FUNDAMENTOS DE LÓGICA DIGITAL CON DISEÑO VHDL

SEGUNDA EDICIÓN

Stephen Brown y Zvonko Vranesic
Departamento de Ingeniería Eléctrica y Computación
University of Toronto

Traducción
Lorena Peralta Rosales
Víctor Campos Olguín
Traductores profesionales

Revisión técnica
Jorge Valeriano Assem
Coordinador de la carrera de Ingeniería en Computación
Universidad Nacional Autónoma de México

Felipe Antonio Trujillo Fernández
Profesor del Departamento de Ingenierías
Universidad Iberoamericana, Campus Santa Fe

MÉXICO • BOGOTÁ • BUENOS AIRES • CARACAS • GUATEMALA • LISBOA
MADRID • NUEVA YORK • SAN JUAN • SANTIAGO
AUCKLAND • LONDRES • MILÁN • MONTREAL • NUEVA DELHI
SAN FRANCISCO • SINGAPUR • ST. LOUIS • SIDNEY • TORONTO

capítulo

7

FLIP-FLOPS, REGISTROS, CONTADORES Y UN PROCESADOR SIMPLE

OBJETIVOS DEL CAPÍTULO

En este capítulo se estudian los temas siguientes:

- Circuitos lógicos que pueden almacenar información
- Flip-flops, que almacenan un solo bit
- Registros, que almacenan varios bits
- Registros de corrimiento, que desplazan el contenido del registro
- Contadores de diversos tipos
- Constructores de VHDL usados para implementar elementos de almacenamiento
- Diseño de subsistemas pequeños

En capítulos anteriores consideramos los circuitos combinacionales en los que el valor de cada salida depende exclusivamente de los valores de las señales aplicadas a las entradas. Hay otro tipo de circuitos lógicos en los que los valores de la salida no sólo dependen de los valores presentes de las entradas, sino también del comportamiento previo del circuito. Estos circuitos incluyen elementos de almacenamiento que guardan los valores de las señales lógicas. Se dice que el contenido de los elementos de almacenamiento representa el *estado* del circuito. Cuando las entradas del circuito cambian de valor, los nuevos valores de entrada dejan el circuito en el mismo estado o hacen que éste entre en un estado nuevo. Con el tiempo el circuito pasa por una secuencia de estados como resultado de los cambios en las entradas. Los circuitos que se comportan de esta manera se conocen como *circuitos secuenciales*.

En este capítulo estudiaremos los circuitos que se utilizan como elementos de almacenamiento, pero antes, por medio de un ejemplo simple, explicaremos por qué son necesarios. Supóngase que deseamos controlar un sistema de alarma como se muestra en la figura 7.1. El mecanismo de alarma responde a la entrada de control $On/\overline{Off}$. Está encendido cuando $On/\overline{Off} = 1$, y está pagado cuando $On/\overline{Off} = 0$. La operación que se busca es que la alarma se active cuando el sensor genere una señal de voltaje positivo, *Set*, en respuesta a algún hecho no deseado. Una vez que la alarma se activa debe permanecer en tal estado aun si la salida del sensor regresa a 0. La alarma se apaga manualmente por medio de una entrada *Reset*. El circuito requiere un elemento de memoria para recordar que la alarma ha de estar activa hasta que la señal *Reset* llegue.

En la figura 7.2 se muestra un elemento de memoria rudimentario que consta de un ciclo con dos inversores. Si suponemos que $A = 0$, entonces $B = 1$. El circuito mantendrá estos valores indefinidamente. Decimos entonces que el circuito se halla en el *estado* definido por ellos. Si suponemos que $A = 1$, entonces $B = 0$, y el circuito mantendrá este estado de manera indefinida. Por tanto, el circuito tiene dos estados posibles. Este circuito no es útil, ya que carece de un medio práctico para cambiar de estado.

En la figura 7.3 se muestra un circuito más útil. Incluye un mecanismo para cambiar el estado del circuito de la figura 7.2 mediante dos compuertas de transmisión del tipo estudiado en la sección 3.9. Una de las compuertas, *TG*1, se usa para conectar la terminal de entrada *Data* con el

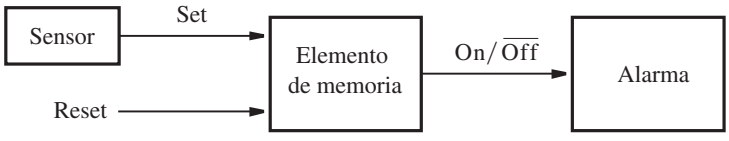


Figura 7.1 Control de un sistema de alarma.

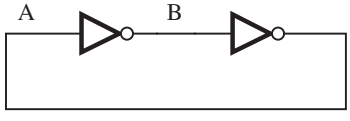


Figura 7.2 Un elemento de memoria simple.

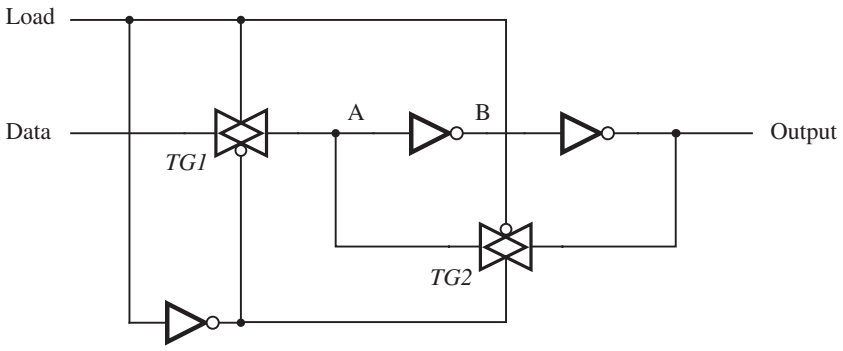


Figura 7.3 Un elemento de memoria controlado.

punto *A* del circuito. La otra compuerta, *TG*2, sirve como interruptor en el *lazo de retroalimentación* que mantiene el estado del circuito. Las compuertas de transmisión están controladas por la señal *Load*. Si *Load* = 1, entonces *TG*1 está abierta y el punto *A* tendrá el mismo valor que la entrada *Data*. Como el valor almacenado actualmente en *Output* tal vez no sea el mismo que *Data*, el lazo de retroalimentación se interrumpe al hacer que *TG*2 se cierre cuando *Load* = 1. Cuando *Load* = 0, entonces *TG*1 se cierra y *TG*2 se abre. La ruta de retroalimentación se cierra y el elemento de memoria conservará su estado siempre que *Load* = 0. Este elemento de memoria no puede aplicarse de modo directo al sistema de la figura 7.1, pero es útil para muchas otras aplicaciones, como veremos más adelante.

7.1 EL LATCH BÁSICO

En vez de usar las compuertas de transmisión podemos construir un circuito similar empleando compuertas lógicas ordinarias. En la figura 7.4 se presenta un elemento de memoria construido con compuertas NOR. Sus entradas, *Set* y *Reset*, proveen los medios para cambiar el estado, Q, del circuito. Una forma más común de dibujar éste se muestra en la figura 7.5*a*, donde se dice que las dos compuertas NOR están conectadas en un estilo de acoplamiento cruzado. El circuito se conoce como *latch básico* y su comportamiento se describe en la tabla de la figura 7.5*b*. Cuando las dos entradas, *R* y *S*, son iguales a cero, el latch mantiene su estado actual, que puede ser $Q_a = 0$ y $Q_b = 1$, o $Q_a = 1$ y $Q_b = 0$, lo cual se indica en la tabla señalando que las salidas Q_a y Q_b tienen valores

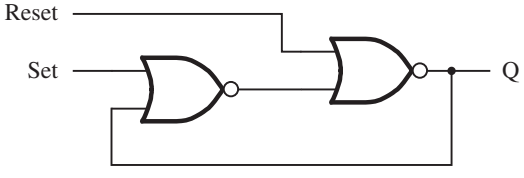
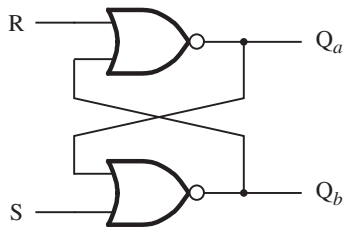


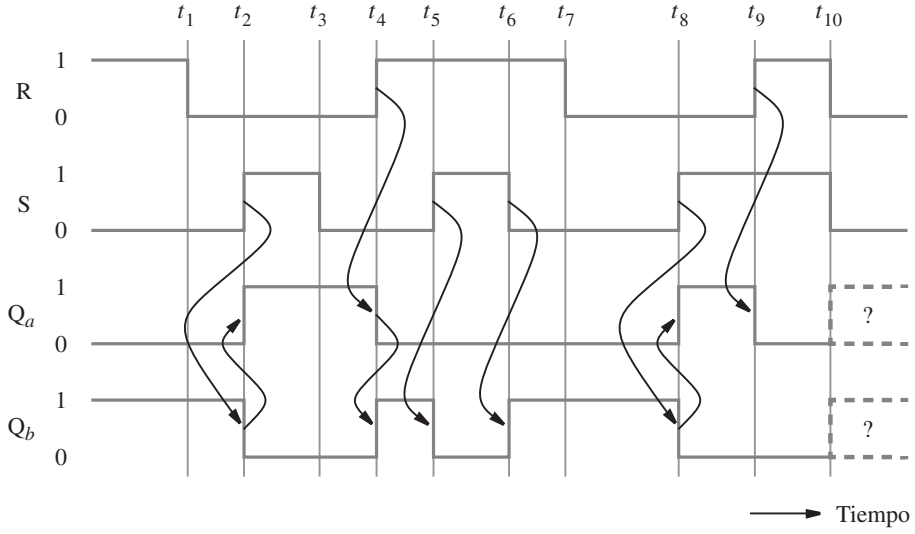
Figura 7.4 Un elemento de memoria con compuertas NOR.



a) Circuito

S	R	Q_a	Q_b	
0	0	0/1	1/0	(sin cambio)
0	1	0	1	
1	0	1	0	
1	1	0	0	

b) Tabla característica



c) Diagrama de tiempo

Figura 7.5 Un latch básico construido con compuertas NOR.

0/1 y 1/0, respectivamente. Obsérvese que en este caso Q_a y Q_b se complementan mutuamente. Cuando $R = 0$ y $S = 1$, el latch se *fija* en un estado donde $Q_a = 1$ y $Q_b = 0$ (estado conocido como *set*). Cuando $R = 1$ y $S = 0$, el latch se regresa a un estado en el que $Q_a = 0$ y $Q_b = 1$ (estado conocido como *reset*). La cuarta posibilidad es tener $R = S = 1$. En este caso, tanto Q_a como Q_b serán 0. La tabla de la figura 7.5*b* se parece a una tabla de verdad. Sin embargo, como no representa un circuito combinacional donde los valores de las salidas se determinan exclusivamente por los valores presentes en las entradas, recibe el nombre de *tabla característica* en vez de tabla de verdad.

En la figura 7.5*c* se presenta un diagrama de tiempo para el latch, suponiendo que el retraso de propagación a través de las compuertas NOR es insignificante. Desde luego, en un circuito de verdad los cambios en la forma de la onda estarán retrasados de acuerdo con los retrasos de propagación de las compuertas. Supóngase que inicialmente $Q_a = 0$ y $Q_b = 1$. El estado del latch permanece sin cambios hasta el tiempo t_2, cuando S se vuelve igual a 1, lo que ocasiona

que Q_b cambie a 0, lo cual a su vez hace que Q_a cambie a 1. La relación de causalidad se indica en el diagrama por medio de flechas. Cuando S pasa a 0 en t_3 no hay cambio en el estado porque tanto S como R son iguales a 0. En t_4 tenemos $R = 1$, lo que ocasiona que Q_a se establezca en 0, lo cual a su vez hace que Q_b pase a 1. En t_5 tanto S como R son iguales a 1, lo que causa que tanto Q_a como Q_b sean iguales a 0. Cuando S regresa a 0 en t_6 Q_b se vuelve igual a 1 de nuevo. En t_8 tenemos $S = 1$ y $R = 0$, lo que produce que $Q_b = 0$ y $Q_a = 1$. Una situación interesante ocurre en t_{10}. De t_9 a t_{10} tenemos $Q_a = Q_b = 0$ porque $R = S = 1$. Ahora, si tanto R como S cambian a 0 en t_{10}, entonces Q_a y Q_b se establecerán en 1. Pero tener Q_a y Q_b iguales a 1 forzará de inmediato que $Q_a = Q_b = 0$. Habrá una oscilación entre $Q_a = Q_b = 0$ y $Q_a = Q_b = 1$. Si los retrasos a través de las dos compuertas NOR son exactamente iguales, la oscilación continuará de manera indefinida. En un circuito real siempre habrá una diferencia en los retrasos por esas compuertas y con el tiempo el latch se fijará en uno de sus dos estados estables, pero no sabemos en cuál de ellos. Esta incertidumbre se indica con formas de ondas en líneas punteadas.

Las oscilaciones analizadas en el párrafo anterior ilustran que si bien el latch básico es un circuito simple, debe realizarse un análisis detallado para advertir por completo este comportamiento. En general, todo circuito que contenga una o más rutas de retroalimentación, tal que el estado del circuito dependa de los retrasos de propagación a través de compuertas lógicas, ha de diseñarse con cuidado. Comentaremos de manera pormenorizada las cuestiones relativas al tiempo en el capítulo 9.

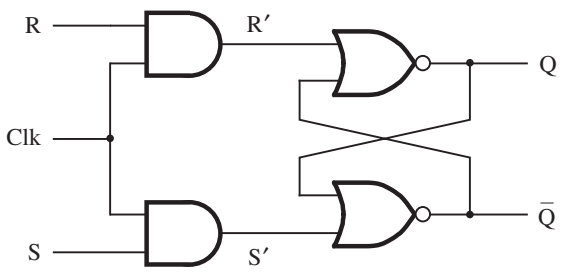
El latch de la figura 7.5*a* puede realizar las funciones necesarias para el elemento de memoria de la figura 7.1 al conectar la señal *Set* a la entrada S y *Reset* a la R. La salida Q_a proporciona la señal $On/\overline{Off}$ buscada. Para inicializar el funcionamiento del sistema de alarma, el latch se vuelve a establecer en 0 (reset). De esta forma la alarma se apaga. Cuando el sensor genera el valor lógico 1, el latch se establece en 1 y Q_a se vuelve igual a 1. Esto activa el mecanismo de alarma. Si la salida del sensor vuelve a 0, el latch conserva su estado donde $Q_a = 1$; por tanto, la alarma permanece activa. El único modo de apagarla es al reinicializar el latch, lo que se logra haciendo que la entrada *Reset* sea igual a 1.

7.2 LATCH SR ASÍNCRONO

En la sección 7.1 vimos que el latch SR básico puede servir como un elemento de memoria útil, ya que recuerda su estado cuando las dos entradas S y R son 0. Modifica su estado en respuesta a los cambios en las señales de estas entradas. Los cambios de estado ocurren en el momento en que suceden los cambios en las señales. Si no podemos controlar cuándo ocurren tales cambios, entonces no sabemos cuándo el latch puede cambiar de estado.

En el sistema de alarma de la figura 7.1, tal vez se quiera habilitarlo o inhabilitarlo todo por medio de una entrada de control, *Enable*. Así, cuando el sistema esté habilitado funcionará como se describió antes. En el modo inhabilitado, el cambiar la entrada *Set* de 0 a 1 no ocasionaría la activación de la alarma. El latch de la figura 7.5*a* no puede brindar la operación buscada, pero su circuito puede modificarse para responder a las señales de entrada S y R sólo cuando *Enable* = 1; de lo contrario, mantendrá su estado.

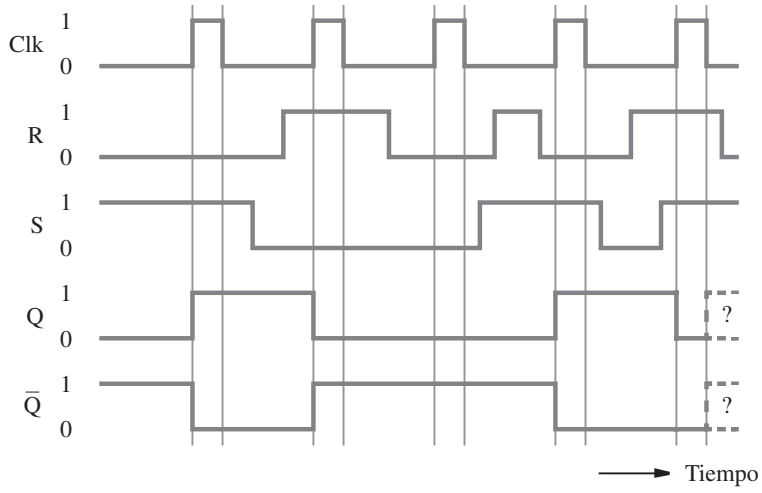
El circuito modificado se presenta en la figura 7.6*a*. Incluye dos compuertas AND que proporcionan el control descrito. Cuando la señal de control *Clk* es igual a 0, las entradas al latch S' y R' serán 0, independientemente de los valores de las señales S y R. Por consiguiente, el latch mantendrá



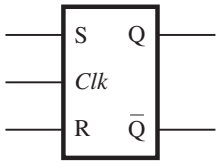
a) Circuito

Clk	S	R	Q(*t* + 1)
0	x	x	Q(*t*) (sin cambio)
1	0	0	Q(*t*) (sin cambio)
1	0	1	0
1	1	0	1
1	1	1	x

b) Tabla característica



c) Diagrama de tiempo



d) Símbolo gráfico

Figura 7.6 Latch SR asíncrono.

su estado actual siempre que *Clk* = 0. Cuando *Clk* cambia a 1, las señales S' y R' serán iguales que las señales *S* y *R*, respectivamente. Por tanto, en este modo el latch se comportará como describimos en la sección 7.1. Obsérvese que hemos usado el nombre *Clk* para la señal de control que permite que el latch se establezca en 1 o se inicialice en 0, en vez de llamarla señal *Enable*. La razón es que estos circuitos se usan en los sistemas digitales donde se busca permitir que los cambios de

estado de los elementos de memoria ocurran sólo en intervalos de tiempo bien definidos, como si estuvieran controlados por un reloj. La señal de control que define estos intervalos de tiempo suele denominarse señal de reloj *clock*. El nombre *Clk* pretende reflejar la naturaleza de esta señal.

Los circuitos de este tipo, que usan una señal de control, se denominan *latches asíncronos*. Puesto que nuestro circuito cuenta con la capacidad de establecerse en 1 e inicializarse en 0, se le llama *latch SR asíncrono* [*S* por *set* y *R* por *reset*, establecerse e inicializarse, respectivamente]. En la figura 7.6*b* se describe su comportamiento. Ahí se define el estado de la salida Q en el tiempo $t + 1$, es decir, $Q(t + 1)$, como una función de las entradas *S*, *R* y *Clk*. Cuando $Clk = 0$, el latch permanece en el estado en que está en el instante *t*, es decir, $Q(t)$, independientemente de los valores de *S* y *R*. Esto se indica al especificar $S = x$ y $R = x$, donde x significa que el valor de la señal puede ser 0 o 1. (Recuérdese que ya empleamos esta notación en el capítulo 4.) Cuando $Clk = 1$, el circuito se comporta como el latch básico de la figura 7.5. Se establece en 1 mediante $S = 1$ y se inicializa por medio de $R = 1$. En la última fila de la tabla, donde $S = R = 1$, se muestra que el estado $Q(t + 1)$ no está definido porque no sabemos si será 0 o 1, lo cual corresponde a la situación descrita en la sección 7.1 junto con el diagrama de tiempo de la figura 7.5 en el tiempo t_{10}. En este tiempo las entradas *S* y *R* pasan de 1 a 0, lo que ocasiona el comportamiento oscilatorio que ya explicamos. Si $S = R = 1$, esta situación ocurrirá en cuanto *Clk* pase de 1 a 0. Para asegurar una operación significativa del latch SR asíncrono es esencial evitar la posibilidad de tener las dos entradas *S* y *R* iguales a 1 cuando *Clk* cambia de 1 a 0.

Un diagrama de tiempo para el latch SR asíncrono se muestra en la figura 7.6*c*. *Clk* se presenta como una señal periódica igual a 1 en intervalos de tiempo regulares para indicar que tal es casi siempre la forma en que aparece la señal de reloj en un sistema real. En el diagrama se advierte el efecto de varias combinaciones de valores de señal. Nótese que hemos etiquetado una salida como Q y la otra como su complemento $\overline{Q}$, en vez de Q_a y Q_b como en la figura 7.5. Puesto que el modo indefinido, donde $S = R = 1$, debe evitarse en la práctica, la operación normal del latch tendrá las salidas como complementos unas de las otras. Además, con frecuencia decimos que el latch se establece en 1 cuando $Q = 1$, y se *inicializa* cuando $Q = 0$. En la figura 7.6*d* se muestra el símbolo gráfico para el latch SR asíncrono.

7.2.1 LATCH SR ASÍNCRONO CON COMPUERTAS NAND

Hasta ahora hemos implementado el latch básico con compuertas NOR con acoplamiento cruzado. También podemos construir el latch con compuertas NAND. Si utilizamos este método es posible implementar el latch SR asíncrono como se muestra en la figura 7.7. El comportamiento de este circuito se describe en la tabla de la figura 7.6*b*. Obsérvese que en este circuito el reloj

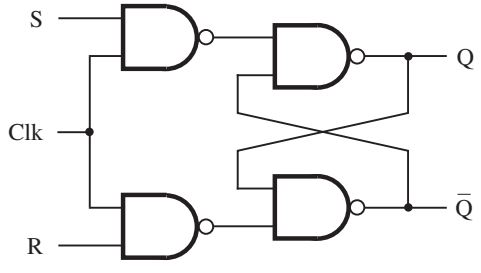


Figura 7.7 Latch SR asíncrono con compuertas NAND.

está controlado por compuertas NAND en vez de AND. Adviértase también que las entradas *S* y *R* están invertidas en comparación con el circuito de la figura 7.6*a*. El circuito con compuertas NAND requiere menos transistores que el circuito con compuertas AND, por lo cual optaremos por usar el circuito de la figura 7.7 en lugar del de la figura 7.6*a*.

7.3 LATCH D ASÍNCRONO

En la sección 7.2 presentamos el latch SR asíncrono y mostramos de qué manera usarlo como el elemento de memoria en el sistema de alarma de la figura 7.1. Este latch es útil para muchas otras aplicaciones. En esta sección describimos otro latch asíncrono aún más útil en la práctica. Tiene una sola entrada de datos, llamada *D*, y almacena el valor de esta entrada bajo el control de una señal de reloj. Se llama *latch D asíncrono*.

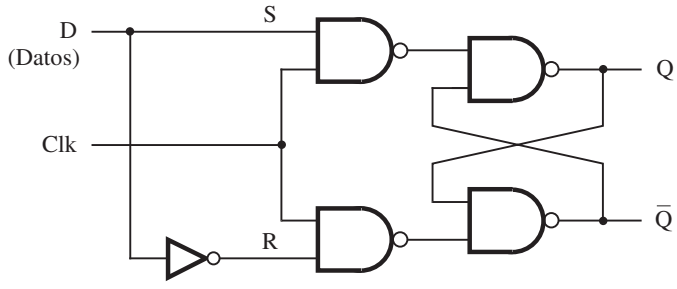
Para generar la necesidad de un latch D asíncrono, considérese la unidad de sumador/restador expuesta en el capítulo 5 (figura 5.13). Cuando describimos cómo se usa ese circuito para sumar números, no explicamos lo que es probable que ocurra con los bits de suma producidos por el sumador. Las unidades de sumador/restador se emplean como parte de una computadora. El resultado de una operación de suma o resta con frecuencia se utiliza como un operando en una operación posterior. Por tanto, es preciso recordar los valores de los bits de suma generados por el sumador en caso que se requieran de nuevo. Podríamos pensar en usar los latches básicos para recordar esos bits, un latch por bit. En este contexto, en vez de decir que un latch recuerda el valor de un bit, resulta más claro decir que el latch *almacena* el valor del bit, o simplemente "almacena el bit". Debemos pensar en el latch como un elemento de almacenamiento.

Pero ¿podemos obtener la operación buscada con latches básicos? Sin duda, es posible inicializar todos los latches antes que la operación de adición comience. Luego esperaríamos que al conectar un bit de suma a la entrada *S*, el latch se establezca en 1 si el bit de suma tiene el valor de 1; de lo contrario, el latch permanecería en el estado 0. Esto funcionaría bien si todos los bits de suma fueran 0 al comienzo de la operación de adición y, después de cierto retraso de propagación a través del sumador, algunos de esos bits se vuelven iguales a 1 para proporcionar la suma buscada. Lamentablemente, los retrasos de propagación que hay en el circuito del sumador causan un problema mayor en este arreglo. Supóngase que usamos un sumador con acarreo en cascada. Cuando las entradas *X* y *Y* se aplican al sumador, las salidas de la suma pueden alternar entre 0 y 1 varias veces a medida que los acarreos pasan por el circuito. Esta situación se ilustró en el diagrama de tiempo de la figura 5.21. El problema es que cuando conectamos un bit de suma a la entrada *S* de un latch, entonces si temporalmente el bit de suma es 1 y luego se establece en 0 en el resultado final, el latch permanecerá en 1 por error.

El problema ocasionado por los valores alternos de los bits de suma en el sumador podría resolverse si empleamos latches SR asíncronos en vez de latches básicos. Luego podríamos arreglar que la señal de reloj sea 0 durante el tiempo que requiera el sumador para producir una suma correcta. Después de permitir el retraso de propagación máximo en el circuito sumador, el reloj pasaría a 1 para almacenar los valores de los bits de suma en los latches asíncronos. En cuanto los valores se hayan almacenado, el reloj puede regresar a 0, lo cual asegura que los valores almacenados se conservarán hasta la siguiente vez que el reloj pase por 1. Para lograr la operación buscada también podríamos hacer que todos los latches se inicialicen en 0 antes de cargar los valores del bit de suma en ellos. Ésta es una forma poco práctica de encarar el problema y es preferible usar latches D asíncronos.

En la figura 7.8*a* se muestra el circuito para un latch D asíncrono. Se basa en el latch SR asíncrono, pero en vez de utilizar entradas S y R por separado, sólo tiene una entrada de datos, D. Por conveniencia hemos etiquetado los puntos del circuito equivalentes a las entradas S y R. Si $D = 1$, entonces $S = 1$ y $R = 0$, lo que obliga al latch a entrar en el estado Q = 1. Si $D = 0$, entonces $S = 0$ y $R = 1$, lo cual hace que Q = 0. Desde luego, los cambios de estado sólo ocurren cuando $Clk = 1$.

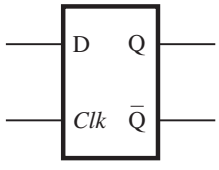
Es importante observar que en este circuito resulta imposible que se presente el problema de $S = R = 1$. En el latch D asíncrono, la salida Q simplemente sigue al valor de la entrada D cuando $Clk = 1$. En cuanto Clk pasa a 0, el estado del latch se congela hasta la siguiente vez que la señal de reloj pasa a 1.Por consiguiente, el latch D asíncrono almacena el valor de la entrada D



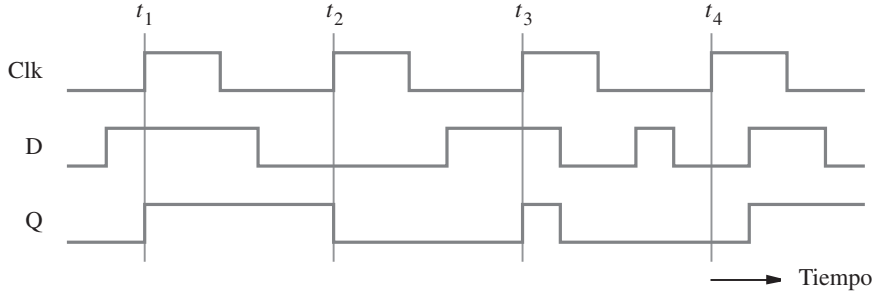
a) Circuito

Clk	D	Q($t + 1$)
0	x	Q(t)
1	0	0
1	1	1

b) Tabla característica



c) Símbolo gráfico



d) Diagrama de tiempo

Figura 7.8 Latch D asíncrono.

visto en el instante que el reloj cambia de 1 a 0. En la figura 7.8 además se proporciona la tabla característica, el símbolo gráfico y el diagrama de tiempo para el latch D asíncrono.

El diagrama de tiempo ilustra lo que ocurre si la señal *D* cambia mientras *Clk* = 1. Durante el tercer pulso del reloj, comenzando en t_3, la salida Q cambia a 1 porque *D* = 1. Pero a mitad del camino que recorre el pulso *D* pasa a 0, lo cual ocasiona que Q se establezca en 0. Este valor de Q se almacena cuando *Clk* cambia a 0. Ahora ya no ocurre ningún cambio en el estado del latch hasta el siguiente pulso del reloj en t_4. Lo más importante que debe advertirse es que mientras el reloj tiene el valor 1, la salida Q sigue a la entrada *D*. Pero cuando el reloj tiene el valor 0, la salida Q no puede cambiar. En el capítulo 3 vimos que los valores lógicos se implementan como niveles de voltaje altos y bajos. Puesto que la salida del latch D asíncrono está controlada por el nivel de la entrada del reloj, se dice que el latch es *sensible al nivel*. Los circuitos de las figuras 7.6 a 7.8 son sensibles al nivel. En la sección 7.4 mostraremos que es posible diseñar elementos de almacenamiento para los que la salida sólo cambia en el instante en que el reloj cambia de un valor a otro. Se dice que tales circuitos se *disparan por flanco*.

En este punto debemos considerar de nuevo el circuito de la figura 7.3. Al examinarlo detalladamente veremos que se comporta igual que el circuito de la figura 7.8*a*. Las entradas *Data* y *Load* corresponden a las entradas *D* y *Clk*, respectivamente. *Output*, que tiene el mismo valor de señal que el punto *A*, corresponde a la salida Q. El punto *B* corresponde a $\overline{Q}$. Por tanto, el circuito de la figura 7.3 también es un latch D asíncrono. Una ventaja de este circuito es que puede implementarse con menos transistores que el circuito de la figura 7.8*a*.

7.3.1 EFECTOS DE LOS RETRASOS DE PROPAGACIÓN

En el análisis anterior ignoramos los efectos de los retrasos de propagación. En los circuitos prácticos es esencial tomarlos en cuenta. Considérese el latch D asíncrono de la figura 7.8*a*. Almacena el valor de la entrada *D* que se presenta en el momento en que la señal de reloj cambia de 1 a 0. Opera en forma apropiada si la señal *D* es estable (es decir, si no cambia) en el instante en que *Clk* pasa de 1 a 0. Pero puede conducir a resultados impredecibles si la señal *D* también cambia en ese tiempo. Por ende, el diseñador de un circuito lógico que genera la señal *D* debe cerciorarse que ésta sea estable cuando ocurra el cambio crítico en la señal de reloj.

En la figura 7.9 se ilustra la región de temporización crítica. El tiempo mínimo que la señal *D* debe permanecer estable antes del flanco negativo de la señal *Clk* se llama *tiempo de preparación*, t_{su}, del latch. El tiempo mínimo que la señal *D* debe permanecer estable después del flanco

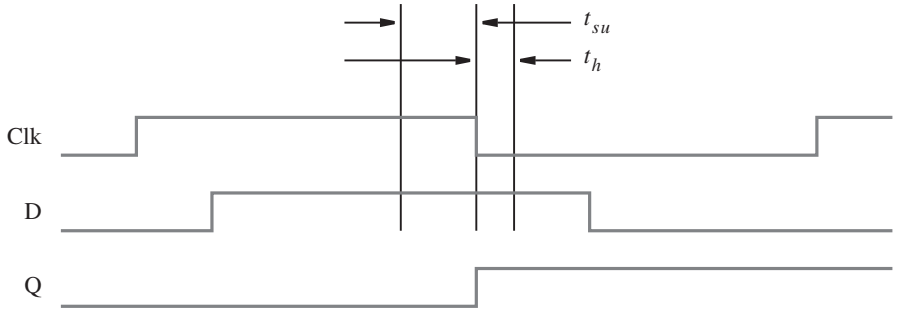


Figura 7.9 Tiempos de preparación y de espera.

negativo de la señal *Clk* recibe el nombre de *tiempo de espera*, t_h, del latch. Los valores de t_{su} y t_h dependen de la tecnología empleada. Los fabricantes de los chips de circuitos integrados proporcionan esta información en las hojas de datos que describen sus chips. Los valores típicos para la tecnología CMOS son $t_{su} = 3$ ns y $t_h = 2$ ns. En la sección 7.13 daremos ejemplos de la manera en que los tiempos de preparación y de espera afectan la velocidad de operación de los circuitos. En la sección 10.3.3 se explica el comportamiento de los elementos de almacenamiento cuando se incumplen los tiempos de preparación o de espera.

7.4 FLIP-FLOPS D MAESTRO-ESCLAVO Y DISPARADO POR FLANCO

En los latches sensibles al nivel, el estado del latch sigue cambiando de acuerdo con los valores de las señales de entrada mientras la señal de reloj está activa (igual a 1 en nuestros ejemplos). Como se verá en las secciones 7.8 y 7.9, también existe la necesidad de elementos de almacenamiento que puedan cambiar sus estados no más de una vez durante un ciclo del reloj. Estudiaremos dos tipos de circuitos que presentan este comportamiento.

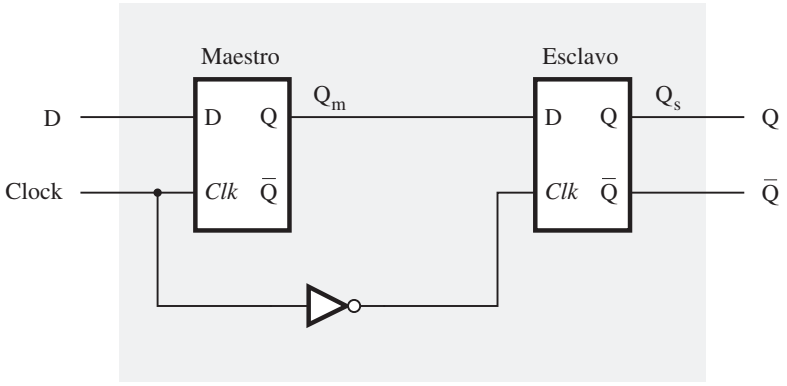
7.4.1 FLIP-FLOP D MAESTRO-ESCLAVO

Considérese el circuito de la figura 7.10*a*, el cual se compone de dos latches D asíncronos. El primero, llamado *maestro*, cambia su estado mientras *Clock* = 1. El segundo, denominado *esclavo*, lo hace mientras *Clock* = 0. El funcionamiento del circuito es tal que cuando el reloj está en nivel alto, el latch maestro sigue el valor de la señal de entrada *D* y el esclavo no cambia. Por tanto, el valor de Q_m sigue cualquier cambio en *D* y el de Q_s permanece constante. Cuando la señal de reloj cambia a 0, la etapa de maestro deja de seguir los cambios en la entrada *D*. Al mismo tiempo, la etapa de esclavo responde al valor de la señal Q_m y por consiguiente cambia de estado. Como Q_m no cambia mientras *Clock* = 0, la etapa de esclavo puede sufrir cuando mucho un cambio de estado durante un ciclo del reloj. Desde el punto de vista del observador externo, es decir, del circuito conectado a la salida de la etapa de esclavo, el circuito maestro-esclavo cambia su estado en el flanco del reloj que va a ser negativo. El *flanco negativo* es el flanco donde la señal de reloj cambia de 1 a 0. Independientemente del número de cambios en la entrada *D* a la etapa de maestro durante un ciclo del reloj, el observador de la señal Q_s verá sólo el cambio que corresponde a la entrada *D* en el flanco negativo del reloj.

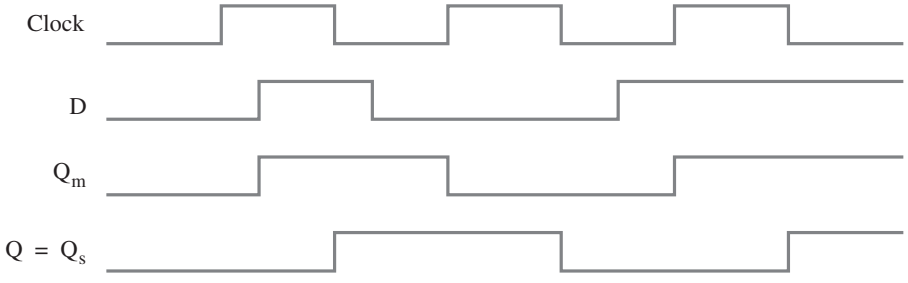
El circuito de la figura 7.10 se llama *flip-flop D maestro-esclavo*. El término *flip-flop* indica un elemento de almacenamiento que cambia el estado de su salida en el flanco de una señal controladora de reloj. El diagrama de tiempo de este flip-flop se muestra en la figura 7.10*b*. En la figura 7.10*c* aparece el símbolo gráfico correspondiente; en él usamos el signo $>$ para indicar que el flip-flop responde al "flanco activo" del reloj. Colocamos una burbuja en la entrada del reloj para indicar que el flanco activo de este circuito en particular es el flanco negativo.

7.4.2 FLIP-FLOP D DISPARADO POR FLANCO

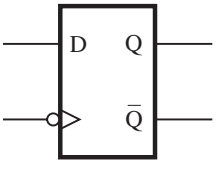
La salida del flip-flop D maestro-esclavo de la figura 7.10*a* responde al flanco negativo de la señal de reloj. El circuito puede modificarse para que responda al flanco positivo del reloj al conectar la etapa de esclavo directamente al reloj y la etapa de maestro al complemento de éste.



a) Circuito



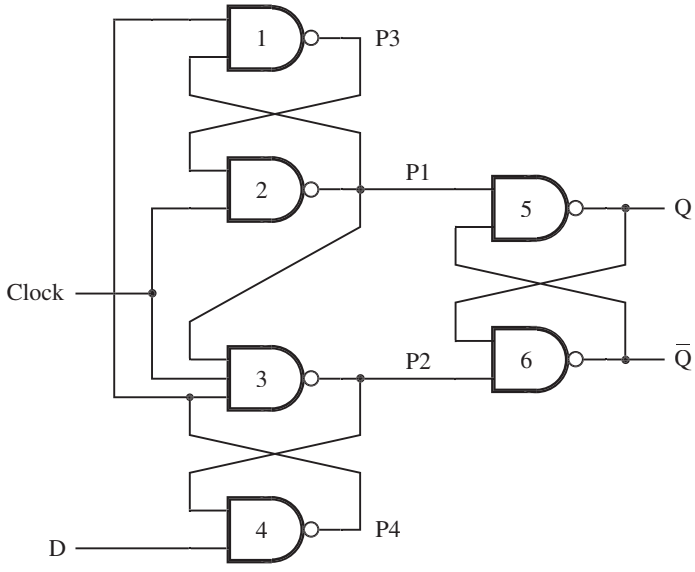
b) Diagrama de tiempo



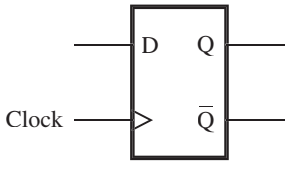
c) Símbolo gráfico

Figura 7.10 Flip-flop D maestro-esclavo.

En la figura 7.11*a* se presenta un circuito diferente que realiza la misma tarea. Requiere sólo seis compuertas NAND y, por consiguiente, menos transistores. El funcionamiento del circuito es como sigue. Cuando *Clock* = 0, las salidas de las compuertas 2 y 3 están en nivel alto. Por tanto, $P1 = P2 = 1$, lo que mantiene el latch de salida, compuesto por las compuertas 5 y 6, en su estado actual. Al mismo tiempo, la señal $P3$ es igual a D, y $P4$ es igual a su complemento $\overline{D}$.



a) Circuito



b) Símbolo gráfico

Figura 7.11 Un flip-flop D disparado por el flanco positivo.

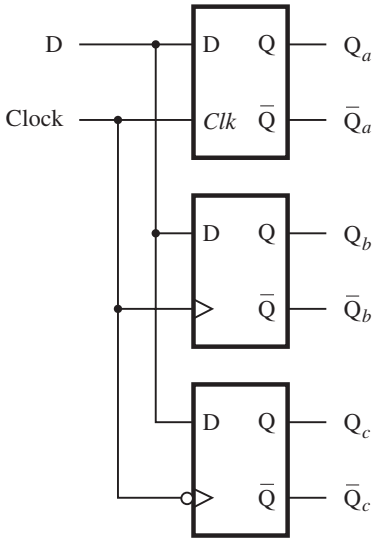
Cuando *Clock* cambia a 1, ocurren los cambios siguientes. Los valores de *P*3 y *P*4 se transmiten a través de las compuertas 2 y 3 para hacer que $P1 = \overline{D}$ y $P2 = D$, lo que establece $Q = D$ y $\overline{Q} = \overline{D}$. Para que funcionen de manera confiable, *P*3 y *P*4 deben hallarse estables cuando el reloj cambie de 0 a 1. Por tanto, el tiempo de preparación del flip-flop es igual al retraso de la entrada *D* a través de las compuertas 4 y 1 hacia *P*3. El tiempo de espera está dado por el retraso de la compuerta 3 porque una vez que *P*2 está estable, los cambios en *D* ya no importan.

Para que haya un funcionamiento adecuado es preciso mostrar que, después que *Clock* cambie a 1, cualquier cambio posterior en *D* no afectará el latch de salida siempre que *Clock* = 1. Hay que considerar dos casos. Supóngase primero que *D* = 0 en el flanco positivo del reloj. Entonces *P*2 = 0, lo cual mantendrá la salida de la compuerta 4 igual a 1 siempre que *Clock* = 1, independientemente del valor de la entrada *D*. El segundo caso es cuando *D* = 1 en el flanco positivo del reloj. Entonces *P*1 = 0, lo que obliga a que las salidas de las compuertas 1 y 3 sean iguales a 1, independientemente de la entrada *D*. Por ende, el flip-flop ignora los cambios en la entrada *D* mientras *Clock* = 1.

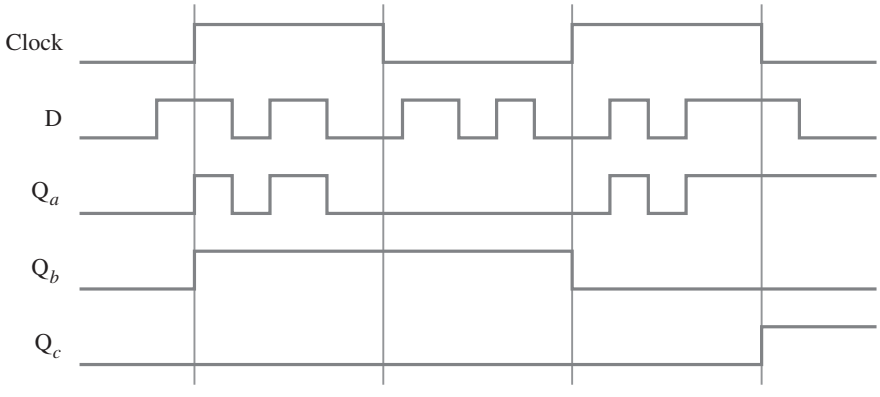
En la figura 7.11*b* se presenta un símbolo gráfico para este flip-flop. La entrada del reloj indica que el flanco positivo del reloj es el activo. Un circuito similar, construido con compuertas NOR, puede usarse como un flip-flop disparado por el flanco negativo.

Elementos de almacenamiento sensibles al nivel *versus* elementos disparados por flanco

En la figura 7.12 se muestran tres tipos de elementos de almacenamiento manejados por los mismos datos y entradas del reloj. El primer elemento es un latch D asíncrono, sensible al nivel. El segundo es un flip-flop D disparado por el flanco positivo y el tercero es un flip-flop D disparado por el flanco negativo. Para acentuar las diferencias entre estos elementos de almacenamiento,



a) Circuito



b) Diagrama de tiempo

Figura 7.12 Comparación de elementos de almacenamiento D sensibles al nivel con elementos de almacenamiento D disparados por flanco.

la entrada *D* cambia sus valores más de una vez durante cada semiciclo del reloj. Obsérvese que el latch D asíncrono sigue a la entrada *D* mientras el reloj esté en nivel alto. El flip-flop disparado por el flanco positivo responde sólo al valor de *D* cuando el reloj cambia de 0 a 1. El flip-flop disparado por el flanco negativo responde sólo al valor de *D* cuando el reloj cambia de 1 a 0.

7.4.3 FLIP-FLOPS D CON CLEAR Y PRESET

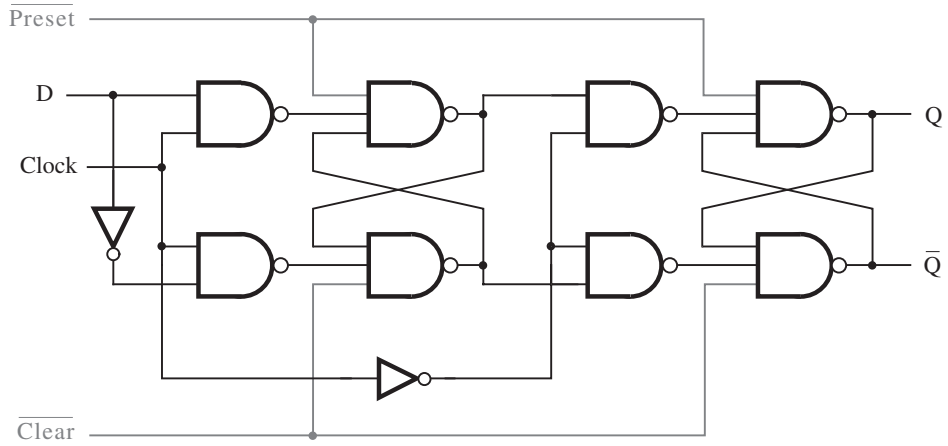
Los flip-flops suelen utilizarse para implementar circuitos que pueden tener muchos estados posibles, donde la respuesta del circuito depende no sólo de los valores que hay en las entradas sino también de los valores del estado particular en que se halla el circuito en ese instante. En el capítulo siguiente analizaremos una forma general de tales circuitos. Un ejemplo simple es un circuito contador que cuenta el número de ocurrencias de algún evento, tal vez el paso del tiempo. Estudiaremos los contadores con detalle en la sección 7.9. Un contador comprende una serie de flip-flops, cuyas salidas se interpretan como un número. El circuito contador debe tener la capacidad de aumentar o disminuir este número. También es importante poder forzarlo a entrar en un estado inicial conocido (*count*). Lógicamente, debe ser posible borrar el contador y dejarlo en cero, lo que significa que todos los flip-flops deben tener Q = 0. También es útil poder preestablecer cada flip-flop en Q = 1 para insertar una cuenta específica como el valor inicial del contador. Estas características pueden incorporarse a los circuitos de las figuras 7.10 y 7.11 como sigue.

En la figura 7.13*a* se muestra una implementación del circuito de la figura 7.10a usando compuertas NAND. La etapa de maestro es simplemente el latch D asíncrono de la figura 7.8*a*. En vez de usar otro latch del mismo tipo para la etapa de esclavo, podemos usar el latch SR asíncrono ligeramente más simple de la figura 7.7. Con ello se elimina una compuerta NOT del circuito.

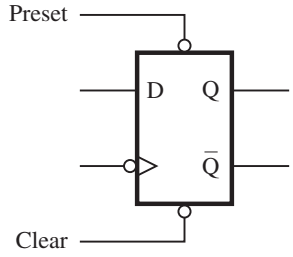
Una forma sencilla de proporcionar la capacidad de borrar (*clear*) y preestablecer (*preset*) consiste en añadir una entrada adicional a cada compuerta NAND en los latches con acoplamiento cruzado, como se indica en gris oscuro. Colocar un 0 en la entrada *Clear* hará que el flip-flop entre en el estado Q = 0. Si *Clear* = 1, entonces esta entrada no producirá efecto alguno en las compuertas NAND. De forma semejante, *Preset* = 0 obliga a que el flip-flop entre en el estado Q = 1, mientras que *Preset* = 1 no produce efecto. Para indicar que las entradas *Clear* y *Preset* están activas cuando sus valores son 0 ponemos una línea arriba de los nombres en la figura. Cabe señalar que el circuito que utiliza este flip-flop no debe intentar forzar que tanto *Clear* como *Preset* se establezcan en 0 al mismo tiempo. Un símbolo gráfico para este flip-flop se muestra en la figura 7.13*b*.

Una modificación similar puede hacerse en el flip-flop disparado por flanco de la figura 7.11*a*, como se indica en la figura 7.14*a*. De nuevo, tanto las entradas *Clear* como *Preset* están activas en nivel bajo. No alteran el flip-flop cuando son iguales a 1.

En los circuitos de las figuras 7.13*a* y 7.14*a*, el efecto de una señal baja en la entrada *Clear* o en *Preset* es inmediato. Por ejemplo, si *Clear* = 0 entonces el flip-flop entra en el estado Q = 0 inmediatamente, con independencia del valor de la señal de reloj. En un circuito como éste, donde la señal *Clear* sirve para borrar un flip-flop sin importar la señal de reloj, decimos que el flip-flop tiene un *borrado asíncrono*. En la práctica, a menudo es preferible borrar los flip-flops en el flanco activo del reloj. Este *borrado síncrono* puede lograrse como se muestra en la figura 7.15. El flip-flop opera con normalidad cuando la entrada *Clear* es igual a 1; pero si pasa a 0, entonces en el siguiente flanco negativo del reloj el flip-flop se establecerá en 0. Examinaremos el borrado de los flips-flops con más detalle en la sección 7.10.



a) Circuito

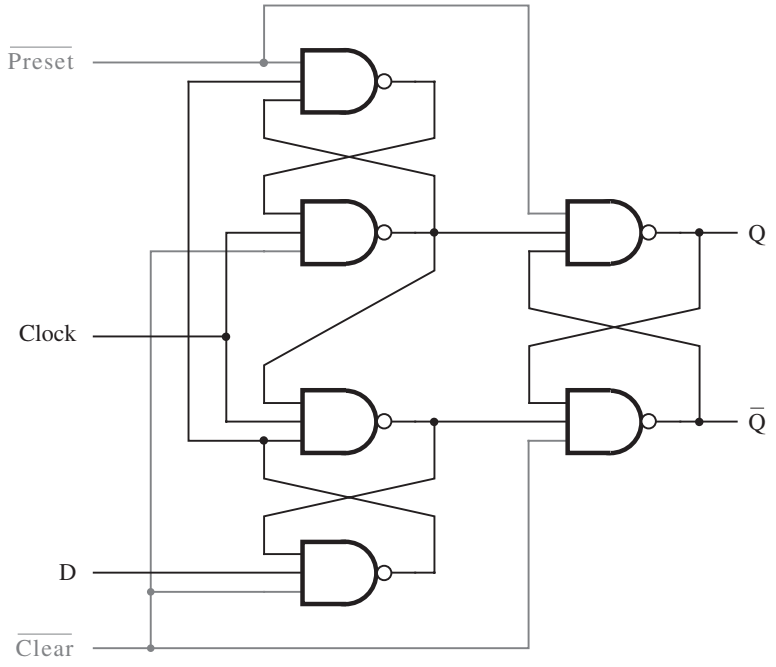


b) Símbolo gráfico

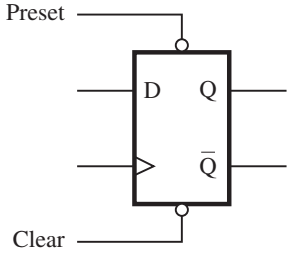
Figura 7.13 Flip-flop D maestro-esclavo con Clear y Preset.

7.5 FLIP-FLOP T

El flip-flop D es un elemento de almacenamiento polifacético que sirve para muchos propósitos. Al incluir un sistema de circuitos lógico simple para manejar su entrada, el flip-flop D puede parecer un elemento de almacenamiento distinto. Una modificación interesante se presenta en la figura 7.16*a*. Este circuito usa un flip-flop D disparado por el flanco positivo. Las conexiones de *retroalimentación* hacen que la señal de entrada D sea igual a cualquier valor de Q o $\overline{Q}$ bajo el control de la señal etiquetada con T. En cada flanco positivo del reloj, el flip-flop puede cambiar su estado Q(t). Si $T = 0$, entonces $D =$ Q y el estado seguirá siendo el mismo, es decir, Q(t + 1) = Q(t). Pero si $T = 1$, entonces $D = \overline{Q}$ y el nuevo estado será Q(t + 1) = $\overline{Q}$(t). Por tanto, la operación general del circuito es que éste conserva su estado presente si $T = 0$ y lo invierte si $T = 1$.



a) Circuito



b) Símbolo gráfico

Figura 7.14 Flip-flop D disparado por el flanco positivo con Clear y Preset.

El funcionamiento del circuito se especifica como una tabla característica en la figura 7.16*b*. Cualquier circuito que implemente esta tabla se llama *flip-flop T*. El nombre flip-flop T proviene del comportamiento del circuito, el cual "alterna entre dos estados" (*toggle*) cuando $T = 1$. La función toggle hace que el flip-flop T sea útil para construir circuitos contadores, como veremos en la sección 7.9.

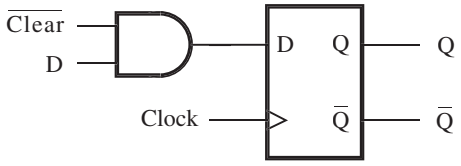
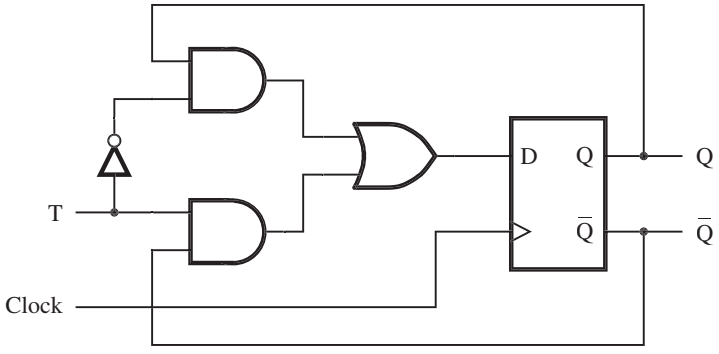


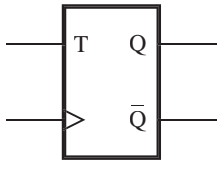
Figura 7.15 Inicialización síncrona para un flip-flop D.



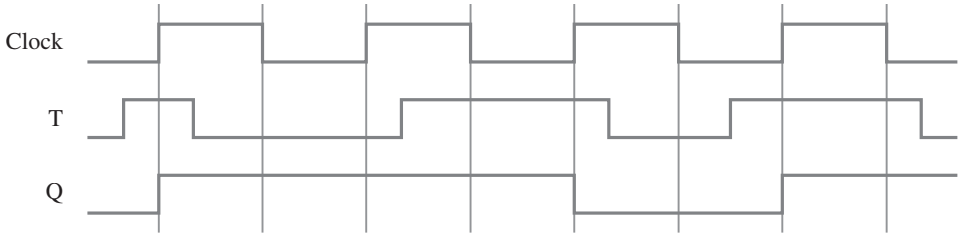
a) Circuito

T	Q(*t* + 1)
0	Q(*t*)
1	$\overline{Q}(t)$

b) Tabla característica



c) Símbolo gráfico



d) Diagrama de tiempo

Figura 7.16 Flip-flop T.

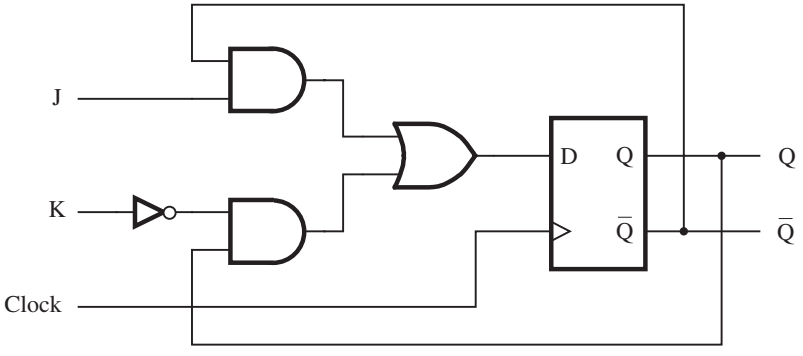
7.5.1 FLIP-FLOPS CONFIGURABLES

Para algunos circuitos un tipo de flip-flop puede llevar a una implementación más eficaz que otro tipo. En los chips de uso general como los PLD, los flip-flops que se proporcionan a veces son *configurables*, lo cual significa que un circuito de flip-flop puede configurarse para ser D, T o de otro tipo. Por ejemplo, en algunos PLD los flip-flops pueden configurarse ya sea como tipo D o T (véanse los problemas 7.6 y 7.8).

7.6 FLIP-FLOP JK

Otro circuito interesante puede derivarse de la figura 7.16*a*. En vez de usar una sola entrada de control, *T*, podemos usar dos entradas, *J* y *K*, como se indica en la figura 7.17*a*. Para este circuito la entrada *D* se define como

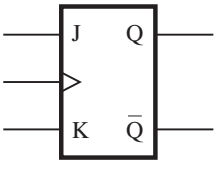
$$D = J\overline{Q} + \overline{K}Q$$



a) Circuito

J	K	Q(*t* + 1)
0	0	Q(*t*)
0	1	0
1	0	1
1	1	$\overline{Q}(t)$

b) Tabla característica



c) Símbolo gráfico

Figura 7.17 Flip-flop JK.

En la figura 7.17*b* se presenta la tabla característica correspondiente. El circuito se llama *flip-flop JK*. Combina el comportamiento de los flip-flops SR y T de una manera útil. Se comporta como el flip-flop SR, donde $J = S$ y $K = R$, para todos los valores de entrada excepto $J = K = 1$. En este último caso, el cual debe evitarse en el flip-flop SR, el flip-flop JK alterna su estado del mismo modo que el flip-flop T.

El flip-flop JK es un circuito multifuncional. Puede usarse para almacenamiento sencillo, como los flip-flops D y SR. Pero también sirve como un flip-flop T si se conectan juntas las entradas *J* y *K*.

7.7 RESUMEN DE TERMINOLOGÍA

Hemos usado la terminología más común. Pero el lector debe estar consciente de que puede haber diferentes interpretaciones de los términos *latch* y *flip-flop* en otras obras. Nuestra terminología puede resumirse como sigue:

El **latch básico** es una conexión de retroalimentación de dos compuertas NOR o NAND, las cuales pueden almacenar un bit de información. Este latch puede establecerse en 1 utilizando la entrada *S* e inicializarse en 0 con la entrada *R*.

El **latch asíncrono** es un latch básico que incluye compuertas de entrada y una señal de entrada de control. El latch conserva su estado actual cuando la entrada de control es igual a 0. Su estado puede cambiar cuando la señal de control es igual a 1. En nuestra exposición nos referimos a la entrada de control como el reloj. Consideramos dos tipos de latches asíncronos:

- El **latch SR asíncrono** usa las entradas *S* y *R* para establecer el latch en 1 o inicializarlo en 0, respectivamente.
- El **latch D asíncrono** usa la entrada *D* para obligar al latch a entrar en un estado que tiene el mismo valor lógico que la entrada *D*.

Un flip-flop es un elemento de almacenamiento basado en el principio del latch asíncrono, el cual únicamente puede cambiar su estado de salida en el flanco de la señal de reloj controladora. Consideramos dos tipos de flip-flop:

- El **flip-flop disparado por flanco**, que se ve afectado sólo por los valores de entrada presentes cuando el reloj está en el flanco activo.
- El **flip-flop maestro-esclavo** se construye con dos latches asíncronos. La etapa de maestro está activa durante la mitad del ciclo del reloj, y la etapa de esclavo durante la otra mitad. El valor de salida del flip-flop cambia en el flanco del reloj que activa la transferencia en la etapa de esclavo. El flip-flop maestro-esclavo es disparado por flanco o sensible al nivel. Si la etapa de maestro es un latch D asíncrono, entonces se comporta como flip-flop disparado por flanco. Si la etapa de maestro es un latch SR asíncrono, entonces el flip-flop es sensible al nivel (véase el problema 7.19).

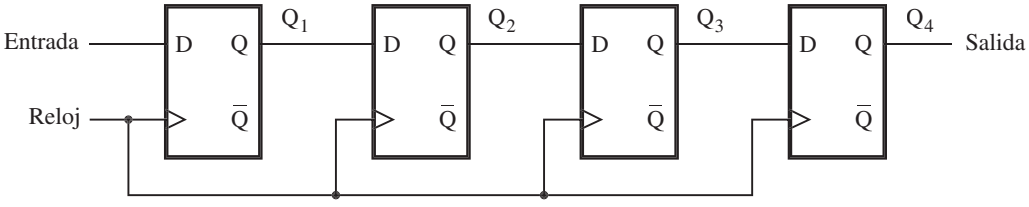
7.8 REGISTROS

Un flip-flop almacena un bit de información. Cuando un conjunto de *n* flip-flops se usa para almacenar *n* bits de información, como un número de *n* bits, nos referimos a él como un *registro*. Para cada flip-flop de un registro se usa un reloj común, y cada flip-flop funciona como se describió en la sección anterior. El término *registro* es simplemente una convención para referirse a las estructuras de *n* bits que se componen de flip-flops.

7.8.1 REGISTRO DE CORRIMIENTO

En la sección 5.6 explicamos que un número se multiplica por 2 si sus bits se desplazan una posición de un bit a la izquierda y se inserta un 0 como el nuevo bit menos significativo. De modo similar, el número se divide entre 2 si los bits se desplazan una posición de un bit a la derecha. Un registro que proporciona la capacidad para recorrer su contenido se llama *registro de corrimiento*.

En la figura 7.18*a* se muestra un registro de corrimiento de cuatro bits que se utiliza para desplazar su contenido una posición de un bit a la derecha. Los bits de datos se cargan en el registro de corrimiento en forma serial usando la entrada *In*. El contenido de cada flip-flop se transfiere al flip-flop siguiente



a) Circuito

	Entrada	Q_1	Q_2	Q_3	Q_4 = Salida
t_0	1	0	0	0	0
t_1	0	1	0	0	0
t_2	1	0	1	0	0
t_3	1	1	0	1	0
t_4	1	1	1	0	1
t_5	0	1	1	1	0
t_6	0	0	1	1	1
t_7	0	0	0	1	1

b) Una secuencia de muestra

Figura 7.18 Un registro de corrimiento simple.

en cada flanco positivo del reloj. En la figura 7.18*b* se presenta una ilustración de la transferencia; también se muestra lo que ocurre cuando los valores de la señal en *In* durante ocho ciclos del reloj consecutivos son 1, 0, 1, 1, 1, 0, 0 y 0, suponiendo que el estado inicial de todos los flip-flops es 0.

Para implementar un registro de corrimiento es necesario usar flip-flops disparados por flanco o maestro-esclavo. Los latches asíncronos sensibles al nivel no son adecuados, ya que un cambio en el valor de *In* se propagaría a través de más de un latch durante el tiempo que el reloj sea igual a 1.

7.8.2 REGISTRO DE CORRIMIENTO CON ACCESO EN PARALELO

En los sistemas de cómputo con frecuencia se requiere transferir elementos de datos de *n* bits. Esta transferencia puede realizarse transmitiendo todos los bits de una sola vez con *n* cables independientes, caso en el que decimos que la transferencia se realiza en *paralelo*. Pero también es posible transferir todos los bits por un solo cable, realizando la transferencia un bit a la vez, en *n* ciclos del reloj consecutivos. Nos referimos a este esquema como transferencia *serial*. Para transferir un elemento de datos de *n* bits en forma serial es posible usar un registro de corrimiento que puede cargarse con todos los *n* bits en paralelo (en un ciclo del reloj). Luego, durante los ciclos del reloj siguientes el contenido del registro puede desplazarse hacia afuera para la transferencia serial. También se requiere la operación inversa. Si los bits se reciben en forma serial, entonces después de *n* ciclos del reloj puede accederse al contenido del registro en paralelo como un elemento de *n* bits.

En la figura 7.19 se presenta un registro de corrimiento de cuatro bits que permite el acceso en paralelo. En vez de usar la conexión de registro de corrimiento normal, la entrada *D* de cada flip-flop se conecta a dos fuentes; una de ellas es el flip-flop precedente, el cual se necesita para que el registro de corrimiento opere. La otra fuente es la entrada externa que corresponde al bit que se va a cargar en el flip-flop como parte de la operación de carga en paralelo. La señal de control $\overline{Shift}/Load$ se usa para seleccionar el modo de operación. Si $\overline{Shift}/Load = 0$, entonces el circuito opera como un registro de corrimiento. Si $\overline{Shift}/Load = 1$, los datos de entrada en paralelo se cargan en el registro. En ambos casos la acción ocurre en el flanco positivo del reloj.

En la figura 7.19 elegimos etiquetar las salidas de los flip-flops como Q_3,..., Q_0 porque los registros de corrimiento con frecuencia se usan para alojar números binarios. Puede accederse en paralelo al contenido del registro al observar las salidas de todos los flip-flops. Los flip-flops también pueden accederse en forma serial, si se observan los valores de Q_0 durante ciclos del reloj consecutivos mientras el contenido se está desplazando. Un circuito en el que los datos pueden cargarse en serie y luego accederse en paralelo se llama convertidor de serial a paralelo. De forma similar, el tipo opuesto de circuito es un convertidor de paralelo a serial. El circuito de la figura 7.19 puede realizar estas dos funciones.

7.9 CONTADORES

En el capítulo 5 abordamos el tema de los circuitos que realizan operaciones aritméticas. Mostramos cómo pueden diseñarse los circuitos sumadores/restadores usando una estructura en cascada (*ripple-carry*) simple, que no es costosa pero sí lenta, o bien utilizando una estructura con acarreo hacia adelante más compleja, que es más cara aunque más rápida. En esta sección examinamos tipos especiales de operaciones de suma y resta usadas con el propósito de contar. En particular, queremos diseñar circuitos que puedan aumentar o disminuir un conteo en 1. Los circuitos

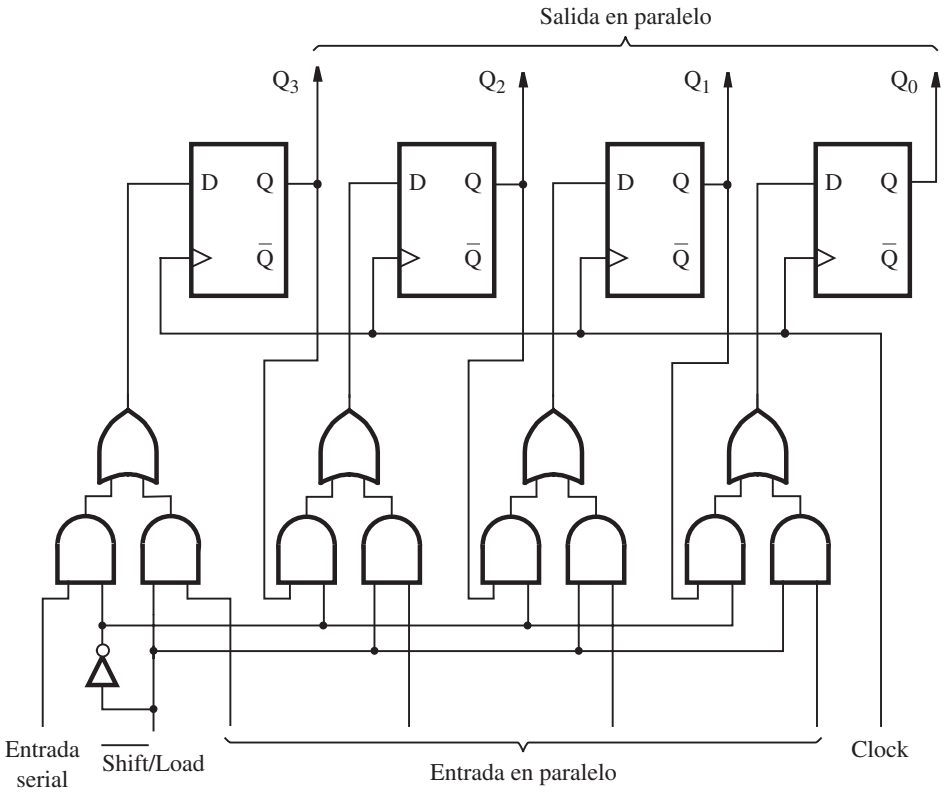


Figura 7.19 Registro de corrimiento con acceso en paralelo.

contadores se utilizan en los sistemas digitales para muchos fines. Pueden contar el número de ocurrencias de ciertos eventos, generar los intervalos de tiempo para el control de varias tareas en un sistema, llevar un registro del tiempo transcurrido entre eventos específicos, etcétera.

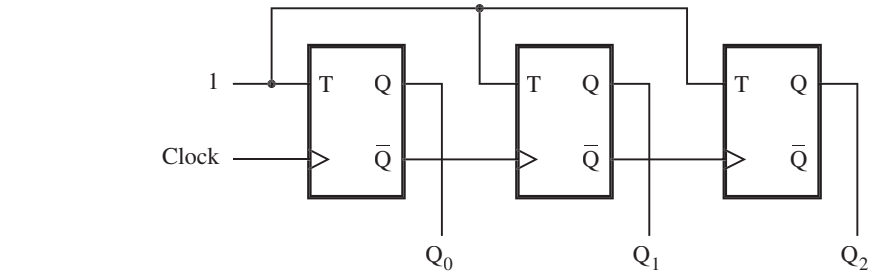
Los contadores pueden implementarse usando los circuitos sumadores/restadores estudiados en el capítulo 5 y los registros expuestos en la sección 7.8. Sin embargo, puesto que sólo necesitamos cambiar el contenido de un contador por 1, no es necesario usar circuitos muy elaborados. En vez de ello, empleamos circuitos mucho más sencillos que tienen un costo considerablemente menor. Mostraremos cómo se diseñan los circuitos contadores usando flip-flops T y D.

7.9.1 CONTADORES ASÍNCRONOS

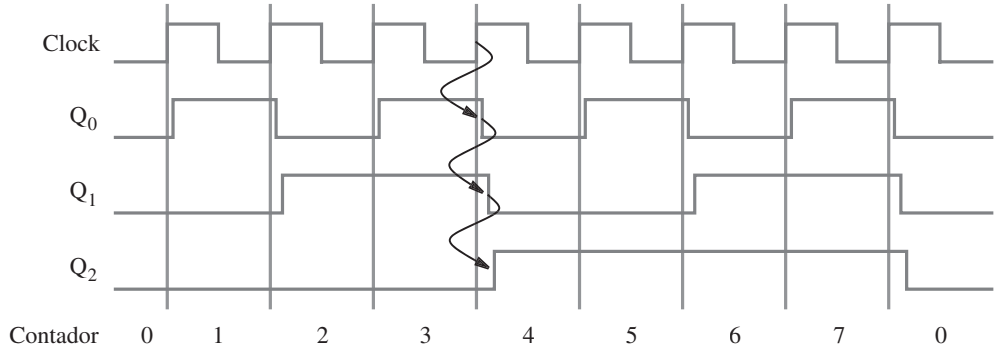
Los circuitos contadores más simples pueden construirse usando flip-flops T debido a que la función *toggle* se adecua de manera natural a la implementación de un conteo.

Contador ascendente con flip-flops T

En la figura 7.20*a* se describe un contador de tres bits que puede contar de 0 a 7. Las entradas del reloj de los tres flip-flops están conectadas en cascada. La entrada *T* de cada flip-flop está conectada a una constante 1, lo que significa que el estado del flip-flop se invertirá (cambiará a



a) Circuito



b) Diagrama de tiempo

Figura 7.20 Un contador ascendente de tres bits.

un segundo estado) en cada flanco positivo de su reloj. Estamos suponiendo que el propósito de este circuito es contar el número de pulsos que ocurren en la entrada principal llamada *Clock*. Por tanto, la entrada del reloj del primer flip-flop está conectada a la línea *Clock*. Las entradas del reloj de los otros dos flip-flops están manejadas por la salida $\overline{Q}$ del flip-flop anterior. Por consiguiente, alternan su estado siempre que el flip-flop precedente cambia su estado de $Q = 1$ a $Q = 0$, lo que resulta en el flanco positivo de la señal $\overline{Q}$.

En la figura 7.20*b* se muestra un diagrama de tiempo para el contador. El valor de Q_0 alterna una vez cada ciclo del reloj. El cambio ocurre poco tiempo después del flanco positivo de la señal *Clock*. El retraso es causado por la propagación a través del flip-flop. Como el segundo flip-flop está sincronizado por $\overline{Q}_0$, el valor de Q_1 cambia poco tiempo después del flanco negativo de la señal Q_0. De modo similar, el valor de Q_2 cambia poco tiempo después del flanco negativo de la señal Q_1. Si observamos los valores $Q_2Q_1Q_0$ como el conteo, entonces el diagrama de tiempo indica que la secuencia de conteo es 0, 1, 2, 3, 4, 5, 6, 7, 0, 1 y así sucesivamente. Este circuito es un contador módulo 8. Como cuenta en dirección ascendente, podemos llamarlo *contador ascendente*.

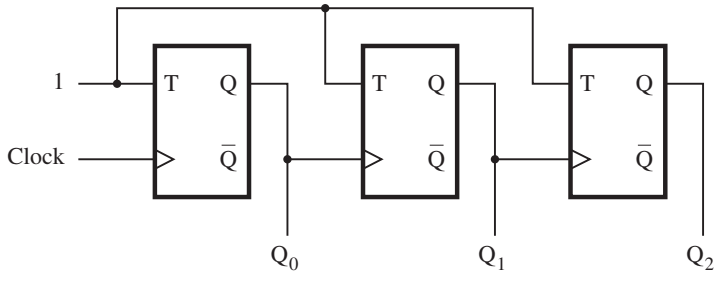
El contador de la figura 7.20*a* tiene tres *etapas*, y cada una consta de un solo flip-flop. Sólo la primera etapa responde directamente a la señal *Clock*; decimos que esta etapa está *sincronizada* con el reloj. Las otras dos etapas responden después de un retraso adicional. Por ejemplo, cuando *Count* = 3, el siguiente pulso del reloj hará que *Count* vaya a 4. Como se indica con las flechas

en el diagrama de tiempo de la figura 7.20*b*, este cambio requiere la alternación de los estados de los tres flip-flops. El cambio en Q_0 se observa sólo después de un retraso de propagación del flanco positivo de *Clock*. Los flip-flops Q_1 y Q_2 aún no han cambiado; por consiguiente, el conteo es $Q_2Q_1Q_0 = 010$ por un breve lapso. El cambio en Q_1 aparece después de un segundo retraso de propagación, momento en el que el conteo es 000. Finalmente, el cambio en Q_2 ocurre después de un tercer retraso, instante en el que el circuito llega a su estado estable y el conteo es 100. Este comportamiento se parece a la cascada de los acarreos en el circuito sumador de acarreo en cascada de la figura 5.6. El circuito de la figura 7.20*a* es un *contador asíncrono*, o un *contador en cascada*.

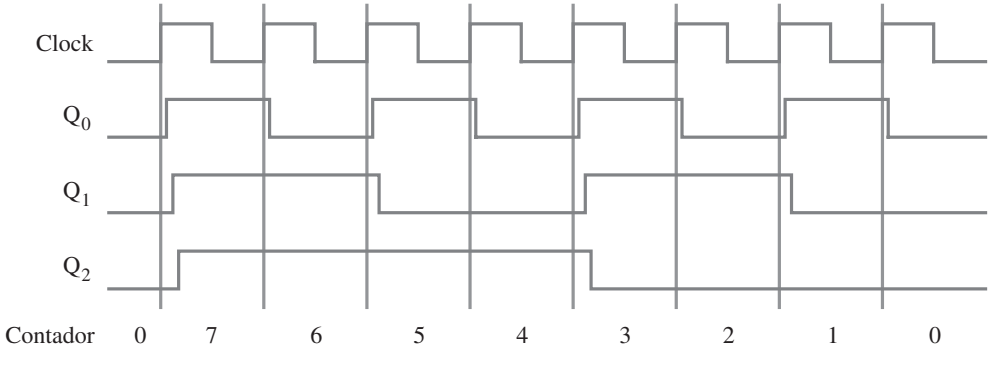
Contador descendente con flip-flops T

Una ligera modificación del circuito de la figura 7.20*a* se presenta en la figura 7.21*a*. La única diferencia es que en esta última figura las entradas del reloj del segundo y tercer flip-flops están manejadas por las salidas Q de las etapas precedentes, en vez de estarlo por las salidas $\overline{Q}$. En el diagrama de tiempo, dado en la figura 7.21*b*, se muestra que este circuito cuenta en la secuencia 0, 7, 6, 5, 4, 3, 2, 1, 0, 7 y así sucesivamente. Como cuenta en dirección descendente, decimos que es un *contador descendente*.

Es posible combinar la funcionalidad de los circuitos de las figuras 7.20*a* y 7.21*a* para formar un contador que pueda contar ascendente o descendentemente. Un contador como éste se



a) Circuito



b) Diagrama de tiempo

Figura 7.21 Un contador descendente de tres bits.

llama *contador ascendente/descendente*. Dejamos la deducción de este contador como ejercicio para el lector (problema 7.16).

7.9.2 CONTADORES SÍNCRONOS

Los contadores asíncronos de las figuras 7.20*a* y 7.21*a* son simples, pero no muy rápidos. Si se construye así un contador con un número grande de bits, las demoras causadas por el esquema de sincronización en cascada pueden volverse demasiado grandes para satisfacer los requisitos de desempeño deseados. Podemos construir un contador más rápido si sincronizamos todos los flip-flops al mismo tiempo aplicando el método descrito enseguida.

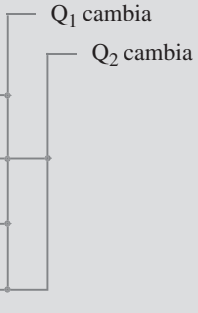
Contador síncrono con flip-flops T

En la tabla 7.1 se muestra el contenido de un contador ascendente de tres bits para ocho ciclos del reloj consecutivos, suponiendo que el conteo empieza en 0. Al observar el patrón de bits de cada fila de la tabla es claro que el bit Q_0 cambia en cada ciclo del reloj. El bit Q_1 cambia sólo cuando $Q_0 = 1$. El bit Q_2 cambia únicamente cuando Q_1 y Q_0 son iguales a 1. En general, para un contador ascendente de *n* bits, un flip-flop dado cambia su estado sólo cuando todos los flip-flops anteriores están en el estado Q = 1. Por consiguiente, si usamos flip-flops T para hacer el contador, entonces las entradas *T* se definen como

$$
\begin{aligned}
T_0 &= 1 \\
T_1 &= Q_0 \\
T_2 &= Q_0 Q_1 \\
T_3 &= Q_0 Q_1 Q_2 \\
&\vdots \\
T_n &= Q_0 Q_1 \cdots Q_{n-1}
\end{aligned}
$$

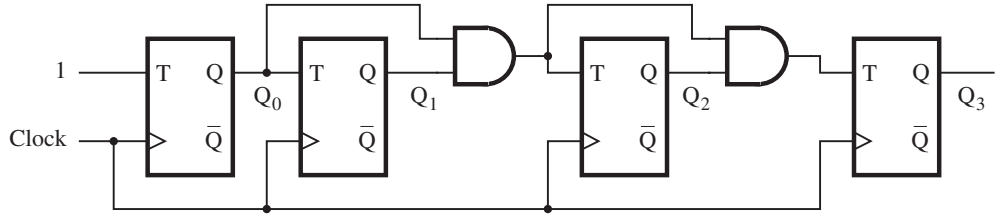
Tabla 7.1 Derivación del contador síncrono ascendente

Ciclo del reloj	Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8	0	0	0

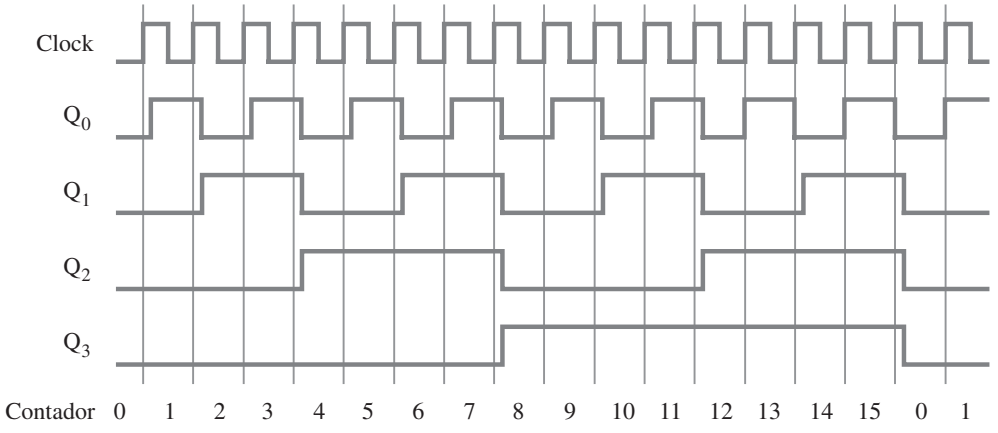


Un ejemplo de un contador de cuatro bits basado en estas expresiones aparece en la figura 7.22*a*. En vez de usar compuertas AND de tamaño aumentado para cada etapa, lo que puede suscitar problemas del factor de carga de entrada (*fan-in*), empleamos un arreglo factorizado, como se muestra en la figura. Este arreglo no reduce la respuesta del contador, ya que todos los flip-flops cambian sus estados después de un retraso de propagación desde el flanco positivo del reloj. Nótese que un cambio en el valor de Q_0 debe propagarse por varias compuertas AND para llegar a los flip-flops en las etapas superiores del contador, lo cual requiere cierta cantidad de tiempo. Este tiempo no debe exceder el periodo del reloj. En realidad, debe ser menor que tal periodo menos el tiempo de preparación para los flip-flops.

En la figura 7.22*b* se presenta un diagrama de tiempo. Se muestra que el circuito se comporta como un contador ascendente módulo 16. Debido a que todos los cambios ocurren con el mismo retraso después del flanco activo de la señal *Clock*, el circuito se llama *contador síncrono*.



a) Circuito



b) Diagrama de tiempo

Figura 7.22 Un contador síncrono ascendente de cuatro bits.

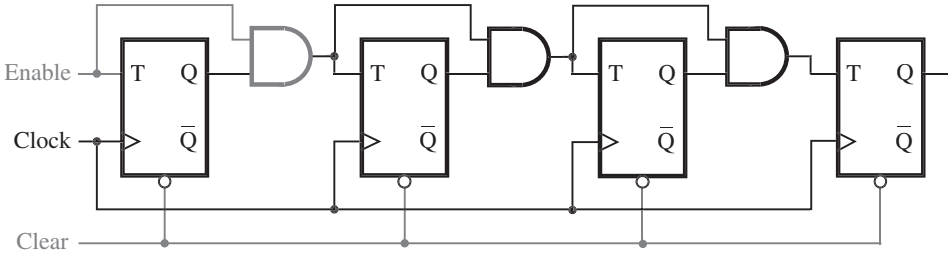


Figura 7.23 Inclusión de las capacidades Enable y Clear.

Capacidades Enable y Clear

Los contadores de las figuras 7.20 a 7.22 cambian su contenido en respuesta a cada pulso del reloj. Con frecuencia es aconsejable inhibir el conteo, de modo que siga en su etapa presente. Ello puede lograrse incluyendo una señal de control *Enable*, como se indica en la figura 7.23. El circuito es el contador de la figura 7.22, donde la señal *Enable* controla directamente la entrada *T* del primer flip-flop. Conectar además *Enable* a la cadena de la compuerta AND significa que si *Enable* = 0, entonces todas las entradas *T* serán iguales a 0. Si *Enable* = 1, entonces el contador funcionará como se explicó antes.

En muchas aplicaciones es preciso empezar con el conteo igual a cero, lo cual se logra fácilmente si los flip-flops pueden borrarse, como se explicó en la sección 7.4.3. Las entradas clear de todos los flip-flops pueden unirse y manejarse por medio de una entrada de control *Clear*.

Contador síncrono con flip-flops D

Si bien la función toggle hace que los flip-flops T sean una opción natural para la implementación de los contadores, también es posible construir contadores usando otros tipos de flip-flops. Los flip-flops JK pueden emplearse exactamente igual que los flip-flops T, ya que si las entradas *J* y *K* se unen, un flip-flop JK se convierte en un flip-flop T. Ahora consideraremos ocupar flip-flops D para este propósito.

No es claro cómo pueden utilizarse los flip-flops D para implementar un contador. Presentaremos un método formal para derivar estos circuitos en el capítulo 8. Aquí sólo mostramos una estructura de circuito que cumple los requisitos, pero dejaremos la deducción para dicho capítulo. En la figura 7.24 se observa un contador ascendente de cuatro bits que cuenta en la secuencia 0, 1, 2, . . . ,14, 15, 0, 1 y así sucesivamente. El conteo se indica mediante las salidas del flip-flop $Q_3Q_2Q_1Q_0$. Si suponemos que *Enable* = 1, entonces las entradas *D* de los flip-flops se definen mediante las expresiones

$$D_0 = \overline{Q}_0 = 1 \oplus Q_0$$
$$D_1 = Q_1 \oplus Q_0$$
$$D_2 = Q_2 \oplus Q_1Q_0$$
$$D_3 = Q_3 \oplus Q_2Q_1Q_0$$

Para un contador más grande la etapa *i*-ésima se define por

$$D_i = Q_i \oplus Q_{i-1}Q_{i-2} \cdots Q_1Q_0$$

En el capítulo 8 mostraremos cómo derivar estas ecuaciones.

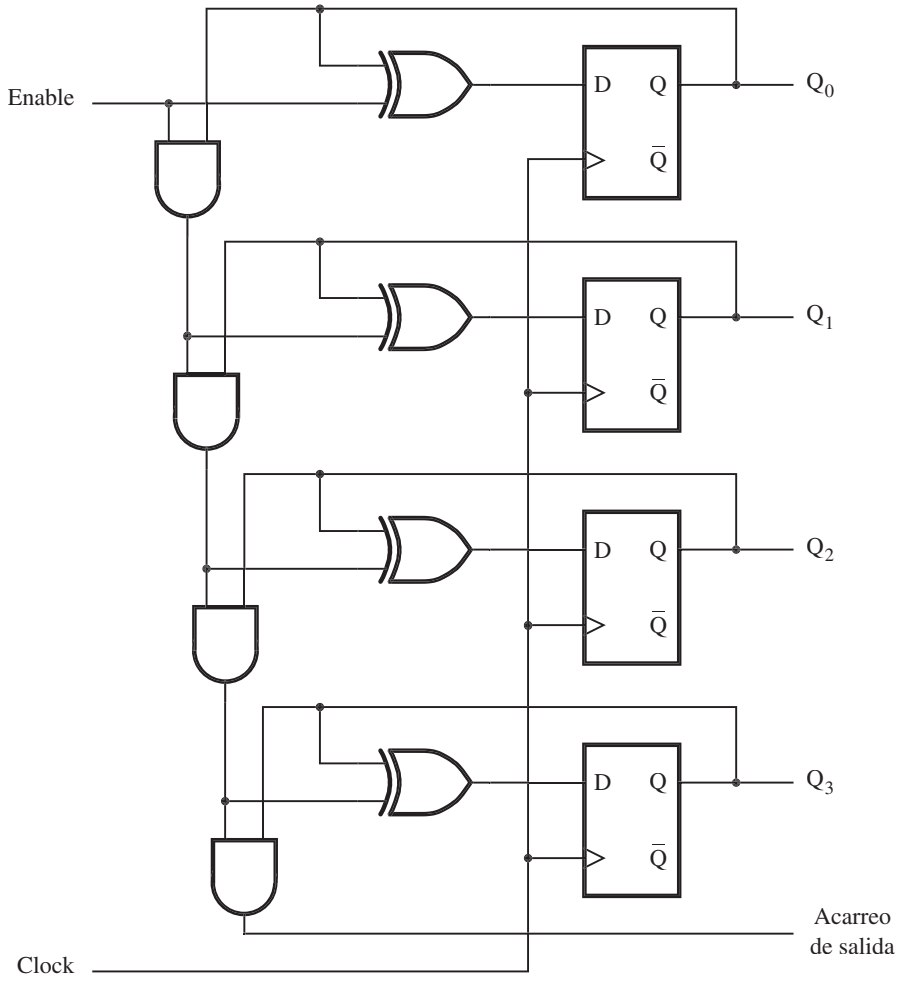


Figura 7.24 Un contador de cuatro bits con flip-flops D.

Hemos incluido la señal de control *Enable* de manera que el contador cuente, valga la expresión, los pulsos del reloj sólo cuando *Enable* = 1. De hecho, las ecuaciones anteriores se modificaron para implementar el circuito de la figura como sigue

$$D_0 = Q_0 \oplus Enable$$
$$D_1 = Q_1 \oplus Q_0 \cdot Enable$$
$$D_2 = Q_2 \oplus Q_1 \cdot Q_0 \cdot Enable$$
$$D_3 = Q_3 \oplus Q_2 \cdot Q_1 \cdot Q_0 \cdot Enable$$

El funcionamiento del contador se basa en nuestra observación de la tabla 7.1 acerca de que el estado del flip-flop en la etapa *i* cambia sólo si todos los flip-flops anteriores están en el estado Q = 1.

Esto ocasiona que la salida de la compuerta AND que alimenta la etapa *i* sea igual a 1, lo cual hace que la salida de la compuerta XOR conectada a D_i sea igual a $\overline{Q}_i$. De lo contrario, la salida de la compuerta XOR proporciona $D_i = Q_i$, y el flip-flop permanece en el mismo estado. Esto se parece a la propagación de acarreo en un circuito sumador con acarreo de adelanto (véase la sección 5.4); por consiguiente, la cadena de la compuerta AND puede considerarse la *cadena de acarreo*. Aun cuando el circuito es sólo un contador de cuatro bits, hemos incluido un AND adicional que produce el "acarreo de salida". Esta señal facilita la concatenación de dos de estos contadores de cuatro bits para crear un contador de ocho bits.

Finalmente, el lector debe notar que el contador de la figura 7.24 es, en esencia, el mismo que el circuito de la figura 7.23. En la figura 7.16*a* mostramos que un flip-flop T puede formarse a partir de un flip-flop D si se proporcionan las compuertas adicionales que dan

$$\begin{aligned} D &= Q\overline{T} + \overline{Q}T \\ &= Q \oplus T \end{aligned}$$

De esta manera, en cada etapa de la figura 7.24 el flip-flop D y la compuerta XOR asociada implementan la funcionalidad de un flip-flop T.

7.9.3 CONTADORES CON CARGA EN PARALELO

A menudo es necesario empezar a contar a partir de 0. Tal estado puede lograrse usando la capacidad para borrar los flip-flops como se indica en la figura 7.23. Pero a veces es conveniente comenzar con un conteo distinto. Para permitir este modo de operación un circuito contador debe tener algunas entradas a través de las cuales pueda cargarse el conteo inicial. Usar las entradas *Clear* y *Preset* para tal propósito es una posibilidad, mas enseguida se expone un mejor método.

El circuito de la figura 7.24 puede modificarse para brindar la capacidad de carga en paralelo como se muestra en la figura 7.25. Un multiplexor de dos entradas se inserta antes de cada entrada *D*. Una entrada del multiplexor se usa para proporcionar la operación de conteo normal; la otra es un bit de datos que puede cargarse directamente en el flip-flop. Una entrada de control, *Load*, se utiliza para elegir el modo de operación. El circuito cuenta cuando $Load = 0$. Un nuevo valor inicial, $D_3D_2D_1D_0$, se carga en el contador cuando $Load = 1$.

7.10 RESET SÍNCRONO

Ya mencionamos que es importante poder borrar, o *inicializar*, el contenido de un contador antes de empezar una operación de conteo. Ello se logra usando la capacidad de borrado de cada flip-flop. Sin embargo, es posible que también estemos interesados en establecer el conteo en 0 durante el proceso de conteo normal. Un contador ascendente de *n* bits funciona naturalmente como un contador módulo 2^n. Supóngase que queremos tener un contador que cuente en módulo cierta base que no sea una potencia de 2. Por ejemplo, tal vez queramos diseñar un contador módulo 6, para el que la secuencia de conteo sea 0, 1, 2, 3, 4, 5, 0, 1 y así sucesivamente.

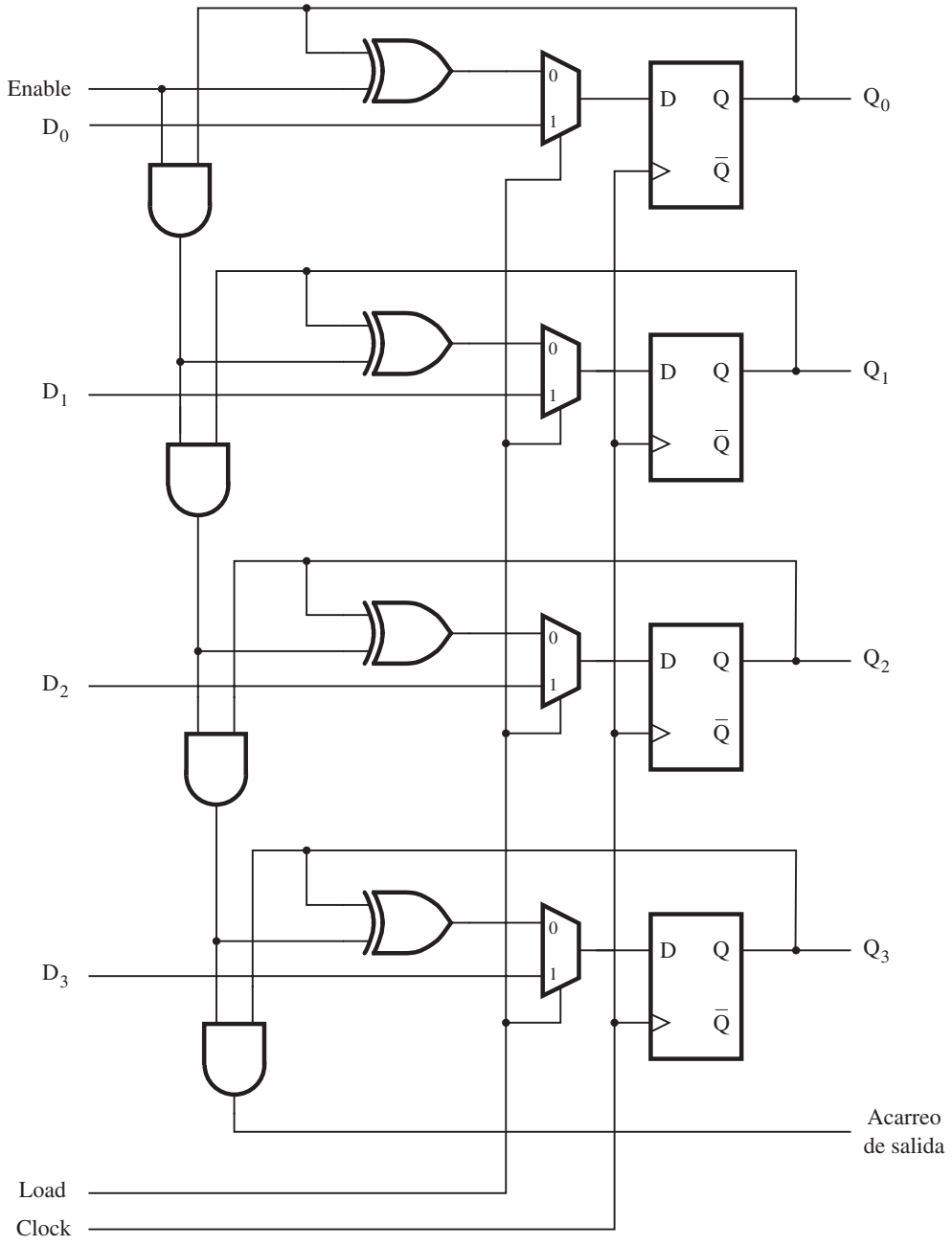
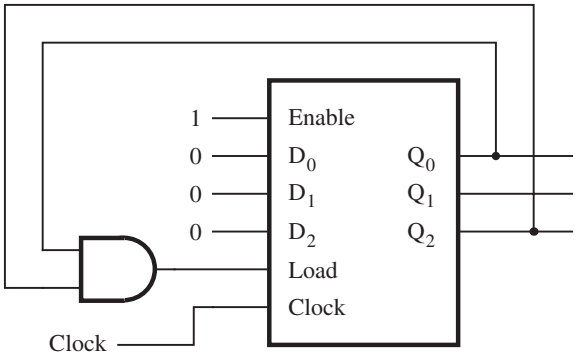
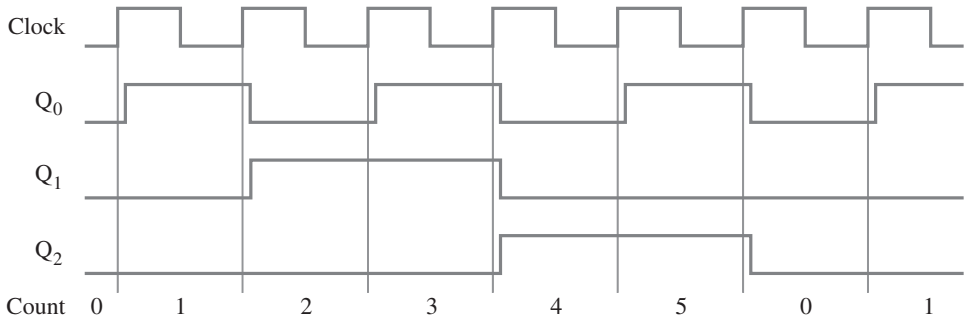


Figura 7.25 Un contador con capacidad de carga en paralelo.

El método más sencillo es reconocer cuándo el conteo llega a 5 y luego inicializar el contador. Puede usarse una compuerta AND para detectar que el conteo alcancó el 5. En realidad, basta determinar que $Q_2 = Q_0 = 1$, lo cual es verdadero sólo para 5 en nuestra secuencia de conteo. En la figura 7.26*a* aparece un circuito basado en este método. Utiliza un contador síncrono de tres bits del tipo representado en la figura 7.25. La función de carga en paralelo del contador se usa para inicializar su contenido cuando el contador llega a 5. La acción de inicialización ocurre en el flanco positivo del reloj después que el conteo alcanza el 5. Comprende la carga de $D_2D_1D_0 = 000$ en los flip-flops. Como se vio en el diagrama de tiempo de la figura 7.26*b*, la secuencia de conteo buscada se logra con cada valor del conteo establecido para un ciclo completo del reloj. Como el contador se inicializa en el flanco activo del reloj, decimos que este tipo de contador tiene un *reset síncrono*.



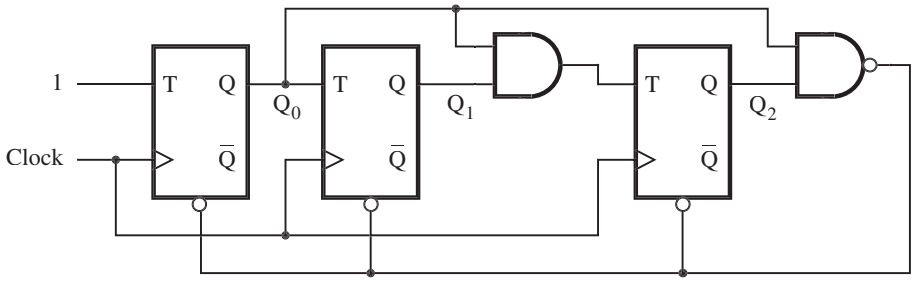
a) Circuito



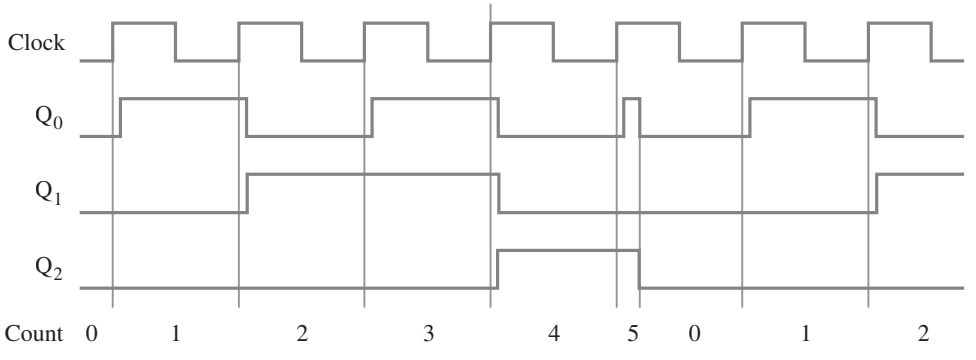
b) Diagrama de tiempo

Figura 7.26 Un contador módulo 6 con reset síncrono.

Considérese ahora la posibilidad de usar la función de borrado de los flip-flops individuales, en vez del método de carga en paralelo. En el circuito de la figura 7.27*a* se ilustra una posibilidad; utiliza la estructura del contador de la figura 7.22*a*. Como las entradas *Clear* están activas en nivel bajo, se utiliza una compuerta NAND para detectar la ocurrencia del conteo de 5 y borrar los tres flip-flops. Desde el punto de vista conceptual, esto parece funcionar bien, pero una revisión más a fondo revela un problema potencial. El diagrama de tiempo para este circuito se presenta en la figura 7.27*b*. Muestra una dificultad que surge cuando el conteo llega a 5. En cuanto el contador llega a su valor, la compuerta NAND desencadena la acción de inicialización. Los flip-flops se borran en 0 poco tiempo después que la compuerta NAND ha detectado el conteo de 5. Este instante depende de los retrasos de compuerta en el circuito, pero no del reloj. Por consiguiente, los valores de señal $Q_2Q_1Q_0 = 101$ se conservan durante un tiempo mucho menor que un ciclo del reloj. Según la aplicación concreta de dicho contador, éste puede ser adecuado pero también completamente inaceptable. Por ejemplo, si el contador se usa en un sistema digital donde todas las operaciones del sistema están sincronizadas por el mismo reloj, entonces este pulso estrecho que indica *Count* = 5 no sería visto por el resto del sistema. Para resolver el problema



a) Circuito



b) Diagrama de tiempo

Figura 7.27 Un contador módulo 6 con un reset asíncrono.

podríamos tratar de usar un contador módulo 7 en su lugar, suponiendo que el sistema ignoraría el pulso corto que indica el conteo de 6. Ésta no es una buena forma de diseñar circuitos porque los pulsos no deseados a menudo causan dificultades imprevistas en la práctica. Se dice que el método empleado en la figura 7.27*a* usa un *reset asíncrono*.

Los diagramas de tiempo de las figuras 7.26*b* y 7.27*b* indican que el reset síncrono es una mejor opción que el asíncrono. La misma observación es cierta si la secuencia de conteo natural ha de dividirse por la carga de un valor distinto de cero. El nuevo valor de conteo puede establecerse limpiamente mediante la función de carga en paralelo. La opción de usar las capacidades de clear y preset de los flip-flops individuales para establecer en 1 sus estados a fin de reflejar el conteo deseado plantea los mismos problemas estudiados junto con el reset asíncrono.

7.11 OTROS TIPOS DE CONTADORES

En esta sección estudiaremos otros tres tipos de contadores que pueden encontrarse en las aplicaciones prácticas. El primero usa la secuencia de conteo decimal y los otros dos generan secuencias de códigos que no representan números binarios.

7.11.1 CONTADOR BCD

Los contadores binario-codificado-decimal (BCD) pueden diseñarse con el método expuesto en la sección 7.10. En la figura 7.28 se muestra un contador de dos dígitos BCD. Consta de dos contadores módulo 10, uno por cada dígito BCD, que implementamos usando el contador de cuatro bits de carga en paralelo de la figura 7.25. Obsérvese que en un contador módulo 10 es preciso inicializar los cuatro flip-flops después de que se ha obtenido el conteo de 9. De esta manera la entrada *Load* para cada etapa es igual a 1 cuando $Q_3 = Q_0 = 1$, lo que ocasiona que los ceros se carguen en los flip-flops en el siguiente flanco positivo de la señal de reloj. Siempre que el conteo en la etapa 0, BCD_0, llega a 9 es necesario habilitar la segunda etapa de modo que se incremente cuando llegue el siguiente pulso del reloj. Ello se logra al mantener la señal *Enable* para BCD_1 baja en todo momento excepto cuando $BCD_0 = 9$.

En la práctica debe ser posible borrar el contenido del contador al activar una señal de control. Dos compuertas OR se incluyen en el circuito para tal fin. La entrada de control *Clear* puede usarse para cargar los ceros en el contador. Nótese que en este caso *Clear* está activa cuando es alta. El código de VHDL para un contador BCD de dos dígitos se da en la figura 7.77.

En cualquier sistema digital suele haber una o más señales de reloj usadas para manejar todo el sistema de circuitos síncrono. En el contador anterior, así como en todos los presentados en las figuras previas, hemos supuesto que el objetivo es contar el número de pulsos del reloj. Desde luego, estos contadores sirven para contar el número de pulsos de cualquier señal que pueda usarse en vez de la de reloj.

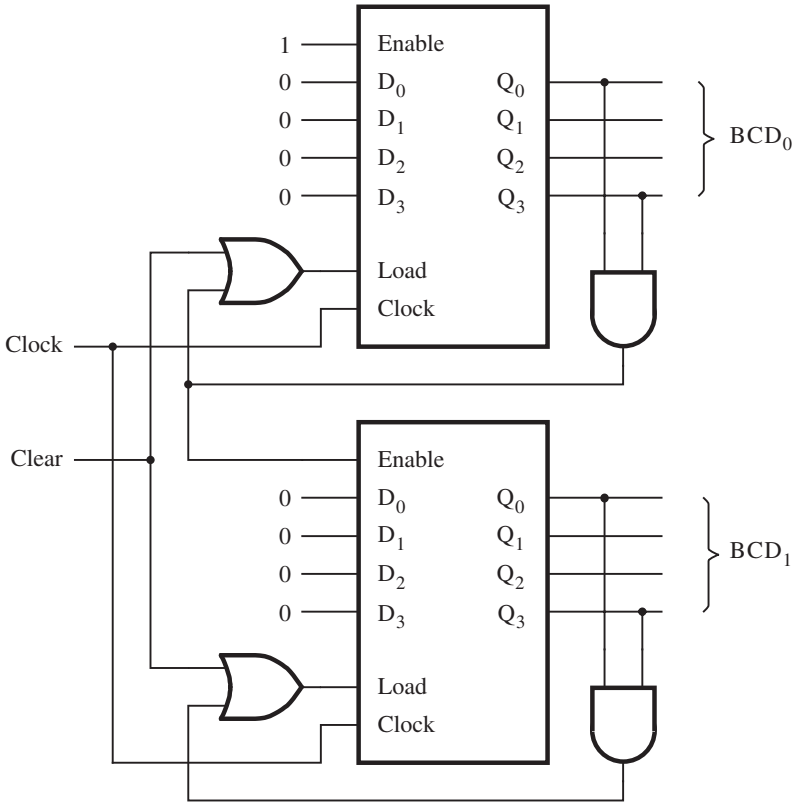
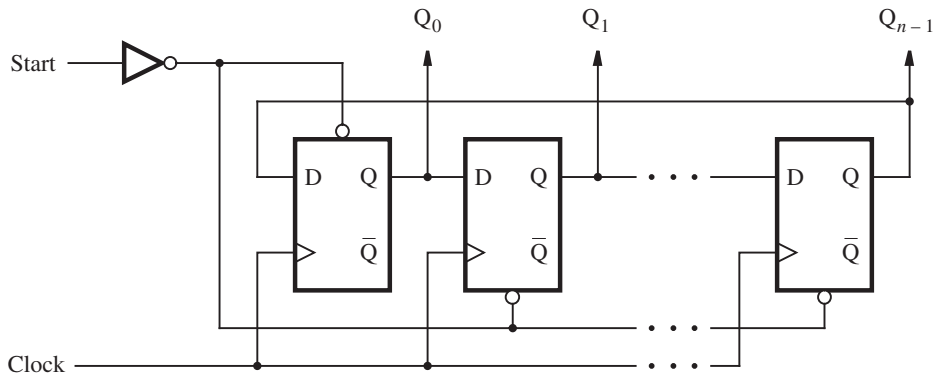


Figura 7.28 Un contador BCD de dos dígitos.

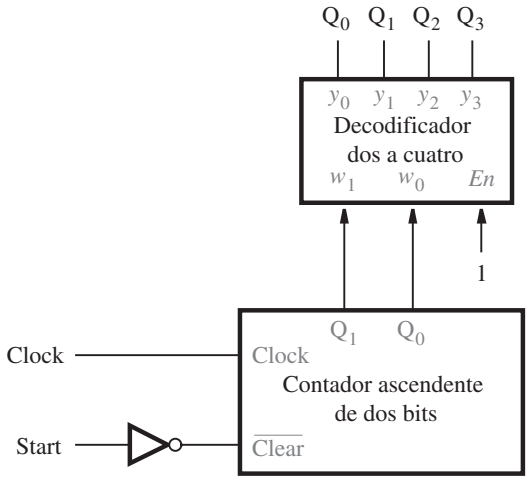
7.11.2 CONTADOR EN ANILLO

En los contadores anteriores el conteo se indica por medio del estado de los flip-flops del contador. En todos los casos el conteo es un número binario. Al usar dichos contadores, si se va a tomar una medida como resultado de un conteo en particular, es necesario detectar la ocurrencia de este conteo. Esto puede hacerse usando compuertas AND, como se ilustra en las figuras 7.26 a 7.28.

Es posible concebir un circuito tipo contador en el que cada flip-flop alcance el estado $Q_i = 1$ para exactamente un conteo, mientras que para todos los demás conteos $Q_i = 0$. Entonces Q_i indica directamente una ocurrencia del conteo correspondiente. En realidad, como esto no representa números binarios, es mejor decir que las salidas de los flips-flops representan un código. Un circuito como éste puede construirse a partir de un registro de corrimiento simple, como se indica en la figura 7.29*a*. La salida Q de la última etapa en el registro de corrimiento se realimenta como la entrada a la primera etapa, la cual crea una estructura en forma de anillo. Si un solo 1 se inyecta en el anillo, este 1 se desplazará a través del anillo en ciclos del reloj sucesivos.



a) Un contador de *n* bits en anillo



b) Un contador de cuatro bits en anillo

Figura 7.29 Contador en anillo.

Por ejemplo, en una estructura de cuatro bits, los códigos posibles $Q_0Q_1Q_2Q_3$ serán 1000, 0100, 0010 y 0001. Como dijimos en la sección 6.2, dicha codificación, donde hay un solo 1 y el resto de las variables de código son 0, se llama *codificación de 1 activo*.

El circuito de la figura 7.29*a* se conoce como *contador en anillo*. Su operación tiene que inicializarse inyectando un 1 en la primera etapa, lo que se logra utilizando la señal de control *Start*, que prestablece en 1 el flip-flop del extremo izquierdo y borra los otros flip-flops hasta dejarlos en 0. Suponemos que todos los cambios en el valor de la señal *Start* ocurren poco tiempo después de un flanco activo del reloj, de manera que los parámetros de tiempo del flip-flop no sean incumplidos.

El circuito de la figura 7.29*a* puede usarse para construir un contador en anillo con cualquier número de bits, *n*. Para el caso específico de $n = 4$, el inciso (*b*) de la figura muestra cómo puede construirse un contador en anillo usando un contador ascendente de dos bits y un decodificador. Cuando *Start* se establece en 1, el contador se restablece en 00. Después que *Start* cambia de nuevo a 0, el contador aumenta su valor de manera normal. El decodificador dos a cuatro, descrito en la sección 6.2, cambia la salida del contador en codificación de 1 activo. Para los valores de conteo 00, 01, 10, 11, 00, etc., el decodificador produce $Q_0Q_1Q_2Q_3 = 1000, 0100, 0010, 0001, 1000$ y sucesivamente. Esta estructura del circuito puede usarse para contadores en anillo más grandes, siempre que el número de bits sea una potencia de dos. En la sección 7.14 daremos un ejemplo de un circuito más grande que emplea el contador en anillo de la figura 7.29*b* como un subcircuito.

7.11.3 CONTADOR JOHNSON

Una variación interesante del contador en anillo se obtiene si, en vez de la salida Q, tomamos la salida $\overline{Q}$ de la última etapa y la retroalimentamos en la primera etapa, como se muestra en la figura 7.30. Este circuito se conoce como *contador Johnson*. Un contador de *n* bits de este tipo genera una secuencia de conteo de longitud 2*n*. Por ejemplo, un contador de cuatro bits produce la secuencia 0000, 1000, 1100, 1110, 1111, 0111, 0011, 0001, 0000, etc. Nótese que en esta secuencia solamente un bit tiene un valor diferente para dos códigos consecutivos.

Para empezar la operación del contador Johnson es preciso inicializar todos los flip-flops, como se muestra en la figura. Obsérvese que ni el contador Johnson ni el contador en anillo generarán la secuencia de conteo buscada si no se inicializa apropiadamente.

7.11.4 COMENTARIOS SOBRE EL DISEÑO DEL CONTADOR

Los circuitos secuenciales presentados en este capítulo, concretamente los registros y contadores, tienen una estructura que permite que los circuitos se diseñen mediante un método intuitivo. En el capítulo 8 presentaremos un enfoque más formal para diseñar circuitos secuenciales y mostrar cómo los circuitos presentados en este capítulo pueden derivarse siguiéndolo.

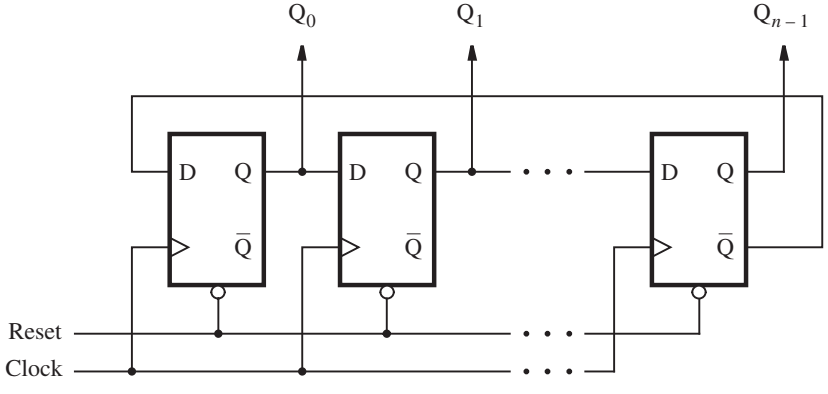


Figura 7.30 Contador Johnson.

7.12 USO DE ELEMENTOS DE ALMACENAMIENTO CON HERRAMIENTAS CAD

En esta sección se muestra cómo diseñar circuitos con elementos de almacenamiento usando ya sea una captura esquemática o código de VHDL.

7.12.1 INCLUSIÓN DE ELEMENTOS DE ALMACENAMIENTO EN ESQUEMAS

Una forma de crear un circuito consiste en dibujar un esquema que construya latches y flip-flops a partir de compuertas lógicas. Como estos elementos de almacenamiento se utilizan en muchas aplicaciones, la mayor parte de los sistemas CAD los proporcionan como módulos preconstruidos. En la figura 7.31 se muestra un esquema creado con una herramienta de captura esquemática; incluye tres tipos de flip-flops que se importan de una biblioteca provista como parte del sistema CAD. El elemento superior es un latch D asíncrono, el de en medio es un flip-flop D disparado por flanco positivo y el inferior es un flip-flop T disparado por flanco positivo. Los flip-flops D y T tienen entradas clear y preset asíncronas, activas en nivel bajo. Si estas entradas no están conectadas en un esquema, entonces la herramienta CAD las vuelve inactivas asignándoles el valor predeterminado de 1.

Cuando el latch D asíncrono se sintetiza para implementarlo en un chip, la herramienta CAD tal vez no genere las compuertas NOR o NAND con acoplamiento cruzado expuestas en la sección 7.2. En algunos chips, como los CPLD, el circuito AND-OR descrito en la figura 7.32 puede ser preferible. Este circuito equivale funcionalmente a la versión con acoplamiento cruzado de la sección 7.2. El circuito de suma de productos se usa porque es más adecuado para

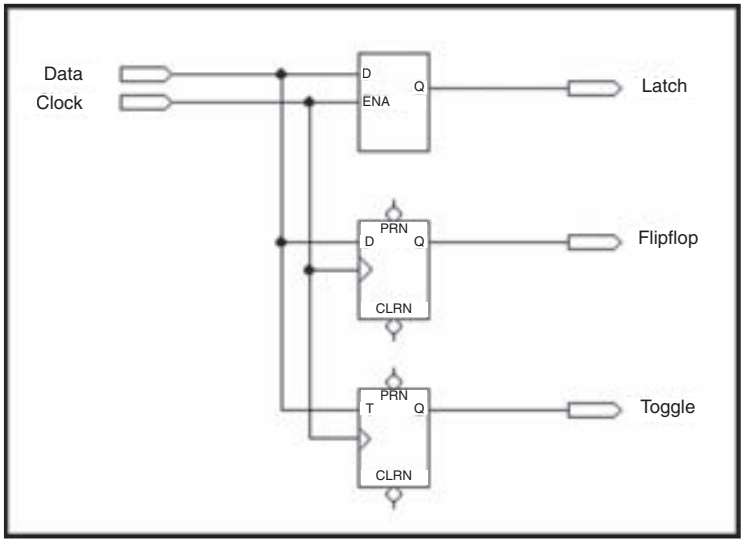


Figura 7.31 Tres tipos de elementos de almacenamiento en un esquema.

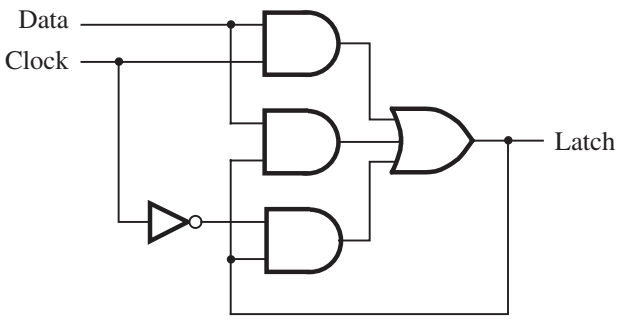


Figura 7.32 Latch D asíncrono generado mediante herramientas CAD.

la implementación en una macrocelda CPLD. Cabe destacar un aspecto de este circuito. Desde el punto de vista funcional, parece que el circuito puede simplificarse si se elimina la compuerta AND con las entradas *Data* y *Latch*. Sin esta compuerta, la compuerta AND superior establece el valor almacenado en el latch cuando el reloj está en 1, y la compuerta AND inferior mantiene el valor almacenado cuando el reloj es 0. Pero sin esta compuerta el circuito tiene un problema de tiempo conocido como *riesgo estático*. En la sección 9.6 explicaremos pormenorizadamente los riesgos.

El circuito de la figura 7.31 puede implementarse en un CPLD como se muestra en la figura 7.33. Los flip-flops D y T se realizan usando en el chip flip-flops configurables como tipos D o T. En la figura se representan en gris oscuro las compuertas y los cables requeridos para implementar el circuito de la figura 7.31.

Los resultados de una simulación de tiempo para la implementación de la figura 7.33 se dan en la figura 7.34. La señal *Latch*, que es la salida del latch D asíncrono, implementado como se indicó en la figura 7.32, sigue a la entrada *Data* siempre que la señal de *Clock* es 1. Debido a los retrasos de propagación en el chip, la señal *Latch* se retarda en el tiempo respecto a la señal *Data*. Como la señal *Flipflop* es la salida del flip-flop D, cambia sólo después de un flanco positivo del reloj. De igual modo, la salida del flip-flop T, llamada *Toggle* en la figura, alterna cuando *Data* = 1 y ocurre un flanco positivo del reloj. El diagrama de tiempo ilustra el retraso cuando el flanco positivo del reloj ocurre en el pin de entrada del chip hasta que un cambio en la salida del flip-flop aparece en el pin de salida del chip. A este tiempo se le llama *tiempo desde reloj hasta la salida*, t_{co}.

7.12.2 USO DE CONSTRUCTORES DE VHDL PARA ELEMENTOS DE ALMACENAMIENTO

En la sección 6.6 describimos una serie de instrucciones de asignación de VHDL. Las instrucciones IF y CASE se presentaron como dos tipos de instrucciones de asignación secuencial. En esta sección mostramos cómo usar tales instrucciones para describir elementos de almacenamiento.

En la figura 6.43, que se repite en la figura 7.35, se brinda un ejemplo de código de VHDL que tiene memoria implícita. Como el código no especifica qué valor debe tener la señal *AeqB* cuando la condición para la instrucción IF no está satisfecha, la semántica especifica que en este

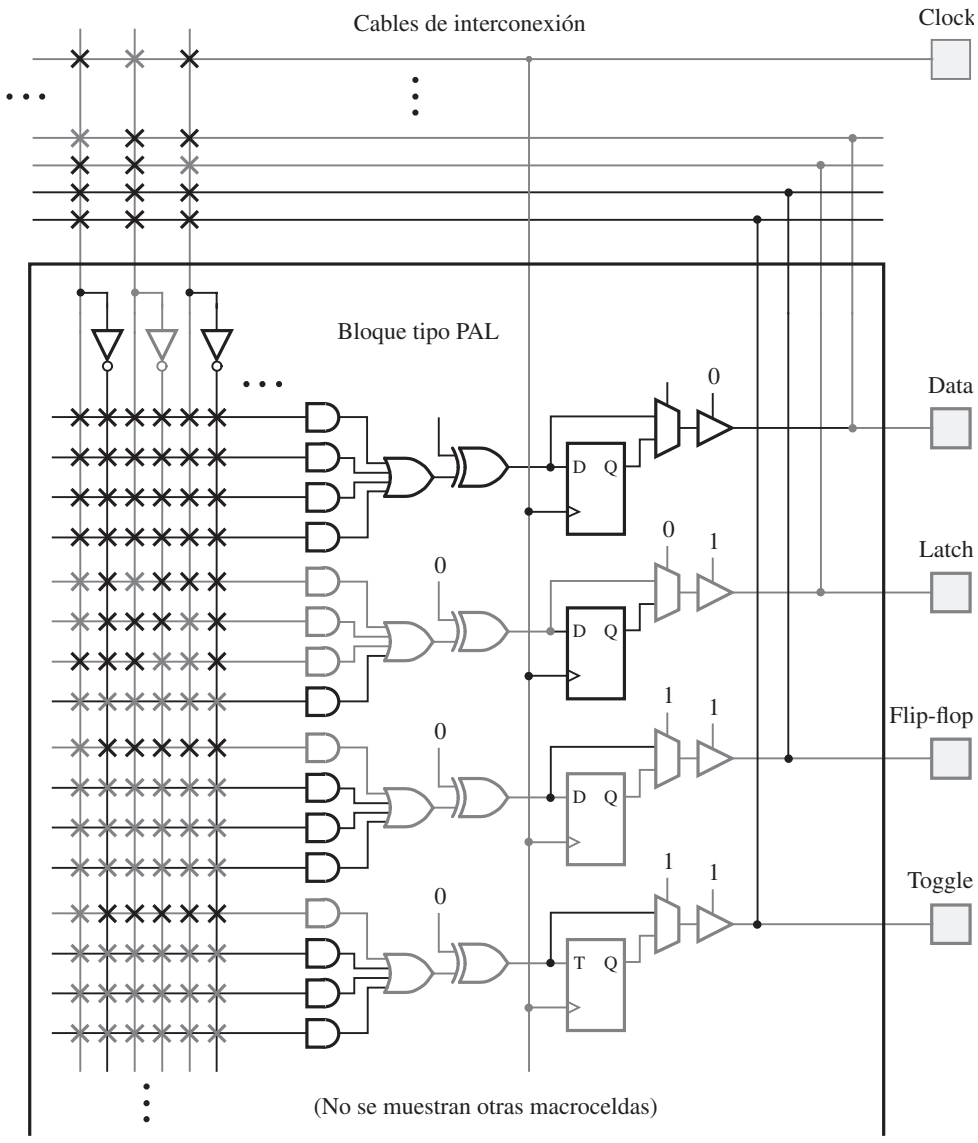


Figura 7.33 Implementación del esquema de la figura 7.31 en un CPLD.

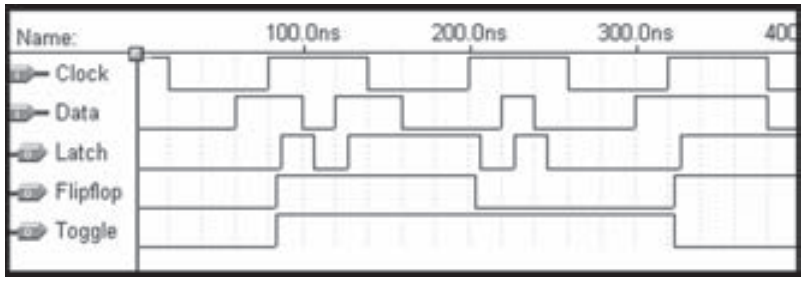


Figura 7.34 Simulación de tiempo para los elementos de almacenamiento de la figura 7.31.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY implied IS
    PORT ( A, B   : IN     STD_LOGIC ;
           AeqB : OUT  STD_LOGIC ) ;
END implied ;

ARCHITECTURE Behavior OF implied IS
BEGIN
    PROCESS ( A, B )
    BEGIN
        IF A = B THEN
            AeqB <= '1' ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Figura 7.35 El código de la figura 6.43, ilustrando la memoria implícita.

caso *AeqB* debe conservar su valor actual. La memoria implícita es el concepto clave usado para describir los elementos de circuitos secuenciales, los cuales ilustraremos con varios ejemplos.

Ejemplo 7.1

CÓDIGO PARA UN LATCH D ASÍNCRONO El código de la figura 7.36 define una entidad llamada *latch*, la cual tiene las entradas *D* y *Clk* y la salida Q. El proceso utiliza una instrucción if-then-else para definir el valor de la salida Q. Cuando *Clk* = 1, Q toma el valor de *D*. Para el caso en que *Clk* no es 1, el código no especifica qué valor debe tener Q. Por consiguiente, Q conservará su valor actual en este caso, y el código describe un latch D asíncrono. La lista de

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY latch IS
    PORT ( D, Clk  : IN     STD_LOGIC ;
           Q       : OUT    STD_LOGIC) ;
END latch ;

ARCHITECTURE Behavior OF latch IS
BEGIN
    PROCESS ( D, Clk )
    BEGIN
        IF Clk = '1' THEN
            Q <= D ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Figura 7.36 Código para un latch D asíncrono.

sensibilidad del proceso incluye tanto *Clk* como *D* porque estas señales pueden causar un cambio en el valor de la salida Q.

Ejemplo 7.2 **CÓDIGO PARA UN FLIP-FLOP D** En la figura 7.37 se define una entidad llamada *flipflop*, que es un flip-flop D disparado por flanco positivo. El código es idéntico al de la figura 7.36 con dos excepciones. Primera, la lista de sensibilidad del proceso contiene sólo la señal de reloj que puede causar un cambio en la salida Q. Segunda, la instrucción if-then-else utiliza una condición diferente de la empleada en el latch. La sintaxis Clock'EVENT usa un constructor de VHDL llamado *atributo*. Un atributo se refiere a la propiedad de un objeto, como una señal. En este caso el atributo 'EVENT se refiere a cualquier cambio en la señal *Clock*. Combinar la condición Clock'EVENT con la condición *Clock* = 1 significa que "el valor de la señal *Clock* acaba de cambiar, y el valor ahora es igual a 1". Por consiguiente, la condición se refiere a un flanco positivo del reloj. Puesto que la salida Q cambia sólo como resultado de un flanco positivo del reloj, el código describe un flip-flop D disparado por flanco positivo.

Ejemplo 7.3 **CÓDIGO OPCIONAL PARA UN FLIP-FLOP D** El proceso de la figura 7.38 emplea una sintaxis distinta de aquella de la figura 7.37 para describir el flip-flop D. Utiliza la instrucción WAIT UNTIL Clock'EVENT AND Clock = '1', que produce el mismo efecto que la instrucción IF de la figura 7.37. Un proceso que usa una instrucción WAIT UNTIL es un caso especial porque se omite la lista de sensibilidad. El constructor WAIT UNTIL implica que dicha lista incluye sólo la señal de reloj. En nuestro uso de VHDL, el cual es para la síntesis de circuitos, un proceso puede utilizar una instrucción WAIT UNTIL sólo si es su primera instrucción.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY flipflop IS
    PORT ( D, Clock  : IN   STD_LOGIC ;
           Q         : OUT  STD_LOGIC) ;
END flipflop ;

ARCHITECTURE Behavior OF flipflop IS
BEGIN
    PROCESS ( Clock )
    BEGIN
        IF Clock'EVENT AND Clock = '1' THEN
            Q <= D ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Figura 7.37 Código para un flip-flop D.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY flipflop IS
    PORT ( D, Clock  : IN   STD_LOGIC ;
           Q         : OUT  STD_LOGIC ) ;
END flipflop ;

ARCHITECTURE Behavior OF flipflop IS
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL Clock'EVENT AND Clock = '1' ;
        Q <= D ;
    END PROCESS ;
END Behavior ;
```

Figura 7.38 Código equivalente al de la figura 7.37, que usa una instrucción WAIT UNTIL.

En realidad, el atributo 'EVENT es redundante en la instrucción WAIT UNTIL. Podemos simplemente escribir

```
WAIT UNTIL Clock = '1';
```

lo cual también implica que la acción ocurre cuando la señal *Clock* se vuelve igual a 1, es decir, en el flanco donde la señal cambia de 0 a 1. Sin embargo, algunas herramientas CAD de síntesis requieren la inclusión del atributo 'EVENT, razón por la que usamos este estilo en el libro.

En general, siempre que se quiere incluir en el código de VHDL flip-flops sincronizados por el flanco positivo del reloj se usa la condición Clock'EVENT AND Clock '1'. Cuando esta condición aparece en una instrucción IF, cualesquiera señales a las que se asignen valores dentro de la instrucción IF se implementan como las salidas de los flip-flops. Cuando la condición se usa en una instrucción WAIT UNTIL, cualquier señal a la que se asigna un valor en todo el proceso se implementa como la salida de un flip-flop.

Las diferencias entre usar instrucciones IF e instrucciones WAIT UNTIL se estudian con más detalle en el apéndice A, sección A.10.3.

Ejemplo 7.4 **BORRADO ASÍNCRONO** En la figura 7.39 se muestra un proceso parecido al de la figura 7.37. Describe un flip-flop D con una entrada de reset asíncrono activo en nivel bajo (clear). Cuando *Resetn*, la entrada de inicialización, es igual a 0, la salida Q del flip-flop se establece en 0.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY flipflop IS
    PORT ( D, Resetn, Clock  : IN     STD_LOGIC ;
           Q                 : OUT  STD_LOGIC) ;
END flipflop ;

ARCHITECTURE Behavior OF flipflop IS
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
        IF Resetn = '0' THEN
            Q <= '0' ;
        ELSIF Clock'EVENT AND Clock = '1' THEN
            Q <= D ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Figura 7.39 Flip-flop D con reset asíncrono.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY flipflop IS
    PORT ( D, Resetn, Clock  : IN   STD_LOGIC ;
           Q                 : OUT  STD_LOGIC) ;
END flipflop ;

ARCHITECTURE Behavior OF flipflop IS
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL Clock'EVENT AND Clock = '1' ;
        IF Resetn = '0' THEN
            Q <= '0' ;
        ELSE
            Q <= D ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Figura 7.40 Flip-flop D con reset síncrono.

Ejemplo 7.5

BORRADO SÍNCRONO En la figura 7.40 se muestra cómo describir un flip-flop D con una entrada de reset síncrono. En este caso la señal reset sólo actúa cuando llega un flanco positivo del reloj. El código genera el circuito de la figura 7.15, el cual tiene una compuerta AND conectada a la entrada D del flip-flop.

En la figura A.33*a* del apéndice A se muestra cómo el mismo circuito se especifica usando una instrucción IF en vez de WAIT UNTIL.

7.13 USO DE REGISTROS Y CONTADORES CON HERRAMIENTAS CAD

En esta sección mostraremos cómo incluir registros y contadores en circuitos diseñados con herramientas CAD. En los ejemplos expuestos se usa captura esquemática y código de VHDL.

7.13.1 INCLUSIÓN DE REGISTROS Y CONTADORES EN ESQUEMAS

En la sección 5.5.1 explicamos que un sistema CAD suele incluir bibliotecas de subcircuitos integrados. Presentamos la biblioteca de módulos parametrizados (LPM) y usamos el módulo su-

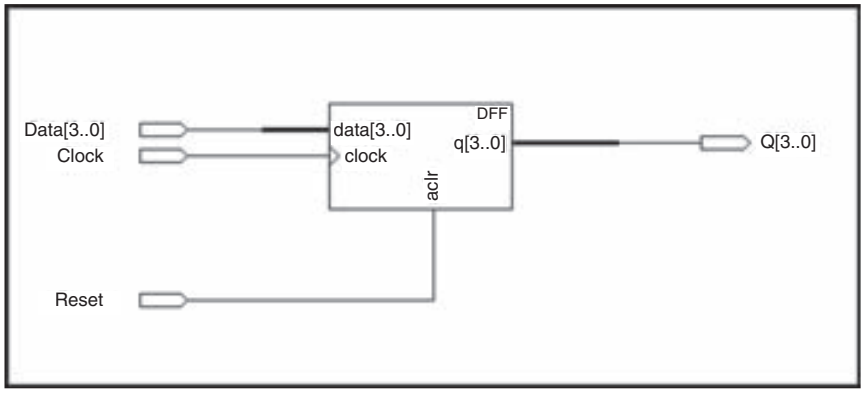


Figura 7.41 El módulo de flip-flop parametrizado *lpm_ff*.

mador/restador *lpm_add_sub* como ejemplo. La LPM incluye subcircuitos que tienen módulos que constituyen los flip-flops, registros, contadores y muchos otros circuitos útiles. En la figura 7.41 se muestra un símbolo que representa el módulo *lpm_ff*. Este módulo es un registro con uno o más flip-flops disparados por flanco positivo que pueden ser de tipo D o T. El módulo tiene parámetros que permiten elegir el número y tipo de flip-flops. En este caso decidimos tener cuatro flip-flops D. El tutorial del apéndice C explica cómo se realiza la configuración del módulo.

Las entradas *D* a los cuatro flip-flops, llamadas *data* en el símbolo gráfico, están conectadas a las cuatro señales de entrada de cuatro bits *Data*[3..0]. La entrada reset asíncrono (borrado) activa en nivel alto, *aclr*, se muestra en el esquema. Las salidas del flip-flop, *q*, están conectadas al símbolo de salida etiquetado como Q[3..0].

En la sección 7.3 dijimos que una aplicación útil de los flip-flops D consiste en almacenar los resultados de un cálculo aritmético, como la salida de un circuito sumador. Un ejemplo se muestra en la figura 7.42, donde se utilizan dos módulos de la LPM, *lpm_add_sub* y *lpm_ff*. El módulo *lpm_add_sub* se describió en la sección 5.5.1. Sus parámetros, no mostrados en la figura 7.42, están preparados para configurar el módulo como un circuito sumador de cuatro bits. La entrada de datos de cuatro bits *data* del sumador está manejada por la señal de entrada *Data*[3..0]. Los bits de suma, *result*, están conectados a las entradas de datos del *lpm_ff*, el cual está configurado como un registro *D* de cuatro bits con borrado asíncrono. El registro genera la salida del circuito, Q[3..0], que aparece en el lado izquierdo del esquema. Esta señal se realimenta hacia la entrada *datab* del sumador. Los bits de suma del sumador también se proporcionan como una salida del circuito, *Sum*[3..0], para facilidad de referencia en el análisis siguiente. Si el registro se borra primero y se establece en 0000, entonces el circuito puede emplearse para sumar los números binarios de la entrada *Data*[3..0] a una suma que está siendo acumulada en el registro, si un número nuevo se aplica a la entrada durante cada ciclo del reloj. El circuito que realiza esta función recibe el nombre de circuito *acumulador*.

Sintetizamos un circuito a partir del esquema e implementamos el sumador de cuatro bits usando la estructura de acarreo de adelanto. En la figura 7.43 aparece una simulación de tiempo para el circuito. Después de inicializar el circuito, la entrada *Data* se establece en 0001. El

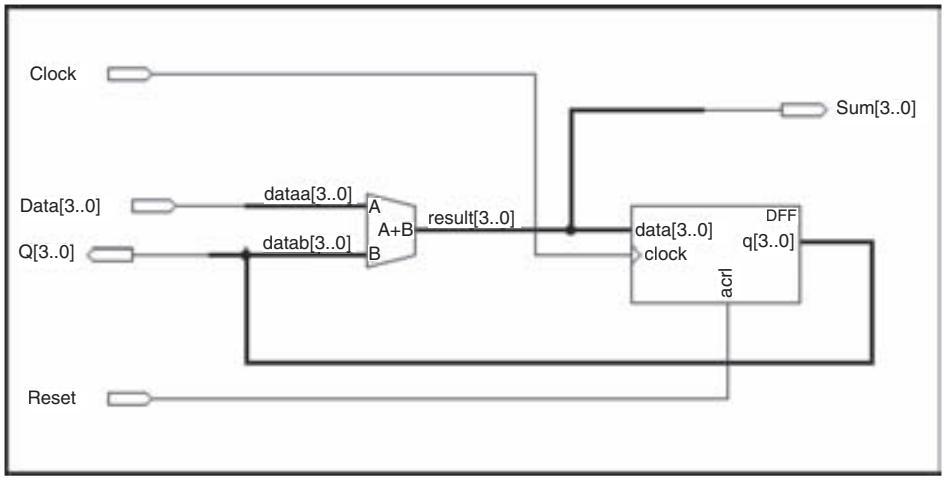


Figura 7.42 Un sumador con retroalimentación registrada.

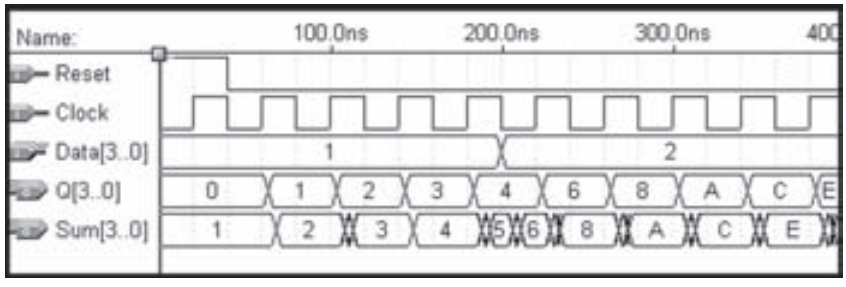


Figura 7.43 Simulación de tiempo del circuito de la figura 7.42.

sumador produce la suma 0000 + 0001 = 0001, la cual luego se sincroniza con el registro en el tiempo 60 ns. Después del retraso t_{co}, Q[3..0] se vuelve 0001, y esto hace que el sumador produzca la nueva suma 0001 + 0001 = 0010. El tiempo necesario para generar la suma nueva está determinado por la velocidad del circuito sumador, el cual produce la suma después de 12.5 ns en este caso. La suma nueva no aparece en la salida Q hasta después del siguiente flanco positivo del reloj, en 100 ns. El sumador produce después 0011 como la suma siguiente. Cuando *Sum* cambia de 0010 a 0011, aparecen algunas oscilaciones en el diagrama de tiempo, ocasionadas por la propagación de las señales de acarreo a través del circuito sumador. Estas oscilaciones no se ven en la salida Q, ya que *Sum* es estable en el momento que ocurre el siguiente flanco positivo del reloj. Al avanzar al tiempo 180 ns, *Sum* = 0100, y este valor se marca en el registro. El sumador produce la nueva suma 0101. Entonces, a los 200 ns, *Data* cambia a 0010, lo que causa que la

suma cambie a 0100 + 0010 = 0110. En el siguiente flanco positivo del reloj, Q se establece en 0110; el valor *Sum* = 0101 que estuvo presente temporalmente en el circuito no se observó en la salida Q. El circuito continúa sumando 0010 a la salida Q en cada flanco positivo del reloj.

Tras simular el comportamiento del circuito, debemos considerar si es posible concluir o no con cierta seguridad que el circuito funciona de manera adecuada. Idealmente, es prudente probar todas las combinaciones posibles de las entradas de un circuito antes de declarar que funciona como se desea. No obstante, en la práctica estas pruebas con frecuencia no son factibles debido al número de combinaciones de entrada que existen. Para el circuito de la figura 7.42 podríamos verificar que el sumador produjo una suma correcta y también que cada uno de los cuatro flip-flops en el registro almacenan ya sea 0 o 1 en forma apropiada. Comentaremos los aspectos relativos a la prueba de los circuitos en el capítulo 11.

Para que el circuito de la figura 7.42 trabaje bien deben cumplirse las restricciones de tiempo siguientes. Cuando el registro marque un flanco positivo del reloj, ha de propagarse un cambio en el valor de la señal en la salida del registro a través de la trayectoria de retroalimentación a la entrada *datab* del sumador. El sumador entonces produce una suma nueva, la cual debe propagarse a la entrada *data* del registro. Para el chip utilizado a fin de implementar el circuito, el retraso total incurrido es de 14 ns. El retraso puede dividirse como sigue: demora 2 ns desde que el registro se sincroniza hasta que un cambio en su salida llega a la entrada *datab* del sumador. El sumador produce una suma nueva en 8 ns, y requiere 4 ns para que se propague a la entrada *data* del registro. En la figura 7.43 el periodo del reloj es 40 ns. Por consiguiente, una vez que una suma nueva llega a la entrada *data* del registro, quedan 40 − 14 = 26 ns hasta que ocurra el siguiente flanco positivo del reloj. La entrada *data* debe estar estable para la cantidad de tiempo de preparación, t_{su} = 3 ns, antes del flanco del reloj. En consecuencia, tenemos 26 − 3 = 23 ns libres. El periodo del reloj puede disminuirse hasta 23 ns y el circuito aún seguirá trabajando. Pero si el periodo del reloj es menor que 40 − 23 = 17 ns, entonces el circuito no funcionará bien. Desde luego, si se usa un chip distinto para implementar el circuito, entonces se producirían resultados de sincronización diferentes. Los sistemas CAD proporcionan herramientas que pueden determinar automáticamente el periodo del reloj mínimo permitido durante el cual un circuito funcionará de manera correcta. El tutorial del apéndice C muestra cómo hacer esto usando las herramientas que acompañan al libro.

7.13.2 REGISTROS Y CONTADORES EN CÓDIGO DE VHDL

Los subcircuitos predefinidos en la biblioteca LPM pueden instanciarse en código de VHDL. En la figura 7.44 se instancia el módulo *lpm_shiftreg*, el cual es un registro de corrimiento de *n* bits. Los parámetros del módulo se fijan usando el constructor GENERIC MAP, como se muestra. El constructor GENERIC MAP es similar al constructor PORT MAP que se emplea para asignar nombres de señal a los puertos de un subcircuito. GENERIC MAP se utiliza para asignar valores a los parámetros del subcircuito. El número de flip-flops en el registro de corrimiento se establece en cuatro usando el parámetro LPM_WIDTH => 4. El módulo puede configurarse para desplazarse a la izquierda o a la derecha. El parámetro LPM_DIRECTION => RIGHT establece que la dirección de desplazamiento será de izquierda a derecha. El código utiliza la entrada de borrado asíncrona activa en nivel alto, *aclr*, del módulo, y la entrada de carga en paralelo activa en nivel alto *load*, la cual permite que el registro de corrimiento se cargue con los datos en paralelo en la entrada *data* del módulo. Cuando el corrimiento se lleva a cabo, el valor de la entrada

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
LIBRARY lpm ;
USE lpm.lpm_components.all ;

ENTITY shift IS
    PORT ( Clock          : IN   STD_LOGIC ;
           Reset          : IN   STD_LOGIC ;
           Shiftin, Load  : IN   STD_LOGIC ;
           R              : IN   STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           Q              : OUT  STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END shift ;

ARCHITECTURE Structure OF shift IS
BEGIN
    instance: lpm_shiftreg
        GENERIC MAP (LPM_WIDTH => 4, LPM_DIRECTION => "RIGHT")
        PORT MAP (data => R, clock => Clock, aclr => Reset,
            load => Load, shiftin => Shiftin, q => Q ) ;
END Structure ;
```

Figura 7.44 Instanciación del módulo *lpm_shiftreg*.

shiftin se desplaza en el flip-flop del extremo izquierdo y el bit corrido hacia afuera aparece en el bit del extremo derecho de la salida *q* en paralelo. El código usa la asociación mencionada, descrita en la sección 5.5.2, para conectar las señales de entrada y salida de la entidad *shift* a los puertos del módulo. Por ejemplo, la señal de entrada *R* se conecta al puerto *data* del módulo. Cuando se traslada a un circuito, el *lpm_shiftreg* tiene la estructura que se exhibe en la figura 7.19.

También hay módulos predefinidos para varios tipos de contadores, los cuales se necesitan comúnmente en los circuitos lógicos. Un ejemplo es el módulo *lpm_counter*, el cual es un contador de ancho variable con entradas de carga en paralelo.

7.13.3 USO DE INSTRUCCIONES SECUENCIALES DE VHDL PARA REGISTROS Y CONTADORES

En vez de instanciar los subcircuitos predefinidos para los registros, los registros de corrimiento, los contadores, etc., los circuitos pueden describirse en VHDL con instrucciones secuenciales. En la figura 7.39 se presenta el código para un flip-flop D. Una manera directa de describir un registro de *n* bits es mediante código jerárquico que incluya *n* instancias del subcircuito del flip-flop D. Un enfoque más simple se muestra en la figura 7.45. Utiliza el mismo código de la figura 7.39, excepto que la entrada D y la salida Q se definen como señales multibit. El código representa un registro de ocho bits con borrado asíncrono.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY reg8 IS
    PORT ( D              : IN   STD_LOGIC_VECTOR(7 DOWNTO 0) ;
           Resetn, Clock  : IN   STD_LOGIC ;
           Q              : OUT  STD_LOGIC_VECTOR(7 DOWNTO 0) ) ;
END reg8 ;

ARCHITECTURE Behavior OF reg8 IS
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
        IF Resetn = '0' THEN
            Q <= "00000000" ;
        ELSIF Clock'EVENT AND Clock = '1' THEN
            Q <= D ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Figura 7.45 Código para un registro de ocho bits con borrado asíncrono.

Ejemplo 7.6 **UN REGISTRO DE *N* BITS** Como los circuitos lógicos a menudo necesitan registros de diferentes tamaños, es conveniente definir una entidad de registro para la que el número de flip-flops pueda cambiar fácilmente. En la figura 7.46 se muestra cómo puede extenderse el código de la figura 7.45 para incluir un parámetro que establezca el número de flip-flops. El parámetro es un entero, *N*, que se define usando el constructor de VHDL llamado GENERIC. El valor de *N* se establece en 16 usando el operador de asignación :=. Al cambiar este parámetro, el código puede representar un registro de cualquier tamaño. Si el registro se declara como un componente, entonces sirve como subcircuito en otro código. Ese código puede usar el valor predeterminado para el parámetro GENERIC o especificar de alguna otra forma otro parámetro con el constructor GENERIC MAP. Un ejemplo que muestra cómo se usa GENERIC MAP aparece en la figura 7.44.

Las señales *D* y Q de la figura 7.46 están definidas en términos de *N*. La instrucción que inicializa todos los bits de Q en 0 usa la extraña sintaxis Q <= (OTHERS => '0'). Para el valor predeterminado de $N = 16$, esta instrucción equivale a la instrucción Q <= "0000000000000000". La sintaxis (OTHERS => '0') tiene como resultado un dígito '0' que se asigna a cada uno de los bits de Q, independientemente de cuántos bits tenga Q. Permite que el código se utilice para cualquier valor de *N*, en vez de sólo para $N = 16$.

Ejemplo 7.7 **UN REGISTRO DE CORRIMIENTO DE CUATRO BITS** Suponga que queremos escribir código de VHDL que represente el registro de corrimiento de cuatro bits de la figura 7.19. Un enfoque consiste en escribir código jerárquico que use cuatro subcircuitos. Cada subcircuito se compone de un flip-flop D con un multiplexor dos a uno conectado a la entrada *D*. En la figura 7.47 se define la entidad llamada *muxdff*, que representa este subcircuito. Los dos datos de en-

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY regn IS
    GENERIC ( N : INTEGER := 16 ) ;
    PORT ( D              : IN   STD_LOGIC_VECTOR(N−1 DOWNTO 0) ;
           Resetn, Clock  : IN   STD_LOGIC ;
           Q              : OUT  STD_LOGIC_VECTOR(N−1 DOWNTO 0) ) ;
END regn ;

ARCHITECTURE Behavior OF regn IS
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
        IF Resetn = '0' THEN
            Q <= (OTHERS => '0') ;
        ELSIF Clock'EVENT AND Clock = '1' THEN
            Q <= D ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Figura 7.46 Código para un registro de *n* bits con borrado asíncrono.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY muxdff IS
    PORT ( D0, D1, Sel, Clock  : IN   STD_LOGIC ;
           Q                   : OUT  STD_LOGIC ) ;
END muxdff ;

ARCHITECTURE Behavior OF muxdff IS
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL Clock'EVENT AND Clock = '1' ;
        IF Sel = '0' THEN
            Q <= D0 ;
        ELSE
            Q <= D1 ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Figura 7.47 Código para un flip-flop D con un multiplexor dos a uno en la entrada *D*.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY shift4 IS
    PORT ( R            : IN     STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           L, w, Clock  : IN     STD_LOGIC ;
           Q            : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END shift4 ;

ARCHITECTURE Structure OF shift4 IS
    COMPONENT muxdff
        PORT ( D0, D1, Sel, Clock  : IN  STD_LOGIC ;
               Q                   : OUT STD_LOGIC ) ;
    END COMPONENT ;
BEGIN
    Stage3: muxdff PORT MAP ( w, R(3), L, Clock, Q(3) ) ;
    Stage2: muxdff PORT MAP ( Q(3), R(2), L, Clock, Q(2) ) ;
    Stage1: muxdff PORT MAP ( Q(2), R(1), L, Clock, Q(1) ) ;
    Stage0: muxdff PORT MAP ( Q(1), R(0), L, Clock, Q(0) ) ;
END Structure ;
```

Figura 7.48 Código jerárquico para un registro de corrimiento de cuatro bits.

trada se llaman D_0 y D_1, y se seleccionan por medio de la entrada *Sel*. La instrucción del proceso (process) especifica que si $Sel = 0$ en el flanco positivo del reloj, entonces el valor de D_0 se asigna a Q; de lo contrario, se le asigna el valor de D_1.

En la figura 7.48 se define el registro de corrimiento de cuatro bits. La instrucción etiquetada *Stage*3 instancia el flip-flop del extremo izquierdo, el cual tiene la salida Q_3, y la instrucción *Stage*0 instancia el flip-flop del extremo derecho, Q_0. Cuando $L = 1$, se carga en paralelo desde la entrada *R*, y cuando $L = 0$, el corrimiento se realiza de izquierda a derecha. Los datos seriales se corren hacia el bit más importante, Q_3, desde la entrada *w*.

Ejemplo 7.8 **CÓDIGO OPCIONAL PARA UN REGISTRO DE CORRIMIENTO DE CUATRO BITS** Un estilo diferente de código para el registro de corrimiento de cuatro bits se muestra en la figura 7.49. Las líneas de código están numeradas para que sea más fácil remitirse a ellas. En vez de utilizar subcircuitos, el registro de corrimiento se describe mediante instrucciones secuenciales. Por la instrucción WAIT UNTIL de la línea 13, cualquier señal a la que se asigne un valor dentro del proceso debe implementarse como la salida de un flip-flop. Las líneas 14 y 15 especifican la carga en paralelo del registro de corrimiento cuando $L = 1$. La cláusula ELSE en las líneas 16 a 20 especifica la operación de corrimiento. La línea 17 desplaza el valor de Q_1 hacia el flip-flop con la salida Q_0. Las líneas 18 y 19 desplazan los valores de Q_2 y Q_3 hacia los flip-flops con las salidas Q_1 y Q_2, respectivamente. Por último, la línea 20 desplaza el valor de *w* hacia el flip-flop en el extremo izquierdo, el cual tiene la salida Q_3. Observe que la semántica del proceso, descrita en la sección 6.6.6, estipula que las cuatro tareas de las líneas 17 a 20 están programadas para ocurrir sólo después de que todas las instrucciones del proceso se han evaluado. Por consiguiente, los cuatro flip-flops cambian sus valores al mismo tiempo, según se requiere en el registro de corrimiento. El código genera el mismo circuito de registro de corrimiento que el de la figura 7.48.

```
1   LIBRARY ieee ;
2   USE ieee.std_logic_1164.all ;

3   ENTITY shift4 IS
4       PORT ( R      : IN      STD_LOGIC_VECTOR(3 DOWNTO 0) ;
5              Clock  : IN      STD_LOGIC ;
6              L, w   : IN      STD_LOGIC ;
7              Q      : BUFFER  STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
8   END shift4 ;

9   ARCHITECTURE Behavior OF shift4 IS
10  BEGIN
11      PROCESS
12      BEGIN
13          WAIT UNTIL Clock'EVENT AND Clock = '1' ;
14          IF L = '1' THEN
15              Q <= R ;
16          ELSE
17              Q(0) <= Q(1) ;
18              Q(1) <= Q(2);
19              Q(2) <= Q(3) ;
20              Q(3) <= w ;
21          END IF ;
22      END PROCESS ;
23  END Behavior ;
```

Figura 7.49 Código opcional para un registro de corrimiento.

Es importante considerar el efecto de invertir el orden de las líneas 17 a 20 de la figura 7.49, como se indica en la figura 7.50. En este caso la primera operación de corrimiento especificada en el código, en la línea 17, desplaza el valor de *w* hacia el flip-flop del extremo izquierdo con la salida Q_3. Debido a la semántica de la instrucción de proceso (process), la asignación a Q_3 no surte efecto hasta que se evalúan todas las instrucciones subsiguientes dentro del proceso. Por consiguiente, la línea 18 desplaza el valor presente de Q_3 antes de que cambie como resultado de la línea 17, hacia el flip-flop con la salida Q_2. De forma similar, las líneas 19 y 20 desplazan los valores presentes de Q_2 y Q_1 hacia los flip-flops con las salidas Q_1 y Q_0, respectivamente. El código produce el mismo circuito generado con el ordenamiento de las instrucciones de la figura 7.49.0

Ejemplo 7.9

REGISTRO DE CORRIMIENTO DE *N* BITS En la figura 7.51 se muestra el código que puede utilizarse para representar registros de corrimiento de cualquier tamaño. El parámetro *N* de GENERIC, que tiene el valor predeterminado de 8 en la figura, establece el número de flip-flops. El código es idéntico al de la figura 7.49 con dos excepciones. Primera, *R* y Q están definidas en términos de *N*. Segunda, la cláusula ELSE que describe la operación de corrimiento está generalizada para funcionar con cualquier número de flip-flops.

```
1   LIBRARY ieee ;
2   USE ieee.std_logic_1164.all ;

3   ENTITY shift4 IS
4       PORT ( R       : IN     STD_LOGIC_VECTOR(3 DOWNTO 0) ;
5              Clock   : IN     STD_LOGIC ;
6              L, w    : IN     STD_LOGIC ;
7              Q       : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
8   END shift4 ;

9   ARCHITECTURE Behavior OF shift4 IS
10  BEGIN
11      PROCESS
12      BEGIN
13          WAIT UNTIL Clock'EVENT AND Clock = '1' ;
14          IF L = '1' THEN
15              Q <= R ;
16          ELSE
17              Q(3) <= w ;
18              Q(2) <= Q(3) ;
19              Q(1) <= Q(2);
20              Q(0) <= Q(1) ;
21          END IF ;
22      END PROCESS ;
23  END Behavior ;
```

Figura 7.50 Código que invierte el orden de las instrucciones de la figura 7.49.

Las líneas 18 a 20 especifican la operación de corrimiento para los flip-flops $N - 1$ del extremo derecho, los cuales tienen las salidas Q_{N-2} a Q_0. El constructor usado se llama FOR LOOP. Es parecido a la instrucción FOR GENERATE presentada en la sección 6.6.4, que sirve para generar un conjunto de instrucciones concurrentes. FOR LOOP se utiliza para producir una serie de instrucciones secuenciales. La primera iteración del ciclo desplaza el valor presente de Q_1 hacia el flip-flop con la salida Q_0. La siguiente iteración desplaza Q_2 hacia el flip-flop con la salida Q_1, y así por el estilo, con la iteración final desplazando Q_{N-1} hacia el flip-flop con la salida Q_{N-2}. La línea 21 completa la operación de corrimiento al desplazar el valor de la entrada serial *w* hacia el flip-flop del extremo izquierdo con la salida Q_{N-1}.

Ejemplo 7.10 **CONTADOR ASCENDENTE** En la figura 7.52 se muestra el código para un contador ascendente de cuatro bits que tiene una entrada de inicialización, *Resetn*, y una entrada de habilitación, *E*. En el cuerpo de arquitectura los flip-flops del contador se representan mediante la señal llamada *Count*. La instrucción del proceso (process) especifica un reset asíncrono de *Count* cuando *Resetn* = 0. La cláusula ELSIF especifica que en el flanco positivo del reloj, si $E = 1$,

```
1   LIBRARY ieee ;
2   USE ieee.std_logic_1164.all ;

3   ENTITY shiftn IS
4       GENERIC ( N : INTEGER := 8 ) ;
5       PORT ( R       : IN      STD_LOGIC_VECTOR(N−1 DOWNTO 0) ;
6              Clock   : IN      STD_LOGIC ;
7              L, w    : IN      STD_LOGIC ;
8              Q       : BUFFER  STD_LOGIC_VECTOR(N−1 DOWNTO 0) ) ;
9   END shiftn ;

10  ARCHITECTURE Behavior OF shiftn IS
11  BEGIN
12      PROCESS
13      BEGIN
14          WAIT UNTIL Clock'EVENT AND Clock = '1' ;
15          IF L = '1' THEN
16              Q <= R ;
17          ELSE
18              Genbits: FOR i IN 0 TO N-2 LOOP
19                  Q(i) <= Q(i + 1) ;
20              END LOOP ;
21              Q(N-1) <= w ;
22          END IF ;
23      END PROCESS ;
24  END Behavior ;
```

Figura 7.51 Código para un registro de corrimiento de *n* bits, de izquierda a derecha.

el conteo se incrementa. Si $E = 0$, el código asigna explícitamente Count <= *Count*. Esta instrucción no se requiere para describir correctamente el contador debido a la semántica de la memoria implícita, pero podría incluirse en aras de la claridad. Las salidas Q se asignan al valor de *Count* al final del código. El código produce el circuito mostrado en la figura 7.23 si el compilador de VHDL opta por usar flip-flops T, y genera el circuito de la figura 7.24 (con la entrada reset añadida) si el compilador elige flip-flops D.

Ejemplo 7.11

USO DE SEÑALES INTEGER (ENTEROS) EN UN CONTADOR Los contadores con frecuencia se definen en VHDL usando el tipo INTEGER, que explicamos en la sección 5.5.4. El código de la figura 7.53 define el contador ascendente que tiene una entrada de carga en paralelo, además de una entrada reset. Los datos en paralelo, *R*, así como la salida del contador, Q, se definen mediante el tipo INTEGER. Puesto que están en el límite de 0 a 15, ambas señales representan cantidades de cuatro bits. En la figura 7.52 la señal *Count* se define para representar

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;

ENTITY upcount IS
    PORT ( Clock, Resetn, E  : IN   STD_LOGIC ;
           Q                 : OUT  STD_LOGIC_VECTOR (3 DOWNTO 0)) ;
END upcount ;

ARCHITECTURE Behavior OF upcount IS
    SIGNAL Count : STD_LOGIC_VECTOR (3 DOWNTO 0) ;
BEGIN
    PROCESS ( Clock, Resetn )
    BEGIN
        IF Resetn = '0' THEN
            Count <= "0000" ;
        ELSIF (Clock'EVENT AND Clock = '1') THEN
            IF E = '1' THEN
                Count <= Count + 1 ;
            ELSE
                Count <= Count ;
            END IF ;
        END IF ;
    END PROCESS ;
    Q <= Count ;
END Behavior ;
```

Figura 7.52 Código para un contador ascendente de cuatro bits.

los flip-flops del contador. Esta señal no es necesaria si las salidas Q están en modo BUFFER, como se muestra en la figura 7.53. La instrucción if-then-else al principio del proceso incluye el mismo reset asíncrono de la figura 7.53. La cláusula ELSIF especifica que en el flanco positivo del reloj, si $L = 1$, los flip-flops del contador se cargan en paralelo desde las entradas R. Si $L = 0$, el conteo se incrementa.

Ejemplo 7.12 **CONTADOR DESCENDENTE** En la figura 7.54 se muestra el código para un contador descendente llamado *downcnt*. Para facilitar el cambio del conteo inicial, se define como un parámetro GENERIC llamado *modulus*. En el flanco positivo del reloj, si $L = 1$ el contador se carga con los valores *modulus* – 1, y si $L = 0$, el conteo disminuye. El contador también incluye una entrada de habilitación, E. Establecer $E = 1$ permite que el conteo disminuya cuando ocurre un flanco activo del reloj.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY upcount IS
    PORT ( R                 : IN     INTEGER RANGE 0 TO 15 ;
           Clock, Resetn, L  : IN     STD_LOGIC ;
           Q                 : BUFFER INTEGER RANGE 0 TO 15 ) ;
END upcount ;

ARCHITECTURE Behavior OF upcount IS
BEGIN
    PROCESS ( Clock, Resetn )
    BEGIN
        IF Resetn = '0' THEN
            Q <= 0 ;
        ELSIF (Clock'EVENT AND Clock = '1') THEN
            IF L = '1' THEN
                Q <= R ;
            ELSE
                Q <= Q + 1 ;
            END IF;
        END IF;
    END PROCESS;
END Behavior;
```

Figura 7.53 Un contador de cuatro bits con carga en paralelo que usa señales INTEGER.

7.14 EJEMPLOS DE DISEÑO

En esta sección se presentan dos ejemplos de sistemas digitales que utilizan algunos bloques de construcción descritos en este capítulo y en el capítulo 6.

7.14.1 ESTRUCTURA DE BUS

Los sistemas digitales suelen contener un conjunto de registros empleados para almacenar datos. En la figura 7.55 se presenta un ejemplo de un sistema que tiene k registros de n bits, $R1$ a Rk. Cada registro se conecta a un conjunto común de n cables, los cuales se ocupan para transferir datos dentro y fuera de los registros. Este conjunto común de cables suele llamarse *bus*. Además de los registros, en un sistema real otros tipos de bloques de circuitos se conectarían al bus. En la figura se muestra cómo n bits de datos pueden colocarse en el bus desde otro bloque de circuito, usando la entrada de control *Extern*. Los datos almacenados en cualquiera de los registros pueden transferirse por el bus a un registro distinto o al bloque de circuitos conectado al bus.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY downcnt IS
    GENERIC ( modulus : INTEGER := 8 ) ;
    PORT ( Clock, L, E  : IN     STD_LOGIC ;
           Q            : OUT  INTEGER RANGE 0 TO modulus−1 ) ;
END downcnt ;

ARCHITECTURE Behavior OF downcnt IS
    SIGNAL Count : INTEGER RANGE 0 TO modulus−1 ;
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL (Clock'EVENT AND Clock = '1') ;
        IF L = '1' THEN
            Count <= modulus−1 ;
        ELSE
            IF E = '1' THEN
                Count <= Count−1 ;
            END IF ;
        END IF ;
    END PROCESS;
    Q <= Count ;
END Behavior ;
```

Figura 7.54 Código para un contador descendente.

Resulta esencial asegurar que sólo un bloque de circuito intenta colocar datos en los cables del bus en cierto instante. En la figura 7.55 cada registro está conectado al bus por medio de un buffer triestado de n bits. Se utiliza un circuito de control para asegurar que sólo una de las entradas enable del buffer triestado, $R1_{out}, \ldots, Rk_{out}$, se valide en cualquier momento dado. El circuito de control también produce las señales $R1_{in}, \ldots, Rk_{in}$, las cuales controlan cuándo se cargan los datos en cada registro. En general, el circuito de control podría realizar una serie de funciones, como la transferencia de los datos almacenados en un registro hacia otro y así por el estilo. En la figura 7.55 se muestra una señal de entrada llamada *Function* que instruye el circuito de control para que realice una tarea específica. El circuito de control está sincronizado por una entrada del reloj, que es la misma señal de reloj que controla los registros k.

En la figura 7.56 se presenta una vista más detallada de cómo pueden conectarse a un bus los registros de la figura 7.55. Para mantener la simplicidad se muestran dos registros de dos bits, pero el mismo esquema puede usarse para registros más grandes. Para el registro $R1$, dos buffers triestado habilitados por $R1_{out}$ se usan a fin de conectar la salida de cada flip-flop a un cable en el bus. La entrada D en cada flip-flop está conectada a un multiplexor dos a uno, cuya entrada select está controlada por $R1_{in}$. Si $R1_{in} = 0$, los flip-flops se cargan desde sus salidas Q; por consiguiente, los datos almacenados no cambian. Pero si $R1_{in} = 1$, los datos se cargan en los flip-flops desde el bus. En vez de utilizar multiplexores en las entradas de los flip-flops,

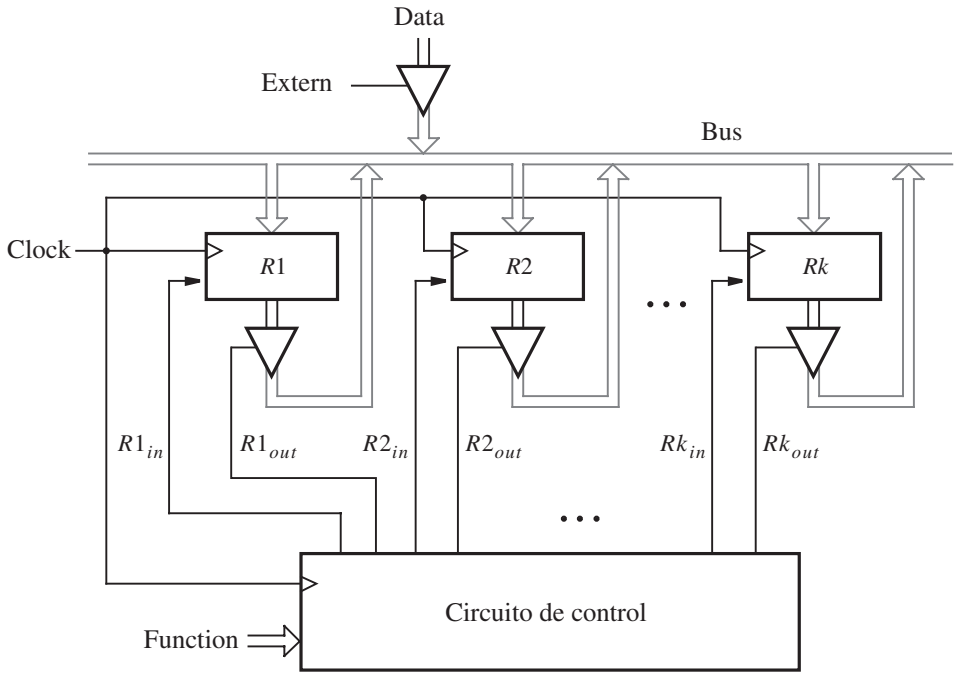


Figura 7.55 Un sistema digital con registros *k*.

podría intentarse conectar las entradas D en los flip-flops directamente al bus. Entonces es necesario controlar las entradas del reloj en todos los flip-flops para asegurar que están sincronizadas sólo cuando deben cargarse datos nuevos en el registro. Este enfoque no es conveniente porque puede ocurrir que diferentes flip-flops se sincronicen en tiempos ligeramente distintos, lo que causaría un problema conocido como *desviación del reloj (clock skew)*. Un análisis detallado de los problemas relacionados con la sincronización de los flip-flops se proporciona en la sección 10.3.

El sistema de la figura 7.55 puede utilizarse de muchas formas, según el diseño del circuito de control y de cuántos registros y otros bloques de circuitos estén conectados al bus. Como ejemplo simple considérese un sistema con tres registros, $R1$, $R2$ y $R3$. Cada uno de ellos está conectado al bus como se indicó en la figura 7.56. Diseñaremos un circuito de control que cumple una sola función: intercambia el contenido de los registros $R1$ y $R2$, y usa $R3$ para almacenamiento temporal.

El intercambio requerido se realiza en tres pasos, cada uno de los cuales necesita un ciclo del reloj. En el primer paso el contenido de $R2$ se transfiere a $R3$. Luego el contenido de $R1$ se transfiere a $R2$. Finalmente, el contenido de $R3$, que tiene el contenido original de $R2$, se transfiere a $R1$. Nótese que decimos que el contenido de un registro, R_i, se "transfiere" a otro registro, R_j. Esta terminología suele usarse para indicar que el nuevo contenido de R_j será una copia del contenido de R_i. El contenido de R_i no cambia como resultado de la transferencia. Por consiguiente, sería más preciso decir que el contenido de R_i se "copia" en R_j.

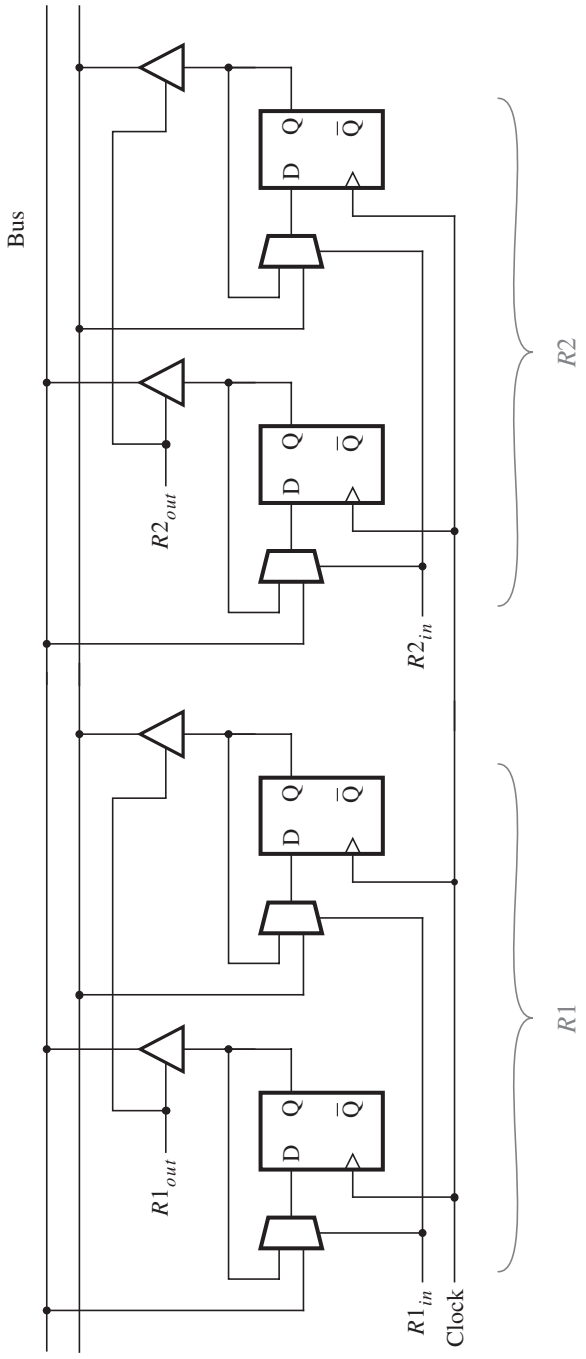


Figura 7.56 Detalles para conectar los registros a un bus.

Uso de un registro de corrimiento para control

Hay muchas formas de diseñar un circuito de control adecuado para la operación de intercambio. Una posibilidad es usar el registro de corrimiento de izquierda a derecha mostrado en la figura 7.57. Supóngase que la entrada reset se emplea para borrar los flip-flops y establecerlos en 0. Por tanto, las señales de control $R1_{in}$, Rl_{out} y demás no se validan, ya que las salidas del registro de corrimiento tienen el valor 0. La entrada serial w normalmente tiene el valor 0. Suponemos que los cambios en el valor de w están sincronizados para que ocurran poco tiempo después del flanco activo del reloj. Tal suposición es razonable porque comúnmente w se genera como la salida de algún circuito controlado por la misma señal de reloj. Cuando el intercambio buscado debe realizarse, w se establece en 1 para un ciclo del reloj, y luego regresa a 0. Después del siguiente flanco activo del reloj, la salida del flip-flop en el extremo izquierdo se vuelve igual a 1, lo cual valida tanto $R2_{out}$ como $R3_{in}$. El contenido del registro $R2$ se coloca en los cables del bus y se carga en un registro $R3$ en el siguiente flanco activo del reloj. Este flanco del reloj también desplaza el contenido del registro de corrimiento, con lo cual $R1_{out} = R2_{in} = 1$. Obsérvese que como w ahora es 0, el primer flip-flop se borra, lo que ocasiona que $R2_{out} = R3_{in} = 0$. El contenido de $R1$ ahora está en el bus y se carga en $R2$ en el siguiente flanco del reloj. Después de este flanco, el registro de corrimiento contiene 001 y, por ende, valida $R3_{out}$ y $R1_{in}$. El contenido de $R3$ ahora está en el bus y se carga en $R1$ en el siguiente flanco del reloj.

Al usar el circuito de control de la figura 7.57, cuando w cambia a 1 la operación de intercambio no comienza sino hasta después del siguiente flanco activo del reloj. Podemos modificar el circuito de control de modo que la operación de intercambio comience en el mismo ciclo del reloj en el que w cambia a 1. Un enfoque posible se ilustra en la figura 7.58. La señal reset se utiliza para establecer en 100 el contenido del registro de corrimiento, al prestablecer el flip-flop del extremo izquierdo en 1 y borrar los otros dos flip-flops. Siempre que $w = 0$, las señales de control de salida no se validan. Cuando w cambia a 1, las señales $R2_{out}$ y $R3_{in}$ se validan de inmediato y el contenido de $R2$ se coloca en el bus. El siguiente flanco activo del reloj carga estos datos en $R3$ y también desplaza el contenido del registro de corrimiento a 010. Puesto que la señal $R1_{out}$ ahora se valida, el contenido de $R1$ aparece en el bus. El siguiente flanco del reloj carga estos datos en $R2$ y cambia el contenido del registro de corrimiento a 001. El contenido de $R3$ ahora está en el bus; estos datos se cargan en $R1$ en el siguiente flanco del reloj, el cual también cambia el contenido del registro de corrimiento a 100. Suponemos que w tuvo el valor de 1 sólo para un ciclo del reloj; por consiguiente, las señales de control de la salida no se validan en este

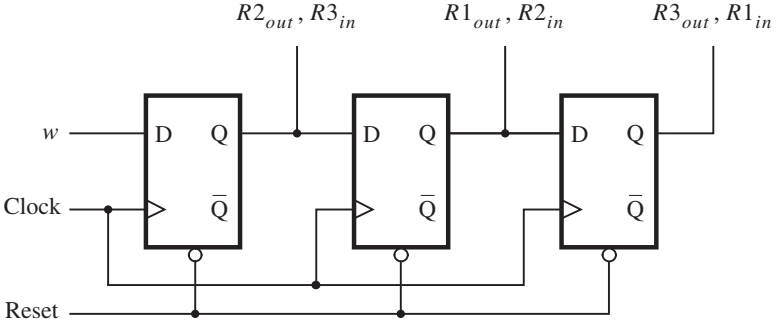


Figura 7.57 Un circuito de control del registro de corrimiento.

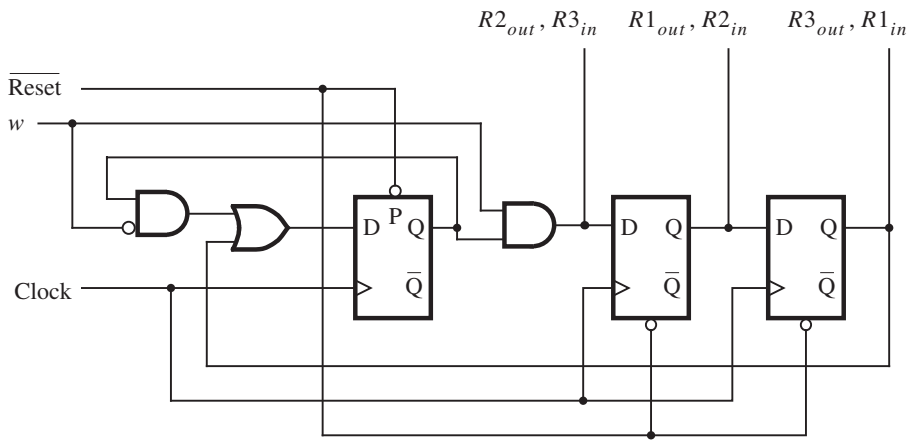


Figura 7.58 Un circuito de control modificado.

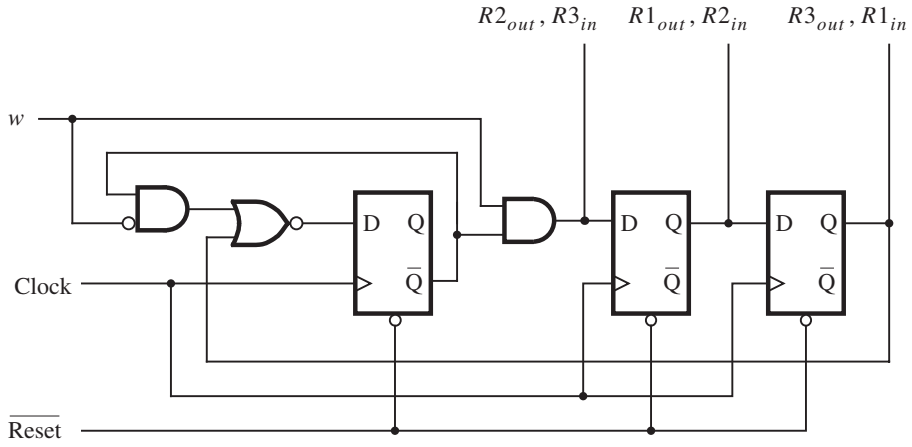


Figura 7.59 Una versión modificada del circuito de la figura 7.58.

punto. Tal vez no sea evidente para el lector cómo diseñar un circuito como el mostrado en la figura 7.58 debido a que hemos presentado el diseño según lo requiere el caso. En la sección 8.3 mostraremos cómo diseñar este circuito utilizando un enfoque más formal.

El circuito de la figura 7.58 supone que una entrada preset está disponible en el flip-flop del extremo izquierdo. Si el flip-flop tiene sólo una entrada clear, entonces podemos usar el circuito equivalente que aparece en la figura 7.59. En este circuito usamos la salida $\overline{Q}$ del flip-flop del extremo izquierdo y también complementamos la entrada a este flip-flop con una compuerta NOR en vez de una OR.

Uso de un multiplexor para implementar un bus

En la figura 7.55 usamos buffers triestado para controlar el acceso al bus. Un enfoque opcional consiste en utilizar multiplexores, como se describe en la figura 7.60. Las salidas de cada registro están conectadas a un multiplexor. La salida de este multiplexor está conectada a las entradas de los registros, con lo que se crea el bus. Las entradas select del multiplexor determinan el contenido de cuál registro aparece en el bus. Aun cuando la figura muestra sólo un símbolo de multiplexor, en realidad necesitamos un multiplexor por cada bit de los registros, $R1$ a $R4$, además de la entrada *Data* de ocho bits suministrada externamente. Para interconectarlas requerimos ocho multiplexores cinco a uno. En la figura 7.57 usamos un registro de corrimiento para implementar el circuito de control. Un enfoque similar puede utilizarse con los multiplexores. Las señales que controlan cuándo se cargan los datos en un registro, como $R1_{in}$, aún pueden conectarse directamente a las salidas del registro de corrimiento. Sin embargo, en vez de usar señales de control como $R1_{out}$ para colocar el contenido de un registro en el bus, hemos de generar las entradas select para los multiplexores. Una forma de hacerlo es conectar las salidas del registro de corrimiento a un circuito codificador que produce las entradas select para el multiplexor. Estudiamos los circuitos codificadores en la sección 6.3.

Los enfoques de buffer triestado y de multiplexor para implementar el bus son igualmente válidos. No obstante, ciertos tipos de chips, como el grueso de los PLD, no contienen un número suficiente de buffers triestado para producir buses incluso de tamaño mediano. En estos chips el enfoque del multiplexor es la única opción real. En la práctica, los circuitos se diseñan con herramientas CAD. Si el diseñador describe el circuito mediante buffers triestado pero no hay suficientes buffers en el dispositivo objetivo, entonces las herramientas CAD producen de manera automática un circuito equivalente que utiliza multiplexores.

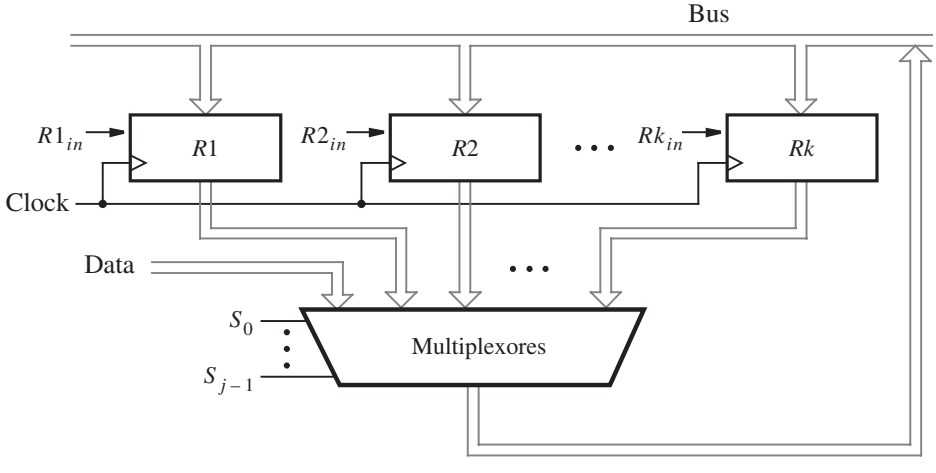


Figura 7.60 Uso de multiplexores para implementar un bus.

Código de VHDL

En esta sección se presenta el código de VHDL para nuestro circuito de ejemplo que intercambia el contenido de dos registros. Primero damos el código para el estilo de circuito de la figura 7.55 que utiliza buffers triestado para implementar el bus y luego ofrecemos el código para el estilo de circuito de la figura 7.60, que emplea multiplexores. El código está escrito de una manera jerárquica, usando subcircuitos para los registros, buffers triestado y el registro de corrimiento. En la figura 7.61 se presenta el código para un registro de *n* bits del tipo de la figura 7.56. El número de bits del registro se establece por medio del parámetro genérico *N*, el cual tiene el valor predeterminado de 8. El proceso que describe el registro especifica que la entrada *Rin* = 1, entonces los flip-flops se cargan desde la entrada *R* de *n* bits. De lo contrario, los flip-flops conservan sus valores almacenados en ese momento. El circuito sintetizado a partir de este código tiene un multiplexor dos a uno controlado por *Rin* conectado a la entrada *D* en cada flip-flop, como se describe en la figura 7.56.

En la figura 7.62 aparece el código para un subcircuito que representa *n* buffers triestado, cada uno habilitado por la entrada *E*. El número de buffers se establece mediante el parámetro genérico *N*. Las entradas a los buffers son la señal *X* de *n* bits y las salidas son la señal *F* de *n* bits. La arquitectura utiliza la sintaxis (OTHERS => 'Z') para especificar que la salida de cada buffer establece el valor de Z si *E* = 0; de lo contrario, la salida se establece como *F* = *X*.

En la figura 7.63 se proporciona el código de un registro de corrimiento que puede utilizarse para implementar el circuito de control de la figura 7.57. El número de flip-flops se establece por medio del parámetro genérico *K*, el cual tiene el valor predeterminado de 4. El registro de corrimiento tiene una entrada reset asíncrona activa en nivel bajo. La operación de corrimiento se define con un FOR LOOP en el estilo usado en el ejemplo 7.9.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY regn IS
    GENERIC ( N : INTEGER := 8 ) ;
    PORT ( R          : IN   STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
           Rin, Clock : IN   STD_LOGIC ;
           Q          : OUT  STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
END regn ;

ARCHITECTURE Behavior OF regn IS
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL Clock'EVENT AND Clock = '1' ;
        IF Rin = '1' THEN
            Q <= R ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Figura 7.61 Código para un registro de *n* bits del tipo de la figura 7.56.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY trin IS
    GENERIC ( N : INTEGER := 8 ) ;
    PORT ( X  : IN   STD_LOGIC_VECTOR(N−1 DOWNTO 0) ;
           E  : IN   STD_LOGIC ;
           F  : OUT  STD_LOGIC_VECTOR(N−1 DOWNTO 0) ) ;
END trin ;

ARCHITECTURE Behavior OF trin IS
BEGIN
    F <= (OTHERS => 'Z') WHEN E = '0' ELSE X ;
END Behavior ;
```

Figura 7.62 Código para un buffer triestado de *n* bits.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY shiftr IS - - registro de corrimiento de izquierda a derecha con reset asíncrono
    GENERIC ( K : INTEGER := 4 ) ;
    PORT ( Resetn, Clock, w  : IN          STD_LOGIC ;
           Q                 : BUFFER  STD_LOGIC_VECTOR(1 TO K) ) ;
END shiftr ;

ARCHITECTURE Behavior OF shiftr IS
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
        IF Resetn = '0' THEN
            Q <= (OTHERS => '0') ;
        ELSIF Clock'EVENT AND Clock = '1' THEN
            Genbits: FOR i IN K DOWNTO 2 LOOP
                Q(i) <= Q(i−1) ;
            END LOOP ;
            Q(1) <= w ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Figura 7.63 Código para el registro de corrimiento de la figura 7.57.

Para utilizar como subcircuitos las entidades de las figuras 7.61 a 7.63 debemos proporcionar dos declaraciones de componentes por cada una. Por comodidad, las colocamos dentro de un solo paquete llamado *components*, el cual se muestra en la figura 7.64. Este paquete se utiliza en el código de la figura 7.65. Representa el sistema digital de la figura 7.55 con tres registros de ocho bits, *R*1, *R*2 y *R*3.

El circuito de la figura 7.55 incluye buffers triestado que se ocupan para colocar *n* bits de datos suministrados externamente en el bus. En el código de la figura 7.65, estos buffers se instancian en la instrucción etiquetada *tri_ext*. Cada uno de los ocho buffers se habilita mediante la señal de entrada *Extern*, y las entradas de datos en los buffers se conectan a la señal *Data* de ocho bits. Cuando *Extern* = 1, el valor de *Data* se coloca en el bus, que está representado por la señal *BusWires*. El puerto *BusWires* representa la salida del circuito. Este puerto tiene el modo INOUT, que es necesario porque *BusWires* está conectado a las salidas de los buffers triestado, los cuales a su vez están conectados a las entradas de los registros.

Suponemos que existe una señal de control de tres bits llamada *RinExt*, la cual se utiliza para permitir que los datos suministrados externamente se carguen desde el bus en los registros *R*1,

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

PACKAGE components IS

    COMPONENT regn - - registro
        GENERIC ( N : INTEGER := 8 ) ;
        PORT ( R            : IN    STD_LOGIC_VECTOR(N−1 DOWNTO 0) ;
               Rin, Clock   : IN    STD_LOGIC ;
               Q            : OUT   STD_LOGIC_VECTOR(N−1 DOWNTO 0) ) ;
    END COMPONENT ;

    COMPONENT shiftr - - registro de corrimiento de izquierda a derecha con reset asíncrono
        GENERIC ( K : INTEGER := 4 ) ;
        PORT ( Resetn, Clock, w  : IN          STD_LOGIC ;
               Q                 : BUFFER  STD_LOGIC_VECTOR(1 TO K) ) ;
    END component ;

    COMPONENT trin - - buffers triestado
        GENERIC ( N : INTEGER := 8 ) ;
        PORT ( X   : IN    STD_LOGIC_VECTOR(N−1 DOWNTO 0) ;
               E   : IN    STD_LOGIC ;
               F   : OUT   STD_LOGIC_VECTOR(N−1 DOWNTO 0) ) ;
    END COMPONENT ;

END components ;
```

Figura 7.64 Paquete y declaraciones de componentes.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE work.components.all ;

ENTITY swap IS
    PORT ( Data          : IN    STD_LOGIC_VECTOR(7 DOWNTO 0) ;
           Resetn, w     : IN    STD_LOGIC ;
           Clock, Extern : IN    STD_LOGIC ;
           RinExt        : IN    STD_LOGIC_VECTOR(1 TO 3) ;
           BusWires      : INOUT STD_LOGIC_VECTOR(7 DOWNTO 0) ) ;
END swap ;

ARCHITECTURE Behavior OF swap IS
    SIGNAL Rin, Rout, Q : STD_LOGIC_VECTOR(1 TO 3) ;
    SIGNAL R1, R2, R3 : STD_LOGIC_VECTOR(7 DOWNTO 0) ;
BEGIN
    control: shiftr GENERIC MAP ( K => 3 )
                    PORT MAP ( Resetn, Clock, w, Q ) ;
    Rin(1) <= RinExt(1) OR Q(3) ;
    Rin(2) <= RinExt(2) OR Q(2) ;
    Rin(3) <= RinExt(3) OR Q(1) ;
    Rout(1) <= Q(2) ; Rout(2) <= Q(1) ; Rout(3) <= Q(3) ;

    tri_ext: trin PORT MAP ( Data, Extern, BusWires ) ;
    reg1: regn PORT MAP ( BusWires, Rin(1), Clock, R1 ) ;
    reg2: regn PORT MAP ( BusWires, Rin(2), Clock, R2 ) ;
    reg3: regn PORT MAP ( BusWires, Rin(3), Clock, R3 ) ;
    tri1: trin PORT MAP ( R1, Rout(1), BusWires ) ;
    tri2: trin PORT MAP ( R2, Rout(2), BusWires ) ;
    tri3: trin PORT MAP ( R3, Rout(3), BusWires ) ;
END Behavior ;
```

Figura 7.65 Un sistema digital como el de la figura 7.55.

*R*2 o *R*3. La entrada *RinExt* no se muestra en la figura 7.55 para mantener la simplicidad de ésta, pero se generaría por el mismo bloque de circuito externo que produce *Extern* y *Data*. Cuando *RinExt*(1) = 1, los datos en el bus se cargan en el registro *R*1; cuando *RinExt*(2) = 1, los datos se cargan en *R*2 y cuando *RinExt*(3) = 1, los datos se cargan en *R*3.

En la figura 7.65 el registro de corrimiento de tres bits se instancia en la instrucción etiquetada *control*. Las salidas del registro de corrimiento son la señal Q de tres bits. Las tres instrucciones siguientes conectan Q a las señales de control que determinan cuándo se cargan los datos en cada registro, los cuales están representados por la señal *Rin* de tres bits. Las señales *Rin*(l), *Rin*(2) y *Rin*(3) en el código corresponden a las señales $R1_{in}$, $R2_{in}$ y $R3_{in}$ de la figura 7.55. Como se especificó en la figura 7.57, la salida del registro de corrimiento del extremo izquierdo, Q(1), controla

cuándo se cargan los datos en el registro *R*3. De modo similar, Q(2) controla el registro *R*2 y Q(3) controla *R*1. A cada bit en *Rin* se le suma (OR) correspondiente en *RinExt*, de modo que los datos suministrados externamente puedan almacenarse en los registros, como se vio antes. El código también conecta las salidas del registro de corrimiento para habilitar las entradas, llamadas *Rout*, en los buffers triestado que conectan los registros con el bus. En la figura 7.57 se muestra que Q(1) se usa para poner el contenido de *R*2 en el bus; por consiguiente, el valor de Q(l) se asigna a *Rout*(2). De manera similar, el valor de Q(2) se asigna a *Rout*(1) y el valor de Q(3), a *Rout*(3). Las instrucciones restantes del código instancian los registros y los buffers triestado del sistema.

Código de VHDL que utiliza multiplexores

En la figura 7.66 se indica cómo el código de la figura 7.65 puede modificarse para usar multiplexores en vez de buffers triestado. Al usar la estructura del circuito mostrada en la figura 7.60, el bus se implementa utilizando ocho multiplexores cuatro a uno. Tres de las entradas de datos de cada multiplexor cuatro a uno se conectan a un bit desde los registros *R*1, *R*2 y *R*3. La cuarta entrada de datos se conecta a un bit de la señal de entrada *Data* para permitir que los datos suministrados externamente se escriban en los registros. Cuando el contenido del registro de corrimiento es 000, los multiplexores seleccionan *Data* para colocarla en el bus. Estos datos se cargan en el registro seleccionado por *RinExt*. Se cargan en *R*1 si *RinExt*(1) = 1, en *R*2 si *RinExt*(2) = 1 y en *R*3 si *RinExt*(3) = 1.

La señal *Rout* de la figura 7.65, la cual se utiliza como la entrada enable del buffer triestado conectado al bus, no se necesita para la implementación del multiplexor. En vez de ello, deben proporcionarse las entradas select en los multiplexores. En el cuerpo de arquitectura de la figura 7.66 las salidas del registro de corrimiento se llaman Q. Estas señales se utilizan para generar las señales de control *Rin* para los registros de la misma manera que se muestra en la figura 7.65. En el análisis referente a la figura 7.60 dijimos que se necesita un codificador entre las salidas del registro de corrimiento y las salidas select del multiplexor. Un codificador adecuado se describe en la asignación de la señal seleccionada etiquetada *encoder*. Produce las entradas select del multiplexor, las cuales se llaman *S*. Establece *S* = 00 cuando el registro de corrimiento contiene 000, *S* = 10 cuando el registro de corrimiento contiene 100 y así por el estilo, como se muestra en el código. Los multiplexores se describen mediante la asignación de la señal seleccionada etiquetada *muxes*. Esta instrucción coloca el valor de *Data* en el bus (*BusWires*) si *S* = 00, el contenido del registro *R*1 si *S* = 01, y así sucesivamente. Al usar este esquema, cuando la operación de intercambio no está activa, los multiplexores colocan los bits desde la entrada *Data* en el bus.

En la figura 7.66 empleamos dos asignaciones de señal seleccionada, una para describir un codificador y la otra para describir los multiplexores de bus. Un enfoque más sencillo consiste en utilizar una sola asignación de señal, como se muestra en la figura 7.67. La instrucción etiquetada *muxes* especifica cuál señal debe aparecer en *BusWires* para cada patrón de las salidas del registro de corrimiento. El circuito sintetizado a partir de esta instrucción es similar a un multiplexor ocho a uno con las tres entradas select conectadas a las salidas del registro de corrimiento. No obstante, en realidad sólo la mitad del circuito multiplexor se genera por medio de las herramientas de síntesis debido a que sólo hay cuatro entradas de datos. El circuito producido a partir del código de la figura 7.67 es el mismo que el generado por el de la figura 7.66.

En la figura 7.68 se muestra un ejemplo de una simulación de tiempo para un circuito sintetizado con base en el código de la figura 7.67. En la primera mitad de la simulación el circuito se inicializa, lo mismo que el contenido de los registros *R*1 y *R*2. El valor hexadecimal 55 se carga en *R*1 y el valor AA se carga en *R*2. El flanco del reloj en 275 ns, marcado por la línea de referencia vertical de la figura 7.68, carga el valor de *w* = 1 en el registro de corrimiento. El

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE work.components.all ;
ENTITY swapmux IS
    PORT ( Data      : IN     STD_LOGIC_VECTOR(7 DOWNTO 0) ;
           Resetn, w : IN     STD_LOGIC ;
           Clock     : IN     STD_LOGIC ;
           RinExt    : IN     STD_LOGIC_VECTOR(1 TO 3) ;
           BusWires  : BUFFER STD_LOGIC_VECTOR(7 DOWNTO 0) ) ;
END swapmux ;

ARCHITECTURE Behavior OF swapmux IS
    SIGNAL Rin, Q : STD_LOGIC_VECTOR(1 TO 3) ;
    SIGNAL S : STD_LOGIC_VECTOR(1 DOWNTO 0) ;
    SIGNAL R1, R2, R3 : STD_LOGIC_VECTOR(7 DOWNTO 0) ;
BEGIN
    control: shiftr GENERIC MAP ( K => 3 )
                    PORT MAP ( Resetn, Clock, w, Q ) ;
    Rin(1) <= RinExt(1) OR Q(3) ;
    Rin(2) <= RinExt(2) OR Q(2) ;
    Rin(3) <= RinExt(3) OR Q(1) ;

    reg1: regn PORT MAP ( BusWires, Rin(1), Clock, R1 ) ;
    reg2: regn PORT MAP ( BusWires, Rin(2), Clock, R2 ) ;
    reg3: regn PORT MAP ( BusWires, Rin(3), Clock, R3 ) ;
    encoder:
    WITH Q SELECT
        S <= "00" WHEN "000",
             "10" WHEN "100",
             "01" WHEN "010",
             "11" WHEN OTHERS;
    muxes: --ocho multiplexores cuatro a uno
    WITH S SELECT
        BusWires <= Data WHEN "00",
                    R1 WHEN "01",
                    R2 WHEN "10",
                    R3 WHEN OTHERS ;
END Behavior ;
```

Figura 7.66 Uso de multiplexores para implementar un bus.

```
ARCHITECTURE Behavior OF swapmux IS
    SIGNAL Rin, Q : STD_LOGIC_VECTOR(1 TO 3) ;
    SIGNAL R1, R2, R3 : STD_LOGIC_VECTOR(7 DOWNTO 0) ;
BEGIN
    control: shiftr GENERIC MAP ( K => 3 )
                PORT MAP ( Resetn, Clock, w, Q ) ;
    Rin(1) <= RinExt(1) OR Q(3) ;
    Rin(2) <= RinExt(2) OR Q(2) ;
    Rin(3) <= RinExt(3) OR Q(1) ;

    reg1: regn PORT MAP ( BusWires, Rin(1), Clock, R1 ) ;
    reg2: regn PORT MAP ( BusWires, Rin(2), Clock, R2 ) ;
    reg3: regn PORT MAP ( BusWires, Rin(3), Clock, R3 ) ;

    muxes:
    WITH Q SELECT
        BusWires <= Data WHEN "000",
                    R2 WHEN "100",
                    R1 WHEN "010",
                    R3 WHEN OTHERS ;
END Behavior ;
```

Figura 7.67 Una versión simplificada de la arquitectura de la figura 7.66.

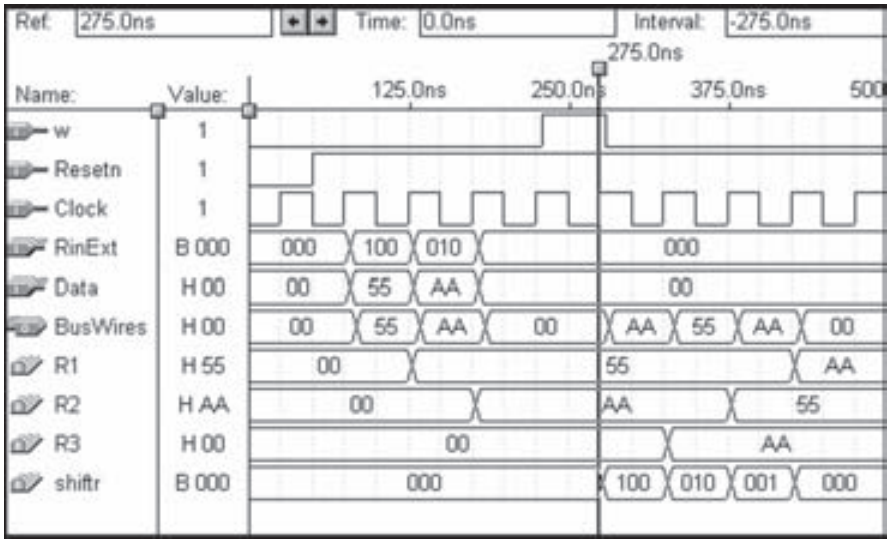


Figura 7.68 Simulación de tiempo para el código de VHDL de la figura 7.67.

contenido de *R*2 (AA) aparece entonces en el bus y se carga en *R*3 cuando el flanco del reloj está en 325 ns. Después de este flanco, el contenido del registro de corrimiento es 010 y los datos almacenados en *R*1 (55) están en el bus. El flanco del reloj en 375 ns carga estos datos en *R*2 y cambia el registro de corrimiento a 001. El contenido de *R*3(AA) ahora aparece en el bus y se carga en *R*1 cuando el flanco del reloj está en 425 ns. El registro de corrimiento está ahora en el estado 000 y el intercambio está completo.

7.14.2 PROCESADOR SIMPLE

Un segundo ejemplo de un sistema digital como el de la figura 7.55 se muestra en la figura 7.69. Tiene cuatro registros de *n* bits, *R*0,..., *R*3, que están conectados al bus mediante buffers triestado. Los datos externos pueden cargarse en los registros desde la entrada *Data* de *n* bits, la cual se conecta al bus por buffers triestado habilitados por medio de la señal de control *Extern*. El sistema también incluye un módulo sumador/restador. Una de sus entradas de datos es provista por un registro de *n* bits, *A*, que está conectado al bus, mientras que la otra entrada de datos, *B*, está directamente conectada al bus. Si la señal *AddSub* tiene el valor 0, el módulo genera la suma $A + B$; si $AddSub = 1$, el módulo genera la diferencia $A - B$. Para realizar la sustracción, suponemos que el sumador/restador incluye las compuertas XOR requeridas para formar el complemento a 2 de *B*, como se vio en la sección 5.3. El registro *G* almacena la salida producida por el sumador/restador. Los registros *A* y *G* están controlados por las señales A_{in}, G_{in} y G_{out}.

El sistema de la figura 7.69 puede realizar varias funciones, según el diseño del circuito de control. Como ejemplo, diseñaremos un circuito de control que pueda cumplir las cuatro acciones indicadas en la tabla 7.2. La columna izquierda muestra el nombre de cada operación y sus operandos; la derecha indica la función realizada en la operación. Para la operación *Load* el significado de $Rx \leftarrow Data$ es que los datos de la entrada *Data* externa se transfieren a través del bus a cualquier registro, *Rx*, donde *Rx* puede ser de *R*0 a *R*3. La operación *Move* copia los datos almacenados en el registro *Ry* en el registro *Rx*. En la tabla, los corchetes, como en [*Rx*], se refieren al *contenido* de un registro. Puesto que sólo se precisa una transferencia a través del bus, las operaciones *Load* y *Move* requieren sólo un paso (ciclo del reloj) para completarse. Las operaciones *Add* y *Sub* necesitan tres pasos, como sigue: en el primero el contenido de *Rx* se transfiere a través del bus en un registro *A*. Luego, en el paso siguiente el contenido de *Ry* se coloca en el bus. El módulo sumador/restador realiza la función requerida y los resultados se guardan en el registro *G*. Por último, en el tercer paso el contenido de *G* se transfiere a *Rx*.

Un sistema digital que realiza los tipos de operaciones señalados en la tabla 7.2 se conoce como *procesador*. La operación específica que va a llevarse a cabo en un momento dado se indica mediante la entrada del circuito de control llamada *Function*. La operación se inicia al establecer *w* en 1, y el circuito de control valida la salida *Done* cuando la operación se completa.

En la figura 7.55 usamos un registro de corrimiento para implementar el circuito de control. Es posible emplear un diseño similar para el sistema de la figura 7.69. A fin de ilustrar un enfoque distinto basaremos el diseño del circuito de control en un contador. Este circuito debe generar las señales de control requeridas en cada paso de cada operación. Como las operaciones más largas (*Add* y *Sub*) necesitan tres pasos (ciclos del reloj), puede usarse un contador de dos bits. En la figura 7.70 se muestra un contador de dos bits conectado a un decodificador dos a cuatro. Los decodificadores se estudiaron en la sección 6.2. El decodificador se habilita en todo momento

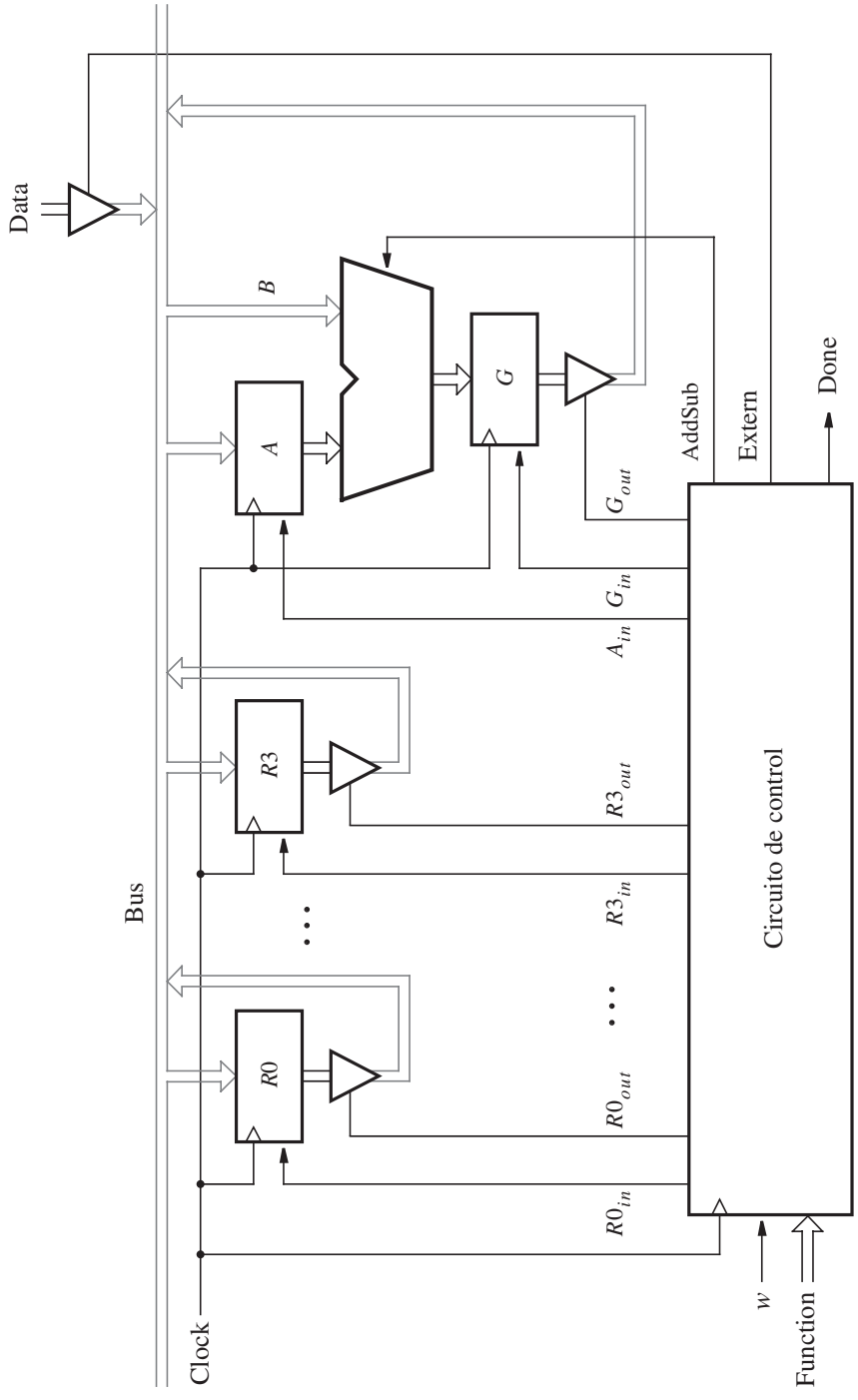


Figura 7.69 Sistema digital que implementa un procesador simple.

Tabla 7.2 Operaciones realizadas en el procesador.

Operación	Función realizada
Load *Rx*, *Data*	$Rx \leftarrow Data$
Move *Rx*, *Ry*	$Rx \leftarrow [Ry]$
Add *Rx*, *Ry*	$Rx \leftarrow [Rx] + [Ry]$
Sub *Rx*, *Ry*	$Rx \leftarrow [Rx] - [Ry]$

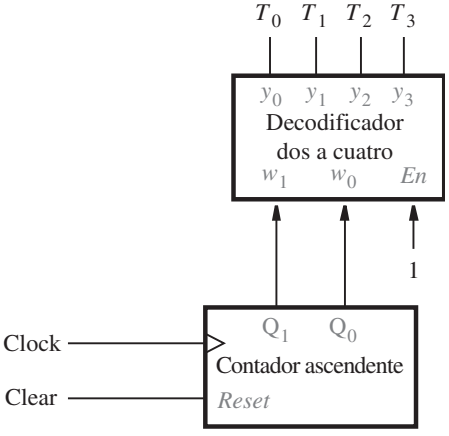


Figura 7.70 Parte de un circuito de control para el procesador.

al establecer de modo permanente su entrada de habilitación (*En*) en el valor 1. Cada una de las salidas del decodificador representa un paso de una operación. Cuando ninguna operación se está realizando en un momento dado, el valor de conteo es 00; por tanto, la salida T_0 del decodificador se valida. En el primer paso de una operación, el valor de conteo es 01 y T_1 se valida. Durante el segundo y tercer pasos de las operaciones *Add* y *Sub*, T_2 y T_3 se validan, respectivamente.

En cada uno de los pasos T_0 a T_3, varios valores de la señal de control deben generarse por medio del circuito de control, según la operación que vaya a realizarse. En la figura 7.71 se muestra que la operación se especifica con seis bits, los cuales forman la entrada *Function*. Los dos bits del extremo izquierdo, $F = f_1 f_0$ se usan como un número de dos bits que identifica la operación. Para representar *Load*, *Move*, *Add* y *Sub* usamos los códigos $f_1 f_0 = 00, 01, 10$ y 11, respectivamente. Las entradas $Rx_1 Rx_0$ son un número binario que identifica al operando *Rx*, mientras que $Ry_1 Ry_0$ identifica al operando *Ry*. Las entradas *Function* se almacenan en un registro de funciones de seis bits cuando la señal FR_{in} se valida.

En la figura 7.71 también se muestran tres decodificadores dos a cuatro que sirven para decodificar la información codificada en las entradas *F*, *Rx* y *Ry*. En breve veremos que estos

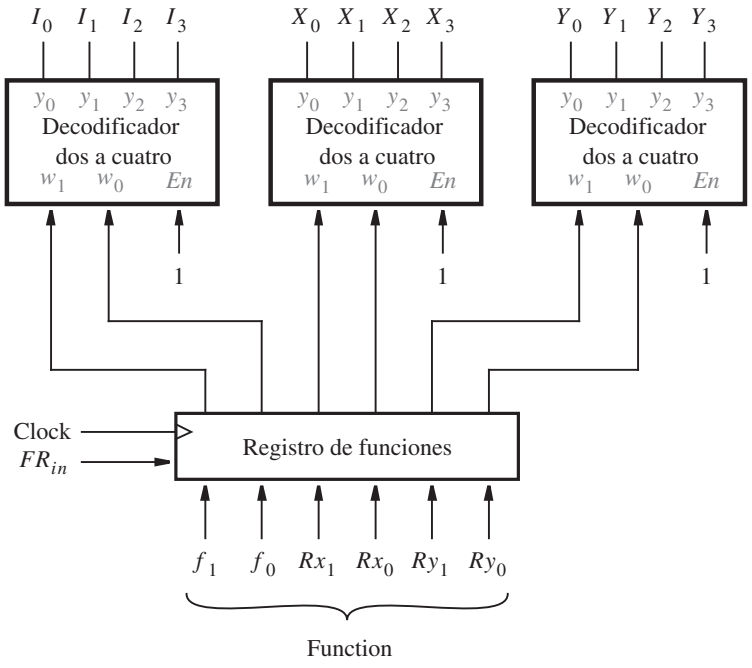


Figura 7.71 El registro de funciones y los decodificadores.

decodificadores se incluyen por conveniencia debido a que sus salidas proporcionan expresiones lógicas de apariencia sencilla para las diversas señales de control.

Los circuitos de las figuras 7.70 y 7.71 forman una parte del circuito de control. Mostraremos cómo derivar el resto del circuito de control usando la entrada w y las señales T_0,..., T_3, I_0,..., I_3, X_0,..., X_3 y Y_0,..., Y_3. Debe generar las salidas *Extern*, *Done*, A_{in}, G_{in}, G_{out}, *AddSub*, $R0_{in}$,..., $R3_{in}$ y $R0_{out}$,..., $R3_{out}$. El circuito de control también debe generar las señales *Clear* y FR_{in} utilizadas en las figuras 7.70 y 7.71.

Clear y FR_{in} están definidas de la misma manera para todas las operaciones. *Clear* se usa para asegurar que el valor de conteo permanezca en 00 siempre que $w = 0$ y ninguna operación se esté ejecutando. También sirve para reiniciar el valor de conteo a 00 al final de cada operación. Por tanto, una expresión lógica apropiada es

$$Clear = \overline{w}\, T_0 + Done$$

La señal FR_{in} se utiliza para cargar los valores de las entradas *Function* en el registro de funciones cuando w cambia a 1. Por consiguiente,

$$FR_{in} = wT_0$$

El resto de las salidas del circuito de control depende del paso específico que vaya a efectuarse en cada operación. Los valores que deben generarse para cada señal se muestran en la tabla 7.3. Cada fila de la tabla corresponde a una operación específica, y cada columna representa un paso

Tabla 7.3 Señales de control validadas en cada operación/paso de tiempo

	T_1	T_2	T_3
(Load): I_0	*Extern*, $R_{in} = X$, *Done*		
(Move): I_1	$R_{in} = X$, $R_{out} = Y$, *Done*		
(Add): I_2	$R_{out} = X$, A_{in}	$R_{out} = Y$, G_{in} *AddSub* = 0	G_{out}, $R_{in} = X$, *Done*
(Sub): I_3	$R_{out} = X$, A_{in}	$R_{out} = Y$, G_{in} *AddSub* = 1	G_{out}, $R_{in} = X$, *Done*

de tiempo. La señal *Extern* se valida sólo en el primer paso de la operación *Load*. Por consiguiente, la expresión lógica que implementa esta señal es

$$Extern = I_0 T_1$$

Done se valida en el primer paso de *Load* y *Move*, así como en el tercer paso de *Add* y *Sub*. Por tanto

$$Done = (I_0 + I_1)T_1 + (I_2 + I_3)T_3$$

Las señales A_{in}, G_{in} y G_{out} se validan en las operaciones *Add* y *Sub*. A_{in} se valida en el paso T_1; G_{in} en T_2, y G_{out} en T_3. La señal *AddSub* debe establecerse en 0 en la operación *Add* y en 1 en la operación *Sub*. Ello se logra con las expresiones lógicas siguientes

$$A_{in} = (I_2 + I_3)T_1$$
$$G_{in} = (I_2 + I_3)T_2$$
$$G_{out} = (I_2 + I_3)T_3$$
$$AddSub = I_3$$

Los valores de $R0_{in}$,..., $R3_{in}$ se determinan utilizando ya sea las señales X_0,..., X_3 o las señales Y_0,..., Y_3. En la tabla 7.3 estas acciones se indican escribiendo $R_{in} = X$ o $R_{in} = Y$. El significado de $R_{in} = X$ es que $R0_{in} = X_0$, $R1_{in} = X_1$ y así sucesivamente. De modo similar, los valores de $R0_{out}$,..., $R3_{out}$ se especifican utilizando ya sea $R_{out} = X$ o $R_{out} = Y$.

Desarrollaremos las expresiones para $R0_{in}$ y $R0_{out}$ al examinar la tabla 7.3 y luego mostraremos cómo derivar las expresiones para las otras señales de control del registro. En la tabla se muestra que $R0_{in}$ se establece en el valor de X_0 en el primer paso de las operaciones *Load* y *Move*, y en el tercer paso de las operaciones *Add* y *Sub*, lo cual nos lleva a la expresión

$$R0_{in} = (I_0 + I_1)T_1X_0 + (I_2 + I_3)T_3X_0$$

De forma similar, $R0_{out}$ se establece en el valor de Y_0 en el primer paso de *Move*. Se establece en X_0 en el primer paso de *Add* y *Sub*, y en Y_0 en el segundo paso de estas operaciones, lo que da

$$R0_{out} = I_1 T_1 Y_0 + (I_2 + I_3)(T_1 X_0 + T_2 Y_0)$$

Las expresiones para $R1_{in}$ y $R1_{out}$ son las mismas que aquellas para $R0_{in}$ y $R0_{out}$, excepto que X_1 y Y_1 se usan en lugar de X_0 y Y_0. Las expresiones para $R2_{in}$, $R2_{out}$, $R3_{in}$ y $R3_{out}$ se derivan de la misma forma.

Los circuitos mostrados en las figuras 7.70 y 7.71, combinados con los circuitos representados por las expresiones anteriores, implementan el circuito de control de la figura 7.69.

Los procesadores son circuitos sumamente útiles de uso muy común. Hemos presentado sólo los aspectos más básicos de su diseño. Sin embargo, las técnicas expuestas pueden ampliarse para diseñar procesadores reales, como los microprocesadores modernos. El lector interesado puede referirse a los libros referentes a la organización de computadoras para que conozca más detalles del diseño de procesadores [1-2].

Código de VHDL

En esta sección damos dos estilos diferentes de código de VHDL para describir el sistema de la figura 7.69. El primero utiliza buffers triestado para representar el bus y da las expresiones lógicas mostradas líneas arriba para las salidas del circuito de control. El segundo estilo de código emplea multiplexores para representar el bus y usa instrucciones CASE que corresponden a la tabla 7.3 a fin de describir las salidas del circuito de control.

El código de VHDL para un contador ascendente se muestra en la figura 7.52. Una versión modificada de este contador, llamada *upcount*, se muestra en el código de la figura 7.72. Tiene una entrada reset síncrona, que está activa en nivel alto. En la figura 7.64 definimos el paquete llamado *components*, el cual proporciona declaraciones de componentes para varios subcircuitos. En el código de VHDL para el procesador usaremos los componentes *regn* y *trin* enumerados en la figura 7.64, pero no el componente *shiftr*. Creamos un paquete nuevo llamado *subccts* para usarlo con el procesador. El código no se muestra aquí, pero incluye declaraciones de componentes para *regn* (figura 7.61), *trin* (figura 7.62), *upcount* y *dec2to4* (figura 6.30).

El código completo para el procesador se presenta en la figura 7.73. En el cuerpo de arquitectura, las instrucciones etiquetadas *counter* y *decT* instancian los subcircuitos de la figura 7.70. Obsérvese que hemos supuesto que el circuito tiene una entrada de inicialización activa en nivel alto, *Reset*, que se utiliza para inicializar el contador en 00. La instrucción Func <= F & Rx & Ry usa el operador de concatenación para crear la señal de seis bits *Func*, la cual representa las entradas al registro de funciones de la figura 7.71. La siguiente instrucción instancia el registro de funciones con las entradas de datos *Func* y las salidas *FuncReg*. Las instrucciones etiquetadas *decI*, *decX* y *decY* instancian los decodificadores de la figura 7.71. Después de estas instrucciones se dan las expresiones lógicas derivadas antes para las salidas del circuito de control. Para $R0_{in}$, . . . , $R3_{in}$ y $R0_{out}$, . . . , $R3_{out}$, una instrucción GENERATE se usa para producir las expresiones.

Al final del código, los buffers triestado y los registros del procesador se instancian, y el módulo sumador/restador se describe con una asignación de la señal seleccionada.

Uso de multiplexores e instrucciones CASE

En la figura 7.60 mostramos que un bus puede implementarse usando multiplexores en vez de buffers triestado. El código de VHDL que describe al procesador usando este enfoque

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;

ENTITY upcount IS
    PORT ( Clear, Clock  : IN      STD_LOGIC ;
           Q             : BUFFER  STD_LOGIC_VECTOR(1 DOWNTO 0) ) ;
END upcount ;

ARCHITECTURE Behavior OF upcount IS
BEGIN
    upcount: PROCESS ( Clock )
    BEGIN
        IF (Clock'EVENT AND Clock = '1') THEN
            IF Clear = '1' THEN
                Q <= "00" ;
            ELSE
                Q <= Q + '1' ;
            END IF ;
        END IF;
    END PROCESS;
END Behavior ;
```

Figura 7.72 Código para un contador ascendente de dos bits con reset síncrono.

aparece en la figura 7.74. La misma declaración de entidad dada en la figura 7.73 puede usarse y no se muestra en la figura 7.74. El código ilustra una manera diferente de describir el circuito de control en el procesador. No da expresiones lógicas para las señales *Extern*, *Done* y el resto, como sí ocurre en la figura 7.73. En vez de ello, se emplean instrucciones CASE para representar la información mostrada en la tabla 7.3. Estas instrucciones se proporcionan dentro del proceso llamado *controlsignals*. A cada señal de control se asigna primero el valor de 0, en forma predeterminada. Esto es necesario porque las instrucciones CASE especifican los valores de las señales de control sólo cuando deben validarse, como se hizo en la tabla 7.3. Según se explicó para la figura 7.35, cuando el valor de una señal no está especificado, la señal conserva su valor presente. Esta memoria implícita da como resultado una conexión de retroalimentación en el circuito sintetizado. Evitamos este problema al proporcionar el valor predeterminado de 0 para cada una de las señales de control implícitas en las instrucciones CASE.

En la figura 7.73 las instrucciones etiquetadas *decT* y *decI* se usan para decodificar la señal *Count* y los valores almacenados de la entrada *F*, respectivamente. El decodificador *decT* tiene las salidas $T_0, \ldots, T_3$ y *decI* produce $I_0, \ldots, I_3$. En la figura 7.74 estos dos decodificadores no se utilizan, ya que no sirven para un propósito útil en este código. En vez de ello las señales *T* e *I* se definen como dos señales de dos bits, las cuales se usan en las instrucciones CASE. El código establece *T* con el valor de *Count*, mientras que *I* se establece con el valor de los dos bits del extremo izquierdo del registro de funciones, que corresponde a los valores almacenados de la entrada *F*.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_signed.all ;
USE work.subccts.all ;

ENTITY proc IS
    PORT ( Data      : IN     STD_LOGIC_VECTOR(7 DOWNTO 0) ;
           Reset, w  : IN     STD_LOGIC ;
           Clock     : IN     STD_LOGIC ;
           F, Rx, Ry : IN     STD_LOGIC_VECTOR(1 DOWNTO 0) ;
           Done      : BUFFER STD_LOGIC ;
           BusWires  : INOUT  STD_LOGIC_VECTOR(7 DOWNTO 0) ) ;
END proc ;

ARCHITECTURE Behavior OF proc IS
    SIGNAL Rin, Rout : STD_LOGIC_VECTOR(0 TO 3) ;
    SIGNAL Clear, High, AddSub : STD_LOGIC ;
    SIGNAL Extern, Ain, Gin, Gout, FRin : STD_LOGIC ;
    SIGNAL Count, Zero : STD_LOGIC_VECTOR(1 DOWNTO 0) ;
    SIGNAL T, I, X, Y : STD_LOGIC_VECTOR(0 TO 3) ;
    SIGNAL R0, R1, R2, R3 : STD_LOGIC_VECTOR(7 DOWNTO 0) ;
    SIGNAL A, Sum, G : STD_LOGIC_VECTOR(7 DOWNTO 0) ;
    SIGNAL Func, FuncReg : STD_LOGIC_VECTOR(1 TO 6) ;
BEGIN
    Zero <= "00" ; High <= '1' ;
    Clear <= Reset OR Done OR (NOT w AND T(0)) ;
    counter: upcount PORT MAP ( Clear, Clock, Count ) ;
    decT: dec2to4 PORT MAP ( Count, High, T );
    Func <= F & Rx & Ry ;
    FRin <= w AND T(0) ;
    functionreg: regn GENERIC MAP ( N => 6 )
                 PORT MAP ( Func, FRin, Clock, FuncReg ) ;
    decI: dec2to4 PORT MAP ( FuncReg(1 TO 2), High, I ) ;
    decX: dec2to4 PORT MAP ( FuncReg(3 TO 4), High, X ) ;
    decY: dec2to4 PORT MAP ( FuncReg(5 TO 6), High, Y ) ;
    Extern <= I(0) AND T(1) ;
    Done <= ((I(0) OR I(1)) AND T(1)) OR ((I(2) OR I(3)) AND T(3)) ;
    Ain <= (I(2) OR I(3)) AND T(1) ;
    Gin <= (I(2) OR I(3)) AND T(2) ;
    Gout <= (I(2) OR I(3)) AND T(3) ;
    AddSub <= I(3) ;
```

. . . continúa en el inciso *b*.

Figura 7.73 Código para el procesador (inciso *a*).

```
    RegCntl:
    FOR k IN 0 TO 3 GENERATE
        Rin(k) <= ((I(0) OR I(1)) AND T(1) AND X(k)) OR
            ((I(2) OR I(3)) AND T(3) AND X(k)) ;
        Rout(k) <= (I(1) AND T(1) AND Y(k)) OR
            ((I(2) OR I(3)) AND ((T(1) AND X(k)) OR (T(2) AND Y(k)))) ;
    END GENERATE RegCntl ;
    tri_extern: trin PORT MAP ( Data, Extern, BusWires ) ;
    reg0: regn PORT MAP ( BusWires, Rin(0), Clock, R0 ) ;
    reg1: regn PORT MAP ( BusWires, Rin(1), Clock, R1 ) ;
    reg2: regn PORT MAP ( BusWires, Rin(2), Clock, R2 ) ;
    reg3: regn PORT MAP ( BusWires, Rin(3), Clock, R3 ) ;
    tri0: trin PORT MAP ( R0, Rout(0), BusWires ) ;
    tri1: trin PORT MAP ( R1, Rout(1), BusWires ) ;
    tri2: trin PORT MAP ( R2, Rout(2), BusWires ) ;
    tri3: trin PORT MAP ( R3, Rout(3), BusWires ) ;
    regA: regn PORT MAP ( BusWires, Ain, Clock, A ) ;
    alu:
    WITH AddSub SELECT
        Sum <= A + BusWires WHEN '0',
               A - BusWires WHEN OTHERS ;
    regG: regn PORT MAP ( Sum, Gin, Clock, G ) ;
    triG: trin PORT MAP ( G, Gout, BusWires ) ;
END Behavior ;
```

Figura 7.73 Código para el procesador (inciso *b*).

Hay dos niveles anidados de instrucciones CASE. En el primero se enumeran los valores posibles de *T*. Por cada cláusula WHEN de esta instrucción CASE, la cual representa una columna de la tabla 7.3, hay una instrucción CASE anidada que enumera los cuatro valores de *I*. Según indican los comentarios del código, las instrucciones CASE anidadas corresponden exactamente a la información de la tabla 7.3.

Al final de la figura 7.74, el bus se describe con una asignación de señal seleccionada (Sel). Esta instrucción representa multiplexores que colocan los datos apropiados en *BusWires*, según los valores de R_{out}, G_{out} y *Extern*.

Los circuitos sintetizados a partir del código de las figuras 7.73 y 7.74 son funcionalmente equivalentes. El estilo del código de la figura 7.74 tiene la ventaja de que no requiere el esfuerzo manual de analizar la tabla 7.3 a fin de generar las expresiones lógicas para las señales de control usadas en la figura 7.73. Al emplear el estilo de código de la figura 7.74, estas expresiones se producen en forma automática mediante el compilador de VHDL como resultado del análisis de las instrucciones CASE. El estilo del código de la figura 7.74 es menos propenso a errores. Además, si este estilo se emplea es más sencillo proporcionar capacidades adicionales en el procesador, como la adición de otras operaciones.

```
ARCHITECTURE Behavior OF proc IS
    SIGNAL X, Y, Rin, Rout : STD_LOGIC_VECTOR(0 TO 3) ;
    SIGNAL Clear, High, AddSub : STD_LOGIC ;
    SIGNAL Extern, Ain, Gin, Gout, FRin : STD_LOGIC ;
    SIGNAL Count, Zero, T, I : STD_LOGIC_VECTOR(1 DOWNTO 0) ;
    SIGNAL R0, R1, R2, R3 : STD_LOGIC_VECTOR(7 DOWNTO 0) ;
    SIGNAL A, Sum, G : STD_LOGIC_VECTOR(7 DOWNTO 0) ;
    SIGNAL Func, FuncReg, Sel : STD_LOGIC_VECTOR(1 TO 6) ;
BEGIN
    Zero <= "00" ; High <= '1' ;
    Clear <= Reset OR Done OR (NOT w AND NOT T(1) AND NOT T(0)) ;
    counter: upcount PORT MAP ( Clear, Clock, Count ) ;
    T <= Count ;
    Func <= F & Rx & Ry ;
    FRin <= w AND NOT T(1) AND NOT T(0) ;
    functionreg: regn GENERIC MAP ( N => 6 )
                 PORT MAP ( Func, FRin, Clock, FuncReg ) ;
    I <= FuncReg(1 TO 2) ;
    decX: dec2to4 PORT MAP ( FuncReg(3 TO 4), High, X ) ;
    decY: dec2to4 PORT MAP ( FuncReg(5 TO 6), High, Y ) ;

    controlsignals: PROCESS ( T, I, X, Y )
    BEGIN
        Extern <= '0' ; Done <= '0' ; Ain <= '0' ; Gin <= '0' ;
        Gout <= '0' ; AddSub <= '0' ; Rin <= "0000" ; Rout <= "0000" ;
        CASE T IS    WHEN "00" => -- ninguna señal se valida en la etapa de tiempo T0
            WHEN "01" => -- define las señales validadas en la etapa de tiempo T1
                CASE I IS
                    WHEN "00" => -- Load
                        Extern <= '1' ; Rin <= X ; Done <= '1' ;
                    WHEN "01" => -- Move
                        Rout <= Y ; Rin <= X ; Done <= '1' ;
                    WHEN OTHERS => -- Add, Sub
                        Rout <= X ; Ain <= '1' ;
                END CASE ;
```

. . . continúa en el inciso *b*.

Figura 7.74 Código alternativo para el procesador (inciso *a*).

Sintetizamos un circuito para implementar el código de la figura 7.74 en un chip. En la figura 7.75 se da un ejemplo de los resultados de una simulación de tiempo. Cada ciclo del reloj en el que $w = 1$ en este diagrama de tiempo indica el comienzo de una operación. En la primera de estas operaciones, a 250 ns en el tiempo de simulación, los valores de las entradas *F* y *Rx* son 00. Por tanto, la operación corresponde a "*Load R*0,*Data*". El valor de *Data* es 2A, el cual se carga

```
            WHEN "10" => -- define las señales validadas en la etapa de tiempo T2
                CASE I IS
                    WHEN "10" => -- Add
                        Rout <= Y ; Gin <= '1' ;
                    WHEN "11" => -- Sub
                        Rout <= Y ; AddSub <= '1' ; Gin <= '1' ;
                    WHEN OTHERS => -- Load, Move
                END CASE ;
            WHEN OTHERS => -- define las señales validadas en la etapa de tiempo T3
                CASE I IS
                    WHEN "00" => -- Load
                    WHEN "01" => -- Move
                    WHEN OTHERS => -- Add, Sub
                        Gout <= '1' ; Rin <= X ; Done <= '1' ;
                END CASE ;
        END CASE ;
    END PROCESS ;
    reg0: regn PORT MAP ( BusWires, Rin(0), Clock, R0 ) ;
    reg1: regn PORT MAP ( BusWires, Rin(1), Clock, R1 ) ;
    reg2: regn PORT MAP ( BusWires, Rin(2), Clock, R2 ) ;
    reg3: regn PORT MAP ( BusWires, Rin(3), Clock, R3 ) ;
    regA: regn PORT MAP ( BusWires, Ain, Clock, A ) ;
    alu: WITH AddSub SELECT
        Sum <= A + BusWires WHEN '0',
               A - BusWires WHEN OTHERS ;
    regG: regn PORT MAP ( Sum, Gin, Clock, G ) ;
    Sel <= Rout & Gout & Extern ;
    WITH Sel SELECT
        BusWires <= R0 WHEN "100000",
                    R1 WHEN "010000",
                    R2 WHEN "001000",
                    R3 WHEN "000100",
                    G WHEN  "000010",
                    Data WHEN OTHERS ;
END Behavior ;
```

Figura 7.74 Código opcional para el procesador (inciso *b*).

en *R*0 en el siguiente flanco positivo del reloj. La operación siguiente carga 55 en el registro *R*1 y la operación subsiguiente carga 22 en *R*2. A 850 ns el valor de la entrada *F* es 10, mientras que *Rx* = 01 y *Ry* = 00. Esta operación es "*Add R*1,*R*0". En el ciclo del reloj siguiente, el contenido de *R*1 (55) aparece en el bus. Estos datos se cargan en el registro A por el flanco del reloj a 950 ns, lo cual también da como resultado que el contenido de *R*0 (2A) se coloque en el bus. El módulo sumador/restador genera la suma correcta (7F), la cual se carga en el registro *G* a 1050 ns.

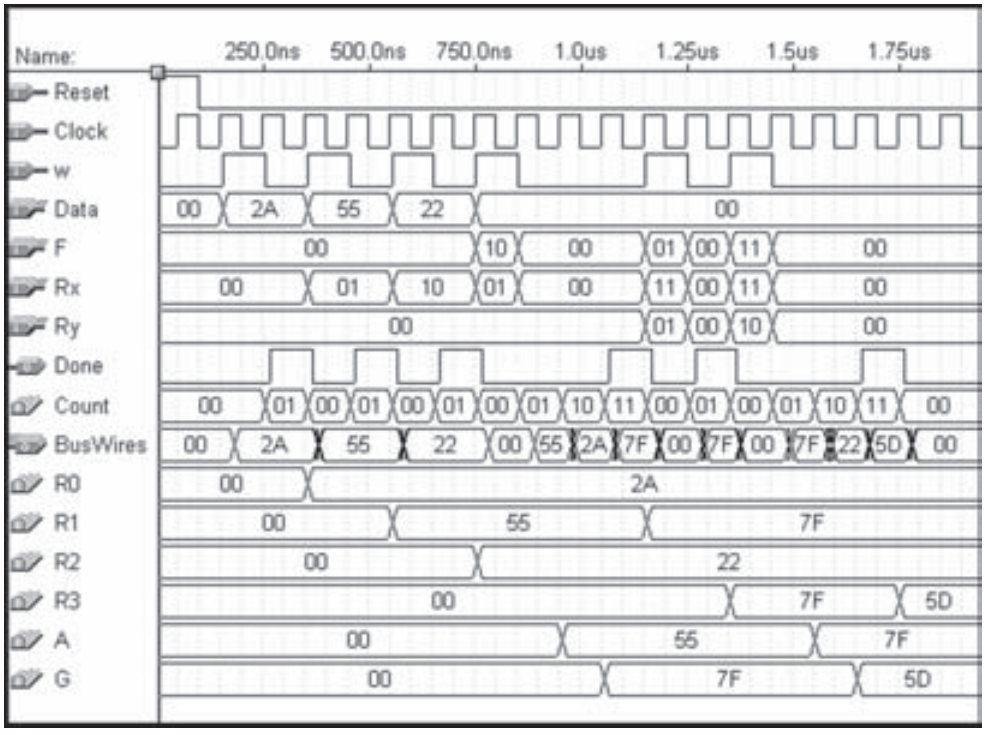


Figura 7.75 Simulación de tiempo para el código de VHDL de la figura 7.74.

Después de este flanco del reloj el contenido nuevo de *G* (7F) se coloca en el bus y se carga en el registro *R*1 a 1150 ns. Dos operaciones más se muestran en el diagrama de tiempo. Una a 1250 ns ("*Move R*3,*R*l") copia el contenido de *R*1 (7F) en *R*3. Finalmente, la operación que empieza en 1450 ns ("*Sub R*3,*R*2") resta el contenido de *R*2 (22) del contenido de *R*3 (7F), lo que produce el resultado correcto, $7F = 22 = 5D$.

7.14.3 CONTADOR DE TIEMPO DE REACCIÓN

En el capítulo 3 mostramos que los dispositivos electrónicos operan a velocidades increíblemente rápidas, con el retraso típico a través de una compuerta lógica menor de 1 ns. En este ejemplo usamos un circuito lógico para medir la velocidad de un tipo de dispositivo más lento: una persona.

Diseñaremos un circuito que puede emplearse para medir el tiempo de reacción de una persona ante un suceso específico. El circuito enciende una pequeña luz, llamada *diodo emisor de luz* (LED, *light-emitting diode*). En respuesta al encendido del LED, la persona intenta oprimir un interruptor lo más rápido posible. El circuito mide el tiempo transcurrido desde el momento en que el LED se encendió hasta que el interruptor se oprime.

Para medir el tiempo de reacción se necesita una señal de reloj con una frecuencia apropiada. En este ejemplo usamos un reloj de 100 Hz, que mide el tiempo en una resolución de 1/100

de segundo. El tiempo de reacción entonces puede exhibirse usando dos dígitos que representan fracciones de segundo desde 00/100 hasta 99/100.

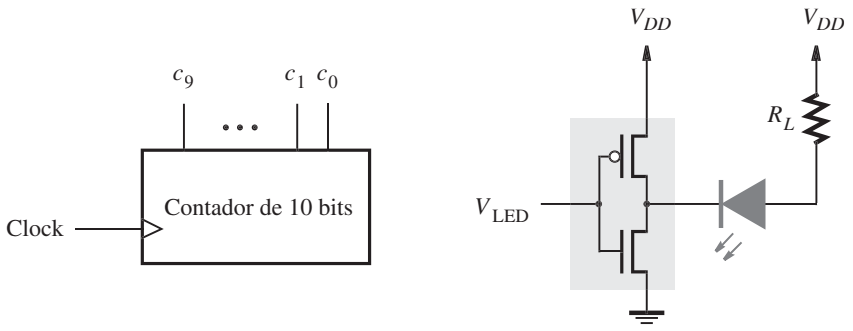
Los sistemas digitales suelen incluir señales de reloj para controlar varios subsistemas. En este caso suponemos la existencia de una señal de reloj de entrada con la frecuencia 102.4 kHz. Desde esta señal podemos derivar la señal de 100 Hz requerida si utilizamos un contador como un *divisor de reloj*. En la figura 7.22 se presenta un diagrama de tiempo para un contador de cuatro bits. Muestra que la salida del bit menos significativo, Q_0, del contador es una señal periódica con la mitad de la frecuencia de la entrada del reloj. Por tanto, podemos considerar que Q_0 divide la frecuencia del reloj entre dos. De forma similar, la salida Q_1 divide la frecuencia del reloj entre cuatro. En general, la salida Q_i en un contador de n bits divide la frecuencia del reloj entre 2^{i+1}. En el caso de nuestra señal de reloj de 102.4 kHz, podemos usar un contador de 10 bits, como se muestra en la figura 7.76*a*. La salida del contador c_9 tiene la frecuencia de 100 Hz requerida porque 102400 Hz/1024 = 100 Hz.

El circuito del contador de tiempo de reacción debe ser capaz de encender un LED y apagarlo. En la figura 7.76*b* se muestra en gris oscuro el símbolo gráfico de un LED. Las flechitas gris oscuro del símbolo representan la luz emitida cuando el LED está encendido. El LED tiene dos terminales: la que está a la izquierda de la figura es el *cátodo*, y la de la derecha es el *ánodo*. Para encender el LED, el cátodo debe establecerse en un voltaje inferior que el ánodo, lo que ocasiona que una corriente fluya por el LED. Si los voltajes en sus dos terminales son iguales, el LED está apagado.

En la figura 7.76*b* se muestra una manera de controlar el LED con un inversor. Si el voltaje de entrada $V_{LED} = 0$, entonces el voltaje en el cátodo es igual a V_{DD}; por tanto, el LED está apagado. Pero si $V_{LED} = V_{DD}$, el voltaje del cátodo es 0 V y el LED está encendido. La cantidad de corriente que fluye está limitada por el valor de la resistencia R_L. Esta corriente pasa por el LED en el transistor NMOS del inversor. Como la corriente fluye *hacia* el inversor, decimos que el inversor *hunde* la corriente. La corriente máxima que una compuerta lógica puede hundir sin sufrir un daño por lo general se llama I_{OL}, que significa "la corriente máxima cuando la salida está en un nivel bajo". El valor de R_L se elige de modo que la corriente sea menor que I_{OL}. Como ejemplo supóngase que el inversor se implementó dentro de un PLD. El valor común de I_{OL}, el cual se especificaría en la hoja de datos del PLD, se aproxima a 12 mA. Para $V_{DD} = 5$ V, esto conduce a $R_L \approx 450\ \Omega$ porque 5 V /450 Ω = 11 mA (en realidad hay una pequeña caída de voltaje en el LED cuando se enciende, pero la ignoramos en aras de la sencillez). La cantidad de luz emitida por el LED es proporcional al flujo de corriente. Si 11 mA es insuficiente, entonces el inversor debe implementarse en un chip de buffer como los descritos en la sección 3.5, ya que los buffers ofrecen un volumen mayor de I_{OL}.

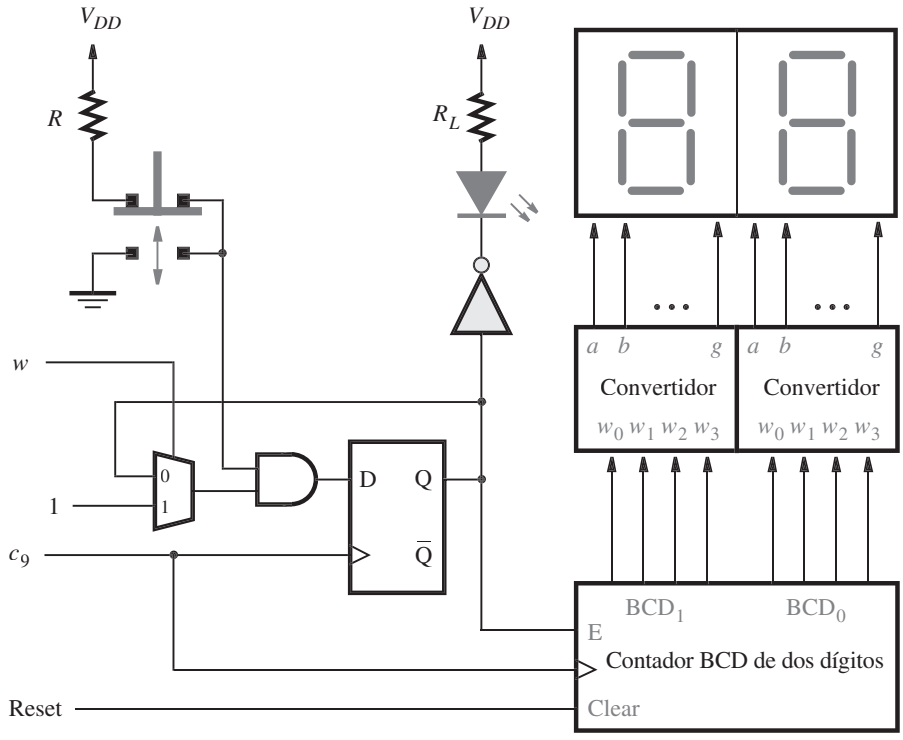
El circuito completo del contador de tiempo de reacción se ilustra en la figura 7.76*c*, con el inversor del inciso *b*) sombreado en gris claro. El símbolo gráfico de un interruptor con un botón para oprimir se muestra a la izquierda del diagrama. El interruptor normalmente hace contacto con las terminales superiores, como se describe en la figura. Cuando se oprime, el interruptor hace contacto con las terminales inferiores; cuando se suelta, automáticamente regresa a la posición superior. En la figura el interruptor está conectado de tal forma que normalmente produce un valor lógico de 1 y un pulso de 0 cuando se presiona.

Oprimir el botón del interruptor causa que el flip-flop D se inicialice de manera síncrona. La salida de este flip-flop determina si el LED está encendido o apagado, y también proporciona la entrada de habilitación del conteo para un contador BCD de dos dígitos. Como estudiamos en la sección 7.11, cada dígito en un contador BCD tiene cuatro bits que toman los valores 0000 a 1001.



a) Divisor del reloj

b) Circuito LED



c) Interruptor de botón, LED, y pantallas de siete segmentos

Figura 7.76 Un circuito contador de tiempo de reacción.

Por tanto, la secuencia de conteo puede verse como números decimales desde 00 a 99. Un circuito para el contador BCD se da en la figura 7.28. En la figura 7.76*c* tanto el flip-flop como el contador se sincronizan por la salida c_9 del divisor del reloj en el inciso *a*) de la figura. El uso buscado del circuito contador de tiempo de reacción consiste en primero oprimir el interruptor para apagar el LED e inhabilitar el contador. Luego la entrada *Reset* se valida para borrar el contenido del contador y dejarlo en 00. La entrada *w* normalmente tiene el valor 0, el cual mantiene el flip-flop borrado e impide que cambie el valor de conteo. La prueba de reacción se inicia al establecer $w = 1$ para un ciclo del reloj c_9. Después del siguiente flanco positivo de c_9, la salida del flip-flop se vuelve 1, con lo que el LED se enciende. Suponemos que *w* vuelve a 0 después de un ciclo del reloj, pero la salida del flip-flop permanece en 1 porque el multiplexor dos a uno está conectado a la entrada D. El contador entonces se incrementa cada 1/100 de segundo. Cada dígito del contador se conecta por medio de un convertidor de código a una pantalla de siete segmentos, la cual se describió en el análisis de la figura 6.25. Cuando el usuario presiona el interruptor, el flip-flop se borra, con lo que el LED se apaga y el contador se detiene. La pantalla de dos dígitos muestra el tiempo transcurrido hasta el 1/100 de segundo más cercano desde que el LED se encendió hasta que el usuario pudo responder oprimiendo el interruptor.

Código de VHDL

Para describir el circuito de la figura 7.76*c* con código de VHDL podemos emplear los subcircuitos para el contador BCD y el código para el convertidor de siete segmentos. Este último se dio en la figura 6.47 y no lo repetiremos aquí. El código para el contador BCD, que representa el circuito de la figura 7.28, se muestra en la figura 7.77. La salida BCD de dos dígitos está representada por las dos señales de cuatro dígitos *BCD*1 y *BCD*0. La entrada *Clear* sirve para proporcionar un reset síncrono para los dos dígitos en el contador. Si $E = 1$, el valor de conteo se incrementa en el flanco positivo del reloj, y si $E = 0$ el valor de conteo no sufre cambio alguno. Cada dígito puede tomar los valores de 0000 a 1001.

En la figura 7.78 se proporciona el código para el contador de tiempo de reacción. La señal de entrada *Pushn* representa el valor producido por el interruptor de botón para oprimir. La señal de salida *LEDn* representa la salida del inversor que se usa para controlar el LED. Las dos pantallas de siete segmentos están controladas por las señales de siete bits *Digit*1 y *Digit*0.

En la figura 7.56 se muestra cómo un registro, *R*, puede diseñarse con una señal de control R_{in}. Si $R_{in} = 1$ los datos se cargan en el registro en el flanco activo del reloj y si $R_{in} = 0$ el contenido almacenado del registro no cambia. El flip-flop de la figura 7.76 se usa de la misma forma. Si $w = 1$, el flip-flop se carga con el valor 1, pero si $w = 0$ el valor almacenado en el flip-flop no cambia. Este circuito se describe mediante el proceso etiquetado *flipflop* de la figura 7.78, que también incluye una entrada de reset síncrono. Hemos optado por usar un reset síncrono porque la salida del flip-flop está conectada a la entrada de habilitación *E* en el contador BCD. Como sabemos a partir de lo explicado en la sección 7.3, es importante que todas las señales conectadas a los flip-flops satisfagan los tiempos de preparación y espera requeridos. El interruptor de botón puede presionarse en cualquier momento y no está sincronizado por la señal c_9 del reloj. Al usar un reset síncrono para el flip-flop de la figura 7.76, evitamos problemas de sincronización potenciales en el contador.

La salida del flip-flop se denomina *LED*, la cual se invierte para producir la señal *LEDn* que controla el LED. En el dispositivo usado para implementar el circuito, *LEDn* sería generada por el buffer que está conectado a un pin de salida en el paquete de chips. Si se emplea un PLD, este buffer tiene el valor asociado de $I_{OL} = 12$ mA que mencionamos antes. Al final de la figura 7.78 el contador BCD y los convertidores de siete segmentos se instancian como subcircuitos.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;

ENTITY BCDcount IS
    PORT ( Clock          : IN     STD_LOGIC ;
           Clear, E       : IN     STD_LOGIC ;
           BCD1, BCD0     : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END BCDcount ;

ARCHITECTURE Behavior OF BCDcount IS
BEGIN
    PROCESS ( Clock )
    BEGIN
        IF Clock'EVENT AND Clock = '1' THEN
            IF Clear = '1' THEN
                BCD1 <= "0000" ; BCD0 <= "0000" ;
            ELSIF E = '1' THEN
                IF BCD0 = "1001" THEN
                    BCD0 <= "0000" ;
                    IF BCD1 = "1001" THEN
                        BCD1 <= "0000";
                    ELSE
                        BCD1 <= BCD1 + '1' ;
                    END IF ;
                ELSE
                    BCD0 <= BCD0 + '1' ;
                END IF ;
            END IF ;
        END IF;
    END PROCESS;
END Behavior ;
```

Figura 7.77 Código para el contador BCD de dos dígitos de la figura 7.28.

Una simulación del circuito contador de tiempo de reacción implementado en un chip se muestra en la figura 7.79. Al principio, *Pushn* se establece en 0 para simular que se oprime el interruptor de modo que se encienda el LED y luego *Pushn* regresa a 1. Además, *Reset* se valida para borrar el contador. Cuando *w* cambia a 1, el circuito establce *LEDn* en 0, lo que representa el encendido del LED. Después de un tiempo el interruptor se presionará. En la simulación establecimos arbitrariamente *Pushn* en 0 después de 18 ciclos del reloj c_9. Por tanto, esta elección representa el caso en que el tiempo de reacción de la persona es de alrededor de 0.18 segundos. En términos humanos, esta duración es muy breve; para los circuitos electrónicos es un tiempo muy largo. ¡Una computadora personal barata puede realizar decenas de millones de operaciones en 0.18 segundos!

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY reaction IS
    PORT ( c9, Reset        : IN     STD_LOGIC ;
           w, Pushn         : IN     STD_LOGIC ;
           LEDn             : OUT    STD_LOGIC ;
           Digit1, Digit0   : BUFFER STD_LOGIC_VECTOR(1 TO 7) ) ;
END reaction ;

ARCHITECTURE Behavior OF reaction IS
    COMPONENT BCDcount
        PORT ( Clock          : IN     STD_LOGIC ;
               Clear, E       : IN     STD_LOGIC ;
               BCD1, BCD0     : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
    END COMPONENT ;
    COMPONENT seg7
        PORT ( bcd    : IN  STD_LOGIC_VECTOR(3 DOWNTO 0) ;
               leds   : OUT STD_LOGIC_VECTOR(1 TO 7) ) ;
    END COMPONENT ;
    SIGNAL LED : STD_LOGIC ;
    SIGNAL BCD1, BCD0 : STD_LOGIC_VECTOR(3 DOWNTO 0) ;
BEGIN
    flipflop: PROCESS
    BEGIN
        WAIT UNTIL c9'EVENT AND c9 = '1' ;
        IF Pushn = '0' THEN
            LED <= '0' ;
        ELSIF w = '1' THEN
            LED <= '1' ;
        END IF ;
    END PROCESS ;

    LEDn <= NOT LED ;
    counter: BCDcount PORT MAP ( c9, Reset, LED, BCD1, BCD0 ) ;
    seg1 : seg7 PORT MAP ( BCD1, Digit1 ) ;
    seg0 : seg7 PORT MAP ( BCD0, Digit0 ) ;
END Behavior ;
```

Figura 7.78 Código para el contador de tiempo de reacción.

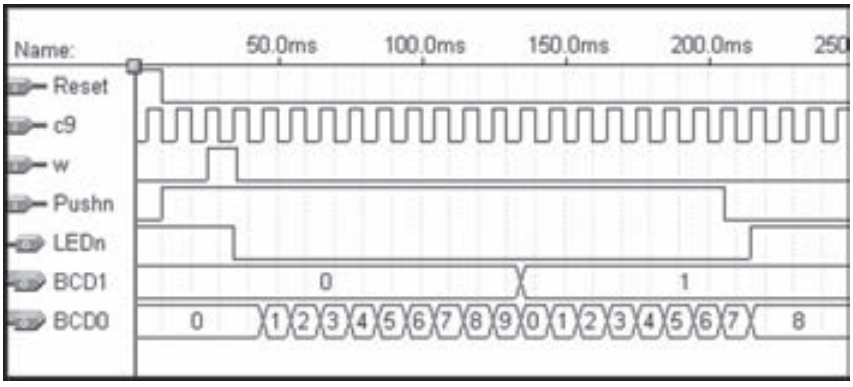


Figura 7.79 Simulación del circuito contador de tiempo de reacción.

7.14.4 CÓDIGO DE NIVEL DE TRANSFERENCIA DE REGISTROS (RTL)

Hasta ahora hemos presentado la mayor parte de los constructores de VHDL necesarios para la síntesis. Casi todos nuestros ejemplos muestran código por comportamiento que utiliza instrucciones IF-THEN-ELSE, CASE, FOR LOOP, etcétera. Es posible escribir código por comportamiento en un estilo parecido a un programa de computadora, en el cual hay un flujo de control complejo con muchos ciclos y ramas. Con un código como ése, a veces llamado código por comportamiento de *alto nivel*, es difícil relacionar el código con la implementación del hardware final; incluso podría ser difícil predecir qué circuito producirá una herramienta de síntesis de alto nivel. En este libro no usamos el estilo de código de alto nivel. En vez de ello, presentamos el código de VHDL de forma tal que pueda relacionarse fácilmente con el circuito que se está describiendo. El grueso de los módulos de diseño presentados es muy pequeño a fin de ofrecer descripciones sencillas. Los diseños más grandes se construyen interconectando los módulos más pequeños. Este enfoque se conoce como estilo de código de *nivel de transferencia de registros* (RTL, *register-transfer level*). Es el enfoque de diseño más popular en la práctica. El código RTL se caracteriza por un flujo de control directo a través del código; comprende subcircuitos bien entendidos y conectados juntos de una manera simple.

7.15 COMENTARIOS FINALES

En este capítulo hemos presentado circuitos que sirven como elementos de almacenamiento básicos en los sistemas digitales. Estos elementos se utilizan para construir unidades más grandes, como registros, registros de corrimiento y contadores. Muchos otros libros abordan esos temas [3-11]. Hemos ilustrado cómo los circuitos con flip-flops pueden describirse con código de VHDL. Más información respecto a VHDL puede encontrarse en [12-17]. En el capítulo siguiente se presentará un método más formal para diseñar circuitos con flip-flops.

7.16 EJEMPLOS DE PROBLEMAS RESUELTOS

En esta sección se presentan algunos problemas comunes que el lector puede encontrar y se muestra cómo resolverlos.

Ejemplo 7.13

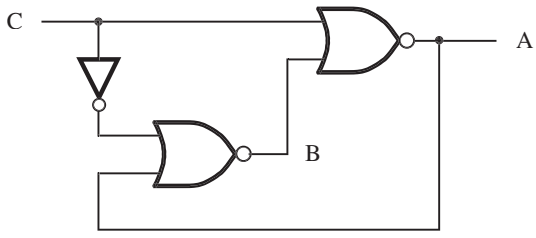
Problema: Considere el circuito de la figura 7.80*a*. Suponga que la entrada *C* está manejada por una señal de onda cuadrada con un ciclo de trabajo de 50%. Dibuje un diagrama de tiempo que muestre las formas de onda en los puntos *A* y *B*. Suponga que el retraso de propagación por cada compuerta es de Δ segundos.

Solución: El diagrama de tiempo se muestra en la figura 7.80*b*.

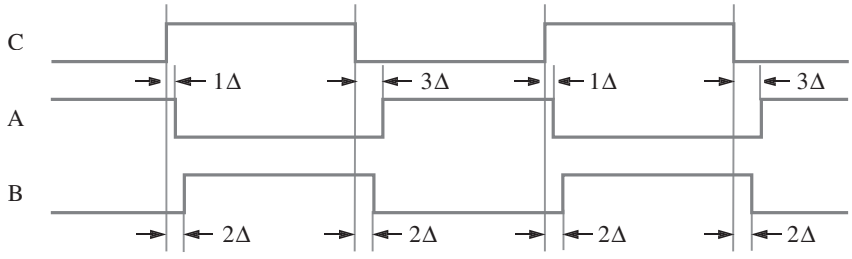
Ejemplo 7.14

Problema: Determine el comportamiento funcional del circuito de la figura 7.81. Suponga que la entrada *w* está manejada por una señal de onda cuadrada.

Solución: Cuando los dos flip-flops se borran, sus salidas son $Q_0 = Q_1 = 0$. Después que la entrada Clear adquiere un nivel alto, cada pulso en la entrada *w* ocasionará un cambio en los



a) Circuito



b) Diagrama de tiempo

Figura 7.80 Circuito para el ejemplo 7.13.

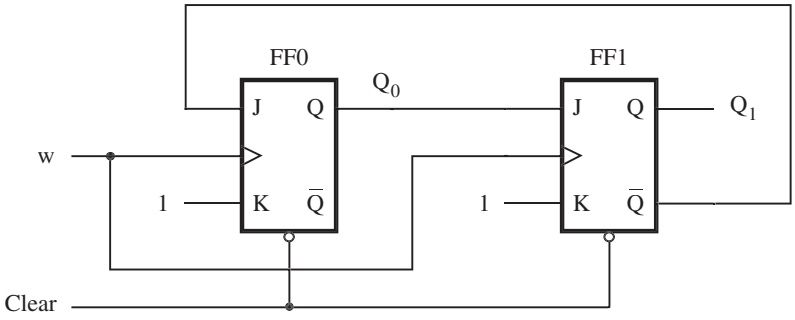


Figura 7.81 Circuito para el ejemplo 7.14.

Intervalo de tiempo	FF0			FF1		
	J_0	K_0	Q_0	J_1	K_1	Q_1
Clear	1	1	0	0	1	0
t_1	1	1	1	1	1	0
t_2	0	1	0	0	1	1
t_3	1	1	0	0	1	0
t_4	1	1	1	1	1	0

Figura 7.82 Resumen del comportamiento del circuito de la figura 7.81.

flip-flops según se indica en la figura 7.82. Observe que la figura muestra el estado de las señales después de los cambios ocasionados por el flanco de subida de un pulso.

En los intervalos de tiempo consecutivos los valores de Q_1 Q_0 son 00, 01, 10, 00, 01 y así sucesivamente. Por consiguiente, el circuito genera la secuencia de conteo 0, 1, 2, 0, 1 etcétera. Por tanto, el circuito es un contador módulo 3.

Ejemplo 7.15 **Problema:** En la figura 7.70 se muestra un circuito que genera cuatro señales de control de tiempo T_0, T_1, T_2 y T_3. Diseñe un circuito que genere seis señales como éstas, T_0 a T_5.

Solución: El esquema de la figura 7.70 puede ampliarse al usar un contador módulo 6, dado en la figura 7.26, y un decodificador que produce las seis señales de sincronización. Una alternativa más simple es usar un contador Johnson. Si empleamos tres flip-flops tipo D en una estructura como la descrita en la figura 7.30, podemos generar seis patrones de bits $Q_0Q_1Q_2$ como se muestra en la figura 7.83. Luego, usando otras seis compuertas AND de dos entradas, como se advierte en la figura, podemos obtener las señales buscadas. Note que los patrones $Q_0Q_1Q_2$ iguales a 010 y 101 no pueden ocurrir en el contador Johnson, así que los casos se tratan como condiciones sin importancia.

Ciclo del reloj	Q_0	Q_1	Q_2	Señal de control
0	0	0	0	$T_0 = \overline{Q}_0\overline{Q}_2$
1	1	0	0	$T_1 = Q_0\overline{Q}_1$
2	1	1	0	$T_2 = Q_1\overline{Q}_2$
3	1	1	1	$T_3 = Q_0Q_2$
4	0	1	1	$T_4 = \overline{Q}_0Q_1$
5	0	0	1	$T_5 = \overline{Q}_1Q_2$

Figura 7.83 Señales de tiempo para el ejemplo 7.15.

Ejemplo 7.16

Problema: Diseñe un circuito que sirva para controlar una máquina expendedora. El circuito tiene cinco entradas: Q (moneda de 25 centavos), D (moneda de 10 centavos), N (moneda de 5 centavos), *Coin* y *Resetn*. Cuando una moneda se deposita en la máquina, un mecanismo detector de monedas genera un pulso en la entrada apropiada (Q, D o N). Para dar significado a la ocurrencia del evento, el mecanismo también genera un pulso en la línea *Coin*. El circuito se inicializa usando la señal *Resetn* (activa en nivel bajo). Cuando se han depositado al menos 30 centavos, el circuito activa su salida, Z. No se da cambio si el monto excede 30 centavos.

Diseñe el circuito usando los componentes siguientes: un sumador de seis bits, un registro de seis bits y cualquier número de compuertas AND, OR y NOT.

Solución: En la figura 7.84 se muestra un circuito posible. El valor de cada moneda se representa por medio de un número correspondiente de cinco bits. Se añade al total actual, el cual se aloja en el registro *S*. La salida requerida es

$$Z = s_5 + s_4s_3s_2s_1$$

El registro se sincroniza por el flanco negativo de la señal *Coin*. Esto permite un retraso de propagación a través del sumador y asegura que una suma correcta se coloque en el registro.

En el capítulo 9 mostraremos cómo este tipo de circuito de control puede diseñarse usando un enfoque más estructurado.

Ejemplo 7.17

Problema: Escriba código de VHDL para implementar el circuito de la figura 7.84.

Solución: En la figura 7.85 se proporciona el código deseado.

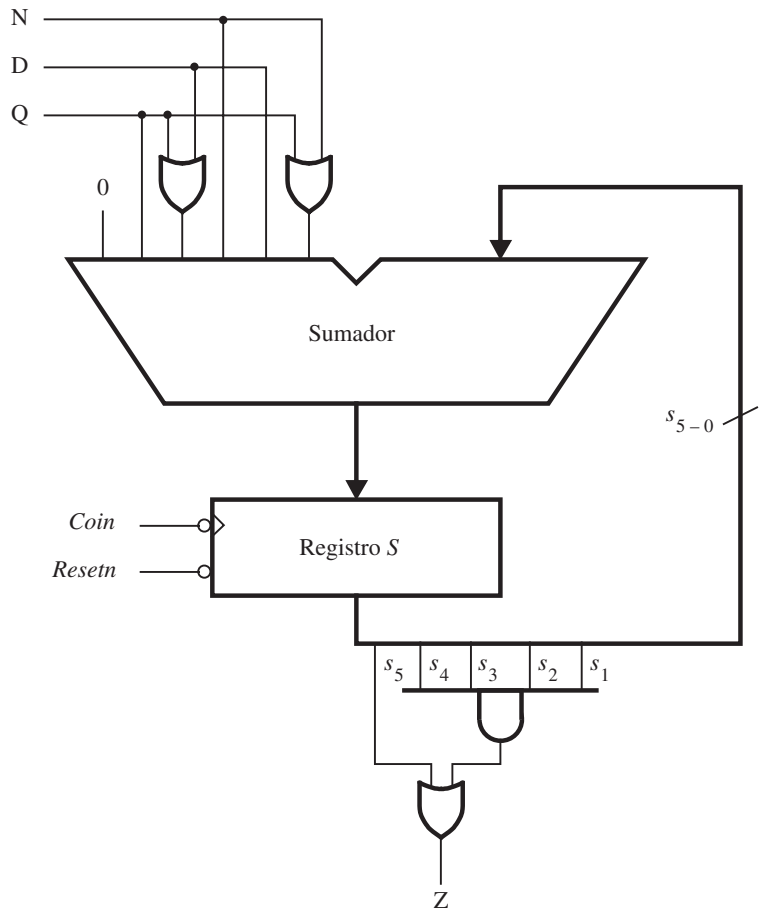


Figura 7.84 Circuito para el ejemplo 7.16.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_signed.all ;

ENTITY vend IS
    PORT ( N, D, Q, Resetn, Coin  : IN   STD_LOGIC ;
           Z                      : OUT  STD_LOGIC ) ;
END vend ;

ARCHITECTURE Behavior OF vend IS
    SIGNAL X:  STD_LOGIC_VECTOR(4 DOWNTO 0) ;
    SIGNAL S:  STD_LOGIC_VECTOR(5 DOWNTO 0) ;
BEGIN
    X(0) <= N OR Q ;
    X(1) <= D ;
    X(2) <= N ;
    X(3) <= D OR Q ;
    X(4) <= Q ;
    PROCESS ( Resetn, Coin )
    BEGIN
        IF Resetn = '0' THEN
            S <= "000000" ;
        ELSIF Coin'EVENT AND Coin = '0' THEN
            S <= ('0' & X) + S ;
        END IF ;
    END PROCESS ;
    Z <= S(5) OR (S(4) AND S(3) AND S(2) AND S(1)) ;
END Behavior ;
```

Figura 7.85 Código para el ejemplo 7.17.

PROBLEMAS

Al final del libro se incluyen las respuestas a los problemas marcados con asterisco.

7.1 Considere el diagrama de tiempo de la figura P7.1. Suponiendo que las entradas *D* y *Clock* mostradas se aplican al circuito de la figura 7.12, dibuje formas de onda para las señales Q_a, Q_b y Q_c.

7.2 ¿El circuito de la figura 7.3 puede modificarse para implementar un latch SR? Explique su respuesta.

7.3 En la figura 7.5 se muestra un latch construido con compuertas NOR. Dibuje un latch parecido usando compuertas NAND. Derive su tabla característica y muestre su diagrama de tiempo.

***7.4** Muestre un circuito que implemente el latch SR asíncrono usando únicamente compuertas NAND.

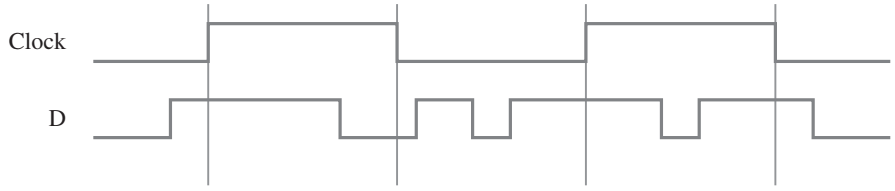


Figura P7.1 Diagrama de tiempo para el problema 7.1.

7.5 Dada una señal de reloj de 100 MHz, derive un circuito usando flip-flops D para generar señales de reloj de 50 y 25 MHz. Trace un diagrama de tiempo para las tres señales de reloj, suponiendo retrasos razonables.

***7.6** Un flip-flop SR es uno que tiene entradas set y reset como un latch SR asíncrono. Muestre cómo puede construirse un flip-flop SR mediante un flip-flop D y otras compuertas lógicas.

7.7 El latch SR asíncrono de la figura 7.6*a* tiene un comportamiento impredecible si las entradas *S* y *R* son iguales a 1 cuando *Clk* cambia a 0. Una forma de resolver este problema es crear un latch SR asíncrono con *set dominante* en el que la condición $S = R = 1$ hace que el latch se establezca en 1. Diseñe un latch SR asíncrono con set dominante y muestre el circuito.

7.8 Muestre cómo un flip-flop JK puede construirse con un flip-flop T y otras compuertas lógicas.

***7.9** Considere el circuito de la figura P7.2. Suponga que las dos compuertas NAND tienen retrasos de propagación mucho más grandes (alrededor de cuatro veces) que las otras compuertas del circuito. ¿Cómo se compara este circuito con los circuitos que hemos estudiado en este capítulo?

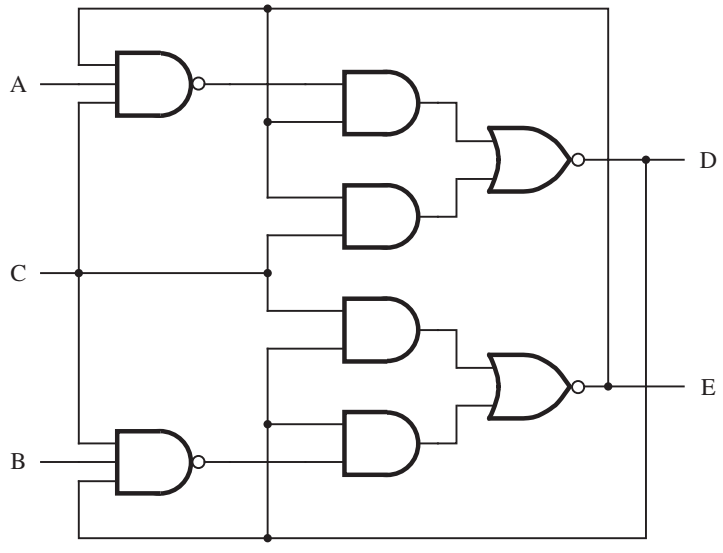


Figura P7.2 Circuito para el problema 7.9.

7.10 Escriba código de VHDL que represente un flip-flop T con una entrada clear asíncrona. Use código por comportamiento en vez de código estructural.

7.11 Escriba código de VHDL que represente un flip-flop JK. Utilice código por comportamiento en vez de código estructural.

7.12 Sintetice un circuito para el código escrito para el problema 7.11 empleando sus herramientas CAD. Simule el circuito y muestre un diagrama de tiempo que verifique la funcionalidad deseada.

7.13 Un registro de corrimiento universal puede desplazarse tanto de izquierda a derecha como de derecha a izquierda, y tiene una capacidad de carga en paralelo. Dibuje un circuito para este registro de corrimiento.

7.14 Escriba código de VHDL para un registro de corrimiento universal con *n* bits.

7.15 Diseñe un contador síncrono de cuatro bits con carga en paralelo. Use flip-flops T en vez de los flip-flops D utilizados en la sección 7.9.3.

***7.16** Diseñe un contador ascendente/descendente de tres bits con flip-flops T. Debe incluir una entrada de control llamada $\overline{\text{Up}}$/Down. Si $\overline{\text{Up}}$/Down $= 0$, entonces el circuito debe comportarse como un contador ascendente. Si es igual a 1, entonces el circuito debe comportarse como un contador descendente.

7.10 Repita el problema 7.16 usando flip-flops D.

***7.18** El circuito de la figura P7.3 parece un contador. ¿Cuál es la secuencia en que cuenta?

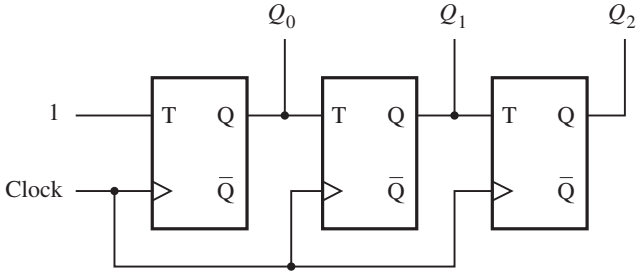


Figura P7.3 Circuito para el problema 7.18.

7.19 Considere el circuito de la figura P7.4. ¿Cómo se compara con el circuito de la figura 7.17? ¿Los dos circuitos pueden usarse con el mismo propósito? Si no es así, ¿cuál es la diferencia principal entre ellos?

7.20 Construya un circuito con compuertas NOR, parecido al de la figura 7.11*a*, el cual implementa un flip-flop D disparado por el flanco negativo.

7.21 Escriba código por comportamiento de VHDL que represente un contador ascendente/descendente de 24 bits con una carga en paralelo y un reset asíncrono.

7.22 Modifique el código de VHDL de la figura 7.52 agregándole un parámetro que establezca el número de flip-flops en el contador.

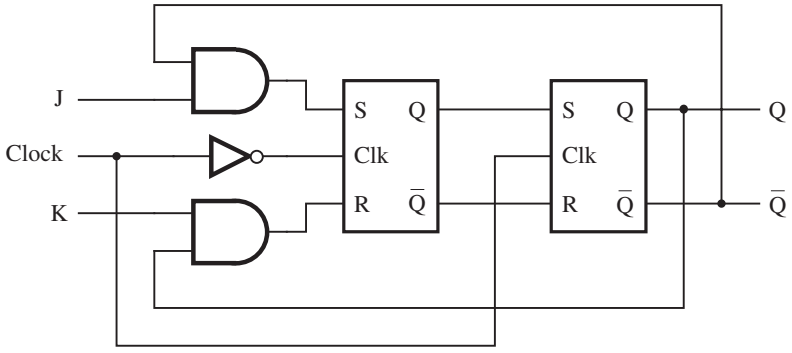


Figura P7.4 Circuito para el problema 7.19.

7.23 Escriba el código por comportamiento de VHDL que represente un contador ascendente módulo 12 con reset síncrono.

***7.28** Para los flip-flops del contador de la figura 7.25, suponga que $t_{su} = 3$ ns, $t_h = 1$ ns y el retraso de propagación a través de un flip-flop es 1 ns. Asuma que cada compuerta AND y XOR y cada multiplexor dos a uno tiene un retraso de propagación igual a 1 ns. ¿Cuál es la frecuencia de reloj máxima para la que el circuito funcionará correctamente?

7.25 Escriba código jerárquico (estructural) para el circuito de la figura 7.28. Use el contador de la figura 7.25 como un subcircuito.

7.26 Escriba código de VHDL que represente un contador Johnson de ocho bits. Sintetice el código con sus herramientas CAD y dé una simulación de tiempo que muestre la secuencia de conteo.

7.27 Escriba código por comportamiento de VHDL en el estilo mostrado en la figura 7.51 que represente un contador en anillo. Su código debe tener un parámetro *N* que establezca el número de flip-flops en el contador.

***7.28** Escriba código por comportamiento de VHDL que describa la funcionalidad del circuito mostrado en la figura 7.42.

7.29 En la figura 7.65 se proporciona un código de VHDL para un sistema digital que intercambia el contenido de dos registros, *R*1 y *R*2, usando el registro *R*3 para almacenamiento temporal. Construya un esquema equivalente con sus herramientas CAD para este sistema. Sintetice un circuito para este esquema y realice una simulación de tiempo.

7.30 Repita el problema 7.29 usando el circuito de control de la figura 7.59.

7.31 Modifique el código de la figura 7.67 para utilizar el circuito de control de la figura 7.59. Sintetice el código para su implementación en un chip y realice una simulación de tiempo.

7.32 En la sección 7.14.2 diseñamos un procesador que efectúa las operaciones indicadas en la tabla 7.3. Diseñe un circuito modificado que realice una operación adicional, Swap Rx, Ry. Esta operación intercambia el contenido de los registros *Rx* y *Ry*. Use tres bits $f_2 f_1 f_0$ para representar la entrada *F* mostrada en la figura 7.71 porque ahora hay cinco operaciones en vez de cuatro. Añada un nuevo registro, llamado *Tmp*, al sistema para que sea el almacenamiento temporal durante la operación de intercambio. Muestre expresiones lógicas para las salidas del circuito de control, como se hizo en la sección 7.14.2.

7.33 Un oscilador en anillo es un circuito que tiene un número impar, n, de inversores conectados en una estructura tipo anillo, como se muestra en la figura P7.5. La salida de cada inversor es una señal periódica con cierto periodo.

a) Suponga que todos los inversores son idénticos; en consecuencia, todos tienen el mismo retraso, llamado t_p. Sea f la salida de uno de los inversores. Dé una ecuación que exprese el periodo de la señal f en términos de n y t_p.

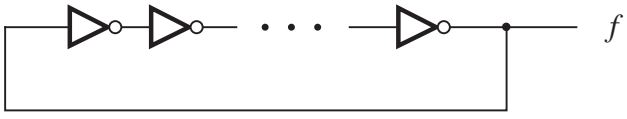


Figura P7.5 Un oscilador en anillo.

b) Para este inciso usted va a diseñar un circuito que pueda usarse para medir en forma experimental el retraso t_p a través de uno de los inversores en el oscilador en anillo. Suponga la existencia de una entrada llamada *Reset* y otra llamada *Interval*. La sincronización de estas dos señales se muestra en la figura P7.6. El periodo para el cual *Interval* tiene el valor 1 se conoce. Suponga que es 100 ns. Diseñe un circuito que utilice las señales *Reset* e *Interval* y la señal f del inciso *a*) para medir experimentalmente t_p. En su diseño puede usar compuertas lógicas y subcircuitos como sumadores, flip-flops, contadores, registros o cualquier otro.

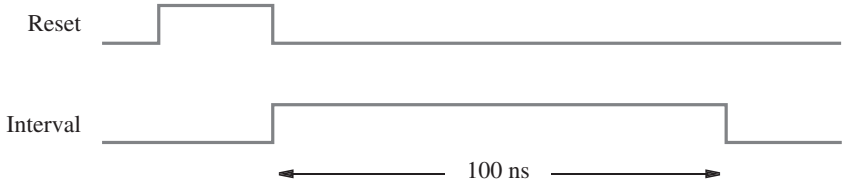


Figura P7.6 Sincronización de las señales para el problema 7.31.

7.34 Un circuito para un latch D asíncrono se muestra en la figura P7.7. Suponga que el retraso de propagación a través de una compuerta NAND o de un inversor es de 1 ns. Complete el diagrama de tiempo dado en la figura, el cual muestra los valores de la señal con resolución de 1 ns.

***7.35** Un circuito lógico tiene dos entradas, *Clock* y *Start*, y dos salidas, f y g. El comportamiento del circuito se describe en el diagrama de tiempo de la figura P7.8. Cuando se recibe un pulso en la entrada *Start*, el circuito produce pulsos en las salidas f y g como se muestra en el diagrama de tiempo. Diseñe un circuito adecuado usando sólo los componentes siguientes: un contador síncrono de tres bits capaz de inicializarse y disparado por el flanco positivo, y compuertas lógicas básicas. Para dar su respuesta suponga que los retrasos a través de todas las compuertas lógicas y el contador son insignificantes.

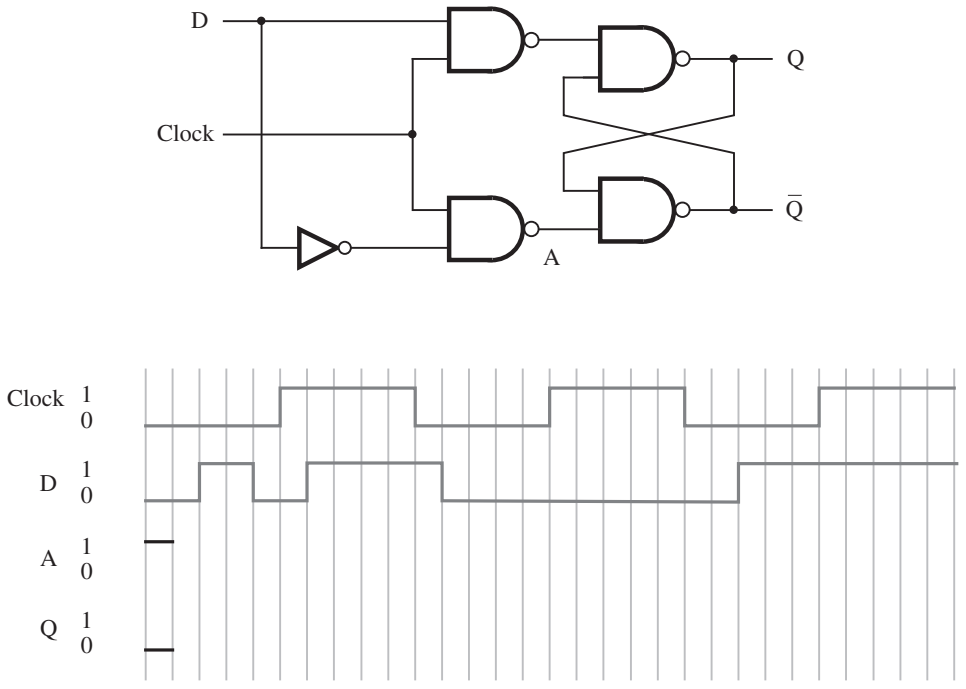


Figura P7.7 Circuito y diagrama de tiempo para el problema 7.32.

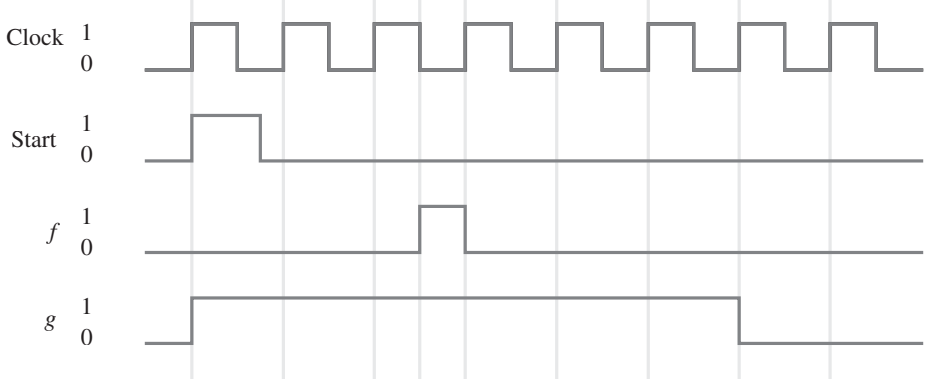


Figura P7.8 Diagrama de tiempo para el problema 7.33.

BIBLIOGRAFÍA

1. V. C. Hamacher, Z. G. Vranesic y S. G. Zaky, *Computer Organization*, 5a. ed. (McGraw-Hill: Nueva York, 2002).
2. D. A. Patterson y J. L. Hennessy, *Computer Organization and Design—The Hardware/Software Interface*, 2a. ed. (Morgan Kaufmann: San Francisco, Ca., 1998).
3. D. D. Gajski, *Principles of Digital Design* (Prentice-Hall: Upper Saddle River, N.J., 1997).
4. M. M. Mano, *Digital Design*, 3a. ed. (Prentice-Hall: Upper Saddle River, N.J., 2002).
5. J. P. Daniels, *Digital Design from Zero to One* (Wiley: Nueva York, 1996).
6. V. P. Nelson, H. T. Nagle, B. D. Carroll y J. D. Irwin, *Digital Logic Circuit Analysis and Design* (Prentice-Hall: Englewood Cliffs, N.J., 1995).
7. R. H. Katz, *Contemporary Logic Design* (Benjamin/Cummings: Redwood City, Ca., 1994).
8. J. P. Hayes, *Introduction to Logic Design* (Addison-Wesley: Reading, Ma., 1993).
9. C. H. Roth Jr., *Fundamentals of Logic Design*, 4a. ed., (West: St. Paul, Mn., 1993).
10. J. F. Wakerly, *Digital Design Principles and Practices*, 3a. ed. (Prentice-Hall: Englewood Cliffs, N.J., 1999).
11. E. J. McCluskey, *Logic Design Principles* (Prentice-Hall: Englewood Cliffs, N.J., 1986).
12. Institute of Electrical and Electronics Engineers, "1076-1993 IEEE Standard VHDL Language Reference Manual", 1993.
13. D. L. Perry, *VHDL*, 3a. ed. (McGraw-Hill: Nueva York, 1998).
14. Z. Navabi, *VHDL—Analysis and Modeling of Digital Systems*, 2a. ed. (McGraw-Hill: Nueva York, 1998).
15. J. Bhasker, *A VHDL Primer*, 3a. ed. (Prentice-Hall: Englewood Cliffs, N.J., 1998).
16. K. Skahill, *VHDL for Programmable Logic* (Addison-Wesley: Menlo Park, Ca., 1996).
17. A. Dewey, *Analysis and Design of Digital Systems with VHDL* (PWS Publishing Co.: Boston, Ma., 1997).