse con facilidad para utilizar paridad impar y cualquier número de bits.

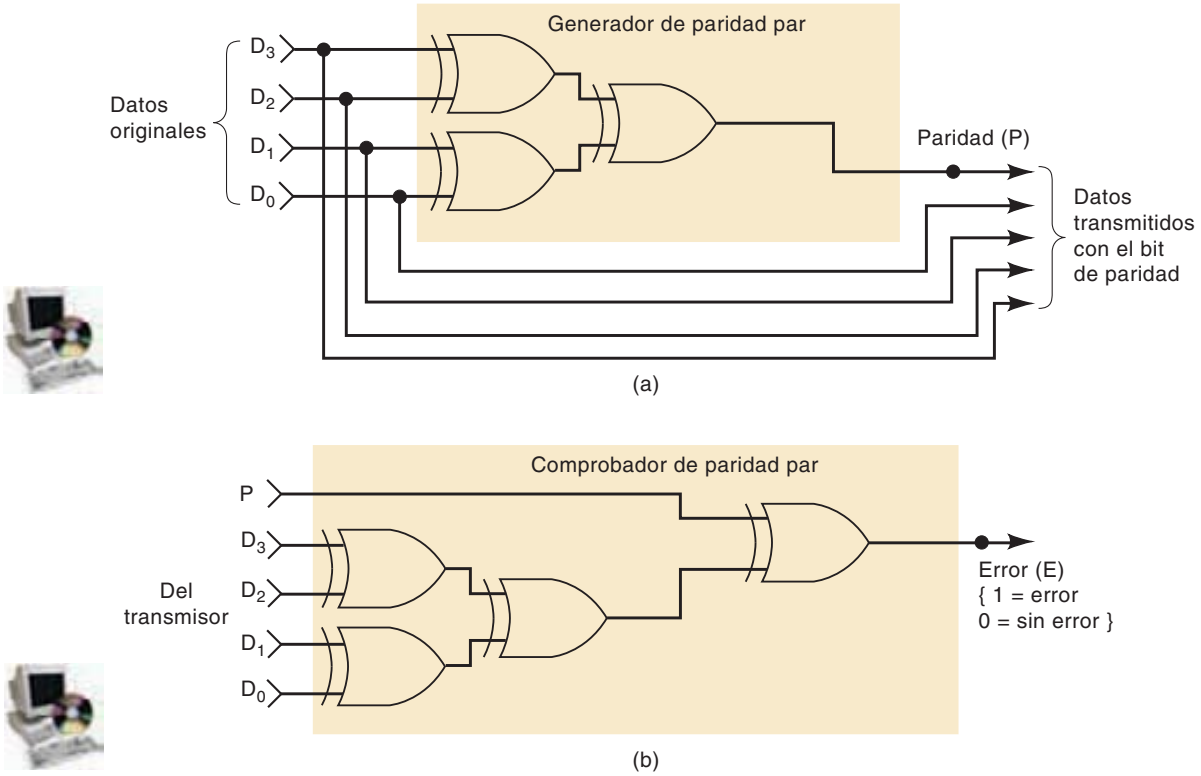


FIGURA 4-25 Compuertas XOR utilizadas para implementar (a) el generador de paridad y (b) el comprobador de paridad para un sistema con paridad par.

En la figura 4-25(a), el conjunto de datos que se van a transmitir se aplica al circuito generador de paridad, el cual produce el bit de paridad par *P* en su salida. Este bit de paridad se transmite al receptor junto con los bits de datos originales, formando un total de cinco bits. En la figura 4-25(b), estos cinco bits (datos + paridad) entran en el circuito comprobador de paridad del receptor, el cual produce una salida de error *E* que indica si ocurrió o no un error de un solo bit.

No debería sorprendernos demasiado que ambos circuitos empleen compuertas XOR, si consideramos que una sola compuerta XOR opera de manera que produce una salida de 1 si un número impar de sus entradas son 1, y una salida de 0 si un número par de sus entradas son 1.

EJEMPLO 4-19

Determine la salida del generador de paridad para cada uno de los siguientes conjuntos de datos de entrada, $D_3D_2D_1D_0$: (a) 0111; (b) 1001; (c) 0000; (d) 0100. Consulte la figura 4-25(a).

Solución

Para cada caso, aplique los niveles de datos a las entradas del generador de paridad y rastréelas a través de cada compuerta, hacia la salida *P*. Los resultados son: (a) 1; (b) 0; (c) 0; y (d) 1. Observe que *P* es 1 sólo cuando los datos originales contienen un número impar de 1s. Por ende, el número total de 1s que se envíen al receptor (datos + paridad) será par.

EJEMPLO 4-20

Determine la salida del comprobador de paridad [vea la figura 4-25(b)] para cada uno de los siguientes conjuntos de datos del transmisor:

	P	D_3	D_2	D_1	D_0
(a)	0	1	0	1	0
(b)	1	1	1	1	0
(c)	1	1	1	1	1
(d)	1	0	0	0	0

Solución

Para cada caso, aplique estos niveles a las entradas del comprobador de paridad y rastréelos a través de las compuertas hacia la salida *E*. Los resultados son: (a) 0; (b) 0; (c) 1; (d) 1. Observe que se produce un 1 en *E* sólo cuando aparece un número impar de 1s en las entradas que van hacia el comprobador de paridad. Esto indica que se ha producido un error, ya que se está utilizando la paridad par.

4-8 CIRCUITOS DE HABILITACIÓN/DESHABILITACIÓN

Cada una de las compuertas lógicas básicas puede utilizarse para controlar el paso de una señal lógica de entrada hacia la salida. Esto se describe en la figura 4-26, en donde se aplica una señal lógica *A* a una entrada de cada una de las compuertas lógicas básicas. La otra entrada de cada compuerta es la entrada de control *B*.

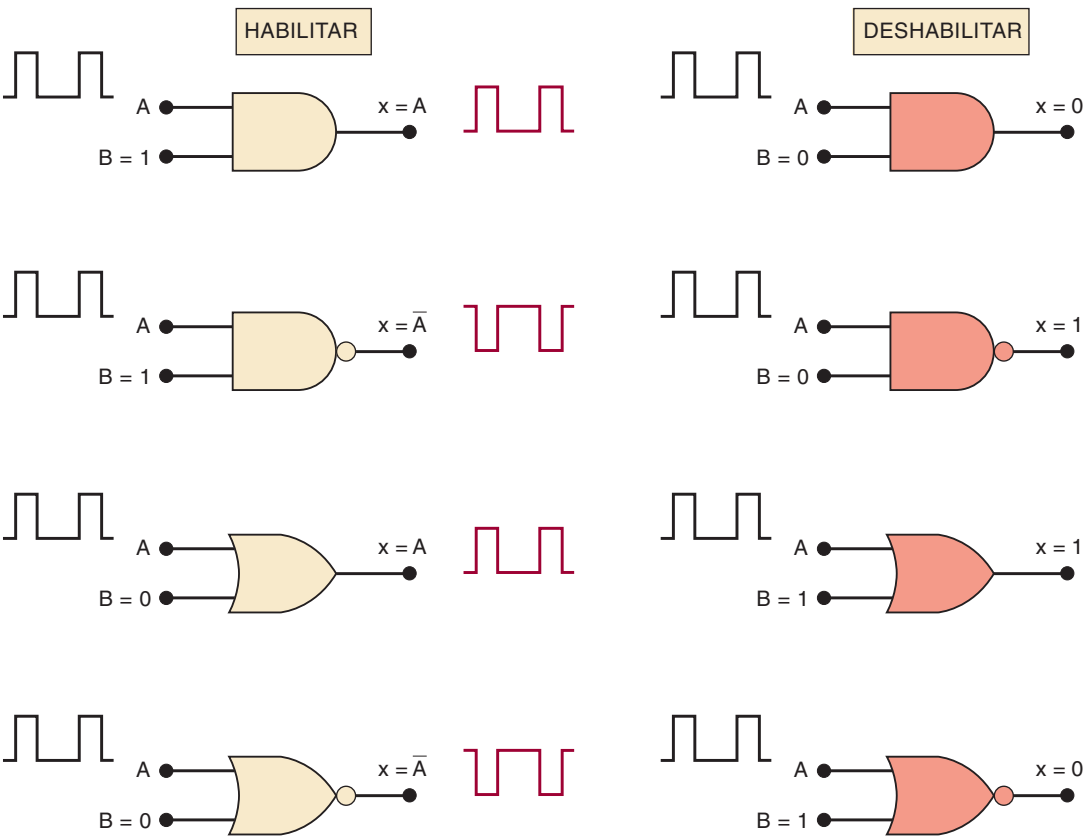


FIGURA 4-26 Las cuatro compuertas básicas pueden habilitar o deshabilitar el paso de una señal de entrada *A*, por medio del nivel lógico en la entrada de control *B*.

El nivel lógico en esta entrada de control determinará si la señal de entrada está **habilitada** para llegar a la salida o **deshabilitada** para que no pueda llegar a la salida. Esta acción de control explica por qué a estos circuitos se les empezó a llamar *compuertas*.

Si examina la figura 4-26 verá que cuando las compuertas no inversoras (AND, OR) están habilitadas, la salida sigue a la señal *A* de una manera exacta. En contraste, cuando las compuertas inversoras (NAND, NOR) están habilitadas, la salida es el inverso exacto de la señal *A*.

Observe también que en la figura las compuertas AND y NOR producen una salida constante en BAJO cuando se encuentran en la condición deshabilitada. En contraste, las compuertas NAND y OR producen una salida constante en ALTO cuando están deshabilitadas.

En el diseño de circuitos digitales se encontrará con muchas situaciones en las que se habilite o deshabilite el paso de una señal lógica, dependiendo de las condiciones presentes en una o más entradas de control. Los siguientes ejemplos muestran varias de estas situaciones.

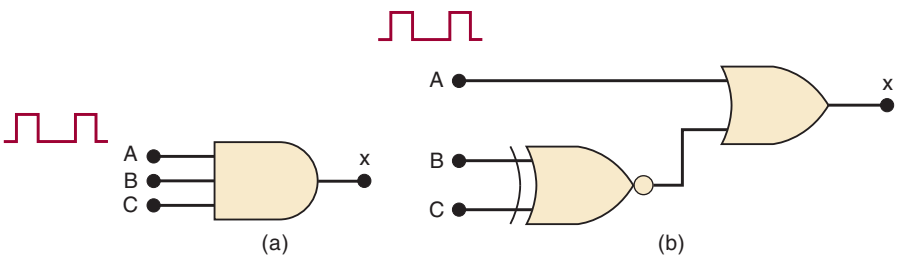
EJEMPLO 4-21

Diseñe un circuito lógico que permita que una señal pase a la salida sólo cuando las entradas de control *B* y *C* estén ambas en ALTO; en caso contrario, la salida deberá permanecer en BAJO.

Solución

Debe usarse una compuerta AND, ya que la señal debe pasarse sin invertir y la condición de salida de deshabilitación es un nivel BAJO. Como la condición de habilitación debe ocurrir sólo cuando $B = C = 1$ se debe usar una compuerta AND de tres entradas, como muestra la figura 4-27(a).

FIGURA 4-27
Ejemplos 4-21 y 4-22.



EJEMPLO 4-22

Diseñe un circuito lógico que permita que una señal pase hacia la salida sólo cuando una (pero no ambas) de sus entradas de control esté en ALTO; en caso contrario, la salida permanecerá en ALTO.

Solución

El resultado se dibuja en la figura 4-27(b). Se utiliza una compuerta OR porque queremos que la condición de deshabilitación de la salida sea un nivel ALTO, y no queremos invertir la señal. Las entradas de control *B* y *C* se combinan en una compuerta XNOR. Cuando *B* y *C* son distintas, la compuerta XNOR envía un nivel BAJO para habilitar la compuerta OR. Cuando *B* y *C* son iguales, la compuerta XNOR envía un nivel ALTO para deshabilitar la compuerta OR.

EJEMPLO 4-23

Diseñe un circuito lógico con la señal de entrada *A*, la entrada de control *B* y las salidas *X* y *Y*, que opere de la siguiente manera:

1. Cuando $B = 1$, la salida *X* seguirá a la entrada *A* y la salida *Y* será 0.
2. Cuando $B = 0$, la salida *X* será 0 y la salida *Y* seguirá a la entrada *A*.

Solución

Las dos salidas serán 0 cuando estén deshabilitadas y seguirán a la señal de entrada cuando estén habilitadas. Por lo tanto, debe utilizarse una compuerta AND para cada salida. Como X se debe habilitar cuando $B = 1$, su compuerta AND debe estar controlada por B, como muestra la figura 4-28. Como Y se debe habilitar cuando $B = 0$, su compuerta AND está controlada por $\overline{B}$. Al circuito de la figura 4-28 se le conoce como *circuito de dirección de pulso*, ya que dirige el pulso de entrada hacia una salida o la otra, dependiendo de B.

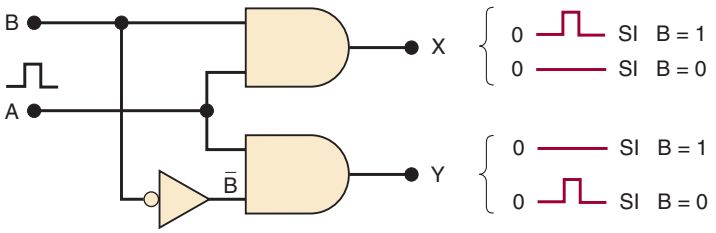


FIGURA 4-28 Ejemplo 4-23.

PREGUNTAS DE REPASO

1. Diseñe un circuito lógico con tres entradas A, B, C y una salida que cambie a BAJO sólo cuando A esté en ALTO mientras que B y C sean distintas.
2. ¿Cuáles compuertas lógicas producen una salida de 1 en el estado deshabilitado?
3. ¿Cuáles compuertas lógicas pasan el inverso de la señal de entrada cuando se habilitan?

4-9 CARACTERÍSTICAS BÁSICAS DE LOS CIS DIGITALES

Los CIs digitales son una colección de resistencias, diodos y transistores fabricados en una sola pieza de material semiconductor (por lo general, silicio), al cual se le conoce como *sustrato*, que por lo común se le denomina *chip*. El chip está encerrado en un encapsulado de plástico o cerámica protectora del cual salen terminales para conectar el CI con otros dispositivos. Uno de los tipos más comunes es el **encapsulado dual en línea (DIP)**, el cual se muestra en la figura 4-29(a), y se le llama así debido a que contiene dos filas paralelas de terminales. Estas terminales se numeran en sentido contrario al de las manecillas del reloj, viéndolas desde la parte superior del encapsulado con respecto a una muesca o punto de identificación en un extremo del encapsulado [vea la figura 4-29(b)]. El DIP que se muestra aquí es un encapsulado de 14 terminales que mide 0.75 pulg por 0.25 pulg; también se utilizan encapsulados de 16, 20, 24, 28, 40 y 64 terminales.

La figura 4-29(c) muestra que el chip de silicio es mucho más pequeño que el DIP; por lo general, es de 0.05 pulgadas cuadradas. El chip de silicio se conecta a las terminales del DIP mediante alambres muy finos [con diámetro de una milésima de pulgada (1 mil)].

El DIP es tal vez el encapsulado de CI digital más común que se encuentra en el equipo digital antiguo, pero actualmente se han hecho más populares otros tipos de encapsulados. El CI que se muestra en la figura 4-29(d) es sólo uno de los muchos encapsulados comunes en los circuitos digitales modernos. Este tipo específico utiliza puntas en forma de J que se encorvan por debajo del CI. En el capítulo 8 veremos otros tipos de encapsulados de CI.

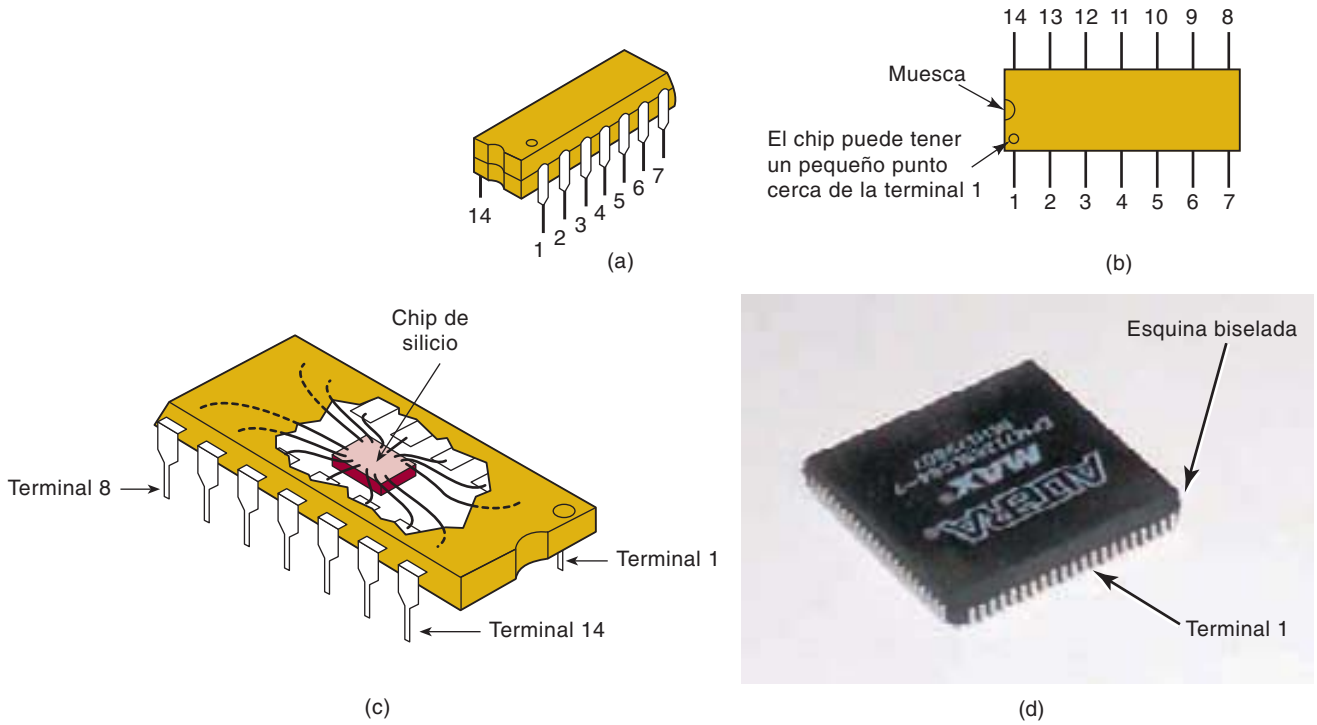


FIGURA 4-29 (a) Encapsulado dual en línea (DIP); (b) vista superior; (c) el chip de silicio es mucho más pequeño que el encapsulado protector; (d) encapsulado PLCC.

A menudo los CIs digitales se clasifican de acuerdo con la complejidad de sus circuitos, con base en el número de compuertas lógicas equivalentes en el sustrato. En la actualidad existen seis niveles de complejidad que, por lo común, se definen como se muestra en la tabla 4-5.

TABLA 4-5

Complejidad	Compuertas por chip
Integración a pequeña escala (SSI)	Menos de 12
Integración a mediana escala (MSI)	De 12 a 99
Integración a gran escala (LSI)	De 100 a 9999
Integración a muy grande escala (VLSI)	De 10,000 a 99,999
Integración a ultra gran escala (ULSI)	De 100,000 a 999,999
Integración a giga escala (GSI)	1,000,000 o más

Todos los CIs específicos a los que se hizo referencia en el capítulo 3 y en este capítulo son chips **SSI** con un número pequeño de compuertas. En los sistemas digitales modernos, los dispositivos con integración a mediana escala (**MSI**) e integración a gran escala (**LSI, VLSI, ULSI, GSI**) realizan la mayor parte de las funciones que alguna vez requirieron de varios tableros de circuitos llenos de dispositivos SSI. No obstante, los chips SSI se siguen utilizando como "interfaz" o "pegamento" entre estos chips más complejos. Los CIs de pequeña escala facilitan el aprendizaje de los fundamentos de los sistemas digitales. En consecuencia, muchos cursos basados en laboratorio utilizan estos CIs para construir y probar pequeños proyectos.

Ahora el mundo industrial de la electrónica digital se ha concentrado en los dispositivos lógicos programables (PLDs) para implementar un sistema digital de cualquier tamaño. Algunos PLDs simples están disponibles en encapsulados DIP,

pero los dispositivos lógicos programables más complejos requieren muchas más terminales de las que están disponibles en los DIPs. Los circuitos integrados más grandes que tal vez necesiten extraerse de un circuito para sustituirlos, por lo general, se fabrican en un encapsulado de soporte de chip de plástico con contactos (PLCC). La figura 4-29(d) muestra el EPM 7128SLC84 de Altera en un encapsulado PLCC, el cual es un PLD muy popular que se utiliza en muchos cursos de laboratorio. Las características clave de este chip son más terminales, un tamaño más compacto, y terminales alrededor de todo su perímetro. Observe que la terminal 1 no está "en la esquina" como en el DIP, sino en medio de la parte superior del encapsulado.

CIs bipolares y unipolares

Los CIs digitales también pueden clasificarse de acuerdo con el tipo principal de componente electrónico utilizado en sus circuitos. Los *CIs bipolares* se fabrican mediante el uso del transistor de unión bipolar (NPN y PNP) como elemento principal del circuito. Los *CIs unipolares* utilizan el transistor unipolar de efecto de campo (MOSFETs de canal P y N) como su elemento principal.

La familia **lógica de transistor/transistor (TTL)** ha sido la familia principal de CIs digitales bipolares durante más de 30 años. La serie 74 estándar fue la primera serie de CIs TTL, pero ya no se utiliza en diseños nuevos debido a que fue sustituida por varias series TTL de mejor desempeño, aunque su arreglo básico de circuitos forma la base para todos los CIs de las series TTL. Este arreglo de circuitos se muestra en la figura 4-30(a) para el INVERSOR TTL estándar. Observe que el circuito contiene varios transistores bipolares como elementos principales del circuito.

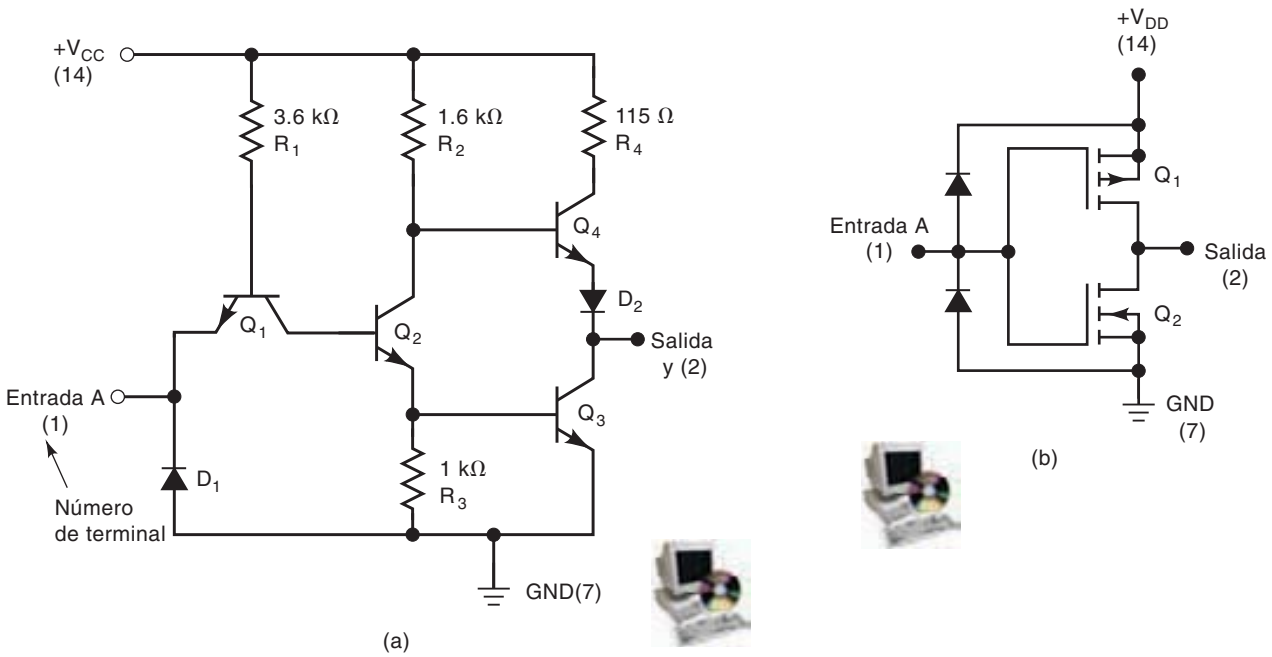


FIGURA 4-30 (a) Circuito INVERSOR TTL; (b) Circuito INVERSOR CMOS. Los números de terminales se muestran entre paréntesis.

La familia TTL fue la familia líder de CIs en las categorías SSI y MSI hasta hace aproximadamente 12 años. Desde entonces la familia de **semiconductor metal-óxido complementario (CMOS)** ha desplazado gradualmente a la familia TTL de esa posición. La familia CMOS pertenece a la clase de CIs digitales unipolares, ya que utiliza MOSFETs de canal P y N como elementos principales del circuito. La figura 4-30(b) es un circuito INVERSOR CMOS estándar. Si comparamos los circuitos TTL y CMOS de la figura 4-30, es evidente que la versión CMOS utiliza menos componentes. Ésta es una de las principales ventajas de CMOS en comparación con TTL.

Debido a la simpleza y tamaño reducido, así como de otros atributos superiores de los circuitos CMOS, los CIs modernos a gran escala se fabrican utilizando en su mayor parte la tecnología CMOS. Los cursos de laboratorio que utilizan dispositivos SSI y MSI a menudo utilizan TTL debido a su resistencia, aunque algunos utilizan CMOS también. En el capítulo 8 veremos un estudio detallado de los circuitos y las características de TTL y CMOS. Por ahora necesitamos conocer sólo algunas de sus características básicas para poder hablar sobre el diagnóstico de fallas en los circuitos combinacionales simples.

Familia TTL

La familia lógica TTL consiste en realidad de varias subfamilias o series. La tabla 4-6 lista el nombre de cada serie TTL, junto con la designación de prefijo que se utiliza para identificar los distintos CIs como parte de esa serie. Por ejemplo, los CIs que forman parte de la serie TTL estándar tienen un número de identificación que comienza con 74. Los CIs 7402, 7438 y 74123 son de esta serie. De igual forma, los CIs que forman parte de la serie TTL Schottky de bajo consumo de energía tienen un número de identificación que comienza con 74LS. Los CIs 74LS02, 74LS38 y 74LS123 son ejemplos de dispositivos de ésta.

TABLA 4-6 Varias series dentro de la familia lógica TTL.

Serie TTL	Prefijo	CI de ejemplo
TTL estándar	74	7404 (INVERSOR hex)
TTL Schottky	74S	74S04 (INVERSOR hex)
TTL Schottky de bajo consumo de energía	74LS	74LS04 (INVERSOR hex)
TTL Schottky avanzado	74AS	74AS04 (INVERSOR hex)
TTL Schottky avanzado de bajo consumo de energía	74ALS	74ALS04 (INVERSOR hex)

Las principales diferencias en las diversas series TTL tienen que ver con sus características eléctricas tales como la disipación de potencia y la velocidad de conmutación. No difieren en cuanto a la distribución de las terminales o las operaciones lógicas realizadas por los circuitos en el chip. Por ejemplo, los CIs 7404, 74S04, 74LS04, 74AS04 y 74ALS04 son circuitos de INVERSORES, cada uno de los cuales contiene *seis* INVERSORES en un solo chip.

Familia CMOS

Actualmente existen varias series CMOS, algunas de ellas se listan en la tabla 4-7. La serie 4000 es la más antigua. Esta serie contiene muchas de las mismas funciones lógicas que la familia TTL, pero no se diseñó para ser *compatible con las terminales* de los dispositivos TTL. Por ejemplo, el chip NOR cuádruple 4001 contiene cuatro compuertas NOR de dos entradas, al igual que el chip 7402 TTL, pero las entradas y las salidas de las compuertas en el chip CMOS no tienen los mismos números de terminales que las señales correspondientes en el chip TTL.

Las series CMOS 74C, 74HC, 74HCT, 74AC y 74ACT son más recientes. Las primeras tres son compatibles con las terminales de los dispositivos TTL con numeraciones correspondientes. Por ejemplo, los dispositivos 74C02, 74HC02 y 74HCT02 tienen la misma distribución de terminales que el 7402, 74LS02, y así en lo sucesivo. Las series 74HC y 74HCT operan a una mayor velocidad que los dispositivos 74C. La serie 74HCT está diseñada para ser *eléctricamente compatible* con los dispositivos TTL; esto es, un circuito integrado 74HCT puede conectarse en forma directa con los dispositivos TTL sin necesidad de circuitos que actúen como interfaz. Las series 74AC y 74ACT son CIs de desempeño avanzado. Ninguno es compatible con las terminales de los circuitos TTL. Los dispositivos 74ACT son eléctricamente compatibles con los circuitos TTL. En el capítulo 8 exploraremos las diversas series TTL y CMOS con más detalle.

TABLA 4-7 Varias series dentro de la familia lógica CMOS.

Serie CMOS	Prefijo	CI de ejemplo
CMOS de compuerta de metal	40	4001 (cuatro compuertas NOR)
Compuerta de metal, compatible con las terminales de TTL	74C	74C02 (cuatro compuertas NOR)
Compuerta de silicio de alta velocidad, compatible con las terminales TTL	74HC	74HC02 (cuatro compuertas NOR)
Compuerta de silicio de alta velocidad, compatible con las terminales y eléctricamente compatible con la familia TTL	74HCT	74HCT02 (cuatro compuertas NOR)
CMOS con desempeño avanzado, no es compatible con las terminales ni es eléctricamente compatible con la familia TTL	74AC	74AC02 (cuatro compuertas NOR)
CMOS con desempeño avanzado, no es compatible con las terminales de TTL, pero es eléctricamente compatible	74ACT	74ACT02 (cuatro compuertas NOR)

Alimentación y tierra

Para usar CIs digitales es necesario realizar las conexiones apropiadas a las terminales del CI. Las conexiones más importantes son: *alimentación de corriente directa(cd)* y *tierra*. Estas conexiones son requeridas para que los circuitos en el chip operen en forma correcta. En la figura 4-30 podemos ver que tanto el circuito TTL como CMOS tienen un voltaje de alimentación de *cd* conectado a una de sus terminales, y tierra en la otra. La terminal de voltaje de alimentación se etiqueta como V_{CC} para el circuito TTL y como V_{DD} para el circuito CMOS. Muchos de los circuitos integrados CMOS más recientes que están diseñados para ser compatibles con los circuitos integrados TTL también utilizan la designación V_{CC} en dicha terminal.

Si el CI no se conecta al voltaje de alimentación o a tierra, las compuertas lógicas en el chip no responderán en forma apropiada a las entradas lógicas y las compuertas no producirán los niveles lógicos de salida esperados.

Intervalos de voltaje de niveles lógicos

Para los dispositivos TTL, el valor nominal de V_{CC} es +5 V. Para los circuitos integrados CMOS, V_{DD} puede variar de +3 a +18 V, aunque el valor más común es +5 V cuando los circuitos CMOS se utilizan en la misma placa con circuitos integrados TTL.

Para los dispositivos TTL estándar, los intervalos de voltaje de entrada aceptables para los niveles de 0 lógico y de 1 lógico se definen como muestra la figura 4-31(a). Un 0 lógico es cualquier voltaje en el intervalo de 0 a 0.8 V; un 1 lógico es cualquier voltaje de 2 a 5 V. Los voltajes que no se encuentran en ninguno de estos intervalos se consideran como **indeterminados** y no deben utilizarse como entradas para un dispositivo TTL. Los fabricantes de circuitos integrados no pueden garantizar la manera en que responderá un circuito TTL a los niveles de entrada que se encuentren en el intervalo indeterminado (entre 0.8 y 2.0 V).

La figura 4-31(b) muestra los intervalos de voltaje de entrada lógicos para los circuitos integrados CMOS que operan con $V_{DD} = +5$ V. Los voltajes entre 0 y 1.5 V se definen como un 0 lógico, y los voltajes desde 3.5 hasta 5 V se definen como un 1 lógico. El intervalo indeterminado comprende los voltajes entre 1.5 y 3.5 V.

Entradas desconectadas (flotantes)

¿Qué ocurre cuando la entrada a un circuito integrado digital se deja desconectada? Por lo general, a una entrada desconectada se le conoce como entrada **flotante**. La respuesta a esta pregunta será distinta para TTL y para CMOS.

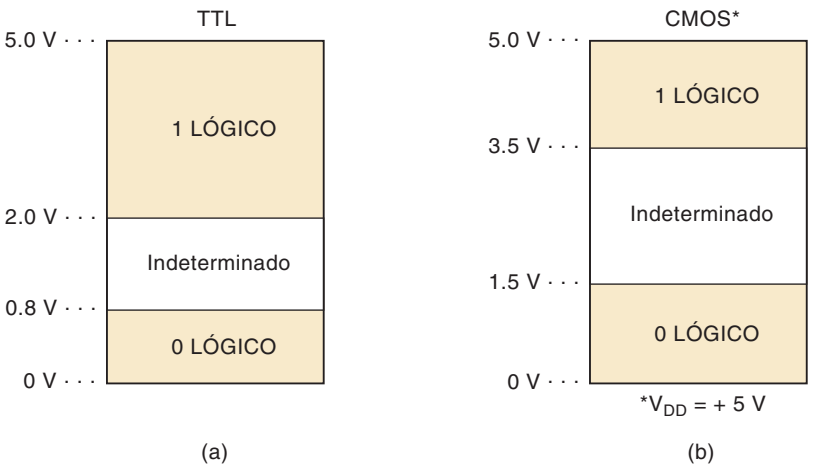


FIGURA 4-31
Intervalos de voltajes de entrada de los niveles lógicos para los CIs digitales (a) TTL y (b) CMOS.

Una entrada TTL flotante actúa justo igual que un 1 lógico. En otras palabras, el CI responderá como si se le hubiera aplicado un nivel lógico ALTO. Esta característica se utiliza a menudo cuando se prueba un circuito TTL. Un técnico descuidado podría dejar ciertas entradas desconectadas en vez de conectarlas a un nivel lógico ALTO. Aunque esto es "lógicamente" correcto no es una práctica recomendada, en especial cuando se trata de diseños de circuitos finales, ya que la entrada TTL flotante es muy susceptible de recoger señales de ruido que podrían afectar en forma adversa la operación del dispositivo.

En algunas compuertas TTL, una entrada flotante puede indicar un nivel de corriente directa de entre 1.4 y 1.8 V, si se comprueba con un voltímetro o un osciloscopio. Aun y cuando estos valores se encuentran en el intervalo indeterminado para TTL, producirá la misma respuesta que un 1 lógico. Es importante tener en cuenta esta característica de una entrada TTL flotante al diagnosticar fallas en circuitos TTL ya que puede ser de gran ayuda.

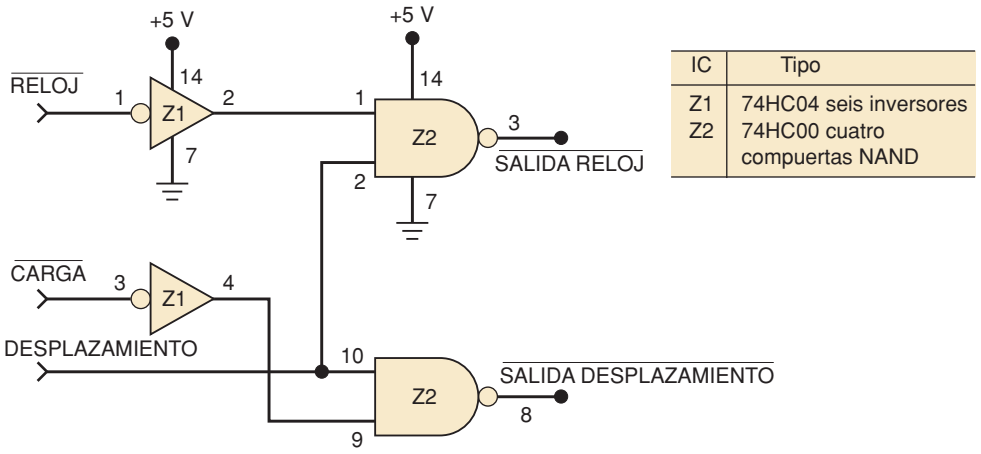
Si una entrada CMOS se deja flotante, pueden producirse resultados desastrosos. El CI podría sobrecalentarse y hasta dañarse. Por esta razón, todas las entradas de un circuito integrado CMOS deben conectarse a un nivel ALTO o BAJO, o a la salida de otro CI. Una entrada CMOS flotante no se medirá como un voltaje específico de corriente directa, sino que fluctuará en forma aleatoria a medida que recoja ruido. Por ende, no actúa como 1 ni como 0 lógico y su efecto sobre la salida es impredecible. Algunas veces la salida oscilará como resultado del ruido que recoja la salida flotante.

Muchos de los CIs CMOS más complejos tienen circuitos integrados en las entradas, los cuales reducen la probabilidad de cualquier reacción destructiva para una entrada abierta. Con estos circuitos, a la hora de experimentar con ellos, no es necesario aterrizar cada una de las terminales que no se utilicen en estos CIs grandes. No obstante, es una buena práctica conectar las entradas no utilizadas al nivel ALTO o BAJO (lo que sea apropiado) en la implementación del circuito final.

Diagramas de conexiones de circuitos lógicos

Un diagrama de conexiones muestra *todas* las conexiones eléctricas, números de terminal, números de CI, valores de los componentes, nombres de las señales y voltajes de alimentación. La figura 4-32 muestra un diagrama de conexiones común para un circuito lógico simple. Examínelo con cuidado y observe los siguientes puntos importantes:

1. El circuito utiliza compuertas lógicas de dos CIs distintos. Los dos INVERSORES forman parte de un chip 74HC04, designado como Z1. El 74HC04 contiene seis INVERSORES; dos de ellos se utilizan en este circuito y cada uno está etiquetado como parte del chip Z1. De manera similar, las dos compuertas NAND son parte de un chip 74HC00 que contiene cuatro compuertas NAND. Todas



IC	Tipo
Z1	74HC04 seis inversores
Z2	74HC00 cuatro compuertas NAND

FIGURA 4-32
Diagrama de conexiones de un circuito lógico común.

las compuertas en este chip están designadas con la etiqueta Z2. Al numerar cada compuerta como Z1, Z2, Z3 etcétera, podemos llevar el registro de cuál compuerta forma parte de cada chip. Esto es muy útil en los circuitos más complejos que contienen muchos CIs con varias compuertas por chip.

2. El número de terminal de las entradas y salidas de cada compuerta se indica en el diagrama. Estos números de terminales y las etiquetas de los CIs se utilizan para facilitar la referencia a cualquier punto en el circuito. Por ejemplo, la terminal 2 de Z1 se refiere a la terminal de salida del INVERSOR superior. De manera similar, podemos decir que la terminal 4 de Z1 está conectada a la terminal 9 de Z2.
3. Las conexiones de alimentación y de tierra para cada CI (no para cada compuerta) se muestran en el diagrama. Por ejemplo, la terminal 14 de Z1 está conectada a +5 V, y la terminal 7 de Z1 está conectada a tierra. Estas conexiones proveen de energía a los seis INVERSORES que forman parte de Z1.
4. Para el circuito mostrado en la figura 4-32, las señales que son entradas están a la izquierda. Las señales que son salidas están a la derecha. La barra sobre el nombre de la señal indica que ésta es activa en BAJO. Las burbujas se colocan también en los símbolos del diagrama para indicar el estado activo en BAJO. En este caso, es obvio que cada señal es un solo bit.
5. Las señales se definen en forma gráfica en la figura 4-32 como entradas y salidas, y la relación entre ellas (la operación del circuito) se describe en forma gráfica mediante el uso de símbolos lógicos interconectados.

Por lo general, los fabricantes de equipo electrónico suministran diagramas esquemáticos detallados que utilizan un formato similar al de la figura 4-32. Estos diagramas de conexiones son virtualmente necesarios cuando se diagnostican fallas en un circuito defectuoso. Hemos optado por identificar los CIs individuales como Z1, Z2, Z3, etcétera. Otras designaciones que se utilizan con frecuencia son IC1, IC2, IC3, … o U1, U2, U3, ….

Para dibujar circuitos lógicos pueden utilizarse computadoras personales con software para diagramas esquemáticos. Las aplicaciones computacionales que pueden interpretar estos símbolos gráficos y las conexiones de las señales pueden traducirlos en relaciones lógicas, a las cuales por lo común se les conoce como herramientas de captura de diagramas esquemáticos. El sistema de desarrollo MAX +PLUS de Altera para lógica programable permite al usuario introducir archivos de diseño gráfico (.gdf) mediante el uso de técnicas de captura de diagramas esquemáticos. Por ende, el diseño del circuito es tan sencillo como dibujar el diagrama esquemático en la pantalla de la computadora. Observe que en la figura 4-33 no hay números de terminales ni designaciones de chips en los símbolos lógicos. Los circuitos no se implementarán mediante el uso de chips SSI o MSI, sino que la funcionalidad de la lógica equivalente se “programará” en un PLD. Más adelante en este capítulo explicaremos esto con más detalle.

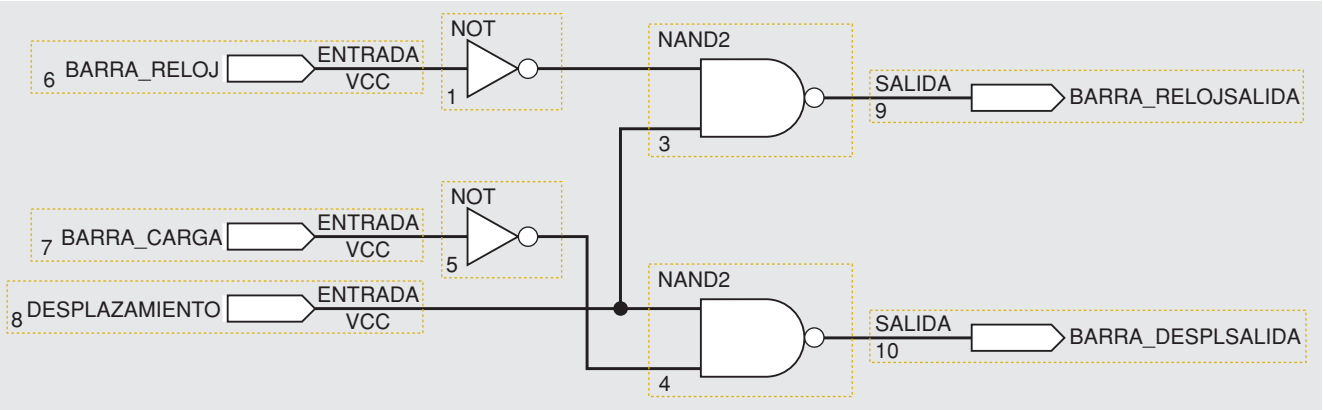


FIGURA 4-33 Diagrama lógico que utiliza captura de diagramas esquemáticos.

PREGUNTAS DE REPASO

1. ¿Cuál es el tipo más común de encapsulado de CI digital?
2. Nombre las seis categorías comunes de CIs digitales, de acuerdo con su complejidad.
3. *Verdadero o falso:* un chip 74S74 contiene la misma lógica y distribución de terminales que el 74LS74.
4. *Verdadero o falso:* un chip 74HC74 contiene la misma lógica y distribución de terminales que el 74AS74.
5. ¿Cuál serie CMOS no es compatible con las terminales de TTL?
6. ¿Cuál es el intervalo de voltaje de entrada aceptable de un 0 lógico para TTL? ¿Para un 1 lógico?
7. Repita la pregunta 6 para un CMOS que opera a $V_{DD} = 5$ V.
8. ¿Cómo responde un circuito integrado TTL a una entrada flotante?
9. ¿Cómo responde un circuito integrado CMOS a una entrada flotante?
10. ¿Cuál serie CMOS puede conectarse en forma directa a un TTL sin necesidad de circuitos de interfaz?
11. ¿Cuál es el propósito de los números de terminales en el diagrama de conexiones de un circuito lógico?
12. ¿Cuáles son las similitudes clave de los archivos de diseño gráfico que se utilizan para los diagramas de conexiones de circuitos lógicos tradicionales y de lógica programable?

4-10 DIAGNÓSTICO DE FALLAS DE SISTEMAS DIGITALES

Existen tres pasos básicos para corregir un circuito o sistema digital que tenga una falla:

1. *Detección de fallas.* Observe la operación del circuito/sistema y compárela con la operación correcta esperada.
2. *Aislamiento de fallas.* Realice pruebas y mediciones para aislar la falla.
3. *Corrección de fallas.* Sustituya el componente defectuoso, repare la conexión defectuosa, elimine el corto, o realice la acción pertinente.

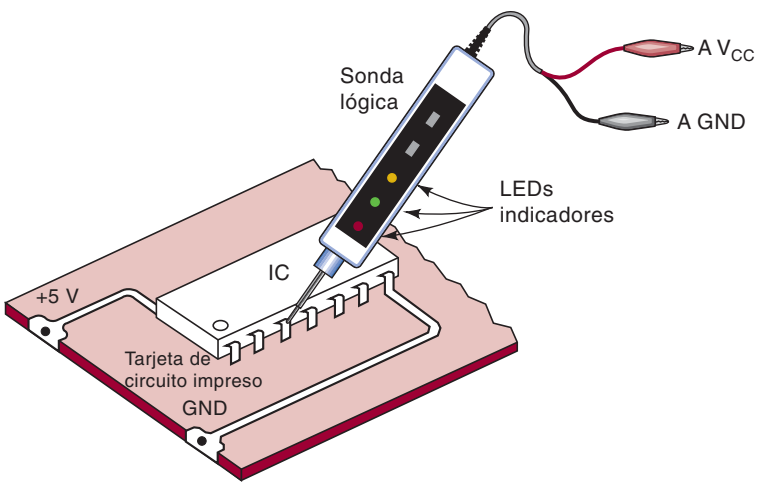
Aunque estos pasos pueden parecer bastante obvios y simples, el procedimiento real de diagnóstico de fallas que se siga dependerá en gran parte del tipo y la com-

plejidad del circuito, y de los tipos de herramientas de diagnóstico de fallas y de la documentación disponible.

Las buenas técnicas de diagnóstico de fallas sólo pueden aprenderse en un entorno de laboratorio, por medio de la experimentación y el diagnóstico de fallas real en circuitos y sistemas defectuosos. No existe en absoluto una forma de convertirse en un técnico de diagnóstico de fallas eficiente que tratar de diagnosticar fallas en todos los circuitos que sea posible; además, por más libros de texto que lea no podrá obtener de ellos ese tipo de experiencia. No obstante, podemos ayudarle a desarrollar las habilidades analíticas que forman la parte más esencial de una técnica eficiente de diagnóstico de fallas. Describiremos los tipos de fallas comunes en los sistemas compuestos en su mayor parte de CIs digitales y le indicaremos cómo reconocerlas. Después le presentaremos ejemplos prácticos triviales para ilustrar los procesos analíticos implicados en el diagnóstico de fallas. Además, habrá problemas al final del capítulo para que usted tenga la oportunidad de pasar por estos procesos analíticos para obtener sus propias conclusiones acerca de los circuitos digitales defectuosos.

Para los análisis y ejercicios sobre diagnóstico de fallas que realizaremos en este libro, supondremos que el técnico de diagnóstico de fallas tiene a su disposición las herramientas básicas para este propósito: *sonda lógica, osciloscopio* y *generador de pulsos lógico*. Desde luego que la herramienta de diagnóstico de fallas más importante y efectiva es el técnico, y ésa es la herramienta que esperamos desarrollar mediante la presentación de los principios y las técnicas de diagnóstico de fallas, ejemplos y problemas, tanto aquí como en los capítulos posteriores.

En las siguientes tres secciones sobre diagnóstico de fallas utilizaremos sólo nuestro cerebro y una **sonda lógica** tal como la que se muestra en la figura 4-34. La sonda lógica tiene una punta de metal afilada con la que se toca el punto específico que deseamos probar. En la figura 4-34 se muestra probando la terminal 3 de un CI. También puede tocar el trazo de una tarjeta de circuitos impresos, un alambre sin aislamiento, la terminal de un conector, la terminal de un componente discreto tal como un transistor o cualquier otro punto conductor en un circuito. El nivel lógico presente en la punta de la sonda se indicará mediante el estado de sus LEDs indicadores. En la tabla de la figura 4-34 se muestran las cuatro posibilidades. Un nivel lógico *indeterminado* no produce luz en los indicadores. Esto incluye la condición en la que la punta de la sonda toca un punto en un circuito que esté abierto o flotante; esto es, que no esté conectado a ninguna fuente de voltaje. Este tipo de sonda también cuenta con un LED amarillo para indicar la presencia de un tren de pulsos. Cualquier transición (de BAJO a ALTO o de ALTO a BAJO) provocará que el LED amarillo destelle durante una fracción de segundo y después se apague. Si las transiciones ocurren con frecuencia, el LED continuará destellando a una frecuencia



LEDs			
Rojo	Verde	Amarillo	Condición lógica
APAG	ENC	APAG	BAJO
ENC	APAG	APAG	ALTO
APAG	APAG	APAG	INDETERMINADO*
X	X	DESTELLANDO	PULSANDO

* Incluye condición abierta o flotante

FIGURA 4-34 Una sonda lógica se utiliza para monitorear la actividad de los niveles lógicos en la terminal de un CI o en cualquier punto accesible en un circuito lógico.

aproximada de 3 Hz. Mediante la observación de los LEDs rojo y verde, junto con el LED amarillo destellante, podrá saber si la señal está la mayor parte del tiempo en ALTO o en BAJO.

4-11 FALLAS INTERNAS EN LOS CIRCUITOS INTEGRADOS DIGITALES

Las fallas internas más comunes en los CIs digitales son:

1. Fallas en los circuitos internos.
2. Entradas o salidas cortocircuitadas a tierra o a V_{CC}.
3. Entradas o salidas sin conectar (circuito abierto).
4. Corto entre dos terminales (que no sean tierra ni V_{CC}).

Ahora describiremos cada uno de estos tipos de fallas.

Fallas en los circuitos internos

Por lo general, estas fallas son ocasionadas debido a que uno de los componentes internos falla por completo u opera fuera de sus especificaciones. Cuando esto ocurre, las salidas del CI no responden de manera apropiada a las entradas. No hay forma de predecir lo que harán las salidas, ya que esto depende del componente interno que esté fallando. Algunos ejemplos de este tipo de fallas serían un corto tipo base-emisor en el transistor Q_4 o un valor de resistencia demasiado grande para R_2 en el INVERSOR TTL de la figura 4-30(a). Este tipo de falla interna del CI no es tan común como las otras tres.

Entrada internamente cortocircuitada a tierra o a la fuente de alimentación

Este tipo de falla interna ocasionará que la entrada de un CI se quede atascada en el estado BAJO o ALTO. La figura 4-35(a) muestra la terminal de entrada 2 de una compuerta NAND cortocircuitada a tierra dentro del CI. Esto hará que la terminal 2 siempre esté en el estado BAJO. Si esta terminal de entrada se excita mediante una señal lógica B, sin duda aterrizará B a tierra. Por ende, este tipo de falla afectará la salida del dispositivo que está generando la señal B.

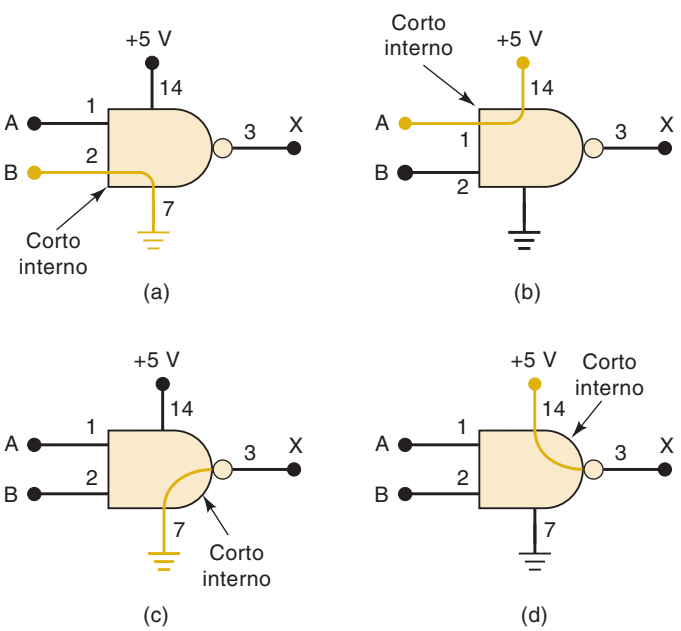


FIGURA 4-35
(a) Entrada de un CI cortocircuitada a tierra en forma interna; (b) entrada de un CI en corto con la fuente de voltaje, de manera interna. Estos dos tipos de fallas obligan a que la señal de entrada en la terminal cortocircuitada se quede en el mismo estado. (c) Salida del CI cortocircuitada a tierra de manera interna; (d) la salida en corto con la fuente de voltaje, de manera interna. Estas dos fallas no afectan a las señales en las entradas del CI.

De manera similar, la terminal de entrada de un CI podría estar en corto interno con +5 V, como en la figura 4-35(b). Esto mantendría a esa terminal atascada en el estado ALTO. Si esta terminal de entrada es excitada por una señal lógica *A*, sin duda pondría en corto a *A* con +5 V.

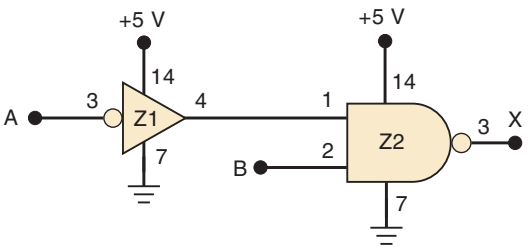
Salida internamente cortocircuitada a tierra o a la fuente de alimentación

Este tipo de falla interna provocará que la terminal de salida se quede atascada en el estado BAJO o ALTO. La figura 4-35(c) muestra a la terminal 3 de la compuerta NAND cortocircuitada a tierra dentro del CI. Esta salida se queda atascada en BAJO y no responderá a las condiciones que se apliquen a las terminales de entrada 1 y 2. En otras palabras, las entradas lógicas *A* y *B* no tendrán efecto sobre la salida *X*.

La terminal de salida de un CI también puede quedar en corto con +5 V dentro del CI, como se muestra en la figura 4-35(d). Esto obliga a que la terminal de salida 3 se quede atascada en ALTO, sin importar el estado de las señales en las terminales de entrada. Observe que este tipo de falla no tiene efecto sobre las señales lógicas en las entradas del CI.

EJEMPLO 4-24

Consulte el circuito de la figura 4-36. Un técnico utiliza una sonda lógica para determinar las condiciones de las diversas terminales del CI. Los resultados están registrados en la figura. Examine estos resultados y determine si el circuito está trabajando en forma apropiada. En caso contrario, sugiera algunas de las posibles fallas.



Terminal	Condición
Z1-3	Pulsando
Z1-4	BAJO
Z2-1	BAJO
Z2-2	ALTO
Z2-3	ALTO

FIGURA 4-36 Ejemplo 4-24.

Solución

La terminal de salida 4 del INVERSOR debe estar pulsando, ya que su entrada está pulsando. No obstante, los resultados registrados muestran que la terminal 4 está atascada en BAJO. Como esta terminal está conectada a la terminal 1 de Z2, la salida de la compuerta NAND se mantiene en ALTO. De nuestra discusión anterior podemos listar tres posibles fallas que podrían producir esta operación.

En primer lugar, podría existir una falla en un componente interno en el INVERSOR que no le permita responder en forma apropiada a su entrada. En segundo lugar, la terminal 4 del INVERSOR podría estar cortocircuitada a tierra en forma interna para Z1, con lo cual se mantendría en BAJO. En tercer lugar, la terminal 1 de Z2 podría estar cortocircuitada a tierra en forma interna para Z2. Esto evitaría que cambiara la terminal de salida del INVERSOR.

Además de estas posibles fallas, puede haber cortos externos a tierra en cualquier parte de la ruta conductora entre la terminal 4 de Z1 y la terminal 1 de Z2. En uno de los siguientes ejemplos veremos cómo aislar la verdadera falla.

Entrada o salida sin conectar (circuito abierto)

Algunas veces el alambre conductor tan fino que conecta la terminal de un CI con los circuitos internos del mismo se rompe, lo cual produce un circuito abierto. La figura 4-37 del ejemplo 4-25 muestra esta situación para una entrada (terminal 13) y una salida (terminal 6). Si se aplica una señal a la terminal 13, no llegará a la

entrada de la compuerta NAND-1 y, por lo tanto, no tendrá efecto sobre su salida. La entrada abierta de la compuerta estará en el estado flotante. Como dijimos antes, los dispositivos TTL responderán como si esta entrada flotante fuera un 1 lógico y los dispositivos CMOS responderán en forma errática, y tal vez podrían dañarse debido al sobrecalentamiento.

La abertura en la salida de la compuerta NAND-4 evita que la señal llegue a la terminal 6 del IC, por lo que no habrá un voltaje estable presente en esa terminal. Si esta terminal se conecta a la entrada de otro CI, producirá una condición flotante en esa entrada.

EJEMPLO 4-25

¿Qué indicaría una sonda lógica en la terminal 13 y en la terminal 6 de la figura 4-37?

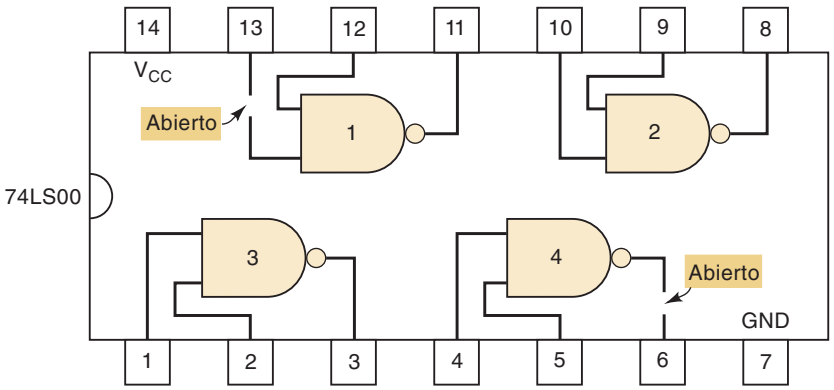


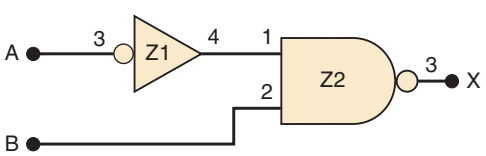
FIGURA 4-37 Un CI con una entrada abierta en forma interna no responderá a las señales que se apliquen a esa terminal de entrada. Una salida abierta en forma interna producirá un voltaje impredecible en esa terminal de salida.

Solución

En la terminal 13, la sonda lógica indicará el nivel lógico de la señal externa que se conecta a la terminal 13 (la cual no se muestra en este diagrama). En la terminal 6, la sonda lógica no tendrá ningún LED encendido para un nivel lógico indeterminado, ya que el nivel de salida de la compuerta NAND nunca llegará a la terminal 6.

EJEMPLO 4-26

Consulte el circuito de la figura 4-38 y las indicaciones registradas de la sonda lógica. ¿Cuáles son algunas de las posibles fallas que producirían esos resultados? Suponga que los CIs son TTL.



Nota: no se muestran las conexiones a V_{CC} ni a tierra de los CI

Terminal	Condición
Z1-3	ALTO
Z1-4	BAJO
Z2-1	BAJO
Z2-2	Pulsando
Z2-3	Pulsando

FIGURA 4-38 Ejemplo 4-26.

Solución

Un análisis de los resultados registrados nos indica que el INVERSOR parece estar funcionando en forma correcta, pero la salida de la compuerta NAND es inconsistente con sus entradas. La salida NAND debería estar en ALTO, ya que su terminal de entrada 1 está en BAJO. Este nivel BAJO debería evitar que la compuerta NAND respondiera a los pulsos en la terminal 2. Es probable que este nivel BAJO no esté llegando a los circuitos internos de la compuerta NAND debido a una abertura interna.

Como el CI es TTL, este circuito abierto produciría el mismo efecto que un nivel ALTO lógico en la terminal 1. Si el CI hubiera sido CMOS, el circuito abierto interno en la terminal 1 podría haber generado una salida indeterminada, lo cual podría provocar un sobrecalentamiento y la destrucción del chip.

De lo antes expuesto en relación con las entradas TTL abiertas, usted podría haber esperado que el voltaje de la terminal 1 de Z2 estuviera entre 1.4 y 1.8 V, y que la sonda lógica debería haberlo registrado como indeterminado. Esto hubiera sido cierto si el circuito abierto fuera *externo* para el chip NAND. Como no hay circuito abierto entre la terminal 4 de Z1 y la terminal 1 de Z2, el voltaje en la terminal 4 de Z1 sí llega a la terminal 1 de Z2, pero se desconecta *dentro* del chip NAND.

Corto entre dos terminales

Un corto interno entre dos terminales del CI obligará a que las señales lógicas en esas terminales siempre sean idénticas. Cada vez que dos señales que se supone deben ser distintas muestran las mismas variaciones de niveles lógicos, existe una buena posibilidad de que las señales estén en corto.

Considere el circuito de la figura 4-39, en donde las terminales 5 y 6 de la compuerta NOR están en corto de manera interna. Este corto hace que las dos terminales de salida del INVERSOR se conecten entre sí, de manera que las señales en la terminal 2 de Z1 y la terminal 4 de Z1 deben ser idénticas, aún y cuando las dos señales de entrada del INVERSOR están tratando de producir diferentes salidas. Para ilustrar esto, considere las formas de onda de entrada que se muestran en el diagrama. Aún cuando estas formas de onda de entrada son distintas, las salidas Z1-2 y Z1-4 son iguales.

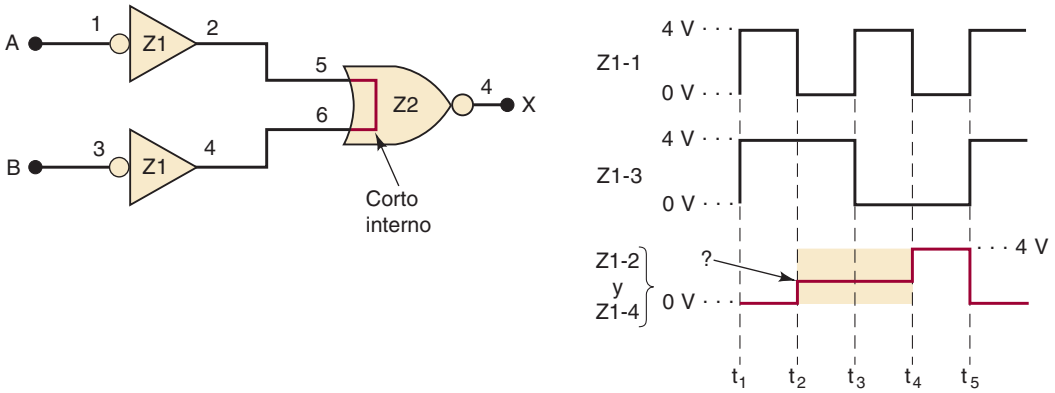


FIGURA 4-39 Cuando se ponen en corto dos terminales de entrada en forma interna, se obliga a las señales que excitan estas terminales a ser idénticas y, por lo general, se produce una señal con tres niveles distintos.

Durante el intervalo de t_1 a t_2, ambos INVERSORES tienen una entrada en ALTO y están tratando de producir una salida en BAJO, por lo que estar en corto no hace ninguna diferencia. Durante el intervalo de t_4 a t_5, ambos INVERSORES tienen una entrada en BAJO y están tratando de producir una salida en ALTO, así que el estar en corto de nuevo no tiene ningún efecto. No obstante, durante los intervalos de t_2 a t_3 y de t_3 a t_4 un INVERSOR está tratando de producir una salida en ALTO, mientras que el otro está tratando de producir una salida en BAJO. A esto se le conoce como **colisión** de señales, ya que las dos señales están "luchando" una con la otra. Cuando esto ocurra, el nivel de voltaje real que aparezca en las salidas en corto dependerá de los circuitos internos del CI. En los dispositivos TTL, por lo general, será un voltaje en el extremo superior del intervalo del 0 lógico (es decir, cerca de 0.8 V), aunque también podría estar en el intervalo indeterminado. En los dispositivos CMOS, por lo general, será un voltaje en el intervalo indeterminado.

Siempre que vea una forma de onda como la señal de Z1-2, Z1-4 en la figura 4-39 con tres niveles distintos, será motivo para sospechar que dos señales de salida pueden estar en corto.

PREGUNTAS DE REPASO

1. Liste las distintas fallas internas en los circuitos integrados.
2. ¿Cuál falla interna de un CI puede producir señales que muestren tres niveles de voltaje distintos?
3. ¿Qué indicaría una sonda lógica en Z1-2 y Z1-4 de la figura 4-39, si $A = 0$ y $B = 1$?
4. ¿Qué es la colisión de señales?

4-12 FALLAS EXTERNAS

Hemos visto cómo reconocer los efectos de diversas fallas internas para los CIs digitales. Hay muchas cosas más que pueden salir mal y que son externas para los CIs; en esta sección describiremos las más comunes.

Líneas de señal abiertas

Esta categoría incluye cualquier falla que produzca una interrupción o discontinuidad en la ruta conductora, de tal forma que se evite que un nivel de voltaje o señal pase de un punto a otro. Algunas de las causas de las líneas de señal abiertas son:

1. Alambre roto.
2. Conexión soldada defectuosamente; conexión de alambre enrollado floja.
3. Grieta o interrupción en la línea de conexión de un circuito impreso (algunas de éstas son del grueso de un cabello, y se pueden ver sólo con una lupa).
4. Terminal doblada o rota en un CI.
5. Zócalo de CI defectuoso, de tal forma que el CI no haga buen contacto con el zócalo.

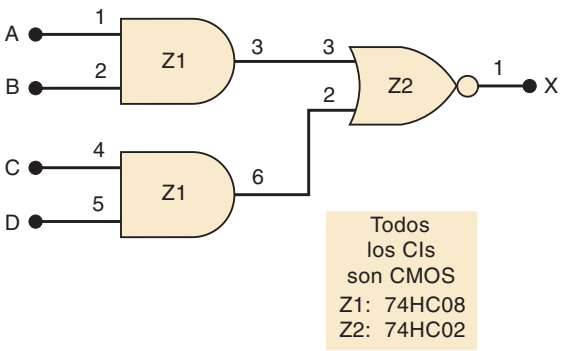
Con frecuencia, este tipo de fallas en los circuitos pueden detectarse mediante una inspección visual cuidadosa y después verificarse desconectando la alimentación del circuito y comprobando si hay continuidad (es decir, una ruta de baja resistencia) con un óhmetro entre los dos puntos en cuestión.

EJEMPLO 4-27

Considere el circuito CMOS de la figura 4-40 y las indicaciones de la sonda lógica que se incluyen. ¿Cuál es la falla más probable del circuito?

Solución

Es probable que el nivel indeterminado en la salida de la compuerta NOR se deba a la entrada indeterminada en la terminal 2. Como hay un nivel BAJO en Z1-6, también debe haber un nivel BAJO en Z2-2. Es evidente que el BAJO de Z1-6 no está llegando a Z2-2, por lo que debe haber un circuito abierto en la ruta de señal entre estos dos puntos. La ubicación de este circuito abierto puede determinarse empezando desde Z1-6 con la sonda lógica y rastreando el nivel BAJO a lo largo de la ruta de la señal hacia Z2-2, hasta que cambie a un nivel indeterminado.



Terminal	Condición
Z1-1	Pulsando
Z1-2	ALTO
Z1-3	Pulsando
Z1-4	BAJO
Z1-5	Pulsando
Z1-6	BAJO
Z2-3	Pulsando
Z2-2	Indeterminado
Z2-1	Indeterminado

FIGURA 4-40 Ejemplo 4-27.

Líneas de señal en corto

Este tipo de falla tiene el mismo efecto que un corto interno entre las terminales de un CI. Provoca que dos señales sean exactamente la misma (colisión de señales). Una línea de señal puede estar cortocircuitada a tierra o a V_{CC} en vez de estar en corto con otra línea de señal. En esos casos se obliga a que la señal permanezca en el estado BAJO o ALTO. Las principales causas de cortos inesperados entre dos puntos en un circuito son las siguientes:

1. *Alambrado mal instalado*. Un ejemplo es cuando se quita demasiado aislante de los extremos de los cables que están muy cerca uno del otro.
2. *Puentes de soldadura*. Son salpicaduras de soldadura que ponen en corto dos o más puntos. Por lo común ocurren entre puntos que están muy cercanos, como las terminales adyacentes en un chip.
3. *Desbaste incompleto*. El cobre entre las rutas conductoras adyacentes en una tarjeta de circuitos impresos no se desbasta de forma adecuada.

De nuevo, con frecuencia una cuidadosa inspección visual puede descubrir este tipo de fallas, y la comprobación con un óhmetro puede verificar que los dos puntos en el circuito están en corto.

Fuente de alimentación defectuosa

Todos los sistemas digitales tienen una o más fuentes de alimentación de corriente directa, las cuales suministran los voltajes V_{CC} y V_{DD} requeridos por los chips. Una fuente defectuosa o una sobrecargada (que suministra más de su valor nominal de corriente) proporcionará voltajes de suministro mal regulados a los CIs, y éstos no operarán u operarán en forma incorrecta.

Una fuente de alimentación puede dejar de regular el voltaje debido a una falla en sus circuitos internos, o porque los circuitos que está alimentando consumen más corriente de la que puede suministrar la fuente. Esto puede ocurrir si un chip o componente tiene una falla que hace que consuma mucha más corriente de la normal.

Una buena práctica de diagnóstico de fallas es comprobar los niveles de voltaje en cada una de las fuentes de alimentación en el sistema, para ver si se encuentran dentro de sus intervalos especificados. También es una buena idea comprobarlas en un osciloscopio para verificar que no haya una cantidad considerable de rizo de corriente alterna en los niveles de corriente directa, y para verificar que los niveles de voltaje permanezcan regulados durante la operación del sistema.

Uno de los signos más comunes de falla en la fuente de alimentación es que uno o más chips operen en forma incorrecta, o que no operen en lo absoluto. Algunos CIs son más tolerantes a las variaciones de la fuente de alimentación y pueden operar en forma apropiada, mientras que otros no. Usted siempre debe comprobar los niveles de voltaje de alimentación y de tierra en cada CI que parezca operar en forma incorrecta.

Carga de salida

Cuando un CI digital tiene su salida conectada a demasiadas entradas de un CI, se excede el valor nominal de su corriente de salida y el voltaje de salida puede caer dentro del intervalo indeterminado. A este efecto se le conoce como *cargar* la señal de salida (en realidad es sobrecargar la señal de salida) y, por lo general, es el resultado de un mal diseño o de una conexión incorrecta.

PREGUNTAS DE REPASO

1. ¿Cuáles son los tipos más comunes de fallas externas?
2. Liste algunas de las causas de circuitos abiertos en la ruta de la señal.
3. ¿Qué síntomas ocasiona una fuente de alimentación defectuosa?
4. ¿Cómo podría afectar la carga a un nivel de voltaje de salida de un CI?

4-13 EJEMPLO PRÁCTICO DE DIAGNÓSTICO DE FALLAS

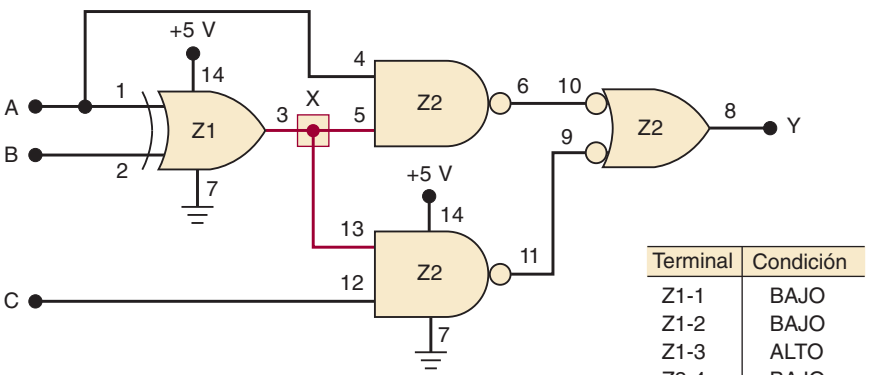
El siguiente ejemplo ilustrará los procesos analíticos involucrados en el diagnóstico de fallas en los circuitos digitales. Aunque el ejemplo es un circuito lógico combinacional bastante simple, el razonamiento y los procedimientos de diagnóstico de fallas pueden aplicarse a los circuitos digitales más complejos que encontraremos en los siguientes capítulos.

EJEMPLO 4-28

Considere el circuito de la figura 4-41. Se supone que la salida Y cambiará a ALTO en cualquiera de las siguientes condiciones:

1. $A = 1, B = 0$ sin importar el nivel en C
2. $A = 0, B = 1, C = 1$

Tal vez desee verificar estos resultados por su cuenta.



Terminal	Condición
Z1-1	BAJO
Z1-2	BAJO
Z1-3	ALTO
Z2-4	BAJO
Z2-5	ALTO
Z2-6,10	ALTO
Z2-13	ALTO
Z2-12	ALTO
Z2-9,11	BAJO
Z2-8	ALTO

FIGURA 4-41 Ejemplo 4-28.

Cuando se prueba el circuito, el técnico observa que la salida *Y* cambia a ALTO cada vez que *A* está en ALTO o que *C* está en ALTO, sin importar el nivel en *B*. Entonces toma mediciones con la sonda lógica para la condición en la que $A = B = 0$, $C = 1$ y obtiene las indicaciones registradas en la figura 4-41.

Examine los niveles registrados y liste las posibles causas del mal funcionamiento. Después desarrolle un procedimiento paso a paso para determinar la falla exacta.

Solución

Todas las salidas de la compuerta NAND son correctas para los niveles presentes en sus entradas. Sin embargo, la compuerta XOR debería producir un nivel BAJO en la terminal 3 de salida, ya que dos de sus entradas se encuentran en el mismo nivel BAJO. Parece que Z1-3 se queda en ALTO, aún y cuando sus entradas deberían producir un nivel BAJO. Hay varias causas posibles de esto:

1. Una falla en un componente interno en Z1 que evite que su salida cambie a BAJO.
2. Un corto externo con V_{CC} desde cualquier punto a lo largo de los conductores conectados al nodo *X* (sombreado en el diagrama de la figura).
3. La terminal 3 de Z1 en corto interno con V_{CC}.
4. La terminal 5 de Z2 en corto interno con V_{CC}.
5. La terminal 13 de Z2 en corto interno con V_{CC}.

Todas estas posibilidades (excepto la primera) pondrán en corto directo el nodo *X* (y todas las terminales del CI que estén conectadas a él) con V_{CC}.

El siguiente procedimiento puede usarse para aislar la falla. Este procedimiento no es el único método que puede utilizarse y, como dijimos antes, el verdadero procedimiento de diagnóstico de fallas que utiliza un técnico es muy dependiente del equipo de prueba que tenga disponible.

1. Compruebe los niveles de V_{CC} y de tierra en las terminales apropiadas de Z1. Aunque es muy poco probable que la ausencia de cualquiera de estas dos señales pueda hacer que Z1-3 se quede en ALTO, es conveniente realizar esta comprobación en cualquier CI que esté produciendo una salida incorrecta.
2. Desconecte la alimentación del circuito y utilice un óhmetro para comprobar si hay un corto (resistencia menor de 1 Ω) entre el nodo *X* y cualquier punto conectado a V_{CC} (tal como Z1-14 o Z2-14). Si no se indica un corto, podemos eliminar las últimas cuatro posibilidades en nuestra lista. Esto significa que es muy probable que Z1 tenga una falla interna y deba sustituirse.
3. Si el paso 2 muestra que hay un corto del nodo *X* a V_{CC}, realice un examen visual detallado de la tarjeta del circuito y busque puentes de soldadura, lengüetas de cobre sin desbastar, alambres sin aislamiento que se toquen unos con otros, y cualquier otra causa posible de un corto externo con V_{CC}. Un punto probable para un puente de soldadura sería entre las terminales adyacentes 13 y 14 de Z2. La terminal 14 se conecta a V_{CC} y la 13 al nodo *X*. Si se encuentra un corto externo, elimínelo y realice una comprobación con un óhmetro para verificar que el nodo *X* ya no esté en corto con V_{CC}.
4. Si el paso 3 no revela un corto externo, las tres posibilidades que quedan son cortos internos con V_{CC} y Z1-3, Z2-13 o Z2-5. Una de éstas está poniendo en corto el nodo *X* con V_{CC}.

Para determinar cuál de estas terminales del CI es la culpable, debemos desconectar cada una de ellas del nodo *X* *una a la vez* y debemos volver a comprobar si hay un corto con V_{CC} después de cada desconexión. Cuando se desconecte la terminal que esté en corto interno con V_{CC}, el nodo *X* ya no estará en corto con V_{CC}.

El proceso de desconectar cada una de las terminales sospechosas del nodo *X* puede ser fácil o difícil, dependiendo de la manera en que esté construido el circuito. Si los CIs están en zócalos, todo lo que necesitamos hacer es extraer el CI de su zócalo, doblar la terminal sospechosa y volver a insertar el CI en su zócalo. Si los CIs están soldados en una tarjeta de circuitos impresos, tendrá que cortar la línea que está conectada con la terminal y reparar la línea cortada cuando termine.

Aunque es bastante simple, el ejemplo 4-28 le muestra el tipo de razonamiento que debe emplear un técnico de diagnóstico de fallas para aislar una falla. Usted tendrá la oportunidad de comenzar a desarrollar sus propias habilidades de diagnóstico de fallas al trabajar en muchos problemas de final de capítulo que hemos designado con una **F** para identificarlos como problemas de diagnóstico de fallas.

4-14 DISPOSITIVOS LÓGICOS PROGRAMABLES*

En las secciones anteriores vimos un poco acerca de las clases de CIs conocidas como dispositivos lógicos programables. En el capítulo 3 presentamos el concepto de describir la operación de un circuito mediante el uso de un lenguaje de descripción de hardware. En esta sección exploraremos aún más estos temas y nos prepararemos para utilizar las herramientas implicadas en el desarrollo y la implementación de sistemas digitales mediante el uso de PLDs. Desde luego que es imposible comprender todos los detalles complejos de cómo funciona un PLD antes de abarcar los fundamentos de los circuitos digitales. A medida que examinemos nuevos conceptos fundamentales, expandiremos nuestro conocimiento sobre los PLDs y los métodos de programación. Presentaremos este material de tal forma que cualquiera que no esté interesado en los PLDs pueda omitir sin problemas estas secciones, sin perder la continuidad en la cobertura de los principios básicos.

Vamos a repasar el proceso que tratamos antes, en relación con el diseño de los circuitos digitales combinacionales. Los dispositivos de entrada se identifican y se les asigna un nombre algebraico tal como *A*, *B*, *C* o *CARGA*, *DESPLAZAMIENTO*, *RELOJ*. De igual forma, los dispositivos de salida reciben nombres tales como X, *Z* o *SALIDA_RELOJ*, *SALIDA_DESPLAZAMIENTO*. Después se crea una tabla de verdad que lista todas las posibles combinaciones de entradas e identifica el estado requerido de las salidas bajo cada condición de entrada. La tabla de verdad es una manera de describir la forma en que va a operar el circuito. Otra manera de describir la operación del circuito es mediante una expresión booleana. A partir de este punto, el diseñador debe encontrar la relación algebraica más simple y seleccionar CIs digitales que puedan alambrarse en conjunto para implementar el circuito. Tal vez usted haya experimentado que estos últimos pasos son los más tediosos, los que consumen más tiempo y los que están más propensos a errores.

Los dispositivos lógicos programables permiten automatizar la mayoría de estos tediosos pasos mediante una computadora y *software de desarrollo* para PLDs. El uso de la lógica programable mejora la eficiencia del proceso de diseño y desarrollo. En consecuencia, la mayoría de los sistemas digitales modernos se implementan de esta forma. El trabajo del diseñador de circuitos es identificar entradas y salidas, especificar la relación lógica de la manera más conveniente y seleccionar un dispositivo programable que sea capaz de implementar el circuito al costo más bajo. El concepto detrás de los dispositivos lógicos programables es simple: poner muchas compuertas lógicas en un solo CI y controlar la interconexión de estas compuertas mediante electrónica.

Hardware de un PLD

En el capítulo 3 vimos que muchos circuitos digitales de la actualidad se implementan mediante el uso de dispositivos lógicos programables (PLDs). Estos dispositivos se configuran en forma electrónica y sus circuitos internos están "alambrados"

* Se pueden omitir todas las secciones que tratan acerca de los PLDs sin perder la continuidad en el balance de los capítulos 1-12.

entre sí mediante electrónica, para formar un circuito lógico. Este alambrado programable puede considerarse como miles de conexiones que están conectadas (1) o desconectadas (0). Es muy tedioso tratar de configurar estos dispositivos en forma manual, colocando 1s y 0s en una rejilla, por lo que la siguiente pregunta lógica es: "¿Cómo controlamos la interconexión de compuertas en un PLD por medio de electrónica?"

Un método común para conectar una de muchas señales que entran en una red a una de muchas líneas de señal que salen de la red es mediante una matriz de conmutación. Consulte la figura 3-44, en donde se introdujo este concepto. Una matriz es tan sólo una rejilla de conductores (alambres) ordenados en filas y columnas. Las señales de entrada se conectan a las columnas de la matriz y las salidas se conectan a las filas de la misma. En cada intersección de una fila y una columna hay un interruptor que puede conectar por medios electrónicos esa fila con esa columna. Los interruptores que conectan filas con columnas pueden ser interruptores mecánicos, enlaces de fusibles, interruptores electromagnéticos (relevadores) o transistores. Ésta es la estructura general que se utiliza en muchas aplicaciones, la cual exploraremos con más detalle cuando estudiemos los dispositivos de memoria en el capítulo 12.

Los PLDs también utilizan una matriz de conmutación, la cual se conoce comúnmente como arreglo programable. Al decidir cuáles intersecciones están conectadas y cuáles no, podemos "programar" la forma en que las entradas se conectan a las salidas del arreglo. En la figura 4-42 se utiliza un arreglo programable para seleccionar las entradas para cada compuerta AND. Observe que en esta simple matriz podemos producir cualquier combinación de productos lógicos de las variables *A*, *B* en cualquiera de las salidas de la compuerta AND. Una matriz o arreglo programable tal como la que se muestra en la figura también puede utilizarse para conectar las salidas de la compuerta AND con compuertas OR. En el capítulo 13 veremos con detalle todo lo relacionado con varias arquitecturas de los PLDs.

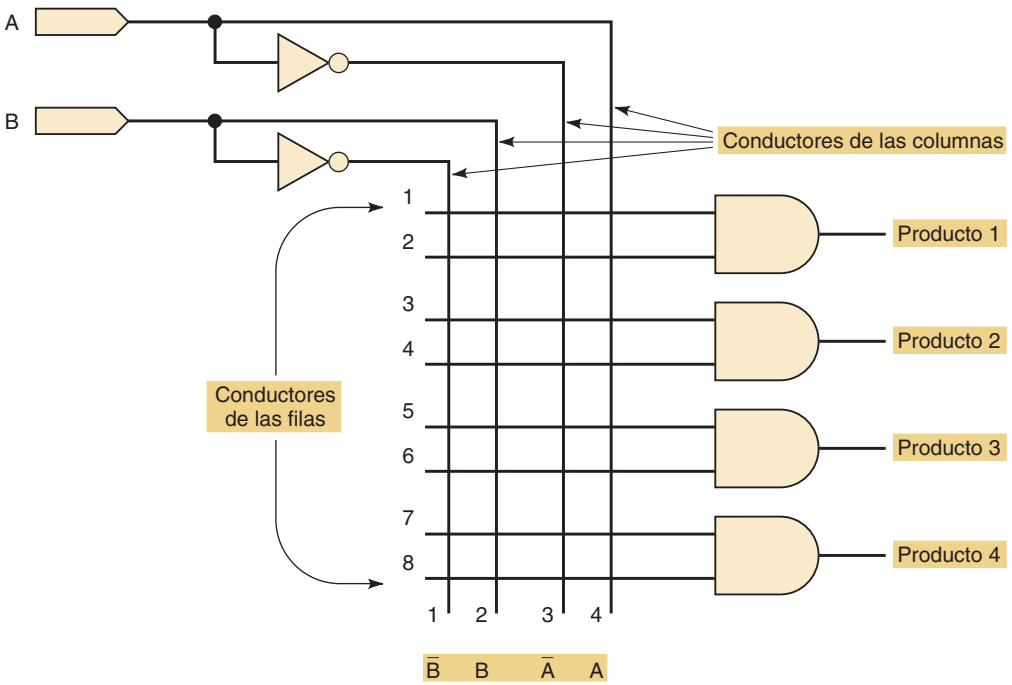


FIGURA 4-42 Un arreglo programable para seleccionar entradas como términos de productos.

Programación de un PLD

Existen dos maneras de "programar" un CI de PLD. Programar significa realizar las conexiones reales en el arreglo. En otras palabras, significa determinar cuáles

de esas conexiones se supone deben estar abiertas (0) y cuáles se supone deben estar cerradas (1). El primer método implica remover el chip de CI del PLD de su tarjeta de circuito. Después el chip se coloca en un dispositivo especial conocido como **programador**, el cual se muestra en la figura 4-43. La mayoría de los programadores modernos se conectan a una computadora personal que ejecuta software que contiene bibliotecas de información acerca de los diversos tipos de dispositivos programables disponibles.

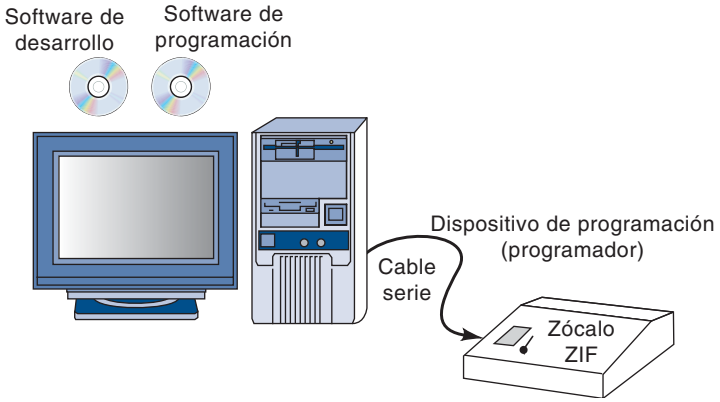


FIGURA 4-43 Un sistema de desarrollo de PLDs.

El software de programación se invoca (se llama y se ejecuta) en la PC para establecer la comunicación con el programador. Este software permite al usuario configurar el programador para el tipo de dispositivo que se va a programar, comprobar si el dispositivo está en blanco, leer el estado de cualquier conexión programable en el dispositivo y proveer las instrucciones para que el usuario programe un chip. En última instancia, la pieza se coloca en un zócalo especial que nos permite insertar el chip y después sujetar los contactos en las terminales. A éste se le conoce como zócalo de **cero esfuerzo de inserción (ZIF)**. Diversos fabricantes ofrecen los *programadores universales* que pueden manejar cualquier tipo de dispositivo programable.

Por fortuna, y a medida que las piezas programables empezaron a proliferar, los fabricantes vieron la necesidad de estandarizar las asignaciones de terminales y los métodos de programación. Como resultado, se formó el Consejo Común de Ingeniería de Dispositivos Electrónicos (**JEDEC**). Uno de los resultados fue el estándar 3 de JEDEC, un formato para transferir datos de programación para PLDs, independiente del fabricante del PLD o del software de programación. También se estandarizaron las asignaciones de terminales para varios encapsulados de CIs, con lo cual los programadores universales se hicieron menos complicados. En consecuencia, los dispositivos de programación pueden programar muchos tipos de PLDs. El software que permite al diseñador especificar una configuración para un PLD sólo necesita producir un archivo de salida que se adapte a los estándares de JEDEC. Después, este archivo JEDEC puede cargarse en cualquier programador de PLDs compatible con JEDEC que sea capaz de programar el tipo deseado de PLD.

El segundo método se conoce como programación en sistema (ISP). Como su nombre implica, el chip no necesita extraerse de su circuito para su programación. El Grupo Común de Acción de Pruebas (**JTAG**) desarrolló una interfaz estándar para probar los CIs sin tener que conectar el equipo de prueba a todas las terminales del CI. También permite la programación interna. Cuatro terminales en el CI se utilizan como un portal para almacenar datos y recuperar información acerca de la condición interna del CI. Muchos CIs, incluyendo los PLDs y los microcontroladores, se fabrican hoy en día para incluir la interfaz JTAG. Un cable de interfaz conecta las cuatro terminales JTAG en el CI a un puerto de salida (como el puerto de la impresora) de una computadora personal. El software que se ejecuta en la PC establece el contacto con el CI y carga la información en el formato apropiado.

Software de desarrollo

Hasta ahora hemos analizado varios métodos para describir circuitos lógicos, incluyendo la captura de diagramas esquemáticos, las ecuaciones lógicas, las tablas de verdad y el HDL. También describimos los métodos fundamentales para almacenar 1s y 0s en un CI de PLD para conectar los circuitos lógicos de la manera deseada. El mayor reto en cuanto a programar un PLD es realizar la conversión desde cualquier forma de descripción hacia el arreglo de 1s y 0s. Por fortuna, esta tarea se logra con bastante facilidad mediante una computadora que ejecute un software de desarrollo. El software al que haremos referencia y que estaremos utilizando en nuestros ejemplos es propiedad de Altera, y permite al diseñador introducir la descripción de un circuito en cualquiera de las distintas formas que hemos visto: archivos de diseño gráfico (diagramas esquemáticos), AHDL y VHDL. También permite el uso de otro HDL conocido como Verilog, y la opción de describir el circuito mediante diagramas de tiempos. Los bloques de circuitos descritos por cualquiera de estos métodos también pueden "conectarse" entre sí para implementar un sistema digital mucho más grande, como se muestra en la figura 4-44. Cualquier diagrama lógico que se presente en este libro podrá redibujarse mediante el uso de las herramientas de introducción de diagramas esquemáticos en el software Altera para crear un archivo de diseño gráfico. En este libro no nos enfocaremos en la introducción mediante el diseño gráfico, ya que es bastante sencillo obtener estas habilidades en el laboratorio. Enfocaremos nuestros ejemplos en los métodos que nos permiten utilizar el HDL como un medio alternativo para describir un circuito. Para obtener más información sobre el software de Altera, vea el CD que se incluye en este libro y los manuales de usuario del sitio Web de Altera (http://www.altera.com).

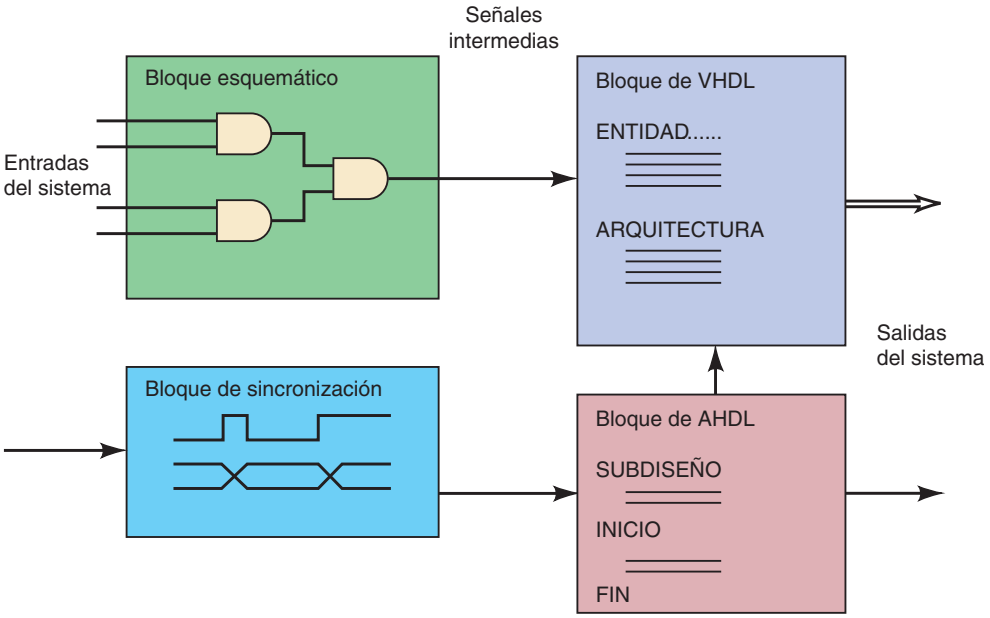


FIGURA 4-44
Combinación de los bloques desarrollados mediante el uso de distintos métodos de descripción.

A este concepto de utilizar bloques de construcción de circuitos se le conoce como **diseño jerárquico**. Pueden definirse circuitos lógicos pequeños y útiles de la manera que sea más conveniente (gráfico, HDL, diagrama de tiempos, etc.) y después se pueden combinar con otros circuitos para formar una selección extensa de un proyecto. Las secciones pueden combinarse y conectarse con otras secciones para formar el sistema completo. La figura 4-45 muestra la estructura jerárquica de un reproductor de CDs mediante el uso de un diagrama de bloques. El cuadro exterior encierra a todo el sistema. Las líneas punteadas identifican cada una de las subsecciones principales y cada subsección contiene circuitos individuales. Aunque no se muestra en este diagrama, cada circuito puede estar compuesto de bloques de cons-

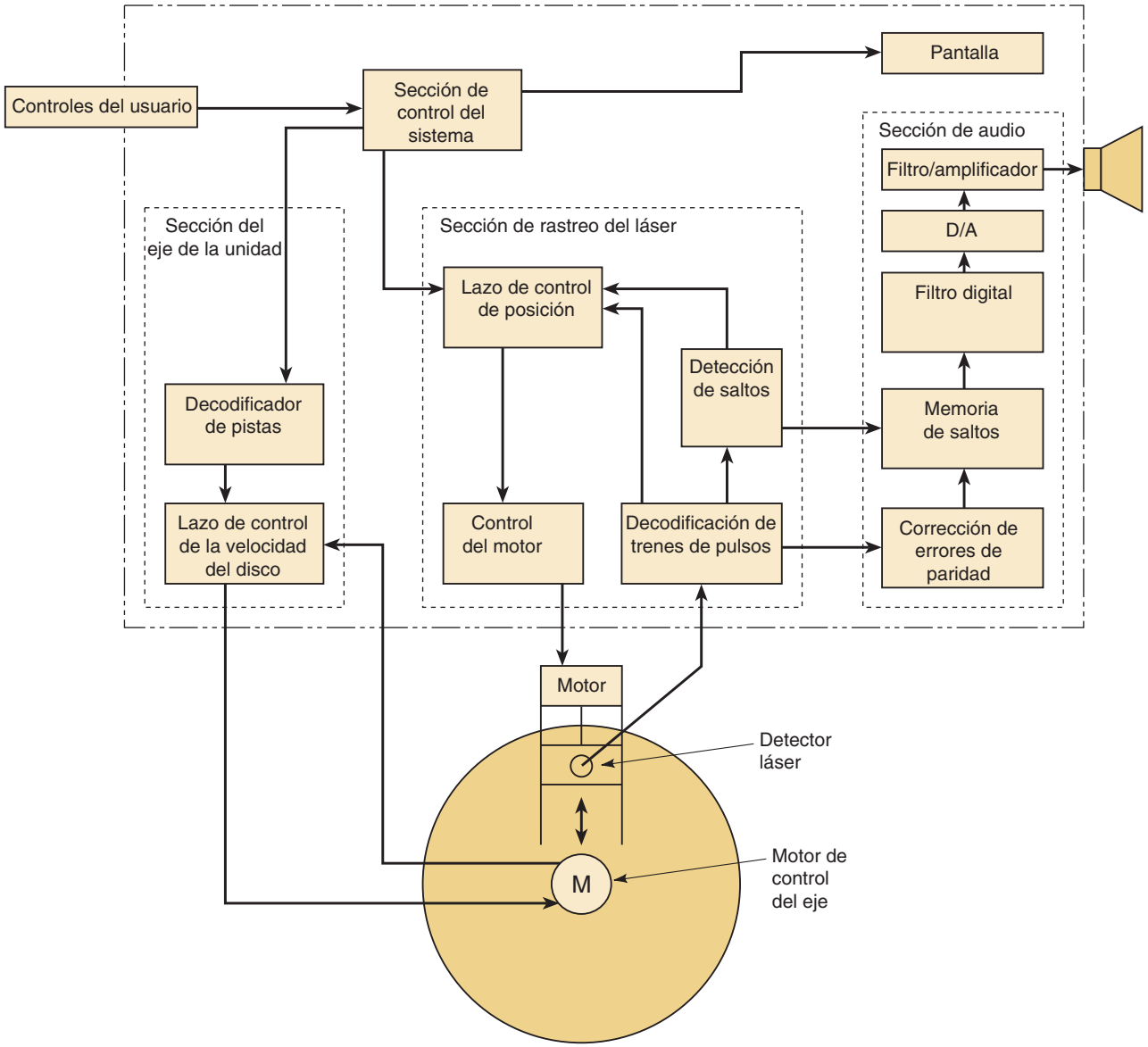


FIGURA 4-45 Diagrama de bloques de un reproductor de CDs.

trucción más pequeños de circuitos digitales comunes. El software de desarrollo de Altera hace que este tipo de diseño y desarrollo modular y jerárquico sea fácil de lograr.

Proceso de diseño y desarrollo

En la figura 4-46 se muestra otra forma en la que podríamos ver la jerarquía de un sistema como el reproductor de CDs que acabamos de describir. El nivel superior representa a todo el sistema completo. Está compuesto de tres subsecciones, cada una de las cuales está a su vez compuesta de los circuitos más pequeños que se muestran. Observe que este diagrama no muestra cómo fluyen las señales en todo el sistema, sino que identifica con claridad los diversos niveles de la estructura jerárquica del proyecto.

Este tipo de diagrama condujo al nombre de uno de los métodos más comunes de diseño: **arriba-abajo**. Con este enfoque de diseño se comienza con la descripción general de todo el sistema, como muestra el cuadro superior de la figura 4-46. Después se definen varias subsecciones que conformarán el sistema. Las subsecciones

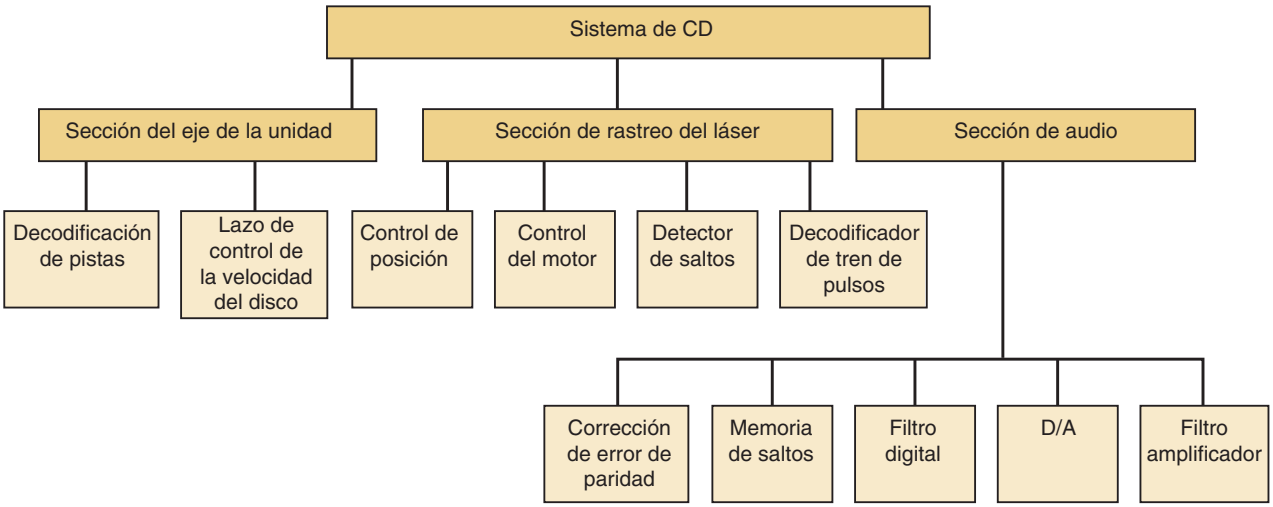


FIGURA 4-46 Un diagrama de jerarquía organizacional.

se refinan aún más en circuitos individuales conectados entre sí. Cada uno de estos niveles jerárquicos tiene definidas las entradas, las salidas y su comportamiento. Cada nivel puede probarse en forma individual, antes de conectarlo a los demás.

Después de definir los bloques de arriba-abajo, el sistema se construye desde abajo hacia arriba. Cada bloque en el diseño de este sistema tiene un archivo de diseño que lo describe. Para diseñar los bloques de los niveles se abre un archivo de diseño y se escribe una descripción de su operación. Después el bloque diseñado se compila mediante el uso de las herramientas de desarrollo. El proceso de compilación determina si usted cometió errores en su sintaxis. La computadora no podrá traducir su descripción en la forma apropiada sino hasta que la sintaxis esté correcta. Una vez que se haya compilado sin errores de sintaxis, deberá probarse para ver si opera en forma correcta. Los sistemas de desarrollo ofrecen programas simuladores que se ejecutan en la PC y simulan la manera en que su circuito responde a las entradas. Un simulador es un programa de computadora que calcula los estados lógicos de salida correctos, con base en una descripción del circuito lógico y las entradas actuales. Se desarrolla un conjunto de entradas hipotéticas y sus correspondientes salidas correctas, las cuales demostrarán que el bloque funciona de la manera esperada. A menudo a estas entradas hipotéticas se les llama **vectores de prueba**. Los procedimientos detallados de prueba durante la simulación incrementan en forma considerable la probabilidad de que el sistema final funcione de manera confiable. La figura 4-47 muestra el archivo de simulación para el circuito descrito en la figura 3-13(a) del capítulo 3. Las entradas *a*, *b* y *c* se introdujeron como vectores de prueba y la simulación produjo la salida *y*.

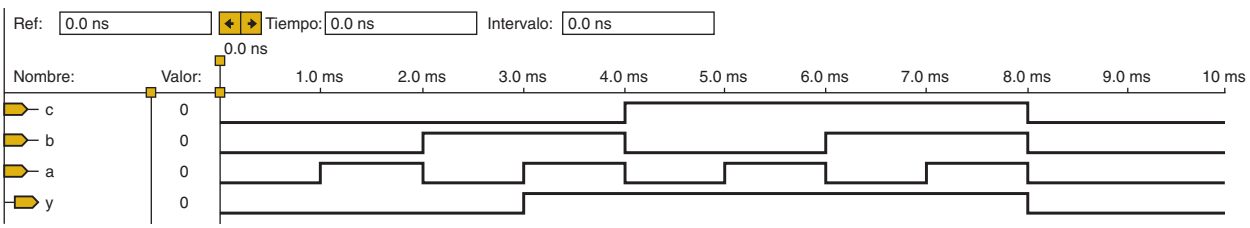
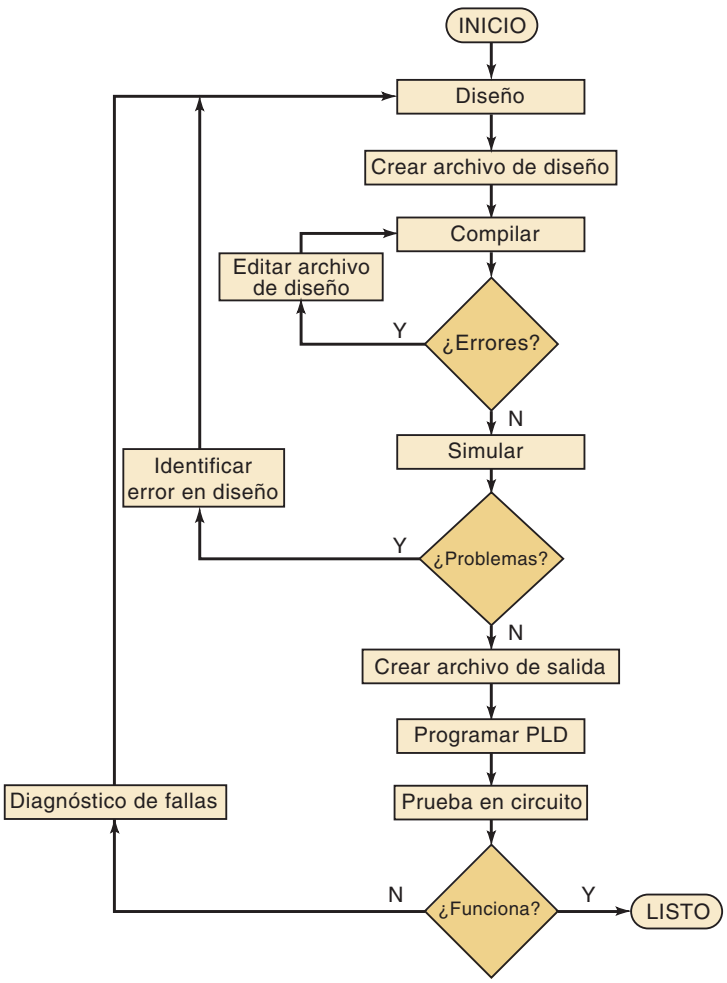


FIGURA 4-47 Una simulación de sincronización de un circuito descrito en HDL.

Cuando el diseñador está satisfecho de que el diseño funciona, éste puede verificarse mediante la programación de un chip y la prueba. Para un PLD complejo, el diseñador puede dejar que el sistema de desarrollo asigne terminales y después se distribuye la tarjeta del circuito final de manera acorde, o puede especificar las

terminales para cada señal mediante el uso de las características del software. Si el compilador asigna las terminales, las asignaciones se pueden encontrar en el archivo de reporte o en el archivo de distribución de terminales, el cual proporciona muchos detalles acerca de la implementación del diseño. Si el diseñador especifica las terminales, es importante conocer las restricciones y limitaciones de la arquitectura del chip. En el capítulo 13 cubriremos estos detalles. El diagrama de flujo de la figura 4-48 sintetiza el proceso de diseño para desarrollar cada bloque.

FIGURA 4-48
Diagrama de flujo del ciclo de desarrollo de un PLD.



Después de probar cada uno de los circuitos en una subsección pueden combinarse todos y se puede probar la subsección siguiendo el mismo proceso utilizado para los circuitos pequeños. Después se combinan las subsecciones y se prueba el sistema. Este método se presta muy bien para un típico entorno de proyecto, en donde un equipo de personas trabajan en conjunto y cada uno es responsable de sus propios circuitos y secciones que al final se reunirán para conformar el sistema.

PREGUNTAS DE REPASO

1. ¿Qué es lo que se "programa" en un PLD?
2. ¿Qué bits (columna, fila) en la figura 4-42 deben conectarse para hacer que el Producto 1 = AB?
3. ¿Qué bits (columna, fila) en la figura 4-42 deben conectarse para hacer que el Producto 3 = $A\overline{B}$?

4-15 REPRESENTACIÓN DE DATOS EN HDL

Los datos numéricos pueden representarse de varias formas. Hemos estudiado el uso del sistema numérico hexadecimal como una forma conveniente de representar patrones de bits. Por naturaleza preferimos usar el sistema numérico decimal para los datos numéricos, pero las computadoras y los sistemas digitales sólo pueden operar con información binaria, como vimos en capítulos anteriores. Cuando escribimos en HDL, a menudo necesitamos utilizar diversos formatos numéricos, y la computadora debe ser capaz de comprender cuál sistema numérico estamos usando. Hasta ahora, en este libro, hemos utilizado un subíndice para indicar el sistema numérico. Por ejemplo, 101_2 es binario, 101_{16} es hexadecimal y 101_{10} es decimal. Cada lenguaje de programación y el HDL tienen su propia manera única de identificar los diversos sistemas numéricos; por lo general, esto se hace mediante un prefijo para indicar el sistema numérico. En la mayoría de los lenguajes un número sin prefijo se considera como decimal. Cuando leemos una de estas designaciones numéricas, debemos considerarla como un símbolo que representa a un patrón binario de bits. Estos valores numéricos se conocen como escalares o **literales**. La tabla 4-8 sintetiza los métodos para especificar valores en binario, hexadecimal y decimal para AHDL y VHDL.

TABLA 4-8 Designación de sistemas numéricos en HDL.

Sistema numérico	AHDL	VHDL	Patrón de bits	Equivalente decimal
Binario	B"101"	B"101"	101	5
Hexadecimal	H"101"	X"101"	100000001	257
Decimal	101	101	1100101	101

EJEMPLO 4-29

Exprese el valor numérico del siguiente patrón de bits en binario, hexadecimal y decimal mediante el uso de la notación de AHDL y VHDL:

11001

Solución

El binario se designa de la misma forma en AHDL y en VHDL: **B "11001"**.

Si convertimos el binario en hexadecimal, tendremos 19_{16}.

En AHDL: **H "19"**

En VHDL: **X "19"**

Si convertimos el binario en decimal, tendremos 25_{10}.

El decimal se designa de la misma forma en AHDL y VHDL: 25.

Arreglos de bits/vectores de bits

En el capítulo 3 declaramos nombres para las entradas y las salidas de un circuito lógico muy simple. Éstas se definieron como bits, o dígitos binarios individuales. ¿Qué pasaría si quisiéramos representar una entrada, salida o señal compuesta por varios bits? En un HDL debemos definir el tipo de la señal y su intervalo de valores válidos.

Para comprender los conceptos utilizados en los HDLs, consideremos primero algunas convenciones para describir bits de palabras binarias en los sistemas digitales comunes. Suponga que tenemos un número de ocho bits que representa la temperatura actual, y que el número está llegando a nuestro sistema digital a través de un puerto de entrada que hemos identificado como P1, como se muestra en la figura 4-49. Podemos referirnos a los bits individuales de este puerto como el bit 0 de P1 para el bit menos significativo, y hasta el bit 7 de P1 para el bit más significativo.

También podemos describir este puerto diciendo que su nombre es P1 y sus bits están numerados del 7 al 0. Los términos **arreglo de bits** y **vector de bits** se utilizan con frecuencia para describir este tipo de estructura de datos. Lo que esto significa es que la estructura de datos en general (puerto de ocho bits) tiene un nombre (P1) y que cada elemento individual (bit) tiene un número de **índice** único (0-7) para describir la posición de cada bit (y tal vez su peso numérico) en la estructura general. Los HDLs y los lenguajes de programación de computadora sacan provecho de esta notación. Por ejemplo, el tercer bit a partir de la derecha se designa como P1[2] y puede conectarse a otro bit de señal mediante el uso de un operador de asignación.

EJEMPLO 4-30

Suponga que hay un arreglo de ocho bits llamado P1 como se muestra en la figura 4-49, y que hay otro arreglo de cuatro bits llamado P5.

(a) Escriba la designación para el bit más significativo de P1.
(b) Escriba la designación para el bit menos significativo de P5.
(c) Escriba una expresión que haga que el bit menos significativo de P5 controle el bit más significativo de P1.

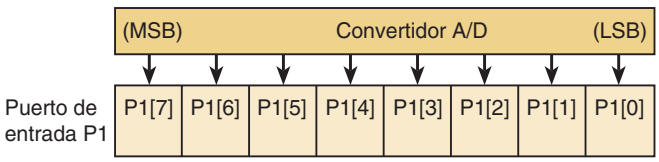


FIGURA 4-49
Notación de arreglo de bits.

Solución

(a) El nombre del puerto es P1 y el bit más significativo es el bit 7. La designación apropiada para el bit 7 de P1 es P1[7].
(b) El nombre del puerto es P5 y el bit menos significativo es el bit 0. La designación apropiada para el bit 0 de P5 es P5[0].
(c) La señal controladora se coloca en el lado derecho del operador de asignación y la señal controlada se coloca a la izquierda: P1[7] = P5[0];.

AHDL

DECLARACIÓN DE ARREGLOS DE BITS EN AHDL

En AHDL, el puerto *p1* de la figura 4-49 se define como un puerto de entrada de ocho bits y para hacer referencia a el valor en este puerto se puede utilizar cualquier sistema numérico tal como hexadecimal, binario, decimal, etc. La sintaxis para AHDL utiliza un nombre para el vector de bits, seguido por el intervalo de las designaciones de los índices, las cuales van encerradas entre corchetes. Esta declaración se incluye en la sección SUBDESIGN. Por ejemplo, para declarar un puerto de entrada de ocho bits llamado *p1*, escribiríamos lo siguiente:

```
p1 [7..0] : INPUT;     define un puerto de entrada de 8 bits
```

EJEMPLO 4-31

Declare una entrada de cuatro bits llamada *teclado* mediante el uso de AHDL.

Solución

```
teclado [3..0]  :  INPUT;
```

Las variables intermedias también pueden declararse como un arreglo de bits. Al igual que los bits individuales, se declaran justo después de las declaraciones de E/S en SUBDESIGN. Como ejemplo, el puerto de temperatura de ocho bits llamado *p1* puede asignarse (conectarse) a un nodo llamado *temp*, de la siguiente manera:

```
VARIABLE temp [7..0] :NODE;
BEGIN
    temp[] = p1[];
END;
```

Observe que los datos se aplican al puerto de entrada *p1* y que está controlando los alambres de la señal llamada *temp*. Podemos considerar el término a la derecha del signo de igual como el origen de los datos, y el término a la izquierda como la designación. Los corchetes vacíos [ ] indican que cada uno de los bits correspondientes en los dos arreglos están conectados. También pueden "conectarse" bits individuales si se especifican los bits dentro de los corchetes. Por ejemplo, para conectar sólo el bit menos significativo de *p1* con el LSB de *temp*, la instrucción sería temp[0] = p1[0];.

DECLARACIONES DE VECTORES DE BITS EN VHDL

En VHDL, el puerto *p1* de la figura 4-49 se define como un puerto de entrada de ocho bits y sólo se puede hacer referencia al valor de este puerto mediante el uso de literales binarias. La sintaxis para VHDL utiliza un nombre para el vector de bits, seguido del modo (:IN), el tipo (**BIT_VECTOR**) y el intervalo de las designaciones de los índices, que van encerradas entre paréntesis. Esta declaración se incluye en la sección ENTITY. Por ejemplo, para declarar un puerto de entrada de ocho bits llamado *p1*, escribiríamos lo siguiente:

```
PORT (p1  : IN   BIT VECTOR  (7 DOWNTO 0);
```

EJEMPLO 4-32

Declare una entrada de cuatro bits llamada *teclado* mediante el uso de VHDL.

Solución

```
PORT(teclado  : IN  BIT VECTOR  (3 DOWNTO 0);
```

También pueden declararse señales intermedias como un arreglo de bits. Al igual que los bits individuales, se declaran justo dentro de la definición ARCHITECTURE. Como ejemplo, la temperatura de ocho bits en el puerto *p1* puede asignarse (conectarse) a una señal llamada *temp*, como se muestra a continuación:

```
SIGNAL     temp  : BIT VECTOR (7 DOWNTO 0);
BEGIN
     temp <5 p1;
END;
```

Observe que los datos se aplican al puerto de entrada *p1* y que está controlando los alambres de la señal llamada *temp*. No se especifican elementos en el vector de bits, lo cual significa que todos los bits están conectados. También pueden "conectarse" bits individuales mediante el uso de asignaciones de señales y especificando los números de los bits entre paréntesis. Por ejemplo, para conectar sólo el bit menos significativo de *p1* con el LSB de *temp*, la instrucción sería temp(0) < = p1(0);.

VHDL es muy específico con respecto a las definiciones de cada tipo de datos. El tipo "bit_vector" describe a un arreglo de bits individuales. Esto se interpreta de manera distinta que un número binario de ocho bits (llamado cantidad escalar), el cual tiene el tipo **integer**. Por desgracia, VHDL no nos permite asignar un valor entero a una señal BIT_VECTOR de forma directa. Los datos pueden representarse mediante cualquiera de los tipos que se muestran en la figura 4-9, pero las asignaciones de datos y demás operaciones deben realizarse entre objetos del mismo tipo. Por ejemplo, el compilador no le permitirá recibir un número de un teclado declarado como entero y conectarlo a cuatro LEDs que estén declarados como salidas BIT_VECTOR. En la tabla 4-9 bajo Posibles valores podrá observar que los **objetos** individuales de datos BIT y STD_LOGIC (por ejemplo: señales, variables, entradas y salidas) se designan mediante comillas, mientras que los valores asignados a los tipos BIT_VECTOR y STD_LOGIC_VECTOR son cadenas de valores de bits válidos encerradas entre comillas dobles.

TABLA 4-9 Tipos de datos comunes en VHDL.

Tipo de datos	Declaración de ejemplo	Posibles valores	Uso
BIT	y :OUT BIT;	'0' '1'	y <= '0';
STD LOGIC	controlador: STD LOGIC	'0' '1' 'z' 'x' '-'	controlador <= 'z';
BIT VECTOR	datos bcd :BIT VECTOR (3 DOWNTO 0);	"0101" "1001" "0000"	digito <= datos bcd;
STD LOGIC VECTOR	dbus :STD LOGIC VECTOR (3 DOWNTO 0);	"0Z1X"	IF rd = '0' THEN dbus <= "zzzz";
INTEGER	SIGNAL z:INTEGER RANGE − 32 TO 31;	−32..−2, −1, 0, 1, 2 . . . 31	IF z > 5 THEN . . .

VHDL también ofrece ciertos tipos de datos estandarizados que se necesitan al utilizar funciones lógicas contenidas en las **bibliotecas**. Como habrá imaginado, las bibliotecas son simples colecciones de pequeñas piezas de código de VHDL, las cuales puede usar en sus descripciones de hardware sin necesidad de empezar desde cero. A menudo estas bibliotecas ofrecen funciones de uso común conocidas como **macrofunciones**, al igual que muchos de los dispositivos TTL estándar que se describen en este libro. En vez de escribir la nueva descripción de un dispositivo TTL conocido, podemos tan sólo sacar su macrofunción de la biblioteca y utilizarla en nuestro sistema. Desde luego que necesita que las señales entren y salgan de estas macrofunciones; además los tipos de las señales en su código deben concordar con los tipos en las funciones (que alguien más escribió). Esto significa que todos deben utilizar los mismos tipos de datos estándar.

Cuando el VHDL se estandarizó a través del IEEE, se crearon muchos tipos de datos a la vez. Lo dos que utilizaremos en este libro son **STD_LOGIC**, que es equivalente al tipo BIT, y **STD_LOGIC_VECTOR**, que es equivalente a BIT_VECTOR. Como podrá recordar, el tipo BIT sólo puede tener los valores '0' y '1'. Los tipos lógicos estándar vienen definidos en la biblioteca IEEE y tienen un intervalo más amplio de valores posibles que sus contrapartes integradas en la biblioteca. Los posibles valores para un tipo STD_LOGIC o para cualquier elemento en un STD_LOGIC_VECTOR se muestran en la tabla 4-10. Los nombres de estas categorías tendrán mucho más sentido una vez que estudiemos las características de los circuitos lógicos en el capítulo 8. Por ahora mostraremos ejemplos con el uso de los valores '1' y '0' solamente.

TABLA 4-10
Valores de STD_LOGIC.

'1'	1 lógico (justo igual que el tipo BIT)
'0'	0 lógico (justo igual que el tipo BIT)
'z'	Alta impedancia*
'-'	No importa (justo igual que como se utilizó en los mapas K)
'U'	Sin inicializar
'X'	Desconocido
'W'	Desconocido débil
'L'	'0' débil
'H'	'1' débil

* En el capítulo 8 estudiaremos la lógica de tres estados.

PREGUNTAS DE REPASO

1. ¿Cómo declararía un arreglo de entrada de seis bits llamado botones_pulsar en (a) AHDL o en (b) VHDL?
2. ¿Qué instrucción utilizaría para extraer el MSB del arreglo de la pregunta 1 y colocarlo en un puerto de salida de un solo bit llamado *z*? Use (a) AHDL o (b) VHDL.
3. En VHDL, ¿cuál es el tipo estándar IEEE que es equivalente al tipo BIT?
4. En VHDL, ¿cuál es el tipo estándar IEEE que es equivalente al tipo BIT_VECTOR?

4-16 TABLAS DE VERDAD MEDIANTE EL USO DE HDL

Ya hemos aprendido que una tabla de verdad es otra forma de expresar la operación de un bloque de circuitos. Relaciona la salida del circuito con cada una de las posibles combinaciones de sus entradas. Como vimos en la sección 4-4, una tabla de verdad es el punto inicial para que un diseñador defina la manera en que debe operar el circuito. Después se deriva una expresión booleana de la tabla de verdad y se simplifica mediante el uso de los mapas K o del álgebra booleana. Por último, el circuito se implementa a partir de la ecuación booleana final. ¿No sería grandioso si pudiéramos partir de la tabla de verdad y llegar en forma directa al circuito final, sin todos esos pasos intermedios? Si utilizamos HDL para introducir la tabla de verdad, podremos hacer justo eso.

TABLAS DE VERDAD MEDIANTE EL USO DE AHDL

El código de la figura 4-50 utiliza AHDL para implementar un circuito y utiliza una tabla de verdad para describir su operación. La tabla de verdad para este diseño se presentó en el ejemplo 4-7. El punto clave de este ejemplo es el uso de la palabra clave TABLE en AHDL. Esta palabra permite al diseñador especificar la operación del circuito, igual que como se llena una tabla de verdad. En la primera línea después de TABLE se listan las variables de entrada (*a*, *b*, *c*) de la misma forma como se crearía un encabezado de columna en una tabla de verdad. Al incluir las tres variables binarias entre paréntesis, indicamos al compilador que deseamos utilizar estos tres bits como un grupo y que nos referiremos a ellos como si fueran un número binario o un patrón de tres bits. Los valores específicos para este patrón de bits se listan debajo del grupo y se les denomina literales binarias. El operador especial (= >) se utiliza en las tablas de verdad para separar las entradas de la salida (*y*).

AHDL

FIGURA 4-50 Archivo de diseño de AHDL para la figura 4-7.

```
%     Figura 4 7 en AHDL
      Sistemas digitales 10a ed
      Neal Widmer
      MAYO 23, 2005                    %
SUBDESIGN FIG4 50
(
      a,b,c :INPUT;           define las entradas del bloque
      y     :OUTPUT;          define la salida del bloque
)
BEGIN
      TABLE
            (a,b,c)          =>    y;     encabezados de columna
            (0,0,0)          =>    0;
            (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)          =>    1;
      END TABLE;
END;
```

La instrucción TABLE en la figura 4-50 es para mostrar la relación entre el código de HDL y una tabla de verdad. Una manera más común de representar los encabezados de los datos de entrada es mediante el uso de un arreglo de bits para representar el valor en *a*, *b*, *c*. Este método requiere que se declare el arreglo de bits en la línea antes de BEGIN, como se muestra a continuación:

```
VARIABLE  bits ent[2..0]          :NODE;
```

Justo antes de la palabra clave TABLE, los bits de entrada pueden asignarse al arreglo *bits_ent[ ]*:

```
bits ent[ ] = (a, b, c);
```

Al proceso de agrupar tres bits independientes en un orden como el anterior se le conoce como **concatenación**; este proceso se lleva a cabo con frecuencia para conectar bits individuales con un arreglo de bits. En este caso, el encabezado de la tabla en los conjuntos de bits de entrada puede representarse mediante *bits_ent[ ]*. Observe que, a medida que listamos las posibles combinaciones de las entradas, tenemos varias opciones. Podemos crear un grupo de 1s y 0s entre paréntesis, como se muestra en la figura 4-50, o podemos representar el mismo patrón de bits utilizando el número equivalente en binario, hexadecimal o decimal. El diseñador es el que decidirá cuál formato es el más apropiado, dependiendo de lo que representen las variables de entrada.

VHDL

TABLAS DE VERDAD MEDIANTE EL USO DE VHDL: ASIGNACIÓN DE SEÑAL SELECCIONADA

El código de la figura 4-51 utiliza VHDL para implementar un circuito mediante una **asignación de señal seleccionada** para describir su operación. Esto le permite al diseñador especificar la operación del circuito, de la misma forma como se llena una tabla de verdad. En el ejemplo 4-7 se presentó la tabla de verdad para este diseño. El objetivo principal de este ejemplo es el uso de la instrucción WITH nombre_señal

```
      Figura 4 7 en VHDL
      Sistemas digitales 10a ed
      Neal Widmer
      MAYO 23, 2005
ENTITY fig4 51 IS
PORT(
      a,b,c :IN BIT;            declara bits de entrada individuales
      y     :OUT BIT);
END fig4 51;

ARCHITECTURE verdad OF fig4 51 IS
      SIGNAL bits ent  : BIT VECTOR(2 DOWNTO 0);
      BEGIN
      bits ent <5 a & b & c;   concatena los bits de entrada en bit vector
            WITH bits ent  SELECT
            y     <=    '0'  WHEN "000",     Tabla de verdad
                        '0'  WHEN "001",
                        '0'  WHEN "010",
                        '1'  WHEN "011",
                        '0'  WHEN "100",
                        '1'  WHEN "101",
                        '1'  WHEN "110",
                        '1'  WHEN "111";
END verdad;
```

FIGURA 4-51 Archivo de diseño de VHDL para la figura 4-7.

SELECT en VHDL. Uno de los objetivos secundarios es mostrar cómo se colocan los datos en un formato que pueda ser utilizado de manera conveniente con la asignación de la señal seleccionada. Observe que las entradas están definidas en la declaración ENTITY como tres bits independientes *a*, *b* y *c*. Nada en esta declaración hace que uno de estos bits sea más significativo que los demás. El orden en el que se listan no importa. Queremos comparar el valor actual de estos bits con cada una de las posibles combinaciones de entrada que podrían presentarse. Si trazáramos una tabla de verdad, decidiríamos cuál bit colocar a la izquierda (MSB) y cuál a la derecha (LSB). En VHDL esto se logra mediante la **concatenación** (conectar en orden) de las variables de bit para formar un vector de bits. El operador de concatenación es "&". Se declara una señal como BIT_VECTOR para recibir el conjunto ordenado de bits de entrada y se utiliza para comparar el valor de entrada con las literales de cadena encerradas entre comillas. A la salida (*y*) se le asigna (<=) un valor de bit ('0' o '1') cuando (WHEN) *bits_ent* contiene el valor que se lista entre comillas dobles.

VHDL es muy estricto en cuanto a la forma en que nos permite asignar y comparar objetos tales como señales, variables, constantes y literales. La salida *y* es un BIT, por lo que se le debe asignar un valor de '0' o de '1'. La señal *bits_ent* es un BIT_VECTOR de tres bits, por lo que debe compararse con un valor literal de cadena de tres bits. VHDL no permitirá que *bits_ent* (un BIT_VECTOR) se compare con un número hexadecimal tal como X "5", o con un número decimal tal como 3. Estas cantidades escalares serían válidas para la asignación o comparación con enteros.

EJEMPLO 4-33

Declare tres señales en VHDL que sean bits individuales de nombre *demasiado_caliente*, *demasiado_frio* y *muy_bien*. Combine (concatene) estos tres bits en una señal de tres bits llamada *estado_temp*, en donde lo caliente estará a la izquierda y lo frío a la derecha.

Solución

1. Declare primero las señales en la arquitectura (ARCHITECTURE).

```
SIGNAL  demasiado caliente, demasiado frio, muy bien  :BIT;
SIGNAL  estado temp  : BIT VECTOR (2 DOWNTO 0);
```

2. Escriba instrucciones de asignación concurrentes entre BEGIN y END.

```
estado temp <5 demasiado caliente & muy bien & demasiado frio;
```

PREGUNTAS DE REPASO

1. ¿Cómo concatenaría los tres bits *x*, *y* y *z* en un arreglo de tres bits llamado *omega*? Use AHDL o VHDL.
2. ¿Cómo se implementan las tablas de verdad en AHDL?
3. ¿Cómo se implementan las tablas de verdad en VHDL?

4-17 ESTRUCTURAS DE CONTROL DE DECISIONES EN HDL

En esta sección examinaremos métodos que nos permiten indicar al sistema digital cómo realizar decisiones "lógicas", en forma muy similar al proceso que utilizamos para hacer decisiones en nuestra vida diaria. En el capítulo 3 vimos que las instrucciones de asignación concurrentes se evalúan de tal forma que el orden en el que se escriben no tiene efecto sobre el circuito que se está describiendo. Cuando utilizamos **estructuras de control de decisiones**, el orden en el que hacemos las preguntas sí es importante. Para resumir este concepto en los términos utilizados en la documentación del HDL, a las instrucciones que se pueden escribir en cualquier secuencia se les llama **concurrentes**, y a las instrucciones que se evalúan en la secuencia en la que se escriben se les llama **secuenciales**. La secuencia de las instrucciones secuenciales afecta a la operación del circuito.

Los ejemplos que hemos visto hasta ahora implican el uso de varios bits individuales. Muchos sistemas digitales requieren entradas que representan un valor numérico. Consulte de nuevo el ejemplo 4-8, en el cual el objetivo del circuito lógico es monitorear el voltaje de la batería mediante un convertidor A/D. El valor digital se representa mediante un número de cuatro bits que proviene del convertidor A/D y que pasa al circuito lógico. Estas entradas no son variables binarias independientes, sino cuatro dígitos binarios de un número que representa el voltaje de la batería. Necesitamos dar a estos datos el tipo correcto que nos permita utilizarlos como un número.

IF/ELSE

Las tablas de verdad son estupendas para listar todas las posibles combinaciones de variables independientes, pero hay mejores formas de manejar los datos numéricos. Como ejemplo, cuando una persona parte hacia la escuela o su trabajo en la mañana, debe realizar una decisión lógica para saber si se va a llevar o no un abrigo. Supongamos que esta persona decide sobre esta cuestión basándose únicamente en la temperatura actual. ¿Cuántos de nosotros razonaríamos de la siguiente manera?

Utilizaré un abrigo si la temperatura es 0.
Utilizaré un abrigo si la temperatura es 1.
Utilizaré un abrigo si la temperatura es 2....
Utilizaré un abrigo si la temperatura es 13.

No utilizaré un abrigo si la temperatura es 14.
No utilizaré un abrigo si la temperatura es 15.
No utilizaré un abrigo si la temperatura es 16….
No utilizaré un abrigo si la temperatura es 40.

Este método es similar a aquél en el que se utiliza la tabla de verdad para describir la decisión. Para cada una de las posibles entradas, esta persona decide cuál debe ser la salida. En realidad, lo que haría para decidirse sería lo siguiente:

Usaré un abrigo si la temperatura es menor de 14 grados.
En caso contrario, *no* utilizaré un abrigo.

Un HDL nos permite describir los circuitos lógicos mediante este tipo de razonamiento. Primero debemos describir las entradas como un *número dentro de un intervalo dado*, y después podemos escribir instrucciones que decidan lo que se debe hacer en las salidas con base en el *valor* del número entrante. Al igual que en los HDLs, en la mayoría de los lenguajes de programación de computadoras estos tipos de decisiones se llevan a cabo mediante el uso de una estructura de control IF/THEN/ELSE. Cada vez que la decisión está entre realizar una acción o no realizarla, se utiliza la instrucción **IF/THEN**. La palabra clave IF va seguida de una instrucción que es verdadera o falsa. Si (IF) es verdadera, entonces (THEN) se hace lo que esté especificado. En el caso en el que la instrucción sea falsa, no se realiza ninguna acción. La figura 4-52(a) muestra en forma gráfica la manera como funciona esta decisión. La figura de diamante representa la decisión que se está llevando a cabo mediante la evaluación de la instrucción contenida dentro del diamante. Toda decisión tiene dos posibles resultados: verdadero o falso. En este ejemplo, si la instrucción es falsa no se realiza ninguna acción.

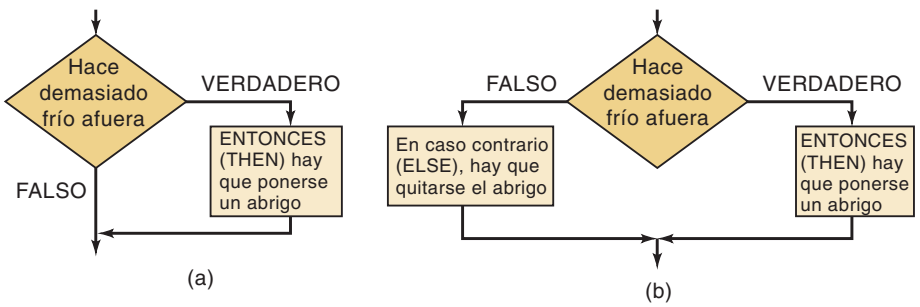


FIGURA 4-52
Flujo lógico de las instrucciones (a) IF/THEN y (b) IF/THEN/ELSE.

En algunos casos no sólo basta con decidir entre actuar y no actuar, sino que debemos elegir entre dos acciones distintas. Por ejemplo, en nuestra analogía sobre la decisión de usar un abrigo, si la persona ya tiene su abrigo puesto cuando llegue a esta decisión, no se lo quitará. El uso de la lógica IF/THEN supone que de inicio no está usando su abrigo.

Cuando las decisiones demandan dos posibles acciones, se utiliza la estructura de control IF/THEN/**ELSE** como se muestra en la figura 4-52(b). Aquí se evalúa de nuevo la instrucción como verdadera o falsa. La diferencia es que, cuando la instrucción es falsa se realiza una acción distinta. Debe realizarse una de las dos acciones con esta instrucción. Podemos describirla en forma verbal como, "SI (IF) la instrucción es verdadera, ENTONCES (THEN) hay que hacer esto. EN CASO CONTRARIO (ELSE) hay que hacer esto otro". En nuestra analogía del abrigo esta estructura de control funcionaría, sin importar que la persona haya traído puesto su abrigo o no desde un principio.

En el ejemplo 4-8 se mostró un ejemplo simple de un circuito lógico que tiene un valor numérico como entrada, el cual representa el voltaje de la batería de un convertidor A/D. Las entradas *A*, *B*, *C*, *D* son en realidad dígitos binarios en un número de 4 bits, en donde *A* es el MSB y *D* es el LSB. La figura 4-53 muestra el mismo cir-

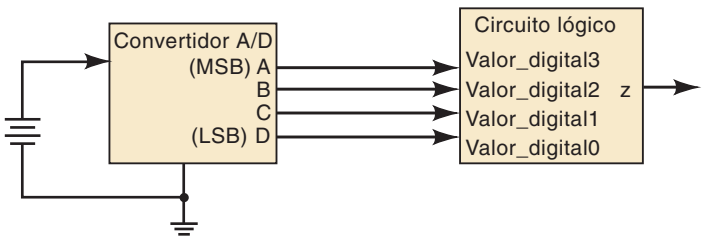


FIGURA 4-53 Circuito lógico similar al del ejemplo 4-8.

cuito con las entradas etiquetadas en forma de un número de cuatro bits, de nombre *valor_digital*. La relación entre los bits es la siguiente:

A	*valor_digital[3]*	bit 3 del valor digital (MSB)
B	*valor_digital[2]*	bit 2 del valor digital
C	*valor_digital[1]*	bit 1 del valor digital
D	*valor_digital[0]*	bit 0 del valor digital (LSB)

La entrada puede considerarse como un número decimal entre 0 y 15, si especificamos el tipo correcto de la variable de entrada.

AHDL

IF/THEN/ELSE MEDIANTE EL USO DE AHDL

En AHDL, las entradas pueden especificarse como un número binario formado de varios bits mediante la asignación de un nombre de variable, seguido de una lista de las posiciones de los bits, como se muestra en la figura 4-54. El nombre es *valor_digital* y las posiciones de los bits varían desde 3 hasta 0. Observe lo simple que se vuelve el código al utilizar este método junto con una instrucción IF/ELSE. La palabra clave IF va seguida de una instrucción que hace referencia al valor completo de la variable de entrada de cuatro bits y lo compara con el número 6. Desde luego que 6 es la forma decimal de una cantidad escalar y *valor_digital[ ]* representa en realidad a un número binario. Como el compilador puede interpretar números en cualquier sistema, crea un circuito lógico que compara el valor binario de *valor_digital* con el número binario correspondiente al 6 decimal y decide si esta instrucción es verdadera o falsa. Si es verdadera, ENTONCES (THEN) se utiliza la siguiente instrucción (z = VCC) para asignar un valor a z. Observe que en AHDL debemos usar VCC para un 1 lógico y GND para un 0 lógico cuando asignamos un nivel lógico a un bit individual. Cuando *valor_digital* es 6 o menos, va después de la instrucción que sigue de ELSE (z = GND). La instrucción END IF termina la estructura de control.

FIGURA 4-54 Versión en AHDL.

```
SUBDESIGN FIG4 54
(
   valor digital[3..0]  :INPUT;     define las entradas del bloque
   z                    :OUTPUT;    define la salida del bloque
)
BEGIN
   IF valor digital[ ] > 6 THEN
         z = VCC;                    la salida es 1
   ELSE  z = GND;                    la salida es 0
   END IF;
END;
```

IF/THEN/ELSE MEDIANTE EL USO DE VHDL

En VHDL la cuestión importante es la declaración del tipo de entradas. (Consulte la figura 4-55.) La entrada se trata como una variable individual llamada *valor_digital*. Como su tipo se declara como INTEGER, el compilador sabe que debe tratarla como un número. Al especificar un intervalo de 0 a 15, el compilador sabe que es un número de cuatro bits. Observe que RANGE no especifica el número de índice de un vector de bits, sino los límites del valor numérico del entero. En VHDL los enteros se tratan de manera distinta a los arreglos de bits (BIT_VECTOR). Un entero puede compararse con otros números mediante el uso de operadores de desigualdad. Un BIT_VECTOR no puede usarse con operadores de desigualdad.

FIGURA 4-55
Versión en VHDL.

```
ENTITY fig4 55 IS
PORT(  valor digital :IN INTEGER RANGE 0 TO 15;    entrada de 4 bits
       z             :OUT BIT);
END fig4 55;

ARCHITECTURE decision OF fig4 55 IS

BEGIN
   PROCESS (valor digital)
      BEGIN
         IF (valor digital > 6) THEN
            z <= '1';
         ELSE
            z <= '0';
      END IF;
END PROCESS ;
END decision;
```

Para utilizar la estructura de control IF/THEN/ELSE, VHDL requiere que el código se coloque dentro de un proceso (PROCESS). Las instrucciones que ocurren dentro de un proceso son *secuenciales*, lo cual significa que el orden en el que se escriben afecta a la operación del circuito. La palabra clave **PROCESS** va seguida de una lista de variables, a la cual se le llama **lista de sensibilidad**, que es una lista de variables a las cuales el código del proceso debe responder. Cada vez que *valor_digital* cambia, hace que se vuelva a evaluar el código del proceso. Aún y cuando sabemos que *valor_digital* es en realidad un número binario de cuatro bits, el compilador lo evaluará como un número entre los valores decimales equivalentes de 0 y 15. Si (IF) la instrucción entre paréntesis es verdadera, entonces (THEN) se aplica la siguiente instrucción (a *z* se le asigna el valor de 1 lógico). Si esta instrucción no es verdadera, la lógica sigue la cláusula ELSE y asigna a z un valor de 0. La instrucción END IF; termina la estructura de control y END PROCESS; termina la evaluación de las instrucciones secuenciales.

ELSIF

A menudo tenemos que elegir de entre muchas posibles acciones, dependiendo de la situación. La instrucción IF decide si se va a realizar o no un conjunto de acciones. La instrucción IF/ELSE selecciona una de dos posibles acciones. Mediante la combinación de las decisiones con IF y ELSE podemos crear una estructura de control que

se conoce como **ELSIF**, la cual selecciona uno de muchos resultados posibles. En la figura 4-56 se muestra la estructura de decisión en forma gráfica.

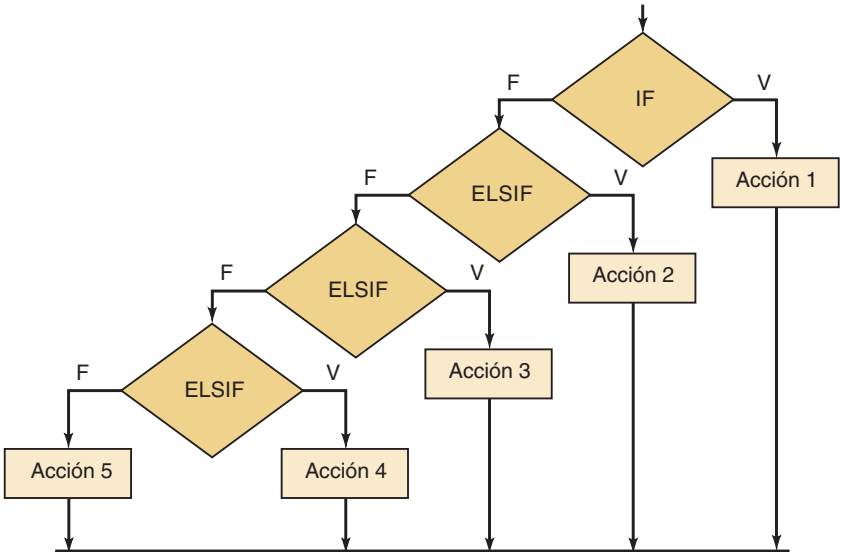


FIGURA 4-56
Diagrama de flujo para decisiones múltiples en las que se utiliza IF/ELSIF.

Observe que, a medida que se evalúa cada condición, se realiza una acción si es verdadera o en caso contrario pasa a evaluar la siguiente condición. Cada acción está asociada con una condición y no existe la probabilidad de seleccionar más de una acción. Observe también que las condiciones que se utilizan para decidir la acción apropiada pueden ser cualquier expresión que se evalúe como verdadera o como falsa. Este hecho permite al diseñador utilizar los operadores de desigualdad para seleccionar una acción con base en un intervalo de valores de entrada. Como ejemplo de esta aplicación, consideremos el sistema de medición de temperatura que utiliza un convertidor A/D, como se describe en la figura 4-57. Suponga que deseamos indicar cuando la temperatura se encuentra en cierto Intervalo, al cual denominaremos como Demasiado frío, Templado y Demasiado caliente.

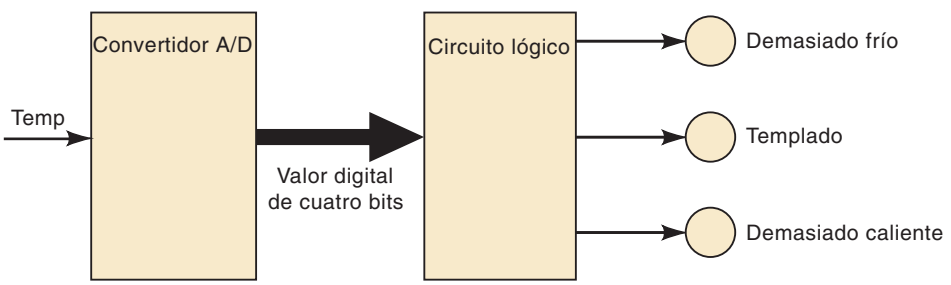


FIGURA 4-57
Circuito indicador de un intervalo de temperatura.

La relación entre los valores digitales para la temperatura y las categorías es

Valores digitales	*Categoría*
0000-1000	Demasiado frío
1001-1010	Templado
1011-1111	Demasiado caliente

Podemos expresar el proceso de toma de decisiones para este circuito lógico de la siguiente manera:

> Si (IF) el valor digital es menor o igual a 8, entonces (THEN) hay que encender sólo el indicador "Demasiado frío".
>
> En caso contrario, si (ELSE IF) el valor digital es mayor que 8 y (AND) menor que 11, entonces (THEN) hay que encender sólo el indicador "Templado".
>
> En caso contrario (ELSE) hay que encender sólo el indicador "Demasiado caliente".

AHDL

ELSIF MEDIANTE EL USO DE AHDL

El código de AHDL de la figura 4-58 define las entradas en forma de un número binario de cuatro bits. Las salidas son tres bits individuales que controlan los tres indicadores de Intervalo. En este ejemplo se utiliza una variable intermedia (*estado*), la cual nos permite asignar un patrón de bits que representa las tres condiciones de *demasiado_frio*, *templado* y *demasiado_caliente*. La sección secuencial del código utiliza las instrucciones IF, ELSIF, ELSE para identificar el intervalo en el cual se encuentra la temperatura y asigna a *estado* el patrón de bits correcto. En la última instrucción, los bits de *estado* se conectan a los bits del puerto de salida real. Estos bits están ordenados en un grupo que se relaciona con los patrones de bits asignados a *estado[ ]*. Esto también podría haberse escrito mediante tres instrucciones concurrentes: demasiado_frio = estado[2]; templado = estado[1]; demasiado_caliente = estado[0];

```
SUBDESIGN fig4 58
(
   valor digital[3..0]           :INPUT;    define las entradas del bloque
   demasiado frio, templado, demasiado caliente :OUTPUT;    define las salidas
)
VARIABLE
estado[2..0] :NODE;   guarda el estado de demasiado frio, templado, demasiado caliente
BEGIN
   IF       valor digital[] <= 8 THEN estado[] = b"100";
   ELSIF    valor digital[] > 8 AND valor digital[] < 11 THEN
         estado[] = b"010";
   ELSE  estado[] = b"001";
   END IF;
   (demasiado frio, templado, demasiado caliente) 5 estado[ ];    actualiza los bits de
    salida
END;
```

FIGURA 4-58 Ejemplo de Intervalos de temperatura en AHDL mediante el uso de ELSIF.

VHDL

ELSIF MEDIANTE EL USO DE VHDL

El código de VHDL en la figura 4-59 define las entradas como un entero de cuatro bits. Las salidas son tres bits individuales que controlan los tres indicadores de Intervalo. En este ejemplo se utiliza una señal intermedia (*estado*), la cual nos permite asignar un patrón de bits que representa a las tres condiciones *demasiado_frio*, *muy_bien* y *demasiado_caliente*. La sección de proceso del código utiliza las instrucciones IF, ELSIF y ELSE para identificar el intervalo en el cual se encuentra la temperatura y asigna a *estado* el patrón de bits correcto. En las últimas tres instrucciones, cada bit de *estado* se conecta al bit del puerto de salida correspondiente.

```
ENTITY fig4 59  IS
PORT(valor digital: IN INTEGER RANGE 0 TO 15;       declara la entrada de 4 bits
      demasiado frio, templado, demasiado caliente : OUT BIT);
END fig4 59 ;

ARCHITECTURE quetancaliente  OF fig4 59 IS
SIGNAL estado   :BIT VECTOR (2 downto 0);
BEGIN
   PROCESS (valor digital)
      BEGIN
         IF (valor digital <5 8) THEN  estado <5 “100”;
         ELSIF (valor digital > 8 AND valor digital < 11) THEN
               estado <= "010";
         ELSE  estado <= "001";
         END IF;
      END PROCESS ;
   demasiado frio    <= estado(2);        asigna los bits de estado a la salida
   templado  <= estado(1);
   demasiado caliente     <= estado(0);
END quetancaliente;
```

FIGURA 4-59 Ejemplo de Intervalos de temperatura en VHDL mediante el uso de ELSIF.

CASE

Existe otra estructura de control que es útil para elegir acciones con base en las condiciones actuales. Tiene varios nombres dependiendo del lenguaje de programación, pero casi siempre se utiliza la palabra **CASE**. Esta instrucción determina el valor de una expresión u objeto y después analiza una lista de posibles valores (casos) para la expresión u objeto que se está evaluando. Cada caso tiene una lista de acciones que deben llevarse a cabo. Una instrucción CASE es distinta a una instrucción IF/ELSIF debido a que un caso correlaciona un valor único de un objeto con un conjunto de acciones. Recuerde que una instrucción IF/ELSIF correlaciona un conjunto de acciones con una aseveración verdadera. Sólo puede haber una coincidencia para una instrucción CASE, mientras que una instrucción IF/ELSIF puede tener más de una, pero entonces (THEN) realizará solo la acción asociada con la primera aseveración verdadera que evalúe.

Otro punto importante en los ejemplos que siguen a continuación es la necesidad de combinar cierto número de variables independientes en un conjunto de bits, al cual se le conoce como vector de bits. Recuerde que a esta acción de enlazar varios bits en un orden específico se le conoce como *concatenación* y nos permite considerar el patrón de bits como un grupo ordenado.

AHDL

CASE MEDIANTE EL USO DE AHDL

El ejemplo de AHDL en la figura 4-60 muestra el uso de una instrucción CASE para implementar el circuito de la figura 4-9 (vea también la tabla 4-3). Utiliza bits individuales como sus entradas. En la primera instrucción después de BEGIN, estos bits se concatenan y se asignan a la variable intermedia llamada *estado*. La instrucción CASE evalúa a la variable *estado* y busca el patrón de bits (que sigue después de la palabra clave WHEN) que concuerde con el valor de *estado*. Después realiza la acción descrita después del signo =>. En este ejemplo, sólo asigna un 0 lógico a la salida para cada uno de los tres casos especificados. Los *demás* casos producen un 1 lógico en la salida.

FIGURA 4-60
Representación de la figura 4-9 en AHDL.

```
SUBDESIGN fig4 60
(
   p, q, r          :INPUT;          define las entradas del bloque
   s                :OUTPUT;         define las salidas
)
VARIABLE
  estado[2..0]     :NODE;
BEGIN
   estado[]= (p, q, r);      enlaza los bits de entrada en orden
   CASE estado[] IS
      WHEN b"100"       => s = GND;
      WHEN b"101"       => s = GND;
      WHEN b"110"       => s = GND;
      WHEN OTHERS       => s = VCC;
   END CASE;
END;
```

CASE MEDIANTE EL USO DE VHDL

El ejemplo de VHDL en la figura 4-61 demuestra el uso de la instrucción CASE para implementar el circuito de la figura 4-9 (vea también la tabla 4-3). Utiliza bits individuales como entradas. En la primera instrucción después de BEGIN, estos bits se concatenan y se asignan a la variable intermedia llamada *estado* mediante el uso del operador &. La instrucción CASE evalúa la variable *estado* y busca el patrón de bits (que va después de la palabra clave WHEN) que concuerde con el valor de *estado*. Después realiza la acción descrita después del signo =>. En este ejemplo simple, sólo asigna un 0 lógico a la salida para cada uno de los tres casos especificados. Los *demás* casos producen un 1 lógico en la salida.

FIGURA 4-61
Representación de la figura 4-9 en VHDL.

```
ENTITY fig4 61 IS
PORT( p, q, r      :IN bit;              declara 3 bits de entrada
      s            :OUT BIT);
END fig4 61;

ARCHITECTURE copia OF fig4 61 IS
SIGNAL estado      :BIT VECTOR (2 downto 0);
BEGIN
   estado <= p & q & r;                 enlaza los bits en orden.
   PROCESS (estado)
      BEGIN
         CASE estado IS
            WHEN "100" => s <= '0';
            WHEN "101" => s <= '0';
            WHEN "110" => s <= '0';
            WHEN OTHERS =>s <= '1';
         END CASE;
      END PROCESS;
END copia;
```

EJEMPLO 4-34

Un detector en una máquina despachadora acepta monedas de 25, 10 y 5 centavos, y activa la señal digital correspondiente (*Q, D, N*) sólo cuando está presente la moneda correcta. Físicamente es imposible que haya varias monedas presentes al mismo tiempo. Un circuito digital debe utilizar las señales *Q*, *D* y *N* como entradas y debe producir un número binario que represente el valor de la moneda según se muestra en la figura 4-62. Escriba el código en AHDL y VHDL.

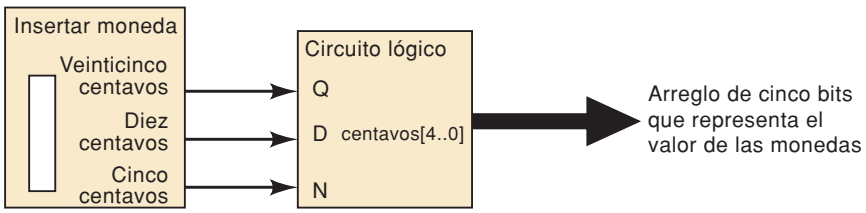


FIGURA 4-62 Un circuito detector de monedas para una máquina despachadora.

Solución

Ésta es una aplicación ideal de la instrucción CASE para describir la operación correcta. Las salidas deben declararse como números de cinco bits para poder representar hasta 25 centavos. La figura 4-63 muestra la solución en AHDL y la figura 4-64 la solución en VHDL.

AHDL

```
SUBDESIGN         fig4_63
(
      q, d, n         :INPUT;      -- define veinticinco, diez y cinco centavos
      centavos[4..0]  :OUTPUT;     -- define el valor binario de las monedas
)
BEGIN
      CASE   (q, d, n)  IS         --agrupa las monedas en un conjunto ordenado
        WHEN b"001" => centavos[ ] = 5;
        WHEN b"010" => centavos[ ] = 10;
        WHEN b"100" => centavos[ ] = 25;
        WHEN OTHERS => centavos[ ] = 0;
      END CASE;
END;
```

FIGURA 4-63 Un detector de monedas en AHDL.

VHDL

```
ENTITY  fig4_64  IS
PORT( q, d, n: IN BIT;                              --veinticinco, diez y cinco centavos
        centavos :OUT INTEGER RANGE 0 TO 25); -- valor binario de las monedas
END fig4_64;
ARCHITECTURE detector OF fig4_64 IS
      SIGNAL  monedas :BIT_VECTOR(2 DOWNTO 0); -- agrupa los sensores de monedas
      BEGIN
         monedas <=  (q & d & n);               --asigna sensores al grupo
         PROCESS  (monedas)
            BEGIN
               CASE  (centavos)  IS
                  WHEN "001"  => centavos <= 5;
                  WHEN "010"  => centavos <= 10;
                  WHEN "100"  => centavos <= 25;
                  WHEN OTHERS => centavos <= 0;
               END CASE;
            END PROCESS;
      END detector;
```

FIGURA 4-64 Un detector de monedas en VHDL.

PREGUNTAS DE REPASO

1. ¿Cuál estructura de control decide qué hacer o qué no hacer?
2. ¿Cuál estructura de control decide hacer esto o lo otro?
3. ¿Cuál(es) estructura(s) de control decide(n) la acción específica, entre varias, que se va a realizar?
4. Declare una entrada llamada *conteo* que pueda representar una cantidad numérica tan grande como 205. Use AHDL o VHDL.

RESUMEN

1. Las dos formas generales para las expresiones lógicas son la forma de suma de productos y la forma de producto de sumas.
2. Un método para el diseño de un circuito lógico combinacional es (1) construir su tabla de verdad, (2) convertir la tabla de verdad en una expresión de suma de productos, (3) simplificar la expresión mediante álgebra booleana o mapeo K, (4) implementar la expresión final.
3. El mapa K es un método gráfico para representar la tabla de verdad de un circuito y generar una expresión simplificada para la salida del circuito.
4. Un circuito OR exclusivo tiene la expresión $x = A\overline{B} + \overline{A}B$. Su salida x estará en ALTO sólo cuando las entradas A y B estén en niveles lógicos opuestos.
5. Un circuito NOR exclusivo tiene la expresión $x = \overline{A}\,\overline{B} + AB$. Su salida x estará en ALTO sólo cuando las entradas A y B estén en el mismo nivel lógico.
6. Cada una de las compuertas básicas (AND, OR, NAND, NOR) pueden usarse para habilitar o deshabilitar el paso de una señal de entrada hacia su salida.
7. Las principales familias de CIs digitales son las familias TTL y CMOS. Los CIs digitales están disponibles en una amplia gama de densidades (compuertas por chip), desde las funciones lógicas básicas hasta las de alta complejidad.
8. Para el diagnóstico de fallas básico se requiere (como mínimo) una comprensión de la operación del circuito, un conocimiento de los tipos de posibles fallas, un diagrama de conexiones del circuito lógico completo y una sonda lógica.
9. Un dispositivo lógico programable (PLD) es un CI que contiene un extenso número de compuertas lógicas cuyas interconexiones pueden ser programadas por el usuario para generar la relación lógica deseada entre las entradas y las salidas.
10. Para programar un PLD se necesita un sistema de desarrollo, el cual consiste de una computadora, software para desarrollo de PLDs y un dispositivo programador que se encarga de la programación física del chip PLD.
11. El sistema Altera permite técnicas convenientes de diseño jerárquico mediante el uso de cualquier forma de descripción de hardware.
12. El tipo de los objetos de datos debe especificarse, de manera que el compilador del HDL conozca el intervalo de números que van a representarse.
13. Las tablas de verdad pueden introducirse de manera directa en el archivo fuente mediante el uso de las características del HDL.
14. Pueden utilizarse las estructuras de control lógicas tales como IF, ELSE y CASE para describir la operación de un circuito lógico, con lo cual se simplifica aún más el código y la solución al problema.

TÉRMINOS IMPORTANTES

agrupamiento
arreglo de bits
arriba-abajo
asignación de señal selecta
bibliotecas
BIT_VECTOR
CASE
cero esfuerzo de inserción (ZIF)
colisión
comprobación de paridad
concatenación
concurrentes
condición de "no importa"
diseño jerárquico
ELSE
ELSIF
encapsulado dual en línea (DIP)
entero
estructuras de control de decisiones
flotante
generación de paridad
habilitar/deshabilitar
IF/THEN
Índice
indeterminado
JEDEC
JTAG
lista de sensibilidad
literales
lógica de transistor/-transistor (TTL)
macrofunción
mapa de Karnaugh (mapa K)
NOR exclusivo (XNOR)
objetos
OR exclusivo (XOR)
PROCESS
producto de sumas (POS)
programador
secuenciales
Semiconductor Metal-Óxido-complementario (CMOS)
sonda lógica
SSI, MSI, LSI, VLSI, ULSI, GSI
STD_LOGIC
STD_LOGIC_VECTOR
suma de productos (SOP)
vector de bits
vectores de prueba

PROBLEMAS

SECCIONES 4-2 Y 4-3

B 4-1.* Simplifique las siguientes expresiones mediante el uso del álgebra booleana.

(a) $x = ABC + \overline{A}C$
(b) $y = (Q + R)(\overline{Q} + \overline{R})$
(c) $w = ABC + A\overline{B}C + \overline{A}$
(d) $q = \overline{RST}(\overline{R + S + T})$
(e) $x = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}BC + ABC + A\,\overline{B}\,\overline{C} + A\overline{B}C$
(f) $z = (B + \overline{C})(\overline{B} + C) + \overline{\overline{A} + B + \overline{C}}$
(g) $y = (\overline{C + D}) + \overline{A}C\overline{D} + A\overline{B}\,\overline{C} + \overline{A}\,\overline{B}CD + AC\overline{D}$
(h) $x = AB(\overline{\overline{C}D}) + \overline{A}BD + \overline{B}\,\overline{C}\,\overline{D}$

B 4-2. Simplifique el circuito de la figura 4-65 mediante el uso del álgebra booleana.

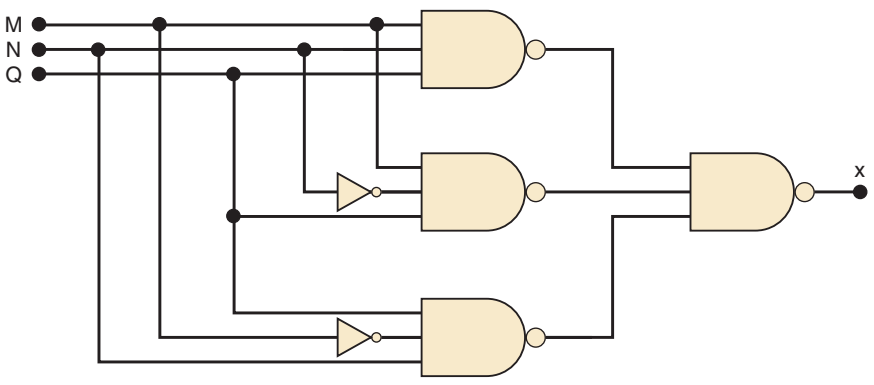


FIGURA 4-65
Problemas 4-2 y 4-3.

* Encontrará las respuestas a los problemas marcados con un asterisco al final del libro.

B 4-3.* Cambie cada una de las compuertas del problema 4-2 por compuertas NOR y simplifique el circuito mediante álgebra booleana.

SECCIÓN 4-4

B, D 4-4.* Diseñe el circuito lógico que corresponde a la tabla de verdad que se muestra en la tabla 4-11.

TABLA 4-11

A	*B*	*C*	*x*
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

B, D 4-5. Diseñe un circuito lógico cuya salida esté en ALTO *sólo* cuando la mayoría de las entradas *A*, *B* y *C* estén en BAJO.

D 4-6. Una planta de manufactura necesita tener un sonido de bocina para indicar la hora de salida. La bocina deberá activarse cuando se cumpla cualquiera de las siguientes condiciones:

1. Es después de las 5 en punto y todas las máquinas están apagadas.
2. Es viernes, se completó la producción del día y todas las máquinas están apagadas.

Diseñe un circuito lógico que controle la bocina. (*Sugerencia:* use cuatro variables lógicas de entrada para representar las diversas condiciones; por ejemplo, la entrada *A* estará en ALTO sólo cuando sean las 5 en punto o más tarde.)

D 4-7.* Un número binario de cuatro bits se representa como $A_3\,A_2\,A_1A_0$, en donde A_3, A_2, A_1 y A_0 representan los bits individuales y A_o es igual al LSB. Diseñe un circuito lógico que produzca una salida en ALTO cada vez que el número binario sea mayor que 0010 y menor que 1000.

D 4-8. La figura 4-66 muestra un diagrama para un circuito de alarma de automóvil que se utiliza para detectar ciertas condiciones indeseables. Los tres inte-

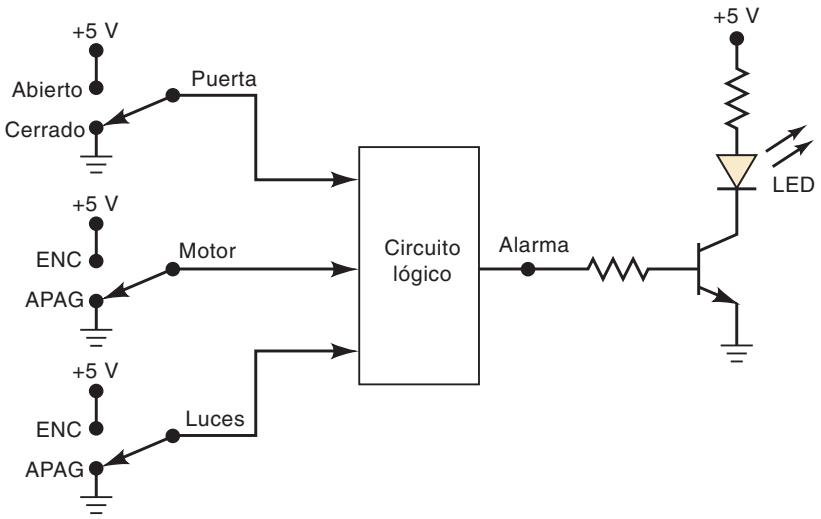


FIGURA 4-66
Problema 4-8.

rruptores se utilizan para indicar el estado de la puerta del lado del conductor, el motor y las luces, en forma respectiva. Diseñe el circuito lógico con estos tres interruptores como entradas, de manera que la alarma se active cada vez que exista cualquiera de las siguientes condiciones:

- Las luces estén encendidas mientras que el motor esté apagado.
- La puerta esté abierta mientras que el motor esté encendido.

4-9.* Implemente el circuito del problema 4-4, utilizando sólo compuertas NAND.

4-10. Implemente el circuito del problema 4-5, utilizando sólo compuertas NAND.

SECCIÓN 4-5

B 4-11. Determine la expresión mínima para cada uno de los mapas K en la figura 4-67. Ponga especial atención al paso 5 para el mapa en (a).

FIGURA 4-67
Problema 4-11.

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1	1	1	1
$\bar{A}B$	1	1	0	0
AB	0	0	0	1
$A\bar{B}$	0	0	1	1

(a)*

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1	0	1	1
$\bar{A}B$	1	0	0	1
AB	0	0	0	0
$A\bar{B}$	1	0	1	1

(b)

	$\bar{C}$	C
$\bar{A}\bar{B}$	1	1
$\bar{A}B$	0	0
AB	1	0
$A\bar{B}$	1	X

(c)

B 4-12. Para la tabla de verdad que se muestra a continuación, cree un mapa K de 2×2, agrupe los términos y simplifique. Después analice de nuevo la tabla de verdad para ver si la expresión es verdadera para todas las entradas en la tabla.

A	*B*	*y*
0	0	1
0	1	1
1	0	0
1	1	0

B 4-13. Empezando con la tabla de verdad en la tabla 4-11, utilice un mapa K para encontrar la ecuación SOP más simple.

B 4-14. Simplifique la expresión en (a)* el problema 4-1(e), usando un mapa K. (b) el problema 4-1(g), usando un mapa K. (c)* el problema 4-1(h), usando un mapa K.

B 4-15.* Obtenga la expresión de salida para el problema 4-7, usando un mapa K.

C,D 4-16. La figura 4-68 muestra un *contador BCD* que produce una salida de cuatro bits, la cual representa el código BCD para el número de pulsos que se han aplicado a la entrada del contador. Por ejemplo, después de haberse producido cuatro pulsos, las salidas del contador son $DCBA = 0100_2 = 4_{10}$. El contador se restablece a 0000 en el décimo pulso y empieza a contar de nuevo. En otras palabras, las salidas *DCBA* nunca representarán a un número mayor de $1001_2 = 9_{10}$.

(a)* Diseñe el circuito lógico que produzca una salida en ALTO cada vez que el conteo sea 2, 3 o 9. Use el mapeo K y aproveche las condiciones "no importa".

(b) Repita el proceso para $x = 1$ cuando $DCBA = 3, 4, 5, 8$.

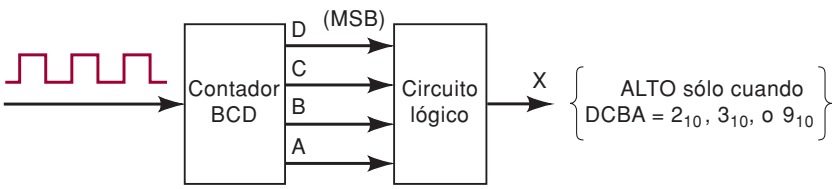


FIGURA 4-68
Problema 4-16.

D 4-17.* La figura 4-69 muestra cuatro interruptores que forman parte de los circuitos de control en una máquina copiadora. Los interruptores están en varios puntos a lo largo de la ruta del papel, a medida que éste pasa a través de la máquina. Cada interruptor está, por lo general, abierto, y a medida que el papel pasa a través de un interruptor, éste se cierra. Es imposible que los interruptores SW1 y SW4 estén cerrados al mismo tiempo. Diseñe el circuito lógico para producir una salida en ALTO cada vez que *dos o más* interruptores estén cerrados al mismo tiempo. Use el mapeo K y aproveche las condiciones "no importa".

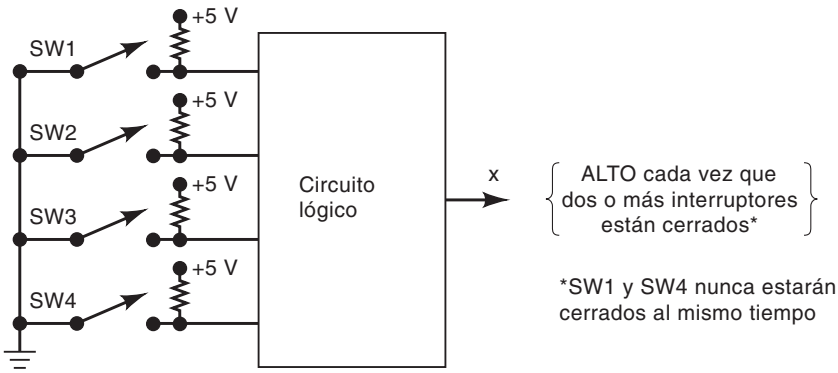


FIGURA 4-69
Problema 4-17.

B 4-18. El ejemplo 4-3 demostró la simplificación algebraica. El paso 3 produjo la ecuación SOP $z = \overline{A}\,\overline{B}\,C + \overline{A}\,C\,\overline{D} + \overline{A}\,B\,\overline{C}\,\overline{D} + A\overline{B}C$. Use un mapa K para demostrar que esta ecuación puede simplificarse aún más que la respuesta que se muestra en el ejemplo.

C 4-19. Utilice el álgebra booleana para llegar al mismo resultado que se obtuvo mediante el método del mapa K del problema 4-18.

SECCIÓN 4-6

B 4-20. (a) Determine la forma de onda de salida para el circuito de la figura 4-70.

(b) Repita el proceso con la entrada *B* mantenida en BAJO.

(c) Repita el proceso con la entrada *B* mantenida en ALTO.

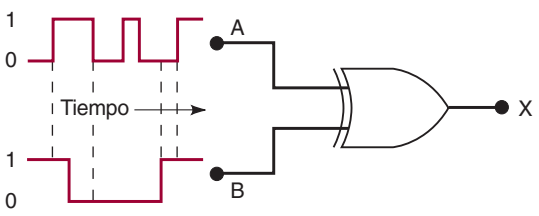
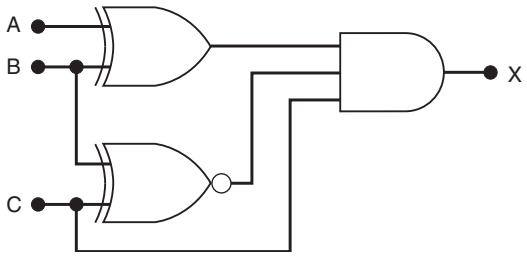


FIGURA 4-70 Problema 4-20.

B 4-21.* Determine las condiciones de entrada necesarias para producir $x = 1$ en la figura 4-71.

FIGURA 4-71 Problema 4-21.



B 4-22. Diseñe un circuito que produzca una salida en ALTO sólo cuando las tres entradas estén en el mismo nivel.

(a) Use una tabla de verdad y un mapa K para producir la solución SOP.

(b) Use compuertas XOR de dos entradas y otras compuertas para encontrar una solución. (*Sugerencia:* recuerde la propiedad transitiva del álgebra… si $a = b$ y $b = c$ entonces $a = c$.)

B 4-23.* Un chip 7486 contiene cuatro compuertas XOR. Muestre cómo hacer una compuerta XNOR utilizando sólo un chip 7486. *Sugerencia:* vea el ejemplo 4-16.

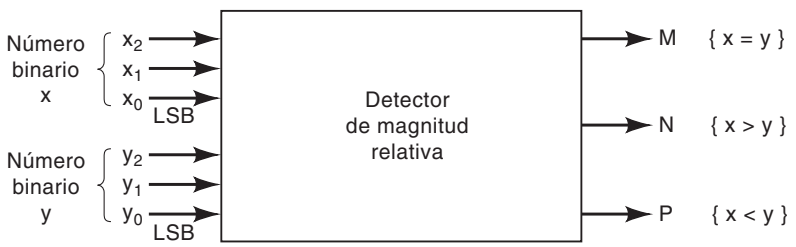
B 4-24.* Modifique el circuito de la figura 4-23 para comparar dos números de cuatro bits y producir una salida en ALTO cuando los dos números concuerden de manera exacta.

B 4-25. La figura 4-72 representa un *detector de magnitud relativa* que toma dos números binarios de tres bits ($x_2x_1x_0$ y $y_2y_1y_0$) y determine si son iguales; en caso de no ser así, que determine cuál es más grande. Hay tres salidas, que se definen de la siguiente manera:

1. $M = 1$ sólo si los dos números de entrada son iguales.
2. $N = 1$ sólo si $x_2x_1x_0$ es mayor que $y_2y_1y_0$.
3. $P = 1$ sólo si $y_2y_1y_0$ es mayor que $x_2x_1x_0$.

Diseñe los circuitos lógicos para este detector. El circuito tiene *seis* entradas y *tres* salidas, por lo que es demasiado complejo como para manejarlo mediante el método de la tabla de verdad. Consulte el ejemplo 4-17 como una sugerencia sobre cómo podría empezar a resolver este problema.

FIGURA 4-72
Problema 4-25.



MÁS PROBLEMAS DE DISEÑO

C,D 4-26.* La figura 4-73 representa un circuito multiplicador que toma dos números binarios de dos bits (x_1x_0 y y_1y_0) y produce un número binario de salida $z_3z_2z_1z_0$ que es igual al producto aritmético de los dos números de entrada. Diseñe el circuito lógico para el multiplicador. (*Sugerencia:* el circuito lógico tendrá cuatro entradas y cuatro salidas.)

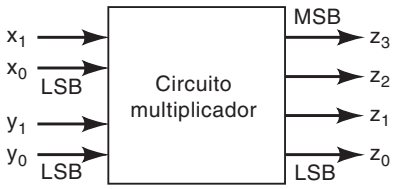


FIGURA 4-73 Problema 4-26.

D 4-27. Un código BCD se está transmitiendo a un receptor remoto. Los bits son A_3, A_2, A_1 y A_0, en donde A_3 es el MSB. Entre los circuitos del receptor incluye un circuito *detector de errores BCD*, el cual examina el código recibido para ver si es un código de BCD legal (es decir, ≤1001). Diseñe este circuito para producir un nivel ALTO para cualquier condición de error.

D 4-28.* Diseñe un circuito lógico cuya salida esté en ALTO cada vez que *A* y *B* estén ambas en ALTO, siempre y cuando *C* y *D* estén ambas en BAJO o ambas en ALTO. Trate de hacer esto sin utilizar una tabla de verdad. Después compruebe su resultado construyendo una tabla de verdad a partir de su circuito, para ver si concuerda con la declaración del problema.

D 4-29. Cuatro tanques grandes en una planta química que contienen distintos líquidos se están calentando. Se utilizan sensores de nivel de líquido para detectar cuando el tanque *A* o el tanque *B* se eleva por encima de un nivel predeterminado. Los sensores de temperatura en los tanques *C* y *D* detectan cuando la temperatura en cualquiera de estos tanques cae por debajo de un límite prescrito. Suponga que las salidas *A* y *B* del sensor de nivel de líquido están en BAJO cuando el nivel es satisfactorio y en ALTO cuando el nivel es demasiado alto. Además, las salidas *C* y *D* del sensor de temperatura están en BAJO cuando la temperatura es satisfactoria y en ALTO cuando la temperatura es demasiado baja. Diseñe un circuito lógico que detecte cada vez que el nivel en el tanque *A* o en el tanque *B* es demasiado alto, al mismo tiempo que la temperatura en el tanque *C* o en el tanque *D* sea demasiado baja.

C,D 4-30.* La figura 4-74 muestra la intersección de una autopista principal con un camino de acceso secundario. Se colocaron sensores de detección de vehículos a lo largo de los carriles *C* y *D* (camino principal) y de los carriles *A* y *B* (camino de acceso). Las salidas de estos sensores están en BAJO (0) cuando

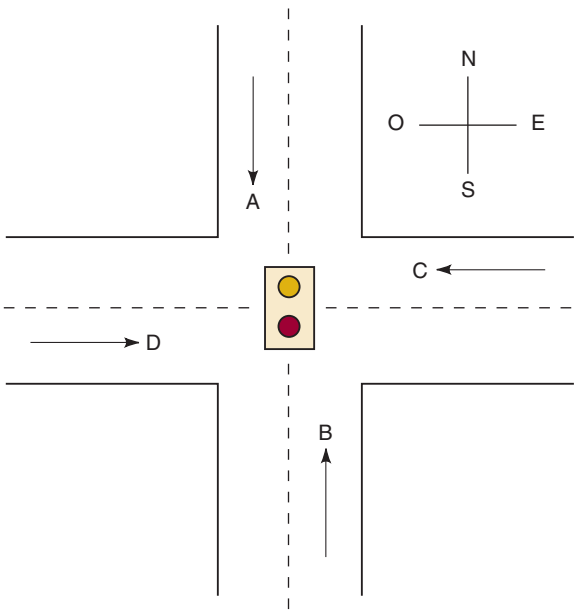


FIGURA 4-74
Problema 4-30.

no hay vehículos presentes, y en ALTO (1) cuando hay vehículos presentes. El semáforo de la intersección debe controlarse de acuerdo con la siguiente lógica:

1. El semáforo este-oeste (E-O) se pondrá en verde cada vez que estén ocupados ambos carriles *C* y *D*.
2. El semáforo E-O estará en verde cada vez que *C o D* estén ocupados, pero cuando *A* y *B* no estén ambos ocupados.
3. El semáforo norte-sur (N-S) se pondrá en verde cada vez que *ambos* carriles *A* y *B* estén ocupados, pero cuando *C* y *D* no estén *ambos* ocupados.
4. El semáforo N-S también se pondrá en verde cuando *A* o *B* estén ocupados, mientras que *C* y *D* estén *ambos* vacantes.
5. El semáforo E-O cuando *no* haya vehículos presentes.

Utilizando las salidas del sensor *A*, *B*, *C* y *D* como entradas, diseñe un circuito lógico para controlar el semáforo. Debe haber dos salidas, N-S y E-O, que cambien a ALTO cuando la luz correspondiente se vaya a poner en *verde*. Simplifique el circuito lo más que se pueda y muestre *todos* los pasos.

SECCIÓN 4-7

D 4-31. Rediseñe el generador y comprobador de paridad de la figura 4-25 para que (a) opere usando paridad impar. (*Sugerencia:* ¿cuál es la relación entre un bit de paridad impar y un bit de paridad par para el mismo conjunto de bits de datos?) (b) Opere con ocho bits de datos.

SECCIÓN 4-8

B 4-32. (a) ¿Bajo qué condiciones permitirá una compuerta OR que una señal lógica pase hacia su salida sin modificarla?

(b) Repita el inciso (a) para una compuerta AND.

(c) Repita el proceso para una compuerta NAND.

(d) Repita el proceso para una compuerta NOR.

B 4-33.* (a) ¿Puede utilizarse un INVERSOR como un circuito de habilitación/deshabilitación? Explique.

(b) ¿Puede usarse una compuerta XOR como un circuito de habilitación/deshabilitación? Explique.

D 4-34. Diseñe un circuito lógico que permita que la señal de entrada *A* pase hasta la salida sólo cuando la entrada de control *B* esté en BAJO, mientras que la entrada de control *C* esté en ALTO; en caso contrario, la salida debe estar en BAJO.

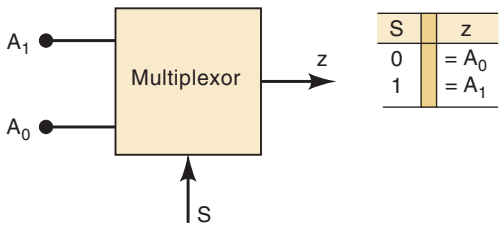
D 4-35.* Diseñe un circuito que *deshabilite* el paso de una señal de entrada sólo cuando las entradas de control *B*, *C* y *D* estén todas en ALTO; la salida deberá estar en ALTO para la condición deshabilitada.

D 4-36. Diseñe un circuito lógico que controle el paso de la señal *A*, de acuerdo con los siguientes requerimientos:

1. La salida *X* será igual a *A* cuando las entradas de control *B* y *C* sean iguales.
2. *X* permanecerá en ALTO cuando *B* y *C* sean distintas.

D 4-37. Diseñe un circuito lógico que tenga dos señales de entrada A_1 y A_0, y una entrada de control *S* de manera que funcione de acuerdo con los requerimientos descritos en la figura 4-75. (A este tipo de circuito se le conoce como *multiplexor*, el cual veremos en el capítulo 9.)

FIGURA 4-75 Problema 4-37.



D 4-38.* Use el mapeo K para diseñar un circuito que cumpla con los requerimientos del ejemplo 4-17. Compare este circuito con la solución en la figura 4-23. Aquí se recalca que el método del mapa K no puede aprovechar la lógica de las compuertas XOR y XNOR. El diseñador debe ser capaz de determinar cuándo pueden utilizarse estas compuertas.

SECCIONES 4-9 A 4-13

F 4-39. (a) Un técnico que está probando un circuito lógico se da cuenta de que la salida de cierto INVERSOR está atascada en BAJO mientras que su entrada esté pulsando. Liste todas las posibles razones que pueda para esta operación defectuosa.

(b) Repita el inciso (a) para el caso en el que la salida del INVERSOR se quede atascada en un nivel lógico indeterminado.

F 4-40.* Las señales que se muestran en la figura 4-76 se aplican a las entradas del circuito de la figura 4-32. Suponga que hay un circuito abierto interno en Z1-4.

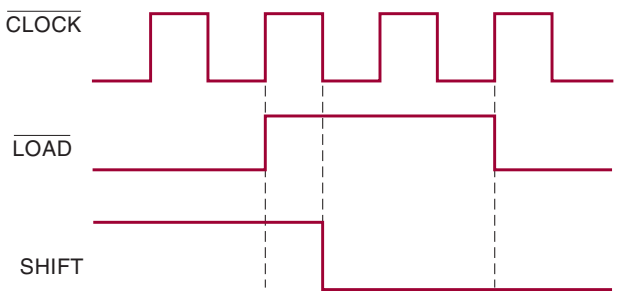
(a) ¿Qué indicará una sonda lógica en Z1-4?

(b) ¿Qué lectura de voltaje de corriente directa esperaría en Z1-4? (Recuerde que los CIs son TTL.)

(c) Haga un bosquejo de cuál cree usted que será la apariencia de las señales $\overline{CLKOUT}$ y $\overline{SHIFTOUT}$.

(d) En vez del circuito abierto en Z1-4, suponga que las terminales 9 y 10 de Z2 están en corto interno. Haga un bosquejo de las probables señales en Z2-10, $\overline{CLOCKOUT}$ y $\overline{SHIFTOUT}$.

FIGURA 4-76 Problema 4-40.



F 4-41. Suponga que los CIs de la figura 4-32 son CMOS. Describa cómo se vería afectada la operación del circuito debido a un circuito abierto en el conductor que conecta a Z2-2 y Z2-10.

F 4-42. En el ejemplo 4-24 listamos tres posibles fallas para la situación de la figura 4-36. ¿Qué procedimiento seguiría usted para determinar cuál de las fallas es la que se está produciendo en realidad?

F 4-43.* Consulte el circuito de la figura 4-38. Suponga que los dispositivos son CMOS. Suponga además que la indicación de la sonda lógica en Z2-3 es

* Recuerde que F indica un ejercicio de diagnóstico de fallas.

"indeterminado", en vez de "pulsando". Liste las posibles fallas y escriba un procedimiento a seguir para determinar la verdadera falla?

F 4-44.* Consulte el circuito lógico de la figura 4-41. Recuerde que se supone que la salida Y debe estar en ALTO para cualquiera de las siguientes condiciones:

1. $A = 1, B = 0$, sin importar C
2. $A = 0, B = 1, C = 1$

Al probar el circuito, el técnico observa que Y cambia a ALTO sólo para la primera condición, pero permanece en BAJO para todas las demás condiciones de entrada. Considere la siguiente lista de posibles fallas. Para cada una de ellas, escriba sí o no para indicar si podría o no ser la verdadera falla. Explique su razonamiento para cada una de las opciones en las que conteste que no.

(a) Un corto interno a tierra en Z2-13.
(b) Un circuito abierto en la conexión a Z2-13.
(c) Un corto interno con V_{CC} en Z2-11.
(d) Un circuito abierto en la conexión de V_{CC} con Z2.
(e) Un circuito abierto interno en Z2-9.
(f) Un circuito abierto en la conexión de Z2-11 a Z2-9.
(g) Un puente de soldadura entre las terminales 6 y 7 de Z2.

F 4-45. Desarrolle un procedimiento para aislar la falla que esté produciendo el funcionamiento defectuoso descrito en el problema 4-44.

F 4-46.* Suponga que todas las compuertas en la figura 4-41 son CMOS. Cuando el técnico prueba el circuito descubre que opera en forma correcta, excepto para las siguientes condiciones:

1. $A = 1, B = 0, C = 0$
2. $A = 0, B = 1, C = 1$

Para estas condiciones, la sonda lógica indica niveles indeterminados en Z2-6, Z2-11 y Z2-8. ¿Cuál cree usted que sea la probable falla en el circuito? Explique su razonamiento.

F 4-47. La figura 4-77 es un circuito lógico combinacional que opera una alarma en un automóvil, cada vez que están ocupados los asientos del conductor y/o del pasajero y que los cinturones de seguridad no están abrochados cuando el automóvil arranca. Las señales *CONDUC* y *PASAJ* activas en ALTO indican la presencia del conductor y del pasajero en forma respectiva, y se obtienen mediante interruptores operados por presión en los asientos. La señal *ENC* es activa en ALTO cuando el interruptor de encendido está activado. La señal $\overline{CINTC}$ es activa en BAJO e indica que el cinturón de seguridad

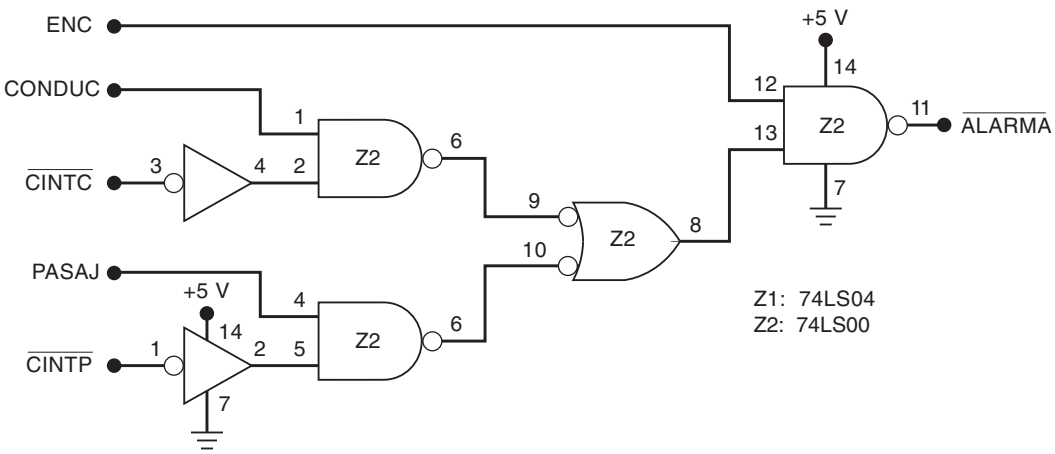


FIGURA 4-77 Problemas 4-47, 4-48 y 4-49.

del conductor *no está abrochado*; $\overline{CINTP}$ es la señal correspondiente para el cinturón de seguridad del pasajero. La alarma se activará (BAJO) cada vez que se encienda el automóvil, que cualquiera de los asientos delanteros esté ocupado y que su cinturón de seguridad no esté abrochado.

(a) Verifique que el circuito funcione de la manera descrita.

(b) Describa cómo operaría este sistema de alarma si Z1-2 estuviera en corto interno con tierra.

(c) Describa cómo operaría si hubiera una conexión abierta de Z2-6 a Z2-10.

F 4-48.* Suponga que el sistema de la figura 4-77 está funcionando de manera que la alarma se active tan pronto como el conductor o algún pasajero estén sentados y el automóvil esté encendido, sin importar el estado de los cinturones de seguridad. ¿Cuáles son las posibles fallas? ¿Qué procedimiento seguiría usted para encontrar la verdadera falla?

F 4.49.* Suponga que el sistema de alarma de la figura 4-77 está operando de manera que la alarma se encienda en forma continua tan pronto como se encienda el automóvil, sin importar el estado de las demás entradas. Liste las posibles fallas y escriba un procedimiento para aislar la falla.

PREGUNTAS DE PRÁCTICA SOBRE PLDS (50 A 55)

4-50.* *Verdadero o falso*:

(a) El diseño de arriba hacia abajo comienza con una descripción general de todo el sistema y sus especificaciones.

(b) Un archivo JEDEC puede usarse como archivo de entrada para un programador.

(c) Si un archivo de entrada se compila sin errores, significa que el circuito PLD funcionará en forma correcta.

(d) Un compilador puede interpretar código a pesar de los errores de sintaxis.

(e) Los vectores de prueba se utilizan para simular y probar un dispositivo.

H,B 4-51. ¿Qué son los caracteres % que se utilizan para el archivo de diseño de AHDL?

H,B 4-52. ¿Cómo se indican los comentarios en un archivo de diseño de VHDL?

B 4-53. ¿Qué es un zócalo ZIF?

B 4-54.* Enliste tres modos de entrada utilizados para introducir la descripción de un circuito en el software de desarrollo de PLDs.

B 4-55. ¿Qué significan JEDEC y HDL?

SECCIÓN 4-15

H,B 4-56. Declare los siguientes objetos de datos en AHDL o VHDL.

(a)* Un arreglo de ocho bits de salida llamados *aparatos*.

(b) Un bit individual de salida llamado *chicharra*.

(c) Un puerto de entrada numérico de 16 bits llamado *altitud*.

(d) Un bit individual intermedio dentro de un archivo de descripción de hardware llamado *alambre2*.

H,B 4-57. Exprese los siguientes números literales en hexadecimal, binario y decimal, utilizando la sintaxis de AHDL o VHDL.

(a)* 152_{10}

(b) 1001010100_2

(c) $3C4_{16}$

H,B 4-58.* La siguiente definición de E/S que se da para AHDL y para VHDL. Escriba cuatro instrucciones de asignación concurrentes que conecten las entradas con las salidas, como se muestra en la figura 4-78.

```
SUBDESIGN hw
(
    bitsent[3..0]   :INPUT;
    bitssal[3..0]   :OUTPUT;
)
```

```
ENTITY hw IS
PORT (
     bitsent     :IN BIT VECTOR (3 downto 0);
     bitssal     :OUT BIT VECTOR (3 downto 0)
     );
END hw;
```

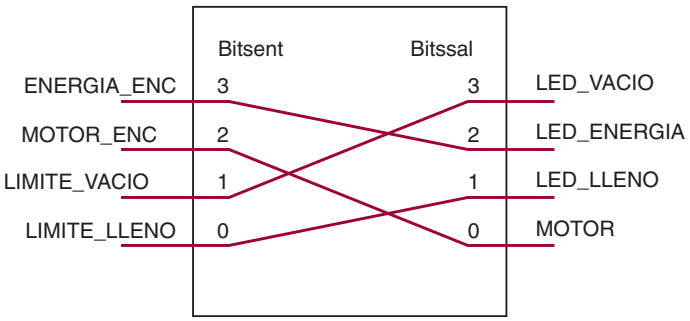


FIGURA 4-78
Problema 4-58.

SECCIÓN 4-16

H,D 4-59. Modifique la tabla de verdad de AHDL de la figura 4-50 para implementar la ecuación $AB + A\overline{C} + \overline{A}B$.

H,D 4-60.* Modifique el diseño de AHDL en la figura 4-54, de manera que $z = 1$ sólo cuando el valor digital sea menor que 1010_2.

H,D 4-61. Modifique la tabla de verdad de AHDL de la figura 4-51 para implementar $AB + A\overline{C} + \overline{A}B$.

H,D 4-62.* Modifique el diseño de VHDL de la figura 4-55, de manera que $z = 1$ sólo cuando el valor digital sea menor que 1010_2.

H,B 4-63. Modifique el código de (a) la figura 4-54 o (b) la figura 4-55 de tal forma que la salida z esté en BAJO sólo cuando valor_digital se encuentre entre 6 y 11 (inclusivo).

H,D 4-64. Modifique (a) el diseño de AHDL de la figura 4-60 para implementar la tabla 4-1. (b) el diseño de VHDL de la figura 4-61 para implementar la tabla 4-1.

H,D 4-65.* Escriba la ecuación booleana del archivo de diseño de descripción de hardware para implementar el ejemplo 4-9.

4-66. Escriba la ecuación booleana del archivo de diseño de descripción de hardware para implementar un generador de paridad de cuatro bits, como se muestra en la figura 4-25(a).

PREGUNTA DE PRÁCTICA

B 4-67. Defina cada uno de los siguientes términos.

(a) Mapa de Karnaugh.
(b) Forma de suma de productos.
(c) Generador de paridad.
(d) Octeto.

(e) Circuito de habilitación.
(f) Condición "no importa".
(g) Entrada flotante.
(h) Nivel de voltaje indeterminado.
(i) Colisión.
(j) PLD.
(k) TTL.
(l) CMOS.

APLICACIONES DE MICROCOMPUTADORA

C 4-68. En una microcomputadora, la unidad del microprocesador (MPU) siempre se está comunicando con uno de los siguientes elementos: (1) memoria de acceso aleatorio (RAM), la cual almacena los programas y datos que pueden modificarse con facilidad; (2) memoria de sólo lectura (ROM), la cual almacena programas y datos que nunca se modifican; y (3) dispositivos externos de entrada/salida (E/S) tales como teclados, pantallas de video, impresoras y unidades de disco. Al ejecutar un programa, la MPU genera un código de dirección que selecciona el tipo de dispositivo (RAM, ROM o E/S) con el que desea comunicarse. La figura 4-79 muestra un arreglo común en donde la MPU produce como salida un código de dirección de ocho bits ($A_{15} - A_8$). En realidad la MPU produce como salida un código de dirección de 16 bits, pero los bits de menor orden ($A_7 - A_0$) no se utilizan en el proceso de selección de dispositivos. El código de dirección se aplica a un circuito lógico que lo utiliza para generar las señales de selección de dispositivos: $\overline{RAM}$, $\overline{ROM}$ y $\overline{E/S}$.

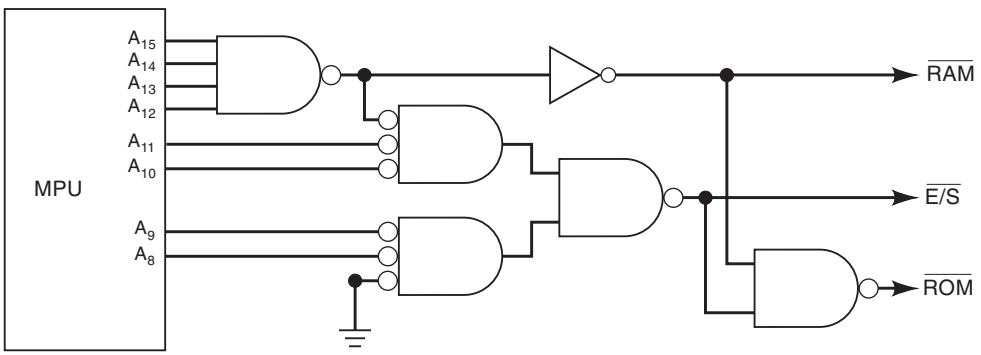


FIGURA 4-79
Problema 4-68.

Analice este circuito y determine lo siguiente:.

(a)* El intervalo de direcciones de A_{15} hasta A_8 que activa la señal $\overline{RAM}$.
(b) El intervalo de direcciones que activa la señal $\overline{E/S}$.
(c) El intervalo de direcciones que activa la señal $\overline{ROM}$.

Exprese las direcciones en binario y en hexadecimal. Por ejemplo, la respuesta para (a) es: A_{15} a $A_8 = 00000000_2$ a $11101111_2 = 00_{16}$ a EF_{16}.

C,D 4-69. En algunas microcomputadoras la MPU puede *deshabilitarse* por breves periodos mientras que otro dispositivo controla los dispositivos de RAM, ROM y E/S. Durante estos intervalos, la MPU activa una señal de control especial ($\overline{DMA}$), la cual se utiliza para deshabilitar (desactivar) la lógica de selección de dispositivos, de manera que las señales $\overline{RAM}$, $\overline{ROM}$ y $\overline{E/S}$ se encuentren en su estado inactivo. Modifique el circuito de la figura 4-79, de tal forma que $\overline{RAM}$, $\overline{ROM}$ y $\overline{E/S}$ se desactiven cada vez que la señal $\overline{DMA}$ esté activa, sin importar el estado del código de dirección.

RESPUESTAS A LAS PREGUNTAS DE REPASO DE LAS SECCIONES

SECCIÓN 4-1

1. Sólo (a). 2. Sólo (c).

SECCIÓN 4-3

1. La expresión (b) no se encuentra en la forma de suma de productos debido al signo de inversión sobre las variables C y D (es decir, el término $A\overline{CD}$). La expresión (c) no se encuentra en la forma de suma de productos debido al término $(M + \overline{N})P$.
3. $x = \overline{A} + \overline{B} + \overline{C}$

SECCIÓN 4-4

1. $x = \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}C\overline{D} + \overline{A}B\overline{C}\,\overline{D}$ 2. Ocho

SECCIÓN 4-5

1. $x = AB + AC + BC$ 2. $x = A + BCD$ 3. $S = \overline{P} + QR$ 4. Una condición de entrada para la cual no haya una condición de salida requerida específica; es decir, tenemos la libertad de hacerla 1 o 0.

SECCIÓN 4-6

2. Un nivel BAJO constante. 3. No; la compuerta XOR disponible puede utilizarse como un INVERSOR si se conecta una de sus entradas a un nivel ALTO constante (vea el ejemplo 4-16).

SECCIÓN 4-8

1. $x = \overline{A(B \oplus C)}$ 2. OR, NAND 3. NAND, NOR

SECCIÓN 4-9

1. DIP 2. SSI, MSI, LSI, VLSI, ULSI, GSI 3. Verdadero. 4. Verdadero.
5. series 40, 74AC y 74ACT 6. De 0 a 0-8 V; de 2.0 a 5-0 V 7. De 0 a 1.5 V; de 3.5 a 5.0 V 8. Como si la entrada estuviera en ALTO. 9. Es impredecible; podría sobrecalentarse y destruirse. 10. 74HCT y 74ACT 11. Describen la manera exacta de cómo interconectar los chips para distribuir el circuito y diagnosticar fallas.
12. Se definen las entradas y las salidas, y se describen las relaciones lógicas.

SECCIÓN 4-11

1. Entradas o salidas abiertas; entradas o salidas en corto con V_{CC}; entradas o salidas cortocircuitadas a tierra; terminales en corto; fallas internas del circuito. 2. Terminales en corto. 3. Para TTL, un nivel BAJO; para CMOS, indeterminado. 4. Dos o más salidas conectadas entre sí.

SECCIÓN 4-12

1. Líneas de señal abiertas; líneas de señal en corto; fuente de poder defectuosa; carga en la salida. 2. Alambres rotos; conexiones de soldadura pobres; grietas o cortes en la tarjeta de circuito impreso; terminales del CI dobladas o rotas; zócalos de CI defectuosos.
3. Los CIs operan en forma errática o no operan. 4. Nivel lógico indeterminado.

SECCIÓN 4-14

1. Las conexiones controladas por electricidad se programan como abiertas o cerradas. 2. (4, 1)(2, 2) o (2, 1)(4, 2) 3. (4, 5)(1, 6) o (4, 6)(1, 5)

SECCIÓN 4-15

1. (a) botones_pulsar[5..0] :INPUT; (b) botones_pulsar :IN BIT_VECTOR (5 DOWNTO 0);
2. (a) z = botones_pulsar[5]; (b) z <= botones_pulsar(5); 3. STD_LOGIC
4. STD_LOGIC_VECTOR

SECCIÓN 4-16

1. (AHDL) omega[ ] = (x, y, z); (VHDL) omega <= x & y & z; 2. Mediante el uso de la palabra clave TABLE. 3. Mediante el uso de las asignaciones de señal selecta.

SECCIÓN 4-17

1. IF/THEN 2. IF/THEN/ELSE 3. CASE o IF/ELSIF
4. (AHDL) conteo[7..0] :INPUT; (VHDL) conteo: IN INTEGER RANGE 0 TO 205

TEOREMAS BOOLEANOS

1. $x \cdot 0 = 0$
2. $x \cdot 1 = x$
3. $x \cdot x = x$
4. $x \cdot \bar{x} = 0$
5. $x + 0 = x$
6. $x + 1 = 1$
7. $x + x = x$
8. $x + \bar{x} = 1$
9. $x + y = y + x$
10. $x \cdot y = y \cdot x$
11. $x + (y + z) = (x + y) + z = x + y + z$
12. $x(yz) = (xy)z = xyz$

13a. $x(y + z) = xy + xz$

13b. $(w + x)(y + z) = wy + xy + wz + xz$

14. $x + xy = x$

15a. $x + \bar{x}y = x + y$

15b. $\bar{x} + xy = \bar{x} + y$

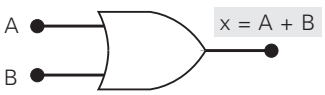
16. $\overline{x + y} = \bar{x}\,\bar{y}$

17. $\overline{xy} = \bar{x} + \bar{y}$

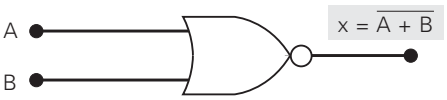
TABLAS DE VERDAD DE LAS COMPUERTAS LÓGICAS

A	B	OR $A + B$	NOR $\overline{A + B}$	AND $A \cdot B$	NAND $\overline{A \cdot B}$	XOR $A \oplus B$	XNOR $\overline{A \oplus B}$
0	0	0	1	0	1	0	1
0	1	1	0	0	1	1	0
1	0	1	0	0	1	1	0
1	1	1	0	1	0	0	1

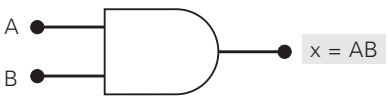
SÍMBOLOS DE COMPUERTAS LÓGICAS



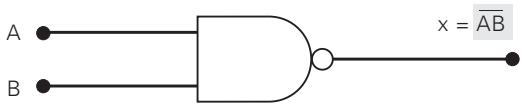
Compuerta OR



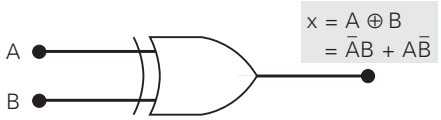
Compuerta NOR



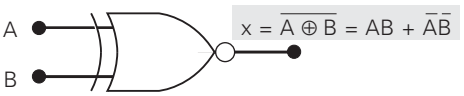
Compuerta AND



Compuerta NAND



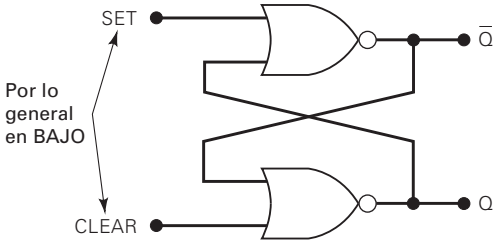
XOR

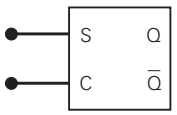


XNOR

FLIP-FLOPS

Latch NOR

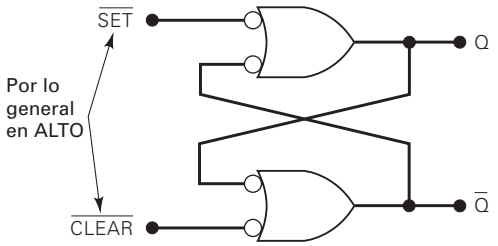


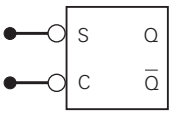


(Símbolo alternativo)

S	C	Q
0	0	Sin cambio
1	0	Q = 1
0	1	Q = 0
1	1	Inválido

Latch NAND

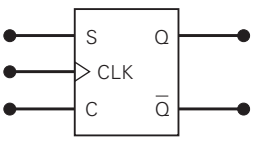




(Símbolo alternativo)

S	C	Q
0	0	Inválido
1	0	Q = 0
0	1	Q = 1
1	1	Sin cambio

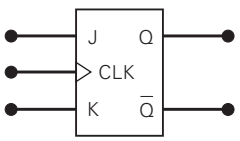
Sincronizado por reloj en S-C



S	C	CLK	Q
0	0	↑	Q_0 Sin cambio
1	0	↑	1
0	1	↑	0
1	1	↑	Ambiguo

↓ de CLK no tiene efecto sobre Q

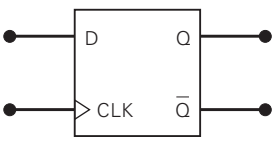
Sincronizado por reloj en J-K



J	K	CLK	Q
0	0	↑	Q_0 Sin cambio
1	0	↑	1
0	1	↑	0
1	1	↑	$\overline{Q_0}$ (conmuta)

↓ de CLK no tiene efecto sobre Q

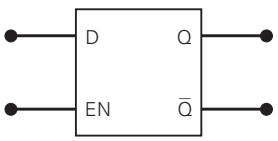
Sincronizado por reloj en D



D	CLK	Q
0	↑	0
1	↑	1

↓ de CLK no tiene efecto sobre Q

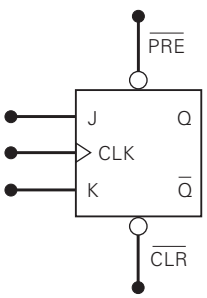
Latch D



EN	D	Q*
0	X	Sin cambio
1	0	0
1	1	1

*Q sigue la entrada D cuando EN está en ALTO

Entradas asíncronas



$\overline{PRE}$	$\overline{CLR}$	Q*
1	1	Sin efecto; el FF puede responder a J, K y CLK
1	0	Q = 0 sin importar J, K, CLK
0	1	Q = 1 1 sin importar J, K, CLK
0	0	Ambigua (no se utiliza)

*CLK puede encontrarse en cualquier estado