Capítulo 4

CIRCUITOS DIGITALES

4.1 REPRESENTACIÓN GRÁFICA DE CIRCUITOS

Ahora estamos en condiciones de representar gráficamente los circuitos digitales cuyas expresiones o tablas de verdad poseemos y simplificamos. Necesitaremos mínimamente para ello, símbolos que representen a los diferentes operadores del álgebra de las funciones de conmutación. A saber: el NOT, el AND y el OR.

En la figura 4.1 están representadas cada una de los símbolos que usaremos. A el dispositivo que implementa cada uno de estos operadores le llamaremos *compuerta*. Por cierto se ha incluido en la figura la representación de una compuerta asociada al XOR, una operación que no es de las que podríamos llamar *básicas*, ya que el XOR de A y B, denotado con el operador $\oplus$ se define como:

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

La representación del XOR se ha incluido por ser muy usual, la tabla de verdad es la 4.1.

Estos símbolos gráficos básicos admiten modificadores, una pequeña *bolita* en un cable, se entrada o salida de una compuerta indica que la se;al que viaja por ese cable se ha negado. De hecho el símbolo del negador era originalmente el de un rectificador y al añadirle la bolita se convierte en inversor. Suelen añadirse también líneas de entrada a las compuertas, así podemos tener OR o AND de tres o cuatro entradas.

A	**B**	**A⊕B**
0	0	0
0	1	1
1	0	1
1	1	0

Tabla 4.1. Tabla de verdad del OR exclusivo, disyunción exclusiva o XOR.

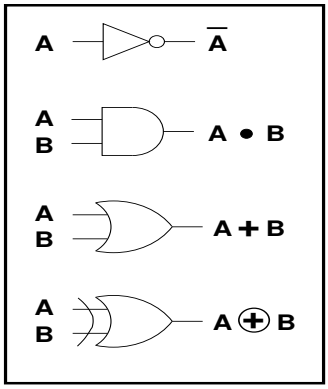

Figura 4.1. Representación gráfica de las principales compuertas. De arriba hacía abajo: NOT, AND, OR y XOR. Esta última no es indispensable, pero sí muy conveniente.

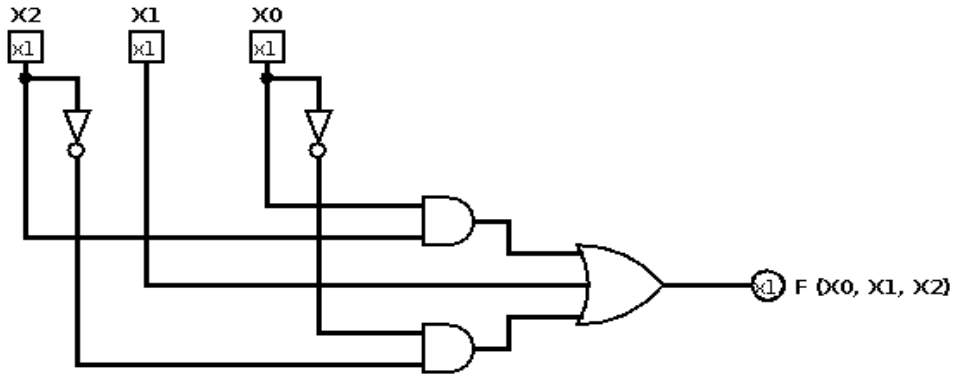

Figura 4.2. Circuito para calcular la función de la tabla 3.2.

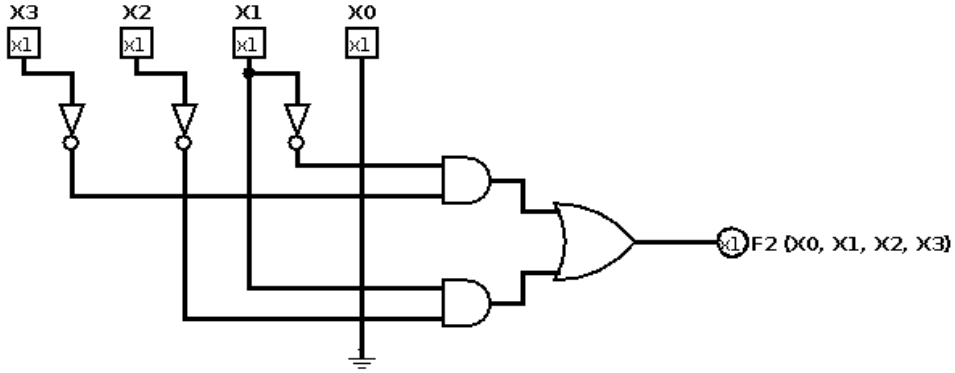

Figura 4.3. Circuito para calcular la función de la tabla 3.5.

■ **EJEMPLO 4.1**

El circuito para calcular la función de la tabla 3.2 se muestra en la figura 4.2.

■

■ **EJEMPLO 4.2**

El circuito para calcular la función de la tabla 3.5 se muestra en la figura 4.3.

■

A	B	$\overline{AB}$	$\overline{A+B}$
0	0	1	1
0	1	1	0
1	0	1	0
1	1	0	0

Tabla 4.2. Tabla de verdad del NAND y el NOR.

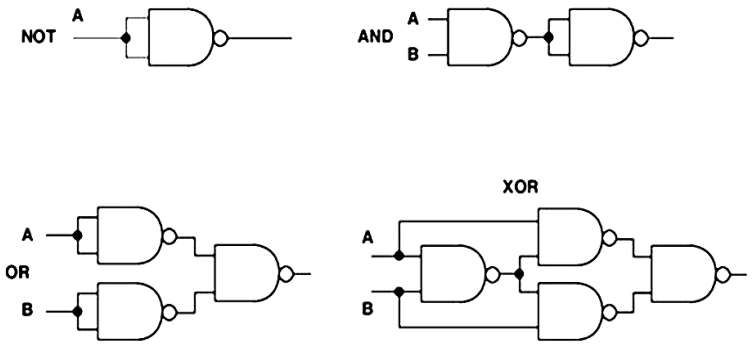

Figura 4.4. Las compuertas NOT, AND, OR y XOR expresadas en función de la NAND.

4.2 CONJUNTO DE OPERADORES COMPLETO

Como cualquier función de conmutación puede escribirse con base en los operadores AND, OR y NOT, se dice que éste es un *conjunto de operadores completo*. Pero no es el único.

Se puede usar sólo NAND

Es posible, por ejemplo, escribir cualquier función de conmutación usando solamente NAND o NOR, cuyas tablas de verdad se muestran en la tabla 4.2. Por supuesto ambos operadores se representan con la respectiva compuerta pero negando su salida con una bolita. Para demostrarlo basta observar la figura 4.4. Formalmente bastaría aplicar las leyes de DeMorgan.

4.3 LÓGICA COMBINACIONAL

Los circuitos que hemos construido hasta ahora poseen la característica de que su salida está completamente determinada por los valores de las

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Tabla 4.3. Tabla de verdad de un semisumador.

líneas de entrada. A este tipo de circuitos se les denomina *combinacionales* y su diseño consiste, esencialmente, en los pasos que hemos ido llevando a cabo:

1. Para cada combinación de valores de las variables de entrada, determinar la salida. Esto es, construir la tabla de verdad del circuito.
2. Minimizar la expresión de la función booleana calculada por el circuito.
3. Diseñar el circuito que la calcula o, alternativamente, especificar el circuito en algún lenguaje de descripción de hardware como VHDL o Verilog.

EJEMPLO 4.3

En la tabla 4.3 se muestran un par de funciones de conmutación. Podríamos considerar que S es la suma y C el acarreo que se produce luego de sumar los bits A y B. No estamos considerando, por ahora, un posible acarreo de entrada, por lo que el circuito que calcula las funciones se denomina *semisumador* de un bit.

Es bastante evidente que $S = A \oplus B$ y $C = A\,B$.

■

EJEMPLO 4.4

La tabla 4.4 muestra un par de funciones de conmutación, S y C_{i_1} cuyo valor depende de tres variables: C_i, A y B. Si pensamos la pareja C_{i+1}, S como el acarreo de salida y la suma de los bits de entrada, respectivamente, lo que tenemos es la especificación

$\mathbf{C_i}$	**A**	**B**	$\mathbf{C_{i+1}}$	**S**
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Tabla 4.4. Tabla de verdad de un par de funciones de conmutación.

de un sumador completo. Para ser más específicos, la tabla 4.4 muestra las funciones suma (S) y acarreo de salida (C_{i+1}) de los operandos A y B con acarreo de entrada C_i.

Podríamos hacer los circuitos que corresponden de acuerdo con los mintérminos, S no puede simplificarse. C_{i+1} queda con tres en vez de cuatro términos al simplificar. Pero podemos ahorrarnos trabajo si pensamos en que el resultado debería ser la suma de A y B, sumada con C_i y el acarreo de salida debería ser el que resulte de sumar A y B o bien el que resulte de sumar esa suma con el acarreo de entrada. Podemos entonces usar dos semisumadores: encadenar las sumas para obtener S y hacer un OR de los acarreos para obtener C_{i+1}.

En la figura 4.5 se muestra el circuito correspondiente, se le denomina *sumador completo* o *full adder* en inglés.

■

Con el resultado del ejemplo previo podríamos pensar en hacer un sumador de un número arbitrario de bits. En la figura 4.6 se muestra un sumador de 8 bits.

Así es como se procede normalmente, construyendo módulos que realizan alguna operación básica que luego puede usarse para llevar a cabo una más completa.

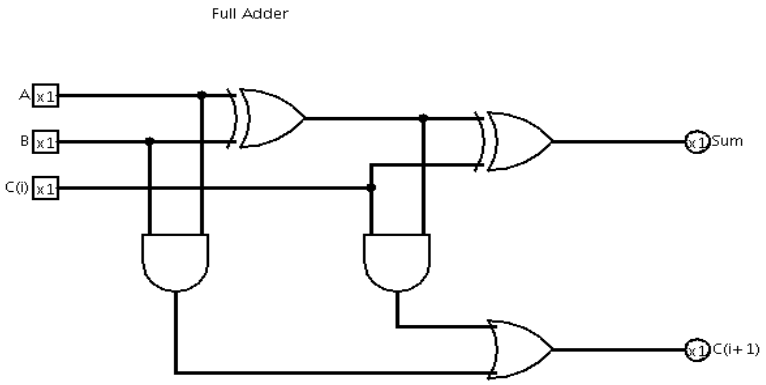

Figura 4.5. Sumador completo de un bit.

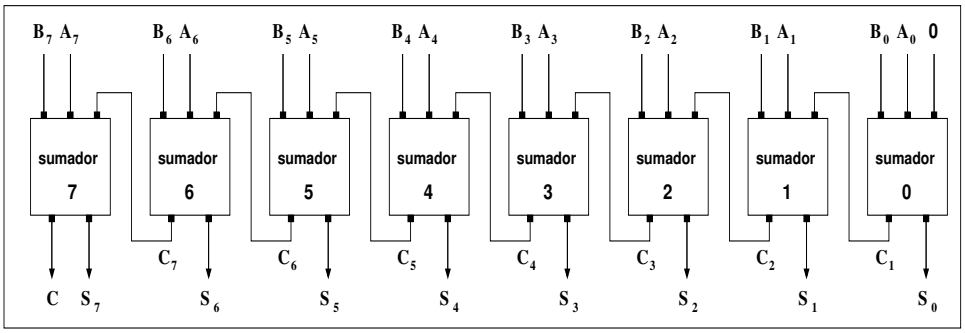

Figura 4.6. Sumador de 8 bits construido con base en sumadores de un bit.

Este sumador de 8 bits funcionaría, ciertamente, pero no resultaría muy práctico. ¿Cuál sería el retardo necesario para que acarreo se propague hasta la salida? habría que pasar por dos niveles de compuerta, al menos, por cada sumador, así que se deben pasar por un total de $2n$ compuertas, inaceptable.

Pero podemos hacer otra cosa. Definamos:

$$P_i = A_i \oplus B_i$$

y

$$G_i = A_i \, B_i$$

La suma y el acarreo se expresarían entonces:

$$S_i = P_i \oplus C_i$$

y

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

Acarreo anticipado

Es decir: el acarreo de salida vale uno cuando los dos sumandos valen uno o bien cuando la suma de ellos vale uno y hay un acarreo de entrada. Podemos usar esta última expresión para escribir el valor de los acarreos:

$$\begin{aligned}
C_1 &= G_0 + P_0 \, C_0 \\
C_2 &= G_1 + P_1 \, C_1 = G_1 + P_1 \, G_0 + P_0 \, P_1 \, C_0 \\
C_3 &= G_2 + P_2 \, C_2 = G_2 + P_2 \, G_1 + P_1 \, P_2 \, G_0 + P_0 \, P_1 \, P_2 \, C_0 \\
C_4 &= G_3 + G_2 \, P_3 + G_1 \, P_2 \, P_3 + G_0 \, P_1 \, P_2 \, P_3 + C_0 \, P_0 P_1 P_2 P_3 \\
\vdots &= \vdots \\
C_{n+1} &= G_n + \sum_{k=1}^{n} \left[G_{k-1} \prod_{i=k}^{n} P_i \right] + C_0 \prod_{i=0}^{n} P_i
\end{aligned}$$

En la figura 4.7 se muestra el resultado.

4.4 DECODIFICADORES Y MULTIPLEXORES

Otros circuitos combinacionales de singular importancia y que son, por cierto, similares, son los decodificadores y los multiplexores.

Un decodificador recibe una entrada codificada, un número binario de k bits y entrega un conjunto de 2^k lineas de salida, pero sólo una de ellas con valor 1 y el resto en 0. El código de entrada es el índice de la línea que debe salir en 1. En la figura 4.8 se muestra un decodificador 3 a 8.

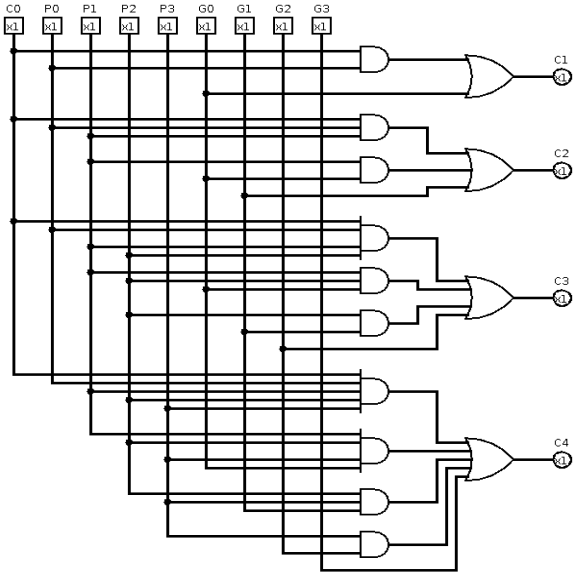

Figura 4.7. Acarreo anticipado.

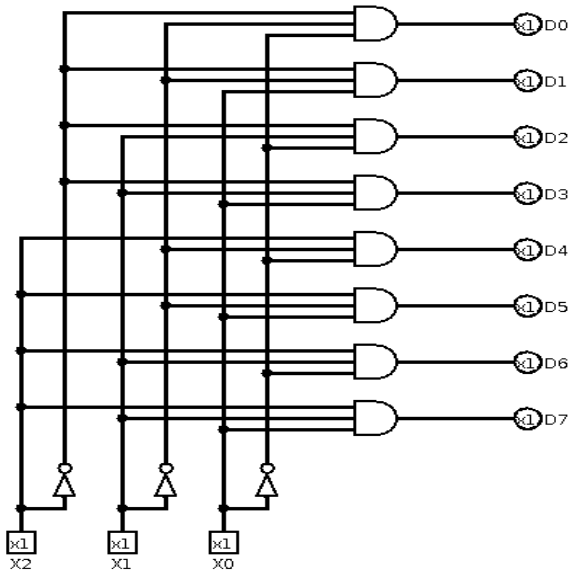

Figura 4.8. Decodificador 3 a 8.

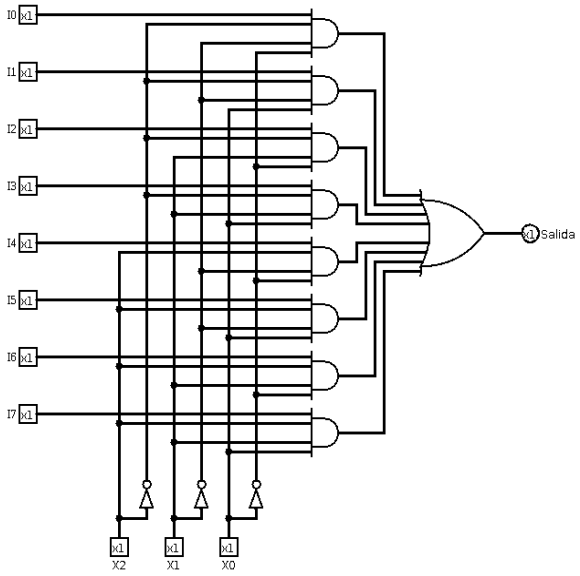

Figura 4.9. Multiplexor 8 a 1. A es el bit más significativo y C el menos significativo.

Un multiplexor es un circuito que, cómo el decodificador, recibe un código de k bits de entrada que determina, la salida. Pero además recibe también como entrada 2^k líneas diferentes, cada una de ellas con valor 0 o 1. El multiplexor tiene sólo una línea de salida y por ella es transportado el valor de una y sólo una de las líneas de entrada, a saber, aquella cuyo índice es el código dado al multiplexor. En la figura 4.9 se muestra un multiplexor 8 a 1.

4.5 LÓGICA SECUENCIAL

Un circuito secuencial es aquel cuya salida depende tanto de sus entradas como del estado en el que se encontraba el circuito al recibir dicha entrada. Esto significa que el circuito debe "saber" en que estado está, debe recordarlo; es una entidad con memoria.

Biestable

Podemos almacenar un bit usando compuertas con un circuito llamado *biestable* (figura 4.10). Una vez que este circuito adquiere una configuración, en teoría, la mantiene *ad infinitum*. Lo malo es que no admite entrada alguna. Necesitamos un circuito que admita una entrada y la mantenga encerrada, esto es lo que se conoce como *cerrojo* o *latch*. De un latch necesitamos que se pueda poner en 1 (*set*) y este valor sea conservado hasta que otra cosa pase, por ejemplo restablecer el latch (dar *reset*). Con la letra Q identificaremos el estado del latch.

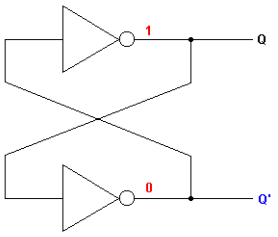

Figura 4.10. Circuito biestable.

S	R	Q	$\overline{Q}$
1	0	1	0
0	0	1	0
0	1	0	1
0	0	0	1
1	1	0	0

Tabla 4.5. Tabla de función del latch SR. Hay dos renglones con $S = R = 0$, el primero suponemos que ocurre luego de hacer set, el segundo luego de reset, es decir, cuando ambas entradas están en cero el circuito mantiene el estado que hayamos establecido previamente.

4.5.1 Latch SR

El latch SR hace justamente lo que hemos descrito. Tiene dos líneas de entrada que, en principio, no pueden estar simultáneamente en 1: *set* y *reset*, para establecer (poner en 1) o borrar (poner en cero) el estado del latch respectivamente. Si ambas líneas son 0 significa que el circuito debe permanecer en el estado que posea actualmente y que, por supuesto, es 0 o 1. El circuito posee dos salidas, una con el estado Q y otra con el complemento de este $\overline{Q}$. En la tabla 4.5 se muestra la función del latch y en la figura 4.11 el circuito implementado con compuertas NOR, se puede también implementar con NAND. Hay que señalar que poner ambas entradas en 1 lleva el latch a un estado inconsistente en el que ambas salidas son 0, lo que no debería ocurrir.

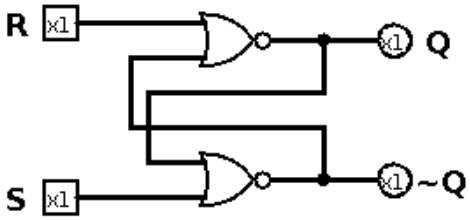

Figura 4.11. Latch SR implementado con compuertas NOR.

$\mathbf{Q_t}$	**S**	**R**	$\mathbf{Q_{t+1}}$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	X
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	X

Tabla 4.6. Tabla de transición de estados del latch SR.

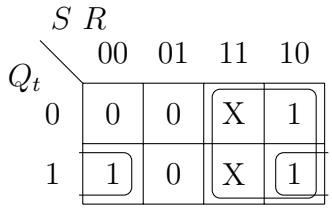

Figura 4.12. Mapa de Karnaugh para las transiciones del latch SR.

S	**R**	Q_{t+1}
0	0	Q_t
0	1	0
1	0	1
1	1	¿?

Tabla 4.7. Tabla característica del latch SR.

Podemos considerar también la tabla de verdad que determina el estado del latch en función de su estado previo y sus entradas (tabla 4.6).

Es posible simplificar la función (véase el mapa de Karnaugh en la figura 4.12), con lo que se obtiene:

$$\begin{aligned} Q_{t+1} &= S + \overline{R}\, Q_t \\ S \cdot R &= 0 \end{aligned} \tag{4.1}$$

A la que se le suele llamar *ecuación característica* y expresa lo mismo que la llamada *tabla característica* 4.7.

4.5.2 Latch D

Una solución rápida y simple para evitar la indeseable condición indeterminada en el latch SR es ponerle una sola línea de entrada y obligar a que S y R sean siempre complementarios. Podríamos de hecho hacer algo más, poner una línea de control que permita la entrada al latch o no. En la figura 4.13 se muestra el resultado.

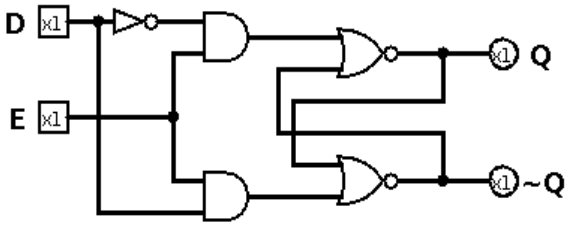

Figura 4.13. Latch D. La línea D es la entrada, la línea E es "enable" y tiene la función de controlar la entrada al latch.

Q_t	**D**	Q_{t+1}
0	0	0
0	1	1
1	0	0
1	1	1

Tabla 4.8. Tabla de transición del latch D.

La tabla 4.8 muestra las transiciones del latch D. Su ecuación es muy simple:

$$Q_{t+1} = D \tag{4.2}$$

La introducción de nuestra línea de control es una buena idea, así el biestable no cambiará cuando cambie su entrada sino cuando decidamos que así debe ser o cuando estemos seguros de que lo que va a entrar no es una se;al espuria. Normalmente estas líneas de control se conectan a un oscilador, un reloj que, con cierta frecuencia, habilita y deshabilita la entrada del latch. En estas condiciones el latch es síncrono y suele llamarse entonces *flip-flop*.

4.5.3 Flip-flop JK

Otra posibilidad de corregir el latch SR es colocándole una líneas de retroalimentación para que en caso de entrar 1 por ambas entradas el resultado sea el estado complementario del actual. El resultado es el flip-flop (por añadirse el reloj) JK mostrado en la figura 4.14.

La tabla de transición es la 4.9, la simplificación mostrada en el mapa de la figura 4.15 da lugar a:

$$Q_{t+1} = J\ \overline{Q_t} + \overline{K}\ Q_t \tag{4.3}$$

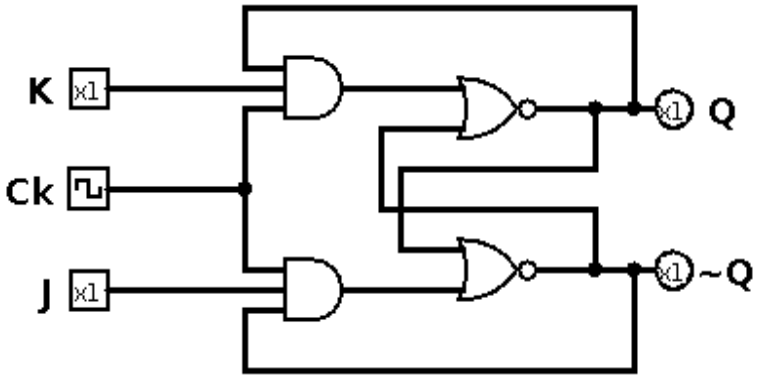

Figura 4.14. Flip-flop JK.

Q_t	J	K	Q_{t+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Tabla 4.9. Tabla de transición de estados del flip-flop JK.

Esto se puede simplificar como se observa en el mapa de Karnaugh de la figura 4.15

Finalmente la tabla característica del flip-flop JK es la 4.10.

4.6 ANÁLISIS Y DISEÑO DE CIRCUITOS SECUENCIALES

Lo visto en la sección anterior se puede resumir en la tabla 4.11.

Máquinas de estados finitos

Usando flip-flops es posible construir, en hardware, máquinas de estados finitos. Esos modelos de cómputo restringidos de acuerdo con el tamaño y las características de acceso a su memoria. Los fip-flops son

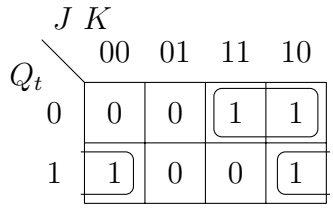

Figura 4.15. Mapa de Karnaugh para las transiciones del flip-flop JK.

J	**K**	Q_{t+1}
0	0	Q_t
0	1	0
1	0	1
1	1	$\overline{Q_t}$

Tabla 4.10. Tabla característica del flip-flop JK.

Q_t	Q_{t+1}	S	R	J	K	D
0	0	0	X	0	X	0
0	1	1	0	1	X	1
1	0	0	1	X	1	0
1	1	X	0	X	0	1

Tabla 4.11. Síntesis de las características de los flip-flops.

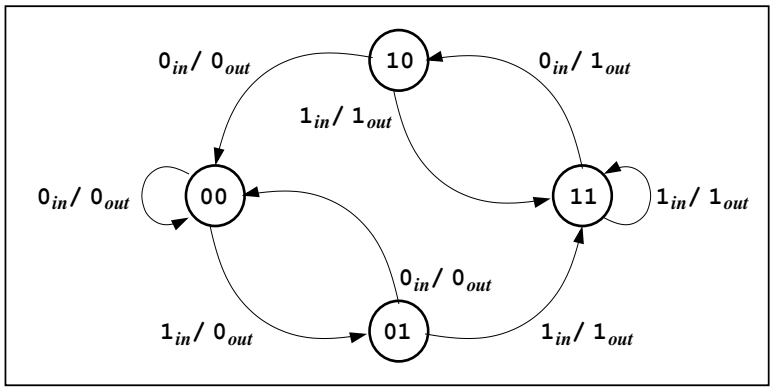

Figura 4.16. Un ejemplo de máquina de estados finitos.

Edo. actual	Edo. Sig.		Salida	
	In=0	In=1	In=0	In=1
00	00	01	0	0
01	00	11	0	1
10	00	11	0	1
11	10	11	1	1

Tabla 4.12. Tabla de transiciones para la máquina de estados finitos.

el medio que necesitamos para recordar el estado en que nos encontramos. Usaremos las funciones características para llevar a cabo las transiciones del autómata.

■ **EJEMPLO 4.5**

En la figura 4.16 se muestra una máquina de estados finitos. Más adelante develaremos su utilidad, por ahora sólo queremos diseñar su dispositivo de control con base en flip-flops. Usaremos los del tipo JK.

Podemos sintetizar el comportamiento de la máquina mediante la tabla 4.12.

Las etiquetas de los estados deben medir 2 bits, dado que hay cuatro estados. Como cada flip-flop puede almacenar un bit, debemos usar entonces dos flip-flops para representar el estado, lla-

A	B	In	A_s	B_s	J_A	K_A	J_B	K_B	Out
0	0	0	0	0	0	X	0	X	0
0	0	1	0	1	0	X	1	X	0
0	1	0	0	0	0	X	X	1	0
0	1	1	1	1	1	X	X	0	1
1	0	0	0	0	X	1	0	X	0
1	0	1	1	1	X	0	1	X	1
1	1	0	1	0	X	0	X	1	1
1	1	1	1	1	X	0	X	0	1

Tabla 4.13. Tabla de transiciones en el estado de los flip-flops.

memos A, al más significativo, y B al menos significativo. Si denotamos con A_s y B_s al contenido *siguiente* de los flip-flops A y B respectivamente, luego de que el circuito reciba una entrada dada In, podemos representar en las cinco primeras columnas de la izquierda de la tabla 4.13 lo que el ocurre al autómata dado su estado actual y la entrada que recibe. En las columnas etiquetadas con J_A y K_A se ha puesto lo que debe recibir por sus entradas J y K el flip-flop A y en las etiquetadas J_B y K_B lo correspondiente al flip-flop B. La última columna (Out) denota lo que se supone debe producir de salida el autómata.

Tenemos ahora completamente especificadas las cinco funciones que necesitamos, a saber: J_A, K_A, J_B, K_B y Out. Luego de simplificarlas como se muestra en los mapas de Karnaugh:

$$J_A = B \cdot In.$$

$$K_A = \overline{B} \cdot \overline{In}.$$

$$J_B = In.$$

$$K_B = \overline{In}.$$

$$Out = B \cdot In + A \cdot In + A \cdot B.$$

El resultado final se muestra en la figura 4.22.

A \ $B\ In$	00	01	11	10
0	0	0	1	0
1	X	X	X	X

Figura 4.17. J_A.

A \ $B\ In$	00	01	11	10
0	X	X	X	X
1	1	0	0	0

Figura 4.18. K_A.

A \ $B\ In$	00	01	11	10
0	0	1	X	X
1	0	1	X	X

Figura 4.19. J_B.

A \ $B\ In$	00	01	11	10
0	X	X	0	1
1	X	X	0	1

Figura 4.20. K_B.

A \ $B\ In$	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Figura 4.21. *Out*.

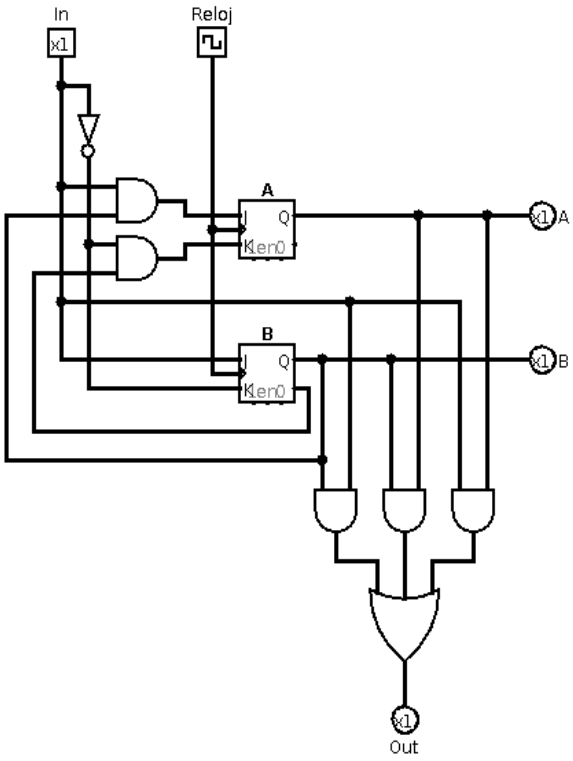

Figura 4.22. Circuito secuencia para el autómata del ejemplo.