

APÉNDICE B

Lógica digital

B.1. Álgebra de Boole

B.2. Puertas

B.3. Circuitos combinacionales

Implementación de las funciones booleanas

Multiplexores

Decodificadores

Array lógico programable

Memoria de solo lectura

Sumadores

B.4. Circuitos secuenciales

Biestables

Registros

Contadores

B.5. Lecturas recomendadas y sitios web

Sitio Web recomendado

B.6. Problemas

El funcionamiento de los computadores digitales se basa en la memorización y procesamiento de datos binarios. A lo largo de este libro, hemos supuesto la existencia de elementos de memoria que pueden estar en uno de dos estados estables y de circuitos que pueden operar con datos binarios bajo la acción de señales de control para implementar distintas funciones. En este apéndice, sugerimos cómo se pueden implementar estos elementos de memoria y circuitos, en lógica digital, concretamente con circuitos combinacionales y secuenciales. El apéndice comienza con un breve repaso del álgebra de Boole, que es el fundamento matemático de la lógica digital. Luego presentaremos el concepto de puerta. Finalmente, se describen los circuitos combinacionales y secuenciales, que se construyen con puertas.

B.1. ÁLGEBRA DE BOOLE

La circuitería digital en computadores digitales y otros sistemas digitales, se diseña y se analiza con el uso de una disciplina matemática denominada *álgebra de Boole*. El nombre es en honor al matemático inglés George Boole, que propuso los principios básicos de este álgebra en 1854 en su tratado, *An investigation of the laws of thought on which to found the mathematical theories of logic and probabilities*. En 1938, Claude Shannon, un investigador asistente en el departamento de Ingeniería Eléctrica del M.I.T., sugirió que el álgebra de Boole podría usarse para resolver problemas de diseño de circuitos de conmutación [SHAN38]. Las técnicas de Shannon se usaron, consecuentemente, en el análisis y diseño de circuitos electrónicos digitales. El álgebra de Boole resulta ser una herramienta útil en dos áreas:

- **Análisis:** es una forma concisa de describir el funcionamiento de los circuitos digitales.
- **Diseño:** dada una función deseada, se puede aplicar el álgebra de Boole para desarrollar una implementación de complejidad simplificada de esta función.

Como con cualquier álgebra, el álgebra de Boole usa variables y operaciones. En este caso, las variables y las operaciones son lógicas. Por tanto, una variable puede tomar el valor 1 (VERDADERO) o 0 (FALSO). Las operaciones lógicas básicas son AND, OR y NOT, que se representan simbólicamente con los signos punto, más y rayado superior:

$$\begin{aligned}A \text{ AND } B &= A \cdot B \\A \text{ OR } B &= A + B \\ \text{NOT } A &= \bar{A}\end{aligned}$$

La operación AND es verdadera (valor binario 1) si y solo si los dos operandos son verdaderos. El resultado de la operación OR es verdad si y solo si uno o ambos operandos son verdad. La operación unitaria NOT invierte el valor del operando. Por ejemplo, consideremos la ecuación

$$D = A + (\bar{B} \cdot C)$$

D es igual a 1 si A es 1 o si B = 0 y C = 1. En otro caso D es igual a 0.

Se necesitan varias aclaraciones en relación con la notación. En ausencia de paréntesis, la operación AND es preferente a la operación OR. Además, cuando no hay ambigüedad, la operación AND se representa con una simple concatenación en lugar de con el operador punto. Por tanto,

Tabla B.1. Operaciones booleanas.

P	Q	NOT P	P AND Q	P OR Q	P XOR Q	P NAND Q	P NOR Q
0	0	1	0	0	0	1	1
0	1	1	0	1	1	1	0
1	0	0	0	1	1	1	0
1	1	0	1	1	0	0	0

$$A + B \cdot C = A + (B \cdot C) = A + BC$$

lo que quiere decir: hacer AND con B y C; luego hacer la OR con el resultado y A.

La Tabla B.1 define las operaciones lógicas básicas en una forma conocida como *tabla verdad*, que simplemente enumera el valor de una operación para cada combinación posible de los valores de los operandos. La tabla también enumera otros tres operadores útiles: XOR, NAND y NOR. La *exclusive-or* (XOR) de dos operandos lógicos es 1 si y solo si, uno de los operandos vale 1. La función NAND es el complemento (NOT) de la función AND, y la NOR es el complemento de la OR:

$$A \text{ NAND } B = \text{NOT}(A \text{ AND } B) = \overline{AB}$$

$$A \text{ NOR } B = \text{NOT}(A \text{ OR } B) = \overline{A + B}$$

Como veremos, estas tres operaciones nuevas pueden ser útiles para implementar ciertos circuitos digitales.

La Tabla B.2 resume las identidades clave del álgebra de Boole. Las ecuaciones se han organizado en dos columnas para mostrar la complementariedad, o dualidad, propias de las operaciones AND

Tabla B.2. Identidades básicas del álgebra de Boole.

Postulados básicos		
$A \cdot B = B \cdot A$	$A + B = B + A$	Conmutativa
$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$	$A + (B \cdot C) = (A + B) \cdot (A + C)$	Distributiva
$1 \cdot A = A$	$0 + A = A$	Identidad
$A \cdot \overline{A} = 0$	$A + \overline{A} = 1$	Complemento
Otras identidades		
$0 \cdot A = 0$	$1 + A = 1$	
$A \cdot A = A$	$A + A = A$	
$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	$A + (B + C) = (A + B) + C$	Asociativa
$\overline{A \cdot B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A} \cdot \overline{B}$	Teorema de DeMorgan

y OR. Hay dos clases de identidades: las reglas básicas (o *postulados*) que se afirman sin demostración, y otras identidades que se pueden derivar de los postulados básicos. Los postulados definen la manera en la que expresiones booleanas se interpretan. Una de las dos leyes distributivas merece ser destacada ya que difiere de lo que encontraríamos en un álgebra normal:

$$A + (B \cdot C) = (A + B) \cdot (A + C)$$

Las dos últimas expresiones, se denominan teorema de DeMorgan. Se pueden reescribir de la siguiente forma:

$$\begin{aligned} A \text{ NOR } B &= \overline{A \text{ AND } B} \\ A \text{ NAND } B &= \overline{A \text{ OR } B} \end{aligned}$$

Se invita al lector a verificar las expresiones de la Tabla B.2 sustituyendo las variables A, B y C por valores reales (unos y ceros).

B.2. PUERTAS

El bloque fundamental de construcción de todos los circuitos lógicos digitales son las puertas. Las funciones lógicas se implementan interconectando puertas.

Una puerta es un circuito electrónico que produce como señal de salida una operación booleana sencilla de las señales de entrada. Las puertas básicas usadas en lógica digital son AND, OR, NOT, NAND y NOR. La Figura B.1 muestra estas cinco puertas. Cada puerta se define de tres formas: símbolo gráfico, notación algebraica y tabla verdad. La simbología usada aquí y a lo largo del apéndice es el estándar IEEE, IEEE Std 91 [IEEE84]. Hay que destacar que la operación de inversión (NOT) se denota por un círculo.

Cada puerta tiene una o dos entradas y una salida. Cuando los valores de entrada cambian, la señal de salida correcta aparece casi instantáneamente, retrasada solo por el tiempo de propagación de la señal a través de la puerta (conocido como *retardo de puerta*). El significado de esto se verá en la Sección B.3.

Además de las puertas indicadas en la Figura B.1, se pueden usar puertas con tres, cuatro o más entradas. Por tanto, se puede implementar $X + Y + Z$ con una simple puerta OR de tres entradas.

Normalmente, no se usan todos los tipos de puertas en implementación. El diseño y la fabricación pueden ser más sencillos si solo se usan uno o dos tipos de puertas. Por tanto, es importante identificar conjuntos de puertas *funcionalmente completos*. Esto significa que cualquier función booleana se puede implementar usando solo las puertas del conjunto. Los siguientes conjuntos son funcionalmente completos:

- AND, OR, NOT
- AND, NOT
- OR, NOT
- NAND
- NOR





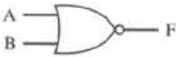
Nombre	Símbolo gráfico	Función algebraica	Tabla verdad															
AND		$F = A \cdot B$ or $F = AB$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	F	0	0	0	0	1	0	1	0	0	1	1	1
A	B	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = A + B$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	1
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
NOT		$F = \overline{A}$ or $F = A'$	<table><tr><th>A</th><th>F</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	F	0	1	1	0									
A	F																	
0	1																	
1	0																	
NAND		$F = \overline{(AB)}$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	1	0	1	1	1	0	1	1	1	0
A	B	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = \overline{(A + B)}$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	1	0	1	0	1	0	0	1	1	0
A	B	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																

Figura B.1. Puertas lógicas básicas.

Debería quedar claro que las puertas AND, OR y NOT constituyen un conjunto funcionalmente completo, ya que representan tres operaciones del álgebra de Boole. Para que las puertas AND y NOT formen un conjunto completo, debe haber una forma de sintetizar la operación OR a partir de las operaciones AND y NOT. Esto se puede hacer aplicando el teorema de DeMorgan:

$$A + B = \overline{\bar{A} \cdot \bar{B}}$$

$$A \text{ OR } B = \text{NOT}((\text{NOT } A) \text{ AND } (\text{NOT } B))$$

De igual forma, las operaciones OR y NOT son funcionalmente completas porque se pueden usar para sintetizar la operación AND.

La Figura B.2 muestra cómo se pueden implementar las funciones AND, OR y NOT únicamente con puertas NAND, y la Figura B.3 muestra lo mismo para NOR. Por esta razón, se pueden implementar circuitos digitales únicamente con puertas NAND o NOR, como frecuentemente se hace.

Con las puertas, se alcanza el nivel más primitivo de la ciencia e ingeniería de computadores. Un examen de las combinaciones de transistores usadas para construir puertas sale de este mundo para entrar en el mundo de la ingeniería electrónica. Para nuestros propósitos, sin embargo, nos es

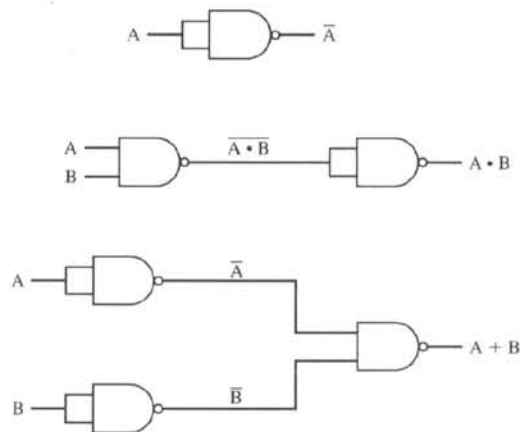


Figura B.2. Uso de las puertas NAND.

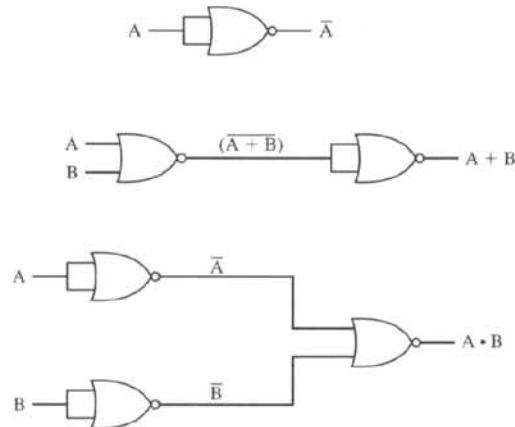


Figura B.3. Uso de las puertas NOR.

suficiente describir cómo se pueden usar las puertas como bloques de construcción para implementar los circuitos lógicos esenciales de un computador digital.

B.3. CIRCUITOS COMBINACIONALES

Un circuito combinacional es un conjunto de puertas interconectadas cuya salida, en un momento dado, es función solamente de la entrada en ese instante. Como ocurre con una puerta sencilla, la aparición de la entrada viene seguida casi inmediatamente por la aparición de la salida, con solo retardos de puerta.

En general, un circuito combinacional consiste en n entradas binarias y m salidas binarias. Como una puerta, un circuito combinacional puede definirse de tres formas:

- **Tabla verdad:** para cada una de las 2^n combinaciones posibles de las n señales de entrada, se enumera el valor binario de cada una de las m señales de salida.
- **Símbolo gráfico:** describe la organización de las interconexiones entre puertas.
- **Ecuaciones booleanas:** cada señal de salida se expresa como una función booleana de las señales de entrada.

IMPLEMENTACIÓN DE LAS FUNCIONES BOOLEANAS

Cualquier función booleana se puede implementar en electrónica en forma de red de puertas. Para una función dada, hay una serie de realizaciones alternativas. Considérese la función booleana representada por la tabla verdad de la Tabla B.3. Podemos expresar esta función sencillamente detallando las combinaciones de los valores de A, B, y C que hacen que F valga 1:

$$F = \overline{A}BC + A\overline{B}C + ABC \quad (\text{B.1})$$

Hay tres combinaciones de los valores de entrada que hacen que F valga 1, y si se da cualquiera de estas tres combinaciones, el resultado será 1. Este tipo de expresión, por razones evidentes, se conoce como la forma *suma de productos* (SOP, *sum of products*). La Figura B.4, muestra una sencilla implementación con puertas AND, OR y NOT. Se puede obtener también otra forma de la tabla verdad. La forma SOP indica que la salida es 1 si cualquiera de las combinaciones de entrada que producen 1 es cierta. También se puede decir que la salida es 1 si ninguna de las combinaciones de entrada que producen 0 es cierta. Por tanto:

$$F = (\overline{\overline{A}BC}) \cdot (\overline{A\overline{B}C}) \cdot (\overline{ABC}) \cdot (\overline{A\overline{B}C}) \cdot (\overline{ABC})$$

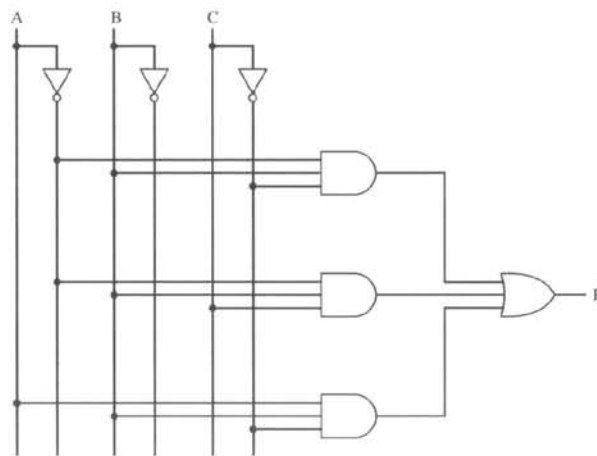


Figura B.4. Implementación de la suma de productos de la Tabla B.3.

Esta expresión se puede reescribir usando una generalización del teorema de DeMorgan:

$$\overline{(X \cdot Y \cdot Z)} = \overline{X} + \overline{Y} + \overline{Z}$$

Por tanto,

$$\begin{aligned} F &= (\overline{A} + \overline{B} + \overline{C}) \cdot (\overline{A} + \overline{B} + \overline{C}) \cdot (\overline{A} + \overline{B} + \overline{C}) \cdot (\overline{A} + \overline{B} + \overline{C}) \cdot (\overline{A} + \overline{B} + \overline{C}) \\ &= (A + B + C) \cdot (A + B + C) \cdot (A + B + C) \cdot (A + B + C) \cdot (A + B + C) \end{aligned} \quad (\text{B.2})$$

Esta última expresión está en la forma de *producto de sumas* (POS, *product of sums*), como se ilustra en la Figura B.5. Por claridad, no aparecen las puertas NOT. En su lugar, suponemos que se dispone de cada señal de entrada y de su complemento. Esto simplifica el diagrama lógico y hace más legibles las entradas a las puertas.

Por tanto, se puede realizar una función booleana tanto en la forma SOP como en la forma POS. En este momento, podría parecer que la elección dependería de si la tabla verdad contiene más unos o ceros para la función de salida: la SOP tiene un término para cada 1, y la POS tiene un término para cada 0. Sin embargo, hay otras consideraciones:

- Generalmente es posible obtener una expresión booleana más sencilla de la tabla verdad que de las formas SOP o POS.
- Puede ser preferible implementar la función con puertas sencillas (NAND o NOR).

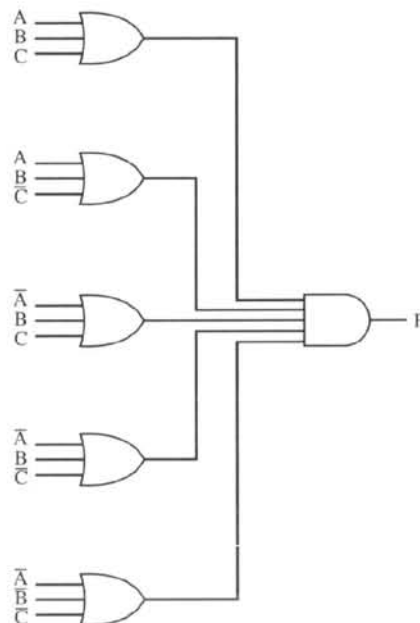


Figura B.5. Implementación del producto de sumas.

El significado del primer punto es que, con una expresión booleana más sencilla, se necesitan menos puertas para implementar la función. Para llevar a cabo esta simplificación se pueden usar tres métodos:

- Simplificación algebraica
- Mapas de Karnaugh
- Tablas de Quine-McKluskey

Simplificación algebraica. La simplificación algebraica supone la aplicación de las identidades de la Tabla B.3, que reduce la expresión booleana a otra con menos elementos.

Por ejemplo, supongamos de nuevo la Ecuación B.1. Un poco de razonamiento debería convencer al lector de que una expresión equivalente es

$$F = \overline{A}B + B\overline{C} \quad (\text{B.3})$$

O, incluso más sencillamente,

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

Esta expresión se puede implementar como se indica en la Figura B.6. La simplificación de la Ecuación B.1 se ha hecho esencialmente por observación. Para obtener una expresión más compleja, se necesita un procedimiento más sistemático.

Mapas de Karnaugh. Si se quiere simplificar, los mapas de Karnaugh son una forma conveniente de representar una función booleana con pocas variables (de cuatro a seis). El mapa es un conjunto de 2^n cuadrículas, que representan las posibles combinaciones de los valores de n variables binarias. La Figura B.7a muestra el mapa de cuatro cuadrículas para una función de dos variables. Es conveniente, para futuros propósitos, enumerar las combinaciones en el orden 00, 01, 11, 10. Como las cuadrículas corresponden a combinaciones que se van a usar para escribir información, las combinaciones se escriben habitualmente externamente, en la parte superior de las cuadrículas. En el caso de tres variables, la representación es un conjunto de ocho cuadrículas (Figura B.7b), con los valores de una de las variables a la izquierda y para las otras dos variables encima de las cuadrículas. Para cuatro variables, se necesitan 16 cuadrículas, con la disposición indicada en la Figura B.7c.

El mapa se puede usar para representar cualquier función booleana de la siguiente forma. Cada cuadrícula corresponde a un único producto en la forma de suma de productos, con valor 1 correspondiente a la variable y valor 0 correspondiente a la NOT de dicha variable. Por tanto, el producto

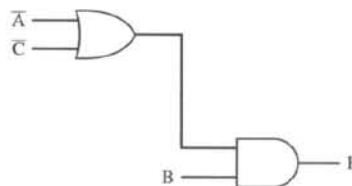


Figura B.6. Implementación simplificada de la Tabla B.3.

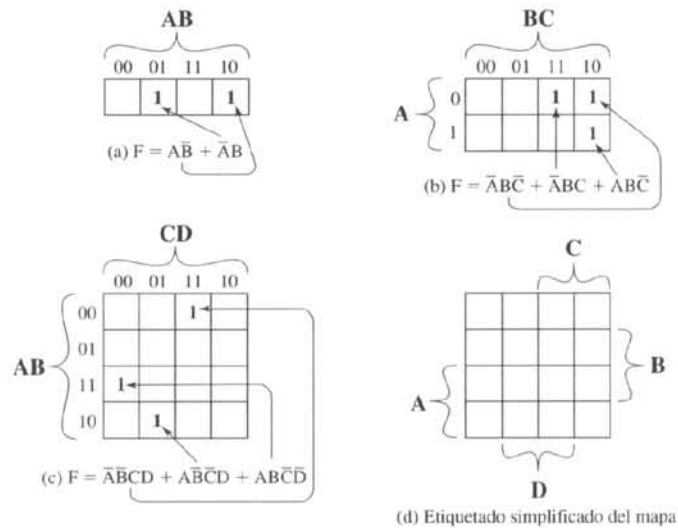


Figura B.7. La utilización de los mapas de Karnaugh para representar funciones booleanas.

$\bar{A}\bar{B}$ corresponde a la cuarta cuadrícula de la Figura B.7a. Para cada uno de estos productos de la función, se coloca un 1 en la cuadrícula correspondiente. Por tanto, para el ejemplo de dos variables, el mapa corresponde a $\bar{A}\bar{B} + AB$. Dada la tabla verdad de una función booleana, es fácil construir el mapa: para cada combinación de los valores de las variables que dan como resultado 1 en la tabla verdad, se pone un 1 en la cuadrícula correspondiente. La Figura B.7b muestra el resultado para la tabla verdad de la Tabla B.3. Para pasar de una expresión booleana a un mapa, primero es necesario poner la expresión en lo que se denomina forma *canónica*: cada término de la expresión debe contener cada variable. Así, por ejemplo, si se tiene la Ecuación A-3, debemos expandirla primero a la forma completa de la Ecuación A-1 y después pasarla al mapa.

Los rótulos usados en la Figura B.7d enfatizan la relación entre las variables y las filas y columnas del mapa. Aquí, las dos filas que abarca el símbolo A son aquellas en las que la variable

Tabla B.3. Función booleana de tres variables.

A	B	C	D
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

A vale 1; las filas que no abarca el símbolo A son aquellas en las que A vale 0 (lo mismo ocurre para B, C, y D).

Una vez que se ha creado el mapa de una función, podemos escribir, a menudo, una expresión algebraica sencilla anotando el conjunto de unos del mapa. El principio es el siguiente. Dos casillas adyacentes cualesquiera difieren en solo una de las variables. Si dos casillas adyacentes contienen un 1, entonces los correspondientes términos producto difieren solo en una variable. En tal caso, los dos términos se pueden fundir en uno eliminando esta variable. Por ejemplo, en la Figura B.8a, las dos casillas adyacentes corresponden a los términos $\overline{A}BCD$ y $\overline{A}BC\overline{D}$. Por tanto, la función se puede expresar así:

$$\overline{A}BCD + \overline{A}BC\overline{D} = \overline{A}BD$$

Este proceso se puede ampliar de varias formas. Primero, el concepto de adyacencia se puede ampliar para incluir el recubrimiento alrededor del borde del mapa. Por tanto, la casilla más alta de

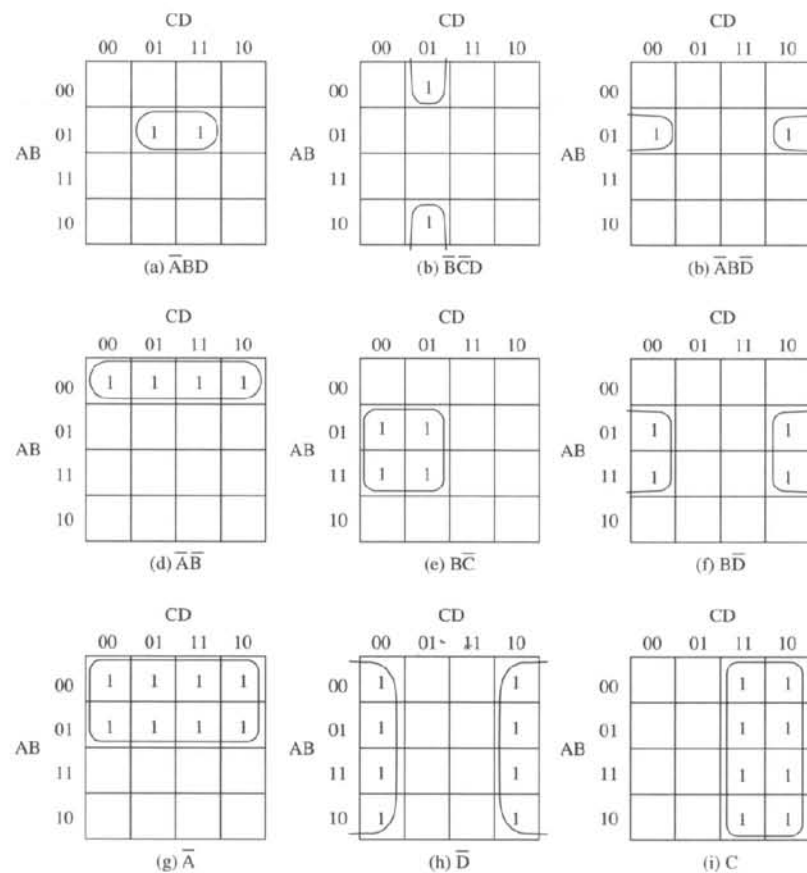


Figura B.8. Utilización de los mapas de Karnaugh.

una columna es adyacente a la más baja, y la casilla más a la izquierda de la fila es adyacente a la que está más a la derecha. Estas condiciones se ilustran en las Figuras B. 8b y c. Segundo, podemos agrupar no solo dos casillas, sino 2^n casillas adyacentes, es decir, 4, 8, etc. Los tres siguientes ejemplos de la Figura B.8, muestran agrupaciones de cuatro casillas. Hay que destacar que en este caso, se pueden eliminar dos de las variables. Los tres últimos ejemplos muestran grupos de ocho cuadrículas, que permiten eliminar tres variables.

Para simplificar podemos resumir las reglas como sigue:

1. Entre las casillas marcadas (casillas con un 1), encontrar aquellas que pertenezcan a un único bloque lo mayor posible ya sea de 1, 2, 4 u 8 casillas, y rodear el bloque con un círculo.
2. Seleccionar bloques adicionales de casillas marcadas tan grandes y tan pocas como sea posible, pero que incluyan a cada casilla marcada al menos una vez. Los resultados pueden no ser únicos en algunos casos. Por ejemplo, si una casilla marcada se combina exactamente con otras dos, y no hay una cuarta para completar un grupo mayor, entonces se puede elegir entre dos agrupaciones. Cuando se seleccionan grupos se permite usar el mismo uno más de una vez.
3. Seguir dibujando círculos alrededor de las casillas aisladas marcadas, o de parejas de casillas marcadas adyacentes, o de grupos de cuatro, ocho, etc. de forma que cada cuadrado marcado pertenezca al menos a un círculo; luego utilizar el menor número posible de bloques para incluir a todas las casillas marcadas.

La Figura B.9a, basada en la Tabla B.3, ilustra el procedimiento. Si queda algún 1 aislado después de haber agrupado, entonces, cada uno de ellos se rodea con un círculo como si fuera un grupo de unos. Finalmente, antes de pasar el mapa a una expresión simplificada Booleana, cualquier grupo de unos que esté completamente solapado por otros grupos se puede eliminar. Es lo que se muestra

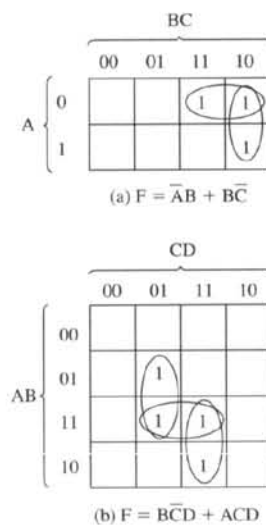


Figura B.9. Grupos solapados.

en la Figura B.9b. En este caso el grupo horizontal es redundante y se puede ignorar a la hora de crear la expresión booleana.

Es necesario mencionar una característica adicional de los mapas de Karnaugh. En algunos casos, ciertas combinaciones de valores de las variables no se dan nunca, y por consiguiente, la salida correspondiente no se produce tampoco. Estas se denominan condiciones de «indiferencia». Para cada una de estas condiciones, se coloca la letra «d» en la casilla correspondiente del mapa. Cuando se hace la agrupación y simplificación, cada «d» puede tratarse como un 1 o un 0, eligiendo lo que conduzca a una expresión más sencilla.

Un ejemplo, presentado en [HAYE88], ilustra lo que hemos estado discutiendo. Nos gustaría desarrollar las expresiones booleanas para un circuito que suma un 1 a los dígitos decimales empaquetados. Recordemos de la Sección 9.2 que con decimales empaquetados, cada dígito decimal se representa con un código de cuatro bits, de una forma obvia.

Así, $0 = 0000$, $1 = 0001$, ..., $8 = 1000$, y $9 = 1001$. Las combinaciones de cuatro bits restantes, de 1010 a 1111, no se usan. Este código también se denomina Decimal Codificado en Binario (BCD, *Binary Coded Decimal*).

La Tabla B.4 muestra la tabla verdad para producir un resultado de cuatro bits que es la entrada BCD de cuatro bits incrementada en 1. La suma es en módulo 10. Así, $9 + 1 = 0$. También, hay que notar que seis de los códigos de entrada producen «indiferencias» como resultado, ya que esas no son entradas BCD válidas. La Figura B.10 muestra el resultado de los mapas de Karnaugh para cada una de las variables de salida. Las casillas «d» se usan para lograr las mejores agrupaciones posibles.

Tabla B.4. Tabla verdad de un contador digital de un dígito.

Número	Entrada				Número	Salida			
	A	B	C	D		W	X	Y	Z
0	0	0	0	0	1	0	0	0	1
1	0	0	0	1	2	0	0	1	0
2	0	0	1	0	3	0	0	1	1
3	0	0	1	1	4	0	1	0	0
4	0	1	0	0	5	0	1	0	1
5	0	1	0	1	6	0	1	1	0
6	0	1	1	0	7	0	1	1	1
7	0	1	1	1	8	1	0	0	0
8	1	0	0	0	9	1	0	0	1
9	1	0	0	1	0	0	0	0	0
Indife- rencias	1	0	1	0		d	d	d	d
	1	0	1	1		d	d	d	d
	1	1	0	0		d	d	d	d
	1	1	0	1		d	d	d	d
	1	1	1	0		d	d	d	d
	1	1	1	1		d	d	d	d

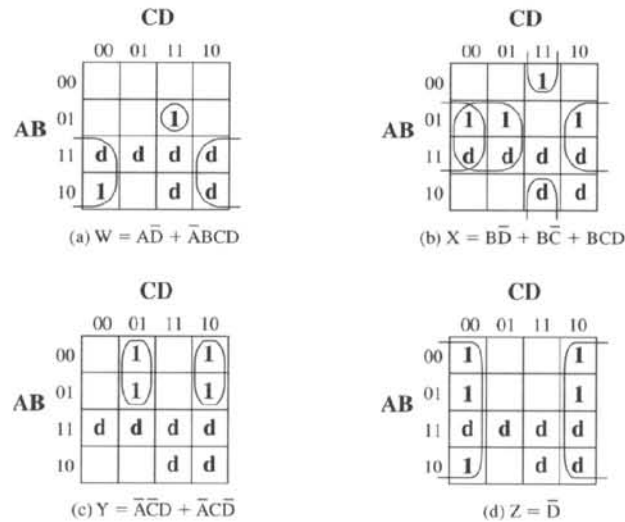


Figura B.10. Mapas de Karnaugh para el ejemplo del contador.

El método de Quine-McKluskey. Cuando se incrementa en más de cuatro variables, el método del mapa de Karnaugh se va haciendo cada vez más incómodo. Con cinco variables, se necesitan dos mapas de 16×16 , con un mapa situado encima del otro, en tres dimensiones, para conseguir la adyacencia. ¡Seis variables requieren cuatro tablas de 16×16 en cuatro dimensiones! Un procedimiento alternativo es una técnica tabular, denominada método de Quine-McKluskey. Este método es adecuado para programar en un computador y así tener una herramienta automática que produzca expresiones booleanas minimizadas.

Este método se explica mejor mediante un ejemplo. Consideremos la siguiente expresión:

$$ABCD + A\bar{B}\bar{C}D + ABC\bar{D} + \bar{A}BCD + \bar{A}BC\bar{D} + \bar{A}BCD + \bar{A}BC\bar{D} + \bar{A}\bar{B}CD$$

Supongamos que esta expresión se ha obtenido de una tabla verdad. Nos gustaría conseguir una expresión mínima adecuada para implementarla con puertas.

El primer paso es construir una tabla en la que cada fila corresponda a un término producto de la expresión. Los términos se agrupan de acuerdo con el número de variables complementadas. Es decir, empezamos con el término sin complementos, si existe, luego todos los términos con un complemento, etc. La Tabla B.5 muestra la lista para nuestra expresión ejemplo, indicando con las líneas horizontales las agrupaciones. Para más claridad, cada término se representa con un 1 para cada variable sin complementar y con un 0 para cada variable complementada. Por tanto, agrupamos términos de acuerdo con el número de unos que contiene. La columna «índice» es simplemente el equivalente decimal y es útil para lo que se explicará más adelante.

El siguiente paso es encontrar todas las parejas de términos que difieren en solo una variable, es decir, todos los pares de términos que son iguales excepto que, una variable es 0 en uno de los términos y 1 en el otro. Debido a la manera en la que se agrupan términos, podemos hacerlo empezando

Tabla B.5. Primer paso del método de Quine-McKluskey
(para $ABCD + ABC\bar{D} + AB\bar{C}D + \bar{A}BCD + \bar{A}BC\bar{D} + \bar{A}\bar{B}CD + \bar{A}\bar{B}\bar{C}D$).

Término producto	Índice	A	B	C	D	
$\bar{A}\bar{B}\bar{C}D$	1	0	0	0	1	✓
$\bar{A}B\bar{C}D$	5	0	1	0	1	✓
$\bar{A}BC\bar{D}$	6	0	1	1	0	✓
$AB\bar{C}\bar{D}$	12	1	1	0	0	✓
$\bar{A}BCD$	7	0	1	1	1	✓
$A\bar{B}CD$	11	1	0	1	1	✓
$AB\bar{C}D$	13	1	1	0	1	✓
$ABCD$	15	1	1	1	1	✓

con el primer grupo y comparar cada término del primer grupo con cada término del segundo grupo. Después comparamos cada término del segundo grupo con todos los términos del tercer grupo, y así sucesivamente. Cuando se encuentra un emparejamiento, se coloca una marca en cada término, se combina el par eliminando la variable en la que difieren los dos términos y se añade a la nueva lista. Así, por ejemplo, los términos $\bar{A}BC\bar{D}$ y $\bar{A}BCD$ se combinan para producir el término $\bar{A}BC$. Este proceso continúa hasta que se haya analizado la tabla original entera. El resultado es una nueva tabla con los siguientes elementos:

$$\begin{array}{lll}
 \bar{A}CD & \bar{A}BC & ABD\checkmark \\
 & B\bar{C}D\checkmark & ACD \\
 & \bar{A}BC & BCD\checkmark \\
 & \bar{A}BD\checkmark &
 \end{array}$$

La nueva tabla se organiza en grupos, como indicamos antes, de la misma forma que la primera tabla. La segunda tabla se procesa, después, de la misma manera que la primera. Es decir, se comprueban términos que difieren en solo una variable y se produce un nuevo término para una tercera tabla. En este ejemplo, la tercera tabla que se hace contiene solamente un término: BD .

En general, el proceso continuaría a través de sucesivas tablas hasta una tabla en la que no haya emparejamientos. En este caso, hay implicadas tres tablas.

Una vez se haya completado el proceso descrito anteriormente, tenemos que eliminar muchos de los posibles términos de la expresión. Aquellos términos que no hayan sido eliminados se usan para construir una matriz, como se ilustra en la Tabla B.6. Cada fila de la matriz corresponde a uno de los términos que no se han eliminado (no tiene marca) en cualquiera de las tablas usadas anteriormente. Cada columna se corresponde con uno de los términos de la expresión original. Se coloca una X en cada intersección de una fila y una columna tal que el elemento de la fila sea "compatible" con el elemento de la columna. Es decir, las variables presentes en el elemento de la fila tienen el mismo valor que las variables presentes en el elemento de la columna. Después, se rodea con un círculo cada X que esté sola en una columna. Entonces, se sitúa un cuadrado alrededor de cada X en cualquier fila

Tabla B.6. Último paso del método de Quine-McKluskey
(para $F = ABCD + AB\bar{C}D + AB\bar{C}\bar{D} + A\bar{B}CD + \bar{A}BCD + \bar{A}B\bar{C}D + \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}C\bar{D}$).

	$ABCD$	$AB\bar{C}D$	$AB\bar{C}\bar{D}$	$A\bar{B}CD$	$A\bar{B}\bar{C}D$	$A\bar{B}\bar{C}\bar{D}$	$\bar{A}BCD$	$\bar{A}B\bar{C}D$
BD	X	X			X		X	
$\bar{A}CD$							X	⊗
$\bar{A}BC$					X	⊗		
ABC		X	⊗					
ACD	X			⊗				

que tenga un círculo en la X. Si cada columna tiene ahora una X encerrada en un cuadrado o en un círculo, entonces ya se ha concluido, y los elementos de esa fila cuyas X estén marcadas constituyen la expresión mínima. Por tanto, en nuestro ejemplo, la expresión final es:

$$ABC + ACD + \bar{A}BC + \bar{A}CD$$

En los casos en los que algunas columnas no tengan ni un círculo ni un cuadrado, se necesita un proceso adicional. Esencialmente, seguimos añadiendo elementos a la expresión hasta que se cubran todas las columnas.

Resumamos el método Quine-McKluskey para intentar justificar intuitivamente cómo se realiza la minimización. La primera fase de la operación es razonablemente sencilla. El proceso elimina variables innecesarias en términos producto. Entonces, la expresión $ABC + ABC$ es equivalente a AB , dado que

$$ABC + ABC = AB(C + \bar{C}) = AB$$

Tras la eliminación de las variables, nos queda una expresión que es claramente equivalente a la expresión original. Sin embargo, puede haber términos redundantes en la expresión, como encontramos agrupaciones redundantes en los mapas de Karnaugh. La organización de la matriz asegura que se incluye (cubre) cada término de la expresión original y lo hace de forma que minimiza el número de términos en la expresión final.

Implementaciones NAND y NOR. Otra consideración en la implementación de funciones booleanas concierne a los tipos de puertas usados. Es a menudo deseable implementar una función booleana solo con puertas NAND o solo con puertas NOR. Aunque pueda no ser la implementación con un mínimo de puertas, tiene la ventaja de la regularidad, que puede simplificar el proceso de fabricación. Consideremos de nuevo la Ecuación B.3:

$$F = B(\bar{A} + \bar{C})$$

Como el complemento del complemento es el valor original,

$$F = B(\bar{A} + \bar{C}) = \overline{(\overline{\bar{A}}) + (\overline{\bar{C}})} = \overline{AB + BC}$$

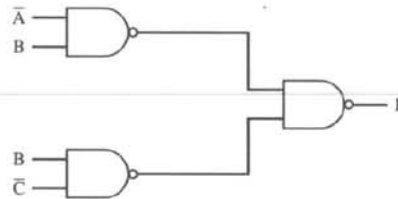


Figura B.11. Implementación con NAND de la Tabla B.3.

aplicando el teorema de DeMorgan,

$$F = \overline{(\overline{A}B)} \cdot \overline{(BC)}$$

que tiene tres operadores NAND, como se ilustra en la Figura B.11.

MULTIPLEXORES

El multiplexor conecta varias entradas con una única salida. En un momento dado, se selecciona una de las entradas para que pase a la salida. La Figura B.12 muestra una representación en diagrama de bloques general.

Representa un multiplexor de 4 a 1. Hay cuatro líneas de entrada, llamadas D0, D1, D2 y D3. Se selecciona una de estas líneas para dar la señal de salida F. Para seleccionar una de las cuatro entradas posibles, se necesita un código de selección de dos bits, que se implementa con dos líneas de selección llamadas S1 y S2.

La tabla verdad de la Tabla B.7 define un ejemplo de un multiplexor de cuatro a uno. Es una forma simplificada de una tabla verdad. En vez de mostrar todas las combinaciones posibles de las variables de entrada, muestra la salida como dato procedente de la línea D0, D1, D2, o D3. La Figura B.13 muestra una implementación usando puertas AND, OR y NOT. S1 y S2 se conectan a las puertas AND de forma que, para cualquier combinación de S1 y S2, tres de las puertas AND tengan la salida a 0. La cuarta puerta AND sacará el valor de la línea seleccionada que será 0 o 1. Por tanto,

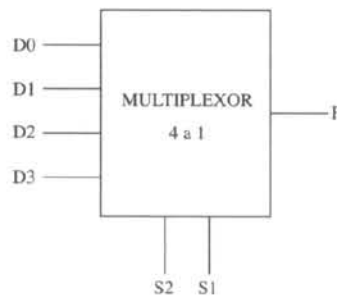
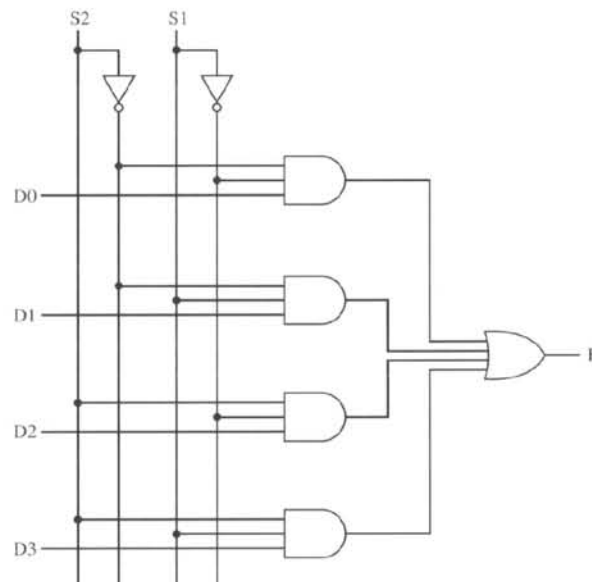


Figura B.12. Representación de multiplexor de 4 a 1.

Tabla B.7. Tabla verdad de un multiplexor 4 a 1.

S2	S1	F
0	0	D0
0	1	D1
1	0	D2
1	1	D3


Figura B.13. Implementación de un multiplexor.

tres de las entradas de la puerta OR son siempre 0, y la salida de la puerta OR será igual al valor de la puerta de entrada seleccionada. Usando esta organización regular, es fácil construir multiplexores de ocho a uno, de 16 a uno, etc.

Los multiplexores se usan en circuitos digitales para controlar el enrutamiento de señales y datos. Un ejemplo es la carga del contador de programa (PC). El valor a cargar en el contador de programa puede venir de una o varias fuentes diferentes:

- De un contador binario, si el PC se va a incrementar para la siguiente instrucción.
- Del registro de instrucción, si se acaba de ejecutar una instrucción de salto usando direccionamiento directo.
- De la salida de la ALU, si la instrucción de salto especifica la dirección usando modo de desplazamiento.

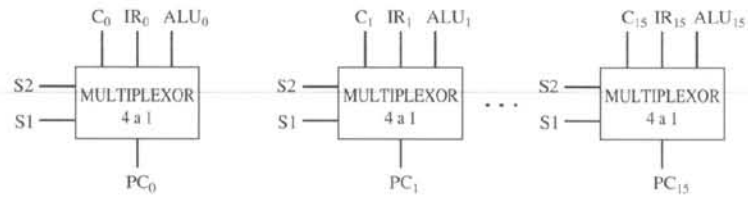


Figura B.14. Multiplexor de entrada al contador de programa.

Las distintas entradas se pueden conectar a las líneas de entrada de un multiplexor con el PC conectado a la línea de salida. Las líneas seleccionadas determinan cuál es el valor a cargar en el PC. Como el PC contiene varios bits, se usan varios multiplexores, uno por bit. La Figura B.14 ilustra esto para direcciones de 16 bits.

DECODIFICADORES

Un decodificador es un circuito combinacional con varias líneas de salida, con una sola de ellas seleccionada en un instante dado, dependiendo del patrón de líneas de entrada. En general, un decodificador tiene n entradas y 2^n salidas. La Figura B.15 muestra un decodificador con tres entradas y ocho salidas.

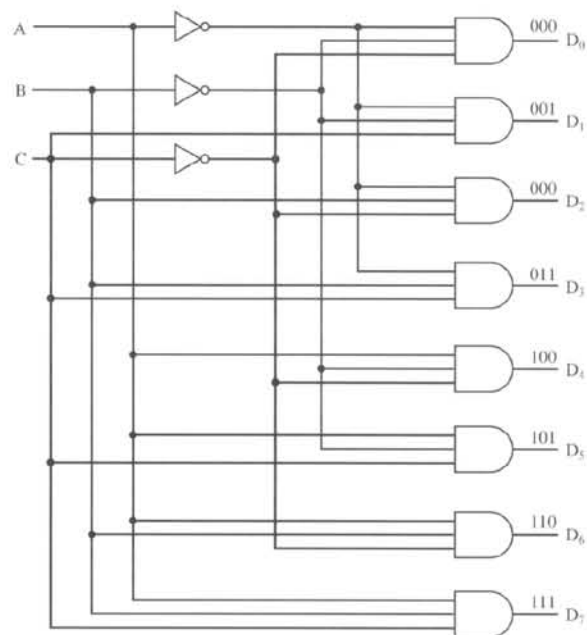


Figura B.15. Decodificador con tres entradas y $2^3 = 8$ salidas

Los decodificadores tienen muchos usos en computadores digitales. Un ejemplo es la decodificación de direcciones. Supongamos que queremos construir una memoria de 1 KB usando cuatro chips RAM de 256×8 bits. Queremos un espacio de direcciones único y unificado, que se pueda descomponer como sigue:

Dirección	Chip
0000-00FF	0
0100-01FF	1
0200-02FF	2
0300-03FF	3

Cada chip requiere ocho líneas de dirección, y estos se toman de los ocho bits menos significativos de la dirección. Los dos bits más significativos de los diez bits de dirección, se usan para seleccionar uno de los cuatro chips RAM. Para ello, se usa un decodificador de dos a cuatro cuya salida habilita uno de los cuatro chips, como se muestra en la Figura B.16.

Con una línea adicional de entrada, se puede usar el decodificador como demultiplexor. El demultiplexor realiza la función inversa de un multiplexor; conecta una única entrada a una o varias salidas. Esto se ve en la Figura B.17. Como antes, se decodifican n entradas para producir una única salida de las 2^n . Con todas las 2^n líneas de salida se hace la operación AND con un dato de la línea de entrada.

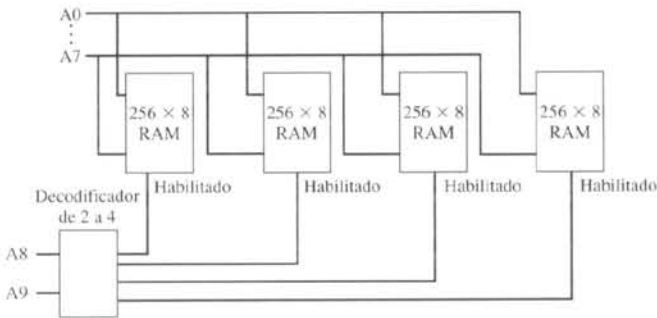


Figura B.16. Decodificación de una dirección.

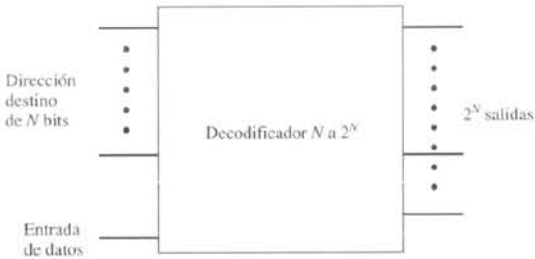


Figura B.17. Implementación de un demultiplexor usando un decodificador.

Por tanto, las n entradas actúan como una dirección para seleccionar una línea de salida concreta, y el valor del dato de la línea de entrada (0 o 1) se encamina a dicha línea de salida.

La configuración de la Figura B.17 se puede ver de otra forma. Cambiar el nombre de la nueva línea, *Entrada de datos*, por *Habilitar*. Esto permite el control de la temporización del decodificador. La salida decodificada aparece solo cuando está presente la entrada codificada y la línea de habilitación valga uno.

ARRAY LÓGICO PROGRAMABLE (PLA, *PROGRAMMABLE LOGIC ARRAY*)

Hasta ahora, hemos tratado las puertas individuales como bloques de construcción, para realizar funciones arbitrarias. El diseñador podría seguir una estrategia de minimización del número de puertas que van a usarse, manipulando las expresiones booleanas correspondientes.

Como el nivel de integración de los circuitos integrados aumenta, se tienen en cuenta otras consideraciones. Los primeros circuitos integrados, usando una escala de integración pequeña (SSI), contenían de una a diez puertas en un chip. Cada puerta se trata independientemente, en su uso como bloques de construcción descrito hasta ahora. La Figura B.18 es un ejemplo de algunos chips SSI. Para construir una función lógica, se disponen varios de estos chips en una tarjeta de circuito impreso y se hacen las interconexiones adecuadas entre los terminales de los chips.

El incremento del nivel de integración hace posible añadir más puertas en un chip y hacer interconexiones dentro del chip también. Esto tiene como ventajas la disminución del coste y del tamaño, y el incremento de velocidad (puesto los retardos dentro del chip son menores que los retardos fuera del chip). Sin embargo, surge un problema de diseño. Para cada función lógica particular o conjunto de funciones, hay que diseñar la organización de las puertas e interconexiones en el chip. El coste y el tiempo implicados en el diseño del chip a medida, son altos. Entonces, se convierte en algo atractivo el desarrollo de un chip de uso general que esté listo para adaptarse a usos específicos. Esta es la intención de los *arrays lógicos programables* (PLA, *programmable logic array*).

Los PLA se basan en el hecho de que cualquier función booleana (tabla verdad) se puede expresar en forma de suma de productos (SOP), como hemos visto. Un PLA consiste en una disposición regular de puertas NOT, AND y OR en un chip. Cada entrada al chip pasa a través de una puerta NOT, así que cada entrada y su complemento están disponibles para cada puerta AND. La salida de cada puerta AND está disponible para cada puerta OR, y la salida de cada puerta OR es una salida del chip. Haciendo las conexiones adecuadas, se pueden implementar expresiones SOP arbitrarias.

La Figura B.19a muestra un PLA con tres entradas, ocho puertas, y dos salidas. Los PLAs mayores contienen varios cientos de puertas, de quince a 25 entradas, y de cinco a quince salidas. Las conexiones de las entradas a las puertas AND, y de las puertas AND a las puertas OR, no están especificadas.

Los PLAs se fabrican de dos formas diferentes para permitir una programación más fácil (hacer las conexiones). En la primera, cada conexión posible se hace a través de fusible en cada punto de intersección. Este tipo de PLA se denomina *array lógico programable de campo*. Alternativamente, las propias conexiones se pueden hacer durante la fabricación del chip usando una máscara adecuada para un patrón de interconexión particular. En cualquiera de los casos, la PLA proporciona una forma de implementación de funciones lógicas digitales flexible y barata.

En la Figura B.19b se muestra un diseño que sintetiza dos expresiones booleanas.

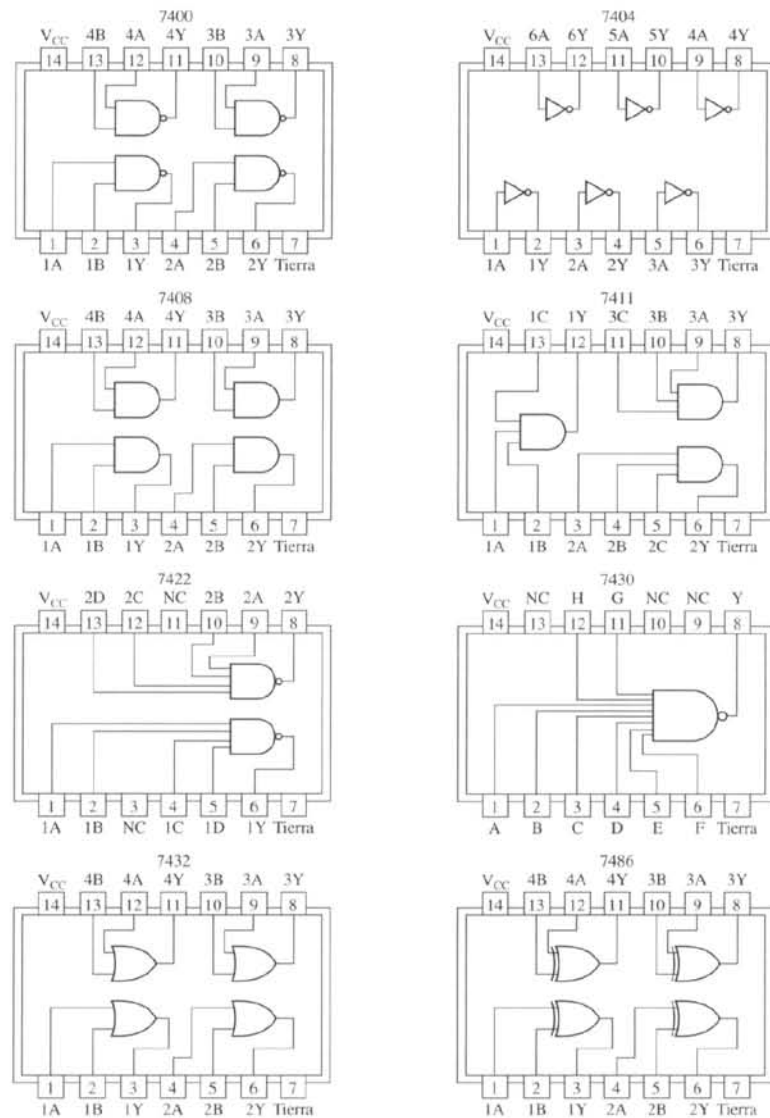


Figura B.18. Algunos chips SSI. Organización de los pines de "The TTL data book for design engineers", copyright 1976 Texas Instruments Incorporated.

MEMORIA DE SOLO LECTURA (ROM, READ ONLY MEMORY)

A los circuitos combinacionales se les llama a veces circuitos «sin memoria», ya que su salida depende solo de la entrada actual y no retiene la historia de las entradas anteriores. Sin embargo, hay un tipo de memoria que se implementa con circuitos combinacionales, llamada *memoria de solo lectura* (ROM).

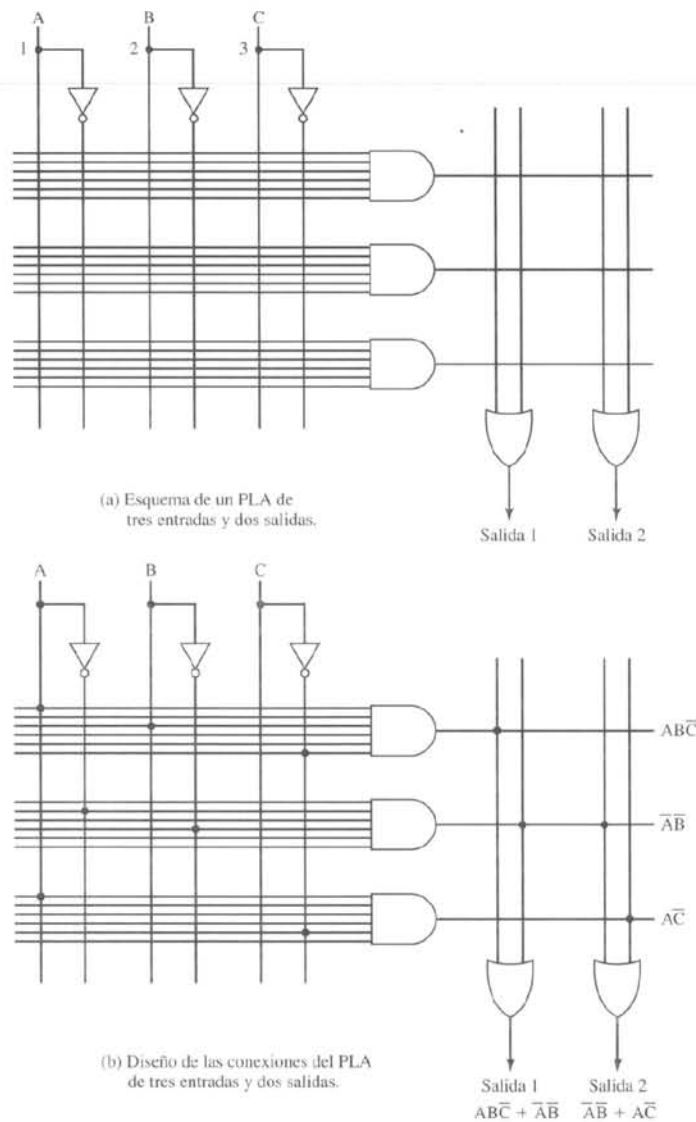


Figura B.19. Ejemplo de un conjunto lógico programable.

Recordemos que una memoria ROM es una unidad de memoria en la que solo se realiza la operación de lectura. Esto implica que la información binaria almacenada en una ROM es permanente y se creó en el proceso de fabricación. Entonces, una entrada dada a la ROM (líneas de direcciones) siempre produce la misma salida (líneas de datos). Como las salidas son función solo de las entradas presentes, la ROM es de hecho un circuito combinacional.

Tabla B.8. Tabla verdad de una ROM.

Entrada				Salida			
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	1	0	0	1
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

Se puede implementar una ROM con un decodificador y un conjunto de puertas OR. Como ejemplo, consideremos la Tabla B.8. Esta se puede ver como una tabla verdad con cuatro entradas y cuatro salidas. Para cada uno de los 16 posibles valores de entrada, se muestra el conjunto correspondiente de valores de salida. También se puede ver como la definición del contenido de una ROM de 64 bits de 16 palabras de cuatro bits cada una. Las cuatro entradas determinan una dirección, y las cuatro salidas especifican el contenido de posición indicada en la dirección. En la Figura B.20 se muestra cómo podría implementarse esta memoria usando un decodificador de 4 a 16 y cuatro puertas OR. Como en un PLA, se usa una organización regular y las interconexiones se hacen de forma que reflejen el resultado deseado.

SUMADORES

Hasta ahora hemos visto cómo se pueden usar puertas interconectadas para implementar funciones como enrutamiento de señales, decodificación y ROM. Un área esencial a la que no nos hemos dirigido todavía es la aritmética. En esta breve visión general, nos contentaremos con ver la función de suma.

La suma binaria difiere de la del álgebra booleana en que el resultado incluye un término de acarreo. Así,

0	0	1	1
+0	+1	+0	+1
0	1	1	10

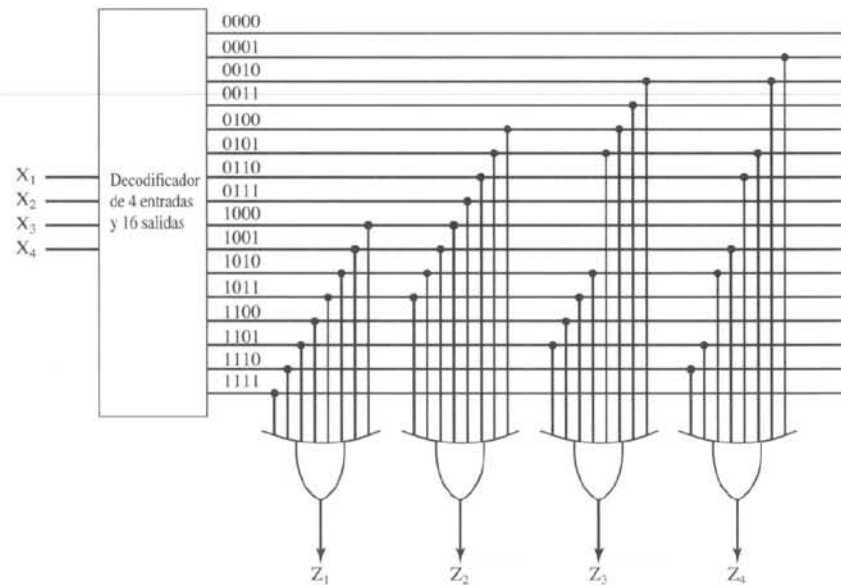


Figura B.20. ROM de 64 bits.

No obstante, la suma se puede implementar con términos booleanos. En la Tabla B.9 puede verse la lógica para sumar dos bits de entrada para producir un bit de suma y un bit de acarreo. Esta tabla verdad podría implementarse fácilmente en lógica digital. Sin embargo, no estamos interesados en realizar la suma con solo un par de bits. Más bien queremos sumar dos números de n bits. Esto se puede hacer poniendo juntos un conjunto de sumadores de forma que el acarreo de un sumador sea la entrada del siguiente. En la Figura B.21 se muestra un sumador de cuatro bits.

Tabla B.9. Tabla verdad de la suma binaria.

(a) Suma con un bit			
A	B	Suma	Acarreo
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(b) Suma con acarreo de entrada				
C_{in}	A	B	Sum	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

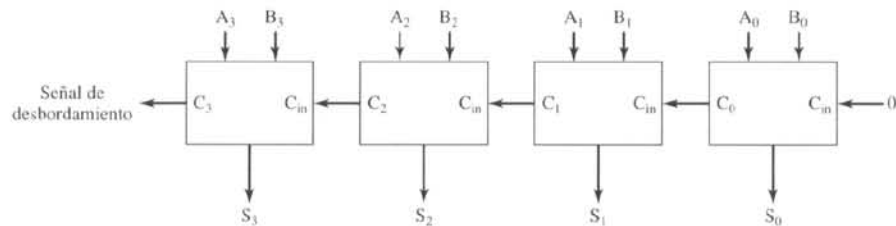


Figura B.21. Sumador de cuatro bits.

Para que funcione un sumador de varios bits, cada uno de los sumadores de un bit debe tener tres entradas, incluyendo el acarreo del sumador inmediato inferior. La tabla verdad revisada aparece en la Tabla B.9b. Las dos salidas se pueden expresar:

$$\begin{aligned} \text{Suma} &= \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC \\ \text{Acarreo} &= AB + AC + BC \end{aligned}$$

La Figura B.22 es una implementación usando puertas AND, OR y NOT.

Entonces tenemos la lógica necesaria para implementar un sumador de varios bits como se muestra en la Figura B.23. Hay que notar que como la salida de cada sumador depende del acarreo del sumador previo, hay un retardo que crece del bit menos significativo al más significativo. Cada

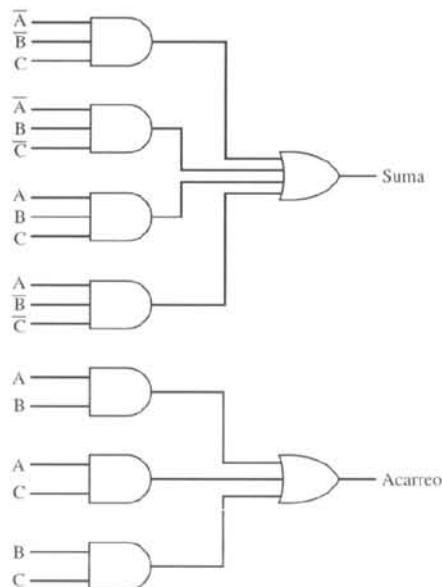


Figura B.22. Implementación de un sumador.

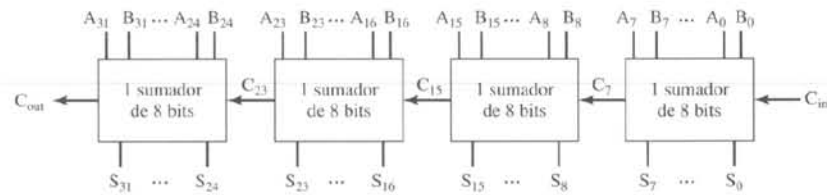


Figura B.23. Construcción de un sumador de 32 bits usando sumadores de 8 bits.

sumador de un bit experimenta cierta cantidad del retardo de puerta, y este retardo de puerta se acumula. Para sumadores grandes, el retardo acumulado puede hacerse inaceptablemente alto.

Si los valores de acarreo se pudieran determinar sin tener que pasar a través de todas las etapas previas, entonces cada sumador de un bit podría funcionar independientemente, y el retardo no se acumularía. Este se puede lograr con un procedimiento conocido como *acarreo anticipado*. Veamos de nuevo el sumador de cuatro bits para explicar este método.

Nos gustaría proponer una expresión que especificara el acarreo de entrada a cualquier etapa del sumador sin referirnos a los valores de acarreo previos. Tenemos

$$C_0 = A_0 B_0 \quad (\text{B.4})$$

$$C_1 = A_1 B_1 + A_1 A_0 B_0 + B_1 A_0 B_0 \quad (\text{B.5})$$

Siguiendo el mismo procedimiento, tenemos

$$C_2 = A_2 B_2 + A_2 A_1 B_1 + A_2 A_1 A_0 B_0 + A_2 B_1 A_0 B_0 + B_2 A_1 B_1 + B_2 A_1 A_0 B_0 + B_2 B_1 A_0 B_0$$

Este proceso se puede repetir para sumadores arbitrariamente grandes. Cada término de acarreo se puede expresar en forma SOP como función de solo las entradas originales, sin dependencia de los acarreos. Por tanto, solo se dan dos niveles de retardo de puerta a pesar del tamaño del sumador.

Para números grandes, este procedimiento se vuelve demasiado complicada. Evaluando la expresión para el bit más significativo de un sumador de n bits, se requiere una puerta OR con $n - 1$ entradas y n puertas AND de 2 a $n + 1$ entradas. Por consiguiente, el acarreo anticipado se hace normalmente con solo cuatro u ocho bits a la vez. La Figura B.23 muestra como se puede construir un sumador de 32 bits a partir de cuatro sumadores de ocho bits. En este caso, el acarreo debe pasar a través de los cuatro sumadores de ocho bits, pero puede ser sustancialmente más rápido que pasar a través de 32 sumadores de un bit.

B.4. CIRCUITOS SECUENCIALES

Los circuitos combinacionales implementan las funciones esenciales de un computador digital. Sin embargo, excepto para el caso especial de ROM, no proporcionan memoria o información de estado, elementos también esenciales para el funcionamiento de un computador digital. Para estos últimos fines, se usa una forma de circuitos lógicos digitales más compleja: los circuitos secuenciales. La

salida actual de un circuito secuencial depende no solo de la entrada actual, sino también de la historia pasada de las entradas. Otra manera más general y útil de ver esto es que la salida actual de un circuito secuencial depende de la entrada actual y del estado actual del circuito.

En esta sección, examinaremos algunos ejemplos sencillos pero útiles de circuitos secuenciales. Como veremos, los circuitos secuenciales se implementan con circuitos combinacionales.

BIESTABLES

La forma más sencilla de un circuito secuencial es un biestable. Hay varios tipos de biestables, y todos ellos comparten dos propiedades:

- El biestable es un dispositivo con dos estados. Está en uno de dos estados, en ausencia de entrada, recordando el último estado. Entonces, el biestable puede funcionar como una memoria de un bit.
- El biestable tiene dos salidas, que son siempre complementarias. Normalmente se denominan Q y \bar{Q} .

El cerrojo (latch) S-R. La Figura B.24 muestra una configuración común conocida como biestable S-R ó cerrojo S-R. El circuito tiene dos entradas, S (Set) y R (Reset), y dos salidas, Q y \bar{Q} , y consiste en dos puertas NOR conectadas por realimentación.

Primero, mostremos que el circuito biestable. Supongamos que S y R valen 0 y que Q es 0. Las entradas a la puerta NOR inferior son $\bar{Q} = 0$ y $S = 0$. Entonces, la salida $\bar{Q} = 1$ significa que las entradas de la puerta NOR superior son $\bar{Q} = 1$ y $R = 0$, con salida $Q = 0$. Por tanto, el estado del circuito es internamente consistente y permanece estable mientras $S = R = 0$. Con un razonamiento similar se llega a que el estado $Q = 1, \bar{Q} = 0$, es también estable para $R = S = 0$.

Por tanto, estos circuitos pueden funcionar como una memoria de 1bit. Podemos ver la salida Q como el «valor» del bit. Las entradas S y R sirven para escribir los valores 1 y 0, respectivamente, en la memoria. Para ver esto, consideremos el estado $Q = 0, \bar{Q} = 1, S = 0, R = 0$. Supongamos que S cambia al valor 1. Ahora las entradas de la puerta NOR inferior son $S = 1, \bar{Q} = 1$. Después de cierto tiempo de retardo (t , la salida de la puerta NOR inferior será $\bar{Q} = 0$ (ver Figura B.25).

Así que, en este momento, las entradas a la puerta NOR superior pasan a ser $R = 0, \bar{Q} = 0$. Después de otro retarde de puerta de (t , la salida Q pasa a 1. Este de nuevo un estado estable. Las entradas de la puerta inferior son ahora $S = 1, Q = 1$, que mantienen la salida $Q = 0$. Mientras

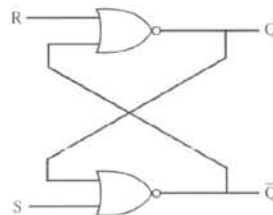


Figura B.24. Implementación del bienestar S-R con puertas NOR.

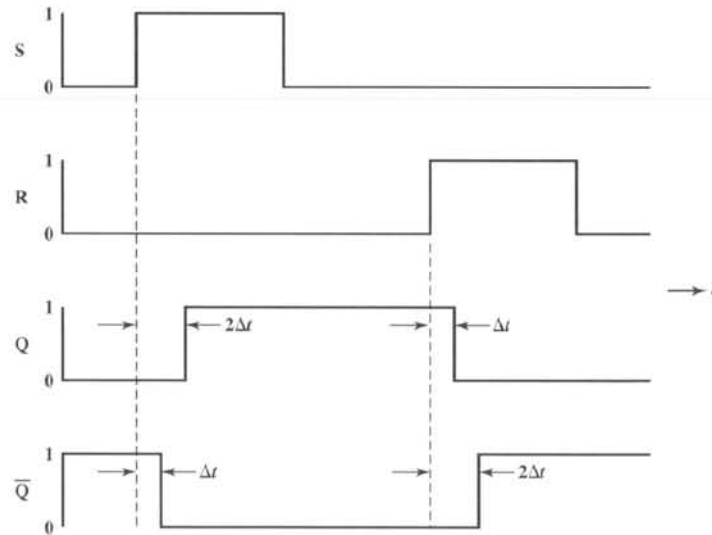


Figura B.25. Diagrama de tiempo del biestable S-R con NOR.

$S = 1$ y $R = 0$, las salidas seguirán siendo $Q = 1$, $\bar{Q} = 0$. Además, si S vuelve a 0, las salidas permanecerán sin cambiar.

La salida R realiza la función contraria. Cuando R vale 1, fuerza $Q = 0$, $\bar{Q} = 1$, sin importar el estado previo de Q y \bar{Q} . De nuevo, hay un tiempo de retardo de $2\Delta t$ antes de que se restablezca la estabilidad (Figura B.25).

El biestable S-R se puede definir con una tabla parecida a la tabla verdad, llamada *tabla característica*, que muestra el siguiente estado o estados de un circuito secuencial en función de los estados y entradas actuales. En el caso del biestable S-R, el estado se puede definir por el valor de Q . La Tabla B.10a muestra la tabla característica resultante. Se observa que las entradas $S = 1$, $R = 1$ no están permitidas, ya que producirían una salida inconsistente (Q y \bar{Q} igual a 0). La tabla se puede expresar de forma más compacta, como en la Tabla B.10b. En la Tabla B.10c se muestra una ilustración del comportamiento del biestable S-R.

Biestable S-R síncrono. La salida del cerrojo S-R cambia, tras un breve tiempo de retardo, en respuesta a un cambio en la entrada. Esto se denomina *operación asíncrona*. La mayoría de los acontecimientos en computadores digitales están sincronizados por un pulso de reloj, así que los cambios ocurren solo en un pulso de reloj. La Figura B.26 muestra esta disposición. Este dispositivo se denomina *biestable S-R síncrono*. Nótese que las entradas S y R se aplican a las entradas de las puertas NOR sólo durante el pulso de reloj.

Biestable D. Un problema con los biestables S-R es que la condición $R = 1$, $S = 1$ debe ser evitada. Una manera de hacerlo es permitir solo una única entrada. El biestable D lo cumple. La Figura B.27 muestra una implementación con puertas y la tabla característica del biestable D. Usando un inversor, las entradas de no reloj de las dos puertas AND garantizan ser una la opuesta de la otra.

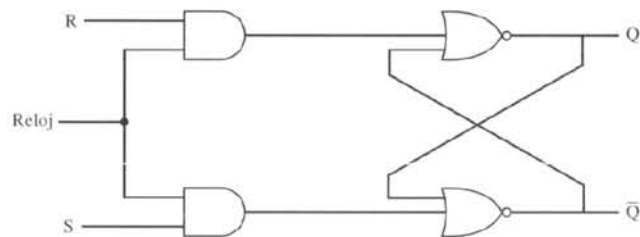
Tabla B.10. El cerrojo S-R.

(a) Tabla característica			(b) Tabla característica simplificada		
Current Inputs	Current State	Next State	S	R	Q_{n+1}
SR	Q_n	Q_{n+1}	0	0	Q_n
00	0	0	0	1	0
00	1	1	1	0	1
01	0	0	1	1	—
01	1	0			
10	0	1			
10	1	1			
11	0	—			
11	1	—			

(c) Respuesta a una serie de entradas										
t	0	1	2	3	4	5	6	7	8	9
S	1	0	0	0	0	0	0	0	1	0
R	0	0	0	1	0	0	1	0	0	0
Q_{n+1}	1	1	1	0	0	0	0	0	1	1

El biestable D a veces se denomina biestable de datos porque es, en efecto, almacén para un bit de datos. La salida del biestable D es siempre igual al valor más reciente aplicado a la entrada. Por tanto, recuerda y produce la última entrada. También se le llama biestable de retardo, porque retrasa un cerp o uno aplicado a la entrada durante un pulso de reloj.

Biestable J-K. Otro biestable útil es el biestable J-K. Como el biestable S-R, tiene dos entradas. Sin embargo, en este caso, todas las combinaciones posibles de los valores de entrada son válidos. La Figura B.28 muestra una implementación con puertas del biestable J-K, y la Figura B.29 muestra su tabla característica (junto con las de los biestables S-R y D). Nótese que las tres primeras


Figura B.26. Bienestable S-R sincronizador.

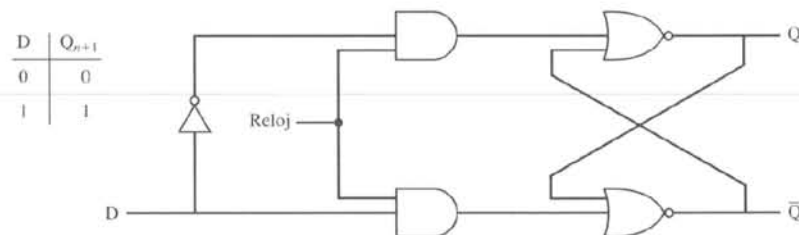


Figura B.27. Biestable D.

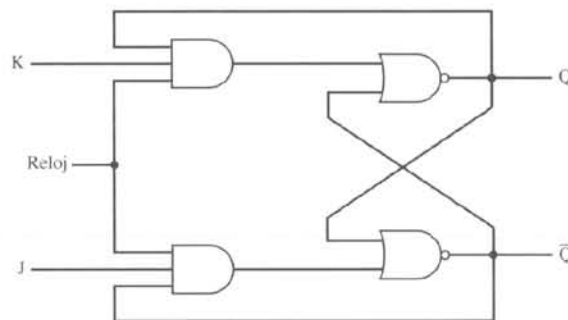


Figura B.28. Biestable J-K.

combinaciones son las mismas que para el biestable S-R. Sin entrada, la salida es estable. La entrada J sola realiza la función de puesta a 1, causando que la salida sea 1; la entrada K solo realiza la función de puesta a cero, provocando que la salida sea 0. Cuando J y K son 1, la función realizada se denomina función de *conmutación*: la salida se invierte.

Si Q vale 1 y se aplica un 1 a J y K, entonces Q se hace 0. El lector debería verificar que la implementación de la Figura B.28 produce esta función característica.

REGISTROS

Como ejemplo del uso de los biestables, examinemos primero uno de los elementos esenciales de la CPU: el registro. Como sabemos, un registro es un circuito digital usado en la CPU para almacenar uno o más bits de datos. Normalmente se usan dos tipos básicos de registros: registros paralelos y de desplazamiento.

Registros paralelos. Un registro paralelo consiste en un conjunto de memorias de un bit que se pueden leer o escribir simultáneamente. Se usa para almacenar datos. Los registros de los que hemos hablado a lo largo de este libro son registros paralelos.

El registro de ocho bits de la Figura B.30 ilustra el funcionamiento de un registro paralelo usando biestables D. Una señal de control, llamada *validación de dato de entrada*, controla la escritura en

Nombre	Símbolo gráfico	Tabla característica															
S-R		<table><tr><th>S</th><th>R</th><th>Q_{n+1}</th></tr><tr><td>0</td><td>0</td><td>Q_n</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>-</td></tr></table>	S	R	Q_{n+1}	0	0	Q_n	0	1	0	1	0	1	1	1	-
S	R	Q_{n+1}															
0	0	Q_n															
0	1	0															
1	0	1															
1	1	-															
J-K		<table><tr><th>J</th><th>K</th><th>Q_{n+1}</th></tr><tr><td>0</td><td>0</td><td>Q_n</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>$\overline{Q_n}$</td></tr></table>	J	K	Q_{n+1}	0	0	Q_n	0	1	0	1	0	1	1	1	$\overline{Q_n}$
J	K	Q_{n+1}															
0	0	Q_n															
0	1	0															
1	0	1															
1	1	$\overline{Q_n}$															
D		<table><tr><th>D</th><th>Q_{n+1}</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	D	Q_{n+1}	0	0	1	1									
D	Q_{n+1}																
0	0																
1	1																

Figura B.29. Biestables básicos.

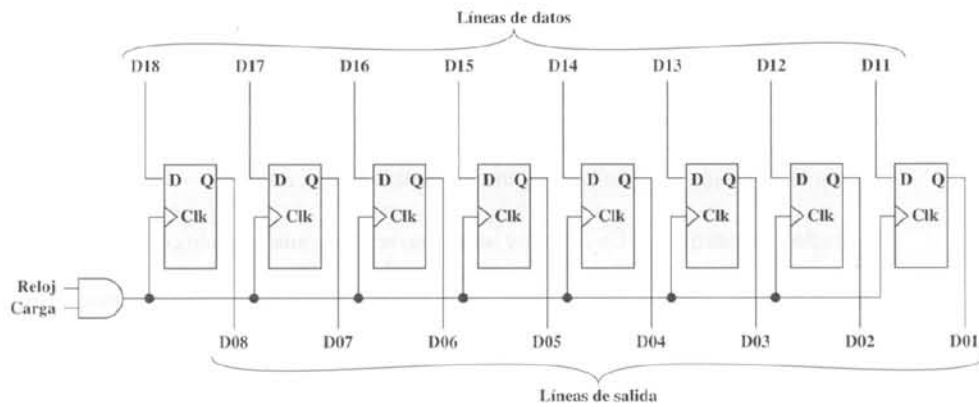


Figura B.30. Registro paralelo de ocho bits.

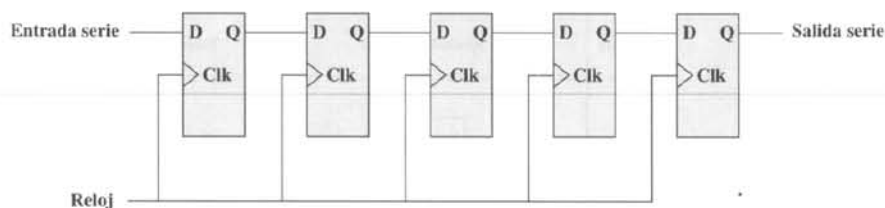


Figura B.31. Registro de desplazamiento de cinco bits.

los registros de los valores provenientes de las líneas de señales, de la D11 a la D18. Estas líneas pueden ser salidas de multiplexores, ya que los datos que se pueden cargar en un registro pueden provenir de una gran variedad de fuentes.

Registro de desplazamiento. Un registro de desplazamiento acepta y/o transfiere información vía serie. Consideremos, por ejemplo, la Figura B.31, que muestra un registro de desplazamiento de cinco bits construido a partir de biestables D síncronos. Los datos se introducen únicamente a través del biestable que está más a la izquierda. Con cada pulso de reloj, los datos se desplazan a la derecha una posición, y el bit más a la derecha se transfiere fuera.

Los registros de desplazamiento se pueden usar como interfaz de dispositivos serie de E/S. Además, pueden usarse en la ALU para realizar desplazamiento lógicos y funciones de rotación. En este último uso, necesitan equiparse con circuitería de lectura/escritura tanto paralela como serie.

CONTADORES

Otra categoría útil de circuitos secuenciales es la de contador. Un contador es un registro cuyo valor se puede incrementar fácilmente en 1, módulo la capacidad del registro. Así, un registro hecho con n biestables puede contar hasta $2^n - 1$. Cuando el contador se incrementa más allá de su valor máximo, se pone a 0. Un ejemplo de un contador en la CPU es el contador de programa.

Los contadores pueden ser asíncronos o síncronos, dependiendo de la forma en que operen. Los contadores asíncronos son relativamente lentos, ya que la salida de un biestable produce un cambio en el estado del siguiente biestable. En un contador síncrono, todos los biestables cambian de estado a la vez. Como el último tipo es mucho más rápido, es el tipo que se usa en CPU. Sin embargo, es útil empezar la discusión con una descripción del contador asíncrono.

Contador asíncrono. Un contador asíncrono se denomina también contador de onda (*ripple*), ya que el cambio que se produce para incrementar el contador empieza en un extremo y se transfiere como una «onda» hasta el otro extremo. La Figura B.32 muestra una implementación de un contador de cuatro bits usando biestables J-K, junto con un diagrama de tiempo que ilustra su comportamiento. El diagrama de tiempo está idealizado ya que no muestra el retardo de propagación que se produce cuando las señales bajan en la serie de biestables. La salida del biestable más a la izquierda (Q_0) es el bit menos significativo. El diseño se podría ampliar fácilmente a un número arbitrario de bits añadiendo en cascada más biestables.

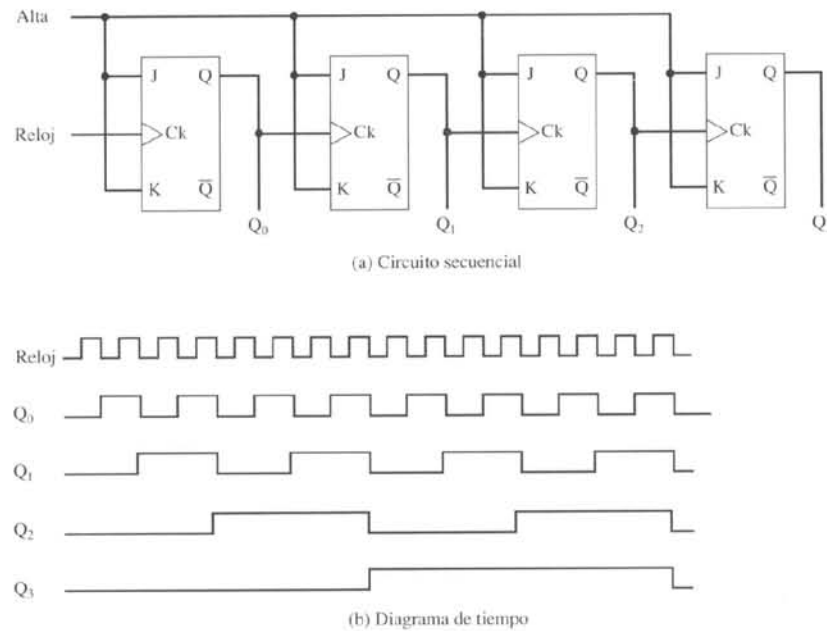


Figura B.32. Contador ondulado.

En la implementación mostrada, el contador se incrementa con cada pulso de reloj. Las entradas J y K de cada biestables se mantienen a 1 constante. Esto quiere decir que, cuando hay un pulso de reloj, la salida en Q se invierte (de 1 a 0; de 0 a 1). Nótese que el cambio de estado se produce cuando cae el flanco del pulso de reloj; esto se conoce como biestable disparado por flanco. Usando biestables que responden a la transición en un pulso de reloj, en vez de en el pulso mismo, proporciona un control del tiempo mejor en circuitos complejos. Si se analizan los patrones de salida de este contador, se puede ver el ciclo 0000, 0001..., 1110, 1111, 0000, etc.

Contadores síncronos. El contador asíncrono tiene la desventaja del retardo asociado al cambio de valor, que es proporcional al tamaño del contador. Para superar esta desventaja, las CPU usan contadores síncronos, en los que todos los biestables del contador cambian al mismo tiempo. En esta subsección, presentamos el diseño de un contador síncrono de tres bits. Al mismo tiempo, ilustraremos algunos conceptos básicos en el diseño de circuitos síncronos.

Para un contador de tres bits, necesitamos tres biestables. Vamos a usar biestables J-K. Llamemos a las salidas sin complementar de los tres biestables A, B, y C respectivamente, donde C representa el bit menos significativo. El primer paso es construir la tabla verdad que relaciona las entradas J-K con las salidas, para poder diseñar el resto del circuito. Dicha tabla verdad se muestra en la Figura B.33a. Las primeras tres columnas muestran las combinaciones posibles de las salidas A, B y C. Están listadas en el orden en que aparecen cuando se incrementa el contador. Cada fila indica el valor actual de A, B, C y las entradas a los tres biestables que se necesitarán para obtener el próximo valor de A, B, y C.

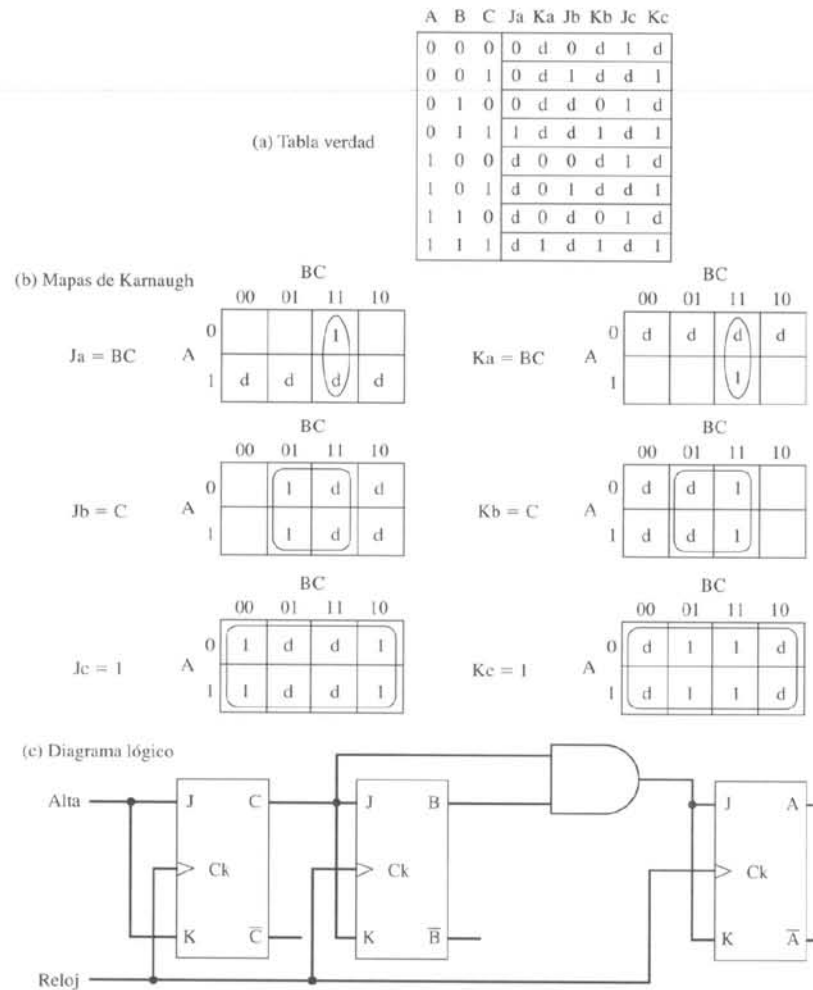


Figura B.33. Diseño de un contador síncrono.

Para comprender la forma en la que se ha hecho la tabla verdad de la Figura B.33a, puede ser útil reestructurar la tabla característica del biestable J-K. Recordemos que esta tabla verdad se presentaba como sigue:

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	\overline{Q}_{n+1}

En esta forma, la tabla muestra el efecto que las entradas J y K tienen en la salida. Ahora consideremos la siguiente reorganización de la misma información:

Q_n	J	K	Q_{n+1}
0	0	d	0
0	1	d	1
1	d	1	0
1	d	0	1

En esta forma, la tabla proporciona el valor de la siguiente salida cuando se conocen las entradas y la salida actual. Esta es exactamente la información que se necesita para diseñar el contador o, de hecho, cualquier circuito secuencial. En esta forma, la tabla se denomina *tabla de excitaciones*.

Volvamos a la Figura B.33a. Consideremos la primera fila. Queremos que el valor de A permanezca a 0, que el valor de B a 0, y que el valor de C cambie de 0 a 1 con la siguiente aplicación del pulso de reloj. La tabla de excitaciones muestra que para mantener una salida a 0, debemos tener las entradas J = 0 e indiferencia para K. Para que haya una transición de 0 a 1, las entradas deben ser J = 1 y K = d. Estos valores se encuentran en la primera fila de la tabla. Con un razonamiento similar, se puede rellenar el resto de la tabla.

Una vez construida la tabla verdad de la Figura B.33a, vemos que la tabla muestra los valores requeridos para todas las entradas J y K como funciones de los valores actuales de A, B, y C. Con la ayuda de mapas de Karnaugh, podemos obtener las expresiones booleanas para estas seis funciones. Esto se muestra en la parte b de la figura. Por ejemplo, el mapa de Karnaugh de la variable Ja (la entrada J del biestable que produce la salida A) da lugar a la expresión $J_a = BC$. Cuando se derivan las seis expresiones, es un problema sencillo diseñar el circuito real, como se muestra en la parte c de la figura.

B.5. LECTURAS RECOMENDADAS Y SITIOS WEB

[GREG98] es una introducción fácil de leer sobre los conceptos de este capítulo. [STON96] es una excelente introducción corta. Hay una serie de libros de texto que hacen un tratamiento más profundo; como [MANO04] y [FARH04].

- FARH04** Farhat, H. *Digital Design and Computer Organization*. Boca Raton: CRC Press, 2004.
- GREG98** Gregg, J. *Ones and Zeros: Understanding Boolean Algebra, Digital Circuits, and the Logic of Sets*. New York: Wiley, 1998.
- MANO04** Mano, M., and Kime, C. *Logic and Computer Design Fundamentals*. Upper Saddle River, NJ: Prentice Hall, 2004.
- STON96** Stonham, T. *Digital Logic Techniques*. London: Chapman & Hall, 1996.



SITIO WEB RECOMENDADO

- **Lógica digital:** colección útil de diagramas de circuitos interactivos.

B.6. PROBLEMAS

- B.1.** Construir la tabla verdad de las siguientes expresiones booleanas:
- $ABC + \overline{A}BC$
 - $ABC + \overline{A}BC + \overline{A}BC$
 - $A(\overline{B}C + \overline{B}C)$
 - $(A + B)(A + C)(\overline{A} + \overline{B})$
- B.2.** Simplificar las siguientes expresiones aplicando la ley conmutativa:
- $A \cdot \overline{B} + \overline{B} \cdot A + C \cdot D \cdot E + \overline{C} \cdot D \cdot E + E \cdot \overline{C} \cdot D$
 - $A \cdot B + A \cdot C + B \cdot A$
 - $(L \cdot M \cdot N)(A \cdot B)(C \cdot D \cdot E)(M \cdot N \cdot L)$
 - $F \cdot (K + R) + S \cdot V + W \cdot \overline{X} + V \cdot S + \overline{X} \cdot W + (R + K) \cdot F$
- B.3.** Aplicar el teorema de DeMorgan a las siguientes ecuaciones:
- $F = \overline{V + A + L}$
 - $F = \overline{A} + \overline{B} + \overline{C} + \overline{D}$
- B.4.** Simplificar las siguientes expresiones:
- $A = S \cdot T + V \cdot W + R \cdot S \cdot T$
 - $A = T \cdot U \cdot V + X \cdot Y + Y$
 - $A = F \cdot (E + F + G)$
 - $A = (P \cdot Q + R + S \cdot T)T \cdot S$
 - $A = \overline{\overline{D} \cdot \overline{D} \cdot E}$
 - $A = Y \cdot (W + X + \overline{Y} + \overline{Z}) \cdot Z$
 - $A = (B \cdot E + C + F) \cdot C$
- B.5.** Obtener la operación XOR a partir de las operaciones booleanas básicas AND, NOR y NOT.
- B.6.** Dibujar el diagrama lógico de una función AND de tres entradas con puertas NOR y NOT.
- B.7.** Escribir la expresión Booleana de una puerta NAND de cuatro entradas.
- B.8.** Se usa un circuito combinacional para controlar un visualizador de dígitos decimales de 7 segmentos, como se muestra en la Figura B.34. El circuito tiene cuatro entradas, que proporciona el código de 4 bits usado en representación decimal compacta (010 = 0000, ..., 910 = 1001). Las siete salidas definen que

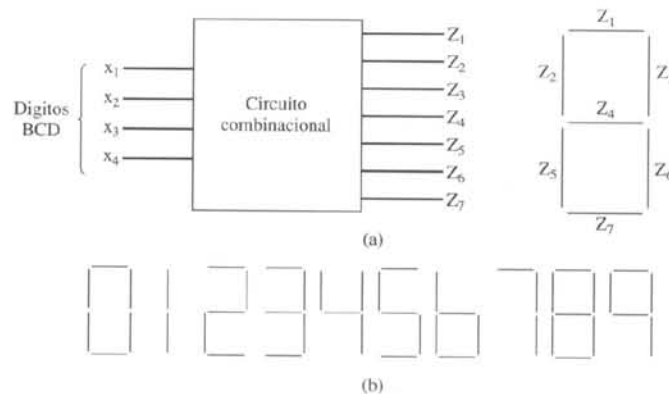


Figura B.34. Ejemplo de un visualizador LED de siete segmentos.

segmentos se van a activar para visualizar un dígito decimal dado. Nótese que no se necesitan algunas combinaciones de entrada ni sus salidas.

- (a) Desarrollar la tabla verdad para este circuito.
- (b) Expresar la tabla verdad en la forma SOP.
- (c) Expresar la tabla verdad en la forma POS.

B.9. Diseñar un multiplexor de 8 a 1.

B.10. Añadir una línea adicional a la Figura B.15 para que funcione como un demultiplexor.

B.11. El código Gray es un código binario para enteros. Difiere de la representación binaria ordinaria en que solo cambia un único bit entre la representación de dos números cualesquiera. Esto es útil en aplicaciones como contadores o conversores analógico digitales donde se genera una secuencia de números. Como solo cambia un bit

Código binario	Código gray
000	000
001	001
010	011
011	010
100	110
101	111
110	101
111	100

Diseñar un circuito que convierta un código binario en un código Gray.

B.12. Diseñar un decodificador de 5×32 usando cuatro decodificadores de 3×8 (con entradas de habilitación) y un decodificador de 2×4 .

B.13. Implementar el sumador completo de la Figura B.22 con solo cinco puertas (*Ayuda:* algunas de las puertas son XOR).

B.14. Considerar la Figura B.22. Suponer que cada puerta produce un retardo de 10ns. Entonces, la salida de suma es válida tras 30 ns y la salida de acarreo tras 0 ns. ¿Cuál es el tiempo total de suma de un sumador de 32 bits

- (a) implementado sin acarreo anticipado, como en la Figura B.21?
- (b) implementado con acarreo anticipado y usando sumadores de 8 bits, como en la Figura B.23?